TU WIEN Informatics

Using Natural Language Processing to Measure the Consistency of Opinions Expressed by Politicians

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Stefan Zaruba, BSc
Matrikelnummer 01325853

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Wien, 14. Oktober 2021

Stefan Zaruba	Horst Eidenberger

TU WIEN Informatics

Using Natural Language Processing to Measure the Consistency of Opinions Expressed by Politicians

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Stefan Zaruba, BSc
Registration Number 01325853

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Vienna, 14th October, 2021

Stefan Zaruba	Horst Eidenberger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Stefan Zaruba, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14. Oktober 2021

Stefan Zaruba

Acknowledgements

I want to thank my advisor, Professor Eidenberger, for being reliable, professional, and always quick to respond. I also want to thank my parents for supporting me during my studies.

Kurzfassung

In dieser experimentellen Studie wird eine Lösung implementiert, um Meinungen, mithilfe von Techniken aus dem maschinellen überwachten Lernens, aus geschriebenem Text zu extrahieren, um in weiterer Folge deren Konsistenz über die Zeit zu visualisieren. Wir prüfen sowohl die praktische Umsetzbarkeit als auch die Nützlichkeit des implementierten Ansatzes.

Wir haben die vom österreichischen Parlament zur Verfügung gestellten Redeprotokolle gesammelt, um zwei Datensätze zu Themen bezüglich Maßnahmen gegen die Verbreitung des Coronavirus zu erzeugen. Um die Einträge für die Datensätze zu gewinnen, haben wir den Rohtext anhand der Satzgrenzen aufgeteilt und relevante Sätze mithilfe einer Schlüsselwortsuche identifiziert. Danach haben wir den Einträgen Meinungslabels per Hand zugewiesen. Anschließend haben wir zwei statistischen Ansätze und drei tiefe Lernnetzwerke verwendet, um die zuvor zugewiesenen Labels mithilfe von maschinellem Lernen zu bestimmen. Wir haben den Vorgang mehrmals wiederholt, um mithilfe einer Monte Carlo Kreuzvalidierung die erzielten Leistungen zu bewerten. Dann haben wir die vorhergesagten Labels des leistungsstärksten Modells verwendet, um die allgemeine Meinung, sowie die Konsistenz von Meinungen über die Zeit, grafisch darzustellen.

Am größeren Datensatz (etwa 5000 Einträge) erzielte ein BERT-Netzwerk die beste Genauigkeit (70%), gefolgt von einem LSTM-Netzwerk (68%), einem MNB-Klassifikator (67%), einem Bag-of-Words-Netzwerk (62%), und einem BM25-Algorithmus aus dem Information Retrieval. Auf einem kleineren Datensatz (etwa 500 Einträge) gewann auch BERT (56%), gefolgt vom MNB (53%), dem LSTM (51%), dem BM25-Ansatz (47%), und dem Bag-of-Words-Netzwerk (42%). Die größten Hürden hinsichtlich der praktischen Umsetzbarkeit waren der manuelle Label-Aufwand, sowie die Herausforderung ein Thema mit einer ausreichenden Anzahl an Meinungsäußerungen zu finden. Daraus schließen wir, dass der umgesetzte Ansatz am besten geeignet ist, wenn geplant ist, ihn über einen längeren Zeitraum und für eine beschränkte Anzahl an Themen einzusetzen. Die Nützlichkeit der vorhergesagten Meinungskonsistenz ist von der Genauigkeit des zugrundeliegenden maschinellen Modells abhängig. Durch den Vergleich der tatsächlichen Graphen mit den vorhergesagten, befanden wir eine Modellgenauigkeit von 70% als ausreichend, um die allgemeinen Meinung zu einem Thema repräsentativ darzustellen. Andererseits erfordert eine nützliche Darstellung der Meinungskonsistenz eine höhere Modellgenauigkeit.

Abstract

This experimental study implements a solution for extracting opinions from written text with the help of supervised machine learning methods to visualize their consistency over time. We examine the practical feasibility and the usefulness of the implemented approach.

We gathered speech transcripts of the Austrian Parliament to create two datasets on topics concerning measures against the spread of the Coronavirus. We split the raw text around sentence boundaries into dataset records and used a keyword search to select relevant sentences. Then, we manually assigned opinion labels and used two statistical machine learning algorithms and three deep learning models to predict the labels. We used Monte Carlo cross-validation to evaluate classification performance. Subsequently, we used the predictions of the best-performing algorithm to plot the general sentiments toward the topic and the consistencies of expressed opinions over time.

On the larger dataset (around 5000 records), a BERT network achieved the best accuracy (70%), followed by an LSTM network (68%), an MNB classifier (67%), a Bag-of-Words network (62%), and a BM25 document ranking classifier (42%). On the smaller dataset (around 500 records), BERT also performed best (56%), followed by the MNB (53%), the LSTM (51%), the BM25 approach (47%), and the Bag-of-Words network (42%). The biggest challenge to practical feasibility was the manual annotation effort and choosing a topic for which enough training samples are available. Thus, the approach is best suited if the intention is to monitor a small selection of topics over a long period. We showed that the usefulness of the predicted opinion consistency values depends on the accuracy of the underlying opinion predictions. By comparing the graphs from actual opinion data to graphs of predicted data, we gathered that a model with 70% accuracy is sufficient to produce a representative impression of the overall sentiment towards a topic. On the other hand, visualizing the consistency of opinions requires a higher classification accuracy to be useful.

Contents

Kurzfassung **ix**

Abstract **xi**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement and Research Questions 2
1.3 Methodological Approach 3
1.4 Outline 4

2 Literature **7**
2.1 Machine Learning Architectures in NLP 7
2.2 Linguistic Processing and NLP Tasks 17
2.3 Supervised Machine Learning in NLP 27
2.4 Related Work 37

3 Design **41**
3.1 Research Method 41
3.2 Requirements: A Definition of Opinion Consistency 42
3.3 Experiment Design 44
3.4 Datasets: Dataset Creation and Analysis 47

4 Experiments **53**
4.1 Opinion Classification: First Experiment 53
4.2 Opinion Classification: Second Experiment 59
4.3 Visualizations of Opinion Data 68
4.4 Opinion Consistency 71

5 Evaluation **81**
5.1 Opinion Classification 82
5.2 Opinion Consistency 86
5.3 Challenges 88
5.4 What is possible and future directions 89

6 Conclusion **91**

List of Figures **95**

List of Tables **97**

Bibliography **99**

CHAPTER 1

Introduction

1.1 Motivation

The trust of citizens in their governments is generally low. A study from the OECD shows that, in 2015, only an average of 43% of people from OECD countries trusted their national government. [OEC] Trust is the necessary foundation for every strong and healthy relationship. A lack of trust severely jeopardizes the stability and effectiveness of any relation. Therefore, finding ways of restoring trust between governments and their citizens is important.

One of the factors influencing the trustworthiness of politicians is the consistency of their expressed opinions. A politician that frequently expresses contrary views on the same topic cannot be trusted because a voter will not have security in how they will be represented by this politician in the future. Therefore, it is important for politicians to express congruent opinions, and for voters to be able to easily and objectively quantify this consistency.

Currently, it would be rather difficult and time-consuming for the voter, to objectively quantify the consistency of speech expressed by politicians. They would have to invest a lot of time in gathering data, reading and analyzing, to make an informed decision. This time commitment, most people simply cannot make. Therefore, they are mostly left with the option of trusting their feelings, which can be easily wrong or considering experts' opinions, which can be severely biased. Both options cannot be considered advantageous positions from which to make important decisions about one's future.

With the help of AI technologies, specifically natural language processing, the consistency of opinions expressed by politicians can potentially be made quantifiable and easily accessible for voters to make informed decisions. Speeches and written statements can be automatically analyzed by NLP algorithms to extract opinions, politicians and parties have, about various topics. The results can be made publicly available and visualized

in an accessible manner for everyone to grasp easily. As a result, it will be easier for politicians to convey their trustworthiness to potential voters and also for voters to choose a party on objective criteria.

The potential effectiveness of such a platform depends not only on the performance of a machine learning algorithm but also on the definition of *opinion consistency*, i.e., how it is calculated from a set of labeled data records. Therefore, one motivation for this work is to find such a definition, which can be used as a basis for further reasoning. Finally, since there does not exist much work on that topic, another motivation is to advance the progress towards being able to effectively reason about the consistency of opinions, through means of NLP.

1.2 Problem Statement and Research Questions

This experimental study aims to implement a solution for extracting opinions from written text with the help of supervised ML methods in order to visualize their consistency over time. In this study, opinions on a topic can be either *positive* (*for*), *neutral* (*indifferent*), or *negative* (*against*). The consistency of opinions should be high when the number of contradictions is low and vice-versa. Two opinions expressed on the same topic are said to be contradictory if one of them is *positive* and the other is *negative*. A more precise definition will be given later. Once an approach is implemented, the lessons learned are used to draw conclusions about the practical feasibility of the approach (Q1a) and the usefulness of produced results (Q1b).

The effectiveness depends not only on the definition of *opinion consistency* but also on the quality of predicted opinions. Since there are not many studies on the model performance of ML algorithms predicting opinions on German texts in the political domain specifically, a performance comparison of different ML models is planned (Q2). In general, it would be helpful to get an idea of the model performances required to predict an *opinion consistency* value to some desired accuracy. (Q3)

Besides *opinion consistency*, visualizing the opinions that an individual speaker holds on a topic is also valuable in understanding the speaker's stance on the topic. Those visualizations can also be drawn for political parties by aggregating over opinions of their members. Again, the usefulness of visualizations created from predicted data will depend on the quality of predictions (Q4).

In summary, we have the following five research questions:

Q1a What is the practical feasibility of monitoring *opinion consistency*, a value representing the consistency of opinions on a topic, through the means of supervised ML methods?

Q1b What is the usefulness of measuring and visualizing the consistency of opinions based on opinion data predicted by supervised ML methods?

Q2 What performance do various ML architectures achieve in predicting opinions in the domain of Austrian political speeches in the German language?

Q3 What could be minimum performance thresholds for ML algorithms to predict the consistency of opinions to a desirable precision?

Q4 How useful are visualizations of opinion data of speakers and parties that are based on predictions made by various supervised ML algorithms?

Now, that the expected outcomes of this work—in the form of answers to the above questions—are established, the next section details a plan of how they are achieved.

1.3 Methodological Approach

The methodological approach for achieving the expected results consists of the following steps.

1. *Performing literature research.* At least the following keywords will be included: natural language processing, natural language understanding, sentiment analysis, topic analysis, opinion mining, German natural language processing. The preferred search engine will be Scopus. Recent review papers will serve as a first entry point to assess the state-of-the-art. When the intended NLP algorithm architecture becomes clearer, more specific search terms will be included.

2. *Defining discussion topics/aspects.* In order to extract opinions from the speech protocols, discussion topics or aspects will be defined, which can be answered by *for* or *against*. They could be broader (e.g., *TAXES*, *HEALTH CARE*) or more specific (e.g., concrete discussion points). Based on the current understanding, broader ones should be preferred because they will be applicable over longer time periods and also make it easier to train and test since more data is available per class.

3. *Gathering and pre-processing of data.* The publicly available speech transcripts of the Austrian Parliament will be downloaded and brought from the *HTML* form into a structured (e.g., *CSV*) form. One record will at least consist of the following information: Date of the speech, name of the speaker, party affiliation, type of speech, governing party (yes/no). Regarding the actual speech, it can be stored in different granularities. A decision will be made, whether it is going to be phrase-, sentence- or an even higher level.

4. *Researching NLP methods and machine learning models.* Since there are many possible approaches to solve this problem, a choice needs to be made, which machine learning models or which NLP methods should be applied. This step goes together with steps 5 and 6 and will be iterated several times, depending on performance results.

5. *Using NLP methods and ML models to overcome the gap between the raw speech data and having it labeled.* Two types of labels need to be assigned: Topic labels and opinion labels (*for*/*against*/*indifferent*). Existing frameworks, models, and algorithms shall be used. The primary implementation language will be Python.

6. *Evaluating classification performance.* The implementation's viability will be evaluated on its produced results, on common performance metrics like *accuracy* and *F1-score*. Additionally, depending on requirements and time available, the opinion classification algorithm can be benchmarked on already annotated German corpora, e.g., *SB-10k* [CDEU17], to verify the implementation.

7. *Visualizing Opinion Data* The predicted opinion labels will be used to plot the number of positive/negative/neutral opinions on a topic per speaker or per political party. The same graphs will be created from the actual opinion labels in order to compare them to the predicted ones.

8. *Defining and Visualizing Opinion Consistency* A formula for computing a value that represents the consistency of opinions will be defined. The computed value will be plotted over time to observe changes in *opinion consistency*. It will also be plotted for multiple speakers or parties in the same graph to compare their consistency values. Again, the graphs based on predicted opinion labels will be compared to those based on the actual opinion labels.

9. *Determining minimum performance thresholds for the used ML algorithms.* Finally, minimum performance thresholds for predicting the *opinion consistency* to some desired accuracy will be determined, either through calculating them or through a sufficient number of simulation runs.

The next section will give the reader an overview of the things to come, such that they can decide which parts are relevant to them and in which order they want to continue reading.

1.4 Outline

A brief overview of the remaining chapters is given. Chapter 2 covers NLP architecture, NLP tasks, supervised learning, and related work. 2.1 examines machine learning architecture in NLP with a focus on supervised methods for the application in opinion mining. It covers statistical methods 2.1.1, word embeddings 2.1.2, convolutional networks 2.1.3, recurrent networks 2.1.4, the attention mechanism and transformer models 2.1.5. 2.2 goes through linguistic processing techniques and NLP tasks relevant for this study. Text segmentation 2.2.1, morphological analysis 2.2.2, POS-tagging and dependency parsing 2.2.3, named entity recognition and coreference resolution 2.2.4, and sentiment analysis and opinion mining 2.2.5 are covered. We introduce the tasks by giving a motivation for their relevance, and we talk about where and how they can be applied,

about different implementations, development history, and state-of-the-art. 2.3 focuses on the learning aspect of machine learning. Particularly on supervised text classification, which is based on training a model on a labeled dataset. An end-to-end perspective of the learning process—from dataset creation 2.3.1, over pre-processing 2.3.2, to training and model selection based on evaluation criteria 2.3.4—is given. 2.3.5 covers the tooling and infrastructure required for learning. 2.3.3 examines the difference between validation, verification, and evaluation in the context of machine learning. 2.4 examines work similar or related to opinion mining on political speeches in the German language.

Chapter 3 defines the framework in which the experiments are performed. 3.1 describes the research method which is applied. 3.2 works out a definition of opinion consistency by which the consistency of opinions can be measured based on speech transcripts. 3.3 defines a plan for the experiments to be performed in order to answer the research questions 1.2. 3.4 details how the data was gathered and used to create the datasets. Furthermore, an analysis of the datasets is performed.

Chapter 4 provides detailed documentation of performed steps and achieved outcomes of the conducted experiments. 4.1 documents the opinion mining process on the first dataset, including a comparison of model performances. 4.2 documents the improved opinion mining process on the second dataset. 4.3 visualizes opinion data aggregated per speaker and party. The chapter is concluded in 4.4 where the opinion consistency, as it was defined earlier, is visualized over time 4.4.1 and the impact of a model's capability to classify opinions on the accuracy of calculated opinion consistency values is examined 4.4.2.

Chapter 5 evaluates the results which are observed during the experiments. After an overall summary of the vision and early steps in the project, 5.1 evaluates the results of classifying opinions during the experiments. 5.2 evaluates the usefulness of opinion consistency charts that were plotted during experiments. 5.3 outlines the challenges related to a general topic-independent approach to monitoring opinions. 5.4 discusses where we are on the road to achieving a generic approach of monitoring the consistency of opinions.

Finally, 6 provides an overall conclusion by summarizing the findings in regards to the research questions and makes suggestions about future work towards robust systems for monitoring the consistency of opinions over time.

CHAPTER 2

Literature

2.1 Machine Learning Architectures in NLP

This section examines machine learning architecture in NLP, focusing on supervised methods for the application in opinion mining.

2.1.1 Statistical Models

Statistical machine learning models became popular for NLP tasks in the 1990s, with the advent and popularization of the world wide web. The rapidly growing amount of textual data shared over the internet enabled the effective learning of such models. Before that, starting in the 1950s, mainly rule-based approaches were used in NLP research in the areas of word/sentence analysis, question answering (QA), and machine translation (MT). Statistical machine learning stayed the preferred option roughly until 2012, when deep learning models were introduced to NLP tasks, quickly becoming state-of-the-art. [ZDLS20] We will introduce the two statistical models that are used in the experiments section—The Naive Bayes classifier and the BM25 document ranking algorithm.

The Naive Bayes classifier is a probabilistic method that works under the assumption that input variables (of the classifier) are independent; thus, the observed outcome of one variable does not influence the likelihood of results of another variable. For example, in document classification, the Naive Bayes assumption implies that the presence of words is independent of their context. Clearly, this is not the case, as some combinations of words appear more frequently together than others, but despite that, the classifier can still be effective in many cases. [MS99] We explain how Naive Bayes can be applied to the task of document classification, according to Li and Jain [LJ98]). Let $C = (c_1, \dots, c_m)$ be m document classes and $D = (w_1, w_2, \dots)$ a document as a set of its words. The most

likely class $\hat{c}$ for the document D can be estimated by

$$\hat{c} = \text{argmax}_{c_j \in C} \; P(c_j) \prod_{w_i \in D} P(w_i|c_j) \tag{2.1}$$

taking the argmax of classes $c_j \in C$ as the product of priori probability of class c_j and product of the posteriori probabilities of words in the document given that it is of class c_j. The probabilities can be calculated from the observed data (labeled corpus documents). The priori probability of class c_j can be calculated by

$$P(c_j) = \frac{N_j}{N} \tag{2.2}$$

with N_j denoting the number of documents of class j and the total number of documents N. The conditional probability of a word w_i occurring in a document of class c_j is calculated as

$$P(w_i|c_j) = \frac{n_{ij} + 1}{n_j + k_j} \tag{2.3}$$

with n_j total number of words in class c_j, n_{ij} number of occurrences of word w_i in class c_j and k_j unique words in class c_j (vocabulary size of c_j). Adding the $+1$ in the numerator and the $+k_j$ in the denominator is a Laplace smoothing technique used to prevent zero-probability factors.

$$w_i^{\text{RSJ}} = \log \frac{(r_i + 0.5)(N - R - n_i + r_i + 0.5)}{(n_i - r_i + 0.5)(R - r_i + 0.5)} \tag{2.4}$$

$$w_i^{\text{IDF}} = \log \frac{N - n_i + 0.5}{n_i + 0.5} \tag{2.5}$$

$$w_i^{\text{BM25}}(tf) = \frac{tf}{k_1 \; ((1-b) + b\frac{dl}{avdl}) + tf} \; w_i^{\text{RSJ}} \tag{2.6}$$

$$P(rel|d,q) = \sum_{\boldsymbol{q}, tf_i > 0} w_i^{\text{BM25}} \tag{2.7}$$

According to Robertson and Zaragoza [RZ09] BM25 is one of the most successful text retrieval algorithms, which comes from the field of information retrieval. It is based on the Probabilistic Relevance Framework (PRF), which originated in the 1970–1980s through the works of Robertson and Jones and was further developed for the following 30 years. The PRF's basic idea is to build document-query pairs and ordering them by decreasing the probability of relevance.

We walk through the calculation of document scores. First, some notation occurring in the formulas is explained. The probabilistic notion of the framework is expressed through the random variable *Rel* which can take the values *rel* (document is relevant) and $\overline{rel}$ (document is not relevant). The authors go into more detail about the probabilistic considerations behind the framework, but we will focus on the final formulas. All possible terms (which can occur in documents and queries) are indexed into the vocabulary set $\mathbf{V}$. A document $d := (tf_1, ..., tf_{|\mathbf{V}|})$ is a vector of term frequencies tf_i that count how

often the i-th term of the vocabulary is present in the document. A query can either be represented as a vector of query term frequencies $q := (qtf_1, ..., qtf_{|V|})$ or as an index set of terms occurring in the query $\boldsymbol{q} := \{i|qtf_i > 0\}$. As shown in equation 2.7, the probability of relevance for a given document d and query q is calculated by summing up over the weights w_i^{BM25} for query terms ($\boldsymbol{q}$) present in the document ($tf_i > 0$). The w_i^{BM25} calculation is the product of two components—a term-frequency calculation and a document-frequency calculation. Equations 2.4 and 2.5 show two ways to calculate the document-frequency component. If information about the relevance of documents is present, thus documents were judged as relevant or not relevant beforehand, the Robertson/Sprck Jones weight w_i^{RSJ} can be applied, in which R denotes the number of documents that are judged as relevant and r_i denotes the number of relevant documents containing term t_i (index i comes from the index set of query terms $\boldsymbol{q}$). N denotes the total number of (judged) documents, while n_i denotes the number of (judged relevant) documents containing t_i. If no relevancy information is present, then R and r_i can be set to zero, and the formula becomes w_i^{IDF}, a classical inverse document frequency (IDF) calculation, in which a term t_i becomes more relevant, the less the number of documents where t_i appears. The term weights w_i^{RSJ} or w_i^{IDF} are multiplied with the term frequency component to get the BM25 term weight w_i^{BM25} (2.6). The term frequency function tf normally denotes the number of times a term appears in the document. The expression $((1-b) + b\frac{dl}{avdl})$ represents a document length normalization, with $0 \leq b \leq 1$ regulating its impact. The larger the length of a document (dl) in relation to the average document length ($avdl$), the less important t_i becomes. The authors suggest $0.5 \leq b \leq 0.8$ and $1.2 \leq k_1 \leq 2$ for the internal parameters. The BM25 method is typically used for document ranking in search engines. MG4J, Xapian, and Zettair are some examples of search engines implementing BM25. [RZ09]

2.1.2 Word Vectors and Word Embeddings

We now examine different ways of representing the textual input of NLP machine learning models. Statistical models are based on counting the occurrence of words and word probabilities. Deep neural networks consist of multiple layers of neurons. The input layer takes a vector of numbers as its input, with the vector's dimensionality equal to the number of input neurons. A simple idea would be to pass the word counts to the network's input layer in the form of a one-hot-encoded input vector. That would result in input vectors with a dimension equal to the total number of unique words. The problem with such sparse vectors is that they make it more challenging to train the network, as the higher the dimensionality, the more data are required. [WLS+15] Distributed representation solves this problem by projecting the high-dimensional word vectors into a relatively low-dimensional space by putting semantically more similar words in closer proximity to each other. [YHPC18] Now, the distance between pairs of word vectors has a meaning, compared to the other approach, in which words are arbitrarily numbered.

Learning word representations dates back already to 1986, where Rumelhart, Hinton, and Williams [RHW86] trained neural networks to show that the internal units represent

features of the task domain. Word embeddings were revolutionized in 2013 by the works of Mikolov et al. [MSC+13], who introduced continuous bag of words (CBOW) models and Skip-Gram models for training. They were significantly more computationally efficient than previous models, which greatly improved the quality of trained word vectors. They also found that the produced word vectors follow the rules of compositionality to some extent. For example, they found that vec("Madrid") - vec("Spain") + vec("France") is closest to the vector vec("Paris"). [MSC+13] CBOW models are trained to predict a word based on its surrounding context words (in a certain window size), while Skip-Gram models are trained to predict the context words, given a center word. N-grams are word sequences composed of n words. Training a distributed word representation from scratch is resource-intensive. [MCCD13] For that reason, often pre-trained word embeddings, which are a list of word vectors, are used in the first layer of a neural network architecture. On the other hand, training domain-specific word embeddings can improve performance on NLP tasks. [LL13] A compromise between the quality of word embeddings and training effort is an approach proposed by Labutov and Lipson. [LL13] They take a pre-trained general word embedding and tweak it for a specific sentiment classification task to achieve improved performance compared to the baseline.

Although word embeddings have improved results in many NLP tasks, they come with shortcomings and challenges, as outlined in [YHPC18]. Traditional word embeddings work well for words with only one meaning, but they struggle with words that have different meanings in different contexts (known as polysemy). Research is investigating the effectiveness of multi-sense word embeddings, in which different word vectors can be inferred based on the word's context. Interestingly, multi-sense embeddings may not give improved results in all NLP tasks. Li and Jurafsky [LJ15] show that multi-sense embeddings improved performance on some tasks (e.g., POS-tagging) but not in others (e.g., sentiment analysis and NER). Another problem is that phrases can have a different meaning than the sum of their constituting words. For example, "hot dog" or idioms, e.g., "beat around the bush." Some methods have approached this problem by learning embeddings for n-grams, e.g., Johnson and Zhang [JZ15]. Another challenge that is particularly relevant for sentiment analysis is that semantically similar words can have negative polarities. For example, the words *good* and *bad* are considered semantically similar since they are likely to occur in similar contexts but have opposite polarities. [SPH+11] The authors of [TWY+14] approach this problem with sentiment-specific word embeddings (SSWE), by which they encode sentiment information in the continuous representation of words.

2.1.3 Convolutional Neural Networks

We now examine different neural network architectures, starting with convolutional feed-forward NN. Distributed representation made it possible to extract features from individual words, but the next step was to extract features of parts of a sentence and an entire sentence. Convolutional neural networks, already successful in image recognition tasks, were found to be useful also in NLP tasks. The idea of CNNs in NLP is to run

filters of many differently sized windows over the sentence, each one extracting a new feature. The network learns to extract relevant features automatically by continually updating the filters' weights to minimize the loss of an objective function—a process that previously required manual feature-engineering work. The downside of this automatic feature extraction is that the network is a black box, and it is more difficult to explain the extracted features. Of the features extracted by convolution, a max-pooling layer then selects the most relevant ones. Those features could then, for example, be used to perform classification tasks by running them through a dense layer with output neurons equivalent to the number of target classes.

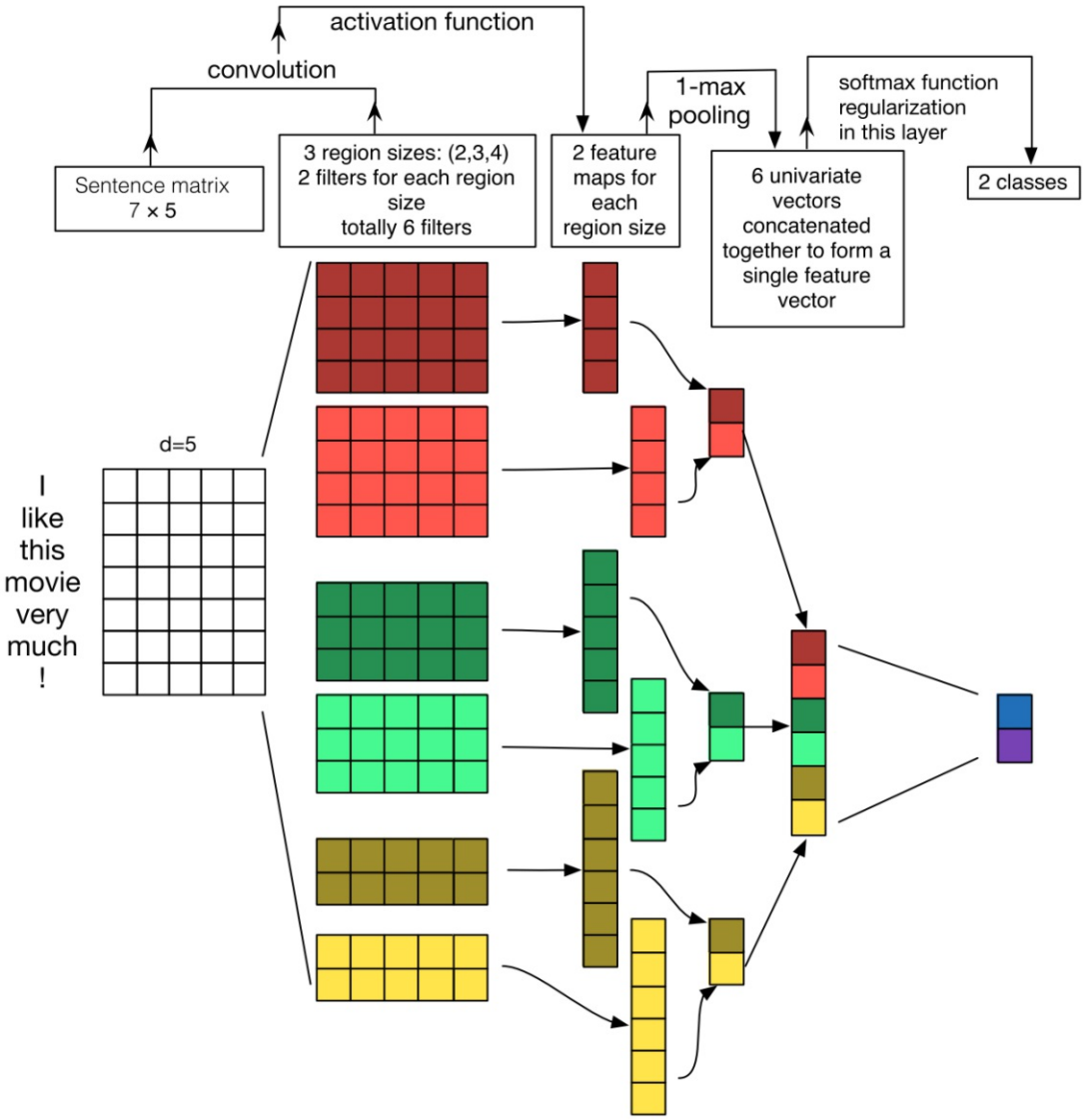


Figure 2.1: CNN architecture for sentence classification [ZW15]

We describe the functioning of a CNN architecture for sentiment analysis in more detail, according to [YHPC18]. Figure 2.1 shows a sample architecture with one convolutional layer. In the first step, the embedding of a sentence is performed. Each word of the sentence is mapped to a vector the size of the embedding dimension d. The result is a sentence matrix $W \in \mathcal{R}^{n \times d}$ with n being the number of words in the sentence. The i-th word of a sentence is denoted by $w_i \in \mathcal{R}^d$. Convolution is performed between regions of input vectors and filters (also called kernels) of different region sizes. The purpose of a filter is to extract features of a sentence for a specific n-gram size. The n-gram size is according to the region size, e.g., a filter with a region size of two extracts features of bi-grams in the sentence, while a filter with region size three extracts features of tri-grams. In this example, there are six filters in total, two for each of the three chosen region sizes. Let $w_{i:i+j}$ be the concatenation of vectors $w_i, w_{i+1}, \ldots, w_j$ and $\mathbf{k} \in \mathcal{R}^{h \times d}$ be a filter of region size h, then new features c_i are extracted via

$$c_i = \Phi(w_{i:i+h-1} \cdot k^T + b) \tag{2.8}$$

In the above equation, Φ denotes an activation function and $b \in \mathcal{R}$ a constant bias term. We can observe that convolution is performed between sliding windows of the input vectors w and the filters k. Convolution is performed with all possible window positions, resulting in feature maps $c = [c_1, c_2, \ldots, c_{n-h+1}]$ of size $n-h+1$. Subsequently, a max-pooling operation $\hat{c} = maxc$ is performed on each feature map in order to produce a fixed-length output and to reduce dimensionality while still keeping the most important n-gram features for each filter. In this example, a 1-max pooling is performed, keeping the largest value of each feature map, resulting in 6 values, which are then concatenated to a single feature vector. Finally, a dense layer with softmax regularization can be connected to the desired number of output classes, e.g., two output neurons, to represent a positive or negative sentiment in the input sentence. Weights of word embeddings can be initialized randomly and trained ad-hoc, or pre-trained word embeddings can be used. This sample architecture features only one convolutional layer, but it is possible to chain multiple layers of convolution and max-pooling to achieve improved feature abstraction capabilities. [YHPC18]

2.1.4 Recurrent Neural Networks

Compared to CNNs, in which more and more abstract features are extracted hierarchically, RNNs build an understanding of the sentence by processing it in sequential order, similar to what a human reader would do. Different RNN architectures use memory components to keep track of information across long distances and gates to filter out unimportant information and keep important information. The authors of [YHPC18] outline several motivating factors to use RNNs over CNNs for language processing. RNNs can deal much easier with variable-length input and very long input (e.g., long sentences, paragraphs, or documents). They are better suited for machine translation due to their ability to handle long-term dependencies and to summarize a sentence to a single vector that can be mapped back to a variable-length target sequence. In contrast, CNNs struggle with

modeling long-distance contextual information and preserving sequential order in their feature representation. Additionally, CNNs require more trainable parameters, which requires more training data. One could think that the RNN architecture is naturally more suited for processing language, which is sequential by nature and thus should achieve better results than CNNs, but this is not the case as [YKYS17] suggests. They found that performance depends on the task and dataset and that there is no clear winner.

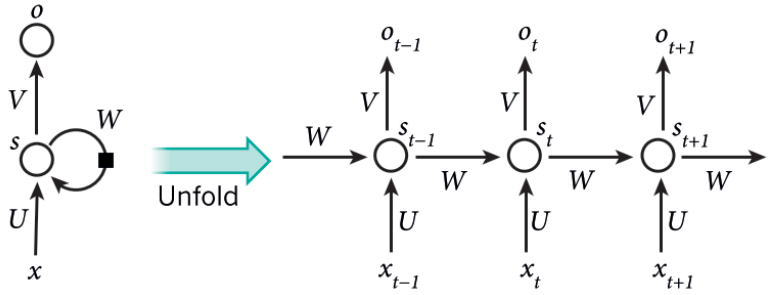


Figure 2.2: Basic RNN architecture [LBH15]

There are different implementations of RNNs, e.g., long-short term memory (LSTM) and gated recurrent units (GRU). In the following, the functioning of RNNs is explained in more detail, according to [CGCB14]. Figure 2.2 shows how a traditional RNN works. We will explain it based on the example of a POS-tagging task. The network uses a hidden state h to keep and update information over time. The sentence is fed into the network word by word, denoted by word vector x_t. The hidden state h_t is computed by

$$h_t = \Phi_1(Ux_t + Wh_{t-1}) \tag{2.9}$$

with Φ_1 being an activation function and U, V, W being the networks weight matrices. We see, that the state of the current time depends on the state of the previous time plus the current input. Thus, information is propagated over time and influences the output based on previous inputs. The current output o_t (POS-tag of the word x_t) can be calculated by

$$o_t = \Phi_2(Vh_t) \tag{2.10}$$

with Φ_2 being another activation function.

Such a simple implementation of RNNs struggles with keeping long-term information because they are affected by the vanishing or exploding gradients problem, causing gradients to go towards zero or infinity, respectively. This effect becomes more pronounced the more timesteps are involved. [BSF94] The LSTM and GRU architectures overcome these problems by using gates, which control the flow of information over time. [YHPC18]

Figure 2.3 shows a schematic overview of the three RNN architectures. An empirical evaluation of the three RNN architectures by the authors of [CGCB14] showed clear superiority of LSTMs and GRUs over simple RNNs. However, they could not determine a clear winner between LSTMs and GRUs.

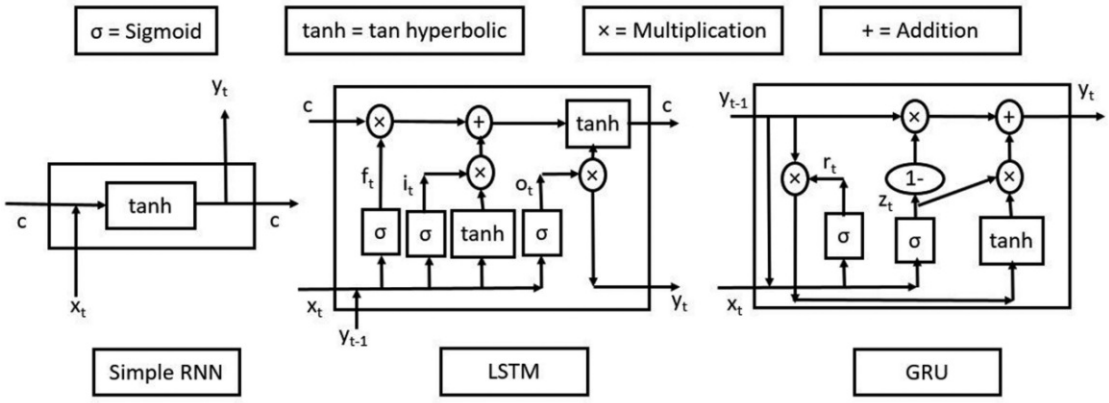


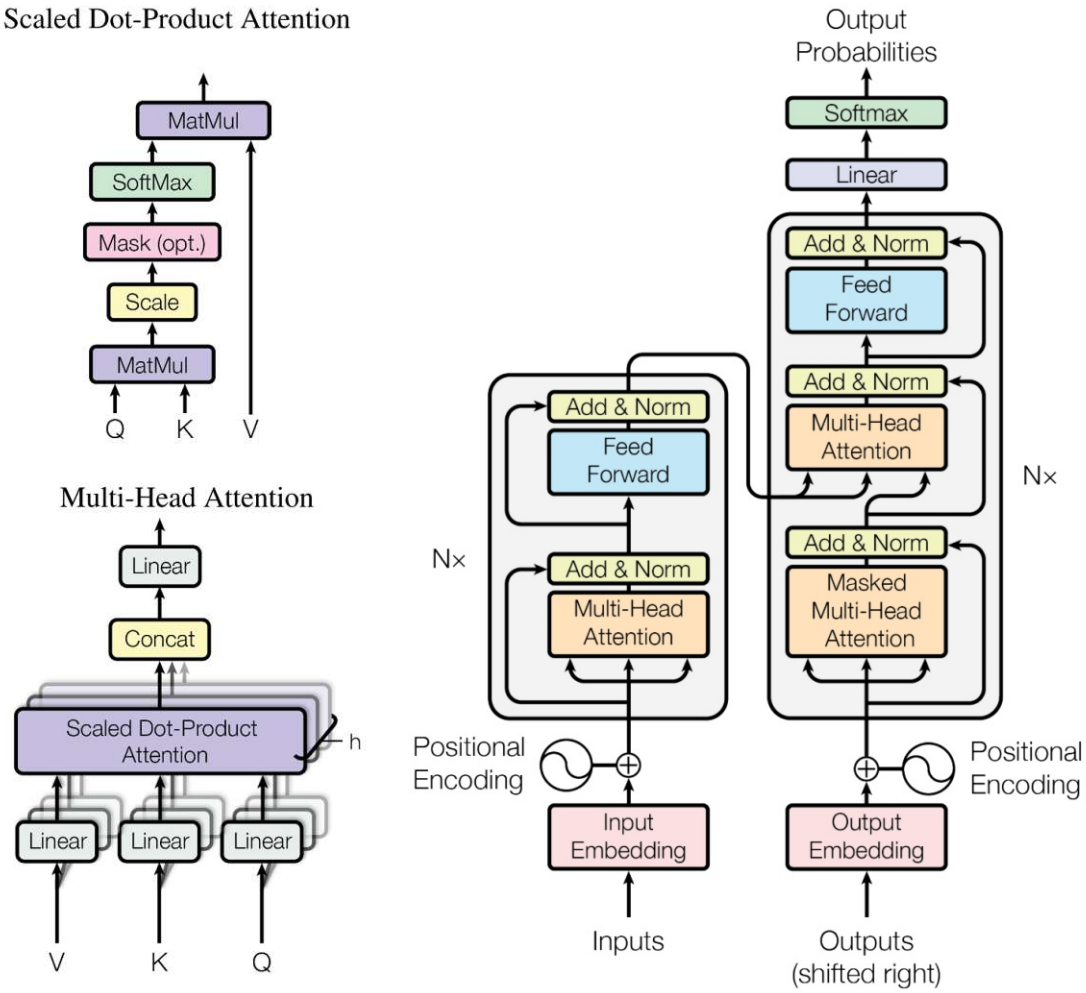
Figure 2.3: Schematic overview of different RNN architectures. [KKDC19] Input vectors are denoted by x, output vectors by y and hidden state vectors by c. Merging arrows indicate a concatenation of vectors and a splitting arrow indicates a copy operation.

2.1.5 Attention and Transformer Models

The next milestones in the development of NLP architecture were the introduction of encoder-decoder architecture and, based on that, the transformer architecture, which is based on the attention mechanism. The idea of encoder-decoder first emerged in sequence-to-sequence (seq2seq) tasks like machine translation but is now also used in other tasks since most of the NLP tasks can be cast to sequence-to-sequence. The transformer architecture formed the basis for current state-of-the-art models, e.g., BERT and GPT-3.

The encoder-decoder architecture uses two RNNs, one to encode a sequence of input tokens (e.g., a sentence in one language) and another to decode a sequence of output tokens (e.g., a translated version of the input sentence). The hidden state generated by the first RNN is used to initialize the hidden state of the second one. [Hu19] There are three major drawbacks with this architecture. First, it does not work well for long sequences because information tends to be forgotten, the more timesteps are involved. Second, because the entire sequence is encoded before it is decoded, there is no alignment between input and output tokens. [Hu19] Thirdly, the sequential nature of RNNs does not allow for parallel processing. [VSP+17] Intuitively, it would be easier to translate a text part by part instead of memorizing it in its entirety and then translating it from memory. That is the idea of the attention mechanism for NLP tasks. The intuition behind attention in transformers is that it allows the decoder to reference the most relevant parts of the input sequence "by focusing its attention" on those parts during the decoding process to improve decoding performance.

We now explain the functioning of transformers in more detail, according to the famous paper "Attention is All You Need." [VSP+17] and to the explanations of Raschka [Ras21]. The basic ingredients form attention blocks. An attention block aims to enhance each



(a) Attention blocks

(b) Transformer architecture

Figure 2.4: Overview of the transformer architecture [VSP+17]

embedded token of an input sequence with context information to all other tokens. The network then learns which relationships between pairs of tokens are more relevant than others. Thus the network learns "to pay more attention" to the relevant context for a specific task. Figure 2.4a depicts a scaled dot-product attention block. It consists of six steps:

1. Given an embedded input sequence $(x_1, \ldots, x_n)$ (e.g., a sentence) which is represented by the matrix $X \in \mathcal{R}^{n \times d_e}$, with d_e being the embedding dimension, the inputs to the attention block are constructed by: $Q = X \times W^q$, $K = X \times W^k$, and $V = X \times W^v$. $W^q, W^k \in \mathcal{R}^{d_e \times d_k}$ and $W^v \in \mathcal{R}^{d_e \times d_v}$ are the embedding weights to create queries, keys and values, with embedding dimension d_k for keys and queries

and embedding dimension d_v for values.

2. The matrix multiplication $QK^T \in \mathcal{R}^{n \times n}$ is performed to determine the relationships between pairs of tokens. For example, the first row of QK^T contains the relationships between the first token $x_1 \in X$ to all tokens $x_1, \ldots, x_n$ of the input sequence.

3. The scaling factor $1/\sqrt{d_k}$ is applied, to counteract small gradients in the softmax function, which can occur for large values of d_k.

4. The next step is an optional masking operation, which is used only in the decoder block to allow the network to "focus attention" only on the current and previous positions in the sequence. Limiting the potential area of attention is achieved by adding a mask value M, which can be either zero for no masking or negative infinity to apply the mask.

5. A softmax function is applied for normalization.

6. The final attention matrix $A \in \mathcal{R}^{n \times d_v}$ is calculated by equation 2.11, which contains a different embedding of the input sequence. For example, the first row of A contains an embedding for the first token of the input sequence, enhanced with attention information to the other tokens of the same input sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T + M}{\sqrt{d_k}})V \tag{2.11}$$

The scaled dot-product attention block is applied h times in a multi-head attention block (Figure 2.4a). Each scaled dot-product attention block can focus on a different task by training the weight matrices W^q, W^k, W^v differently. The embeddings of those blocks are then concatenated, and convolution is applied in a fully-connected linear layer with weights W^O:

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.12}$$

$$\text{head}_i = \text{Attention}(QW_i^q, KW_i^K, VW_i^V) \tag{2.13}$$

with $W^O \in \mathcal{R}^{hd_v \times d_e}$

Figure 2.4b depicts the transformer architecture, which is based on the previously discussed attention blocks. It consists of an encoder on the left side and a decoder on the right side. The network processes a sequence of input tokens in sequential order, where the encoder has access to the entire sequence from the beginning, and the decoder (up to after the masked multi-head attention block) has only access to the current token and already processed tokens because the rest is masked. The encoder and decoder blocks are also stacked multiple times to extract increasingly abstract features. The encoder creates a new embedding of the input sequence with additional attention information (which can be seen as context information for every token in the sequence), and the decoder uses this new embedding to create an output sequence token by token. The

positional encoding step applies position information to each token, making it possible for the network to distinguish between tokens that occur multiple times (e.g., the same word occurs twice) in the sequence. The decoder's final layers look differently based on the NLP task. For example, in machine translation, the final hidden vector embeddings would be fed into a softmax-regulated linear layer with the number of output nodes corresponding to the vocabulary size of the target language. [DCLT19] Transformer models can also be adapted to single token outputs, as described by [SQXH19], who performed text classification with BERT. The authors prepended every input sequence with a placeholder token that contains the classification embedding of the entire sequence. This embedding is used in a softmax classifier to predict the output class.

2.2 Linguistic Processing and NLP Tasks

In this section, we go through linguistic processing techniques and NLP tasks relevant to this study. We introduce the tasks by giving a motivation for their relevance, and we talk about where and how they can be applied, about different implementations, development history, and state-of-the-art.

2.2.1 Text Segmentation

Splitting a text into segments such that it can be further processed is a prerequisite for many natural language processing tasks. Depending on the task, different granularities are necessary or better suited. Possible segmentation levels range from the sub-word and word levels to the topic or document levels and everything in between. The desired granularity can also depend on the processed language. For example, sub-word tokenization on Chinese text may yield better results than word tokenization on subsequent language tasks, which was demonstrated by Peng et al. [PCZ17]. They achieved better results on a sentiment analysis task by using radical-based embeddings over word embeddings. If not otherwise stated, statements of the remaining chapter will per default concern the German or the English language.

A popular way of implementing text segmentation algorithms is by using finite-state sequence-tagging models, like hidden Markov models, discriminative tagging models based on maximum entropy classification, conditional random fields, and large-margin techniques. [MCP05a]

Segmenting the text on the word or sub-word level is known as *tokenization*. At first, the tokenization of words looks like an easy task—split the text around the white spaces and remove punctuation—but in fact there are many considerations to make. One of them is the treatment of compound words. Should *time zone* be considered as one or two words? What if there are different ways of writing the same word, e.g., *timezone*, *time zone* and *time-zone*. There are valid arguments both for treating them as the same word or as different words. In German, many compound words are naturally joined together. For example *Lebensversicherung* (life insurance). Even in that case, it might be desirable to

split them up into separate words, depending on the use case. A possible implementation for splitting compound words is demonstrated by CharSplit [Tug16], which identifies the most likely position to split the word based on probabilities of n-grams occurring in that word. There are many more considerations to make, e.g., the treatment of commas and dots in dates, emoticons, web addresses, brand names, hyphens and apostrophes, differentiation between punctuation belonging to a word and punctuation not belonging to a word, to name a few. [MS99] These examples show, there does not exist only one correct definition of a word, but one has to implement a tokenization algorithm based on specific requirements. Since tokenization is a fundamental task, it is implemented by many NLP frameworks. [SLC17]

Segmenting the text into sentences is also not a trivial task. Some of the challenges are easier to answer than others. For example, punctuation marks that appear mid-sentence, e.g., dots in dates and numbers, should probably not split the sentence. On the other hand, there is no clear answer to whether a semicolon or em dash should start a new sentence, given no additional information. Again, the implementation of sentence splitting will depend on the specific use case.

There are also efforts to detect topic changes in text in order to split the text around those. Results are partly dependent on the definition of a topic change. It is a difficult task for humans because a precise definition of what a topic is can hardly be given, leading to bad inter-annotator agreement. As a result, according to Stede [Ste12], there exists a wide range of segmentation techniques that perform vastly differently on different datasets. He provides a comprehensive overview of approaches to tackle the problem of topic segmentation, divided into four categories: 1) exploiting surface cues, 2) lexical chains, 3) word distributions, and 4) probabilistic models. Beeferman et al. [BBL99] learn the change of a topic based on the boundaries of news articles. Yamron et al. [YCG+98] used hidden Markov models and classical language modeling techniques, to automatically detect boundaries of stories and achieved promising results.

2.2.2 Morphological Analysis, Stemming, Lemmatization and Normalization

While many NLP tasks are concerned with the inter-word analysis, there are several motivations for analyzing the morphological information of individual words. One goal is to reduce vocabulary size by reducing related words to a common base form. For example, in a language like Finnish, in which a verb can have more than 10,000 forms, it is impractical to enumerate them all in the vocabulary. [MS99] There, the processes of stemming (truncating a word to its stem) or lemmatization (identifying a base form for the word depending on context) can be essential pre-processing steps. Another motivation for identifying related words comes from Information Retrieval (IR) systems, where it is used to improve indexing and search results. Singh and Gupta [SG16] evaluated the impact of different stemming algorithms across different languages on IR results. Their comparison clearly shows how morphological analysis is language-dependent. In English, the improvements in retrieval scores gained by applying stemming algorithms

over not applying them are relatively small compared to other languages, like Hungarian. Also, machine learning algorithms can benefit from morphological pre-processing steps. Singh and Gupta [SG16] show that an SVM algorithm for text classification benefits from stemming (all six evaluated stemmers show increased F-scores over a non-stemmed approach).

Stemming and Lemmatization are also subsumed under the term *word normalization*. [TTJ06] In contrast, *text normalization* is concerned with transforming expressions to a canonical form, which is especially important in text-to-speech applications. For example, the written expression "€25" and the spoken expression "twenty-five euros" should be treated as the same entity. Another important application of text normalization is finding a canonical form of different archaic expressions of a word (e.g., normalizing the archaic expressions *theire*, *theiare* and *thayr* to the modern version *their*). [Bol19]

Both stemming and lemmatization share the same goal of reducing different variants of a word to a common base form, but the approach and outcome are different. Stemming is the simpler and more syntactical approach, which usually works by removing affixes from a word. Lemmatization considers semantic information of a word by analyzing its context and applying a POS-tag in order to find a lemma that represents the underlying lexeme (a set of words with a similar meaning). As a result, lemmatizers are more difficult to implement. [Jiv11] The following examples (taken from [Jiv11]) illustrate the different outcomes a stemmer and a lemmatizer could have on the same words.

- Stemming: introduction, introducing, introduces—introduc
- Lemmatizing: introduction, introducing, introduces—introduce
- Stemming: gone, going, goes—go
- Lemmatizing: gone, going, goes, went—go

A few observations can be made:

1. In contrast to the lexemes produced by the Lemmatizer, the stems do not have to be actual words found in a dictionary (also called *bound stems*) but can be (called *free stems)*. [SG16]
2. A lemmatizer reduces the word *went* to the same root as the words *gone*, *going*, and *goes*, while the stemmer reduces *went* to a different root.
3. A stemmer and a lemmatizer can reduce the same word to the same root but do not have to.

A brief overview of different implementations of stemmers and lemmatizers is given. Appl [Jiv11] divides stemming algorithms into truncating, statistical and mixed approaches. Truncating approaches work by (iteratively) applying a set of transformation rules for

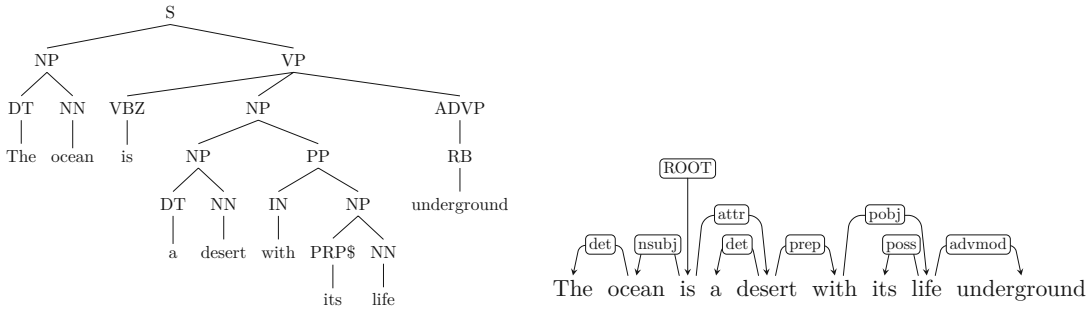
removing affixes. A popular example is the Porter Stemmer [Por80], in which different rules are applied over five steps to find a stem by removing suffixes. In statistical methods, word commonalities are identified by applying unsupervised learning to large corpora. For example, n-gram stemmers cluster words that share a high proportion of character n-grams. Of course, also neural models are applied to the tasks. For example, Lematus [BG18] is a sequence-to-sequence neural model for lemmatization, performing as well or better than the previous models, evaluated on 20 different languages. It is based on the neural machine translation framework Nematus of Sennrich et al. [SFC+17] Instead of taking a sequence of words in one language and outputting a translated sequence of words in the target language, in this case, the input is a space-separated sequence of characters of a word and its context, and the output is the lemma of the word, in the form of a space-separated sequence of characters.

2.2.3 POS-Tagging and Dependency Parsing

After a sentence was split into tokens (during the process of tokenization) it is of interest to identify its parts of speech (POS-tagging) and to map its syntactical structure (constituent parsing and dependency tree parsing) or even its deep semantic structure (dependency graph parsing). While a syntactic parse, in the form of a dependency tree, also provides shallow semantic information, recently, the demand for other representations, allowing to carry deeper semantic information, grew, leading to the exploration of dependency graphs. [Zha20]

POS-tagging is another stepstone towards natural language comprehension. The goal is to assign one tag (e.g., verb, noun, adjective) per word, designating its role in a sentence. A popular tagset is the Penn Treebank POS tagset [TMS03], which contains 48 different tags. A particular challenge is that the same word can have different tags in different environments. For example, the word *play* can be a noun or a verb. How to deal with this issue falls under the research area of word-sense disambiguation. A popular probabilistic algorithm that is used to solve POS-tagging is the Viterbi algorithm [Vit67], applied to Hidden Markov Models. The basic idea is to calculate probabilities with which a word has a certain tag based on the tags of surrounding words. Other categories of approaches include rule-based and transformation. Of course, also deep learning models were applied to POS-tagging, achieving state-of-the-art results.

In syntactic parsing, the goal is to uncover the relationship of words in a sentence based on grammatical rules. There are two common ways of syntactic parsing—*constituent parsing* and *dependency parsing*. As the names suggest, in the former, the sentence is recursively split into constituents, starting from the entire sentence, arriving at individual words, while in the latter, the dependencies between words are uncovered. Figure 2.5 shows an example of the two parsing methods on the same sentence. On the left (Figure 2.5a), we see how the entire clause (S) is split repeatedly into noun phrases (NP) and verb phrases (VP) until we arrive at individual words. Note that one level above the individual word appears its POS-tag indicating which kind of constituent phrase will be built. On the right (Figure 2.5b), the parse shows the relationship between the root



(a) A constituency parse

(b) A dependency parse

Figure 2.5: Different syntactic parses of the same sentence as produced by spaCy [HMVB20]

word (ROOT) and its dependent words, recursively. The annotations are from the Penn Treebank syntactic tagset and the Penn Treebank POS tagset. [TMS03] contains the full list of tags.

According to Zhang [Zha20], the majority of approaches for the dependency parsing problem can be divided along two axes:

1. The first axis describes the framework with which dependency trees can be constructed. Frameworks can be transition-based or graph-based.
2. The second axis describes the learning approach in training a classifier to predict the correct tree within the framework. Classifiers can be divided into statistical classifiers or neural model approaches.

Transition-based frameworks use a set of transition rules to parse a sentence in sequential order. For example, Nivre [Niv03] is a stack-based algorithm, with parser transitions for creating left and right arcs, pushing and popping tokens to and from the stack. In the training process, a classifier is trained on a treebank (e.g., Penn Treebank), learning to predict the correct parser transitions based on a given input sequence of tokens. Classifiers can be of statistical nature, e.g., Nivre [Niv08] uses a state vector machine to evaluate four different transition-based algorithms or neural models.

The basic idea of *graph-based dependency parsing* is to reformulate the problem as a maximum spanning tree problem, e.g., as described in [MCP05b]. A graph is constructed by taking the words as nodes and arcs between them, representing the dependencies between words. Weights are assigned to the arcs according to the likelihood that there is a relationship of dependence between the connected words. By maximizing the sum of weights of a valid (some properties have to be satisfied) spanning sub-tree, the most likely correct dependency parse will be found.

Zhang's comparison of 28 approaches [Zha20] shows that graph-based models perform better than transition-based and neural models perform better than statistical models.

2.2.4 Named Entity Recognition and Coreference Resolution

The term *named entity* (NE) was coined in 1996, during the Sixth Message Understanding Conference [GS96], when research was focused on information extraction (IE). The demand for identifying text passages that refer to real-world entities, e.g., persons, companies, or locations but also for identifying numbers, dates, and percentages increased. Besides marking the text sections which refer to a NE, the task of named entity recognition (NER) also involves assigning an appropriate category label (company, person, location, date, etc.). But what exactly is considered a NE, and which references should be considered? Some refer to NEs as a proper nouns [PCV+00], while others refer to them as *rigid designators* [NS07]. According to the Stanford Encyclopedia of Philosophy *"a rigid designator designates the same object in all possible worlds in which that object exists and never designates anything else."* [LaP18, para. 1] Today, research has come to the consensus of dividing the NEs into two categories: generic NEs (e.g., person and location) and domain-specific NEs (e.g., stock ticker symbols, proteins, or genes) [LSHL20]

Over the period of 1991 to 2006, the implementations of NER shifted from rule-based approaches to machine learning approaches. [NS07] Today, rule-based approaches are still used sometimes today, but machine learning and especially deep learning approaches are clearly more prevalent. Li et al. [LSHL20] divide the approaches into the following categories:

1. Rule-based systems work well when limited data are available. They are often applied to specialized domain-specific use-cases, where they achieve high precision but low recall and cannot be transferred to other domains.

2. Unsupervised learning approaches are often based on a clustering approach, in which NEs are extracted from the clusters based on semantic similarity.

3. In feature-based supervised learning approaches, machine learning algorithms are applied to carefully designed features, such as word-level information (e.g., case, morphology, POS-tag), lookup information in digital gazetteers, or document and corpus information (e.g., occurrence counts).

4. Finally, in deep learning NER—the dominant approach producing state-of-the-art results—a neural network learns (through training) to automatically extract features. The key element of learning is the combination of forward pass (calculating the weighted sum of inputs) and backward pass (calculating a gradient that is based on an objective function with respect to the model's weights) through multiple processing layers.

While NER identifies real-world entities directly, often, once introduced by name, they are subsequently referred to by a descriptive phrase (noun phrase) or a pronoun. For example, we might introduce Michael Jackson by his name, but later in the text refer to him as "the king of pop," "the famous musician," or simply "he." For the sake of language

understanding, it is important to comprehend which of the different referring expressions concern the same underlying real-world entity—a field of study known as coreference resolution. There can also be made a distinction between descriptors that can identify a real-world entity uniquely without additional context (a rigid designator) and those that require contextual information to be understood (a property which in linguistics is referred to as *deictic*). [Lan20] In this example, "the king of pop" is probably sufficient on its own in identifying the real-world entity Michael Jackson, while the noun phrase "the famous musician" needs context before it can be resolved.

Two terms often coming up in relation to coreference resolution are *discourse* or *discourse processing*. According to Stede [Ste12] *discourse processing* refers to language processing beyond the sentence boundary. After processing information of individual sentences, discourse processing augments the information, e.g., by looking at relationships between words originating from different sentences or by examining causal relationships between sentences. Underlying this approach is the assumption of *coherence* in a text, by which its constituting sentences do not exist in isolation but form meaningful relationships (causal or coreferential in nature) around a common topic.

In their review on neural Entity Coreference Resolution, Stylianou and Vlahavas [SV21] provide an overview of the development of CR approaches. They start with pre deep learning approaches in the following categories: Mention-Pair models, Mention-Ranking models, Entity-Based models, and Latent Structured models. Similar to other NLP tasks, deep learning models started to dominate also in CR, with the introduction of word embeddings by Mikolov et al. [MSC+13]. The DL methods evolved in an incremental way and in the same categories as the non-DL models by building on top of each other. The early Mention-Pair models quickly evolved into Mention-Ranking models, which are at the core of Entity-Based models. Latent Structured models and language models build on either Mention-Ranking models or Entity-Based models. A comparison of different implementations shows that the best results are currently achieved by latent structure approaches.

2.2.5 Sentiment Analysis and Opinion Mining

The aim of this work is to extract opinions from written text. There exist many terms related to the process of extracting opinions, e.g., text classification, sentiment analysis, opinion mining, and many more. We start by mapping out the field and determining what is relevant for this work. The terms *opinion mining* and *sentiment analysis* are generally regarded as synonymous. In a comprehensive (over 400 references) survey book covering all important topics and latest developments in the field up to 2012, Liu [Liu12] provides the following definition:

> *Sentiment analysis, also called opinion mining, is the field of study that analyzes people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes.* [Liu12, p.1]

He describes further that many different names exist under the umbrella terms *sentiment analysis* and *opinion mining* with slightly different meaning, e.g., *opinion extraction*, *sentiment mining*, *subjectivity analysis*, *affect analysis*, *emotion analysis* and *review mining*. He further states, that sentiment classification is a text classification problem. Zhang et al. [ZZL15] describe text classification as the problem of assigning predefined categories to free-text documents. Kowsari et al. [KMH+19] describe text classification as labeling a set of data points (i.e., documents, text segments) with a class value from a set of k different discrete value indices.

Based on the above analysis, the following conclusion is drawn. The most important terms for this work are *sentiment analysis* and *opinion mining*, which refer to the same area of research. The terms exist under many different names with slightly different scopes, which need to be considered. Information retrieval and text mining are areas where the techniques of opinion mining and sentiment analysis are applied to. The remainder of this section starts with a brief history of sentiment analysis and finishes with an exploration of the problem definition for various sub-problems.

History and Motivation Although NLP research dates back to the 1950s [ZDLS20], sentiment analysis research only began in the early 2000s. [Liu12] The explosion of opinion data on social media and the potential advantage that can be gained from analyzing and understanding it led to strong interest from politics, industry, and science in the field. Additionally, the amount of opinionated data that could easily be harvested from social media platforms enabled effective research in the first place. [SLC17] Research on text classification dates back further than sentiment analysis, to the 1960s, and gained a major popularity boost in the early 1990s due to the availability of more powerful hardware. [Seb02]

Sentiment analysis is a challenging task for many reasons. Detecting sarcasm can be difficult or even impossible without additional information, e.g., tone of voice and body language (in voiced opinions), the setting in which opinion is expressed, the history of the opinion holder in relation to the opinion target or the intent of the opinion holder. Another challenge is the potential for multiple (overlapping) sentiments in a single sentence. For example, the sentence "I am so glad I did not take the offer" expresses a positive sentiment of relief but also contains a negative sentiment towards "the offer."

Problem Definition The definition by [Liu12] allows for an exhaustive capturing of all sentiments in a single sentence. According to the definition an opinion is a quadruple (g, s, h, t) with g as the opinion (or sentiment) target, s the sentiment about the target, h the opinion holder and t the time when the opinion was expressed. The earlier used example sentence "I am so glad I did not take the offer" would contain two opinion quadruples. In the first one, I (opinion holder) have a positive sentiment (s) towards my action of not taking the offer (g). In the second one, I (h) have a negative sentiment (s) towards the offer (g). The time when the statement was uttered (t) is the same for both opinions.

The problem definition of the sentiment analysis task as a simplified version of Liu [Liu12] can be defined as: Given an opinion document d, discover all opinion quadruples (g, s, h, t) in d. A document d can be any text of arbitrary length, e.g., a single word, a sentence, or also an entire book. In this work we will use the same definition of opinion as a quadruple but will use an adapted problem definition (refer to section 3.2 for more details).

Sub-Disciplines We examine sub-problems of opinion mining as they are outlined by Liu [Liu12]. When the term sentiment analysis is used, often what is understood is the labeling of a text as positive or negative without considering a specific sentiment target. Liu refers to this as *document sentiment classification*. Hence, the opinion quadruple would take the form $(\_, s, \_, \_)$ since we are only concerned with the sentiment s.

While in document sentiment classification the scope of a document is not explicitly specified, when the scope is fixed to a single sentence, this special case is referred to as *sentence-level sentiment classification*. [TQW+15] In this sub-problem, it is assumed that one sentence contains at most one sentiment. It can be solved as a three-class classification problem (positive, negative, neutral or no sentiment), or in a two-step classification process, first filtering out sentences containing no opinion and subsequently performing sentiment classification. Filtering out sentences that contain no sentiment can be done by performing a *subjectivity classification* [HW00], in which sentences are labeled as either subjective or objective. [WBO99]

While document and sentence-level sentiment classification work on simplifying assumptions to reduce problem complexity, *aspect-based sentiment analysis* is a more exhaustive approach. Here the task is to extract all opinion quintuples (target entity, target aspect of entity, sentiment, sentiment holder, time when sentiment was expressed) from a text.

Opinion summarization is a field of study that deals with aggregating information from many opinions. One goal is to condense multiple different opinions into a single summarizing text. Another one is *aspect-based opinion summarization*, in which a summary text is created per entity and aspect, together with counts of positive and negative opinions. *Contrastive view summarization* deals with matching a positive and a negative opinion about the same aspect.

Other sub-disciplines of sentiment analysis include generalization across language (*cross-language sentiment analysis*) or domain (*cross-domain sentiment analysis*). *Multimodal sentiment analysis* is the discipline of combining multiple input types to improve the performance of classification algorithms. [JH18] For example, a video file could provide three input types—spoken text of actors, background music, and the visual layer—all of which can be used for determining a sentiment.

Implementations As sentiment analysis is such a broad field, the used approaches and algorithms depend on the specific problem. To get an idea of what is used and performs well, we look at *SemEval*, the international workshop on semantic evaluation. [PSS+21]

It provides a yearly set of around 12 challenges concerning language understanding of computers. The most recent and relevant (towards sentiment classification) one is *SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media.* The following result summary is taken from [ZMN+19]. The challenge consisted of three sub-tasks:

1. Classification of the tweets in *offensive* and *not offensive*
2. Classification of the offensive tweets in *targeted* and *untargeted*
3. Classification of the targeted tweets into one of the target types *Individual*, *Group*, or *Other*

Hence, based on the earlier problem definitions, (1) can be seen as a document-level sentiment classification task, and (2) and (3) can be approached as a general text classification task, or also as an aspect-based sentiment classification task. Nearly 800 teams participated in the challenge, of which 115 submitted their results.

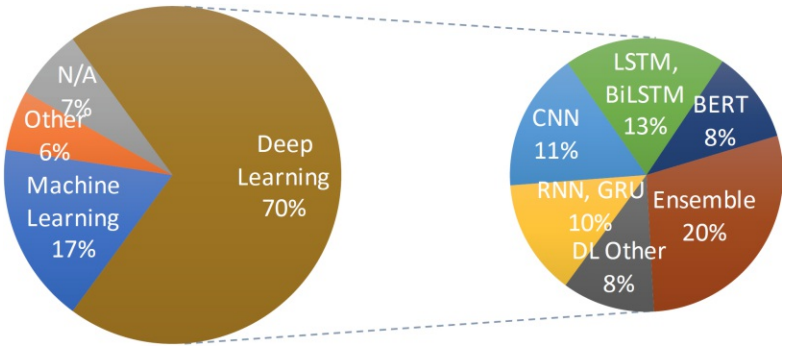


Figure 2.6: Distribution of submitted model types for the SemEval-2019 Task 6 (sub-task A) [ZMN+19]

Figure 2.6 shows the distribution of participating models. To no surprise, the majority consisted of deep learning approaches, followed by traditional machine learning approaches (e.g., SVM, logistic regression). The models were evaluated on the F1-Score. Overall, the best results were achieved by ensemble methods and state-of-the-art deep learning models such as BERT. In sub-task A, seven out of the top ten used BERT (the best had an F1-Score of 0.829), and the best non-BERT was an ensemble method (CNN with BLSTM+BGRU) ranked at place six (F1: 0.806). Interestingly, a rule-based approach took the top spot (0.755) in sub-task B. Ensemble methods outperformed pure BERT approaches in the second sub-task by taking second (0.739) and third (0.719) place, followed by a pure BERT at place four (0.716). On task C, the best model was again a pure BERT (0.660) but followed second by an ensemble of OpenAI Finetune, LSTM, Transformer, SVM, and Random Forest (0.628).

2.3 Supervised Machine Learning in NLP

So far, we have covered architectures and tasks or machine learning. In this section, we focus on the learning aspect of machine learning, particularly on supervised text classification, which is based on learning a model on a labeled dataset.

2.3.1 Dataset Creation

Having access to a sufficient quantity of high-quality data is an essential part of every successful machine learning project. In this section, we will focus on a particular issue that can arise in manually annotated datasets for classification tasks. Manual annotation involves subjective judgment and is prone to human error, both of which can introduce *label noise* into the dataset. Frénay and Verleysen [FV14] distinguish between the true label of a sample and its observed label. The observed label is subjected to a noise process which is referred to as label noise. Thus, label noise refers to noise in the labeling process. Not in the scope of label noise is *feature noise*, which refers to noise in the measurement process. For example, the inaccuracy of a thermometer would introduce feature noise into measured temperature samples. Being aware of the issue of label noise and knowing how to mitigate it is important because it can negatively affect the accuracy of predictions, make a trained model more complex than necessary and increase the required number of training samples. [FV14, GdCL15]

While label noise in image classification has received extensive research attention, label noise in text classification has received less attention. [JPLN19] Still, there are several studies on approaches to mitigating the impact of label noise in the latter. One approach is to perform outlier detection to discard noisy labels. Garg, Ramakrishnan, and Thumbe [GRT21] train a noise model, in conjunction with the main classifier, to predict the likelihood of the presence of label noise. Samples with a higher likelihood of label noise get assigned a lower weight having less impact on the network's training process. Ardehaly and Culotta [AC17] apply an enhanced label regularization technique to make their model more robust against noise. Malik and Bhardwaj [MB11] investigated an automatic label correction approach, in which samples with high-quality class labels are used to validate and correct the other samples. Overall, the methods to deal with label noise can be divided into three categories, as suggested by [FV14]:

1. Label noise-robust methods: Using models that are naturally more robust to label noise. Such models remain effective when there is only a small amount of label noise.

2. Data cleansing methods: Cleansing the dataset by correcting wrong labels or removing samples with wrong labels.

3. Label noise-tolerant methods (probabilistic or model-based): When information about label noise or its consequences is available, then the models can be designed in a way that considers label noise. One method is to train a label noise model

simultaneously to the classifier. These combined classifiers learn to predict the true label. Label noise-tolerant learning algorithms can be further divided into probabilistic methods and model-based methods.

2.3.2 Pre-processing

Often, it is beneficial or even necessary to perform pre-processing operations on a dataset before the use in machine learning models. Conforming to the model's input format, reducing processing time, or improving classification performance are possible motivations. Some examples of common pre-processing techniques specific to text processing are word stemming, lower-casing, and removal of unwanted tokens (e.g., URLs, HTML tags, stop words). Examples of pre-processing operations in a broader problem domain include reducing label noise and dealing with missing values.

Pre-processing operations can significantly improve the classification performance of machine learning models. [NL18, SRS14, HLS13] Therefore, it is important to apply the right pre-processing steps. Which pre-processing steps should be applied depends on the dataset, the algorithm, the machine learning model, and the task, as [JX17] suggests. They performed a comparison of the impact of six different pre-processing methods on the sentiment classification performance of four different classification algorithms on five Twitter datasets. The results showed that some techniques had a significant impact on classification accuracy while others barely affected it. Another study on the impact of pre-processing techniques for Twitter sentiment analysis [SEA18] shows interesting results specifically for pre-processing in neural networks. Using two datasets, they evaluated 16 techniques on four model types (CNN, linear regression, Bernoulli Naive Bayes, and linear SVC). For the CNN, only 2 to 3 (depending on the dataset) of the 16 techniques improved classification accuracy, while the other techniques worsened it. For the non-neural-network approaches, significantly more (5 to 11) pre-processing techniques improved performance. The results suggest that deep learning models benefit less from pre-processing than statistical models. In the context of sentiment analysis, the techniques performing best were lemmatization, replacing repeating punctuations, replacing contractions, and removing numbers.

Pre-processing can also reduce the complexity of machine learning models by removing information that is not relevant for the task (noise) because that information does not have to be modeled. Thus, pre-processing can lead to decreased model sizes, which result in faster training and classification times. [NL18] Additionally, cleansing the data from noise will probably reduce the amount of training data required to achieve the same level of classification performance.

In conclusion, the proper selection of pre-processing techniques is essential, as it can improve classification performance, lower model complexity, as well as training and classification times. What constitutes the proper selection of techniques depends on the task, the dataset characteristics (e.g., language, text format, used vocabulary), and the model architecture. Since the impact of pre-processing techniques depends on many

factors and comprehensive studies exist mainly on specific domains (e.g., sentiment analysis of Twitter posts in English), experimentation with different pre-processing techniques should be performed in other domains.

2.3.3 Validation, Verification and Evaluation

Validation, verification, and evaluation are three terms related to testing machine learning models, sometimes used synonymously. This section aims at exploring the differences, if there are any, and overlaps between the terms.

The Encyclopedia of Machine Learning and Data Mining (by Sammut [SW17]) describes model evaluation as an assessment of the efficacy of a learned model. Most of the time, the primary consideration is the predictive efficacy, that is, how useful are the model's predictions for the use case for which it was deployed. Some well-known criteria for measuring such usefulness are accuracy, precision, recall, and mean squared error. Other evaluation criteria include the model's size or its execution time. Regarding the other terms, verification means to build the system right, while validation means to build the right system. [SW17] We try to interpret this definition in the context of machine learning models: Building the model (system) right implies adhering to some specifications. Building the right model (system) means building a model that is useful in solving a task. The lines between verification and validation are blurry because arguably, one needs a specification to determine what useful behavior is. In the other direction, the specification is usually written to ensure the useful behavior of the model. The difference seems to be in the approach towards determining a model's properties. Verification appears to be more formal, while validation appears to be more empirical. More concretely, in terms of determining the predictive efficacy of the model, it would mean that verification aims at giving guarantees on all possible (unseen) inputs. At the same time, validation is testing the model with some unseen input and extrapolating expected behavior based thereon.

In the textbooks covering the fundamentals of machine learning, we found only little mention of the terms *model verification* and *model validation*. Furthermore, the term *verification* appears hardly at all, and the term *validation* occurs only as part of other terms (e.g., *cross-validation* and *validation set*). [MRT18, WBK20, Lan95] Many papers use the term *V&V* (verification and validation), which also appeared in Boehm's description of the "V-model" in 1984 [Boe84]. It describes the verification and validation (V&V) of software requirements and design specifications during the software lifecycle. In review papers about V&V of neural networks [BEW+18, TDM03], we did not find explicit definitions of the terms in the context of machine learning models. Therefore, we are left with the general interpretations in the context of software development and Boehm's definitions from 1984: *"Verification. The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase."* [Boe84, p.1] and *"Validation. The process of evaluating software at the end of the software development process to ensure compliance with software requirements."*

[Boe84, p.1] Those definitions are different only in that verification checks requirements during development, whereas validation checks requirements at the end of development.

In summary, the evaluation of machine learning models (or algorithms) is the process of evaluating their efficacy based on some property. Model evaluation subsumes the terms model verification and model validation. The difference between validation and verification in the context of machine learning seems not clearly defined. The general notion we have observed in literature is that if evaluation takes a formal approach (giving a guarantee on correctness), it can be called verification, and if it takes an empirical approach, it can be called validation. Additionally, the term V&V can be observed, but we did not find explicit definitions in the context of machine learning models.

2.3.4 Model Training and Evaluation

A machine learning model has to be trained on available data in order to efficaciously make predictions on unseen data. As [MRT18] put it, machine learning refers to computational methods that use experience to make accurate predictions. To determine the efficaciousness of such predictions, they also have to be evaluated. This section covers the basics of training and evaluation of models.

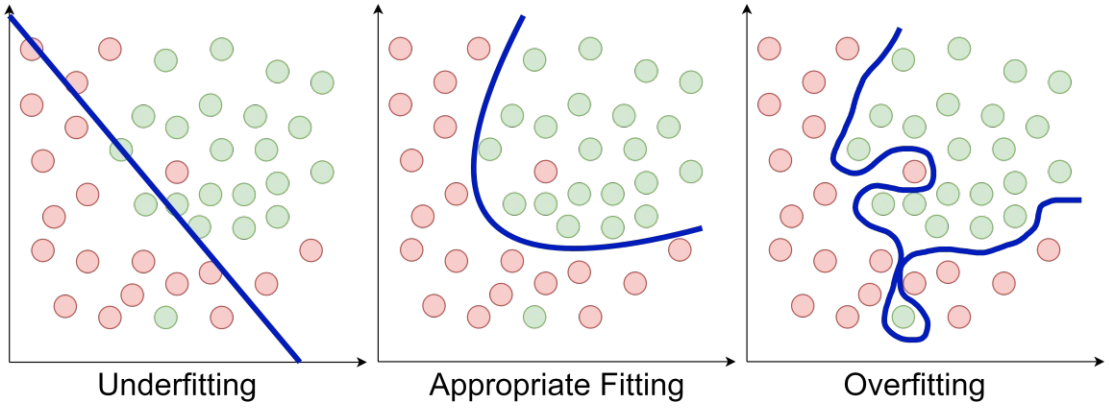


Figure 2.7: Showing the concept of under- and overfitting in a binary classification task in two feature dimensions. The blue line represents a classifier splitting the feature space into two regions. The classifiers, from left to right, are likely to generalize too much, appropriately, and too little.

Generalization A machine learning model should generalize from the data it was trained on to unseen data. Two dangers arise in the training phase. If the model is trained too specifically on the training data, it will likely not generalize well, which is called overfitting. In contrast, underfitting is if the model was not trained to be specific enough, meaning it generalizes too much and important details get lost. In both cases, classification performance on unseen data will drop. [MRT18] Figure 2.7 depicts the concepts of over- and underfitting. Various techniques can minimize the risks of over- or

underfitting. Another factor that has a major impact on classification performance is how well the samples used for training a model represent the total bandwidth of samples that can occur. For example, if the model is trained on outliers only, it has no chance to generalize to the real data.

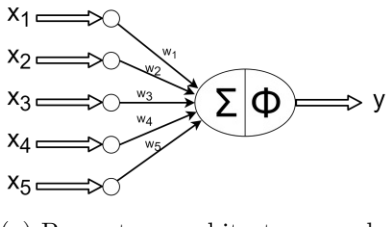
Data Splitting A machine learning model should not be evaluated on the same data it was trained on. One benefit of evaluating a model on "unseen" data is that it can show if a model was overfitted to the training data. [Ber19] If the performance is much better on the training data than it is on the test data, it could indicate overfitting. Different techniques exist for splitting a dataset into training and testing sets. We will discuss holdout, cross-validation, and stratification.

The holdout method splits the data into two disjoint sets by "holding out" some data for evaluation. The method does not specify on which criterion the samples are selected into the holdout set. A common approach is to choose them randomly. A shortcoming of the holdout approach is that the model is evaluated only on a single subsample of the data. Hence it could coincidentally get good results on the subsample, even though it would perform worse overall. Cross-validation approaches tackle that problem by training and evaluating the model multiple times on different dataset parts. The evaluation results are then averaged over all runs to better estimate the model's performance. [Ber19]

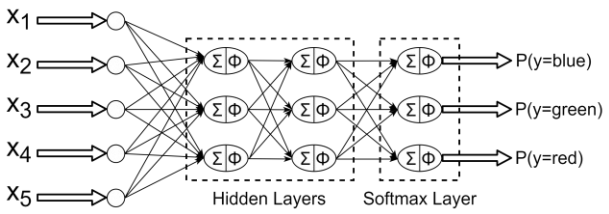
In exhaustive cross-validation (CV) techniques, the model is validated on all possible (as defined by the method) test sets. [AC10] For example, in *leave-p-out CV* in each run, p samples are used for the test set and $n - p$ for the training set. Since it is an exhaustive method, the total number of runs is $\binom{n}{p}$. In the non-exhaustive method *k-fold CV*, the dataset is split into k equally-sized sets. Each run, one set is used for validation, and the remaining $k - 1$ sets are used for training. In *Monte-Carlo CV*, in each run, a random subset of fixed size is selected as the test set, while the complementary set forms the training set. The process is repeated an arbitrary number of times.

A problem with randomly selecting samples into training and testing sets arises in imbalanced datasets. Those are datasets that contain significantly more samples of one class compared to samples of another class. By random selection, certain classes may become severely under or overrepresented. A solution to this problem is stratification by class. The idea is to preserve the ratio between classes in all subsets. For example, if we were to choose randomly in a 20% holdout approach from a dataset containing 100 observations of class 1 and 10 observations of class 2, it can easily happen that we will not choose any observations of class 2. Stratification on classes would ensure that 20 samples of class 1 and two samples of class 2 are chosen. If it is not possible to preserve ratios exactly, then they should be approximated. [AC10]

Learning Now that we have discussed different methods to choose a training set, we describe how a machine learning model learns from that data. Here we will look at learning in neural network models. To understand the training process, we need to understand the basic architecture of neural networks. The following explanations and



(a) Perceptron architecture, applying an activation function Φ to the sum of five weighted input values to produce an output value y.



(b) DNN architecture, with five inputs, two hidden layers and a softmax layer which is used to perform multi-class classification.

Figure 2.8: Basic DNN architecture [Agg18]

formulas are based on [Agg18]. Figure 2.8a shows a perceptron consisting of only one neuron. The perceptron takes n input values $x_1, \ldots, x_n$ and outputs a value

$$y = \Phi(\sum_{i=1}^{n} w_i \cdot x_i) \tag{2.14}$$

with Φ being an activation function (e.g., sigmoid) and $w_1, \ldots, w_n$ being the weights of the input connections. Those weights are adapted during the training to approximate a function that maps the input values of samples from the training set to their actual classes. The perceptron is the simplest neural network architecture. In order to approximate more complex functions, multiple layers of neurons (as in DNNs) are required. If the network should learn to predict more than two classes, the softmax layer architecture can be used: Let $c_1, \ldots, c_k$ be the possible classes and $\bar{v} = (v_1, \ldots, v_k)$ the outputs of the softmax layer, then the i-th output is calculated by:

$$\Phi(\bar{v})_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} \tag{2.15}$$

$\Phi(\bar{v})_i$ corresponds to the probability that the input given to the model is mapped to class c_i. While running an input through a neural network to calculate an output value is called forward pass, taking an outputted value and update the network's weights by calculating gradients of a loss function in relation to the network's weights is called backward pass. The loss function L tells the network how close its predictions were to the ground truth. Let $\hat{y}_1, \ldots, \hat{y}_k$ be probabilities outputted by the network. $\hat{y}_i$ is the probability that the input belongs to class c_i. Then, the cross-entropy loss is calculated as

$$L = -\log(\hat{y}_r) \tag{2.16}$$

with c_r being the correct (ground-truth) class. $\hat{y}_r$ assumes values between 0 and 1, corresponding to the probability of the network predicting the correct class. Therefore the loss is greatest (approaching infinity) when the network is furthest from the truth ($\hat{y}_r = 0$), and the loss is 0 when the network is sure to predict the correct class ($\hat{y}_r = 1$).

Finally, we describe how each weight in the network is updated by calculating the gradient of the loss function with respect to that weight. We describe the calculation on a simplified architecture with only a single sequence of hidden units $h_1, h_2, \ldots, h_k$ followed by a single output unit o. A more complex calculation based on dynamic programming and the multivariable chain rule of derivatives must be performed in a network where multiple paths exist from input to output. In the simple example, the gradient of the loss function L with respect to the weight of the connection between h_{r-1} and h_r is calculated by

$$\frac{\partial L}{\partial w_{h_{r-1},h_r}} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{h_{r-1},h_r}} \quad \forall r \in 1 \ldots k \tag{2.17}$$

We now describe the backpropagation process in more detail. The goal of backpropagation is to change the weights W of the network in such a way as to minimize the total classification error of all samples that were fed into the network. Its notion comes from the fact that incoming weights to a neuron are updated proportionally to the error produced by the neuron's activation value, starting from the last layer, proceeding layer by layer in the direction of the first layer. In other words, the error is propagated back through the layers. The calculation for a single weight in a simplified network is shown in equation 2.17. After the calculation was carried out for all weights ($\frac{\partial L}{\partial W}$), we have the gradient. It tells us which combination of relative changes to the weights will result in the maximum change of the loss. In other words, it tells us in which direction we have to move (from the current configuration of weights) to reduce the loss the most. The weights are updated via:

$$W \leftarrow W - \alpha \frac{\partial L}{\partial W} \tag{2.18}$$

with α being the learning rate or step size. Moving repeatedly in the direction of the negative gradient is known as *gradient descent*. It is common practice to perform the update on a batch $B = \{j_1, \ldots, j_m\}$ of randomly selected samples from the training set—referred to as *mini-batch stochastic gradient descent*—via

$$W \leftarrow W - \alpha \sum_{i \in B} \frac{\partial L_i}{\partial W} \tag{2.19}$$

The learning process is repeated on the training data until convergence or until another stopping criterion is reached. [Agg18]

Evaluation Metrics After training a model, it has to be evaluated. There are different performance metrics for evaluating a machine learning model on its efficaciousness to make predictions. We will explain the metrics used for multi-class (more than two classes) classification in this study, based on [GBV20]. The calculations are based on the confusion matrix, which shows per actual class how often each class was predicted.

Table 2.1 shows a sample confusion matrix with three classes c_1, c_2 and c_3. The rows show actual classes and the columns show predicted classes. For example, the third row

	Pred. c_1	Pred. c_2	Pred. c_3	Total Actual
Actual c_1	9	2	0	11
Actual c_2	3	7	1	11
Actual c_3	4	11	2	17
Total Pred.	16	20	3	39

Table 2.1: Example of a confusion matrix

shows, that for the 17 samples belonging to c_3, four of them were wrongly classified as c_1, 11 were wrongly classified as c_2 and two were correctly classified as c_3. The total number and predicted number of samples with c_k are denoted by $Actual_k$ and $Predicted_k$, respectively:

$$Actual_k = TP_k + FN_k \tag{2.20}$$
$$Predicted_k = TP_k + FP_k \tag{2.21}$$

For calculating the following performance metrics, we introduce four additional counts: The true positives TP_k are the number of times the predicted class was k when the actual class was k. The true negatives TN_k are the number of times the predicted class was different from k when the actual class was also different from k. The false positives FP_k are the number of times the predicted class was k when the actual class was different from k. Finally, the false negatives FN_k are the number of times the predicted class was different from k when the actual class was k.

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{2.22}$$
$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{2.23}$$

The metric $Precision_k$ indicates how often the model is correct where it predicts class k. A high precision value is required in applications when it is costly to make a wrong prediction, for example, in spam filtering. The metric $Recall_k$ indicates how many of the samples with actual class k the model predicts correctly. A high recall is important when it is more costly to miss a prediction than making a wrong prediction. An example could be a cyber security system guarding sensitive information against possible intrusion.

Accuracy is a popular metric that indicates how many samples are predicted correctly out of the total number of samples:

$$Accuracy = \frac{\sum_{k=1}^{N} TP_k}{\sum_{k=1}^{N} TP_k + FN_k} = \frac{\sum_{k=1}^{N} TP_k}{Total} \tag{2.24}$$

It is further possible to calculate the averages of precision and recall across all classes to get a better idea of the overall performance. There are two common ways—macro, and

weighted average:

$$MacroAveragePrecision = \frac{\sum_{k=1}^{N} Precision_k}{N} \tag{2.25}$$

$$MacroAverageRecall = \frac{\sum_{k=1}^{N} Recall_k}{N} \tag{2.26}$$

$$WeightedAveragePrecision = \sum_{k=1}^{N} Precision_k \cdot \frac{Actual_k}{Total} \tag{2.27}$$

$$WeightedAverageRecall = \sum_{k=1}^{N} Recall_k \cdot \frac{Actual_k}{Total} \tag{2.28}$$

For the macro-average values, the sum over the individual class values is taken and divided by the number of classes N. The macro-average treats each class with equal importance, regardless of how many samples belong to each class. The weighted averages calculate averages per class weighed by the number of samples in each class.

The F1-Score provides an aggregated metric of a model's precision and recall. It is the harmonic mean of precision and recall:

$$F1Score_k = \left(\frac{2}{Precision_k^{-1} + Recall_k^{-1}} \right) = 2 \left(\frac{Precision_k \cdot Recall_k}{Precision_k + Recall_k} \right) \tag{2.29}$$

When calculating the averaged F1-score over all classes, two different approaches can be found in the literature. In the first approach [GBV20], the averaged F1-Scores are the harmonic mean of their respective averaged precision and recall values:

$$MacroAverageF1 = 2 \cdot \left(\frac{MacroAveragePrecision \cdot MacroAverageRecall}{MacroAveragePrecision + MacroAverageRecall} \right) \tag{2.30}$$

$$WeightedAverageF1 = 2 \cdot \left(\frac{WeightedAveragePrecision \cdot WeightedAverageRecall}{WeightedAveragePrecision + WeightedAverageRecall} \right) \tag{2.31}$$

Another way of calculating the averaged F1-scores (found in [NPK+16]) is to first determine the F1-Score per class and then take the averages.

$$MacroAverageF1 = \frac{\sum_{k=1}^{N} F1Score_k}{N} \tag{2.32}$$

$$WeightedAverageF1 = \frac{\sum_{k=1}^{N} F1Score_k \cdot Actual_k}{Total} \tag{2.33}$$

The framework scikit-learn has implemented the latter way of calculating average F1-Scores.

2.3.5 Tooling and Infrastructure

Tools and infrastructure play an essential role in the effective training of machine learning models. Depending on the infrastructure, training times can vary significantly. Models can be trained on multi-purpose processors, e.g., conventional CPUs, GPUs, or specialized hardware, e.g., Tensor Processing Units (TPUs). In order to make use of the many cores that GPUs and TPUs offer, the network architecture must allow for parallelized computations. For example, attention models allow for better parallelization than simple RNNs, as discussed earlier. Wang, Wei, and Brooks [WWB19] have performed a detailed study on benchmarking CPU, GPU, and TPU (v2/v3) platforms for deep learning. According to them, domain-specific hardware becomes more and more relevant since improvements in computing power for general-purpose processors have become increasingly difficult to achieve.

Additionally, they found that the TPU architecture makes good use of parallelism due to batch size but does not exploit parallelism due to model depth to the same degree. Furthermore, GPUs show better flexibility for small batch sizes and computations other than matrix multiplication. CPUs achieve the highest FLOPS utilization on RNNs and support the largest models because of their large memory capacity. They found that the speedup of TPU over GPU depends heavily on the nature of the workload. They measured speedups of real workloads between 3 times and 6.8 times. In summary, TPUs are not always superior, although they are most of the time, and optimizing a model's architectural details is essential to gain the maximum benefit on the respective infrastructure. As specialized and also powerful hardware is expensive, renting cloud-based infrastructure can be a viable alternative.

Many tasks in machine learning and language processing have to be performed over and over again. This led to the creation of machine learning frameworks, libraries, and tools. There are general-purpose machine learning frameworks, e.g., Tensorflow[1] and Pytorch[2], and there are specialized libraries specifically for natural language processing, e.g., Spacy[3] and NLTK[4]. Due to the steadily increasing demand for language processing, companies started to offer it as a paid service (e.g., Google's OpenAI[5]). The choice of a machine learning framework not only affects obvious aspects, e.g., ease of use and general capabilities to solve certain tasks, but also impacts performance. [WWB19]

Many users share annotated and unannotated datasets on the web. If sufficiently close to the task domain of a project, such datasets can provide additional data on which machine learning models can be trained and benchmarked. To this end, we examine corpora (datasets) that could be helpful for this work. As is expected, finding unannotated corpora is easier than finding annotated ones, and finding annotated corpora containing political speeches is even more challenging. Annotation types are manifold,

[1] https://www.tensorflow.org/, accessed: 2021-09-20
[2] https://pytorch.org/, accessed: 2021-09-20
[3] https://spacy.io/, accessed: 2021-09-20
[4] https://www.nltk.org/, accessed: 2021-09-20
[5] https://openai.com/, accessed: 2021-09-20

but the only ones which are for sure relevant (both during training and validation) to this work are sentiment annotations. The problem with topic annotations is that it is implausible to find an annotated corpus containing topics applicable to our task. Other common annotation types (e.g., part-of-speech, named entity recognition) could improve classification performance by providing additional information to the classification model.

Various annotated corpora for German sentiment analysis exist on the web. However, most of them seem to focus on shorter messages and simple language, e.g., Twitter posts, comments on news sites, movie/product reviews, and news headlines. It is unclear how well a model, trained on these kinds of texts, will perform on political speech, which is very different in many aspects (e.g., longer sentences, more accurate grammar, more vocabulary, no emoticons, fewer slang words, less offensive, fewer made-up words, and fewer spelling errors).

"One Million Posts" [SST17] is a dataset containing posts from an Austrian newspaper website in the German language. From those, 3599 are annotated by a sentiment label. Unfortunately, only 1% of those labels are positive, while 52% are neutral, and 47% are negative, which will make it challenging to train for positive sentiment. SB-10k [CDEU17] is a sentiment corpus of 9738 Twitter posts featuring the following labels (with frequency): *positive* (1682), *negative* (1077), *neutral* (5266), *mixed* (330), and *unknown* (1428). Barbaresi [Bar18] published a searchable text archive containing German political speeches from 1990 onwards. They contain no annotated sentiment labels or topic labels but are searchable by a query language. *GermaParl* [Bla] contains unannotated plenary speech protocols from the German Bundestag. There also exist sentiment word lists (e.g. SentiWS [RQH10]), which assign to each word a value between -1 and 1, indicating its connotation (negative/positive). Such lists can be used to calculate a cumulative polarity value for a text to determine the overall sentiment.

In summary, the supply of applicable annotated corpora that are useful for this task is low. The most promising corpus found was SB-10k [CDEU17], which is of the German language and has a well-balanced distribution of relevant sentiment labels. The selection of English corpora is significantly more extensive, and there even exists an annotated corpus of political debates. [GBZ18]

2.4 Related Work

This section examines related work with the following four goals in mind.

1. Narrowing down the field of NLP, identifying sub-fields and related areas, to get a better understanding of terms to search for.
2. Finding sources on sentiment and topic analysis, focusing, as much as possible, on German corpora in the political domain.
3. Finding corpora in the German language, annotated with sentiment and topic labels, since they are helpful for training and validation.

4. Searching for papers that use NLP methods to quantify the consistency of opinions over time. This implies looking for sources that combine sentiment analysis and topic analysis.

This project deals with the NLP sub-field of natural language understanding (NLU)—the discipline of machine reading comprehension. For consideration, the analyzed text will be written in German, and, compared to English, the state-of-the-art reading comprehension will lack behind. There are different goals in NLU. Question answering, sentiment analysis, determining the topic of a text, and machine translation are some of them. Relevant for this work will be a combination of topic classification and sentiment analysis.

2.4.1 NLP in General and Related Fields

In their survey paper, the authors of [ZDLS20] view NLP from three perspectives: modeling, learning, and reasoning. We will describe each of them briefly and relate them to this project. Modeling describes the task of creating a neural network structure that can take an encoded natural language sentence and turn it into a sequence of labels or another natural language sentence. In our case, we want a sequence of labels, i.e., the discussed topics and the politician's sentiment. The main modeling techniques used are word and sentence embedding and sequence-to-sequence modeling. Learning deals with the training of the network parameters. In NLP, a multitude of learning algorithms is used. Supervised methods perform very well when enough labeled data are available. If that is not the case, unsupervised methods can be applied. In this project, we use supervised methods based on a manually annotated dataset. Finally, they describe reasoning as the process of generating answers to unseen questions (i.e., questions for which an NLP algorithm did not produce the answers right away) by inferring from available information. In our case, the reasoning part would entail making statements about politicians' consistency on their opinions by analyzing their expressed sentiments for specific topics (determined by the NLP algorithm) over time.

Keyphrase extraction is the technique of automatically reducing a text to some key phrases, containing a summary of the original text that preserves essential information only. Among other tasks, it is helpful for document clustering and classification. [PT20] Keyphrase extraction could be used as a pre-processing step to improve classification performance.

Transfer learning is a subfield of machine learning that studies how the knowledge gained in one domain can apply to other domains. The work of Ruder [Rud19] deals with transfer learning in natural language processing. While transfer learning concepts might come up indirectly during this work, they will not be of primary concern.

2.4.2 Opinion Consistency in Politics

This section checks if specific research exists on using NLP to measure the consistency of expressed opinions over time. The aim is to find those that satisfy as many of the

following aspects as possible:

- Combining sentiment analysis and topic classification
- The evaluation of change in opinions over time
- The domain of political speeches
- A German-language corpus

The technical term for combining sentiment analysis with topic analysis is called *aspect-based sentiment analysis* (ABSA). This sub-discipline of NLP considers the aspects of a text as targets for sentiments perceived in the same text. [NGK20]

ABSA in the Political Domain The authors of [GBZ18] performed ABSA on presidential debates between Hillary Clinton and Donald Trump. Their work provides two main contributions. Firstly, they provide an annotated corpus with sentiments and the aspects *agenda*, *united states*, *group*, *opposition*, *self*, *women*, and *other* in two different annotation schemata. Secondly, they show that the chosen schema has a substantial impact on result performance.

The authors of [AMPZ17] performed ABSA on political news articles. Their work is especially relevant for several reasons. First, because it is one of the few that apply ABSA on larger documents (compared to most, which work with shorter social media posts) in the political domain, secondly, they share the annotated corpus, which contains both sentiment and aspect annotations. Unfortunately, the language is not German, and it remains an open question if corpora in other languages are useful for this project. Thirdly, they develop a classification algorithm and share performance evaluations. Finally, they interpret the results by fitting them into the political and social context.

ABSA on German Language Corpora Only one relevant paper was found. Kersting and Geierhos [KG20] implemented a neural network algorithm to perform ABSA. In order to evaluate their algorithm, they collected German physician reviews and manually annotated them. The dataset contained 11,237 sentences annotated on the aspects "friendliness", "competence", "time taken", and "explanation". The authors tested different opinion extraction methods, e.g., using frequent nouns, making use of opinion and target relations, supervised learning, and topic modeling. They concluded that only supervised approaches were promising. Their algorithm, mainly based on a bidirectional LSTM, achieved an average F1-Score of 0.8 over all four aspects. Their contribution is especially relevant because it was the only one that performed ABSA on a German language corpus.

Evaluating Consistency of Opinions Chandio and Sah [CS19] analyzed changing opinions on four topics relating to UK's decision to leave the EU—*Brexit*, *EU*, *Theresa May*, and *Jeremy Corby*. For each topic, they collected Twitter messages from four

periods. They used a keyword search, with a representative keyword for each topic, against Twitter's API to collect the messages. Then, they used NLTK[6] and TextBlob[7] to calculate a polarity value for each tweet and plotted the proportions of positive, negative, and neutral posts as a pie-chart per topic and time period. The results show that the proportion of positive tweets for Brexit was larger in 2017 (around 32%) than in January of 2019 (29%). After parliament voting in February of 2019, the proportion of positive tweets dropped to around 27%. However, the number of negative tweets on Brexit decreased as well—from 23% to 16.3%. For the keyword EU, the results show a shrinking amount of positive tweets (from 38% to 30%), as well as negative tweets (24% to 18%). According to the authors, the data further shows that people are more supportive of Jeremy Corbyn than Theresa May.

The authors of [CGG+07] present a framework for letting users express their opinions and visualizing individual and collective sentiments over time. Although the paper focuses more on design considerations of a front-end application and less on an actual algorithm, it still provides inputs for designing a platform visualizing opinions.

Evaluating Trustworthiness of Statements Although there is a decent number of articles dealing with assessing the trustworthiness of statements (e.g., fake news detection), none could be found on assessing the trustworthiness of people (or politicians).

In summary, we found no studies combining all of the aspects defined above. The highest number of satisfied aspects in a single paper was two, which shows that this work is a novel contribution.

[6] https://www.nltk.org/, accessed: 2021-09-20
[7] https://textblob.readthedocs.io/en/dev/, accessed: 2021-09-20

CHAPTER 3

Design

This chapter describes the research method, derives requirements for the experiments from the research questions, defines opinion consistency, and documents the dataset creation process.

3.1 Research Method

In this work, we performed experimental research, i.e., we started with a vision in mind but did not know at the time how to get there or how far we could reach with the available resources. That is why the project followed an iterative approach of multiple phases of exploration, design, and implementation. The project specifications were kept loose and open initially and were narrowed down and concretized with increasing project duration, based on gathered insights along the way. The research questions (Q1–Q4) were designed accordingly: Open enough to allow for different implementations but concrete in answering how useful and practically feasible the chosen implementation will be.

The vision was to develop a system that is capable of monitoring the consistency of opinions over time. In order to do that and to answer the research questions, we had to define a formula that can make the consistency of opinions quantifiable. Coming up with such a formula was relatively easy and is described in Section 3.2. The hard part was to define how exactly an opinion can be extracted from a piece of text. It was not easy to put the process that a human performs for identifying opinions into an exact definition. As such, this was an exploratory process (described in Section 3.3) of different ideas, trading off the subtlety of captured opinions with the feasibility of implementation.

After the initial exploration phase, we decided to progress iteratively by first performing supervised opinion classification on a single topic, with the option to expand to more topics or different methods in subsequent iterations. Also, the dataset creation (described in Section 3.4) involved an exploratory process because the manual labeling of opinions

involves many uncertainties. The first iteration of opinion classification (Section 4.1) yielded a low accuracy. With the suspected reason being the small dataset size, we performed a second iteration (Section 4.2) on a larger dataset and were able to improve classification performance significantly. The results of those classification experiments were used to answer research question Q2.

After the second iteration of classification experiments, we moved on to utilizing opinion data. In Section 4.3 we explored how opinion data can be visualized in a useful way (Q4) and in 4.4.1 we investigated the usefulness of visualizing opinion consistency (Q1b). To answer research question Q3, we analysed the impact of model performance on the resulting visualizations in Section 4.4.2. The answer to Q3 also helped in the better answering of Q1a—the practical feasibility of monitoring opinion consistency through the means of supervised ML methods.

3.2 Requirements: A Definition of Opinion Consistency

We start by deriving requirements from the research questions outlined in Section 1.2:

R1 Precise Definitions: The terms *opinion consistency* and *opinion* have to be well-defined, because all other results (Q1–Q4) depend on those definitions. Therefore, those definitions must be easily comprehensible in order to put the results into context.

R2 Extracting Opinions: A method for extracting opinions from text has to be established and documented. The extracted opinions are subsequently also referred to as *opinion data*, and are required for calculating the opinion consistency. (Q1, Q2, Q4)

R3 Measurability and Comparability: The classification results of the used ML algorithms to predict opinions have to be measurable in order to make them comparable. (Q2)

R4 Transparency and Reproducability: The experiment conditions have to be well documented in order to make the experiments reproducible. (Q1–Q4)

R5 Visualizations: To help in determining the feasibility and usefulness of monitoring and visualizing opinion consistency and opinion data in general (Q1, Q4), at least the following graphs should be created:

- A graph that compares the opinion consistencies of multiple speakers over time.
- A comparison of the actual vs. predicted opinion consistencies.
- A comparison of the actual vs. predicted opinion data of extracted opinions.

R6 Estimation of Accuracy: In addition to visualizing opinion consistency based on experimental data, an effort should be made to directly determine the theoretical accuracy of predicted opinion consistency values based on the classification capabilities of underlying machine learning algorithms. (Q3)

In fulfillment of R1, the remainder of this section establishes the definitions of the terms *opinion* and *opinion consistency*. To answer the research questions, we need a way of quantifying the consistency of opinions. We build on Liu's [Liu12] definition of opinions as quadruples, as described in Section 2.2.5. An opinion (g, s, h, t) has a target $g \in G$, expresses a sentiment $s \in S$, and is held by the opinion holder $h \in H$ at time $t \in \mathcal{N}$. O denotes the set of all extracted opinions. In our case, the following interpretations apply:

1. The set of all opinion targets G contains ideas discussed in the Austrian parliament.

2. The possible sentiments $S := \{\text{POSITIVE}, \text{NEGATIVE}, \text{NEUTRAL}\}$ represent the speaker's stance towards the opinion target. A POSITIVE/NEGATIVE sentiment means that the opinion supports/resists the idea. A NEUTRAL sentiment means that the opinion neither clearly supports nor resists the idea.

3. The set of opinion holders H contains all speakers who expressed an opinion in parliament. A speaker h can belong to a political party P, denoted by $h \in P$. The set of all parties is denoted by $\mathbf{P}$.

4. The time t is a timestamp of the date when the speaker expressed the opinion.

Opinion consistency should be high when the number of contradicting opinions is low and vice-versa. Two opinions (g_1, s_1, h_1, t_1) and (g_2, s_2, h_2, t_2) are contradicting each other if $g_1 = g_2$ (they refer to the same topic), $s_1 \neq s_2$ and $s_1 \neq \text{NEUTRAL}$ and $s_1 \neq \text{NEUTRAL}$ (one has a positive sentiment and the other one has a negative sentiment). In this study, we focus on the opinion consistency of a single speaker ($h_1 = h_2$) or the speakers of a political party P ($h_1, h_2 \in P$).

Next, we define three variables to count the number of positive, negative, and neutral opinions. We count opinions for a subset of topics $G' \subseteq G$, a subset of speakers $H' \subseteq H$, up to point t' in time:

$$\mathit{Positive}(G', H', t') = |\{(g, \text{POSITIVE}, h, t) \in O \mid g \in G' \wedge h \in H' \wedge t \leq t'\}| \quad (3.1)$$

$$\mathit{Negative}(G', H', t') = |\{(g, \text{NEGATIVE}, h, t) \in O \mid g \in G' \wedge h \in H' \wedge t \leq t'\}| \quad (3.2)$$

$$\mathit{Neutral}(G', H', t') = |\{(g, \text{NEUTRAL}, h, t) \in O \mid g \in G' \wedge h \in H' \wedge t \leq t'\}| \quad (3.3)$$

We have not yet defined how neutral opinions should affect opinion consistency. Neutral opinions should never affect opinion consistency negatively, but they could increase

opinion consistency. We propose two formulas:

$$OpCons_1(G', H', t') = \frac{max\{Positive, Negative\} + Neutral}{Positive + Negative + Neutral} \tag{3.4}$$

$$OpCons_2(G', H', t') = \frac{max\{Positive, Negative\}}{Positive + Negative} \tag{3.5}$$

In equation 3.4 neutral opinions increase opinion consistency, while in equation 3.5 they have no effect on it.

The proposed definitions fulfill our requirements of making the consistency of opinions quantifiable and comparable. The value correlates positively with the proportion of opinions expressing a non-contradicting sentiment. Furthermore, these definitions are flexible. For example, to calculate a value for a single speaker h on a single topic g, we set $H' := \{h\}$ and $G' := \{g\}$. Or, if we want to calculate the opinion consistency of all speakers in a party P we can set $H' := P$.

These definitions imply that the values will become more insensitive to changing opinions the more opinions are collected over time. Other, more complex calculations could counter that problem. Improved methods could use a rolling window in which opinions are considered or weigh recent opinions more strongly. In this work, we will use the simple definitions from equation 3.4 and equation 3.5 and leave the study of more complex ones to future work.

3.3 Experiment Design

In fulfillment of R2 we had to establish a method for extracting opinions from text. This was an exploratory process, that started in the design phase (this section) and continued through the dataset creation phase (Section 3.4) and to the early stages of opinion classification (Section 4.1). This section documents the process up to the point were we had enough information to start creating a dataset. After we had a definition of opinion consistency, before we could proceed with the experiments, the following questions required an answer:

1. How to identify an opinion in a text document?
2. On which topics should we extract opinions?
3. How should an opinion be extracted on the technical level?
4. How should the data be represented?

We started with the first two questions. To that end, we downloaded a number of speech protocols from the Austrian parliament website[1] and tried to extract opinions manually

[1] https://www.parlament.gv.at/, accessed: 2021-09-24

by reading through them and highlighting text passages from which opinions could be derived. It quickly became apparent that this was a considerably complex task for the following reasons: Without narrowing down the scope of what to look for, each statement can potentially have multiple layers of opinions. Some opinions only become apparent with more context information. Furthermore, some opinions are more apparent than others. Unless an opinion is stated directly and without room for interpretation, which is rarely the case, their identification involves a subjective judgment.

We concluded that it is best to start with a simple method and a narrow scope. For that reason, we decided to extract opinions on the sentence level. In order to determine whether a sentence is of relevance to the topic of interest, we chose to use a keyword search. Regarding the second question, we decided to focus on a single topic, at least in the first experiment. The topic should be polarizing so that diverse opinions exist and it should be relevant so that enough opinions are expressed. The chosen topic that fulfilled both requirements at the time was the discussion about *lockdowns* as a measure to prevent the spread of the coronavirus.

Now that we narrowed the scope to a specific topic, it was unclear how to extract opinions on that topic on the technical level. First, we had to decide between supervised and unsupervised methods. The advantage of the latter would be that a manual annotation process would not be necessary. Ultimately, we decided to use supervised methods because, in opinion mining, they usually outperform unsupervised methods. [SLC17] We dedicated some time to explore rule-based approaches but realized after a short time that statistical and neural network approaches were more promising.

Intermediate Data Formats At that point, we had an answer for the second question and narrowed down the answers to the other questions. Before we could progress further, we required more insight that could be gathered only through experimentation on the data. To support experimentation and analysis on the data, we had to bring the raw data of speech protocols from the HTML format to a format suited for the processing by machine learning algorithms. Since we did not yet know how we would identify opinions, we designed the intermediate data formats with maximum flexibility in mind. We came up with three file formats, called *primary*, *secondary*, and *tertiary*.

The primary format is a comma-separated values (CSV) file that contains the following fields:

- `speaker`: Contains the speaker's title(s), name, and party affiliation.
- `speech`: Contains the speaker's transcribed speech from the moment they begin to speak up to the moment they are interrupted by the president or are done speaking.

In the secondary format (also CSV), the speeches of the primary format are split along sentence boundaries and enhanced with additional information, resulting in the following fields:

- `sent_id`: A unique identifier of the sentence.
- `date`: The date when the sentence was uttered by the speaker.
- `protocol_id`: The id of the protocol in which the sentence is contained.
- `party`: The party affiliation of the speaker that uttered the sentence.
- `speaker`: The speaker that uttered the sentence.
- `governing`: A truth value that indicates whether the party of the speaker was governing at the time the sentence was uttered.
- `text`: The transcribed sentence.

The tertiary format is in the CONLL-X format and contains for each sentence all fields of the secondary format as a comment and additionally the sentence analysis in the CONLL-X format with the following columns (descriptions from [BM06]):

- `ID`: Token counter, starting at 1 for each new sentence.
- `FORM`: Word form or punctuation symbol.
- `LEMMA`: Lemma or stem of word form.
- `CPOSTAG`: Coarse-grained part-of-speech tag.
- `POSTAG`: Fine-grained part-of-speech tag.
- `FEATS`: Unordered set of syntactic and/or morphological features.
- `HEAD`: Head of the current token, which is either a value of `ID` or zero ('0').
- `DEPREL`: Dependency relation to the `HEAD`.
- `PHEAD`: Projective head of current token, which is either a value of `ID` or zero ('0'), or an underscore if not available.
- `PDEPREL`: Dependency relation to the `PHEAD`, or an underscore if not available.

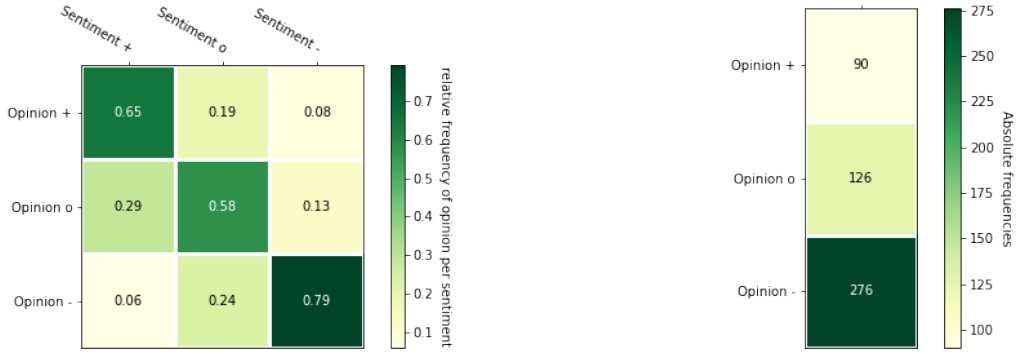
After the three intermediate formats were defined, we started with downloading session protocols in the *HTML* format from the government website. It has to be noted that it takes the transcribers a considerable amount of time before they make the final protocols available. At the time, the finalized protocols were lacking behind approximately five months. To bring the speeches from HTML format to the primary format, we implemented a document parser in Python. We chose to directly apply the pre-processing step of removing HTML tags from the speeches since the goal was to extract opinions only from words uttered by the speaker, without access to additional meta-information.

After transforming all available protocols from the HTML format to the primary format, we implemented two additional parsers, one to convert the primary format into the secondary format and one to convert the secondary format into the tertiary format. To generate the dependency parse for the tertiary format, we used the ParZu library [SVS13]. Finally, we had the data of all available protocols, available in the intermediate formats, for further processing.

3.4 Datasets: Dataset Creation and Analysis

In Section 3.3 we defined some parameters of the first experiment. We chose the topic *LOCKDOWN* and decided that we would gather opinions on the sentence level. Furthermore, we decided to use a keyword search to identify sentences of relevance, i.e., those that are concerning the chosen topic.

We used the regular expression `[lL]ock.?[dD]own` to filter for sentences that concern the topic. This gave us a selection of 492 sentences. As a consequence of the decision to use supervised machine learning approaches to extract opinions, we had to manually annotate the 492 sentences. In the first approach, we tried to assign one of three labels representing the speaker's opinion on the question of lockdowns to each sentence. This proved to be difficult because we constantly doubted whether our definition of opinion was still the same as in the beginning. Therefore, we wrote a definition down, but that did not solve the problem because there constantly appeared border cases that were not covered by the definition.



(a) Opinion distribution per sentiment

(b) Overall opinion distribution

Figure 3.1: Relationship between sentiment and opinion categories in the first dataset

Additionally, in the first classification attempt, we were conservative with giving subjective opinion labels (+ or -), and as a result, only 49 out of 492 samples were subjective. We anticipated that this could be a problem for the machine learning algorithm to learn from such a small sample size. Therefore, we performed a second classification attempt with a bias towards assigning subjective labels. In the second attempt, the number of

subjective labels increased to 366. Figure 3.1b shows the distribution of labels of the second attempt.

In an attempt to approach the annotation process in a more objective way, we assigned labels for multiple categories per sample, each with a more specific definition. We annotated the data on the following seven categories:

1. General sentiment of the sentence (+/−/○)
2. Speaks about somebody else's opinion (x = no, # = speaks about somebody else's opinion, ## = speaks about somebody speaking about somebody else's opinion)
3. Explicit support (+ = expresses explicit support for lockdowns, − = expresses explicit resistance against lockdowns, x = expresses neither support for nor resistance against lockdowns)
4. Impact (+ = mentions explicitly that lockdowns have a positive impact, − = mentions explicitly that lockdowns have a negative impact, ○ = talks neutrally about the impact of lockdowns, x = does not mention the impact of lockdowns)
5. Organisation (+ = expresses positive sentiment towards the organisational aspects surrounding the implementation of lockdowns, − = expresses negative sentiment towards the organisational aspects surrounding the implementation of lockdowns, ○ = talks neutrally about the organisational aspects surrounding the implementation of lockdowns, x = does not mention the organisational aspects surrounding the implementation of lockdowns)
6. Overall opinion, less subjective: The overall opinion on lockdowns with a bias towards neutral opinions (+/−/○)
7. Overall opinion, more subjective: The overall opinion on lockdowns with a bias towards non-neutral opinions (+/−/○)

The labels of categories 3–5 can be prepended by # or ## to denote the opinions of other speakers. To illustrate the nuanced and subjective nature of the labeling process, we go through two example sentences from the LOCKDOWN set. Sentence 1:

> *Egal wann es einen solchen Lockdown gibt, für die Wirtschaft gibt es keinen guten Zeitpunkt für eine solche Maßnahme, und gerade vor dem anlaufenden Weihnachtsgeschäft ist dieser Schritt natürlich besonders schmerzhaft.*

and Sentence 2:

> *Wir wissen, dass wir keine tatsächliche Berichtigung von einer tatsächlichen Berichtigung machen können, aber, Herr Loacker, ich möchte das hier schon richtigstellen: Sie behaupten, Kollegin Hebein hätte gesagt, sie kann sich einen zweiten Lockdown vorstellen.*

Table 3.1 shows the labels we have assigned in the seven categories. For the first sentence, we have assigned an overall negative sentiment (C1 = −). The speaker mentioned that a lockdown is bad for business and especially bad for the Christmas business (C4 = −). The speaker did not explicitly express whether they are for or against a lockdown (C3 = x). They did not talk positively or negatively about the organizational aspect, but they talked neutrally about the timing of a lockdown (C5 = ○). We found that even though they mentioned the negative impact of a lockdown, the way in which they have formulated the sentence implies that they think there is no alternative to a lockdown, which means they are ultimately for a lockdown. Since this is a rather subjective interpretation, we have assigned a neutral opinion in the conservative category (C6 = ○) and a positive opinion in the interpretative category (C7 = +).

The second sentence shows an example of a sentence in which the speaker addressed a statement of colleague Loacker, in which he addressed colleague Hebein, who presumably expressed a positive opinion on a lockdown. Accordingly, we have assigned C2 = ## and C3 = ##+. We considered an overall sentiment of neutral and negative but ultimately went with negative since the sentence is confrontational (C1 = −). Furthermore, the speaker did not talk about the impact (C4 = x) or organizational aspects (C5 = x) and we cannot deduce an opinion for or against lockdowns (C6 = C7 = ○).

	C1	C2	C3	C4	C5	C6	C7
Sentence 1	−	x	x	−	○	○	+
Sentence 2	−	##	##+	x	x	○	○

Table 3.1: The annotations on the two example sentences, according to the seven categories.

The first idea was to rely on explicit support (category 3) only since this would be the most objective way of determining the opinion. An analysis of these data revealed that politicians rarely expressed explicit support or resistance (only 107 out of 492 times). The politicians more frequently expressed a subjective (non-neutral) opinion on the effects of a lockdown (172 times, category 4) and on the implementation details of a lockdown (138 times, category 5). Considering the low amount of explicitly expressed opinions, we concluded that we had to include more subjective opinions as well.

Another idea was to determine the opinion directly from the sentence's overall sentiment (category 1) if the correlation between the overall sentiment and the opinion (category 6 and 7) would be high enough. We plotted the sentence's sentiment against the overall opinion of category 7. Figure 3.1a shows the relative frequency of opinions per sentiment. For example, in the third column, we see that when a sentence has a negative sentiment, it is labeled as a supporting opinion in 8%, as a neutral opinion in 13%, and as a resisting opinion in 79%. Although the correlation is relatively high for negative sentiments, it is not high enough for positive and neutral ones. Ultimately, we came to the conclusion that it was easiest to predict the opinions directly from category 7 and proceeded with the classification on the first dataset (see 4.1).

Second Dataset The classification results of the first experiment were relatively poor. One explanation for the poor results was that the dataset was too small compared to the complexity of the task. Under that assumption, the classification algorithm would not have access to enough training samples to learn all the features that can be used to extract opinions. To examine the impact that dataset size has on the classification performance, we planned to collect a second dataset that was significantly larger than the first one. The first dataset contained approximately 500 entries. The second one should contain at least ten times as many records.

With a defined target for the dataset size, we had to choose a topic that could produce around 5000 records. We tried various terms that were related to the coronavirus pandemic, but it turned out that none of the topics produced nearly enough samples. We looked at possibilities to gather more data from the government website and found a section with tentative speech protocols that are in a preliminary state. Initially, we avoided those protocols because their format is more difficult to parse than that of the finalized ones. Since we required additional data, we had no other choice than to implement another parser that could transform the preliminary protocols to the primary format.

After parsing the preliminary protocols into the intermediate file formats, the number of available sentences increased significantly. We experimented with different regular expressions and grouped similar ones to form topics. The finalized topics, together with their respective regular expressions, are:

- MASKS: `mask|ffp2|mund.?nasen`
- VACCINES: `impf`
- TESTING: `testet|testung|tests|testen|pcr`
- DISTANCING: `distanc|abstand|social.d`
- LOCKDOWN: `lock.?down`

We filtered for sentences that contained at least one of the patterns. The resulting number of sentences per topic is shown in the following table:

Topic	Sentences
MASKS	799
VACCINES	2298
TESTING	1641
DISTANCING	410
LOCKDOWN	855

Table 3.2: Number of sentences per topic

Since none of the topics came close to the target of 5000 samples, we decided to combine all five topics under the topic *MEASURES*. The topic *MEASURES* contains opinions on measures against the spread of the coronavirus. In total, we could gather 5573 sentences on this topic. The total does not equal the sum of individual topics because one sentence can belong to multiple topics.

One of the reasons for gathering the second dataset was to examine if a larger dataset improved classification performance. We argue that the fact that the second dataset is a superset of the first one could help to a minor degree in making the results comparable, but more importantly, it does not hurt the results. Let us say the classification performance is a function of dataset size and the complexity of samples. If we had two mutually exclusive datasets, with the samples in one of the datasets being significantly easier to predict than those of the other, the impact of dataset size would become less important. We argue that the impact of this relation is low in this case because the first dataset has only one-tenth of the samples of the second one. To better study the impact of dataset size, we could have used cross-validation on multiple subsets of the second dataset, but we leave that to future study.

Figure 3.2: Screenshot of the annotation software that aided in the annotation process of the second dataset

After we had gathered the samples for the MEASURES dataset, we had to label them

with the opinion labels. Since this required a considerable effort, we decided to implement an annotation software (Figure 3.2) that aided in the process. The key features of the software are:

- Displaying of unlabelled records in random order and without the speaker's name or their party affiliation to ensure that the decision is not influenced by meta-information that would not be available to the machine learning algorithm.
- Loading of unlabeled records from one file and storage of the labeled record in another file. The software remembers which records were labeled already.
- Convenience features to speed up the process. They include assigning a label via hotkey and highlighting keywords.
- Displaying of context information in the form of the previous and the subsequent sentence. If context information is used in the labeling process, then the dataset will also contain it, i.e., the machine learning algorithm has the same information as the human annotator has.

In the next chapter, we document how we used different classification algorithms to predict the opinions of records from the two datasets.

CHAPTER 4

Experiments

This chapter documents the opinion mining process, with the help of various machine learning algorithms, on the first (Section 4.1) and second (Section 4.2) dataset. In Section 4.3, we visualize opinion data from the second dataset aggregated per speaker and party. The chapter is concluded in Section 4.4, where we calculate the opinion consistency values for speakers and political parties. In Section 4.4.1, we visualize actual and predicted opinion consistency values. Finally, in Section 4.4.2, we explore the impact of a model's capability to classify opinions on the accuracy of opinion consistency values.

4.1 Opinion Classification: First Experiment

After we labeled the first dataset, we examined the capability of various machine learning models to predict the speakers' opinions on lockdowns.

4.1.1 Classification

For the first run, we started with a simple deep learning network. The network's architecture consisted of a bag-of-words embedding layer with 64 dimensions, followed by a fully connected linear output layer with three output neurons. We initialized the weights randomly and started training on the test set. We used a stratified holdout approach to split the data 80-20. From the training set, we split off another 20% to be used for validation during training. We used a stochastic gradient descent optimizer with a batch size of 32 and a cross-entropy loss function. We trained for 20 epochs with no early stopping criteria. Furthermore, no pre-processing was applied.

We manually ran the first setup a few times and achieved classification accuracies between 64% and 80% on the test split. We did not expect the results to be representative of the true performance because we did not use weighted class labels for training, which should be done on imbalanced datasets. We suspected that the network learned to predict the

majority class, therefore statistically achieving good performance. A confusion matrix could have been used to verify that hypothesis. At the time, we decided to approach the problem differently by creating a training set with the same number of samples in each of the three classes.

In the second data splitting approach—with an even distribution of class labels in the training set—we achieved classification accuracies between 20% and 70%. We explained the high variance by the small sample size. Depending on the quality of the randomly chosen training samples, the results on the test set could be significantly better or significantly worse.

Initially, the minority class of our dataset was the positive opinion with only 50 samples. Therefore, the training set consisted of only 150 samples, with the approach of evenly distributing the classes. To verify if that is indeed a cause for the high variance in classification performance, we labeled the dataset a second time with a bias towards non-neutral opinions (referred to as category 7 in Section 3.4). This time, the minority class was still the positive opinion, but with 90 samples instead of 50. We split the data into an evenly distributed training set (90 samples of each class) and put the rest into the test set. With a classification accuracy between 10% and 50%, the results were not better.

At this point, we decided that due to the high variance in the results, we had to perform multiple runs of training evaluation on different splits of the data and average the results to get a better understanding of true performance. Another explanation for the high variance could be the small validation set, which we have used to adjust the learning rate during training dynamically. Since the validation set samples consisted of only 20% of randomly selected samples from the training set, the variance in classification accuracy will be high for such a small dataset.

Due to the overall dataset being small, we performed some runs without a validation set, i.e., we used the training set in place of a separate validation set. This approach increases the risk of overfitting, but it might be the better compromise on our dataset. Again, we used the same holdout approach as before and performed 50 runs of training and evaluation. The results showed us that most of the runs achieved a classification accuracy between 30% and 50% on the test set, with some outliers performing significantly better or worse. Of course, the resulting average accuracy of close to 40% was not satisfying, as a random guessing approach would not be significantly worse with an average accuracy of 33%. To test the impact of pre-processing, we performed another run after applying stop-word removal. The results showed practically no difference in performance.

A more complex model At that point, we had explored various data-splitting methods and one pre-processing method. Next, we wanted to test a more complex model architecture. It consisted of an embedding layer with 300 dimensions, followed by a bi-directional LSTM with two layers and a linear layer with three output neurons. For

the embedding layer we used pre-trained GloVe word embeddings[1]. We had to apply the pre-processing step of lower-casing all words because the pre-trained word vectors were all in lower case. Interestingly, with the data splitting method of even distribution, the results were worse than random guessing.

Next, we tried a different approach to counter the class imbalance. We used a stratified train-test split but used weighted training samples. When calculating the loss, we made the training samples of underrepresented classes more important and those of overrepresented classes less important. The results showed the superiority of this approach compared to the one we had used previously.

An attention model Since we had achieved promising results by using a recurrent network, we were interested in examining the capabilities of the attention mechanism in transformer architectures. For the subsequent runs, we used a BERT architecture with around 109M trainable parameters. It consists of an input layer that takes a sequence of word ids, followed by a pre-trained German BERT model, completed by a fully connected layer.

Due to the large size of this BERT model, it was not feasible to train it on a personal CPU. Instead, we performed all experiments related to BERT on tensor processing units (TPUs) at Google Colab servers, thus managing to reduce training times to reasonable durations. Since the Colab servers disconnect users from time to time, we had to implement a mechanism to store and resume the training progress after each epoch.

To test the capabilities of this pre-trained BERT model, we first used it on the 10kGNAD[2] dataset. We achieved an overall accuracy of 89%, which gave us confidence in the model's architecture.

After verifying the model on the 10kGNAD dataset, we moved on to the LOCKDOWN dataset. We used an 85-15 stratified split, with no validation set still. We applied the additional pre-processing techniques of lower-casing, stopword removal, and stemming. For now, we used the same pre-processing steps in all BERT runs. In the second experiment (Section 4.2), we also compared the impact of different pre-processing techniques. The model architecture quickly brought the machine's hardware capabilities to their limits. Therefore, we had to truncate the sentences to a maximum length of 128 tokens and use a batch size of 64 samples. In the case of the LOCKDOWN dataset, the truncation did not lose any data, as Figure 4.1 shows. We trained the network for a maximum of 20 epochs but used an early stopping policy if there was no improvement in the loss for more than three epochs. Usually, the network converged in epoch 10–12.

The classification report, showing the performance per class, revealed a weakness in the training method. The model predicted zero positive opinions. We explain that circumstance by the low amount of positive samples in the training set. After using a

[1] https://www.deepset.ai/german-word-embeddings, accessed: 2021-09-27
[2] https://github.com/tblock/10kGNAD, accessed: 2021-09-30

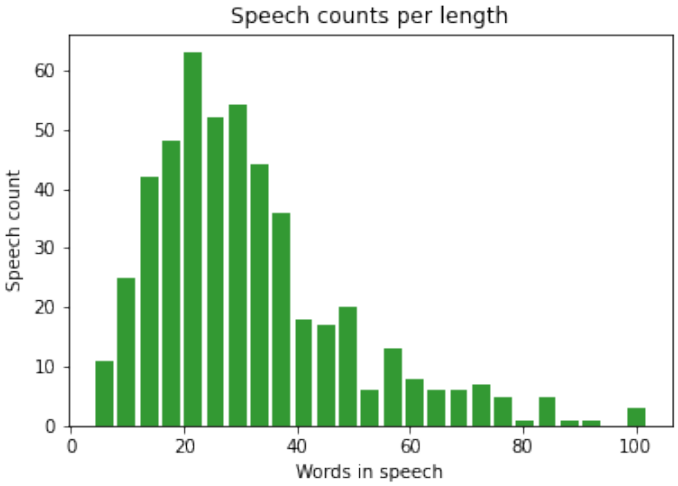


Figure 4.1: Speech lengths in the LOCKDOWN dataset

weighted loss function to give the positive samples more importance during training, the model also learned to predict positive opinions and achieved an impressive performance of 59% accuracy.

Out of curiosity, we also used a second data-splitting method, creating a test set consisting of precisely 20 samples from each class, for a total of 60 samples. We trained the model on the training set consisting of the remaining samples with a weighted loss function. In this case, the evaluation performance dropped to 49% accuracy on the test set. We conclude that it is still more difficult for the model to predict positive samples simply because there are fewer examples to train on, even though they have more impact due to the weighted loss function.

Statistical Models At this point, we had a good overview of the capabilities of deep learning models. Additionally, we wanted to test the capabilities of some statistical models.

We started with a multinomial Bayes (MNB) model, as described in Section 2.1.1. Since in MNB, we did not have to train a network with many parameters but count frequencies of words per class, fitting the model to the training data took considerably less time than for the deep learning models. Therefore, the experiments could be run on an average personal computer with acceptable time investment. As before, we weighted the samples proportionally to the class sizes to account for the class imbalance. We applied the same pre-processing methods as we did for the BERT runs and performed 1000 runs on two data-splitting methods. First, we utilized a random 85-15 stratified split and achieved an accuracy of 53% averaged over all runs. On the second split—by randomly selecting 20 samples from each class into the test set and the remaining samples into the training set

—the model achieved an averaged accuracy of 51%. Interestingly, the difference between results on the two splits is only 2% compared to the 9% observed for the BERT model.

Finally, we applied the BM25 document ranking algorithm (described in Section 2.1.1) to classify opinions. Based on a text query, the document ranking function calculates a relevance score for each document in a set of documents. To make it work for text classification, we used the speech to be classified as the query and used BM25 to calculate relevance scores for all speeches in the training set. Then, we used the samples with the n highest scores to determine the label of the query sample.

We applied the same pre-processing steps as in the BERT run. In each run, we randomly selected 20 samples from each class into the test set and used the remaining samples for calculating the document scores. We used two methods for determining the class label from the ordered list of samples of the lookup set of size N—one weighted by class frequency, the other not. Let $(s_1, l_1), \ldots, (s_N, l_N)$ with $s_i \geq s_j \; \forall i < j$ be the ordered list of tuples with the relevance score s_i describing how relevant the i–th document is in relation to the query, and l_i the class label of the i–th document. Then, we assign class c to the query by calculating:

$$Score_c = \sum_{i \leq n \wedge l_i = c} s_i \cdot w_c \tag{4.1}$$

$$c = argmax\{Score_1, Score_2, Score_3\} \tag{4.2}$$

In the non-weighted case we set $w_c = 1$ and in the weighted case we calculate $w_c = N/N_c$, with N_c being the number of samples belonging to class c. We experimented with different parameters of n, both with and without class-weighing. We attained the best results with $n = 7$ and class-weighing. The average accuracy over 500 runs was about 45%.

In the next section, we plot a table of the overall results and provide an interpretation.

4.1.2 Summary of Results

With the five models that we used to predict opinions on the lookup (training) set, we had a good overview of our expected performance. The results, averaged over all runs, are shown in Table 4.1. The accuracy values are slightly different from those of the previous section. The reason is that after we had performed the second experiment, we reran all algorithms on the first dataset to record also the macro-average F1-Scores, which we did not record initially.

It was expected that BERT outperforms the LSTM, which in turn outperforms the Embedding Bag, but it comes as a surprise that the MNB outperforms the BERT slightly in terms of mean F1, even though it is slightly behind in accuracy. The BM25 approach had the biggest gap between accuracy and F1-score, meaning it was influenced the strongest by the dataset imbalance.

The classification results on the LOCKDOWN set can be considered mediocre, and there are possible explanations.

Approach	Mean Acc	Mean F1	Std Dev	Min. F1.	Max. F1.	Runs
MNB	0.53	0.48	0.055	0.31	0.66	1000
BERT	0.56	0.47	0.088	0.23	0.66	100
LSTM	0.51	0.42	0.051	0.30	0.55	100
Embedding Bag	0.42	0.38	0.066	0.23	0.58	100
BM25	0.47	0.28	0.057	0.10	0.42	100

Table 4.1: Performance comparison of various machine learning approaches on the LOCKDOWN set, sorted by F1-Score.

1. The dataset size is small, which makes it difficult to train a generalized model.
2. Due to the class imbalance, there are even fewer samples to train from in the minority class, making it more challenging to achieve good macro average F1-Scores. Another reason could be that the domain is complex, making it more difficult for the model to learn.
3. The subjectivity of opinions is high, making it difficult to label the samples consistently, increasing the likelihood of introducing label noise, which will make it more difficult for the model.

In the next section, we describe how we applied the same algorithms on the more extensive MEASURES set to study the impact of dataset size on model performance.

4.2 Opinion Classification: Second Experiment

In the second experiment, we performed opinion classification on the MEASURES dataset (refer to Section 3.4 for the collection process), which is over ten times larger than the LOCKDOWN dataset.

4.2.1 An Improved Test Pipeline

Before we ran the classification algorithms on the second dataset, we decided to define an improved test pipeline that provides more detailed documentation of results. The goal was to unify the test conditions as much as possible across the algorithms to allow for a more meaningful comparison.

We decided to perform a random 85-15 train-test split with stratification by class for all algorithms. Where applicable, a stratified validation set should be selected by randomly choosing 15% of the training samples. We used the cross-entropy loss for models that use a loss function commonly used in multiclass classification tasks. We weighed the training samples proportional to their occurrence frequencies when calculating the loss, making less frequent classes more important. Furthermore, we selected the model that achieved the lowest loss on the validation set at the end of each training phase and evaluated it on the test set.

To allow us the easy change of different combinations of experiment parameters, we extracted the following parameters from the code as variables:

1. `RUNS`: How many times the model should be trained and evaluated. Not to be confused with the number of epochs for which a network is trained in each run. Default: 100
2. `TEST_SPLIT`: How many percent of the total data should be used for testing. The remainder is used for training. Default: 15%
3. `VALID_SPLIT`: How many percent of the training data should be used for validation during training. Default: 15%
4. `SHUFFLE`: If the data splits should be sampled randomly. Default: True
5. `STRATIFY`: If the data splits should be stratified by class. Default: True
6. `CLASS_WEIGHTS`: If the training samples should be weighed proportionally to their class' frequency. Default: True
7. `REMOVE_STOP_WORDS`: Whether the pre-processing step of stop word removal should be performed. Default: False
8. `STEMMING`: Whether the pre-processing step of stemming should be performed. Default: False

9. LOWERING: Whether the pre-processing step of lower-casing all words should be performed. Default: False

10. NO_PUNCTUATION: Whether the pre-processing step of removing punctuation marks should be performed. Default: False

11. N_BEST: How many of the best-matching samples should be considered for determining a class label. (only BM25) Default: 7

If not otherwise stated, we used the specified default values. Additionally, we implemented automatic storage of evaluation metrics after each run into a file that contains all of the above parameter values in its file name. That way, we could easily plot graphs of the results later and keep track of the experiment parameters.

4.2.2 Classification

After implementing the improved test pipeline changes, we began to run the models on the MEASURES dataset. The bag-of-words model achieved significantly better accuracies and F1-Scores than on the previous dataset. Figure 4.2 shows the performance metrics of 100 individual train-evaluation runs. As expected, due to the larger dataset size, we observe an overall reduction in the variance of results compared to the previous dataset. The red horizontal line indicates the performance of a random-guessing approach that would select each of the three classes with equal probability. Due to how the F1-Score is calculated, a random-guess approach would achieve only 32% and not 33%, as is the case with accuracy.

(a) Accuracies (b) F1-Scores

Figure 4.2: Results of the bag-of-words neural network on the MEASURES dataset

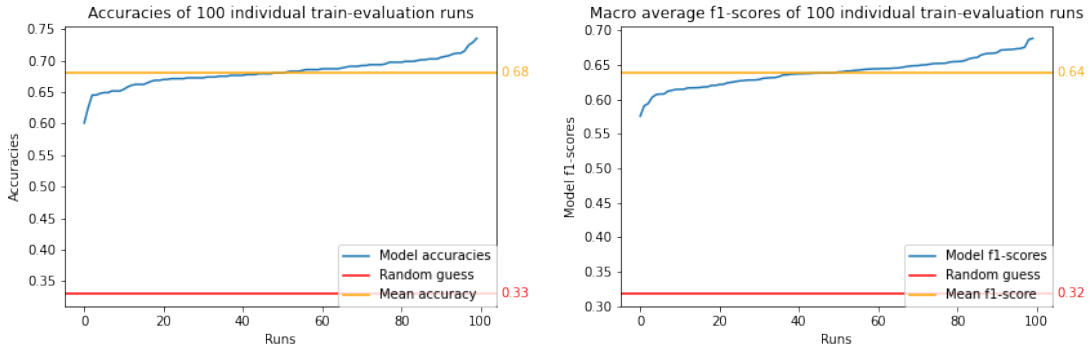
Table 4.2 shows the classification report for the bag-of-words (BOW) model on the MEASURES dataset. All values are averaged over the 100 runs. We observe a direct correlation between the ability to predict a class with the number of samples from that class (Column Support). As was the case in the LOCKDOWN set, the network was able

to predict more frequent classes better, even though we used a weighted loss function. Considering that opinion classification is a complex task, the overall classification accuracy of 62% was already a decent accomplishment.

	Precision	Recall	F1-Score	Support
-	0.55	0.57	0.56	196
o	0.39	0.53	0.45	155
+	0.79	0.67	0.73	416
Accuracy			0.62	767
Macro Avg.	0.58	0.59	0.58	767
Weighted Avg.	0.65	0.62	0.63	767

Table 4.2: Classification report: BOW on the MEASURES set

Next, we ran the second deep network—the LSTM. With the same architecture as in the first experiment and all of the test parameters set to default, we achieved an average accuracy of 68%; six percent better than the bag-of-words model. The classification performance on the test sets of 100 runs can be seen in Figure 4.3.



(a) Accuracies (b) F1-Scores

Figure 4.3: Results for the LSTM neural network on the MEASURES dataset

Table 4.3 shows the classification performance per class and the averaged scores over all classes. We observe that for neutral opinions, the recall is higher than the precision, which could result from giving neutral training examples more weight during training. It is the other way around for the majority class (positive opinions) since we weigh each training example less than samples of the other two classes. Moreover, with medium support, the values of precision and recall are well balanced for negative opinions. The same pattern can be observed in all other classification reports in this section (except for OpenAI, but there the sample size is considerably lower). We assume that we could achieve a better balance between precision and recall if we would increase the importance of the majority class and decrease the importance of the minority class slightly. We leave

that as a topic for future study.

	Precision	Recall	F1-Score	Support
-	0.63	0.63	0.63	196
o	0.47	0.57	0.51	155
+	0.82	0.75	0.78	416
Accuracy			0.68	767
Macro Avg.	0.64	0.65	0.64	767
Weighted Avg.	0.70	0.68	0.69	767

Table 4.3: Classification report: LSTM on the MEASURES set

Since the BERT architecture had considerably more trainable parameters than the other models, performance considerations played a more important role. The maximum speech lengths in the MEASURES dataset were longer than those in the LOCKDOWN dataset. This time, we had to trim some speeches to stay within the server's memory capacity. Figure 4.4 shows a plot of the speech lengths in the MEASURES set. The maximum length that the servers could handle was around 128 tokens. Fortunately, 98% of the speeches were shorter, requiring trimming on less than two percent.

In terms of pre-processing, we did not apply any. The idea was to adapt the complexity of input to the complexity of the model. Since BERT is a complex model, we decided to provide input with a high information density. For example, removing punctuation marks or lower-casing all words would reduce the complexity of the input, but it could lose vital information. A single comma can alter the meaning of a sentence, and also the capitalization of words can carry semantic information.

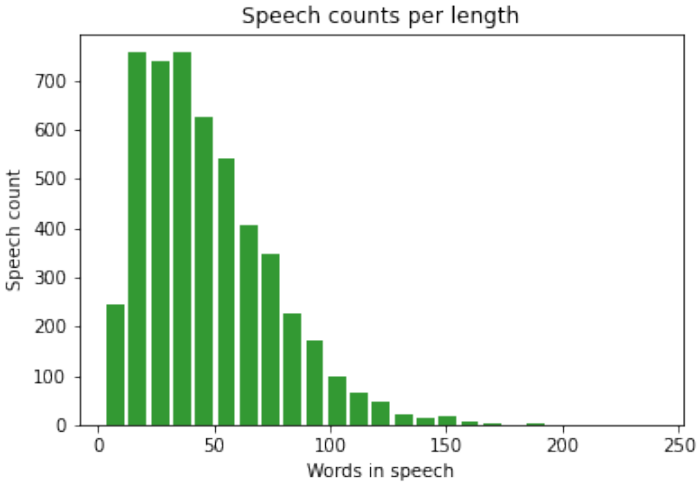
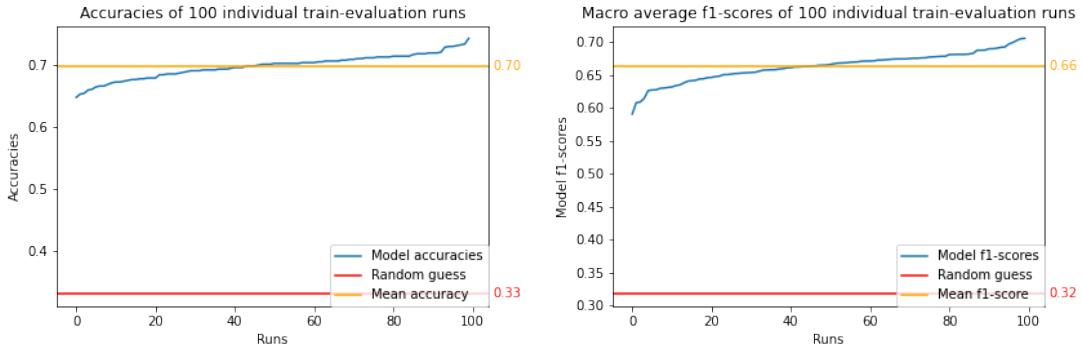


Figure 4.4: Speech lengths in the MEASURES dataset

Figure 4.5 shows the classification performances of the BERT model on the test sets. With 70% accuracy, it could achieve only a 2% improvement over the LSTM on the MEASURES set compared to a 6% improvement that it had achieved on the LOCKDOWN set. The results may suggest that the BERT can deal better with smaller datasets than the LSTM. One explanation could be the pre-training on a large corpus that the BERT received. As with other models, the accuracy is slightly above the macro-average F1-Score, because the performance on minority classes (neutral and negative opinions) is below the average.



(a) Accuracies (b) F1-Scores

Figure 4.5: Results for the BERT neural network on the MEASURES dataset

The detailed class report (Table 4.4) follows the same patterns as for the other models, but with overall higher numbers. We observe an increased precision in the majority class, an increased recall in the minority class, and a balance between precision and recall in the medium support class. As usual, the accuracy is slightly higher than the macro-average F1-Score.

	Precision	Recall	F1-Score	Support
-	0.64	0.65	0.64	195
o	0.53	0.63	0.57	155
+	0.82	0.75	0.78	416
Accuracy			0.70	766
Macro Avg.	0.66	0.67	0.66	766
Weighted Avg.	0.72	0.70	0.70	766

Table 4.4: Classification report: BERT on the MEASURES set

Next, we applied the BM25 for classification on the MEASURES dataset. We initially applied all four pre-processing techniques: removing stop words, stemming, lower-casing, and removing punctuation. We achieved slightly better results by limiting the choice to removing stop words and punctuation, so we went with that for the BM25 runs.

To determine class labels with BM25, we used the same method as in the previous exper-

iment. Because it took a considerable amount of time to run BM25 on the MEASURES set, it was not feasible to perform 100 runs for all combinations of the parameters `N_BEST` and `CLASS_WEIGHTS`. Therefore, we performed a preliminary study with ten runs each on the impact of some combinations of the parameters on classification performance. Figure 4.6 shows, that the best results were achieved with $n = 3$ and no class weights. Looking only at the results without weights, the best accuracy values (blue bars) tie between $n = 3$ and $n = 20$, but when considering the F1-Scores (green bars), the clear winner is $n = 3$.

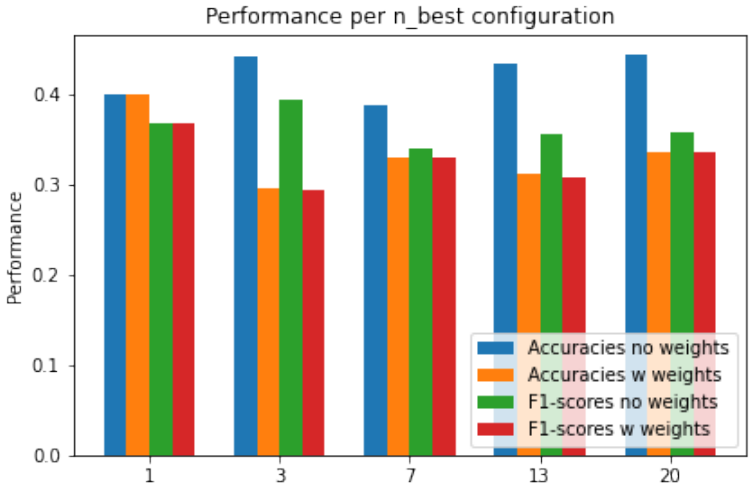
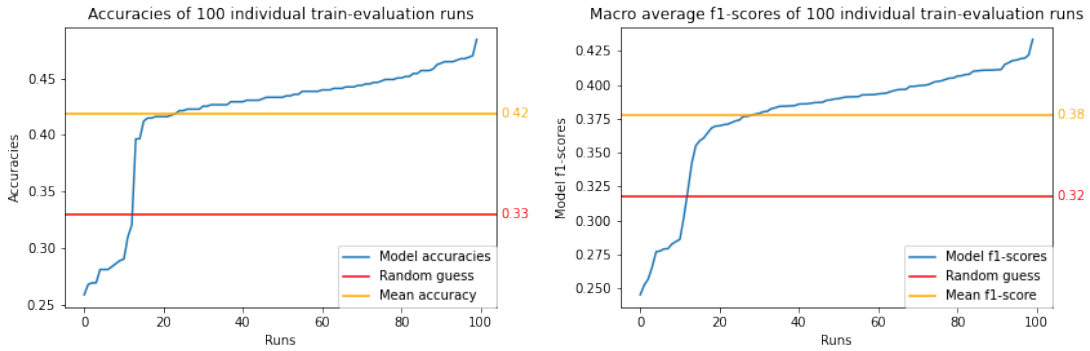


Figure 4.6: N-Best comparison for the BM25 model

After determining the best combination of `N_BEST` and `CLASS_WEIGHTS`, we performed 100 train-evaluation runs on the dataset. For the first time, the graphs of classification performance (Figure 4.7) follow a different pattern. We observe most results between 36% and 42% macro-average F1-Score, but around 15% being significantly worse. We explain that result by expecting that there are some especially important samples in the dataset. If those samples end up in the test set and not in the training set (used for lookup), performance drops significantly.

By looking at Table 4.5 we observe that the F1-Scores for negative and neutral opinions are below 32%, the threshold of a random-guessing approach. Additionally, the metrics are the lowest of all models. The results indicate that the way we used the BM25 algorithm is not well suited for opinion classification.

Next, we ran the Multinomial Bayes (MNB) with different combinations of pre-processing and found that they had no significant impact on the outcome. Finally, we performed 100 train-evaluation runs on the MEASURES set, with no prior pre-processing. The results, shown in Figure 4.8, are significant because MNB was only slightly worse than BERT, was on par with the LSTM, and better than the simple bag-of-words neural network. We were impressed because an MNB model, due to its simplicity, can be trained in a



(a) Accuracies (b) F1-Scores

Figure 4.7: Results for the BM25 approach on the MEASURES dataset

	Precision	Recall	F1-Score	Support
-	0.31	0.31	0.31	195
o	0.25	0.38	0.30	154
+	0.60	0.48	0.53	415
Accuracy			0.42	764
Macro Avg.	0.39	0.39	0.38	764
Weighted Avg.	0.46	0.42	0.42	764

Table 4.5: Classification report: BM25 on the MEASURES set

fraction of the time necessary to train an LSTM or BERT model. This result indicates that MNB classification could be a viable alternative when training a complex neural network on a large dataset would be too costly.

(a) Accuracies (b) F1-Scores

Figure 4.8: Results for the Multinomial Bayes approach on the MEASURES dataset

The classification report (Table 4.6) reveals a gap of 25% between the class-specific F1-Scores of the minority and majority class. This value is in the medium range compared to the other models, with the highest (28%) found in the BOW model and the lowest (21%) observed in the BERT model. Additionally, the MNB achieved a slightly higher precision on positive opinions, a slightly higher recall on neutral opinions, and a slight improvement in F1-Score on negative opinions, compared to the BERT model. All other metrics were equal or slightly lower than those of BERT.

	Precision	Recall	F1-Score	Support
-	0.69	0.62	0.65	196
o	0.42	0.66	0.52	154
+	0.84	0.71	0.77	416
Accuracy			0.67	766
Macro Avg.	0.65	0.66	0.65	766
Weighted Avg.	0.72	0.67	0.69	766

Table 4.6: Classification report: MNB on the MEASURES set

Additionally, on the MEASURES set, we examined the capabilities of a sixth model—the Davinci model from Open AI, which is based on GPT-3. It can be accessed through their web API[3]. Due to the API nature of the model, the test process was different than it was for the other models. We did not have to perform model training since it Davinci is an already trained model. Additionally, the API uses a single endpoint for all NLP tasks. The API detects which task should be performed based on the input format. Furthermore, it requires examples, either provided together with the classification sample or by uploading a file. We chose the second approach since we provided the entire train split, which is many examples. A single query can make use of at most 200 example records. If there are more examples in the file, then the algorithm selects the most relevant ones.

The API works with a credit system. Depending on the length of the query, the used model (Davinci is the most expensive), and the number of provided examples, the API charges a different amount. To our surprise, we managed to label only 262 samples before we had spent our free budget of $18. Since we could only classify 262 out of 767 samples, the results are likely not to be representative.

The results of the 262 samples we managed to classify are shown in Table 4.7. With an F1-Score of 49% and an accuracy of 50%, the results are worse than expected. Our first explanation for the bad results is that the algorithm derives meaning from the class labels. Since we named them "0,1,2", we would expect to perform better by naming them "positive, negative, neutral." Since we used up the budget so quickly, we could not experiment with different input formats. Therefore, the results are likely not to be representative of the potential performance we could have achieved.

[3] https://beta.openai.com/, accessed: 2021-10-01

	Precision	Recall	F1-Score	Support
-	0.51	0.82	0.63	78
o	0.30	0.52	0.38	50
+	0.84	0.31	0.46	134
Accuracy			0.50	262
Macro Avg.	0.55	0.55	0.49	262
Weighted Avg.	0.64	0.50	0.49	262

Table 4.7: Classification report: Open AI Davinci (GPT-3) on the MEASURES set

4.2.3 Summary of Results

For the second experiment, we had access to a significantly larger dataset (~10x). It contains speeches that talk about measures for containing the spreading of the Coronavirus. We ran the same algorithms as on the LOCKDOWN set, with the addition of a GPT-3 model over the Open AI API. Due to the increased dataset size, we could comfortably split a validation set from the training set and use it for model selection. Additionally, we experimented with different combinations of pre-processing steps but found that they had no significant impact on performance.

Approach	Mean Acc	Mean F1	Std Dev	Min. F1.	Max. F1.	Runs
BERT	0.70	0.66	0.022	0.59	0.71	100
MNB	0.67	0.65	0.017	0.60	0.70	1000
LSTM	0.68	0.64	0.021	0.58	0.69	100
Embedding Bag	0.62	0.58	0.022	0.52	0.63	100
Open AI	0.50	0.49	-	0.49	0.49	<1
BM25	0.42	0.38	0.042	0.25	0.43	100

Table 4.8: Performance comparison of various machine learning approaches on the MEASURES set. The standard deviation refers to the F1-Score.

Table 4.8 displays the averaged results of algorithms on the MEASURES set, ordered by F1-score. Overall, the results are significantly better than on the LOCKDOWN set, so dataset size seems to have played a significant role. Like on the LOCKDOWN set, the deep neural network models are ranked in order of sophistication, except for Open AI. Open AI is displayed with less than one run because it was only evaluated on a subset of the test data once. Therefore, the Open AI results are likely not representative of its actual capabilities. The MNB approach again performed very well and is only one percent behind BERT, but also the LSTM performs almost equally well. The BM25 approach has a significantly better F1-score than before, and the gap to the accuracy is not as large as it was on the LOCKDOWN set, meaning it dealt better with the class imbalance this time.

4.3 Visualizations of Opinion Data

In the previous two sections, we explored the capabilities of different machine learning models to predict opinions. In this section, we want to work on the partial fulfillment of Requirement R5 and visualize the opinion data produced by the best-performing algorithm. In order to examine the usefulness of such visualizations, we compare the actual opinion data with the predicted opinion data.

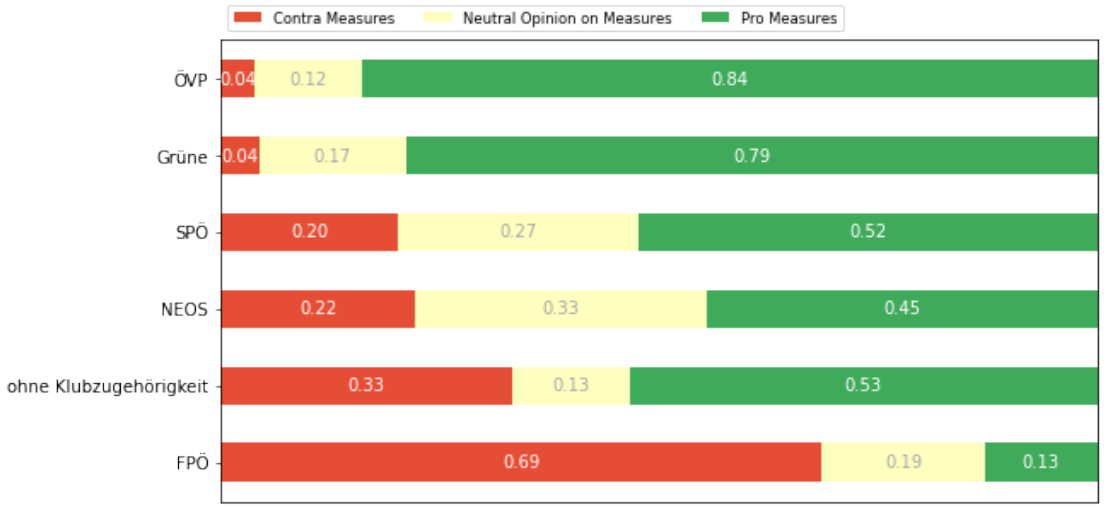
	Precision	Recall	F1-Score	Support
-	0.63	0.67	0.65	1301
o	0.52	0.62	0.57	1032
+	0.83	0.75	0.78	2773
Accuracy			0.70	5106
Macro Avg.	0.66	0.68	0.67	5106
Weighted Avg.	0.72	0.70	0.71	5106

Figure 4.9: Classification report of the labels, predicted with BERT, used in the opinion data visualizations and opinion consistency comparisons

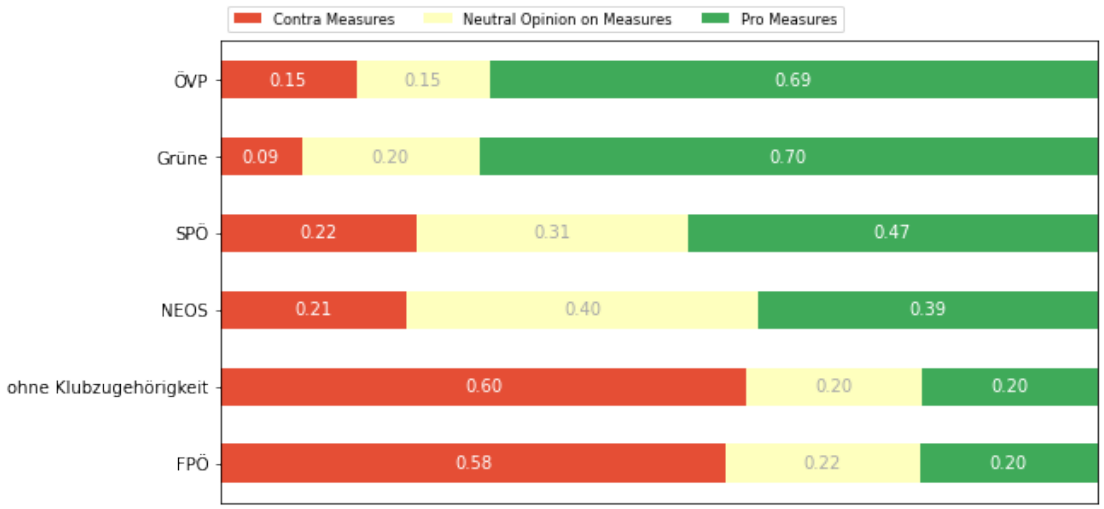
We used the BERT model to generate the predicted opinion data for the entire MEASURES dataset. To this end, we used a stratified k-fold technique by splitting the dataset into ten equally large parts. We predicted the labels of each part by the BERT model trained on the other nine parts. All predictions were aggregated to cover the entire set. These predictions were used to plot the predicted opinion distributions per party and per speaker and also, in the next section, to plot the predicted opinion consistency over time. Table 4.9 shows the classification report of the overall results on the MEASURES dataset. The values are very close to those achieved in the previous section.

First, we plotted the opinion data per political party. Figure 4.10a shows the actual values and Figure 4.10b shows the predicted values. There are five different parties and the category "ohne Klubzugehörigkeit," which means without party affiliation. The difference between the actual and the predicted values is small enough that the predicted graph provides a representative picture of the actual graph, with the caveat that a certain sample size needs to be exceeded. As we can observe, the difference between predicted values and actual values is greatest for "ohne Klubzugehörigkeit" since the sample size is small for that category. For the other parties, the differences are within an acceptable level. We have to note that the predicted graph is based on values produced by an algorithm that has only 70% accuracy, and with higher accuracy values, the predicted graph would become even more accurate.

Multiple interpretations of the data are possible. In our view, we observe a pro-measures attitude in the governing parties Grüne and ÖVP since they are the ones implementing the measures. The strongest opposition comes from the FPÖ. The NEOS have the highest number of neutral opinions since they appear to follow a problem-solving approach, i.e., objectively discussing facts. The SPÖ is somewhere in the middle, indicating that they



(a) Actual opinion distribution per party

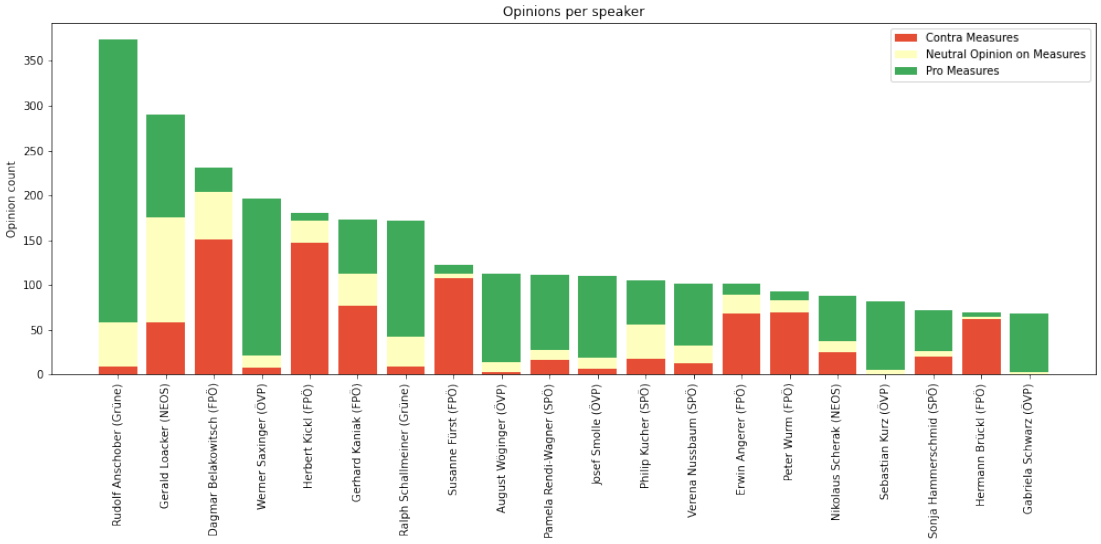


(b) Predicted opinion distribution per party

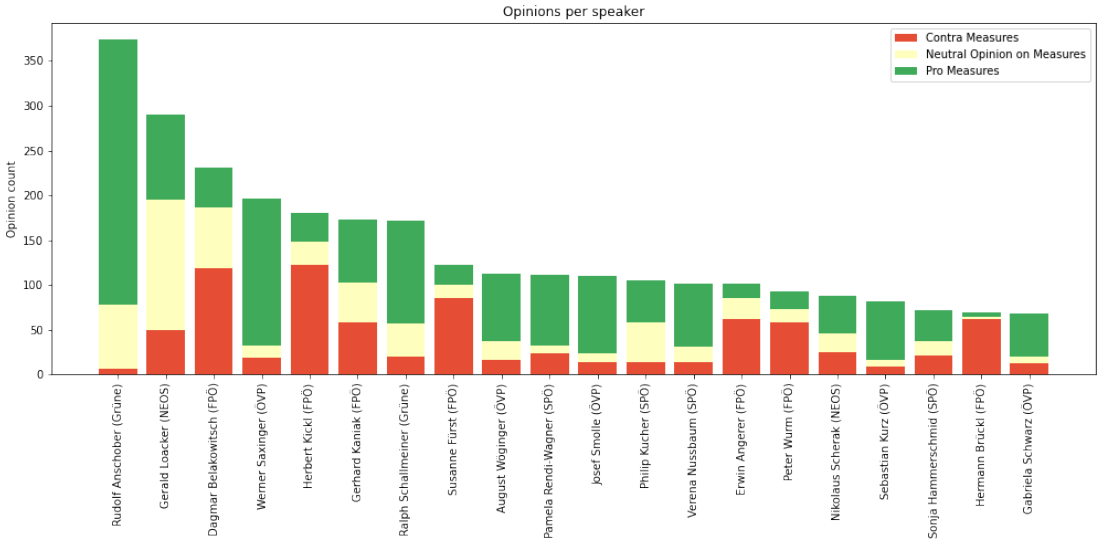
Figure 4.10: Opinions on MEASURES per party

want to "leave multiple doors open" by appealing to a broad audience. They lean towards a pro-measures opinion but do not take extreme positions.

Then, we plotted the opinion data for the twenty speakers that expressed the most opinions on the COVID-19 measures. As we did for the parties, we compare the representative quality of the predicted graph (Figure 4.11b), to the actual graph (Figure 4.11a). In our opinion, the predicted graph provides an adequate representation of the actual graph.



(a) Actual opinions of the top 20 speakers



(b) Predicted opinions of the top 20 speakers

Figure 4.11: Opinions of the top 20 speakers on MEASURES

The general sentiment towards the measures of all the speakers is preserved. For example, it does not happen that a speaker that is clearly for the measures in the actual graph is suddenly against the measures in the predicted graph. Generally, the sentiments of individual speakers are aligned with those of their affiliated parties, with a few exceptions, e.g., Gerhard Kaniak from the FPÖ.

In conclusion, the visualizations of opinions of parties and speakers, based on a prediction

accuracy of 70%, are considered sufficiently representative to be used for reading the overall sentiment towards a topic.

4.4 Opinion Consistency

In the previous section, we examined the sentiment of speakers and political parties on a specific topic. In this section, we want to visualize the consistency of speakers' and political parties' opinions over time.

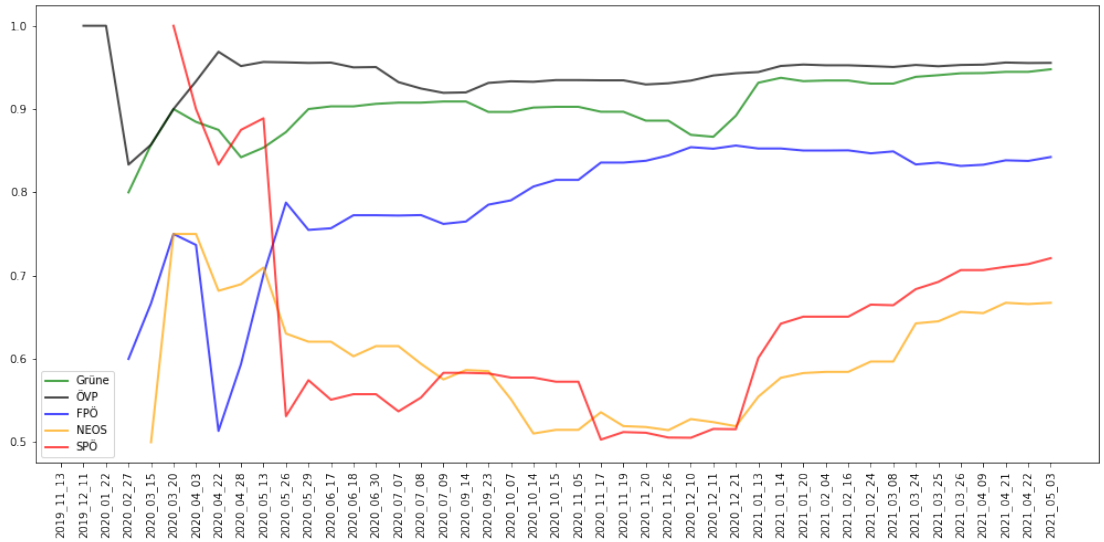
4.4.1 Visualizing Opinion Consistency

To calculate the opinion consistency values, we used the opinion data from the MEASURES dataset, in addition to the labels we have predicted with the BERT model (refer to Table 4.9). As was the case for the previous section, the actual opinion data is collected from the MEASURES dataset, and the predicted opinion data is constructed from the predicted labels. We construct the opinion tuples (g, s, h, t), by setting g to the topic *MEASURES*, h to the speaker and t to the timestamp. The speaker and timestamp are taken from the MEASURES dataset. For the actual opinion tuples, we set s to the actual label (found in the dataset), and for the predicted ones, we set s to the predicted label (provided by the BERT model). In this manner, we construct one actual and one predicted opinion tuple per record in the MEASURES dataset.

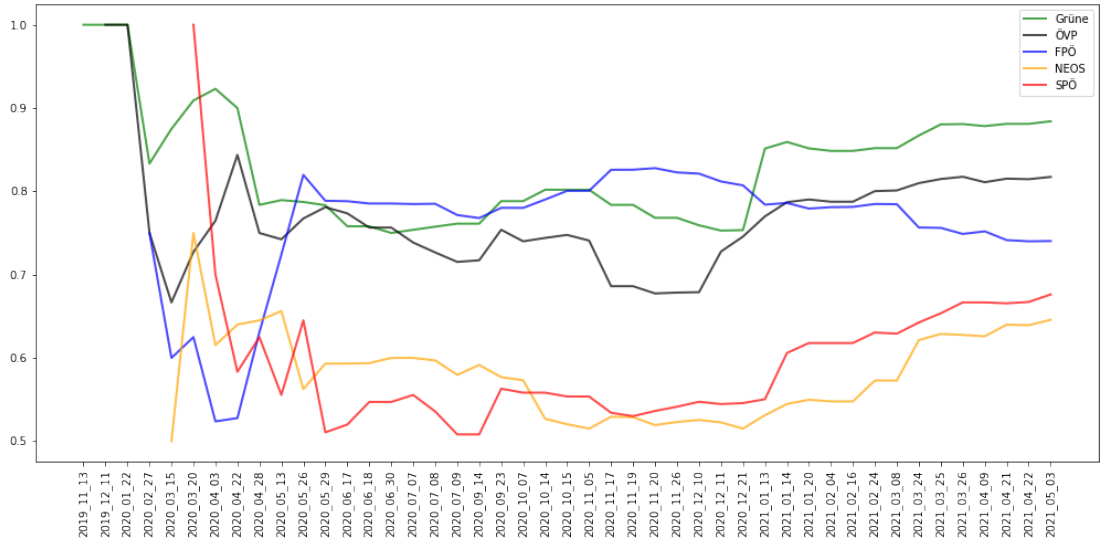
Based on the opinion data, we calculated the opinion consistency scores $OpCons_1(G, H, t)$ and $OpCons_2(G, H, t)$—as defined in Section 3.2—for every speaker and every party. In the set of opinion targets G, we included the topics of the MEASURES set. To calculate the scores of a speaker, we set the set of opinion holders H to include only the speaker. In order to calculate the aggregated score of a party, we set H to include all speakers of that party. We calculated all scores for each day by moving the timestamp t in 24-hour increments.

First, we plot the actual and predicted opinion consistencies for the parties over time. The actual graph (Figure 4.12a) is reasonably helpful. We can see which parties are more consistent, which ones express a more diversified opinion, and how the values change over time. We observe a strong fluctuation of values in the early period, as the consistency value gets more stable with increased sample size. A rolling window could also be interesting to get a more time-local view of the change in opinion consistency.

On the other hand, the predicted graph (4.12b) is of limited usefulness. Parties with a lower consistency score (SPÖ and NEOS) get approximated better, but those with a higher score, especially the ÖVP, are far off the actual values. In the middle periods, the predicted FPÖ value is higher than those of Grüne and ÖVP, which is misleading considering the actual values. Towards the end, the predicted ÖVP value is significantly below the actual value, which also delivers a wrong impression. For some parties, the predicted consistencies are very accurate, but there is a big difference for others. It seems there is a high element of chance involved, based on a model with 70% accuracy.



(a) Actual opinion consistency per party



(b) Predicted opinion consistency per party

Figure 4.12: Opinion consistency over time per party, based on $OpCons_2(G, H, t)$

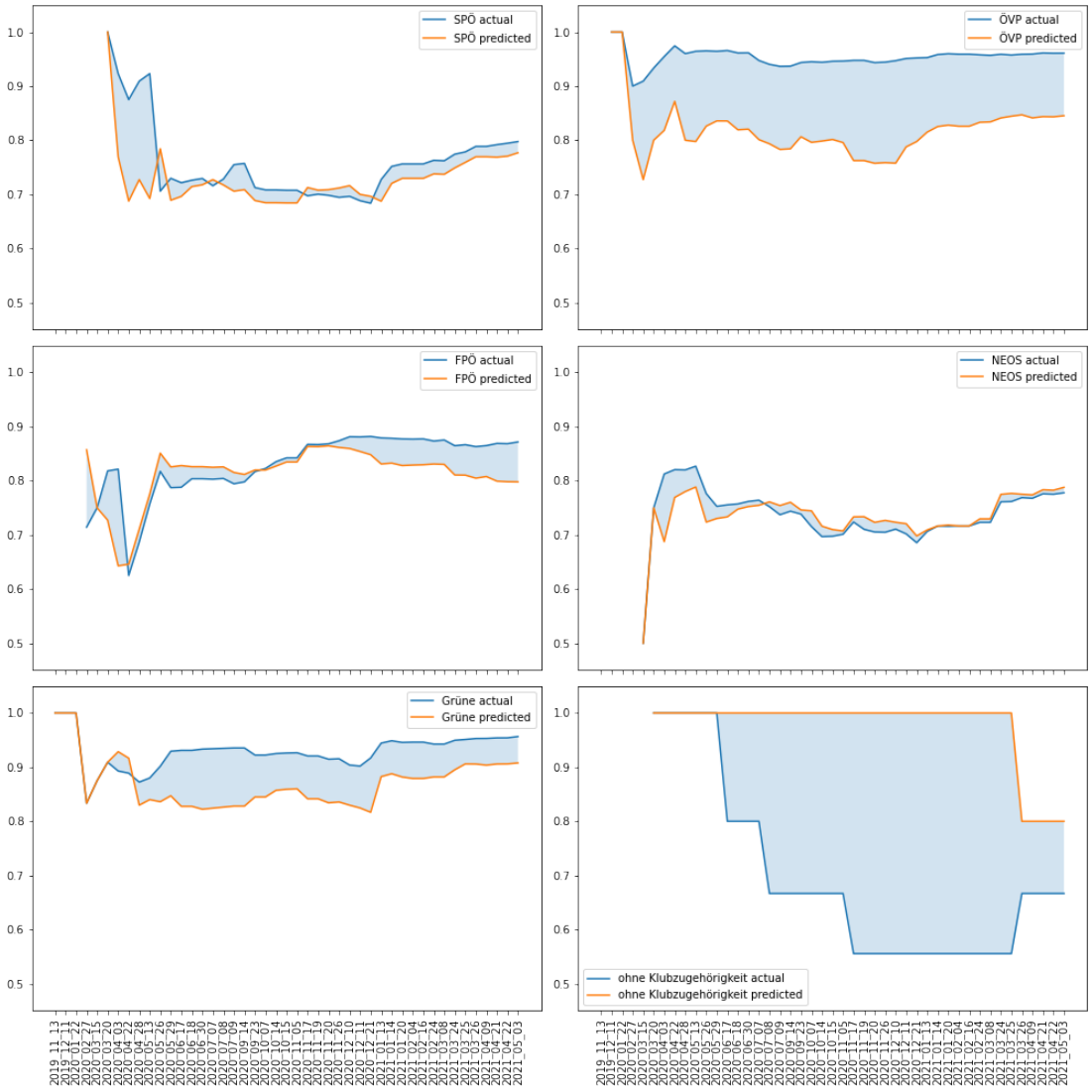
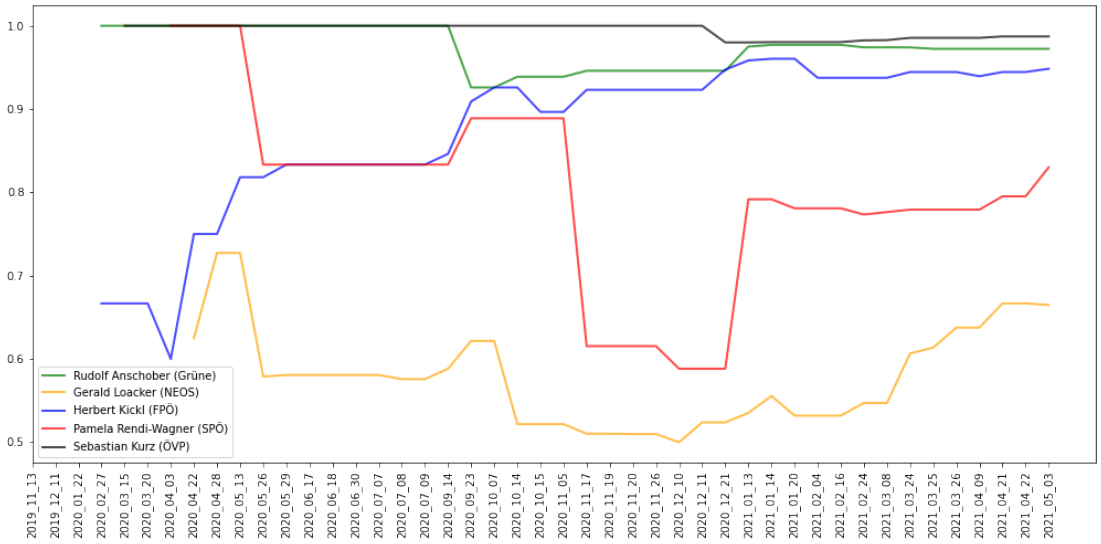
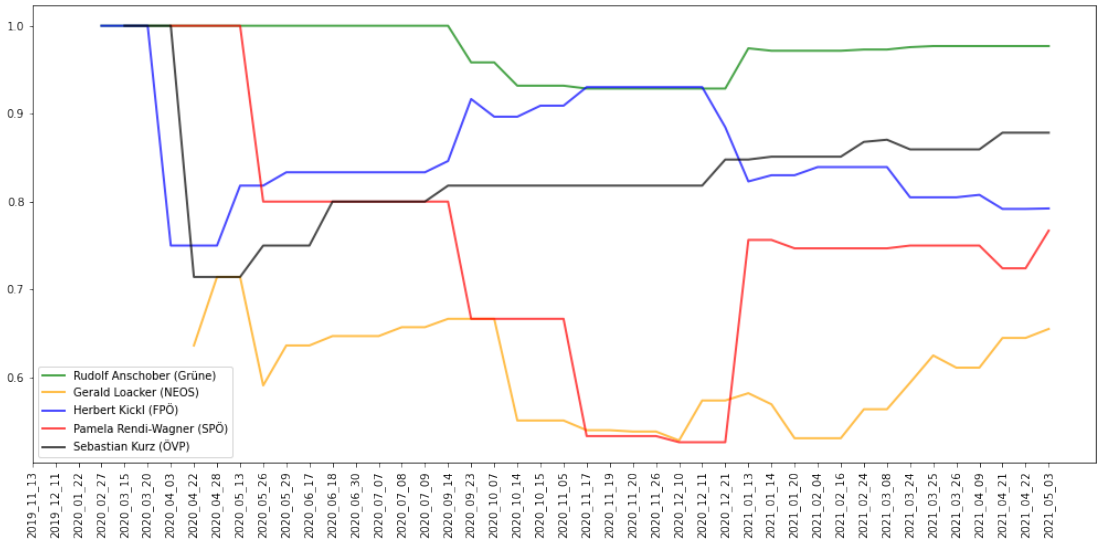


Figure 4.13: The predicted vs. actual opinion consistencies per party, including neutral opinions

Figure 4.13 shows the prediction errors of the opinion consistencies per party, based on the values of $OpCons_1(G, H, t)$. We observe the largest error for values of "ohne Klubzugehörigkeit" since the sample size is very small for this category, which also becomes apparent in the big jumps the graph performs. Additionally, we observe larger errors in the parties ÖVP and Grüne, which could be the result of the fact that especially high or low values are harder to predict than those in the medium range (as will be proved in Section 4.4.2). The predictions of the other parties are relatively accurate. Since the graphs for $OpCons_2(G, H, t)$ look similar, but with overall lower values and overall higher prediction error, we do not show them here.



(a) Actual opinion consistencies of one selected speaker from each party



(b) Predicted opinion consistencies of one selected speaker from each party

Figure 4.14: Opinion consistency over time per speaker, based on $OpCons_2(G, H, t)$

In Figure 4.14, we look at the opinion consistencies of five representative speakers, one from each party. The graph drawn from actual values (Figure 4.14a) is of high interest. It provides a diverse set of scenarios. We can observe that some speakers stay extremely consistent over the observed period, e.g., Sebastian Kurz and Rudolf Anschober, and some stay rather inconsistent (Gerald Loacker). We can identify when a speaker started inconsistently and became more consistent over time (Herbert Kickl), and we can also identify the opposite (Pamela Rendi-Wagner).

The predicted graph (Figure 4.14b) provides a reasonable approximation for speakers of lower consistency but struggles to capture the ones with high consistency. For example, we can observe a divergence of almost 20% between Herbert Kickl's and Rudolf Anschober's consistency values at the end of the predicted graph, whereas the difference is less than five percent in the actual graph.

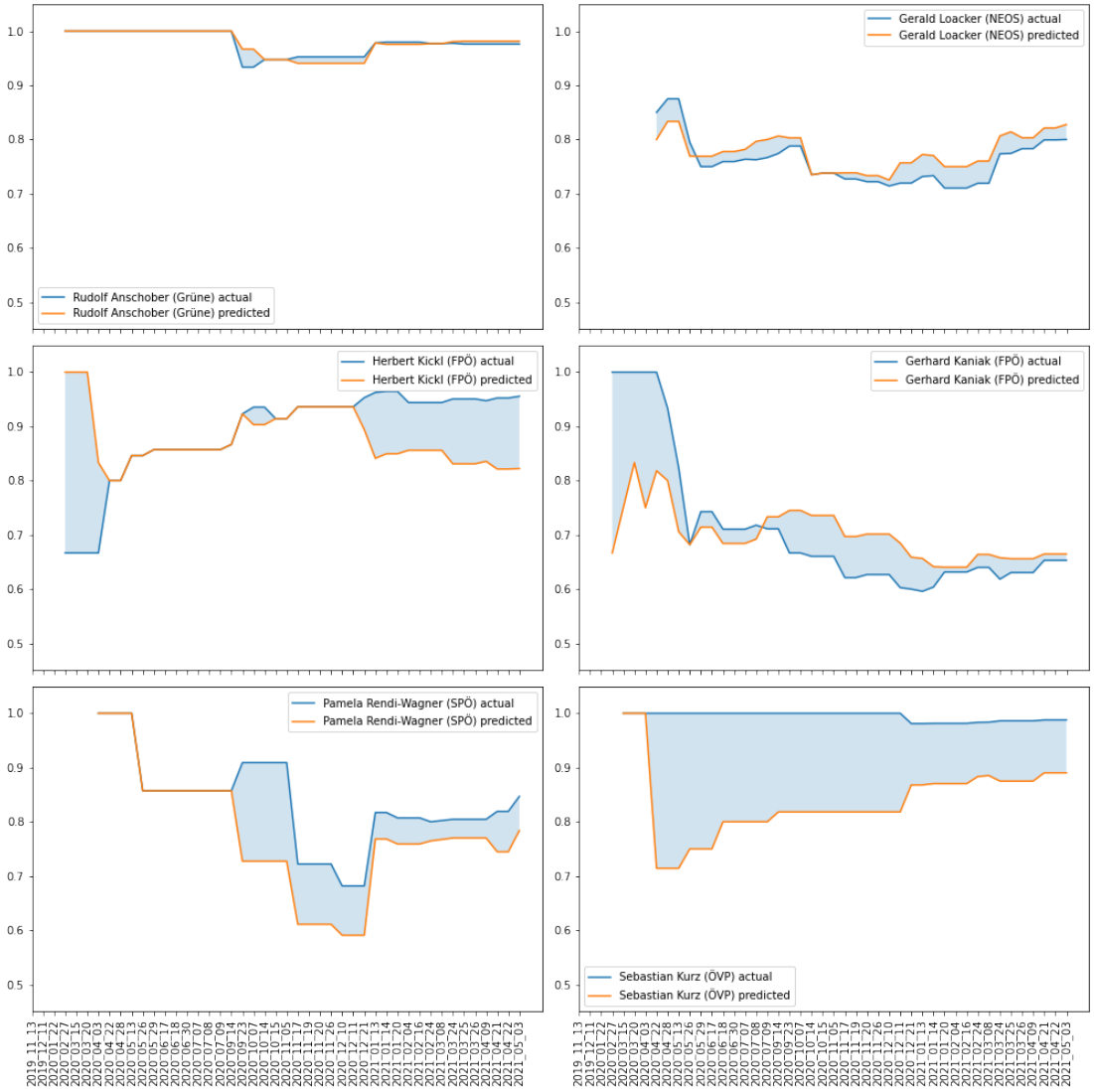


Figure 4.15: The predicted vs. actual opinion consistencies for selected speakers, including neutral opinions

In Figure 4.15, we visualize the classification error between the predicted and the actual opinion consistency values per speaker, based on $OpCons_1(G, H, t)$. Contrary to expectations, we can observe a high opinion consistency value that is accurately predicted

(Rudolf Anschober). An explanation is that the sample size for individual speakers is smaller than for parties, leading to an amplified impact of the element of chance. With Sebastian Kurz, we observe a high consistency value, for which the predictions struggle, as is expected. When looking at Herbert Kickl, we also observe the factor of chance, having close to perfect predictions in the middle part but strongly diverging toward the end. Overall, the opinion consistencies of speakers are more difficult to predict than those of parties due to the lower amount of available data.

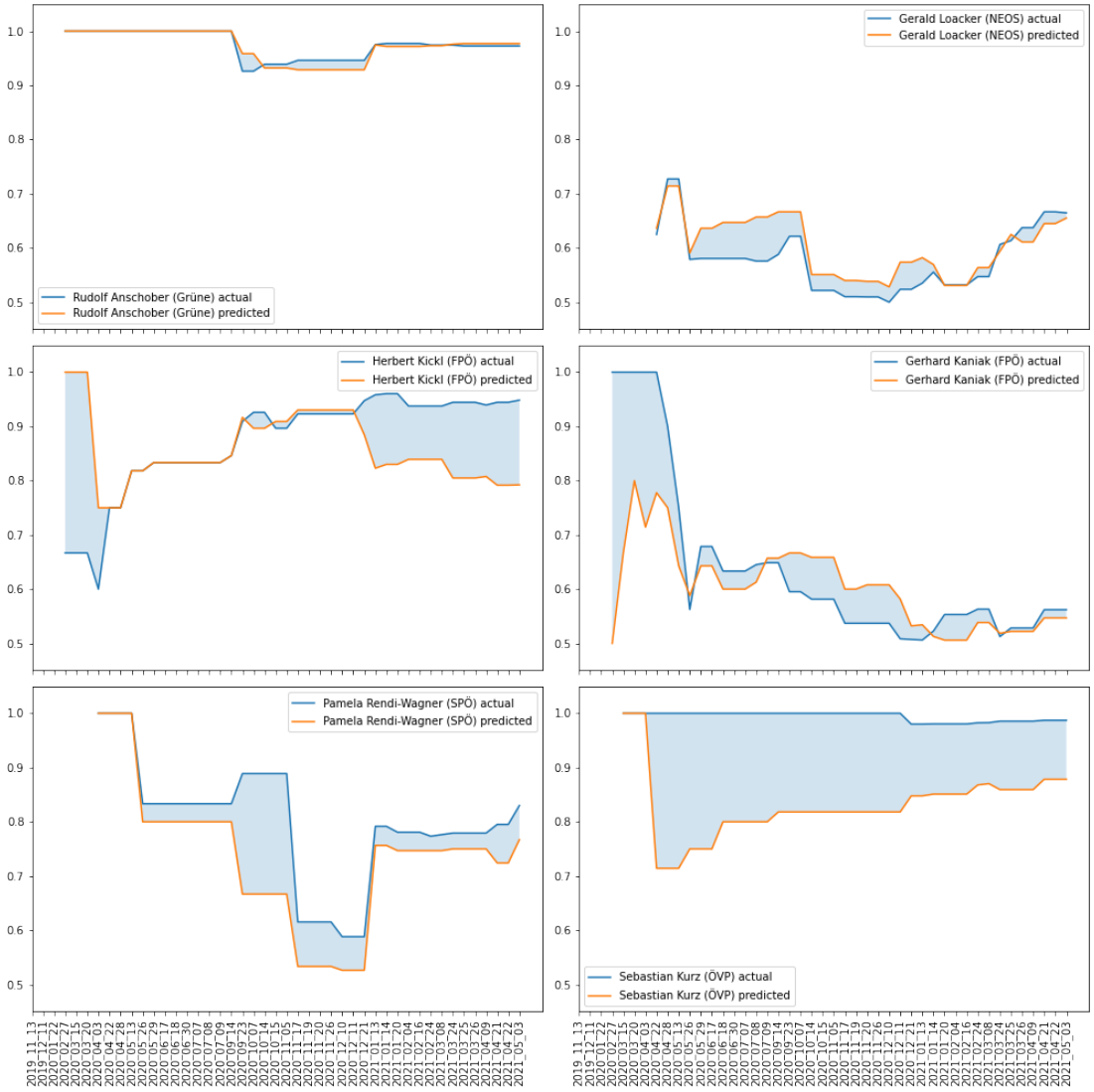


Figure 4.16: The predicted vs. actual opinion consistencies for selected speakers, excluding neutral opinions

In Figure 4.16 we also show the error, but this time excluding neutral opinions. Following

the expectations, the values are overall lower, especially for speakers expressing more neutral opinions (e.g., Gerald Loacker). Contrary to expectations, the classification errors are not significantly higher, which we also explain by the higher element of chance due to the smaller sample size.

Overall, we suggest using $OpCons_1(G, H, t)$ if the goal is to produce more accurate predictions. If a high accuracy model is available, we suggest using $OpCons_2(G, H, t)$ since it will better visualize contradicting opinions.

4.4.2 Impact of Model Performance on Opinion Consistency

In the previous section, we have analyzed the impact of model performance empirically. In this section, we aim to get a general understanding of the impact of model performance on the accuracy of opinion consistency. To this end, we want to determine the minimum required model accuracies for predicting opinion consistency within a certain margin of error. This effort is in fulfillment of Requirement R6.

We want to determine the required minimum accuracies for predicting the opinion consistency within some select confidence intervals for different margins of error and numbers of samples. We chose a 95% confidence interval, with seven margins of error between 0.5% and 10%, and eight sample sizes between ten and 3000.

The margin of error extends in both directions around the actual opinion consistency value. For example, a 5% margin around an actual opinion consistency of 70% would mean the acceptable predictions have to fall within 65% to 75% of predicted opinion consistency.

In addition to error margin and sample size, the required minimum accuracies also change based on the ratios between opinion classes. It is easiest to predict the opinion consistency resulting from the ratios 1:1:1 between negative, neutral, and positive opinions. That is because a uniform distribution of opinions will get approximated by a random guess, for which our model would need no accuracy. The opinion consistency gets more difficult to predict accurately, the more it deviates from the value of 0.67—the value resulting from a uniform opinion distribution. The most challenging opinion consistency value to predict is a value of 1.0—the value resulting from a 1:0:0 or 0:0:1 distribution of opinions. The lowest possible opinion consistency of 0.5 (1:0:1) is easier to predict than 1.0 but harder than 0.67.

To determine the minimum accuracies, we ran a simulation. We created a sequence of actual opinion labels and then used the Algorithm 4.1 to create the predicted labels with the help of a virtual machine learning model with accuracy *acc*. In the algorithm, the virtual machine learning model iterates through the list of actual labels and has a chance of *acc* to output the correct label and otherwise randomly outputs one of the other two labels. The symbol $\langle\rangle$ denotes the initialization of an empty list. The operator :: denotes the concatenation of two lists. The symbol $\oplus$ denotes the exclusive OR operation. The function $random()$ generates a random real number in the interval $[0, 1)$. If a list is given

to the function *random*(), it returns a random element from the list. The expression $\{0, 1, 2\}$ denotes the set containing a negative, neutral, and positive opinion label.

Algorithm 4.1: Simulate predictions of virtual machine learning model

Data: A sequence of actual opinion labels: *actual*. An assumed model accuracy: *acc*
Result: A sequence of predicted opinion labels: *ops_pred*

```
1 ops_pred ← ⟨⟩
2 for op_act ∈ actual do
3 |   if random() < acc then
4 |   |   op_pred ← op_act
5 |   else
6 |   |   op_pred ← random({0, 1, 2} ⊕ {op_act})
7 |   end
8 |   ops_pred ← ops_pred :: ⟨op_pred⟩
9 end
```

For each combination of error margin and sample size, we predicted the actual opinions 1000 times with each accuracy between 0.0 and 1.0 in increments of 0.01. For each of the 1000 prediction sequences, the opinion consistency value was calculated. For each combination of error margin and sample size, the lowest accuracy was chosen, for which 95% of the predicted opinion consistencies still lie inside the margin of error.

We ran the simulations on the opinions of four imaginary individuals. The first one expresses opinions in a uniform distribution (Table 4.17), the second one is absolutely consistent so he only states negative opinions (Table 4.18), the third one is as inconsistent as possible, so his opinions go back and forth between positive and negative (Table 4.19) and the last one is somewhere in-between. He always expresses three negative opinions followed by one neutral and one positive opinion (Table 4.20).

From the tables 4.17–4.20 we can read the minimum accuracies that are required to predict opinion consistency within the desired precision. For example, if we want to predict the opinion consistency of a perfectly consistent individual within a maximum error margin of 10% within a 95% confidence interval (meaning we want to be right 19 out of 20 cases), then we read from Table 4.18 that we need a minimum model accuracy of 89% after 100 samples and 85% after 500 samples.

By looking at the tables, we can make various observations. For low sample sizes and low error margins, almost perfect model accuracy is required. The predictions become more feasible with increased sample sizes or if higher error margins are tolerated. The minimum required accuracies depend on the ratio of the underlying opinions used to calculate *opinion consistency*. As previously mentioned, a ratio of 1:1:1 is easiest to predict, 1:0:0 is the most difficult, and 1:0:1 is in-between. If all ratios should be predicted within the desired error margin, then the minimum model accuracy must be taken from the 1:0:0 table; otherwise, a value between this table and the 1:1:1 table can be assumed.

Error \ Samples	10	30	50	100	300	500	1500	3000
0.5%	0.99	1.00	1.00	1.00	1.00	1.00	0.99	0.97
1.5%	0.99	1.00	1.00	0.99	0.98	0.95	0.86	0.67
2.5%	0.99	1.00	0.98	0.97	0.93	0.85	0.37	0.00
3.5%	0.99	0.98	0.98	0.95	0.83	0.68	0.00	0.00
5%	0.99	0.98	0.94	0.85	0.60	0.00	0.00	0.00
7.5%	0.99	0.92	0.88	0.67	0.00	0.00	0.00	0.00
10%	0.99	0.81	0.77	0.00	0.00	0.00	0.00	0.00

Figure 4.17: Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 1:1:1 (opinion consistency 0.67)

Error \ Samples	10	30	50	100	300	500	1500	3000
0.5%	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.5%	1.00	1.00	1.00	1.00	0.99	0.99	0.98	0.98
2.5%	1.00	1.00	0.99	0.99	0.98	0.98	0.97	0.96
3.5%	1.00	0.98	0.99	0.98	0.96	0.96	0.95	0.95
5%	1.00	0.98	0.97	0.96	0.94	0.93	0.92	0.92
7.5%	1.00	0.95	0.94	0.92	0.90	0.89	0.87	0.87
10%	0.93	0.91	0.91	0.89	0.86	0.85	0.83	0.82

Figure 4.18: Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 1:0:0 (opinion consistency 1.0)

Error \ Samples	10	30	50	100	300	500	1500	3000
0.5%	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1.5%	1.00	1.00	1.00	1.00	0.99	0.99	0.98	0.97
2.5%	1.00	1.00	0.99	0.99	0.97	0.96	0.95	0.94
3.5%	1.00	0.98	0.99	0.98	0.95	0.94	0.92	0.90
5%	1.00	0.98	0.98	0.96	0.92	0.90	0.87	0.85
7.5%	1.00	0.95	0.95	0.91	0.85	0.82	0.78	0.76
10%	0.96	0.93	0.90	0.86	0.77	0.75	0.69	0.67

Figure 4.19: Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 1:0:1 (opinion consistency 0.5)

Error \ Samples	10	30	50	100	300	500	1500	3000
0.5%	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
1.5%	1.00	1.00	1.00	1.00	0.99	0.98	0.96	0.96
2.5%	1.00	1.00	0.99	0.99	0.96	0.95	0.93	0.92
3.5%	1.00	0.98	0.99	0.96	0.93	0.91	0.88	0.87
5%	1.00	0.98	0.96	0.94	0.88	0.87	0.82	0.80
7.5%	1.00	0.92	0.91	0.87	0.79	0.76	0.72	0.70
10%	0.99	0.92	0.87	0.80	0.70	0.66	0.60	0.57

Figure 4.20: Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 3:1:1 (opinion consistency 0.8)

Depending on the application, different minimum accuracies would be necessary. For example, should the algorithm spot contradicting opinions confidently, or should it only give a broad idea of the general consistency of opinions? To answer the question of whether or not the consistency of opinions can be measured through the help of NLP methods over time, it can be said: Yes, under certain circumstances, it is feasible. Enough data needs to be available, and the use-case needs to be able to tolerate an error of at least 5% to reach achievable accuracies. In the future, with the advancement of machine learning models and language understanding, it will become more feasible.

With the simulations performed in this section—providing an understanding of the model accuracies required to predict opinion consistency values within a reasonable error margin—we conclude the experiments. All requirements (R1–R6), as defined in Section 3.2, should have been fulfilled. In the next chapter, we summarize and discuss the results of the experiments.

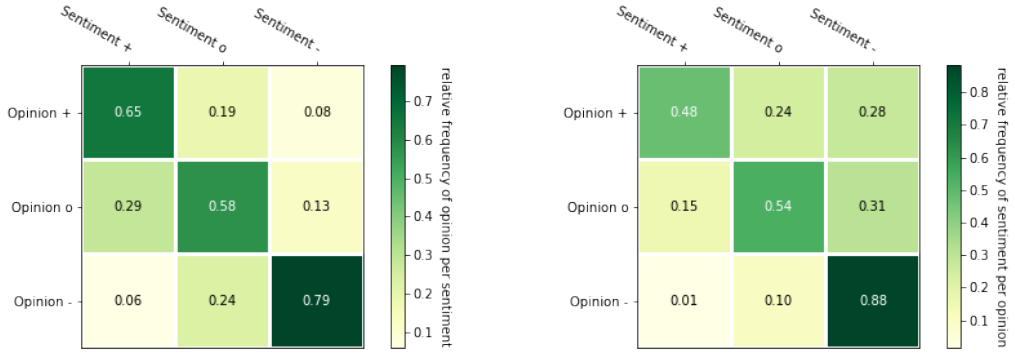
CHAPTER 5

Evaluation

The overall vision for this project was to automatically extract opinions from text and monitor their change over time. Since this is novel and difficult, the goal of this project was to examine what is possible with current technology, conclude what can be reasonably achieved and which areas require more attention to achieve the vision.

It became quickly evident that it was too ambitious to capture opinions in general, so the scope was restricted to the opinion on a specific question: "Should a Lockdown be implemented?" The algorithm should output one of three answers to this question based on a given text: (1) Yes, (2) No, or (3) Neither, which should be given in the case of a non-subjective opinion.

In the beginning, we examined how much the sentiment of a sentence correlates with its derived opinion (Figure 5.1). If the correlation had been reasonably high, we could



(a) Opinion distribution per sentiment label (b) Sentiment distribution per opinion label

Figure 5.1: Relationship between sentiment and opinion categories in the LOCKDOWN dataset

have applied regular sentiment analysis to predict the opinion. After examination, the correlation was not considered high enough; thus, an alternative approach was necessary. The graphic shows the correlation between the sentiment of a sentence and its derived opinion. Sub-figure 5.1a displays the distribution of opinions per sentiment. We can see, for example, in the case of the document having a positive (+) sentiment, the document's derived opinion is positive in 65%, neutral in 29%, and negative (-) in 6% of the cases. Sub-figure 5.1b shows the other direction.

Since we did not consider the sentiment sufficient to predict the opinions, we decided to predict the opinion directly. At first, we explored hard-coded rule-based classifiers and considered unsupervised methods, but then moved to supervised machine learning approaches because, in opinion mining, they usually outperform unsupervised methods. [SLC17]

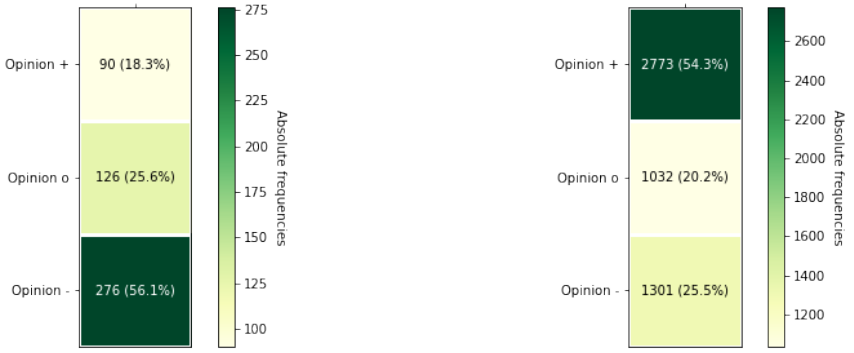
It became apparent that an opinion can be highly subjective, making it difficult to consistently assign labels during the manual labeling process of the dataset. Therefore, we expect the presence of label noise, which is interpreted as assigning wrong labels to some samples. Due to this label noise, the performance of machine learning algorithms can suffer, as was discussed in Section 2.3.1. As a result, the highest achievable performance by the algorithm might be lower than 100%.

5.1 Opinion Classification

We tested different algorithms on the dataset. Since it was comparatively small, the resulting performance depended heavily on the distribution of samples between the training and test set. As a countermeasure, we performed Monte Carlo cross-validation, i.e., ran the experiments multiple times, each time with random train-test splits, and finally, took the averages. The resulting averages provide us with a good idea of the actual performance of the algorithms.

The LOCKDOWN dataset used in the experiments was relatively small because the targeted opinion was a specific question. Additionally, due to the chosen topic, there is a considerable class imbalance. There are many more negative samples than positive (see Figure 5.2a). It makes sense since a Lockdown is generally perceived as a restriction to freedom by the Austrians, and thus politicians are wary of openly speaking in favor of it. When somebody did speak in favor of the Lockdown, they generally indicated that there would be no other choice.

The methods of splitting the LOCKDOWN dataset into training and test partitions evolved during the experiments. We started with a stratified split of 85-15 but encountered a problem with that approach. The accuracy of a random guessing approach should be 33% since there are three possible classes. However, due to the imbalance of the dataset, it was possible to achieve a higher accuracy simply by predicting the most frequent class all the time. With the stratified dataset, this would lead to an accuracy of 56%. Thus, an algorithm like BM25 could achieve a high result simply because it is more likely to



(a) Class distribution in the LOCKDOWN set (b) Class distribution in the MEASURES set

Figure 5.2: Absolute and relative class frequencies of the two datasets

choose the majority class. To make it impossible for such a primitive approach to achieve an accuracy above 33%, we utilized a holdout approach. We chose the same number of samples from each class to form the test set and assigned the rest to the training set, ensuring a uniform class distribution in the test set.

It could be desirable in some applications to have a model learn to predict a class based on frequency, but in this case, every class has the same importance; hence, frequency should not impact the model's decision. Thus, we first used a holdout approach, which provides a test set with the same number of samples in each class. Later, we switched to a stratified split but used the macro-average F1-score to evaluate performance, which we considered a more effective measure to deal with the class imbalance.

We tested three deep learning models: A simple deep learning network based on a bag-of-words embedding, a more advanced LSTM network, and a pre-trained BERT model that we trained for the downstream task.

Besides the deep learning models, we used a Multinomial Bayes (MNB) classifier and adapted the BM25 document ranking algorithm to perform classification. The BM25 algorithm calculates document scores based on how well they match a query. In order to apply it to text classification, the samples of the training set served as a lookup table. Given a sample of the test set, we used the algorithm to rank the samples in the lookup table and return the average class of the top n samples.

Table 5.1 shows a performance comparison between the best versions of each algorithm on the LOCKDOWN set. Each approach was trained and evaluated for the displayed number of times. The MNB was run 1000 times, instead of 100 like the others, because it executed quickly. We evaluated the performance first on the accuracy, which is a solid general-purpose metric. Additionally, we collected the macro-average F1-score since it provides a better evaluation of model performance on imbalanced datasets. We chose it over the micro-average F1-score, which would treat each sample with equal importance, thus coming closer to the accuracy metric. To capture the spread of the F1-Scores across

Approach	Mean Acc	Mean F1	Std Dev	Min. F1.	Max. F1.	Runs
MNB	0.53	0.48	0.055	0.31	0.66	1000
BERT	0.56	0.47	0.088	0.23	0.66	100
LSTM	0.51	0.42	0.051	0.30	0.55	100
Embedding Bag	0.42	0.38	0.066	0.23	0.58	100
BM25	0.47	0.28	0.057	0.10	0.42	100

Table 5.1: Performance comparison of various machine learning approaches on the LOCKDOWN set, sorted by F1-Score

the runs, we also display the standard deviation and minimum and maximum values. As expected, BERT outperforms the LSTM, which in turn outperforms the Bag-of-Words model (Embedding Bag), but it is a surprise that the MNB outperforms the BERT slightly in terms of mean F1, even though it is slightly behind in terms of accuracy. The BM25 approach had the most significant gap between accuracy and F1-Score, indicating a strong influence of the dataset imbalance.

The classification results on the LOCKDOWN set can be considered mediocre, and there are possible explanations: First, the dataset size is small, making it difficult to train a generalized model. Additionally, there are even fewer samples to train from in the minority class due to the class imbalance, making it more challenging to achieve good macro average F1-Scores. Another reason could be that the domain is complex, making it more difficult for the model to learn. Furthermore, the subjectivity of opinions is high, making it difficult to label the samples consistently, increasing the likelihood of introducing label noise, which will make it more difficult for the model.

It is quite possible that with a more extensive dataset, the results would improve. However, it is not always possible to increase the number of samples. That is especially true when tracking the consistency of opinions on a particular topic of interest. Still, it would be beneficial to have a comparison with a larger dataset.

In order to assess the impact of dataset size on classification performance, a second, much more extensive ($\tilde{1}$0x) dataset was collected, referred to as the MEASURES set (see Section 3.4). It contains speeches that are about measures for containing the spreading of the Coronavirus. The possible opinions are 0 (against measures), 1 (neither for nor against measures), and 2 (for measures). The MEASURES set is also imbalanced (Figure 5.2b), roughly to the same degree as the LOCKDOWN set, but due to the increased overall size, it should be less of a concern. Due to the increased labeling effort of the MEASURES set, we used a custom annotation tool, which sped up the process and helped increase the quality of labels.

We reran the same algorithms on the MEASURES set, with an 85-15 stratified train-test split. Due to the increased dataset size, a separate stratified validation set of 15% was split randomly from the training set during training of the deep neural network models. Additionally, we tested Google's Open AI API on the second dataset. Contrary to high

Approach	Mean Acc	Mean F1	Std Dev	Min. F1.	Max. F1.	Runs
BERT	0.70	0.66	0.022	0.59	0.71	100
MNB	0.67	0.65	0.017	0.60	0.70	1000
LSTM	0.68	0.64	0.021	0.58	0.69	100
Embedding Bag	0.62	0.58	0.022	0.52	0.63	100
Open AI	0.50	0.49	-	0.49	0.49	<1
BM25	0.42	0.38	0.042	0.25	0.43	100

Table 5.2: Performance comparison of various machine learning approaches on the MEASURES set. The Standard Deviation refers to the F1-Score.

expectations, the actual results were much lower. The exact reasons were not clear. One issue was the high cost per query, which caused us to use up the available budget quickly, i.e., we only managed to classify 262 samples with an accuracy of 50%. Since the budget was used up in the first run already, there was no possibility of querying the API with different input to investigate the reasons behind the low accuracy. One explanation could be that the API is deriving meaning from the class labels. In our API calls, we defined them as the numbers 0, 1, and 2. Possibly labels like "Negative Opinion" would have achieved better results.

Table 5.2 displays the averaged results of algorithms on the MEASURES set, ordered by F1-Score. Overall, the results are significantly better than on the LOCKDOWN set, so dataset size appears to have played a significant role. Like on the LOCKDOWN set, the deep neural network models are ranked in order of sophistication, except for Open AI. Open AI is displayed with less than one run because it was only evaluated on a subset of the test data once. Therefore, the Open AI results are likely not to be representative of its actual capabilities. The MNB approach again performed very well and is only one percent behind BERT, but also the LSTM performed almost equally well. The BM25 approach has a significantly better F1-score than before, and the gap to the accuracy is not as large as it was on the LOCKDOWN set, meaning it dealt better with the class imbalance this time.

After concluding the performance comparison, we facilitated the BERT model to predict the opinions on the entire MEASURES set as a preliminary step to compare actual to predicted graphs on opinion data. The overall accuracy of those predictions was 70%.

Before we visualized opinion consistencies, we plotted opinion distributions per party and speaker. The graphs between actual and predicted opinions were viewed side-by-side (Figure 5.3). We considered the differences small enough for the predicted graphs to convey a representative impression of the actual sentiments. We concluded that those graphs require a considerably lower model accuracy to remain useful, as compared to the ones for *opinion consistency*.

(a) Actual opinion distribution per party (b) Predicted opinion distribution per party

Figure 5.3: Opinions of the parties on the measures against the Coronavirus

5.2 Opinion Consistency

Ultimately, the idea would be to monitor the consistency of opinions based only on written text to compare the consistency of different groups or individuals in general and over time. We used two formulas (Section 3.2) to calculate the consistency values based on opinion data. The first is defined by the relative amount of samples belonging to the majority class, between positive and negative opinions. So, for example, a group or individual whose statements about a particular topic make up 70% positive and 30% negative statements would be considered 70% consistent. The other definition would also include neutral opinions in the calculation.

The validity of conclusions drawn from opinion consistency graphs depends on the accuracy of the machine learning models with which they are predicted. To get an idea about the usefulness, we look at the predicted opinion consistency vs. the actual opinion consistency of the different political parties (see Figure 5.4) in the case of our MEASURES dataset.

We can observe that the predictions for the SPÖ, FPÖ, and NEOS are better than those for ÖVP and Grüne, which is reasonable since it is more challenging to predict opinion consistencies the more they deviate from average values. As we have determined before, the more the opinion ratios deviate from an even distribution (1:1:1), the higher the required model accuracies to predict the opinion consistencies with the same level of accuracy. We can also observe that the predicted consistency deviates a lot in the case of "ohne Klubzugehörigkeit" (no party affiliation), which is expected because the actual consistency is high, and the sample size is small.

Regarding the usefulness of those specific plots, we consider them rough estimates of the actual values. Unfortunately, the predictions are not resembling very high or very low consistency values as well. Also, a significant number of samples is required before the predicted values become reliable.

Additionally, we determined the necessary accuracy for a machine-learning algorithm to predict the actual *opinion consistency* within a 95 confidence interval. The minimum

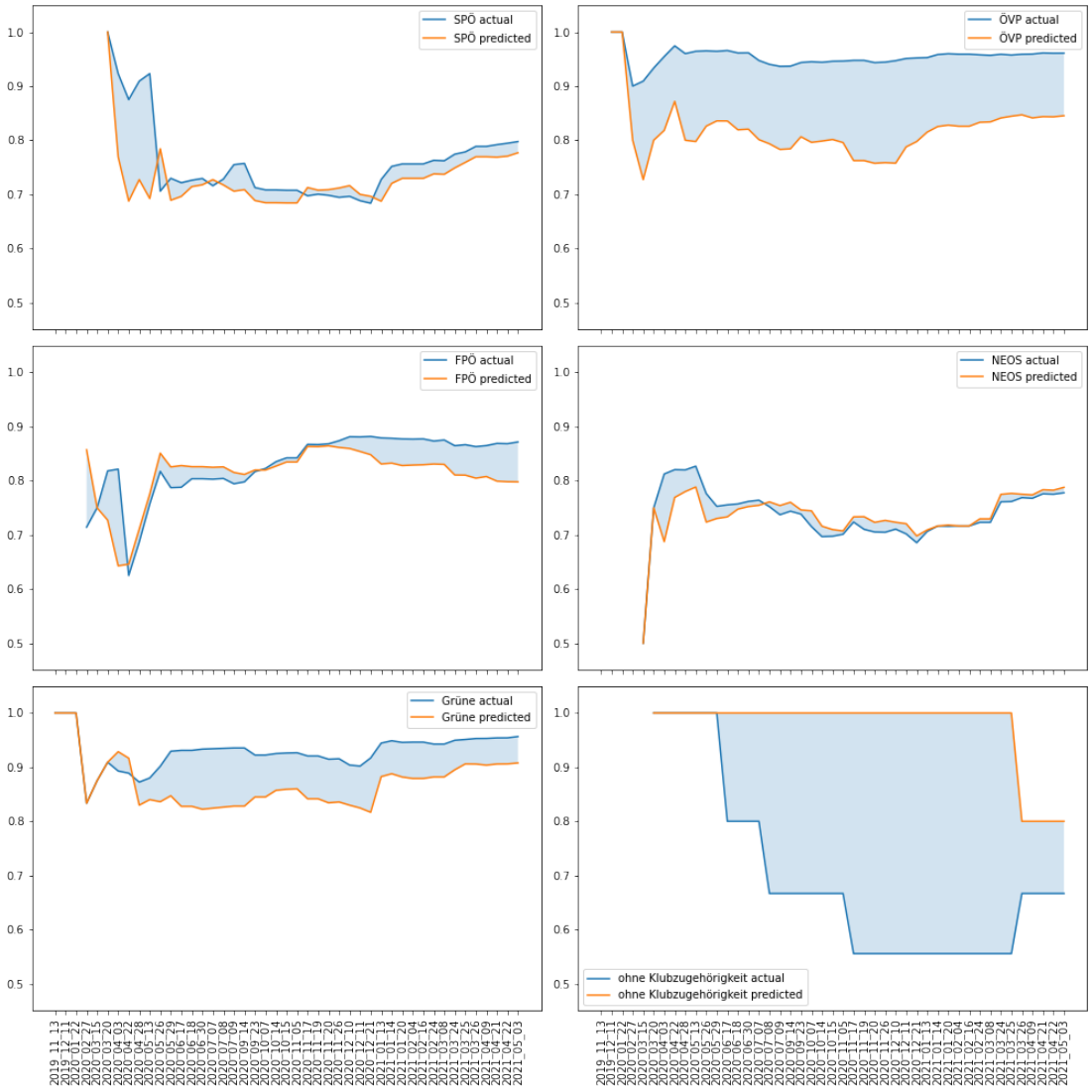


Figure 5.4: The predicted vs. actual opinion consistencies per party, including neutral opinions

results depend on the ratios between negative, neutral, and positive opinions. We determined values for four different ratios (refer to Section 4.4.2 for all results). In Table 5.5 we see the minimum accuracies for a ratio of 3:1:1; thus, for three negative opinions, there is one neutral and one positive each. For example, we read from the table that after 100 expressed opinions, if we want to predict opinion consistency with a maximum error of 5%, we need a model accuracy of at least 88%.

We made various observations by looking at the tables of minimum accuracies (Section 4.4.2). For low sample sizes and low error margins, almost perfect model accuracy is

Error \ Samples	10	30	50	100	300	500	1500	3000
0.5%	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
1.5%	1.00	1.00	1.00	1.00	0.99	0.98	0.96	0.96
2.5%	1.00	1.00	0.99	0.99	0.96	0.95	0.93	0.92
3.5%	1.00	0.98	0.99	0.96	0.93	0.91	0.88	0.87
5%	1.00	0.98	0.96	0.94	0.88	0.87	0.82	0.80
7.5%	1.00	0.92	0.91	0.87	0.79	0.76	0.72	0.70
10%	0.99	0.92	0.87	0.80	0.70	0.66	0.60	0.57

Figure 5.5: Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 3:1:1 (opinion consistency 0.8)

required. The predictions become more feasible with increased sample sizes or if higher error margins can be tolerated. The minimum required accuracies depend on the ratio of the underlying opinions used to calculate opinion consistency. A ratio of 1:1:1 is easiest to predict, 1:0:0 is the most difficult, and 1:0:1 is in-between. Therefore, if all ratios are equally important, the minimum model accuracy must be read from the 1:0:0 table. Depending on the application domain, if we assume that extreme values of opinion consistency will be rare, the lower values of the other tables can be used as a guideline.

Depending on the application, different minimum accuracies would be necessary. For example, if the algorithm spots contradicting opinions confidently, a higher accuracy will be required. A lower one will suffice if it should provide only a broad idea of the general consistency of opinions. *To the initial question of whether the consistency of opinions can be monitored with the help of NLP methods over time, we answer: Yes, under certain circumstances, it is feasible. Enough data need to be available, and the use-case needs to tolerate an error of at least 5% to reach achievable accuracies. In the future, with the advancement of machine learning models and language understanding, it will become more feasible.*

5.3 Challenges

We encountered several challenges during the implementation of our approach of monitoring opinion consistency.

- As seen in our case, sentiment does not always match a statement's opinion, making it more challenging to use a general sentiment classifier to predict an opinion.
- Sometimes the speaker does not talk about their own opinion but about someone else's, which is an additional challenge for a machine learning model.

- Datasets can get very small, depending on the specificity of the topic. The smaller a dataset gets, the more difficult it will be to train a reliable model. Additionally, when a dataset becomes small enough, it could be more feasible to examine the few samples by hand instead of training a machine learning model.
- In supervised learning approaches, it is necessary to label a training set manually, which was the biggest challenge to the practical feasibility of the implemented approach. Additionally, opinions are subjective, and thus, this approach is especially prone to label noise.
- In supervised learning approaches, it is necessary to label a training set manually, which was the biggest challenge to the practical feasibility of the implemented approach. Additionally, opinions are subjective, and thus, this approach is especially prone to label noise.
- Each topic could be drastically different regarding identifying a positive or negative opinion, making it necessary to train a separate model for each topic. Even if transfer-learning is used, the model probably still has to be fine-tuned on each dataset.
- The understanding of what an opinion consists of is diverse. To predict an opinion from a text could mean something different for different people. We have used a simple definition, trading off the nuance of captured opinions for simplifying implementation complexity. Finding the right balance is an additional challenge.
- When aiming to predict an opinion in the sense of "what the person really meant," it might be too subjective to predict in a meaningful way. When the person is not explicitly stating an opinion, it might be impossible to know for sure what the actual opinion is. Depending on the application domain, the issue might become less pronounced, but there will always be some level of subjectivity.

5.4 What is possible and future directions

A generic approach of monitoring opinions in general is out of reach yet. However, it is possible to use NLP methods to predict consistency values for opinions on a specific topic. Although, to make meaningful predictions, the accuracy scores of predictions would have to be significantly higher (at least for graphs of opinion consistency) than what we have achieved in this project, as was shown in section 5.2. Whether these values can be achieved depends on multiple factors, like the complexity of the domain, the definition of an opinion, and the size and quality of the dataset.

Concretely, based on the performed experiments, the following approach is suggested. A topic is defined, which can be identified easily by specific keywords. Manual labeling of opinions is performed on sentences containing those keywords. The best available pre-trained attention model is chosen and trained on the dataset. When the achieved performance is satisfactory, it can be used to monitor opinions on the chosen topic.

Since the amount of data for such an approach needs to be large, it can only be used on widely discussed topics. In the case of the project's domain (parliamentary speech protocols), it would require a long-term time horizon of an established topic for that to be the case. Therefore, it cannot be used when insight needs to be gathered quickly in response to a newly arising topic, but better is used for general topics that have been discussed for several years already.

However, in other domains, such as social media (e.g., Twitter messages), this approach could be easier to implement, as overall, a lot more data are produced, making it easier to accumulate a larger dataset. Additionally, the data itself are less complex, which would make it easier to predict the opinions.

Finally, humans use context information to interpret a text that is not coming from the same text. Simpler models have no access to such context information, but advanced models, based on the concept of transfer learning, like the BERT model, can be said to make use of such information. They are pre-trained on a large set of text documents (e.g., Wikipedia articles) before they are trained on the target dataset. The performance of such models could be further improved, by providing relevant context information, alongside the expression, but this is only an idea. In theory, with enough context information, a general model could be built, coming close or even surpassing the accuracy of humans because they can store and process more data than a typical human could. The transfer learning approach seems to be the most promising in achieving the vision of a general opinion extraction system.

CHAPTER 6

Conclusion

In this experimental study, we explored the possibilities of measuring the consistency of opinions with the help of NLP methods. We have defined the term *opinion* and implemented a method to extract opinions from textual data. We provided two formulas for calculating *opinion consistency*, a value that makes the consistency of opinions quantifiable. We gathered two datasets, annotated them, and ran different machine learning algorithms to extract opinions. We calculated the opinion consistency values for speakers and parties and visualized them. In addition, we examined the impact of a model's accuracy on the accuracy of predicted opinion consistency values. We used the insight gained by implementing such an approach to answer the following research questions:

Q1a What is the practical feasibility of monitoring *opinion consistency*, a value representing the consistency of opinions on a topic, through the means of supervised ML methods?

Using supervised machine learning methods to extract opinions requires an annotated dataset on which a model is trained. The biggest challenge to feasibility in the proposed approach is that every topic requires creating and annotating a different dataset. Thus, the approach is best suited if the intention is to monitor a small selection of topics over a long period of time. The other factors, e.g., training the algorithms and computing opinion consistency values, are minor considerations since they can be automated.

Q1b What is the usefulness of measuring and visualizing the consistency of opinions based on opinion data predicted by supervised ML methods?

As shown in Section 4.4.1, the usefulness depends on the accuracy of predicted opinion consistency values, which in turn, depend on the number of opinions and

the accuracy of the chosen ML method. The precision of the predicted opinion consistency increases with the number of opinions. Additionally, we observed an improved prediction accuracy of opinion labels on the larger dataset compared to the smaller one. We conclude, the proposed method is most useful for topics with a high number of opinions.

Q2 What performance do various ML architectures achieve in predicting opinions in the domain of Austrian political speeches in the German language?

In the Sections 4.1 and 4.2, we compared the classification performances of five different machine learning models on two datasets constructed from speech transcriptions of Austrian politicians. On the first dataset with around 500 records, we found that according to accuracy, BERT performed best (56%), follwed by the MNB (53%), the LSTM (51%), the BM25 (47%), and the Bag-of-Words model (42%). On the second dataset, with around 5000 records, BERT also achieved the highest accuracy (70%), followed by the LSTM (68%), the MNB (67%), the Bag-of-Words (62%), and the BM25 (42%). Notable is the high performance of the MNB, which was almost on par with BERT.

Q3 What could be minimum performance thresholds for ML algorithms to predict the consistency of opinions to a desirable precision?

In Section 4.4.2, we determined the minimum model accuracies that are required to predict opinion consistency within a certain margin of error. We found that the minimum accuracy depends on the number of opinions on which the opinion consistency value is calculated and the ratio between positive, negative, and neutral opinions. Extreme values (exceptionally high or low opinion consistencies) are more difficult to predict accurately than moderate ones. Additionally, we found that almost perfect prediction accuracy is required to achieve low margins of error on low numbers of opinions. Above a certain threshold for error margin and sample size, the required model accuracies become achievable.

Q4 How useful are visualizations of opinion data of speakers and parties that are based on predictions made by various supervised ML algorithms?

In Section 4.3 we visualized the actual and predicted opinion data. We saw that to be useful those visualizations required a lower prediction performance than the visualizations of opinion consistency. We found that the visualizations of opinions of parties and speakers, based on prediction accuracy of 70%, are sufficiently representative to be used for recognizing the overall sentiment towards a topic.

We take some space to reflect critically on the used methodology and achieved results. Supervised machine learning methods are not the only way we could have achieved our goal, but we have chosen them over unsupervised methods because of their better state-of-the-art performance in opinion mining tasks. Future work can explore unsupervised

methods to eliminate the manual annotation process, which we have found to be the greatest obstacle to the feasibility of the presented approach.

We think that we have covered the landscape of deep learning methods reasonably well. We could have covered more statistical models, but based on the state-of-the-art results on similar tasks (e.g., sentiment analysis), deep learning methods are likely to perform better in any case. How we have applied BM25 is debatable, but we think we did a reasonable job considering that it is a document ranking algorithm and not primarily a text classification algorithm. We could have tested more combinations of pre-processing steps to increase performance, but we do not expect significant gains beyond a few percentage points, based on the observations on the combinations we have tested. We provided detailed performance reports on the models we evaluated and used the same metrics to make results comparable. We believe the used Monte Carlo cross-validation approach made the results reliable.

We are satisfied with our definition of opinion as a quadruple, and we are partially satisfied with the definition of opinion consistency. We believe the definition of opinion consistency can be improved by weighing recent opinions more strongly than older ones. To investigate the usefulness of opinion consistency visualizations, we utilized the graphs of Section 4.4.1. We could have improved the comparisons on opinion data by calculating the divergence between actual and predicted values in addition to the visual comparisons. However, we believe the visual comparisons are sufficient because we additionally provided the tables in Section 4.4.2, which tell precisely how accurate the predictions will be, based on the model's accuracy.

In summary, our main contributions are:

1. Proposing a definition of opinion consistency, a value that makes the consistency of opinions quantifiable.

2. Designing, implementing, and testing a method for visualizing the consistency of opinions over time of individuals and groups.

3. Creating various visualizations of opinion data and examining their usefulness.

4. Evaluating the performance of different supervised machine learning models on two datasets from the domain of political speeches in the German language.

5. Determining a table of minimum accuracies that a machine learning model would have to achieve for predicting opinion consistency with a certain accuracy.

In the next section, we propose ideas for future research on the topic that we could not cover in this work.

Future Work In this work, we employed supervised machine learning methods to extract opinions from textual data. The most significant disadvantage with this approach is the manual annotation effort that is involved. In the future, it would be interesting to examine unsupervised methods to eliminate the manual labeling process. One of several ideas is to apply topic modeling to group text passages per topic and then use a sentiment lexicon to determine the opinion's sentiment.

Future work should explore how well a model can generalize across topics. In this work, we have trained separate classifiers per dataset. It would be interesting to examine how well a model trained on one topic will perform on another topic. If it would be possible to train a model that performs well on many topics, it becomes more feasible to apply the technique proposed in this work in a generalized way.

The formulas for calculating the opinion consistency values determine which type of insights can be derived. In this work we have used a formula, that is good at showing the overall consistency over long periods. Future work should investigate other formulas that can answer different questions. For example, if the interest lies more in spotting contradicting opinions in short intervals, a rolling window could be used, or opinions could be weighed by how far they date back.

In this work, we have proposed a two-phase method of first extracting text passages related to a chosen topic and then classifying the opinion on those passages. It would be interesting to test the performance of classifiers that perform both tasks at the same time. Such a classifier could output the additional label of *unrelated* if the text is not about the desired topic. Alternatively, a classifier that outputs two labels—one for the topic and another for the opinion's sentiment—could be employed.

Finally, improving the classification performance is a reliable way to improve the accuracy of predicted opinion consistency values and thus the conclusions formed thereon. In this work, we have achieved a performance of 70% with a BERT model on a dataset of around 5000 records. Considering that the domain is complex, we consider it a fair achievement. Future work could improve performance in that domain, e.g., by hyperparameter tuning, applying different pre-processing techniques, utilizing transfer learning, or using other models. It would also be interesting how such an approach performs in different domains, e.g., on social media.

In this work, we have successfully implemented a method for visualizing the consistency of opinions of individuals and groups. We gathered valuable insights in regards to the practical feasibility and usefulness of the implemented approach. Based on our evaluation, we predict that considerable efforts are required before such an approach becomes useful for a broad range of application domains.

List of Figures

2.1 CNN architecture for sentence classification [ZW15] 11
2.2 Basic RNN architecture [LBH15] 13
2.3 Schematic overview of different RNN architectures. [KKDC19] Input vectors are denoted by x, output vectors by y and hidden state vectors by c. Merging arrows indicate a concatenation of vectors and a splitting arrow indicates a copy operation. 14
2.4 Overview of the transformer architecture [VSP+17] 15
2.5 Different syntactic parses of the same sentence as produced by spaCy [HMVB20] 21
2.6 Distribution of submitted model types for the SemEval-2019 Task 6 (sub-task A) [ZMN+19] 26
2.7 Showing the concept of under- and overfitting in a binary classification task in two feature dimensions. The blue line represents a classifier splitting the feature space into two regions. The classifiers, from left to right, are likely to generalize too much, appropriately, and too little. 30
2.8 Basic DNN architecture [Agg18] 32

3.1 Relationship between sentiment and opinion categories in the first dataset 47
3.2 Screenshot of the annotation software that aided in the annotation process of the second dataset 51

4.1 Speech lengths in the LOCKDOWN dataset 56
4.2 Results of the bag-of-words neural network on the MEASURES dataset . 60
4.3 Results for the LSTM neural network on the MEASURES dataset 61
4.4 Speech lengths in the MEASURES dataset 62
4.5 Results for the BERT neural network on the MEASURES dataset 63
4.6 N-Best comparison for the BM25 model 64
4.7 Results for the BM25 approach on the MEASURES dataset 65
4.8 Results for the Multinomial Bayes approach on the MEASURES dataset . 65
4.9 Classification report of the labels, predicted with BERT, used in the opinion data visualizations and opinion consistency comparisons 68
4.10 Opinions on MEASURES per party 69
4.11 Opinions of the top 20 speakers on MEASURES 70
4.12 Opinion consistency over time per party, based on $OpCons_2(G, H, t)$ . . . 72

4.13 The predicted vs. actual opinion consistencies per party, including neutral opinions 73
4.14 Opinion consistency over time per speaker, based on $OpCons_2(G, H, t)$ . . 74
4.15 The predicted vs. actual opinion consistencies for selected speakers, including neutral opinions 75
4.16 The predicted vs. actual opinion consistencies for selected speakers, excluding neutral opinions 76
4.17 Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 1:1:1 (opinion consistency 0.67) 79
4.18 Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 1:0:0 (opinion consistency 1.0) . 79
4.19 Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 1:0:1 (opinion consistency 0.5) . 79
4.20 Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 3:1:1 (opinion consistency 0.8) . 80

5.1 Relationship between sentiment and opinion categories in the LOCKDOWN dataset 81
5.2 Absolute and relative class frequencies of the two datasets 83
5.3 Opinions of the parties on the measures against the Coronavirus 86
5.4 The predicted vs. actual opinion consistencies per party, including neutral opinions 87
5.5 Minimum required model accuracies for predicting the opinion consistency inside a 0.95 confidence interval within a certain margin of error after a certain amount of samples with opinion ratios of 3:1:1 (opinion consistency 0.8) . 88

List of Tables

2.1 Example of a confusion matrix 34

3.1 The annotations on the two example sentences, according to the seven categories. 49
3.2 Number of sentences per topic 50

4.1 Performance comparison of various machine learning approaches on the LOCKDOWN set, sorted by F1-Score. 58
4.2 Classification report: BOW on the MEASURES set 61
4.3 Classification report: LSTM on the MEASURES set 62
4.4 Classification report: BERT on the MEASURES set 63
4.5 Classification report: BM25 on the MEASURES set 65
4.6 Classification report: MNB on the MEASURES set 66
4.7 Classification report: Open AI Davinci (GPT-3) on the MEASURES set . 67
4.8 Performance comparison of various machine learning approaches on the MEASURES set. The standard deviation refers to the F1-Score. 67

5.1 Performance comparison of various machine learning approaches on the LOCKDOWN set, sorted by F1-Score 84
5.2 Performance comparison of various machine learning approaches on the MEASURES set. The Standard Deviation refers to the F1-Score. 85

Bibliography

[AC10] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[AC17] Ehsan Mohammady Ardehaly and Aron Culotta. Learning from noisy label proportions for classifying online social data. *Social Network Analysis and Mining 2017*, 8(1):1–18, November 2017.

[Agg18] Charu C Aggarwal. Neural networks and deep learning. *Springer*, 10:973–978, 2018.

[AMPZ17] R. Ahmad, H. Mannan, A. Pervaiz, and F. Zaffar. Aspect based sentiment analysis for large documents with applications to US presidential elections 2016. In *AMCIS 2017 - America's Conference on Information Systems: A Tradition of Innovation*, volume 2017-August, 2017.

[Bar18] Adrien Barbaresi. A corpus of German political speeches from the 21st century. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[BBL99] Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. *Machine Learning*, 34(1):177–210, 1999.

[Ber19] Daniel Berrar. Cross-Validation. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 1-3:542–545, January 2019.

[BEW+18] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry. *Journal of Automotive Software Engineering*, 1(1):1–19, December 2018.

[BG18] Toms Bergmanis and Sharon Goldwater. Context Sensitive Neural Lemmatization with Lematus. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies, Volume 1 (Long Papers), pages 1391–1400, New Orleans, Louisiana, 2018. Association for Computational Linguistics.

[Bla] Andreas Blaette. GermaParl. Corpus of Plenary Protocols of the German Bundestag. https://github.com/PolMine/GermaParlTEI. Accessed: 2020-05-21.

[BM06] Sabine Buchholz and Erwin Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, 2006. Association for Computational Linguistics.

[Boe84] Barry W Boehm. Verifying and validating software requirements and design specifications. *IEEE software*, 1(1):75, 1984.

[Bol19] Marcel Bollmann. A Large-Scale Comparison of Historical Text Normalization Systems. *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1:3885–3898, April 2019.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[CDEU17] Mark Cieliebak, Jan Milan Deriu, Dominic Egger, and Fatih Uzdilli. A twitter corpus and benchmark resources for german sentiment analysis. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 45–51, 2017.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *NIPS 2014 Workshop on Deep Learning*, December 2014.

[CGG+07] Ofelia Cervantes, Francisco Gutiérrez, Ernesto Gutiérrez, Esteban Castillo, J. Alfredo Sánchez, and Wanggen Wan. Expression: Visualizing Affective Content from Social Streams. In *Proceedings of the Latin American Conference on Human Computer Interaction - CLIHC '15*, pages 1–8, Córdoba, Argentina, 2007. ACM Press.

[CS19] Muntazar Mahdi Chandio and Melike Sah. Brexit Twitter Sentiment Analysis: Changing Opinions About Brexit and UK Politicians. In *International Conference on Information, Communication and Computing Technology*, pages 1–11. Springer, Cham, October 2019.

[DCLT19] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, 1(Mlm):4171–4186, 2019.

[FV14] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[GBV20] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for Multi-Class Classification: an Overview. *arXiv preprint arXiv:2008.05756*, August 2020.

[GBZ18] Darina Gold, Marie Bexte, and Torsten Zesch. Corpus of aspect-based sentiment in political debates. *14th Conference on Natural Language Processing - KONVENS 2018*, (Konvens):89–99, 2018.

[GdCL15] Luís P.F. Garcia, André C.P.L.F. de Carvalho, and Ana C. Lorena. Effect of label noise in the complexity of classification problems. *Neurocomputing*, 160:108–119, July 2015.

[GRT21] Siddhant Garg, Goutham Ramakrishnan, and Varun Thumbe. Towards Robustness to Label Noise in Text Classification via Noise Modeling. *ICLR 2021 RobustML and S2D-OLAD Workshops*, 2021.

[GS96] Ralph Grishman and Beth Sundheim. Message Understanding Conference-6: A Brief History. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996.

[HLS13] Emma Haddi, Xiaohui Liu, and Yong Shi. The Role of Text Pre-processing in Sentiment Analysis. *Procedia Computer Science*, 17:26–32, January 2013.

[HMVB20] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020. https://spacy.io/. Accessed: 2021-10-11.

[Hu19] Dichao Hu. An Introductory Survey on Attention Mechanisms in NLP Problems. *Advances in Intelligent Systems and Computing*, 1038:432–448, September 2019.

[HW00] Vasileios Hatzivassiloglou and Janyce M. Wiebe. Effects of adjective orientation and gradability on sentence subjectivity. *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*, pages 299–305, 2000.

[JH18] Yahia Hasan Jazyah and Intisar O. Hussien. Multimodal Sentiment Analysis: A Comparison Study. *Journal of Computer Science*, 14(6):804–818, June 2018.

[Jiv11] Anjali Jivani. A Comparative Study of Stemming Algorithms. *Int. J. Comp. Tech. Appl.*, 2:1930–1938, 2011.

[JPLN19] Ishan Jindal, Daniel Pressel, Brian Lester, and Matthew Nokleby. An Effective Label Noise Model for DNN Text Classification. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:3246–3256, March 2019.

[JX17] Zhao Jianqiang and Gui Xiaolin. Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, 5:2870–2879, 2017.

[JZ15] Rie Johnson and Tong Zhang. Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding. *Advances in neural information processing systems*, 28:919, 2015.

[KG20] J. Kersting and M. Geierhos. Aspect phrase extraction in sentiment analysis with deep learning. In *ICAART 2020 - Proceedings of the 12th International Conference on Agents and Artificial Intelligence*, volume 1, pages 391–400, 2020.

[KKDC19] Zadid Khan, Sakib Mahmud Khan, Kakan Dey, and Mashrur Chowdhury. Development and Evaluation of Recurrent Neural Network-Based Models for Hourly Traffic Volume and Annual Average Daily Traffic Prediction. *Transportation Research Record*, 2673(7):489–503, 2019.

[KMH+19] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text Classification Algorithms: A Survey. *Information 2019, Vol. 10, Page 150*, 10(4):150, 2019.

[Lan95] Pat Langley. *Elements of Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[Lan20] Oxford Languages. Oxford Languages Dictionary, 2020. https://languages.oup.com/. Accessed: 2021-10-11.

[LaP18] Joseph LaPorte. Rigid Designators, 2018. https://plato.stanford.edu/archives/spr2018/entries/rigid-designators/. Accessed: 2021-10-11.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[Liu12] Bing Liu. Sentiment Analysis and Opinion Mining. *Synthesis lectures on human language technologies*, 5(1):1–184, May 2012.

[LJ98] Yong H. Li and Anil K. Jain. Classification of text documents. *The Computer Journal*, 41(8):543–545, 1998.

[LJ15] Jiwei Li and Dan Jurafsky. Do Multi-Sense Embeddings Improve Natural Language Understanding? *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, pages 1722–1732, June 2015.

[LL13] Igor Labutov and Hod Lipson. Re-embedding words. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 489–493, 2013.

[LSHL20] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A Survey on Deep Learning for Named Entity Recognition. *IEEE Transactions on Knowledge and Data Engineering*, March 2020.

[MB11] Hassan H. Malik and Vikas S. Bhardwaj. Automatic training data cleaning for text classification. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 442–449, 2011.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, January 2013.

[MCP05a] Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. In *HLT/EMNLP 2005 - Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 987–994. Association for Computational Linguistics (ACL), 2005.

[MCP05b] Ryan Mcdonald, Koby Crammer, and Fernando Pereira. Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, 2005.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT Press, 2018.

[MS99] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations ofwords and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. Neural information processing systems foundation, 2013.

[NGK20] Nikola Nikolić, Olivera Grljević, and Aleksandar Kovačević. Aspect-based sentiment analysis of reviews in the domain of higher education. *The Electronic Library*, 38(1):44–64, January 2020.

[Niv03] Joakim Nivre. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, Nancy, France, April 2003.

[Niv08] Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, December 2008.

[NL18] V. V. Nhlabano and P. E.N. Lutu. Impact of Text Pre-Processing on the Performance of Sentiment Analysis Models for Social Media Data. *2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–6, September 2018.

[NPK+16] Harikrishna Narasimhan, Weiwei Pan, Purushottam Kar, Pavlos Protopapas, and Harish G Ramaswamy. Optimizing the multiclass F-measure via biconcave programming. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1101–1106. IEEE, 2016.

[NS07] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, August 2007.

[OEC] OECD. Trust in Government. https://www.oecd.org/gov/trust-in-government.htm. Accessed: 2021-10-11.

[PCV+00] Georgios Petasis, Alessandro Cucchiarelli, Paola Velardi, Georgios Paliouras, Vangelis Karkaletsis, and Constantine D Spyropoulos. Automatic adaptation of Proper Noun Dictionaries through cooperation of machine learning and probabilistic methods. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00*, New York, New York, USA, 2000. ACM Press.

[PCZ17] Haiyun Peng, Erik Cambria, and Xiaomei Zou. Radical-based hierarchical embeddings for Chinese sentiment analysis at sentence level. In *FLAIRS 2017 - Proceedings of the 30th International Florida Artificial Intelligence Research Society Conference*, pages 347–352. AAAI Press, 2017.

[Por80] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, March 1980.

[PSS+21] Alexis Palmer, Nathan Schneider, Natalie Schluter, Guy Emerson, Aurelie Herbelot, and Xiaodan Zhu, editors. *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, Online, August 2021. Association for Computational Linguistics.

[PT20] Eirini Papagiannopoulou and Grigorios Tsoumakas. A review of keyphrase extraction. *WIREs Data Mining and Knowledge Discovery*, 10(2), March 2020.

[Ras21] Sebastian Raschka. L19.5.1 The Transformer Architecture - Online Lecture, 2021. https://www.youtube.com/watch?v=tstbZXNCfLY. Accessed: 2021-10-11.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[RQH10] Robert Remus, Uwe Quasthoff, and Gerhard Heyer. SentiWS-A Publicly Available German-language Resource for Sentiment Analysis. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, 2010.

[Rud19] Sebastian Ruder. *Neural transfer learning for natural language processing*. PhD Thesis, NUI Galway, 2019.

[RZ09] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[SEA18] Symeon Symeonidis, Dimitrios Effrosynidis, and Avi Arampatzis. A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis. *Expert Systems with Applications*, 110:298–310, November 2018.

[Seb02] Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[SFC+17] Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nădejde. Nematus: a Toolkit for Neural Machine Translation. *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of the Software Demonstrations*, pages 65–68, March 2017.

[SG16] Jasmeet Singh and Vishal Gupta. A systematic review of text stemming techniques. *Artificial Intelligence Review 2016 48:2*, 48(2):157–217, August 2016.

[SLC17] Shiliang Sun, Chen Luo, and Junyu Chen. A review of natural language processing techniques for opinion mining systems. *Information Fusion*, 36:10–25, July 2017.

[SPH+11] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 151–161, 2011.

[SQXH19] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[SRS14] T. Sree Sharmila, K. Ramar, and T. Sree Renga Raja. Impact of applying pre-processing techniques for improving classification accuracy. *Signal, Image and Video Processing*, 8(1):149–157, 2014.

[SST17] Dietmar Schabus, Marcin Skowron, and Martin Trapp. One Million Posts: A Data Set of German Online Discussions. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1241–1244, Shinjuku Tokyo Japan, August 2017. ACM.

[Ste12] Manfred Stede. Discourse Processing. *Synthesis Lectures on Human Language Technologies*, 4(3):1–167, December 2012.

[SV21] Nikolaos Stylianou and Ioannis Vlahavas. A neural Entity Coreference Resolution review. *Expert Systems with Applications*, 168, April 2021.

[SVS13] Rico Sennrich, Martin Volk, and Gerold Schneider. Exploiting Synergies Between Open Resources for German Dependency Parsing, POS-tagging, and Morphological Analysis. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 601–609. INCOMA Ltd. Shoumen, September 2013.

[SW17] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning and data mining*. Springer Publishing Company, Incorporated, 2017.

[TDM03] Brian J Taylor, Marjorie A Darrah, and Christina D Moats. Verification and validation of neural networks: a sampling of research in progress. In Kevin L Priddy and Peter J Angeline, editors, *Intelligent Computing: Theory and Applications*, volume 5103, pages 8–16. International Society for Optics and Photonics, SPIE, 2003.

[TMS03] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The Penn Treebank: An Overview. *Treebanks*, pages 5–22, 2003.

[TQW+15] Duyu Tang, Bing Qin, Furu Wei, Li Dong, Ting Liu, and Ming Zhou. A Joint Segmentation and Classification Framework for Sentence Level Sentiment Classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(11):1750–1761, 2015.

[TTJ06] Michal Toman, Roman Tesar, and Karel Jezek. Influence of word normalization on text classification. *Proceedings of InSciT*, 4:354–358, 2006.

[Tug16] Don Tuggener. *Incremental coreference resolution for German*. PhD thesis, University of Zurich, 2016.

[TWY+14] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, 1:1555–1565, 2014.

[Vit67] Andrew J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, pages 5999–6009, June 2017.

[WBK20] Jeremy Watt, Reza Borhani, and Aggelos K Katsaggelos. *Machine learning refined: foundations, algorithms, and applications*. Cambridge University Press, 2020.

[WBO99] Janyce Wiebe, Rebecca Bruce, and Thomas P. O'Hara. Development and Use of a Gold-Standard Data Set for Subjectivity Classifications. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 246–253, 1999.

[WLS+15] Xin Wang, Yuanchao Liu, Cheng-Jie Sun, Baoxun Wang, and Xiaolong Wang. Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory. *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, 1:1343–1353, 2015.

[WWB19] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. *arXiv preprint arXiv:1907.10701*, July 2019.

[YCG+98] J. P. Yamron, I. Carp, L. Gillick, S. Lowe, and P. Van Mulbregt. A hidden Markov model approach to text segmentation and event tracking. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 1, pages 333–336. Institute of Electrical and Electronics Engineers Inc., 1998.

[YHPC18] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[YKYS17] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative Study of CNN and RNN for Natural Language Processing. *arXiv preprint arXiv:1702.01923*, 2017.

[ZDLS20] Ming Zhou, Nan Duan, Shujie Liu, and Heung Yeung Shum. Progress in Neural NLP: Modeling, Learning, and Reasoning. *Engineering*, 6(3):275–290, 2020.

[Zha20] Mei Shan Zhang. A survey of syntactic-semantic parsing based on constituent and dependency structures. *Science China Technological Sciences*, 63(10):1898–1920, October 2020.

[ZMN+19] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). *arXiv preprint arXiv:1903.08983*, March 2019.

[ZW15] Ye Zhang and Byron Wallace. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1510.03820*, October 2015.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.