Set-Valued Prediction for Part-of-Speech Tagging

Stefan Heid

PADERBORN UNIVERSITY
The University for the Information Society

Department of Electrical Engineering,
Computer Science and Mathematics
Warburger Straße 100
33098 Paderborn

INTELLIGENT SYSTEMS

Intelligent Systems Group (ISG)

Master's Thesis

Set-Valued Prediction for Part-of-Speech Tagging

Stefan Heid

1. Reviewer Prof. Dr. Eyke Hüllermeier
Department of Computer Science
Paderborn University

2. Reviewer Prof. Dr. Michaela Geierhos
Department of English and American Studies
Paderborn University

Supervisor Marcel Wever

December 2, 2019

Stefan Heid sheid@mail.upb.de
Set-Valued Prediction for Part-of-Speech Tagging
Master's Thesis, December 2, 2019
Reviewers: Prof. Dr. Eyke Hüllermeier and Prof. Dr. Michaela Geierhos
Supervisor: Marcel Wever

University of Paderborn
Intelligent Systems Group (ISG)
Department of Computer Science
Pohlweg 51
33098 Paderborn

Abstract

Part-of-speech (POS)-tagging is a method to predict a sequence of word classes given a sequence of words. Set-valued prediction can be used to allow a classifier to make restrained predictions in the face of uncertainty. In this thesis, we present a method for combining set-valued prediction with part of speech tagging to retrieve more reasonable predictions on difficult data. The set size allows the tagger to express its uncertainty of a specific prediction. The devised method can be applied to any POS-tagger capable of predicting a posterior distribution over the tags and provides set-valued predictions in a post-processing step. We implemented the method using the state-of-the-art tagger CoreNLP as our basis. The tagger is tuned to a diachronic corpus of Middle Lower German (MLG) that spans a wide spacial area. Because the corpus also captures human annotator uncertainty, special performance measures have been devised to properly evaluate the tagging performance. The resulting algorithm clearly outperforms our baseline in all considered measures. Our evaluation proves that set-valued prediction can give good predictions with utilities outperforming the accuracy score by large margins. This is especially shown in robustness tests that are difficult for the classifier. Results are compared against a baseline tagger, which profits even more from set-valued prediction.

Acronyms

CP conformal prediction. 1

CRF conditional random fields. 9, 19, 31

HMM hidden Markov model. 5–9, 19, 33, 35, 40, 63

HMP hidden Markov process. 5, 6

IAA inter annotator agreement. 1, 23, 27

KNN k-nearest neighbor. 9

LSTM long short-term memory. 19

MBL memory based learner. 9, 13

MEMM maximum entropy Markov model. 7–9, 19, 28, 31, 63

MLG Middle Lower German. v, 1, 4, 9, 19, 21–23, 25, 27, 29, 40, 41, 56, 58, 61

NLP natural language processing. 1, 3, 9, 10, 19, 23, 63

POS part-of-speech. v, 1–4, 19, 27, 31

PTB Penn Treebank. 23

REN Reference Corpus Middle Low German/Low Rhenish (1200–1650). 22

RNN recurrent neural network. 9

SMAC Sequential Model-based Algorithm Configuration. 15

SVM support vector machine. 8

UBOP Unrestricted Bayes-optimal prediction. 27, 36

Notation

w	word, token
w_i	i-th word in a sequence
$\mathbf{w}$	word sequence, sentence
$\mathcal{W}$	universe of words. Usually we mean the words observed during the training phase.
t	tag
$\mathbf{t}$	tags sequence
T	set of tags
$\mathcal{T}$	universe of tags for example the Penntree-bank tagset
y	single-valued ground truth
$\hat{y}$	single-valued prediction
Y	set-valued ground truth
$\hat{Y}$	set-valued prediction
$\mathcal{Y}$	set of all classes
s	$\lvert\hat{Y}\rvert$ when given to the discount function
K	$\lvert\mathcal{Y}\rvert$
$\#(\cdot)$	frequency of a given data point or feature-set in the corpus
$[\![\cdot]\!]$	Iverson bracket to evaluate boolean predicates: $[\![P]\!] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{otherwise} \end{cases}$

Contents

1 Introduction **1**
1.1 Thesis Structure . . . 2

2 Prerequisites **3**
2.1 Part-of-Speech Tagging . . . 3
2.2 POS-Tagging as a Machine Learning Problem . . . 4
2.3 Evaluation . . . 11

3 Related Work **19**

4 Corpus **21**
4.1 Description . . . 21
4.2 Tagset . . . 24
4.3 Multi-Label Tags . . . 25

5 Algorithm **27**
5.1 Problem Definition . . . 27
5.2 Data Preparation . . . 28
5.3 Modified Performance Measures . . . 30
5.4 CoreNLP Tagger . . . 31
5.5 Extension . . . 34

6 Evaluation **39**
6.1 Dataset Splitting and Experiment Setup . . . 39
6.2 Baseline . . . 40
6.3 Model Selection . . . 40
6.4 In-Domain Performance . . . 46
6.5 Cross Domain Robustness . . . 49
6.6 Robustness Against Unrelated Data . . . 51
6.7 Across Corpus Performance . . . 53
6.8 Error Analysis . . . 54
6.9 Other Experiments . . . 58

7 Conclusion **61**
7.1 Future Work . . . 61

Bibliography **63**

A Appendix **71**
A.1 Corpus . . . 71
A.2 Implementation . . . 71

1 Introduction

To retrieve the basic bits of information from natural language texts, we need to accomplish a sequence of tasks to understand the structure and semantics of language. One such task is to label each word with the part-of-speech (POS), which is a prerequisite for many other tasks in natural language processing (NLP). In POS-tagging, we try to find the most plausible tag sequence $\mathbf{t}$, for a given word sequence $\mathbf{w}$. The tag sequence provides one tag for each word. In order to harvest contextual information, the tagger will, however, usually predict the tags for the whole word sequence combined.

For modern languages as English, the problem can be considered as mostly solved. However, historic languages such as Middle Lower German (MLG) can be more challenging. This is because the language does not have general spelling or punctuation rules that are obeyed by all writers. Koleva et al. (2017) report an inter annotator agreement (IAA) of around 91%, which means that predicting one specific word class for a MLG word is even difficult for human annotators.

The above-mentioned difficulties express themselves as uncertainty or noise in the data. In addition to this noise, we have other ambiguities introduced by the complexity of natural languages. Single words are often ambiguous and can only be interpreted by incorporating contextual information, like surrounding words. However, even for a trained expert, some sentences can be read in different interpretations, resulting in a set of possible POS-tags for single words.

Therefore, instead of forcing the classifier to predict exactly one tag we could allow it to return a set of labels. It is still possible to make single predictions in the form of singletons but it also allows for further flexibility. This approach is motivated by the idea of better abstaining from predicting in the case of uncertainty rather than enforcing it. Unlike classification with reject option, set-valued predictors can express their uncertainty more gradually, by returning larger or smaller sets instead of clear predictions or complete abstention. The framework of set-valued prediction allows the classifier to find a balance between small sets, containing only a few candidate classes, and correctness. Formally, there is a variety of frameworks that can achieve such tasks. One of them is conformal prediction (CP), which is rooted in frequentist statistics and is focused on predicting sets that contain the true class with high confidence. The other framework we will consider is called *set-valued prediction*, which is motivated more by the concept of utility of the prediction (Mortier et al., 2019).

Despite the potential, set-valued prediction hast not been combined with POS-tagging so far. That is mainly because part-of-speech tagging is often a prerequisite of many other NLP-tasks which would need to be capable of working on set-valued input data. However, machine learning tools are also often used as support for manual processes. In NLP-research, gold-standard tagged documents are very important. Having a tagger that can pre-label the document such that the human expert merely needs to validate the prediction is an ideal setting to investigate the potential for set-valued prediction in the realm of part-of-speech tagging.

First, the state-of-the-art of POS-tagging needs to be investigated thoroughly to understand which algorithms are used in this field. From this, we will derive an algorithm that is capable of providing set-valued targets. The algorithm is then tuned

to the corpus provided by the Intergramm research project. The performance of the resulting algorithm is evaluated against a baseline. Set-valued prediction shows promising results on both taggers.

1.1 Thesis Structure

In Chapter 2 we will cover the foundations of POS-tagging including several algorithms. Also, we will introduce methods to evaluate machine learning algorithms. In Chapter 3 an overview of the available literature related to this problem domain will be provided. In Chapter 4 the corpus used for the evaluation of our algorithm will be introduced. This chapter will, therefore, explain the domain in which the algorithm is applied. At the beginning of Chapter 5, we will formalize the problem of set-valued prediction for part of speech tagging. The remainder of this chapter will introduce the CoreNLP tagging algorithm and explain the extensions developed to use it for set-valued prediction. In Chapter 6 we will introduce a simple reference tagger and find an optimal configuration of the introduced algorithm. The remainder of this chapter will investigate the performance of the tagger in several experiments and examine its errors. Finally, the thesis results will be summarized in Chapter 7.

2 Prerequisites

First, we introduce the reader to part of speech tagging. In Section 2.2, we view the topic as a machine learning problem and investigate several algorithms. In Section 2.3, we introduce all measures and procedures needed to evaluate a machine learning algorithm.

2.1 Part-of-Speech Tagging

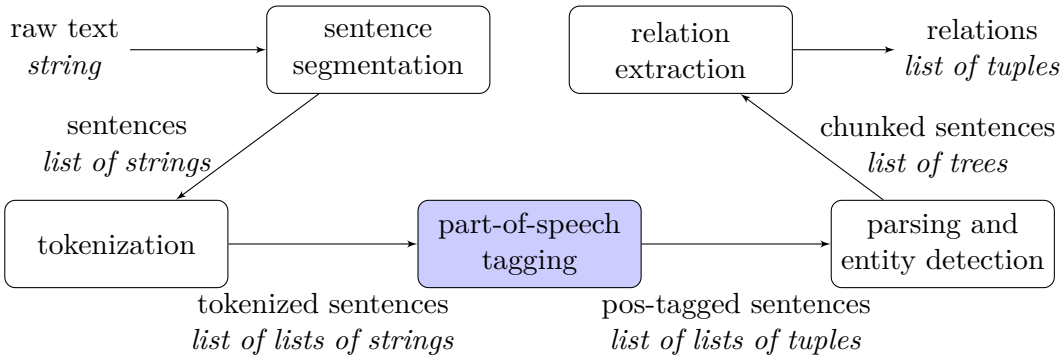


Fig. 2.1.: The NLP pipeline. The scope of this thesis is limited to part-of-speech tagging

The detection of word classes is a central task in most NLP-pipelines. Fig. 2.1 shows an exemplary pipeline for relation extraction, a process to mine structured data from a given corpus. As a prerequisite, raw textual data needs to be split into sentences and then into single tokens. Those tokens are then tagged with their part-of-speech. From the resulting tagged sentences, other tools can extract further grammatical structure and detect certain entities. Fig. 2.2 shows a small example sentence with extracted relation.

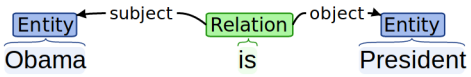


Fig. 2.2.: Example sentence with extracted relations using `https://corenlp.run`

A word class or POS is a grammatical role a word takes on in a given sentence. Examples are *noun*, *verb*, or *adjective*. Some words can only take on one POS. Others might fill in different grammatical roles. The problem of detecting the class is associated with a word inside a given sentence is called *disambiguation*, as sometimes more than one class is possible for a given word. Those word classes are usually called *tags* in the context of POS-tagging.

Each tag can be considered either an *open class* or a *closed class* tag. Closed class tags are tags that often take on structural roles in sentences. Among others, those are determiners like *articles* (the, a, an), *interrogative determiners* (which, ...), or *pronouns* (I, you, it). For those classes, the number of words is rather limited and fixed. When

we consider POS-tagging as a learning problem, we can assume that if our corpus is sufficiently large, we have seen all instances of each of the closed class tags. This means in conclusion that all words unknown to the tagger during inference fall in the category of open class tags.

Open class tags, on the other hand, have a lexical function and convey the meaning of the text. Examples of open class tags are nouns, verbs, and adjectives. Most of the words added to a language have lexical functions, as they express new inventions or ideas. Henceforth, those new words often belong to open classes, meaning that the tagger will likely encounter unknown words, no matter how large the corpus it is trained on.

Since part-of-speech tags express grammatical structure, some tags often appear in sequence. Other sequences, in turn, are very unlikely. In English, for example, the tag sequence $(article, adjective, noun)$ is a common grammatical structure ("The quick fox."). This indicates already that a good tagger will generally need to be able to learn not only from the words but also take into account the tags it is planning to assign to neighboring words. This means that a tagger, tagging only word by word, will most likely be limited in predictive performance compared to a tagger taking into account the entire sequence and hence exploiting contextual information.

Even though many languages are composed of the same grammatical building blocks, the tag sets are not completely static. Since the corpus of this thesis stems from a project interested in historical changes of MLG the tagset contains specific tags to describe a more fine-grained structure. The resolution of specific subclasses of tags is, however, nowhere near homogeneous. While nouns are generally only split up into a couple of classes, verbs are split into more than 20 subclasses for different inflections (finite, infinitive) or concerning their grammatical function (auxiliary verbs). Nouns are usually only subdivided into *common nouns* and *proper nouns* with respective singular and plural forms. The whole tag set is elaborated in Table 4.1.

2.2 POS-Tagging as a Machine Learning Problem

Tagging can formally be seen as the task to assign labels to single word tokens. Those tokens are part of a sentences and hence embedded in a context/sequence. This contextual nature of words in sentences has been vividly described by the famous quote

> "You shall know a word by the company it keeps"
>
> — John Rupert Firth, 1957

One can define a classifier that tags single words without the use of their context, but those capabilities are always limited.

In the following sections, we introduce various algorithms, to give an overview of the history and state-of-the-art of POS-tagging. In Section 2.2.4, we look in more detail at how to convert a word, its preceding and succeeding words into a representation that a machine learning classifier can use. One can think of different features we can extract from a word. E. g. we could use its *length* or binary features that check for a specific suffix. Such features can also be extracted from the surrounding words.

Many of the most potent taggers not only use the word and its surroundings but rather predict sequences of tags to the sequence of words. This offers the potential to not only rely on the data but also on partial predictions the classifier has done for the context.

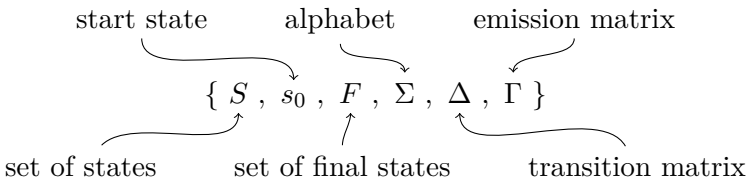
2.2.1 Hidden Markov Model

A hidden Markov model (HMM) is a generative model for tag probabilities in a token sequence. The goal is deriving the most likely tag-sequence $\hat{t}^n$ for a given token-sequence w^n

$$\hat{t}^n = \arg\max_{t^n} P(t^n \mid w^n). \tag{2.1}$$

In its simplest version, the model is a hidden Markov process (HMP) whose states represent tags that emit words. The word *emission probability* distribution in a given state is tuned during training. For example, we could imagine a state *proper noun* emitting proper noun words. However, the output set of states is not necessarily disjoint, as some words can take parts-of-speech depending on the context. The *transition probabilities* between the states are learned as well and express the probability of a tag succeeding another tag. One path from a start state to an end state will generate a sentence.

A Markov process is formally described by the following sixtuple:



The Markov process starts in $s_0 \in S$ and ends its traversal in one of the final states $F \subseteq S$. In our case, there will be only one final state. The process can emit words from the alphabet Σ whenever being in one of the states. The transitions between states are defined by a matrix T whose rows sum up to 1, forming a proper probability distribution over the out-edges. The rows of the emission matrix Γ specify probability distribution over the alphabet for each state.

To formally fix the minor issue that start and end state do not emit words, we can introduce the words $\langle init \rangle$ and $\langle end \rangle$ which will be emitted in the start and end state respectively with probability 1.

Formally an HMM is basically built on two assumptions:

1. The probability of a word is only determined by its tag

$$P(w^n \mid t^n) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

2. The probability of a tag is only determined by its predecessor (Markov assumption)

$$P(t^n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

When we use these simplifying assumptions on Eq. (2.1), we can derive the following:

$$\hat{t}^n = \arg\max_{t^n} P(t^n \mid w^n) \approx \arg\max_{t^n} \prod_{i=1}^{n} \underbrace{P(w_i \mid t_i)}_{emission} \underbrace{P(t_i \mid t_{i-1})}_{transition} \tag{2.2}$$

Deriving such a sequence would be trivial if each word could only be generated by exactly one state. In the next subsection, we introduce the Viterbi algorithm which is usually used to derive the prediction.

This rather simple model has seen many extensions over the years. One issue ignored here is e. g. the handling of *unknown words*. If a word is not known during training, its likelihood can also not be derived. Samuelsson (1993) proposed extension is to train another simple model on word suffixes estimating the posterior $P(t_i \mid w_i)$ which can be converted to the likelihood needed for the HMM with the help of Bayes rule[1]. This approach has been further refined by Brants (2000) which propose statistical smoothing between suffixes of different length. Another shortcoming of the simple model is the fact that the Markov assumption, that a tag depends only on its direct predecessor, is sometimes violated in natural language. According to Jurafsky and Martin (2008, p. 183) bigrams and trigrams of tags are often used to increase the sight distance within the sequence history. Lafferty et al. (2001) and Toutanova, Klein, et al. (2003) have indicated that considering only on previous words and tags is insufficient.

Estimating the probabilities of the HMP is fairly simple because they can be derived directly from the frequencies in the dataset. The emission probabilities of a word given a tag are

$$P(w_i \mid t_i) = \frac{\#(w_i, t_i)}{\#(t_i)}.$$

The transition probabilities between tag states are calculated by dividing the frequency of the two subsequent tags by the total frequency of the preceding tag:

$$P(t_i \mid t_{i-1}) = \frac{\#(t_i, t_{i-1})}{\#(t_{i-1})}$$

We refer for more details to Jurafsky and Martin (2008). They describe the training process in detail and list additional extensions to the ones proposed here.

Viterbi Algorithm

The goal of the inference algorithm is to find the most likely path through the HMP. The trivial solution to generate all paths and return the most likely one is, however, not tractable. Since the problem has an optimal substructure property, we can solve this problem optimally even though the number of potential paths grows exponentially in the sequence length.

The basic idea of the algorithm is to go through the sequence token by token, starting at the beginning. For each new token, we calculate the probability to reach that state from any of the previous states, weighted by the state distribution, multiplied by the probability in each state to generate the token (see Eq. (2.2)). In each step, we also save the most likely predecessor (backpointer) for each state. Finally, we can backtrack these backpointers and derive the complete tagging sequence.

Algorithm 1 is a recursive pseudo-code of the algorithm. The dynamic programming is achieved with the help of caching. Iterative implementation is, however, also possible. Line 7 of Algorithm 1 needs to completely re-evaluate the probability because the maximum depends on many local features. In the last lines, we are only doing a lookup.

To increase computational efficiency and numerical stability, the calculations are usually done in log-space, resulting in the summation of log-probabilities instead of the product of normal probabilities.

[1] Jurafsky and Martin, 2008, p. 193.

Algorithm 1 returns a tuple with the most likely state sequence and its likelihood given a sequence of tokens and a goal state. On initial call of the function, the goal state is $t_{\langle end \rangle}$.

procedure VITERBI($\mathbf{w}, t_{end}$)
  if $|\mathbf{w}| == 0$ **then**
    return $(t_{end}), P(w_1 \mid t_{end}) \cdot P(t_{end} \mid t_{\langle init \rangle})$
  if $t_{end} == t_{\langle end \rangle}$ **then**
    $t_n \leftarrow \arg\max_{t \in \mathcal{T}} \text{VITERBI}(w_{1,\dots,n-1}, t)_2 \cdot P(t_{end} \mid t)$ ▷ Backpointer
    $\mathbf{t} \leftarrow \text{VITERBI}(w_{1,\dots,n-1}, t_n)_1 || t_n$ ▷ Eq. (2.2)
    $P(\mathbf{t} \mid \mathbf{w}) \leftarrow \max_{t \in \mathcal{T}} \text{VITERBI}(w_{1,\dots,n-1}, t)_2 \cdot P(w_{n-1} \mid t_{end}) \cdot P(t_{end} \mid t)$
    return $\mathbf{t}, P(\mathbf{t} \mid \mathbf{w})$
  $t_n \leftarrow \arg\max_{t \in \mathcal{T}} \text{VITERBI}(\mathbf{w}, t)_2$ ▷ most likely end state
  $\mathbf{t} \leftarrow \text{VITERBI}(\mathbf{w}, t_n)_1 || t_n$ ▷ recalculate path from previous result
  $P(\mathbf{t} \mid \mathbf{w}) \leftarrow \text{VITERBI}(\mathbf{w}, t_n)_2$ ▷ recalculate likelihood
  return $\mathbf{t}, P(\mathbf{t} \mid \mathbf{w})$

For illustration, we apply Algorithm 1 to the example HMM in Fig. 2.3. Assuming we are given the sentence "the green fire truck" which we want to tag. The state transition matrix is given as

$$\Delta = \begin{pmatrix} 0 & 0.6 & 0.2 & 0.2 & 0 \\ 0 & 0 & 0.5 & 0.4 & 0.1 \\ 0 & 0.1 & 0.6 & 0.2 & 0.1 \\ 0 & 0.2 & 0.3 & 0.4 & 0.1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Now we can apply the algorithm recursively and derive the most likely path.

Fig. 2.4 visualizes the calculated backpointers. Here we can see again that fire could also be an adjective and therefore also has a backpointer. Since transition edges from the end-state and to the start state only make sense at the bounds of the sequence, they are omitted in the shown graph. In a real-life example, every node would have a backpointer. In our toy example, most paths in the graph are zero, because many of the emission probabilities are zero.

2.2.2 Maximum Entropy Markov Model

As we stated already, the predictive performance of HMMs is limited in some ways. Therefore, we investigate a related approach. Since an HMM is a generative model, the most likely tag sequence is predicted indirectly through the likelihood:

$$\begin{aligned} \hat{t}^n &= \arg\max_{t^n} P(t^n \mid w^n) = \arg\max_{t^n} P(w^n \mid t^n) P(t^n) \\ &= \arg\max_{t^n} \prod_{i=1}^{n} \underbrace{P(w_i \mid t_i)}_{emission} \prod_{i=1}^{n} \underbrace{P(t_i \mid t_{i-1})}_{transition} \end{aligned}$$

In the maximum entropy Markov model (MEMM), the posterior is directly predicted:

$$\hat{t}^n = \arg\max_{t^n} P(t^n \mid w^n) = \arg\max_{t^n} \prod_{i=1}^{n} P(t_i \mid w_i, t_{i-1})$$

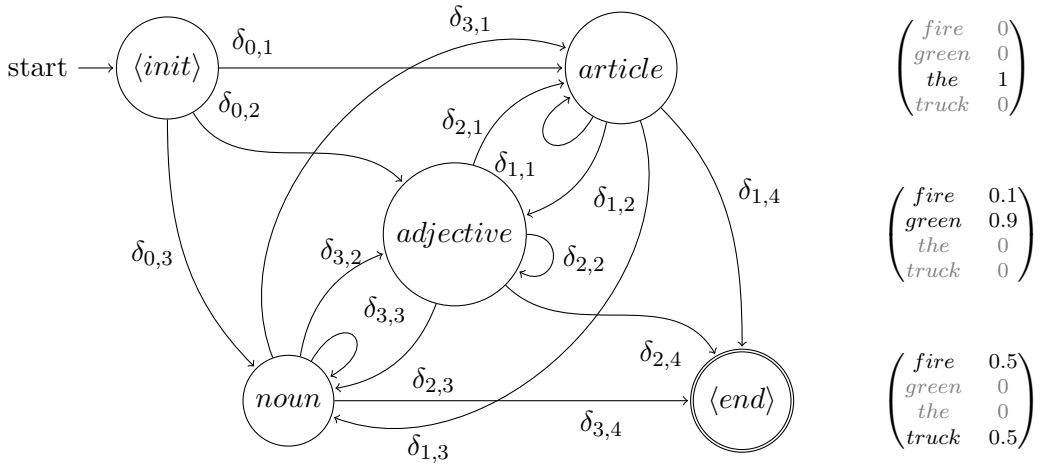


Fig. 2.3.: Visualization of an exemplary hidden Markov process used in an HMM, which might be used to tag the sentence "the green fire truck". The tables on the right side represent the emission probability distributions for each state. The Markov process can start and terminate in each state.

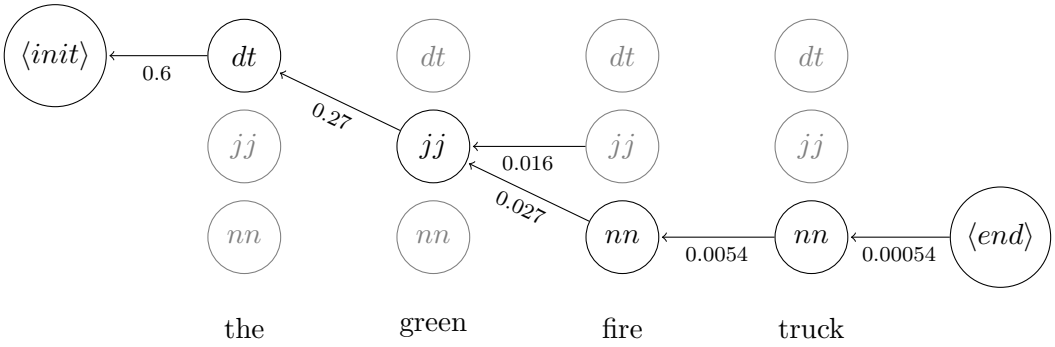


Fig. 2.4.: Visualization of the Viterbi path and the backpointers calculated. The predicted tag-sequence is therefore (dt, jj, nn, nn), where dt represents *article*, jj *adjective* and nn represents *noun*

The name *maximum entropy* is another term for the logistic regression classifier. However, the use of a discriminative model allows us to use many more features for the prediction. Usually, the source of a prediction is not limited to the current word and previous tag, but often consists of a whole sequence of tokens and tags (compare Fig. 2.5). In Section 2.2.4 we see in more detail what features can be used for the training of the classifier.

If we assume for now we have a classifier that can give us a posterior prediction given the features, we can reuse the Viterbi algorithm from the previous section. We only need to replace the probability estimates $P(w_{n-1} \mid t_i) \cdot P(t_i \mid t_{i-1})$ with the posterior $P(t_i \mid f_1, f_2, \ldots)$ where f_1 could be w_1 and f_2 may be t_{i-1}. This will again give us the Bayes optimal sequence given the posterior predictions.

McCallum et al. (2000) initially proposed MEMM using logistic regression as base-learner. Generally, any probabilistic classifier could be used to estimate the posteriors. For example the python library `sklearn` offers a version of an support vector machine (SVM) implementing a method proposed by Platt (1999) to get probabilistic predictions for that classifier.

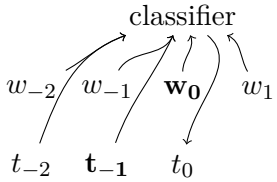


Fig. 2.5.: Example of features that could be used by a MEMM. The bold features are the ones an HMM could use.

As already mentioned in the previous section, deriving the predictions in a unidirectional way only considering the previous tags is not sufficient to grasp the whole complexity of natural languages. To this end, Toutanova, Klein, et al. (2003) have proposed an extension they call *bidirectional-dependency network*. This extension allows the predictor to condition on previous *and* succeeding tags. Their implementation—*CoreNLP*—is widely used in practice and is still maintained by the working group at Stanford University. We discuss CoreNLP in detail in Chapter 5 as our approach mainly relies on a patched version of CoreNLP.

2.2.3 Other Approaches

For the sake of completeness, we discuss some other approaches that have been tried in literature.

Koleva et al. (2017) use, among others, a memory based learner (MBL) to tag MLG texts. MBLs memorize the whole training dataset and compare new samples with the memorized ones, classifying by similarity to the memorized elements. An example of an MBL is k-nearest neighbor (KNN). KNN relies on a distance metric to compare two samples. Inference samples are then compared to all memorized samples. The k-closest elements with respect to the metric are then used and the maximum vote is used as a prediction.

Another approach, specifically tuned for sequence data, has been proposed by Lafferty et al. (2001). The sequentiality of natural language data makes conditional random fields (CRF) applicable in many tasks of NLP. Jurafsky and Martin (2008) state, however, that CRFs are computationally more expensive than MEMMs while providing no gain in predictive performance. Koleva et al. (2017) also use a CRF tagger which clearly outperforms the MBL. However, they do not use any other taggers.

Many of the latest published papers employ neural network-related methods for tagging. Deep neural network implementations like recurrent neural networks (RNNs) are showing promising performance. However, neural network models are computationally demanding in the fitting process. Additionally, those models need more training data to generalize well and to outperform simpler models like logistic regression. The large number of parameters allows the neural network to memorize a small training dataset, which will result in a large out-of-sample error. For an overview and a comparison of different neural network approaches in NLP, we refer to Yin et al. (2017, Table 1).

2.2.4 Feature Extraction

Discriminative machine learning models estimate their posteriors from features extracted from the data. Generally, a classifier tries to find a mapping between vectors of data

features to classes. When applying this idea to NLP we need to establish a way to "convert" textual data into vectors of features. This is achieved by *feature functions*.

Since textual data is sequential, those features are usually not only extracted from a single word or tag but also surrounding data. Feature extractors can generally come in different flavors. They can be *binary* or *categorical*, but also *ordinal* or *numerical*.

An example of a binary feature would be the Boolean value encoding whether a certain *word starts with a capital letter* or whether the current *word is at the beginning of a sentence*.

Categorical features could be the *suffix of the current word*. Since a word could, in general, have different suffixes of which only one is present in the word, one of a set of categories is true. But we cannot represent this category with only one scalar by simply enumerating the classes. If we did so, the tagger would assume that some suffixes are somewhat *closer* to each other, since their numbers are closer. To avoid this misconception we use a so called *one-hot encoding*. To this end, we map an element s_i from a set S of size $|S| = n$ to an n-dimensional vector whose i-th value is 1 and 0 otherwise (see Fig. 2.6).

$$s_2 \mapsto 2 \qquad \text{vs.} \qquad s_2 \mapsto \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline \end{array}$$
$$\{s_1 \;\; s_2 \;\; s_3 \;\; s_4\} = S$$

Fig. 2.6.: Comparison of numerical and one-hot encoding

Ordinal features are categorical features that can indeed be sorted according to an inherent order. Those features are therefore often expressed by scalar values, but they are rare in NLP.

Lastly, we consider numerical features. Those features can be related to discrete counting, but could also be real-valued. One common example from NLP is counting the number of characters of a word. For example, in German, large word lengths indicate compound nouns. Short words often take on structural roles, e. g. particles, and hence indicate that the word is an instance of a closed class tag.

All the previous examples considered only extracting information from single words in the vicinity of the currently tagged words or tags close by. In the realm of sequence data, it is often useful to not only look at single elements but extract features directly of small, fixed-sized sequences of tokens. In NLP, this is called shingling. In a generative model, for example, we cannot only count the relative frequency of a word but also succeeding pairs of words. Those pairs or longer sequences are called n-grams or sometimes also n-shingles. For short sequences, the Greek prefixed names *unigram*, *bigram*, and *trigram* are often used to refer to single tokens, pairs, or triples respectively. Those detected bigrams can then be seen again as categorical features and converted using a one-hot encoding. This can not only be done for tokens but also for the tags to allow the learner to learn common subtuples of tags.

With those ideas, we can easily devise hundreds of different features. However, adding huge amounts of features increases the size of the learning problem and hence often effects the accuracy negatively.

2.3 Evaluation

In this section, we go through several foundations for evaluating machine learning applications. First, we introduce measures to compare the predictive performance of classifiers. Later, we investigate some concepts to properly evaluate machine learning applications while minimizing the risk of overfitting and data-dredging. Afterward, we shortly talk about statistical testing and finally about the hyperparameter optimization.

2.3.1 Types of Classification Problems

The goal of an evaluation measure is to assign a score to the overall prediction of a classifier against the ground truth values. This number can then be used to compare algorithms with each other or give a general idea of the reliability of the predictions.

In Table 2.1, we can see several types of classifications. Binary classification problems are problems that only know two states, often *true* or *false*. One example could be a spam-filter responsible to detect whether a specific mail is spam or not. This concept can easily be generalized to more than one class. In a multi-class setting, there is only exactly one correct label. The tagging problem described above is such a multi-class problem, as every word can be labeled with one of the tags. In multi-label classification, we have several classes as in the multi-class setting. However, not only one label is correct, but a subset $Y \subseteq \mathcal{Y}$ of the labels is correct. An example of such a problem would be to assign a set of topic tags for a news article. However, we can also consider our tagging problem to be a multi-label task. The target label might contain uncertainty and thus all labels in question could be added to the ground truth. The tagger should then try to predict all of those labels.

Tab. 2.1.: Comparison of different classification problems

Type	Ground truth shape	Prediction shape
binary	one of two classes	one of two classes
multi-class	one of n classes	one of n classes
multi-label	subset of n classes	subset of n classes
restrained multi-class	one of n classes	subset of n classes
restrained multi-label	subset of n classes	subset of n classes

One problem that all previously mentioned concepts have in common is the fact that the classifier is forced to make a prediction, even though the entropy of the posterior is high. The idea of letting the classifier abstain, in case it can make no reliable prediction has been studied by Ramaswamy et al. (2018a). The idea of abstaining can be further generalized. Mortier et al. (2019) propose algorithms that allow the classifier not only to predict or reject, but also to predict sets that most likely contain the ground truth element. In case of high uncertainty, more elements can be added to the set. The extreme case of abstention can be reflected by predicting all classes. This concept is also sometimes called *prediction with partial abstention* or *set-valued prediction*. This concept can be further generalized to the multi-label case, where partial abstentions can be super-sets of the underlying ground truth set. The idea is that missing ground truth labels are more severe then predicting additional elements that are not part of the ground truth.

2.3.2 Measures

In the following section, we define the performance measures for the problems defined in the previous section. This is by no means a complete list, but we limit ourselves to the measures used in this thesis. All measures introduced here are bounded by the interval $[0, 1]$. The following definitions are always calculated for a single data element. When calculated over the whole dataset, the measures are usually averaged.

The *accuracy* measures fraction of correct classifications. It can be turned into the $0/1$-loss subtracting the accuracy from one:

$$\begin{aligned} \text{acc}(y, \hat{y}) &= [\![\hat{y} == y]\!] \\ 0/1\text{-loss}(y, \hat{y}) &= 1 - \text{acc}(y, \hat{y}) \end{aligned}$$

with $\hat{y}$ being the prediction and y the respective ground truth (Sokolova and Lapalme, 2009).

When generalizing the accuracy to the problem of set-valued prediction, we can create new utilities by combining the effect of correct classification and discounting the score for the redundant elements.

$$u(y, \hat{Y}) = \underbrace{[\![y \in \hat{Y}]\!]}_{score} \cdot \underbrace{g(|\hat{Y}|)}_{discount} \tag{2.3}$$

$g(\cdot)$ is the discount function. $\hat{Y}$ is the set-valued prediction. In case the prediction is only a singleton, the scoring term will reflect the accuracy and the discount will be 1 by the definition below. The scoring term can, therefore, be seen as a generalization for the accuracy.

Mortier et al. (2019) give an extensive overview of different discount functions. Since our algorithm relies on one of the proposed algorithms in the paper, we must also choose discount functions that satisfy the following properties:

1. $g(1) = 1$, the utility is not discounted if only the ground truth label is predicted.
2. $g(s)$ should be non-increasing
3. $g(s) \geq \frac{1}{s}$, i.e., the utility $u(y, \hat{Y})$ of predicting a set containing the true and $s - 1$ additional classes should not be lower than the expected utility of randomly guessing one of these s classes.

According to Mortier et al. (2019), the last property implements the idea of risk-aversion, where abstaining from a prediction (including more elements into the predicting set) should be favored against random guessing (predicting a randomly chosen element from the candidates).

The authors propose the following discount function:

$$g_{\alpha,\beta}(s) = 1 - \alpha \left(\frac{s - 1}{K - 1} \right)^{\beta} \tag{2.4}$$

K is the total number of classes, whereas $0 < \alpha < 1$ and $\beta \in]0, \infty[$ are parameters to tune the curve of the discount function. α is the penalty for abstention. Small values allow the classifier to add more elements to the prediction set, hence abstaining more. Large values force the classifier to select less elements. β selects whether the function is convex ($\beta < 1$) or concave ($\beta > 1$).

2.3.3 Cross-Validation

The goal of any machine learning classifier is to minimize the so-called out-of-sample error. This is the error, the classifier makes on previously unseen data. This is opposed to the in-sample error the error, the classifier makes on the dataset it is trained on. The notion *overfitting* describes in this context that the in-sample error is low and the out-of-sample error is high. This means the classifier memorized the training data but has not grasped the general concepts behind the data. For MBLs the in-sample error is mostly zero as all training instances are memorized and can be predicted perfectly. This is only violated if the training data contains conflicting data. The performance on the training data is usually significantly higher than on unseen data. To properly evaluate an algorithm, therefore, we need to split the data up into two parts. The algorithm is trained on the first, usually a larger portion. The evaluation is carried out on the second. This evaluation will give an estimate of the real out-of-sample error that can not be measured directly.

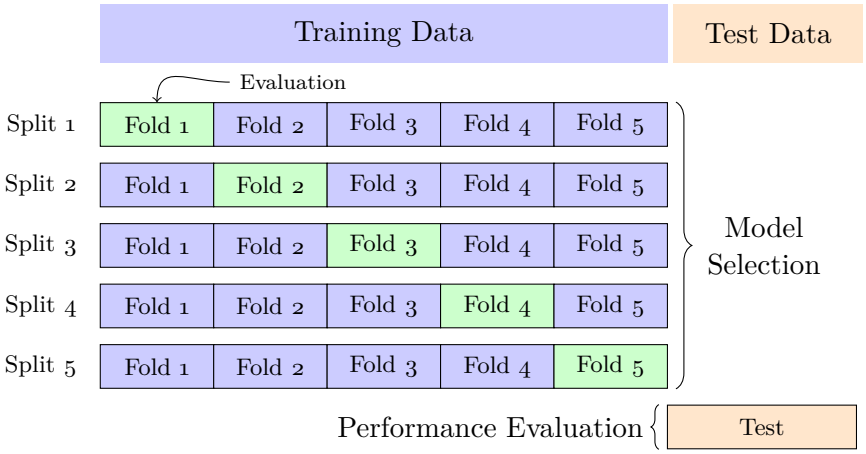


Fig. 2.7.: Splitting of data for cross-validation and performance evaluation. Training data is marked green, evaluation data blue, and testing data is marked in orange.

In Fig. 2.7, we see how the data is generally split for performance evaluation. First, we split-off a fraction of the data that is not involved in any training, the *test data*. The rest of the data, the *training data*, will be used for model selection. Since the classifier's parameters are selected with respect to the training data, it is no longer suited as a prediction for the out-of-sample error. Therefore, the evaluation needs to be done with the previously split-off test data (Arlot, Celisse, et al., 2010).

Fig. 2.7 visualizes the so-called 5-fold cross-validation. In this case, we split the data into 5 parts, use 4 of them for training and evaluate on the remaining instance. The results of the different evaluation data splits will than be used to estimate the performance of the model. In the next sections, we will go in more detail to different types of cross-validation. The notion of model selection is usually focused on the idea that we choose one of the different algorithms. However, hyperparameter optimization discussed later can also be seen as a form of model selection.

Different Methods for Implementing Cross-Validation

The generalization of the above mentioned 5-fold cross-validation is called *k-fold cross-validation* (Geisser, 1975). Theoretically, k could be any integer, but common values in literature are 3,5, and 10. An increase of k will usually also increase evaluated

performance, as the classifier can use $1 - \frac{1}{k}$ of the training data to learn from in each split. The score, derived from the cross-validation is usually the average score across all splits. In Section 2.3.5, we can see other usages of the sequence of scores instead of simple aggregation by mean.

Since we cut the data into pieces, we have to decide how exactly we want to select the cutting positions. Fig. 2.7 already indicates that we select equally sized pieces. However, we can still shuffle the data and select the position of the first cut. All succeeding cuts are then determined by the size of each piece. Independent of the actual choice, we must make sure that the resulting data splits are reproducible between different model selection runs. Otherwise, we evaluate different algorithms on slightly differing datasets and henceforth change two variables at once.

Since data is usually not perfectly uniform, there are many variants of this basic idea. Stratified versions of cross-validation make sure that all target classes occur in the same frequencies in all splits. Yet, this can only be applied if all data elements are statistically independent of each other.

One extreme case of the k-fold cross-validation is the *leave-one-out cross-validation*. In this version, we have n different splits with n being the number of data elements. The classifier is trained on the whole dataset except for one element and evaluated on this element. This form of cross-validation is computationally only feasible if the dataset is reasonably small and the training time of the classifier is small.

The cross-validation described so far assumes that the data has no structure and all individual data points are among equals. Sometimes the data decomposes into groups that should be taken into account when splitting the dataset.

A modification of the leave-one-out cross-validation that is aware of those groups is the *leave-one-group-out cross-validation*. Similarly, we train the data on all groups except for one and evaluate the performance on the remaining group.

In the context of statistical testing, we might want to evaluate an algorithm on more than k splits or choose the fraction used for training and the number of iterations independent of each other. This can be achieved by sampling the training fraction from the dataset (Picard and Cook, 1984). Therefore, we can use a fraction of size p from the dataset for training and by choosing another random split, we can create different combinations. This method is also called *monte-carlo cross-validation*.

For this thesis, we generally consider two basic ideas of splitting our data. The corpus is made up of several documents. The data is quite homogeneous within a document but heterogeneous between documents.

The first mode is in-document splitting, we split each document individually and then combine the splits across all documents. This mode can be used for k-fold cross-validation and Monte-Carlo cross-validation alike. Secondly, we can split the document boundaries. In this case, we can see the documents as groups of data and therefore this splitting method is similar to the leave-one-group-out splitting.

The idea of k-fold cross-validation or the Monte-Carlo version is that we can generally select each data element for itself. Unfortunately, our dataset consists of sequential data and cutting out single tokens destroys their important context. The sentences of such a document can, however, be seen as an independent entity, therefore we can shuffle the dataset before cutting. This could theoretically also be used to implement a constrained version of stratified cross-validation. However, the sentences in our corpus are generally very long and the machine learning programming libraries usually do not support such specific splitting. In later chapters, we will see that the dataset has however no proper sentences and hence, we can not rely on that notion for cross-validation. The lack of fine-grained sentences forces the splitter to make cuts within the document. This might

not be ideal, but since k is large compared to the size of the dataset, this issue can be ignored. In the case of Monte-Carlo cross-validation, only at most two cuts are placed within sentences.

The splitting between documents is easier to carry out. However, the documents in our corpus differ significantly in size, which results in significantly different amounts of training-data depending on which document has been selected for evaluation. Ideally, we would like to cluster the documents to generate groups of similar size, but it is not simple to do that in a principled manner.

2.3.4 Hyperparameter Optimization

To properly compare two algorithms, their parameters need to be optimized. Otherwise, there is no evidence to believe that we compare the best instances of the algorithms, which renders the evaluation meaningless.

Hyperparameter optimization is a search problem in the parameter space. Generally, every search technique can be used, such as handcrafted heuristics, simulated annealing, or evolutionary algorithms. In the last decade, Bayesian optimization techniques have been proven powerful for hyperparameter optimization (Hutter et al., 2011; Bergstra et al., 2011). Apart from that, the early stopping mechanisms are also very powerful, as they start on a wide search space with little computational resources and then invest more resources on most promising instances. An influential algorithm from Li et al. (2016) implements this idea in the form of a multi-armed bandit problem. This concept has also been combined with Bayesian optimization by Falkner et al. (2018). However, the fitting process of our algorithm can not be monitored or stopped, therefore we cannot employ those techniques.

Even though we could simulate some kind of early stopping, by fitting only a subset of the training data, this would be not very reasonable, as this yields sublinear speedups, which means we could as well fit the whole data and get a more reliable score.

Therefore, we opted for Sequential Model-based Algorithm Configuration (SMAC) version 3, which is available as a python library on `https://github.com/automl/SMAC3`. The full documentation can be found at `https://automl.github.io/SMAC3/master`).

2.3.5 Statistical Tests

In the previous sections, we discussed the idea of model selection. Model selection is a form of optimization whose goal is to find the best parameters that result in the best model after training. However, once we have stopped optimization, we never know if we found the optimum. Not only because we might get stuck in a local optimum, but also because the evaluation of the model has uncertainty in it. The measured performance might be over-estimated and other models we evaluated in the past have a lower out-of-sample error, but we do not know that. Since single changes in parameters might have tiny effects on performance, the overall result after a hyperparameter optimization might still be significant.

This can be verified with the statistical toolbox of hypothesis testing. The basic concept in hypothesis testing is that we assume the so-called *null-hypothesis*, which we aim to reject. The null hypothesis assumes that changes in results are only occurring due to random effects, but not because one of the algorithms is better than another one. When rejecting the null hypothesis with high probability, we can be sufficiently certain that one of the algorithms is better than the other one.

One such test, usable for pairwise comparison of two algorithms is the *Wilcoxon signed-rank test* (Sheskin, 2011). This test is used to check whether the median of a series of numbers deviates significantly from zero. In contrast to the paired Student's t-test, the Wilcoxon signed-rank test does not assume that the numbers are sampled from a normal distribution. This makes it more robust, as this assumption cannot be verified for algorithm scores.

The series of numbers are the pairwise differences of the scores of two algorithms when applied to a series of datasets. The source of such a series of datasets might be the Monte-Carlo cross-validation with a fixed random seed. The two algorithms are therefore applied to the same splits of data and we can compare each pair of scores.

Tab. 2.2.: Example performance scores and application of the Wilcoxon signed-rank test.

	A_1	A_2	$A_1 - A_2$	sorted score diffs	signed rank
Split 1	0.61	0.62	-0.01	-0.01	-1
Split 2	0.69	0.60	0.09	-0.02	-2
Split 3	0.83	0.80	0.03	0.03	3
Split 4	0.67	0.57	0.1	0.04	4
Split 5	0.72	0.77	-0.05	-0.05	-5
Split 6	0.81	0.77	0.04	0.06	6
Split 7	0.80	0.82	-0.02	0.07	7
Split 8	0.82	0.75	0.07	0.08	8
Split 9	0.84	0.78	0.06	0.09	9
Split 10	0.95	0.87	0.08	0.1	10
					39

In order to understand the process, we will calculate an example in Table 2.2. Assume we have two algorithms A_1 and A_2, which we would like to compare and test whether one of them performs statistically significantly better than the other. We carry out performance evaluations on 10 different cross-validation splits. Each run will result in one accuracy for each algorithm, estimated on the evaluation data of the split. Now, the score differences are calculated. This sequence of score differences is used as input for the test. If the test rejects the null hypothesis, the median deviates from zero and hence one of the algorithms performed better. The next step is to sort the differences by absolute value. The signs will be applied back to the rank each of the evaluations takes. Finally, we sum up all signed-ranks and receive the test statistic. If this statistic exceeds a critical value, we reject the null hypothesis and the sign of the statistic tells us which of the algorithms has performed better. The critical value depends on the length of the number sequence and the choice of the so-called p-value. The p-value is the probability of falsely rejecting the null hypothesis. This value is usually set to fixed values like $p = 0.05$ in literature. With $n = 10$ and $p = 0.05$, the critical value is 33. Since the calculated value of 39 exceeds the critical value, we can reject the null hypothesis and assume that the algorithm A_1 indeed performs better than A_2. However, with a probability of nearly 0.05, the deviation of the two performance numbers occurred by random chance and our deduction is wrong.

Apart from this simple example shown above, many specific details have to be taken into account, but we will omit them here for simplicity. One such issue, for example, is the handling of duplicate values or 0-values in the number-sequence. Statistical tests like this one are usually readily implemented in most programming libraries.

2.3.6 Pearson Correlation Coefficient

The Pearson correlation coefficient expresses relation between two statistical variables (Pearson, 1895). Let X and Y be two variables and their variance is denoted by σ, then the Pearson correlation is defined as follows:

$$\rho(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Let μ denote the mean of a variable and n the number of values, then the covariance (cov) is defined by:

$$\text{cov}(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y)}{n - 1}$$

$\rho(\cdot)$ is bounded by -1 and 1. Positive values indicate a correlation, negative values a inverse correlation. The absolute value of the measure indicates the strength of this correlation.

3 Related Work

The topic of this thesis is loosely connected to the demands of the Intergramm research project at Paderborn University. The goal of this interdisciplinary project based in linguistics and computer science is to research the language elaboration process of MLG (Seemann et al., 2017). To this end, the CorA(Bollmann et al., 2014) annotation tool has been extended for the needs of this project. The goals for this project demand that human annotator uncertainty and general ambiguity is captured in the corpus. This introduces additional challenges to a machine learning tagger as the tagger has to cope with this noise.

Over the years many approaches have been considered for POS-tagging. Schmid (1994) proposes a tagger inspired by HMM which uses decision trees as baselearner. This approach, initially tailored to English, was adapted to Germany by Schmid (1999). Echelmeyer et al. (2017) investigate the same model on Middle High German, however, it remains unclear why this tagger has been chosen, since it is no longer state-of-the-art. Ratnaparkhi (1996) suggested a similar method but relied on a logistic regression as the baselearner. Their work has been further refined by McCallum et al. (2000) by modeling state transitions differently. Lafferty et al. (2001) build upon that solving the issue they call *label bias problem*. The issue related to biased predictions in sequence models. The proposed algorithm is called CRF. Based on the work of Ratnaparkhi (1996), Toutanova and Manning (2000) refine MEMM models with a focus of unknown words. Toutanova, Klein, et al. (2003) extend this work by incorporate the findings of Lafferty et al. (2001). Their suggestion to solve the label bias problem is, however, computationally less expensive. Instead of calculating a global model for state transitions, they build a MEMM which does not only condition on previous tags, but also on succeeding tags. The resulting algorithm is the CoreNLP tagger, which is extended in this thesis. Unlike the previously mentioned referred literature, Brants (2000) shows, that plain HMMs are however similarly powerful when used with trigrams and suitable statistical smoothing. Shen et al. (2007) propose an entirely different approach using perceptrons. Their algorithm slightly outperforms CoreNLP. Collobert et al. (2011) investigate recent neural network approaches for POS-tagging and other NLP-tasks. Huang et al. (2015) incorporate the bidirectional nature of CRF with long short-term memory (LSTM).

The tagset used for the annotation corpus is strongly based on the HiTs tagset introduced by Dipper et al. (2013). This tagset is designed to handle the specific challenges faced in annotating a historic MLG corpus. Koleva et al. (2017) use a corpus that has many overlapping documents with the corpus used in this thesis. They investigate different kinds of part-of-speech taggers on the documents. The hyperparameters of the parametrized classifier used in their paper are optimized using evolutionary algorithms. Their taggers, however, have no capabilities of working on set-valued training data, neither can they produce predictions that reflect the uncertainty of the machine learning tagger.

The annotator's uncertainty can be seen as a form of noise, obscuring an idealized version of the data. The general problems faced with noisy data have been investigated by Derczynski et al. (2013). The authors use the tagger also used in this thesis. However, they tune the hyperparameters by hand and do not use hyperparameter optimization to

find optimal configurations for their dataset. The POS-tagger implementation used in this paper is the already mentioned CoreNLP tagger that is also used in this thesis.

In the past decades, different approaches have been proposed to make machine learning more reliable. One issue with machine learning is that the classifier is usually forced to give a prediction, irrespective of its confidence in this prediction. To tackle this issue, prediction with reject option has been proposed (C.-K. Chow, 1957; C. Chow, 1970). El-Yaniv and Wiener (2010) investigate the trade-off between classifier coverage and high accuracy in the context of binary classification. This concept has been generalized to the multi-class setting by Ramaswamy et al. (2018b). However, abstention might be too strict for many real-world applications, because it is a binary concept that cannot reflect different degrees of uncertainty. To this end, concepts like conformal prediction or set-valued prediction have been proposed. Conformal prediction, as introduced by Shafer and Vovk (2008), focuses on the statistical ideas of confidence, the probability that the true class is contained in the predicted set of classes. Mortier et al. (2019) focus more on a concept of utility. This concept allows us to specify the trade-off between large prediction sets and high confidence predictions in a more flexible way.

4 Corpus

In this chapter, we introduce the dataset used in this thesis. The dataset has been used to predict the out-of-sample performance of the algorithm, but the algorithm is also tuned specifically to this dataset.

4.1 Description

The corpus consists of several historic documents, written in MLG. Most of the documents are municipal law texts that have been read to the general public during trails (Tophinke, 2009). Tophinke (2012) express also that the boundaries of sentences are not as clear as they are today. We will therefore not be able to rely on sentence splitting. Since some tagger's memory complexity scale linear with the size of sentences and use sentence boundaries as guidance, this is difficult in this dataset. However, we implemented some kind of pseudo sentence splitting, which will be described in more detail later on.

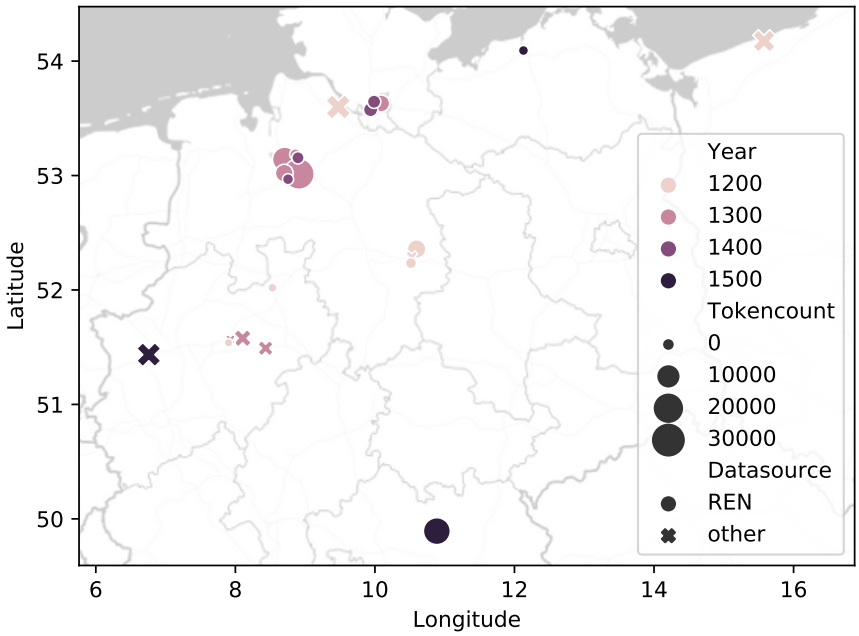


Fig. 4.1.: Scatter plot of document origin and metadata from the corpus. Small jittering has been added to the geographic positions to decrease the overlap of the datapoints.

Additionally MLG is not standardized during the time period of interest, meaning that there is no orthography. This issue was however already solved in the version of the dataset that has been provided for this thesis. The words have been clustered using a Levenshtein distance and the resulting spelling correction rules have been verified by language experts to prevent contamination of the dataset. Therefore we can assume all documents to be orthographically correct, even though this assumption can usually not

be made for MLG. The full list of all documents in the corpus can be found in Table A.1. This table also reports all shorthand for document names used throughout this thesis.

The corpus consists of a total of 23 documents from different time periods and of different sizes. In Fig. 4.1, we can see that the documents span a large region. The east-west extension ranges from Duisburg in the West to Kołobrzeg in the East. Kołobrzeg, now a Polish city, was part of Prussia in the 12th century. In the north, we have texts originating from Rostock and Hamburg. The separated southern text stems from Bamberg.

The earliest documents in the texts are from Kołobrzeg and Stade, written in the 12th century. The latest documents are from Duisburg, Bamberg, and Rostock from the 15th century.

The documents vary significantly in size. The smallest documents contain no more than 500 tokens (Hamburger Urkunden 1301–1350, Werler Urkunden Neheim, Ravensberger Urkunden) while the largest (Bremer Stadtrecht) contains nearly 24,000 tokens.

Fig. 4.2 shows the number of tokens per city and per time period. It shows clearly that a large majority of the documents stem from Bremen. The second most tokens come from the single text from Bamberg. The majority of the documents, namely the ones from Bremen, are written around 1300. The three from Bamberg, Rostock, and Duisburg make up the spike at 1500.

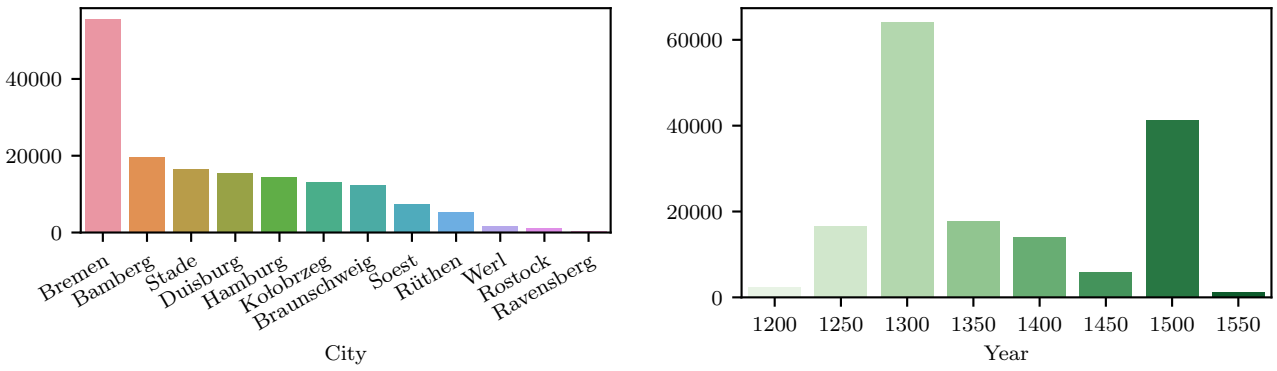


Fig. 4.2.: Distribution of the origin and creation time of the documents. The x-axis is the total number of tokens from documents in the respective category.

The corpus as a whole is taken from the research project InterGramm, which is creating a gold-standard tagged corpus in order to investigate the establishment of grammatical structures and sentences in MLG (Seemann et al., 2017). Part of their dataset, in turn, is taken from Reference Corpus Middle Low German/Low Rhenish (1200–1650) (REN)[1]. The documents from RENare tagged with a slightly different tagset but partially adapted to the tagging schema of the remaining dataset. This issue is visualized in Fig. 4.3. REN7, REN11, and REN12 are particularly small documents, hence it is expected that they only use a small subset of the tags. Generally, the documents which do not stem from the REN corpus have a larger variety of tags, because they are generally labeled more fine-grained. The tag with the largest variety across the dataset is the NE tag, expressing proper nouns. This figure gives also a first impression of the difficulty of the tagging task.

[1]ReN-Team, 2019.

The documents vary not only because of the difficulties that are inherent to diachronic corpora but also because they have been tagged with different annotation guidelines. Some of these ideas will be revisited in Section 6.9.2.

Since MLG differs significantly from modern languages and following the demands of such a specific research project, the set of tags is obviously different than the ones used mostly in NLP. The widely used Penn Treebank (PTB) II tag set[2] contains roughly 40 tags, but it contains also special tags for inter punctuation. In the corpus used here, the punctuation is of little interest and therefore automatically tagged by another, simpler tagger. The used tagset contains more than 90 tags, as seen on the x-axis labels of Fig. 4.3.

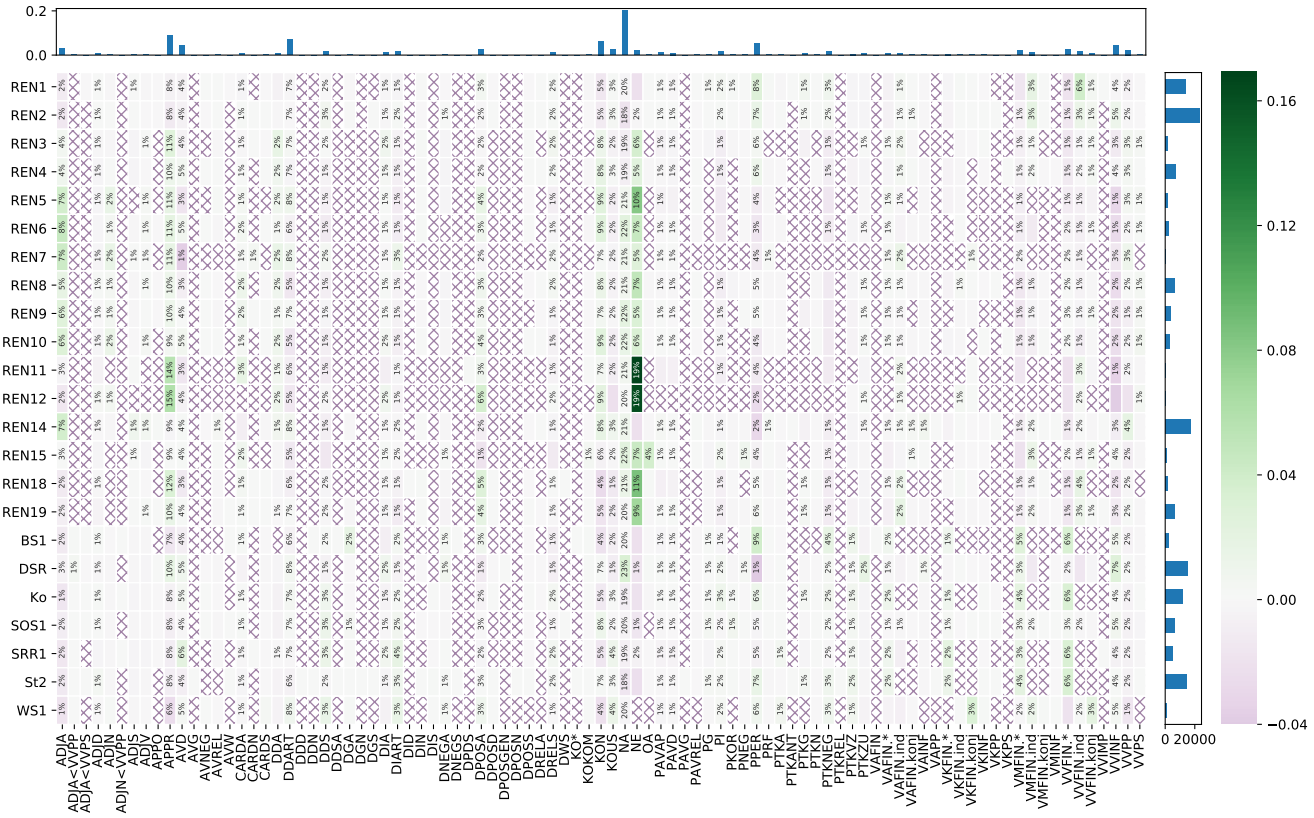


Fig. 4.3.: Heat map reporting the tag distribution of each document. If a tag has not been used in a document, the field is marked with a crosshatch. The top plot shows the prior distribution across the tags. The color in the heatmap marks the difference to the prior, while the numbers in the fields list the probability mass of the tag within each document. The plot on the right shows the relative size of the documents

Another important metric for NLP datasets is the so-called IAA. This measure is used to estimate the uncertainty in a dataset. To this end, two human annotators tag the same dataset and their overlap is calculated. Unfortunately, we do not have such a measure for the dataset we are using here. Therefore, we will report the agreement from Koleva et al. (2017). Their IAA is about 91,4% on the "Soester Schrae, 1300" (SOS1) and 92.2% on the "Statuarrecht Rüthen, 1300" (SRR1). The two documents are also part of our dataset. However, they annotated the documents with a less fine-grained tagset; therefore, we conjecture that such an agreement would be even smaller on our dataset. This number can be seen as an upper bound of our classifier's accuracy, as the

[2] Prasad et al., 2008.

corpus documents are annotated by several experts, all in slight disagreement with each other.

4.2 Tagset

The tagset used by the Intergramm project to tag this corpus is strongly based on the Dipper et al. (2013) tagset. Table 4.1 lists the whole list of tags. The first group of tags is concerning classical word classes like nouns, adjectives, and verbs. The last section consists of three special tags aimed to mark noise in the dataset. Tags like ADJA<VVPP express historic relation of words, for example, that this adjective historically stems from a simple past participle of a main verb. Tags ending with an asterisk like the KO* express groups of tags. In this case, the word would be one KOKOM, KON or KOUS, but a more general class is chosen by the annotator. Some of the documents contain latin words like "capitulum" for "chapter". Those words are tagged with the FM tag. The OA tag is used to mark spelling mistakes like accidental word duplication by the writer. Chapter numbers like "XII" are often tagged with the XY tag.

Tab. 4.1.: The tagset used in the corpus.

Word class	Related tags
Adjectives	ADJA, ADJA<VVPP, ADJA<VVPS, ADJD, ADJN, ADJN<VVPP, ADJS, ADJV
Pre- and postpositions	APPR, APPO
Adverbs	AVD, AVG AVNEG, AVREL, AVW
Cardinal numbers	CARDA, CARDN, CARDS
Determiners	DDA, DDART, DDD, DDN, DDS, DDSA, DGA, DGN, DGS, DIA, DIART, DID, DIN, DIS, DNEGA, DNEGS, DPDS, DPOSA, DPOSD, DPOSGEN, DPOSN, DPOSS, DRELA, DRELS, DWS
Conjunction	KO*, KOKOM, KON, KOUS
Noun	NA, NE
Pronominal adverb	PAVAP, PAVD, PAVG, PAVREL
Pronoun	PG, PI, PKOR, PNEG, PPER, PRF
Particle	PTKA, PTKANT, PTKG, PTKN, PTKNEG, PTKREL, PTKVZ, PTKZU
Auxiliary verb	VAFIN, VAFIN.*, VAFIN.ind, VAFIN.konj, VAINF, VAPP
Copular verb	VKFIN.*, VKFIN.ind, VKFIN.konj, VKINF, VKPP, VKPS
Modal verb	VMFIN.*, VMFIN.ind, VMFIN.konj, VMINF
Main verb	VVFIN.*,VVFIN.ind, VVFIN.konj, VVIMP, VVINF, VVPP, VVPS
Foreign language	FM
No annotation	OA
Not a word	XY

4.3 Multi-Label Tags

To allow the human annotators to express uncertainties in a more detailed way, the tagging schema is not a classical multi-class setting, forcing one tag per word. Each tagging is provided as a weighted multi-label set. The weights do not always sum up to 1 as there are generally three cases we distinguish:

Multiple Correct Tags If a word can be interpreted in two ways, the annotators can assign it two tags. E. g. the MLG word *dhe* could be classified with two different determiner tags $\{(\text{DDS}, 1.0), (\text{DRELS}, 1.0)\}$. Since all of them are correct, they are both weighted 1.

Partially Correct Tags If the annotators are uncertain about the tags they can assign multiple tags in a weighted manner. Those weights sum up to one and express a probability distribution. The human annotators can choose to assign the same probability to all tags or express that they deem one tag more likely. e. g.
so $\{(\text{PTKG}, 0.33), (\text{AVD}, 0.66)\}$
al $\{(\text{DIS}, 0.5), (\text{AVD}, 0.5)\}$

Single Targets If a word can be tagged by the annotator, only one tag is assigned. E. g. *braunschweig* $\{(\text{NE}, 1)\}$

The list above indicates, that no more than two tags can be provided. The weights are also discrete and can only take on the values listed above $(1/3, 1/2, 2/3, 1)$ Although most of the tags are actually single targets (Table 4.2).

Tab. 4.2.: Relative frequency of the multi-label target types in the dataset.

Target Type	Relative Frequency
(1.0)	99.09%
(0.5, 0.5)	0.42%
(0.33, 0.66)	0.28%
(1.0, 1.0)	0.20%

To tackle this peculiarity of the dataset, we will introduce modified performance measures in Section 5.3.

5 Algorithm

The following chapter will define the problem faced in this thesis and the algorithm and extensions used to solve it.

5.1 Problem Definition

In POS-tagging, we aim to predict the word classes for each word. Since the notion of word classes is strongly tied to the concept of a sentence, we assume the problem can be solved optimally on sentence level. Formally, we aim to predict a tag-sequence $\hat{t}^n$ given a words-sequence w^n. This word-sequence represents a sentence. Since historic languages like MLG do not have the notion of a sentence as we know it today, this word-sequence might be a large document consisting of hundreds of tokens.

The corpus used in our dataset is rather small and noisy. This noise is also reflected as uncertainty in the IAA. This uncertainty should be reflected in the predictions of the classifier, as the tagging of the classifier is only used as the first guidance for a human annotator to validate its predictions to generate a gold standard dataset. This capability can be expressed by allowing the classifier to abstain from its prediction for difficult data elements. A strict abstention is albeit not very useful for the human annotator working on the machine-generated tagging. To increase the classifier's expressiveness, it should be capable of predicting a set of classes, adding more in case it is unsure. To numerically grasp the value of such a prediction, we introduced several metrics in Section 2.3.2, specifically Eq. (2.3).

The Unrestricted Bayes-optimal prediction (UBOP) algorithm (Algorithm 5) defined by Mortier et al. (2019) can generally produce Bayes-optimal set-valued predictions with respect to the utility function, for any probabilistic classifier. This is, however, based on the assumption that each data element can be predicted on its own. This assumption is violated in sequence data as the prediction for one token should take predictions for tokens close by into account.

Another problem related to the assumptions of Mortier et al. (2019) and the reality of our dataset is related to the utility function proposed there. The authors propose their algorithm with multi-class problems in mind. However, as described in Section 5.3, the training data provided to the learner is a weighted form of multi-label data.

When considering set-valued prediction on sequences, there are generally two interpretations:

Token-level The tagger will produce a set of classes for each data element. Hence a prediction for a sequence of words is a *sequence of sets of tags*. When combining the sequence of sets with the Cartesian product, we can convert every token-level solution into a sentence-level solution.

Sentence-level Another interpretation would be to produce *sets of prediction sequences*. Each of the predicted sequences must provide a complete tagging for the word sequence. The sequences produced will however likely overlap significantly. Since a sentence-level solution produces sequences directly, they cannot always be factorized into token-level solutions and are therefore more expressive.

Tab. 5.1.: Comparison between token and sentence-level set-valued prediction

Token-level				Sentence-level
$(w_1,$	$w_2,$	$w_3)$		$(w_1,\ w_2,\ w_3)$
$\{t_1, t_2\}$	$\{t_2, t_3\}$	$\{t_4\}$		
induces:			vs.	predicts:
$(t_1,$	$t_2,$	$t_4)$		$(t_1,\ t_2,\ t_4)$
$(t_1,$	$t_3,$	$t_4)$		$(t_1,\ t_3,\ t_4)$
$(t_2,$	$t_2,$	$t_4)$		$(t_2,\ t_3,\ t_4)$
$(t_2,$	$t_3,$	$t_4)$		

The nature of those two views is visualized in Table 5.1. In this small example, we can see that the token-level predictor is not capable of creating the prediction that the sentence-level predictor generates. The token level tagger cannot express that (t_2, t_2, t_4) should be excluded from the prediction.

From a more conceptualized view, sentence-level set-valued prediction allows the tagger to predict different interpretations of the sentence. This would allow the tagger to express unsureness on the general interpretation of a sentence. However, the tagger can also express unsureness for a single word only, which would result in sequences that are identical except for a single position. The token-level prediction only allows the tagger to express this pointwise unsureness. To predict different interpretations of the sentence, the tagger will have to predict unsureness in multiple positions but has no possibility that only a subset of the Cartesian product are valid interpretations. In this case, the tagger would, therefore, generate many more solutions than it would like to express. Since this thesis proposes a way of set-valued prediction via a postprocessing step, the base classifier will be forced to focus on one specific interpretation of the sentence. This is, however, only a practical restriction. Generally, one could devise other algorithmic solutions being capable to solve the sentence-level form.

The problem structure we selected for this thesis is the *token-level* form. The main reason for this choice is the structure of the training corpus used. The modified version of CorA used in the Intergramm project allows the human annotators to assign multiple tags to a word. It does, however, not allow to define any additional restrictions. This means that the training data is in the structure of token-level sets. Apart from that, we assume that sentence-level set-predictions are much harder to achieve because neither MEMM nor any of the other approaches produces a posterior probability for whole sequences.

However, a first informal view on the data indicated that it is generally unlikely that a significant portion of the sentences have completely different interpretations. Most uncertainty is limited to single words. Since the corpus has no proper notion of sentences, the number of sentences is extremely small, which renders the training and evaluation of such an algorithm infeasible. Therefore, the less flexible approach was deemed much more reasonable.

5.2 Data Preparation

At the beginning of every data-related process is a cleaning phase to make the data learnable. In our case, this is mainly fixing tags and removing words that are of no interest to this evaluation, like punctuation. The result of this preparation process is

a dataset that can be passed to the tagging algorithm, which produces the set-valued prediction.

The dataset is provided as CorA export in XML format. This format is not orthographically corrected and contains much annotation specific information which is not needed for this project. Orthographic correction is, however, one important cleaning step, as the classifier will find many words it does not know. MLG does not have a unified spelling, especially not for our diachronic corpus. However, the Intergramm project is capable of reading the CorA XML files and convert them into orthographically unified, easy-to-parse text files. Koleva et al., 2017 show that applying spelling correction as a preprocessing step has no detrimental effects on the performance. They also show that the raw representation including transcription markers has little to no effect on tagging accuracy.

Modifications on Taggings With data cleaning, we are limited to fixing simple errors, that can be solved without extensive domain knowledge. In Fig. 4.3, we saw general inconsistencies of the dataset. This is out of scope for this cleaning phase of this thesis, since this can hardly be solved without expert knowledge. We fixed two types of errors in the cleaning phase: Untagged words and tokens that are some kind of punctuation.

The tagset contains the tag OA which is generally used to tag words that have no tagging because they are writing errors of some sort like words duplicated during writing. Therefore, this tag is suitable for tagging words that have no tag assigned.

Secondly, we tag all tokens that are punctuation of some kind with the $. tag. Those are tokens like:

, : ... .; ~ blankline / • ¶ ‡ | $

The word "blankline" for example is mostly an untagged word and is also clearly English despite the MLG corpus. In hyperparameter optimization, we allow the tagger to enable or disable punctuation symbols, however, they will never be part of the performance evaluation. Since these tokens are never ambiguous, they are easier to tag, which would distort the performance evaluation of the tagger.

Simplification and Fitting Most of the modifications needed for set-valued prediction will concern only the inference process. However, CoreNLP is not capable of handling weighted multi-label values, as we will describe further in Section 5.3. Therefore, we need to simplify the data to multi-class targets in order to fit it. This introduces some issues, as the loss the algorithm is optimized for is not congruent with the actual problem we would like to solve. As we will see in Chapter 6, this works quite well, nevertheless. To convert the weighted multi-label problem to a classical multi-class problem, we simply take one of the highest weighted tags as ground truth.

E. g.:
$$\{(\text{DDS}, 1), (\text{DRELS}, 1)\} \mapsto \text{DDS}$$
$$\{(\text{AVD}, 0.66), (\text{PTKG}, 0.33)\} \mapsto \text{AVD}$$

The concrete tag values have no influence in that selection. Another solution and further details of that issue will be discussed in Section 5.5.1.

Now we have the clean training data and know-how to fit the classifier. The derivation of set-valued taggings will be explained in Section 5.4 and Section 5.5.

5.3 Modified Performance Measures

As the machine learning problem tackled by this thesis is not a classical multi-class problem, we need modified versions of the above-mentioned measures. Concretely, ground truth for any particular word w is not a single tag t, but a weighted set of tags. However, more than 99% of the tags can be seen as classical multi-class targets. A complete description of the different cases of set-valued targets is introduced in Section 4.3 Generally, most of the targets in the training data will be singletons. However, we will devise measures that solve this issue principled.

The meaning of those tags will be further explained in Section 4.2. Since those two different settings are weighted sets, we cannot rely on typical measures for multi-label classification. Godbole and Sarawagi (2004, sec. 5.2) list several metrics that are generally considered in multi-label classification:

$$\text{multi-accuracy}(Y, \hat{Y}) = \frac{|Y \cap \hat{Y}|}{|Y \cup \hat{Y}|}$$

$$\text{multi-precision}(Y, \hat{Y}) = \frac{|Y \cap \hat{Y}|}{|\hat{Y}|}$$

$$\text{multi-recall}(Y, \hat{Y}) = \frac{|Y \cap \hat{Y}|}{|Y|}$$

Y is the target set and $\hat{Y}$ the predicted set as before. The multi-label accuracy is also called Jaccard index. The precision and recall are measures generally used in information retrieval and the realm of binary classification. The recall expresses how many of the actual true elements have been detected. The precision corresponds to the fraction of predicted positives that are true.

Most multi-label performance measures assume that the classifier aims to predict the ground truth. In the face of risk-aversion, this is not true though, since the classifier might add additional elements to the prediction to improve its recall. Considering the utility defined in Eq. (2.3), we want a scoring function that is not affected negatively by the expendable elements, since this is handled by the discount function. The multi-label accuracy and precision are therefore ill-suited as they discount the relative to the predicted set.

Before we carry out the modifications, we should notice that modifications on the scoring function might invalidate the proofs for optimality devised by Mortier et al. (2019).

As already seen in Eq. (2.3), the discount function should have no affect if there are no expandable elements. When considering single-valued prediction, the term will, therefore, be ignored.

$$\alpha(Y) = \begin{cases} 2 & \text{if } \sum_{y \in Y} s(y) > 1 \\ 1 & \text{else} \end{cases}$$

$$\text{ml-acc}(Y, \hat{Y}) = \left(\frac{\sum_{y \in Y \cap \hat{Y}} s(y)}{\sum_{y \in Y} s(y)} \right)^{\alpha(Y)}$$

Let $s(\cdot)$ be a function that derives the weight for a specific prediction. The $s(\cdot)$ is only defined for elements in the ground truth set, but the above equation only queries elements that are in Y. The exponentiation term only affects the results if the overall

score is less than 1, hence $\hat{Y}$ are the missing elements, and the target consists of multiple correct tags (see *multiple correct tags* enumeration above). The exponent 2, therefore, reduces the score if only one of multiple correct tags is predicted. This effect gets clearer when looking at the following examples:

$$\begin{aligned}
\text{ml-acc}\left(\{(\textsc{dds}, 1), (\textsc{drels}, 1)\}, \{\textsc{dds}\}\right) &= 0.25 \\
\text{ml-acc}\left(\{(\textsc{avd}, 0.5), (\textsc{dis}, 0.5)\}, \{\textsc{avd}\}\right) &= 0.5 \\
\text{ml-acc}\left(\{(\textsc{avd}, 0.66), (\textsc{ptkg}, 0.33)\}, \{\textsc{avd}\}\right) &= 0.66 \\
\text{ml-acc}\left(\{(\textsc{ne}, 1)\}, \{\textsc{ne}\}\right) &= 1
\end{aligned}$$

Before we can redefine the utility for our purpose, we need to modify the discount function as well. The current version of the discount function calculates the penalty concerning $|\hat{Y}|$. However, this is unfair, in case the ground truth also contains more than one element. This can be expressed in the following manner:

$$\text{ml-util}(Y, \hat{Y}) = \text{ml-acc}(Y, \hat{Y}) \cdot g(1 + |\hat{Y} \setminus Y|) \tag{5.1}$$

The discount is therefore not calculated with respect to the cardinality of the predicted set, but only by the cardinality of the redundant elements. The increment by 1 is needed to align the modification with the definition of $g(\cdot)$.

5.4 CoreNLP Tagger

In this section, we will introduce the CoreNLP tagger. It is used and extended to solve the problems defined in Section 5.1.

At first, it is important to explain the reason we selected this specific algorithm as a basis to solve the problem. In Chapter 2, we have seen different algorithms to solve the task of POS-tagging. Generally, we looked at different properties to select the most suitable tagger. The most important selection was the *performance we expected* to get out of any specific implementation. Therefore, well-established implementations should be favored against implementing from scratch. Since the early stages of the project indicated that the implementation needs to be modified to solve the tasks, all implementations considered should be open-source and reasonably documented.

The whole domain of neural network implementations has been deemed unsuitable for this thesis, because they tend to need fine hyperparameter tuning and rely on large amounts of training data.

For these reasons, we aimed at more mature implementations:

CoreNLP This library is developed and still maintained by the natural language processing workgroup at Stanford University. It is widely used and the implementation is available on `github` and was known to the author before starting this thesis. CoreNLP implements an extended version of a MEMM. It offers high predictive performance and is specifically tuned for English, but not limited to this language.

CRF++ This library has been discovered during the later phases of the thesis during the study of the paper by Koleva et al. (2017). It implements the CRF approach. According to Jurafsky and Martin (2008), CRFs are computationally more demanding than simple MEMM approaches, but do not offer additional performance in the specific domain of POS-tagging.

Finally, CoreNLP has been chosen for several reasons: It is widely used, mature and available as an open-source implementation.

5.4.1 Algorithm

In Algorithm 2, we see the high-level structure of the CoreNLP inference algorithm. Line 4 of the algorithm calls the classifier fitted on the training corpus. It will be capable of estimating the probability of a word given its word and context. The window defines the contextual tags; however, the classifier also depends on contextual words. The tag context is calculated bidirectionally by this algorithm we will now explain in more detail.

Algorithm 2 Coarse-grained algorithm of CoreNLP, L, R be the left and right viewing offset feature extractor

1: **procedure** INFERENCE($\mathbf{w}$)
2: **for** $w_i \in \mathbf{w}$ **do** ▷ for every position
3: **for** $window_{i,j} \in \prod_{k=L}^{R} valid\_tags(w_{i+k})$ **do** ▷ for every tag context
4: $windowposterior_{i,j} \leftarrow P(T|window_j, \mathbf{w}, i)$ ▷ Emission probability
5: **for** $i \in \{1, ..., |\mathbf{w}|-1\}$ **do**
6: **for** $window_{i,j} \in windows_i$ **do**
7: **for** $window_{i+1,j'} \in windows_{i+1}$ **do**
8: $transition_{i,j,j'} \leftarrow \begin{cases} 1 & \text{if window consistent} \\ 0 & \text{otherwise} \end{cases}$ ▷ layered graph
9: **return** VITERBI(G) ▷ return optimal Viterbi path

The window sizes L and R are determined by the hyperparameters selected. Those feature extractors will be further explained in Section 5.4.2. If the feature extractors look at more preceding or succeeding tags, the algorithmic complexity increases. This can be seen in the Cartesian product in line 3 of the algorithm.

The function

$$valid\_tags : w \to \mathcal{P}(T)$$

calculates a constrained set of tags to be considered for each word. It is needed to reduce the amount of considered tags to a reasonable set to keep the Cartesian product in line 3 as small as possible.

$$valid\_tags(w) = \begin{cases} \text{tags, the word occurred with} & frequency(w) \geq threshold \\ \text{open class tags} & \text{otherwise} \end{cases}$$

This constraint is built on the assumption that words are either only tagged by closed-class tags or only open class tags. The idea behind closed class tags is that every word for this tag can be seen in the training corpus. On the inverse, we can, therefore, assume that for each word, if it is tagged with closed class tags, it appears with all its tags in the training corpus. However, if a word only seldom appears with a specific tag, the tagger might not have seen it with this specific tag during training. Therefore, these constraints might sometimes force the tagger to make misclassifications. If a word is not known, all open class tags are considered for tagging. This is also based on the assumption described before, because the tagger assumes that it knows all unknown words that belong to open classes. The Cartesian product of the constraint tags for a viewing window with $L = -2$ and $R = 1$ can be visualized as follows:

$$\begin{array}{cccc} t_1 & t_2 & \sqcup & t_1 \\ t_4 & & & t_2 \end{array} \quad \to \quad \begin{array}{cccc} t_1 & t_2 & \sqcup & t_1 \\ t_1 & t_2 & \sqcup & t_2 \\ t_4 & t_2 & \sqcup & t_1 \\ t_4 & t_2 & \sqcup & t_2 \end{array}$$

The tagset of this corpus consists of nearly a hundred tags and many of them belong to open class tags. This means that for a sequence of unknown words, the function *valid_tags* allows many tags at each of the positions. This results in a particularly large number of context windows considered. Therefore, the size of the viewing window depends more or less directly on the number of open class tags. Let n be the cardinality of the open class tags and l the width of the viewing window. Then the Cartesian product in a position with l unknown words will result in n^l viewing windows.

The graph on which we apply the Viterbi algorithm is different than the examples we have seen in Fig. 2.3. An example graph of how it is generated inside CoreNLP can be seen in Fig. 5.1. For a sequence of words, we can derive the valid tags as described above.

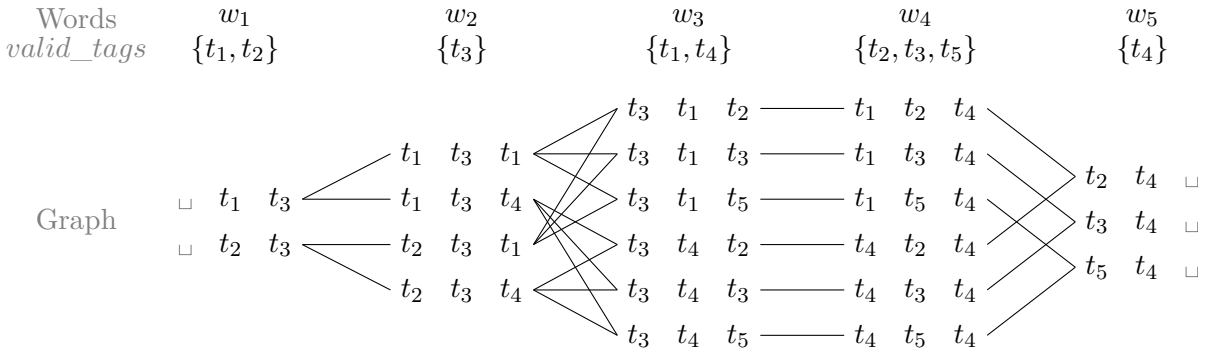


Fig. 5.1.: Example graph on which CoreNLP calculates the optimal tagging. Each node in the graph is a tag context for which the classifier estimates a probability. The Viterbi algorithm is used to find the globally most likely path.

Those tags induce the tagging context windows for every position in the sequence. The edges in the graph are binary, unlike the HMM. Two windows are only connected if their contexts are consistent. *Consistency* in this case is such that the overlapping part of the two windows in question is identical. Formally such a context window is a sequence of tags. Two sequences are consistent if the suffix of length $n-1$ of the first sequence is identical to the prefix of length $n-1$ of the following sequence. Let σ be the first sequence and ρ the succeeding sequence and σ_i be the i-th element of the sequence and n their length.

$$\sigma \text{ and } \rho \text{ are consistent if and only if } \forall 1 \leq i \leq n : \sigma_i = \rho_{i-1}$$

Algorithm 1 is used to calculate the optimal path. Apart from this basic structure, CoreNLP has additional hyperparameters to tune the frequency threshold for $valid\_tags(\cdot)$.

5.4.2 Feature extractors

In Section 2.2.4 we have seen what can generally be used to convert textual features into vector data. CoreNLP allows a large set of feature extractors. From the performance point of view they split up into two sets: Feature extractors that incorporate tag information and extractors that only use the textual data. This is the case because the runtime is mainly defined by the size of the viewing window used in line 3 of Algorithm 2. This means that we can add any feature extractor that works only on the textual data. However, when selecting too many feature extractors that incorporate tag information, the runtime and memory demands can grow intensively. An overview of those feature extractors can be found in Table 5.2.

Tab. 5.2.: Selection of CoreNLP feature extractors

	Effects tag window	Description
words	✗	One feature for each unigram in the given range
biwords	✗	One feature for each bigram in the given range
lowercasewords	✗	One feature for each lowercased unigram in the given range
prefix/suffix	✗	One feature each prefix/suffix of length n at the given relative position
tags	✓	One feature for each tag in the given range
twoTags	✓	One feature for each consecutive pair of tag in the given range
order	✓	One combined feature for all tags in the given range

5.5 Extension

In this section, we will describe all modifications done to the CoreNLP implementation and wrappers around the algorithm. The methods provided allow minimal modifications on the CoreNLP algorithm while providing a solution for the problem specified in Section 5.1. Therefore, the approach can be used with any probabilistic classifier and is not tied to CoreNLP specifically. In fact, during the early phases of this thesis, different classifiers have been used within this framework. In Section 6.2 another probabilistic classifier is introduced, which can directly work within this framework.

First, we discuss preparations to modify our training data, such that it can be used by CoreNLP. Then, the modifications to the implementation are presented. Finally, we will see how to provide a set-valued prediction for the given data.

5.5.1 Set-valued Loss and Data Augmentation

In Section 5.3, we have seen that the training data provided is not classical multi-class data. In Section 5.2, we have however already seen a simple solution for this issue. The following paragraphs are therefore focusing on two aspects: First, we explain why the simplification of the training data has been chosen instead of modifying the loss the tagger is trying to fit. Secondly, we will propose another solution to the data simplification that provides the classifier more data about the ground truth.

Since more than 99% of the tags are classical multi-class tags, a modification of the loss function is unlikely to have a large effect on the overall accuracy. Our data-augmentation approach gives the classifier generally access to all the data and therefore the multi-label case is encoded in the loss in some way. Because this method needs no modification on the base-classifier, it can still be applied to other data-sets easily.

The basic idea behind our approach is to provide the classifier with multiple instances for sentences if a sentence happens to have any kind of multi-label ground-truth. Assume we have a sentence $\mathbf{w}$ with the tagging sequence $\mathbf{t}$. If this tagging sequence is only a sequence of singletons, we can interpret it as proper multi-class data and pass it to the classifier as it is. Otherwise we could generate all sequences of tags $\mathbf{T}$ for the sentence $\mathbf{w}$ that can be generated by the Cartesian product over $\mathbf{t}$. The classifier would then be provided with each combination of $\mathbf{w}$ and a $\mathbf{t}' \in \mathbf{T}$. However, since we have no sentence boundaries, even with a small probability of multi-label tags, $\mathbf{T}$ will most likely be

intractable for at least one sentence. To this end, we devised an augmentation method that generates less duplicate sentences formalized in Algorithm 3. This algorithm makes

Algorithm 3 Let $\mathbf{t}$ be a sequence of weighted sets. It returns a set of sequences $\mathbf{T}$

```
1: procedure AugmentSentences(t)
2:     for j ∈ {1, max_{t∈t} |t|} do          ▷ as many sequences as the sets have elements
3:         for i ∈ |t| do
4:             k ← max(1, |t_i| − j)          ▷ take any element, but don't exceed set
5:             t'_i ← t_{i,k}
6:         T_j ← t'
7:     return T
```

sure that each tag is shown to the classifier at least once. However, not all contextual constellations are shown to the tagger. More precisely, Algorithm 3 will return only one or two sequences, since the maximal size of a multi-label tag is 2 (see Section 4.3).

5.5.2 Set-valued Prediction in General

As we described in Section 5.1, we will focus on the token-level interpretation of set-valued prediction in this thesis. Ideally, we would like to modify CoreNLP to produce set-valued prediction directly. To tackle that, we would need to make modifications to the training loss and in the inference process. For the time being, we ignore the training and focus on the inference. We saw that CoreNLP uses a graph of context windows to derive the optimal tagging sequence. This yields two potential entry points for the implementation: Either we make the context windows set-valued or the set-valued prediction is somehow generated during the inference on the graph. However, trivial solutions have large computational complexity and solving the problem heuristically is hard to evaluate.

As an example of the computational complexity of the naive solution, we will shortly look at one example. Let us assume we plan to implement set-valued prediction by substituting the context windows by windows with set-valued tag contexts. The number

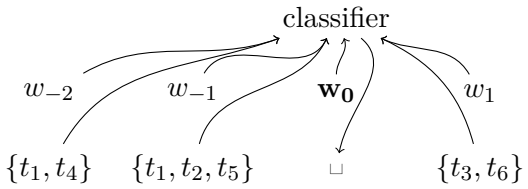


Fig. 5.2.: Visualization of a set-valued context window in CoreNLP

of considered windows for CoreNLP at a specific position in the sentence is the Cartesian product of the valid tags for each position in the window. With set-valued prediction, the complexity increases further, because we need to consider a Cartesian product of powersets of valid tags. That is because the need to consider all sets to stay in alignment with the consistency idea in the connection between windows for the HMM graph. To execute this, we would then need to find a way to score set-valued context windows. This could be either done by combining scores for single-valued context windows that make up the set-valued windows of some sort. However, we found no principled solution to solve the fundamental problem of the increased computational complexity.

5.5.3 Modifications on the CoreNLP Implementation

To apply set-valued prediction in a post-processing step, we need to be able to export much of the internal information. The implementation has therefore been modified in two ways: First, we implemented that CoreNLP stores all posteriors for each of the finally selected context-windows; secondly, we implemented a command-line interface, to be able to relax the constraints, set by the valid tags function.

Early analysis showed that the selection of valid tags will sometimes force the classifier to make mistakes, as the correct tag is not contained in the valid tags. The classifier could be provided with deterministic rules R which can be applied to a set of tags T and generate a relaxed set $T' \supseteq T$ (see Algorithm 4). The selection of the ruleset R can, therefore, be seen as a hyperparameter. T' will, therefore, allow more tags as valid and therefore increase the search space considered by the Viterbi algorithm. More in-depth explanation and experiments will be shown in Section 6.9.3.

Algorithm 4 Let T be set of valid tags before the deterministic expansion. R be a set of tagsets representing the expansion rules. The expanded tagset T' is returned.

procedure ExpandValidTags(T, R)
  $T' \leftarrow T$
  for $r \in R$ **do**
    if $T \cap r \neq \emptyset$ **then**
      $T' \leftarrow T' \cup r$
  return T'

5.5.4 Set-valued Prediction via Postprocessing

The basic idea behind this method is that the tagger can give us a distribution over the tags for each word. This can be done either by predicting tags context-free, like the baseline tagger in Section 6.2, or by taking tag context into account, as the CoreNLP tagger does. Algorithm 5 describes the algorithm used for predicting set, given a posterior and a discount function. The algorithm has been introduced by Mortier et al. (2019).

Algorithm 5 (UBOP) – **input:** $g(\cdot)$, context $\boldsymbol{x}$, set of classes $\mathcal{Y} = \{c_1, \ldots, c_K\}$, and posterior probability P

$\hat{Y} \leftarrow \emptyset$, $p_{\hat{Y}} \leftarrow 0$, $U^* \leftarrow 0$ ▷ Init. best solution, its probability and utility
$\mathcal{Q} \leftarrow \emptyset$ ▷ Init. priority queue of classes sorted by decreasing $P(c \mid \boldsymbol{x})$
for $c \in \mathcal{Y}$ **do** ▷ For all classes
  $\mathcal{Q}$.add($(c, P(c \mid \boldsymbol{x}))$) ▷ Query the distribution P to get $P(c \mid \boldsymbol{x})$
while $\mathcal{Q} \neq \emptyset$ **do** ▷ Loop until the list of sorted classes is empty
  $(c, p_c) \leftarrow \mathcal{Q}$.pop() ▷ Pop the most probable element from $\mathcal{Q}$
  $\hat{Y} \leftarrow \hat{Y} \cup \{c\}$, $p_{\hat{Y}} \leftarrow p_{\hat{Y}} + p_c$ ▷ Update the current solution and its probability
  $U_{\hat{Y}} \leftarrow p_c \times g(|\hat{Y}|)$
  if $U^* \leq U_{\hat{Y}}$ **then** ▷ If the utility increased
    $\hat{Y}_u^* \leftarrow \hat{Y}$, $U^* \leftarrow U_{\hat{Y}}$ ▷ Replace the current best solution
  else ▷ If there is no improvement
    break
return $\hat{Y}_u^*$ ▷ Return the set of classes with the highest utility

Since the post-processor does not rely on the predicted tags, but only on the posterior, it can completely override the predictions made by CoreNLP or any other tagger. When viewing the contextual nature of the problem, this could be a problem, as the results from the Viterbi algorithm can be revoked. However, the posterior distribution also reflects the chosen Viterbi path, as the final path determines the context for each posterior distribution.

To validate whether the post-processor will revoke the tagger's predictions, we calculated how often the single tag predicted by the tagger is contained in the set-prediction at the specific position. For known words, this was the case in every single position. For unknown words, more than 99.5% of the single tags were contained in their respective set-prediction.

When we compare our approach to the two views on the problem of set-valued prediction on sequence data (see Section 5.1), we can see that this approach is very well in alignment with the token-level setting. That is, predicting one interpretation for the sentence, and then relaxing the prediction locally for every position of the sequence.

6 Evaluation

In the previous chapter, we introduced all the work done to solve the research problem defined in Section 5.1. In this chapter, we aim to evaluate the provided algorithm on the corpus described in Chapter 4. First, we select the optimal model in Section 6.3. Then, we evaluate this model in different ways.

6.1 Dataset Splitting and Experiment Setup

To allow for a clean performance evaluation, we will split the dataset in the beginning into a test and a training part. The test part will not be involved in the process of model-selection and can, therefore, provide a clean performance estimate. The training part is used in cross-validation to find the optimal model during hyperparameter optimization. Fig. 6.1 visualizes the data splitting process. The corpus decomposes into multiple documents. The test part is composed of 20% of each document. The test part is cut out on a random position. Since the documents are not chunked into sentences, there is no way to shuffle data without destroying the word context.

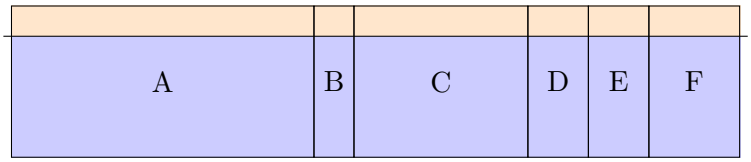


Fig. 6.1.: The training and test data is split across all documents. The blue fraction is used for model-selection, while the orange part is held back for performance evaluation. The fraction is taken from random sections of each document.

For Section 6.5, the evaluation is less obvious, as we will set the splits differently. Since the performance of this setting is anyway hardly comparable with the other two settings, it makes no sense to exclude the test set for training. The goal of this setting is to evaluate the robustness of the classifier.

Different performance measures are used for model selection and evaluation. Due to the nature of this dataset, the classifier itself is optimized concerning the $0/1$-loss. However, for hyperparameter optimization, we can use a measure that is more suitable for our corpus. Therefore, we use the multi-label ground-truth accuracy (ml-acc) devised in Section 5.3 for this means. Since the topic of this thesis is set-valued prediction, we are not so much interested in the accuracy of our classifier, but rather in the utility it achieves. But it is not obvious for which utility we would like to optimize. As already indicated in Section 2.3.2, there are many choices for the discount function $g(\cdot)$ and many of them can be parameterized even further. Optimizing for one of these measure instances does not indicate good performance for other measures, as the utility highly depends on the structure of the posterior probability distributions of the classifier. Therefore, we opted to optimize the model with respect to the ml-acc measure via 5-fold cross-validation.

The different evaluation settings we investigate are hardly comparable to each other. Especially for small datasets like this, the performance is significantly influenced by the amount of training data considered (see Section 6.8.3). Therefore, we use the baseline algorithm defined in the following section as a relative performance indicator. This tagger will always be trained and evaluated on the same parts as the classifier we are interested in. For the evaluation, we use the ml-acc and ml-util measures. The utility will be calculated for different parameters with the discount function defined in Eq. (2.4). If nothing is specified, ml-utilwill be configured with the discount function $g_{\alpha=1,\beta=1}(\cdot)$. Similarly to the work of Derczynski et al. (2013) and Toutanova, Klein, et al. (2003) we will analyze the performance of known and unknown words in the error analysis. We would like to remind the reader again that Koleva et al. (2017) indicated that the inter-annotator agreement on their MLG documents is only about 91%. Since the documents in the dataset stem from different sources, they have been annotated by several different annotators. Therefore, it is unlikely that a tagger can learn a performance above this annotator agreement when trained and evaluated across the whole corpus. The an explanation of the abbreviations for the dataset documents is provided in Table A.1.

Hardware Setup All experiments where run on an Arch Linux (kernel 5.3.8 x64) machine with an Intel 8700k hexa-core CPU and 32GiB of RAM DDR4-2166. Detail on the programming language and libraries used can be found in Appendix A.2.

6.2 Baseline

The baseline model explained in this section is even simpler than an HMM. This model will predict a tag with the highest probability and depend solely on the word itself. In case the word is known, the predictor is as follows:

$$t_i = \arg\max_{t \in \mathcal{T}} P(t \mid w_i)$$

This posterior probability is estimated by the relative word frequency:

$$P(t \mid w) = \frac{\#(t, w)}{\#(w)}$$

If the word is unknown, we will fallback on predicting the globally most frequent tag via the prior:

$$t_i = \arg\max_{t \in \mathcal{T}} P(t)$$

In our corpus this is the NA tag which represents a common noun. In alignment with CoreNLP, this tagger cannot predict tags that are not seen in the training data.

Since this tagger is very simple, it is well suited for lower bounding the performance of any tagger for a specific corpus. It will, therefore, take on the function of indicating the difficulty of the dataset itself.

6.3 Model Selection

The goal of the model selection is to find a configuration of the tagging algorithm, that is fit to the data as good as possible. Therefore, we use the within document splitting across the whole dataset. This setting is described further in Section 6.7. The resulting model is kept fixed for all evaluation settings in this chapter.

6.3.1 Configuration Space

Before we define the configuration space considered for model selection we would like to mention that we had to constrain the configuration space strictly. The theoretical configuration space for CoreNLP is not only very large, but it contains also many configurations that are not practically useful. As discussed in Section 5.4.1 large tag window sizes will result in extraordinary run time during evaluation or will exceed main memory. Apart from that, many configurations will crash for other reasons or run extremely long, slowing down the search process.

Therefore, we chose to only consider a small subspace for the feature extractors. We also split up the model selection into two parts: First, we optimized the feature extractor using the default parameters for all other hyperparameters. Secondly, we fixed the optimal feature extractor and optimized the remaining hyperparameters. This constraint search might exclude the optimal model of the combined search space. Although the results of the second optimization step indicate that their effect is generally minor anyway and the feature extractor selected with default parameters might be close enough to the optimum. All parameters not mentioned here have been kept at their default setting. Accidentally, the parameter *lang* has been set to *german* instead of the default value *english* during the evaluation. However, Section 6.3.3 shows that this parameter has no effect on the final model. Therefore we assume that it would not have influenced the model selection in the first place.

The configuration space for the feature extraction is defined in Table 6.1. The default model for CoreNLP is specified by *words(-1,1)* and *order(-2,0)*; therefore, it is natural that we include it. The *words* feature extracts one feature for each word in the specified range relative to the current word. We added larger frames to give the classifier a larger word context. We allowed the classifier to also select prefixes of length 1,2, and 3 for rare words. The selection for suffixes considered is guided by the special rules created for a expert system of the Intergramm project. We selected the configuration space such that the classifier could theoretically recreate the inference rules crafted by the human experts.

The second list in Table 6.1 concerns the tag extractors. Since the width of the tag context window significantly influences the classifiers computational complexity, we extracted the offset into a variable and specified all tag related features relative to this offset. The maximum tag window size that can be kept in main memory is 3. The *order* feature has been kept fixed in size, but might be moved such that one tag in front and one tag behind the current word is considered. The *wordTag* feature extracts a specific rule for the current word combined with the tag at the specified index. In the default configuration it is disabled completely. This selection is again modified by expert rules from the Intergramm project. The full list of architectural parameters is available at the projects website[1][2].

The last section in Table 6.1 lists parameters of our wrapper algorithm. Since punctuation very easy to tag, we choose to exclude it from the evaluation of our classifier. However, also the unregulated punctuation of MLG could convey contextual information that can be exploited by the tagger. Therefore, we allowed the model selection to enable or disable punctuation. Apart from that we allowed the hyperoptimizer to enable or

[1] https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/ExtractorFrames.html

[2] https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/ExtractorFramesRare.html

Tab. 6.1.: Configuration space for the feature extraction architecture. The blue marked elements are the default values from which the search was started.

Parameter name	Description	Values
words	Interval of words to consider for tagging	$\{(-1,1),(-2,2),(-3,2)\}$
prefix	One feature for each prefix length extracted from the current word.	$\{\{\},\{1,2,3\}\}$
suffix	One feature for each suffix length extracted from the current word.	$\{\{\},\{2\},\{1,2,3\},\{2,3,4,5\}\}$
x	Offset tag related features	$\{-2,-1\}$
order	One combined feature for the tags of the provided index range.	$\{(x,2+x)\}$
wordTag	One feature for a word and tag. First parameter is the word index, second parameter is the tag index. The default value is $\emptyset$.	$\text{pow}\left(\{(0,x+i) \mid i \in \{0,1,2\}\}\right)$
punctuation	Exclude punctuation words. They do not contribute to the score evaluation.	$\{True, False\}$
augment_data	Enable process explained in Section 5.5.1	$\{True, False\}$

disable the data augmentation approach introduced in Section 5.5.1. All in all, this search space contains $3 \times 2 \times 4 \times 2 \times 1 \times 2^3 \times 2 \times 2 = 1536$ configurations.

The second phase of model selection is concerning classical numerical hyperparameters. The first parameter listed in Table 6.2 influences the regularization of the underlying optimizer for the logistic regression model inside of CoreNLP. The other parameters manipulate the feature-selection and pruning. The extractor frames specified above will be generally used as templates to add features. However, those frames will create thousands of singular features, from which many are only triggered once or twice during the whole training process. The parameters below will decide when which features can be dropped, to reduce the dimensionality of the optimization surface the logistic regression has to fit. The last parameter will add additional features not mentioned before. Those features will detect extremely common words. This configuration space is theoretically of infinite size, as it contains real valued parameters.

Apart from that, we could have specified open and closed class tags. However, the assumptions for open- and closed-class tags are violated in our dataset, as some of the closed-class words are not seen in the training dataset as our training set is only small, compared to other natural language datasets. The effect on classifiers performance will be further evaluated in Section 6.3.3. The other command line parameters for CoreNLP should not effect the fitting and inference process. The full list of the command line parameters for CoreNLP can be found on the project website[3]

6.3.2 Resulting Configuration

Since the CoreNLP training and inference algorithm is a black box evaluation, we can not interrupt and resume this process. Therefore, we can not use bandit like hyperoptimizers

[3] https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/tagger/maxent/MaxentTagger.html

Tab. 6.2.: Configuration space for the feature extraction architecture.

Parameter name	Description	Range : Default
sigmaSquared	Regularization parameter for model fitting	$(0.01, 10) : 0.5$
curWordMinFeatureThreshold	Above this threshold, feature extractors for this word are generated	$(1, 5) : 2$
minFeatureThreshold	Features below this threshold are marked for pruning	$(1, 20) : 5$
rareWordThresh	Below this threshold, rare word feature extractors for this word are generated	$(1, 100) : 5$
rareWordMinFeatureThresh	Rare word features below this threshold are marked for pruning	$(1, 100) : 10$
veryCommonWordThresh	Words more frequent than this threshold, get a extra detecting feature	$(10, 500) : 250$

like Hyperband[4] and BOHB[5]. Those type of hyperoptimizers evaluate the algorithm with a specific time budget and use an intermediate score to decide which configurations they want to evaluate further. In deep learning applications, for example, the training process takes very long and the validation is usually done on intermediate models that are further trained. The fitting of a logistic regression is very fast in comparison. Because of the complexity of our algorithm, most of the evaluation time is spent in the inference of the sentences. In deep learning, fitting and evaluation are interweaved while we execute this processes in sequence. That is, because evaluation is expected to be cheap in comparison to fitting; therefore, we can execute full performance evaluations multiple times within the fitting process.

To this end we selected smac3[6] HPO for hyperoptimization. The algorithm combines Bayesian optimization with random forest classifiers to approximate the parameter space. Smac has been run in the sequential form as the model evaluation with 5-fold cross-validation was sufficiently parallelized on the 6-core machine.

The first configuration space has been searched starting from the default configuration with a budget of up to 14h and 100 configuration evaluations. The 100 configurations where finished after 8h 36 minutes. The resulting cofiguration is as follows:

words	$(-3, 2)$	order	$(-2, 0)$
prefix	$\{1, 2, 3\}$	wordTag	$(0, -1)$
suffix	$\{2, 3, 4, 5\}$	punctuation	True
		augment_data	False

The optimizer selected the largest word context and based its tag context on the two preceding tags. It seems that the *wordTag* feature is only useful when combining the preceding tag with the current word. Punctuation seems to have a positive effect on this configuration, but maybe this is also related to the large word context window, allowing the classifier to look past the punctuation as well. Surprisingly, the data augmentation has detrimental effects on this model. Early experiments on the default configuration indicated that the data augmentation is helpful. For more detail on these two parameters refer to Section 6.3.3.

[4] Li et al., 2016.
[5] Falkner et al., 2018.
[6] Lindauer et al., 2017.

The second half of the parameter space has been optimized with the same budget as before. Since the intermediate model is larger than CoreNLP's default configuration, the average configuration in the parameter space takes probably longer to evaluate. The optimization runtime for this part takes about 11h 23 minutes summing to a total hyperoptimization time of 20 hours. The resulting parameters are:

sigmaSquared	0.768	rareWordThresh	6
curWordMinFeatureThreshold	4	rareWordMinFeatureThresh	1
minFeatureThreshold	1	veryCommonWordThresh	234

Interestingly, the pruning parameters *minFeatureThreshold* and *rareWordMinFeatureThresh* were set to their minimal reasonable value of 1, therefore, disabling pruning. The *curWordMinFeatureThreshold* value has been doubled compared to the default and and thus these features will be added only for more frequent words. The rest of the parameters are fairly close to their default setting.

6.3.3 Model Analysis

In this section, we will analyze statistical significance of the performance improvement and analyze the robustness of the selected model with respect to the punctuation and data augmentation parameters as well as the selection for open and closed classes.

CoreNLP provides some predefined feature extraction architectures. For our analysis, we select the two default models *left3words* and *left5words*. Their feature extraction architectures are defined by "words (-1,1); order (-2,0)" and "words (-2,2); order (-2,0)". All other parameters are set to the defaults defined in Table 6.1 and Table 6.2. To validate the improvement, we rerun each of the models with a 15-fold Monte Carlo cross-validation with 80% training data and 20% validation. The resulting accuracy scores have been tested pairwise for statistical significance with the Wilcoxon signed-rank test. All evaluations are performed on the specific dataset at hand and might turn out differently for other datasets or other splittings of this dataset.

Performance Improvement

After the optimization, we end up with two default models and two optimized models, we call *opt intermediate* and *opt final*, respectively. Fig. 6.2 reports the average accuracy

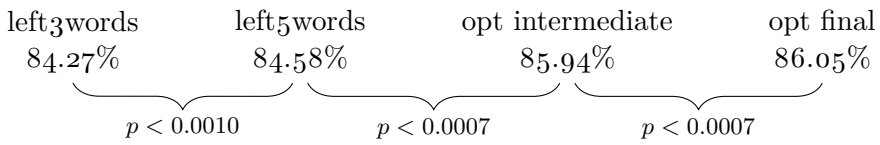


Fig. 6.2.: Performance comparison of the two default models and the two results of our optimization runs

of each model and shows that all model differences are indeed statistically significant. The probability of a falsely rejected null-hypothesis is less than 0.1%.

Punctuation

As we stated in Section 5.2, the punctuation in our dataset is not standardized. Hence, dots and other symbols might also be inside the sentence and might not even have a

proper meaning. This means that punctuation can potentially destroy word contexts by splitting them apart. On the other hand, punctuation also indicates structure and can therefore contribute positively to the classifiers performance.

The hyperoptimization yielded a model which enabled punctuation. However, we deemed this was related to the large word context windows of this model. To this end, we evaluated the effect on punctuation on the simpler default model *left3words* and the optimal model. Fig. 6.3 shows that specifically the simple model profits from punctuation. The effect is statistically significant according to the Wilcoxon signed-rank test. Surprisingly, the parameter has no effect on the performance of the optimized model.

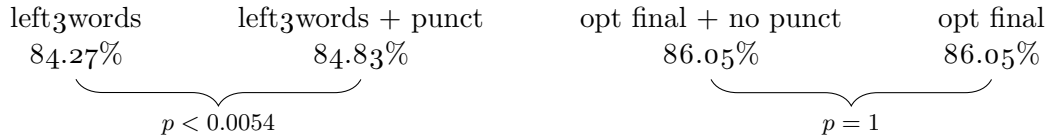


Fig. 6.3.: Performance effect of punctuation analyzed on the simpler default model and the final optimization model

Data Augmentation

Additionally, we checked whether data augmentation has a positive effect on the accuracy. Fig. 6.4 shows that the data augmentation has a negative effect on the models performance. We stated in Table 4.2 that only 0.2% of the data is actually multi-labeled. The way the augmentation is done might add conflicting information to the dataset, as even weighted targets as $\{(\text{AVD}, 0.66), (\text{PTKG}, 0.33)\}$ are added twice to the dataset and the classifier might think the PTKG tag is equally valid as the adverb tag AVD. Therefore, the data augmentation effects the accuracy negatively with statistical significance.

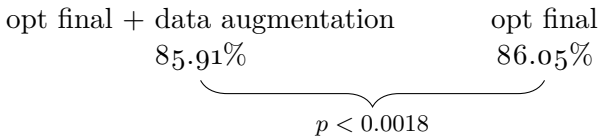


Fig. 6.4.: Performance effect of data augmentation analyzed on the final optimization model

Open and Closed Tags

CoreNLP knows three ways to specify the set of open and closed tags. First, you can set a language which will select a predefined set of closed class tags (parameter *lang*). Second, the closed class tags can be learned from the corpus (parameters *learnClosedClassTags* and *closedClassTagThreshold*). Third, we can specify the set of open or closed class tags directly.

The selection of open and closed class tags selects the valid tags constraints that influence the search space of the Viterbi algorithm. Selecting the wrong segmentation can therefore incur significant performance losses as the algorithm is not allowed to evaluate context windows that might lie on the optimal path.

During hyperparameter optimization, the *lang* parameter was accidentally set to *german*, so we would like to find out if the default parameter *english* would have yielded a better model. Fig. 6.5 shows, that the resulting models for those parameters

are actually more or less identical and neither of the two can statistically significant outperform the other.

As a next step, we set the open class tags to the linguistically correct value for this tagset. This means we will set the open class tags to contain all verbs (V*), nouns (N*), adjectives (ADJ*), numbers (CARD*), and all special error classes (OA, FM, XY). All remaining tags will be consequently interpreted as closed class tags. The CoreNLP algorithm is built on the assumption that all closed class words are seen during training data and therefore, unknown words can only fall into the category of open class tags. This assumption is too strong for our dataset and hence influences the performance negatively. This loss is statistically significant according to the Wilcoxon signed-rank test.

Finally, we investigated if learning the segmentation could improve the performance. However, the performance was again identical to the *german* end *english* setting of the *lang* parameter.

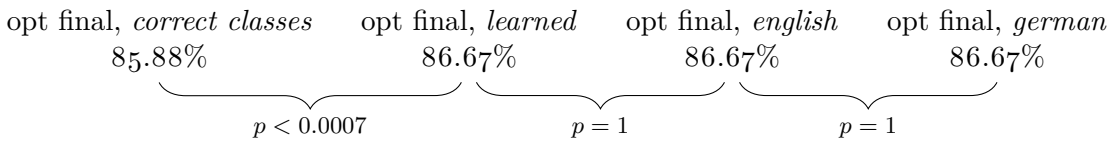


Fig. 6.5.: Performance effect of the open and closed class segmentation analyzed on the final optimization model

6.4 In-Domain Performance

As discussed earlier, the documents of the corpus are not homogeneous. Therefore, we assume that a classifier that is trained only on one specific document will likely perform well on that dataset (see Fig. 6.6). In some way, this can be seen as an upper bound performance, because of the similarity assumption of the data. However, this will also

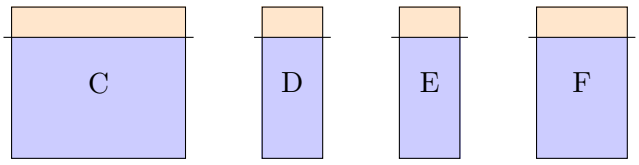


Fig. 6.6.: Evaluation of in-domain performance. The classifier is trained on the training part of one document and evaluated on the test part of the same document.

limit the training data, which means that estimates of very small datasets might be less reliable. Since our corpus is quite large, we limited this evaluation to documents with at least 7.000 tokens. The comparison of the sizes of the different documents can be seen in Fig. 6.7

We will maintain the train and test splits used for model selection; however, we will constraint each experiment to one of the documents. The performance is evaluated relative to the baseline classifier.

The plots in Fig. 6.7 show that the classifier outperforms the baseline always with a clear margin. This margin is generally larger for the accuracy. This might be due to the fact that the accuracy is generally the more distinctive measure or that the utility is more forgiving. Since the postprocessor is smoothing out the uncertainties of

the classifier, as large sets are less penalized than misclassifications. When comparing document size and the performance in Fig. 6.7, we can see that the accuracy hardly correlates with the amount of training data. The REN2 document, which is the "Bremer Stadtrecht, 1300" is the largest in the corpus and still, the classifier has difficulties to fit it. The relation between tagger performance and training data size will be further evaluated in Section 6.8.3.

When comparing the performance of the classifier on the "Soester Schrae, 1300"(SOS1) to the results of Koleva et al. (2017) we can see that their version and tagging scheme of the document must have been significantly different. They achieve a baseline performance of 80.90% when training and evaluating this document with the same size of training and evaluation parts. Our baseline performance is only 75.17%. This is also the document where CoreNLP has the least relative gain to the baseline classifier. Overall, CoreNLP achieves most of the time accuracies above 80%, while the baseline performs mostly 5-10 percent points worse.

The utility is generally much higher than the accuracy and also closes the gap between the two classifiers. The utility measure is usually more than 10 percentage points higher for the baseline tagger. The gain for CoreNLP is usually larger, but also very significant. However, the utility does not express much when ignoring the average set size. As hinted already, most of the baseline classifier's gains are achieved by adding many elements into the predicted set. The set size for this tagger is usually around 3 or 4. CoreNLP on the other hand often achieves set sizes between 2 and 2.5, hence reflecting the confidence for single class classifications. Table 6.3 lists all results for this set of experiments.

Tab. 6.3.: Results of the in-domain experiment. The performance of the main classifier and the baseline are compared across 9 different datasets with the performance measures accuracy, utility and average set size

		DSR	KO	REN1	REN14	REN19	REN2	REN4	SOS1	ST2
ml-acc	CoreNLP	88.21	86.92	82.89	88.28	86.78	83.13	79.78	77.64	88.43
	baseline	77.71	79.73	77.11	79.55	80.23	73.68	75.41	75.17	80.27
ml-util	CoreNLP	92.82	96.34	92.77	95.62	94.35	93.45	90.95	91.84	95.95
	baseline	91.09	94.76	91.01	92.33	93.84	89.18	88.05	90.84	94.90
$\lvert\hat{Y}\rvert$	CoreNLP	1.79	2.18	2.41	2.01	2.12	2.66	3.04	2.74	2.01
	baseline	4.61	3.66	4.31	4.08	3.05	4.63	5.18	4.73	3.44

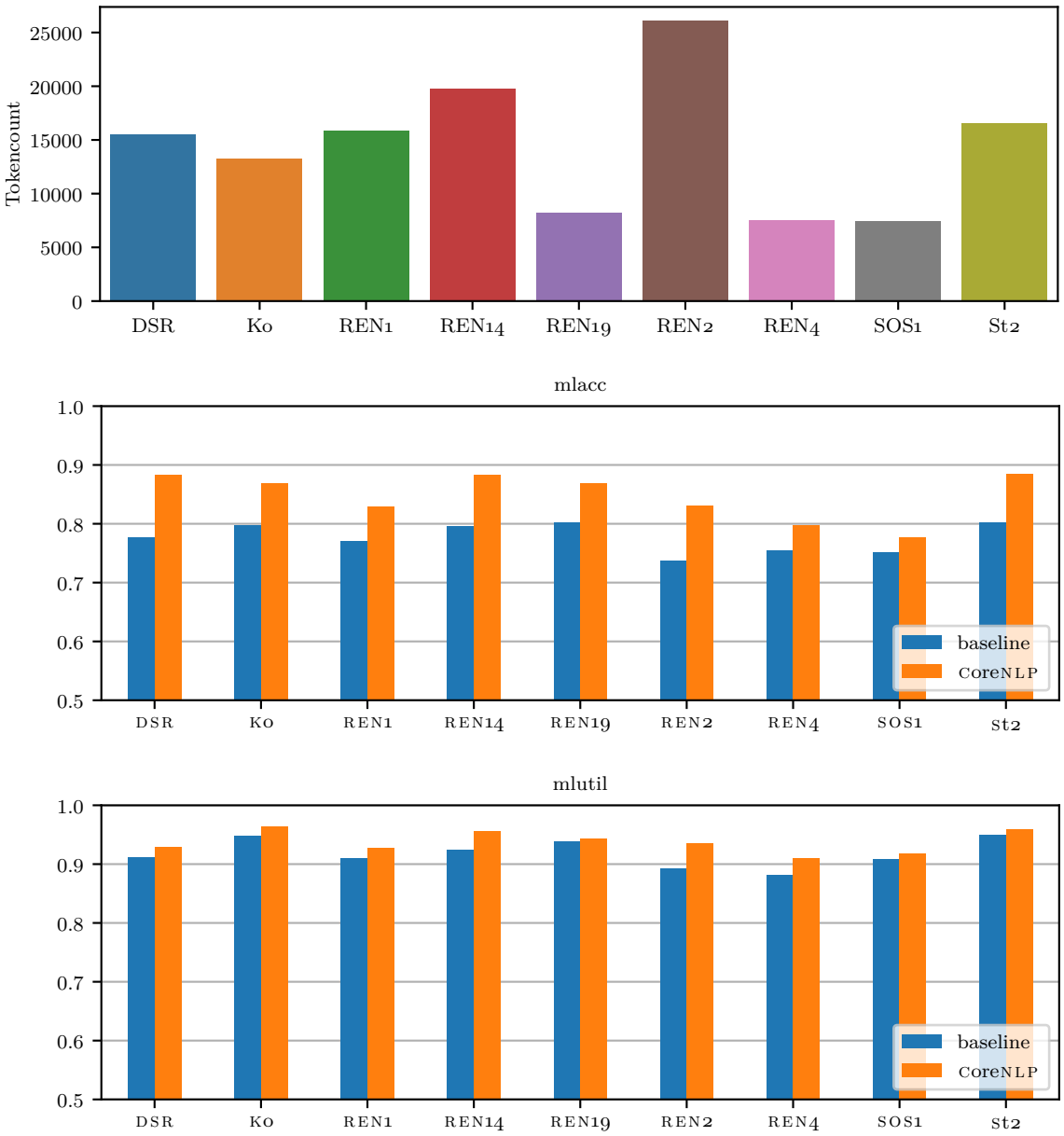


Fig. 6.7.: Performance of the baseline and the main tagger. The top plot shows the performance in terms of accuracy. The bottom plot shows the utility with $g_{\alpha=1,\beta=1}(\cdot)$.

6.5 Cross Domain Robustness

To evaluate the robustness of the classifier trained on some corpus and applied to unseen data we will split up the corpus along the document boundaries. Since we select one document for testing and the remaining documents for training we have to decide how to handle the test and training part boundaries setup for model-selection. For the training documents we have to decide whether to include their test part and for the evaluation document we have to decide if we want to use the training part for this document for the evaluation. The two extreme cases are are visualized in Fig. 6.8. Including the test part of documents C, D, E will give the tagger more training data. However, the model-selection is only optimized to the training part. For the evaluation on document F, we face the following trade-off: On the one hand, we would like to have a good estimate, but on the other hand, we also want the estimate to be clean and not overly optimistic. Since the model-selection has been done including the test-part of F, the classifier might perform better on that part of F, than in its test part. If we look back to Section 6.4,

Fig. 6.8.: Visualization of the two evaluation variants for across-document performance. The blue fraction is used for model-fitting, while the orange part is held back for performance evaluation.

we saw that the estimate of only 20% of one document seems to be sufficiently precise if the document is large enough. Therefore, we constrain the experiment similarly to only use documents for testing which are larger than 7.000 tokens. Additionally, we choose to exclude the test part of the training documents, because this might give the baseline a unfair advantage as it is not specifically tuned to the training part of the document. To this end, we select the left option of Fig. 6.8, using only that part for evaluation that has not been involved in model-selection.

For this experiment, we will train the two classifiers on the whole corpus except for one document each and evaluate its performance on the test part of this excluded document.

Unlike in Section 6.4, the evaluation in this chapter cannot be seen without considering the structure of the whole dataset. If the excluded document is in some way isolated from the dataset, the classifier will most likely perform bad on that document.

When comparing the overall accuracies in Fig. 6.9 to Section 6.4 or Section 6.7, we can see that both classifiers perform generally worse. This is, however, not surprising, as the classifier knows nothing about the document he is tested on. For some documents, the corpus contains texts which are roughly similar and hence allow reasonably high performance. The utility is generally in alignment with the accuracy but significantly increased.

Two documents are particularly interesting in this experiment: "Duisburger Stadtrecht, 1500" (DSR) and "Bremer Urkunden, 1351–1400" (REN4). The performance on DSR is much worse than on any other document. This document is one of the latest documents in the corpus. The two other documents from the 15th century are from Rostock and Bamberg and therefore originate far away from this document. This makes it very difficult to tag. On this text, both classifiers perform similarly bad with respect to accuracy. However, when letting the classifiers choose more elements, CoreNLP

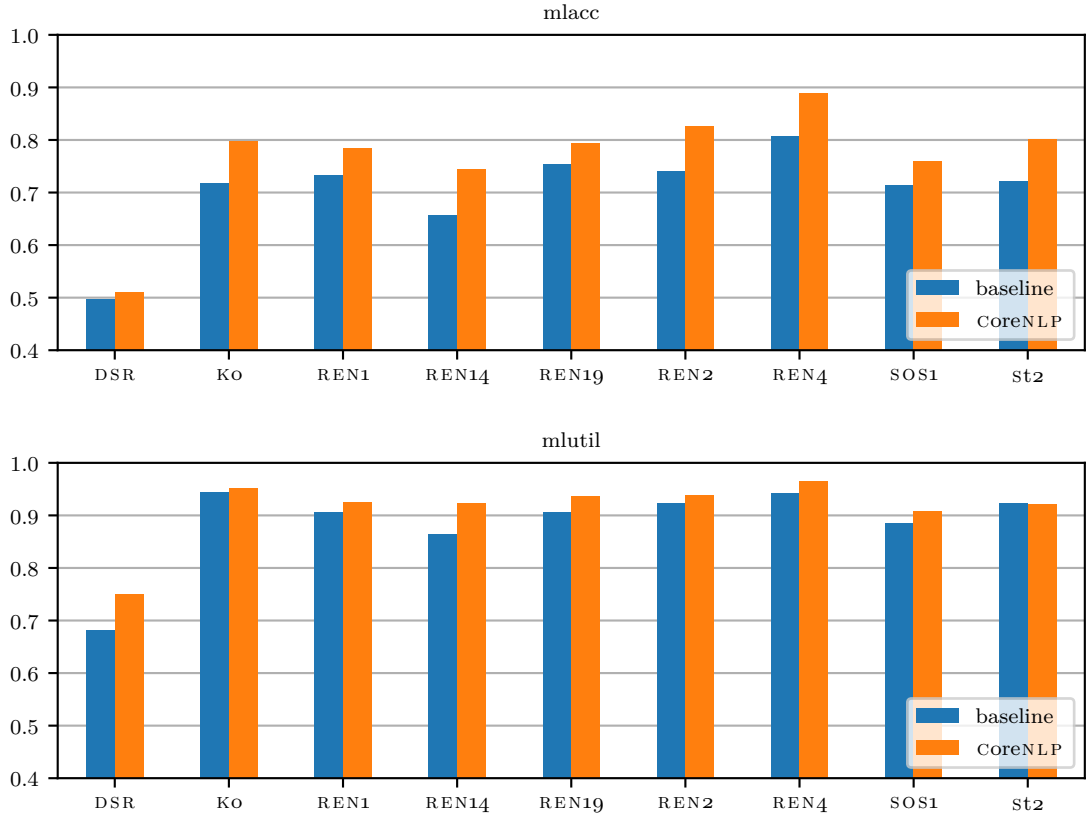


Fig. 6.9.: Performance of the baseline and the main tagger. The top plot shows the performance in terms of accuracy. The bottom plot shows the utility with $g_{\alpha=1,\beta=1}(\cdot)$.

outperforms the baseline clearly. This can especially be noticed in the average set size on this dataset. The baseline selects more than twice as many elements as our selected model.

The performance on REN4 is particularly high. The document is a collection of deeds. Those documents are part of a larger collection of deeds from Bremen ranging from 1301–1500. Additionally, the corpus contains many other documents from Bremen in the same period. REN2, for example, is the Bremen municipal law test from 1300. This high performance is also reflected in an average set size below 2 for the CoreNLP tagger. The accuracy even exceeds the performance on this document when trained on the test part of this dataset. Note that despite from Section 6.4, the classifiers have not seen anything of the REN4 document during training.

Another fact worth noting is that the utility of the two classifiers is sometimes extremely close (Ko, St2). However, the set size shows that their performance is not similar. This is in alignment with the argumentation in the beginning of this chapter, stating that the utility is very sensitive to the chosen parameters for the discount function. Table 6.4.

Tab. 6.4.: Results of the cross-domain robustness experiment. The performance of the main classifier and the baseline are compared across 9 different datasets with the performance measures accuracy, utility and average set size. The training has been conducted on the whole corpus except for the evaluation dataset.

		DSR	KO	REN1	REN14	REN19	REN2	REN4	SOS1	ST2
ml-acc	CoreNLP	51.05	79.76	78.51	74.53	79.34	82.68	88.98	75.93	80.26
	baseline	49.68	71.90	73.30	65.76	75.51	74.15	80.66	71.43	72.10
ml-util	CoreNLP	74.96	95.14	92.54	92.28	93.76	93.93	96.59	90.91	92.07
	baseline	68.23	94.41	90.65	86.41	90.58	92.29	94.19	88.55	92.41
$\lvert\hat{Y}\rvert$	CoreNLP	5.84	2.23	2.43	3.00	2.57	2.29	1.94	2.72	2.25
	baseline	13.05	3.79	4.57	8.25	4.72	4.15	3.94	5.69	3.75

6.6 Robustness Against Unrelated Data

In the previous section, we saw that the dataset is inhomogeneous. In particular the DSR text differs significantly from the rest of the dataset. To validate whether the classifier can deal with unrelated data, we conducted similar tests as in Section 6.4, but we trained on the whole dataset instead.

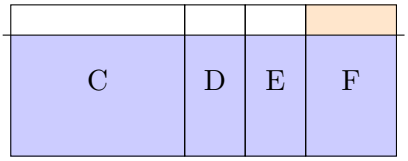


Fig. 6.10.: Training and Test data used for this experiment. The classifier is trained across the whole dataset and tested on the testing parts of single documents.

Fig. 6.11 shows the accuracy difference to the in-domain setting. On the first glance we can see that CoreNLP almost never loses predictive performance due to the additional training data. This is also true for the difficult DSR text. The baseline tagger on the other hand, can lose up to 5% points. The "Kolberger Kodex, 1300" (KO) and the ST2 document lose significantly for this tagger. Those datasets are among the older ones in the dataset and adding many other documents from later period might reflect an overall language shift. The contextual information available to CoreNLP seems to allow the tagger to distinguish the different usages. Since the baseline tagger ignores the word context, it will suffer performance losses for word usages that are uncommon for the dataset.

The performance gains are usually also larger for CoreNLP compared to the baseline tagger. This might be result of another effect. Fig. 6.15 indicates that the training data from one document is insufficient to fully train out the CoreNLP classifier. In combination with the observation from Section 6.5, that the dataset contains additional documents similar to REN4, it is little surprising that the classifier performs well. They gain 10 percent points against the classifier being trained only on REN4.

We have observed that the baseline classifier sometimes suffers from additional data, and even our main classifier can loose performance by adding additional data. When we observe the utility, on the other hand, both classifiers gain in nearly every dataset. This might be based on the same effect for which the baseline classifier outperformed our classifier on the DSR in Section 6.5, but did not outperform it in the utility. If the classifier has two strong candidates he is forced choose greedily the most probable one. Set-valued prediction allows to add multiple values and accept a small penalty

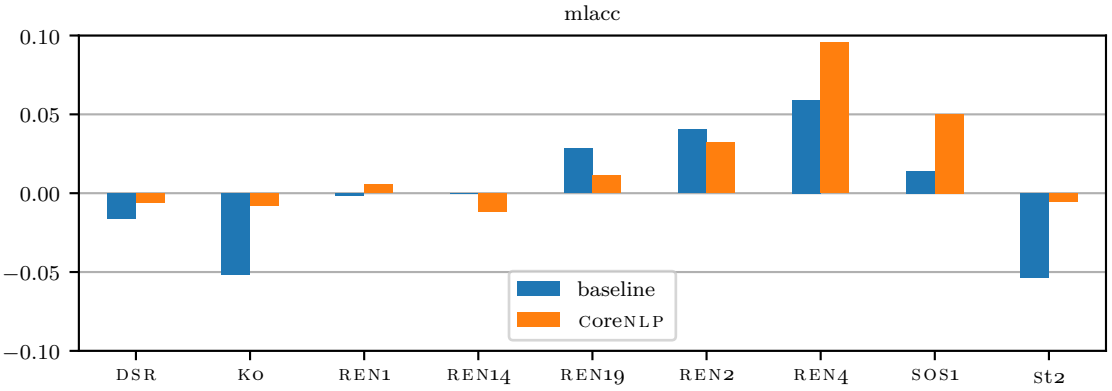


Fig. 6.11.: Accuracy gain in comparison to Section 6.4 across several datasets.

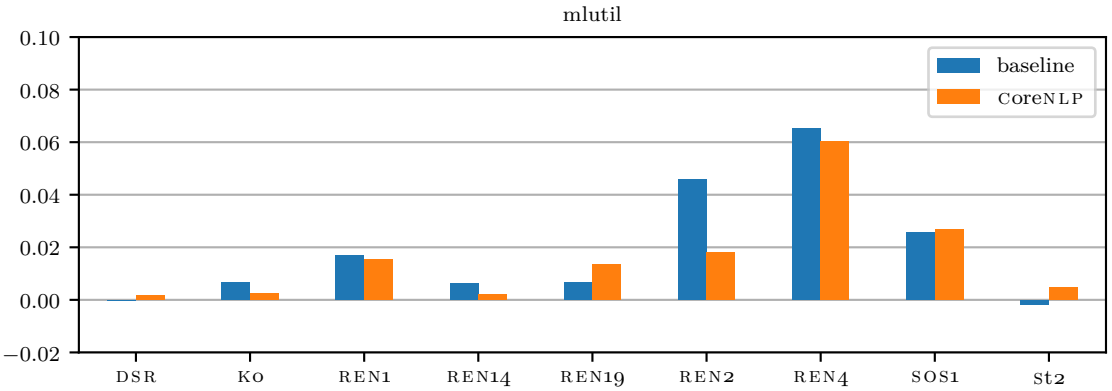


Fig. 6.12.: Utility gain in comparison to Section 6.4 across several datasets.

for each prediction. This will overall increase the performance of the classifier. This improvement in utility is not even penalized with large increases in average set size compared to the in-domain setting. For the KO document, both classifiers add less than 0.2 additional elements. The largest improvements on the utility happen however on the datasets that also gain the most accuracy. Table 6.5 lists the absolute performance numbers of this experiment.

Tab. 6.5.: Absolute performance of the classifier on a selection of datasets when trained on the whole training dataset.

		DSR	KO	REN1	REN14	REN19	REN2	REN4	SOS1	ST2
ml-acc	CoreNLP	87.62	86.12	83.45	87.12	87.89	86.33	89.34	82.67	87.92
	baseline	76.11	74.59	76.95	79.53	83.05	77.74	81.32	76.59	74.94
ml-util	CoreNLP	93.01	96.58	94.30	95.84	95.69	95.27	96.98	94.51	96.42
	baseline	91.07	95.42	92.72	92.95	94.49	93.75	94.58	93.41	94.73
$\lvert\hat{Y}\rvert$	CoreNLP	1.82	2.14	2.24	2.06	2.02	2.19	1.90	2.29	2.04
	baseline	4.84	3.45	3.90	4.54	3.23	3.95	3.82	4.28	3.35

6.7 Across Corpus Performance

The overall performance of the classifier is evaluated on the whole dataset. For this experiment, we train on the whole training data used for model-selection and test on the remaining 20% (compare Fig. 6.1)

In Fig. 6.13, we can see the comparison of the accuracy and different utility scores. Since we use the discount function of Eq. (2.4), we have two parameters that we can modify. Since the value of α changes only the amount of abstention, we decided to keep it fixed at the default value of 1 and only modified the convexity parameter β. Mortier et al. (2019) made similar decisions for their evaluation. A small value of β means the function is convex, hence adding more elements to the prediction is penalized strongly. Large values form a concave discount function, penalizing large sets only by a little.

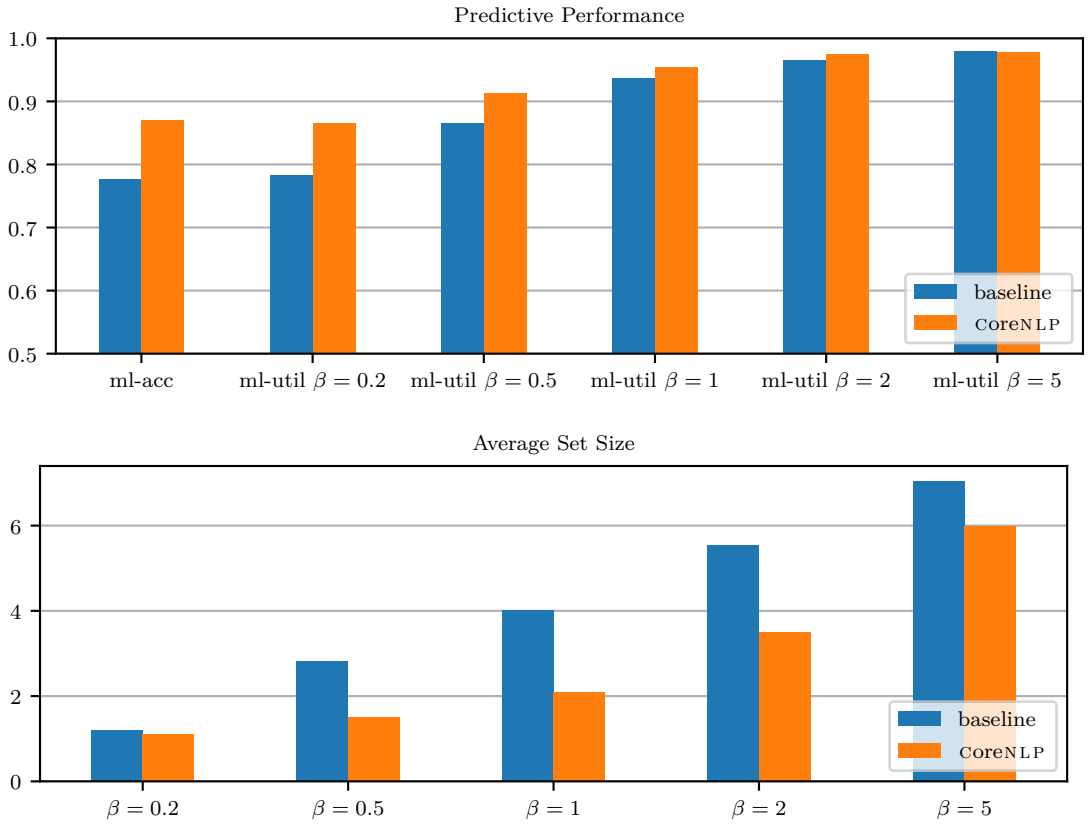


Fig. 6.13.: Performance of the baseline and the main tagger. The top plot shows the performance in terms of accuracy and utility for different choices the discount functions parameter β. The bottom plot shows the set size for the different utility measures.

For our dataset, larger values of β allow the classifiers to achieve very high recall. As we noted earlier, the utility is fundamentally connected to the accuracy by a factor of the discount function. Hence, achieving a utility that is lower than the accuracy value is generally possible, because the improved recall is penalized by the discount function. However, this is not very likely if we increase β, because this will reduce the penalty for larger sets. For $\beta = 0.2$, the penalty is so large, that the classifiers only seldom predicts more than one element. In this case, the utility is very close to the accuracy. In the extreme case for $\beta \to 0$, the utility and accuracy coincide. One nice effect of the utility is, that bad classifiers profit more than good classifiers. Therefore, we can also

achieve high scores with the baseline classifier. This comes, however, with the expense of preciseness. The set size increases significantly when selecting more concave discount measures. Interestingly, the relative difference of the set size is the highest $\beta = 1$, which means a linear discount. Here, the baseline selects nearly twice as many elements. This effect has also been seen in the previous experiments. For $\beta = 0.2$ and $\beta = 5$, the difference is much smaller. The sets are of the baseline are only 10%–20% larger.

Tab. 6.6.: Results of the overall corpus performance experiment. The performance of the main classifier and the baseline are compared different choices of the convexity parameter of the utility discount function $g(\cdot)$.

		$\beta = 0.2$		$\beta = 0.5$		$\beta = 1$		$\beta = 2$		$\beta = 5$	
	ml-acc	ml-util	$\lvert\hat{Y}\rvert$	ml-util	$\lvert\hat{Y}\rvert$	ml-util	$\lvert\hat{Y}\rvert$	ml-util	$\lvert\hat{Y}\rvert$	ml-util	$\lvert\hat{Y}\rvert$
CoreNLP	86.99	86.43	1.10	91.28	1.50	95.44	2.09	97.37	3.51	97.77	5.97
baseline	77.64	78.33	1.21	86.45	2.82	93.59	4.02	96.47	5.54	97.85	7.05

6.8 Error Analysis

In the previous section, we analyzed how the performance of the classifier in a broad way. The goal of this section to analyze the weaknesses of the classifier. If not stated differently, the training and test datasets are taken across the whole dataset, exactly as for the hyperparameter optimization and the experiments in Section 6.7.

6.8.1 Unknown Word Performance

The performance of part-of-speech taggers is usually quite powerful for known words. Table 6.7 shows high performance of the baseline tagger compared to the CoreNLP tagger. However, for unknown words, our baseline tagger can only predict with respect to the prior distribution. The high entropy of this prior distribution yields to a constant prediction set with 22 elements, which is basically a quarter of the whole label space. CoreNLP on the other hand is capable of tagging unknown words with around 70% accuracy. With 7.4% of the words in the test being unknown, this yields a much stronger overall performance. The amount of unknown words will probably explain also a part of the performance of the experiments in Section 6.4 and Section 6.5, however, they have been left out for reasons of space.

Tab. 6.7.: Tagger performance on known and unknown words

	Known			Unknown			Total		
	ml-acc	ml-util	$\lvert\hat{Y}\rvert$	ml-acc	ml-util	$\lvert\hat{Y}\rvert$	ml-acc	ml-util	$\lvert\hat{Y}\rvert$
CoreNLP	88.49	95.95	1.89	70.17	89.72	4.25	86.99	95.44	2.09
baseline	81.49	95.94	2.59	29.38	64.07	22.00	77.64	93.59	4.02

6.8.2 Per Tag Performance

Next, we will analyze the classifiers performance for each tag. Some tags of the dataset are extremely rare, so that only 81 of the 92 total tags show up as ground truth tags

in the test set. To better understand the errors, we report the most frequent tags in Table 6.8.

Tab. 6.8.: Relative tag frequency across the whole dataset of the 10 most frequent tags.

	NA	APPR	DDART	KON	PPER	AVD	VVINF	ADJA	DPOSA	KOUS
Probability	19.61%	8.58%	6.72%	6.27%	5.01%	4.19%	4.09%	3.04%	2.49%	2.47%

Fig. 6.14 shows significantly differing performance across the different classifiers. The accuracy and relative frequency of a tag in the training data is correlated (0.44 Pearson correlation). This means that a the frequency of a tag effects the classifiers performance (compare Section 6.8.3), however, those low performances have smaller influence on the overall performance. However, for some tags like the DDA (definite, attributive determiner), 0.6% of the training data are sufficient to train the classifier to above 97.4% accuracy. Since this is a closed class tag, we would expect other closed class tags to perform similarly, as we assume most words of this have been seen during training. The results on the DDS tag (definite, substituting determiner) is, however, much lower. Despite 1.7% of the words in the training data belong to this class the accuracy of on this tag is only 72.82%. Overall, many of the verb classes (V-prefix) have only about 60% accuracy. This might be result of the large number of subclasses of verb tags.

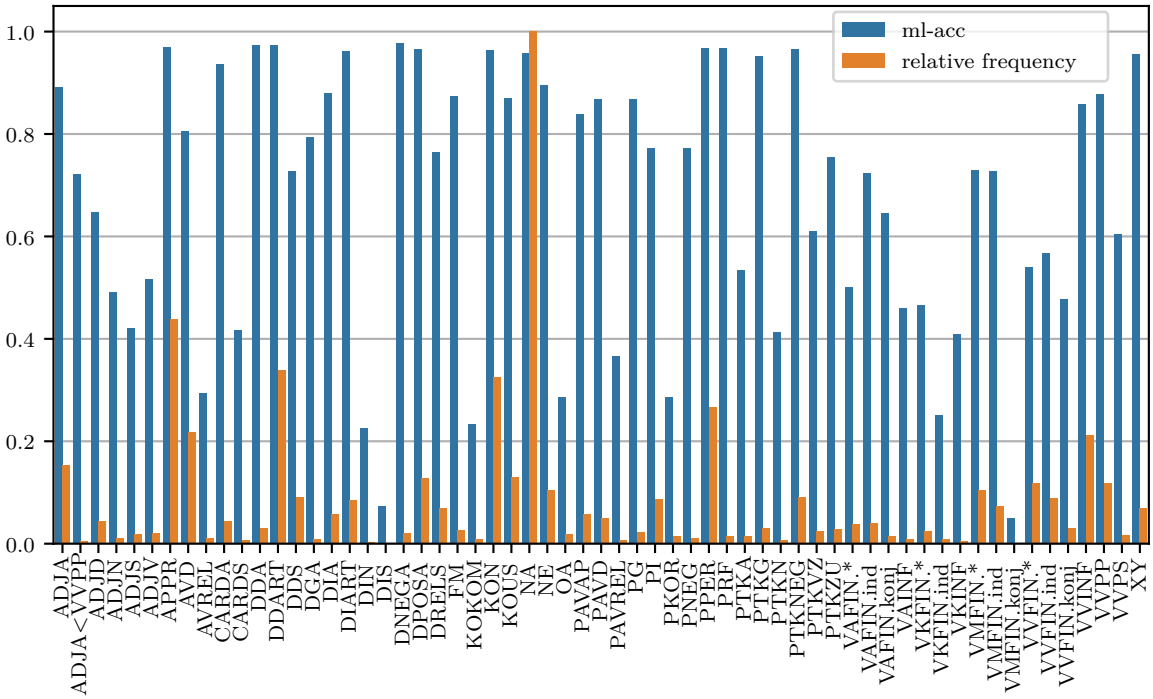


Fig. 6.14.: ml-acc and relative frequency in the training data for each tag. The relative frequency is normalized to the most frequent tag (NA). Only tags with at least 20 occurrences in the test data are shown.

This issue gets even more visible when Table 6.9. There are generally two main types of errors: Missclassified verb forms and words that are falsely classified as common nouns. The 5 out of the top ten errors are verb form misclassifications. This is significantly more than Koleva et al. (2017) report. However, their tagset is much less fine grained

with respect to verb forms. Misclassifications as (VVFIN.*, VVFIN.ind) are however much more reasonable than (VVFIN.*, NA). Although when trying to establish a gold standard tagging, the error has to be weighted equally. Confusing infinitive verbs (VVINF) with nouns (NA) and the other way around seems to be a very common tagging error for MLG. Koleva et al. (2017) report similar errors for their tagger. Also, classifying adverbs (ADV), adjectives (ADJ) and finite verbforms of full verbs (VVFIN.*, VVFIN.ind) as common nouns has been reported in their paper. Apart from that, we can see that the OA tag ("ohne Annotat") shows overall very poor performance. This tag is meant to capture tagging errors and is therefore a mix of different word classes. Words from this class will come in very different shapes and many different word contexts. It is therefore expected that the classifier performs poorly on this tag.

Tab. 6.9.: The 10 most misclassified tags and their most common misclassifications. The last two columns lists the total number of errors and the accuracy of this tag on the test dataset. The tagging errors (second column) explain 34.84% of all errors on the test set.

Correct tag	Most common errors	Total errors	ml-acc
VVFIN.*	VVFIN.ind (107), VVINF (102), NA (68)	354	54.03
NA	VVINF (56), ADJA (36), VVFIN.* (28), NE (23), VVFIN.ind (14)	207	95.85
AVD	PAVD (31), NA (28), KON (25), APPR (16), ADJV (15)	203	80.59
VVFIN.ind	VVFIN.* (92), NA (27), VVPP (18), VAFIN.ind (16)	201	56.65
VMFIN.*	VMFIN.ind (142)	159	72.96
DDS	DDART (68), DRELS (31), KOUS (16)	137	72.82
VVINF	NA (118)	125	85.85
OA	NA (37), APPR (15), XY (8), ADJA (7), NE (7)	120	28.57
VAFIN.*	VAFIN.ind (52), VVFIN.* (23), VAFIN.konj (12), VVFIN.ind (9)	118	50.00
ADJA	NA (54), ADJS (10), NE (8), VVPP (8)	102	89.23

The errors related to the NA tags raise the questions on the correlation of word ambiguity and tag confusion. Words that generally show up with similar tags might be confused more easily. Inversely, we can calculate the similarity of bag of words of two tags and hence express their risk for confusion. If this risk is high, we would expect the classifier to make many tagging errors on those tags, with low distance and to confuse them often. In order to formally measure this distance between two tags we will use the Ruzicka similarity, a multi-set generalization of the Jaccard similarity (Cha, 2007)

$$\mathrm{ruz}(\mathbf{x}, \mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}, \tag{6.1}$$

with $\mathbf{x}$ and $\mathbf{y}$ being multi-sets and i iterating over the elements in the multi-set. Applied to the tag context of taggings, $\mathbf{x}$ and $\mathbf{y}$ are bags of words labeled with tag t_i and t_j in the training dataset. We can therefore calculate a similarity of two tags with respect to relative overlap in their induced words.

Two tags with high Ruzinca similarity are APPR (preposition) and PAVAP (pronominal adverb, prepositional part). But even though their similarity is 0.246, the total number of confusions is only 28. Notice that APPR is the second most common tag; therefore, we might expect more errors. The contextual differences are allow the classifier to disambiguate the words well.

When analyzing the tagging errors related to the noun tags (NA), the similarities are much smaller, but the tagging errors are still more severe. The similarity between NA and ADJA is 0.025 and between NA and ADV is 0.015. This indicates that the tag confusion might be the result of less distinctive contextual information.

6.8.3 Generalization vs. Training Dataset Size

Another interesting analysis is the connection between generalization performance and available training data. Generally, simpler models are less flexible to fit complex problems, and achieve very low out-of-sample errors. On the other hand, simpler models achieve their maximal performance with less training data, as they have less parameters to tune. With this said, we would expect the simple baseline classifier to perform better with few training data and expect CoreNLP to take over once the data is sufficient to fit a reasonable model.

To investigate this further, we selected the document that achieved highest performance in Section 6.4. The "Stadtrecht Stade, 1250" (ST2) is with around 16,000 tokens fairly large and therefore suited for such an experiment. The evaluation is done on the whole test part of the text, but the training is done on only a fraction p of the test dataset. We trained and evaluated the classifiers with 1%–100% of the training data. The results are visualized in Table 6.10. The specific steps in between have been selected to a roughly smooth curve with limited amount of computation time. The kink at around 1600 tokens is result of the in homogeneity of the training part. The additional 8000 tokens achieved nearly no improvement. The estimation could be improved by selecting different sections of the training data for each specific value of p and averaging the results. We are confident that this would remove the local concavity of the function. Due to time constraints we omitted further investigation of this issue.

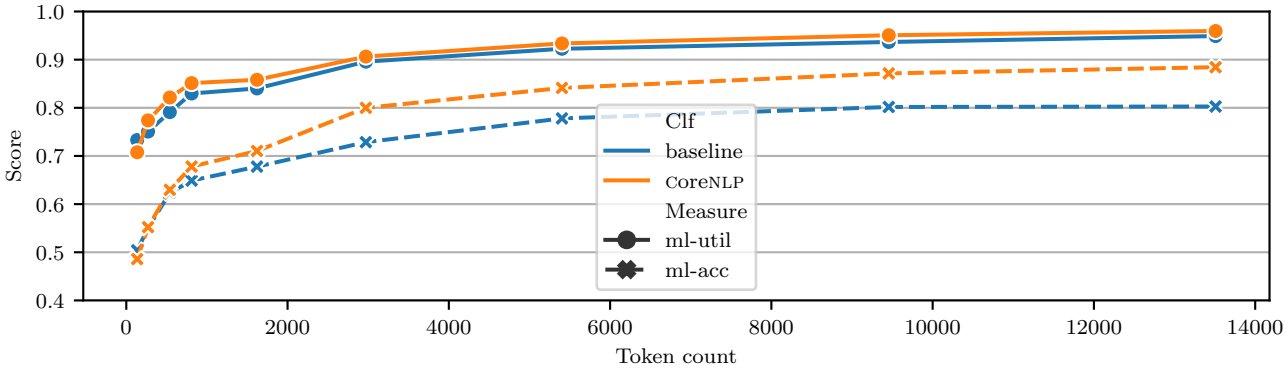


Fig. 6.15.: Classifiers performance and Utility evaluated trained and evaluated on the ST2 dataset. The fraction of the total training data is gradually increased from 135 tokens up to the whole training part of 13509 tokens.

Fig. 6.15 proves our assumption of the baseline outperforming CoreNLP for small datasets correct. However, this is only true when the classifiers are trained on only 135 tokens. Already when trained on 2% of the training data, our classifier is on even terms with the baseline. With around 3000 tokens, the accuracy of the two classifiers is clearly separated. Although they still profit from additional training data. The baseline performance barely increases between 70% and 100% of the available training data. The main tagger, on the other hand, continues to gain performance and could probably profit from even more training data. When looking at the utility, we see again that the performance numbers are much less distinct. The performance gap between the two classifiers gets smaller when observing the utility. Nevertheless, the set size shows reflects the different confidence we observe in the accuracy from around 3000 tokens and more. The exact values are reported in Table 6.10.

Tab. 6.10.: Classifiers accuracy, utility and set size ST2. The training set is gradually increased. The performance is evaluated on the whole test data.

		135	270	540	810	1621	2971	5403	9456	13509
ml-acc	CoreNLP	48.59	55.22	62.97	67.80	71.04	80.00	84.13	87.15	88.43
	baseline	50.40	55.04	62.33	64.89	67.79	72.86	77.80	80.17	80.27
ml-util	CoreNLP	70.77	77.38	82.14	85.13	85.82	90.64	93.37	95.07	95.96
	baseline	73.31	75.06	79.10	82.97	84.02	89.58	92.26	93.67	94.90
$\lvert\hat{Y}\rvert$	CoreNLP	10.40	8.08	6.01	5.05	3.59	2.81	2.29	2.10	2.01
	baseline	10.35	8.22	6.89	6.39	5.05	4.45	3.68	3.58	3.44

6.9 Other Experiments

In this section, we present observations that were not incorporated because their effects demand further research.

6.9.1 Performance Effect of Shuffling Sentences

As discussed in previous sections, the sentence splitting in MLG is not reliable, therefore we finally selected not to chunk the documents into sentences. However, in early experiments, we chunked the sentences with respect to the punctuation and XY tags. The preliminary performance evaluation showed that shuffling those document sentences improved the performance. This observation hints that it is advantageous to shuffle the dataset when annotating the dataset by a human annotator. The effect might be explained the following way: If we assume the document itself is not entirely homogeneous, but instead decomposes into several sections A, B, C, ... Shuffling the dataset first, before selecting the training and test part, gives us training data from each section and test data across the whole document. If, on the contrary, we set a random split inside the document and then select a continuous chunk for training, we might end up with sections B, C, D as training data, but section A as testing data. Therefore, the classifier will show worse performance than by selecting training data from all sections.

6.9.2 Pruning Dataset

The dataset used in this thesis is fairly diverse. Therefore, we analyzed the possibility of selecting a subset of the training data with respect to the testing data which we would like to tag. Early ideas were to define a distance metric on the documents meta-data and exclude documents whose distance is above some threshold. It has been investigated with full knowledge of the performance, meaning that we fitted different subsets of the training data and analyzed if the score on the evaluation exceeds the score on the whole training data. This very simple method yielded only very small improvements. When trying to apply this technique in inference, we have to define the selection algorithm beforehand and than use the prediction that the tagger produces with this training data. In inference, there is no possibility to check which set of training data would yield best tagger performance.

Later evaluations, however, showed that the preselection of the training data seems to affect only the accuracy of the classifier positive. Section 6.6 indicates, that the utility is almost never effected negatively with increasing training data, although the set size could potentially improve.

6.9.3 Constraint Reduction

The selection of valid tags for a specific word is very strict in CoreNLP and strongly tied to the selection of open and closed class words. To relax this constraint the original implementation applies a procedure, similar to Algorithm 4. For English they apply the rules $R = \{\{\text{VB}, \text{VBP}\}, \{\text{VBD}, \text{VBN}\}\}$. The reasoning behind those rules is that some tags are often confused and therefore all confusion pairs should be considered.

For this reason, we introduced above the Ruzicka similarity between tags, calculated over their induced bag of words (see Eq. (6.1)). Table 6.11 shows the ten tag pairs with the highest similarity values. In order to investigate the effect of the tag expansion rules we selected the 15 tag pairs with the highest similarity and allowed the hyperoptimizer to choose any subset of $R' \subseteq R$ among these expansion rules. While fixing all other hyperparameters this results in a parameter space of $2^{15} = 32\,768$ configurations.

Tab. 6.11.: Top ten closest tag pairs by Ruzicka similarity of the induced words.

	DDD DDSA	AVG PAVG	DGN DGS	PAVD PAVREL	AVG AVW	DIN DIS	DRELS PKOR	VAFIN.ind VKFIN.ind	VAFIN.konj VKFIN.konj	AVW PAVG
$\text{ruz}(\cdot,\cdot)$	0.5	0.5	0.5	0.3913	0.375	0.3704	0.3585	0.3392	0.3077	0.2857

We stopped the optimization in this configuration space after around 25 evaluations executed in 3h 15 minutes, since none of the so-far tested rules showed significant improvements.

7 Conclusion

In this thesis, we developed a part-of-speech tagger, capable of providing set-valued predictions. For this reason, we investigated existing tagger implementations and developed a wrapper that can provide set-valued predictions from any tagger capable of predicting a posterior distribution. To test the performance of the algorithm we selected a historic corpus in MLG. We modified the classical performance measures to match the needs of this specific dataset. Additionally, we introduced a baseline tagger to be able to compare the performance of our tuned tagger.

The algorithm has been tuned to the dataset in several hyperparameter optimization steps. We investigated the performance of our classifier with several experiments on different fractions of the dataset. The CoreNLP tagger was able to beat the baseline in every setting. For some settings, the margin was very large. In all experiments we saw, that set-valued prediction allows both classifiers to perform much better than in their classical setting. The performance difference in terms of utility was much smaller than for the accuracy score. However, this performance increase came with the cost of larger sets.

The error analysis showed once more that the classifiers gain especially when they are uncertain. For example on unknown words, nearly 9 of 10 correct tags where retrieved. We also showed that CoreNLP is also capable to provide good predictions with very little data and outperformed our simple baseline with only 1000 tokens of training data. The learning curves indicate that the classifier is not even trained out completely and could learn even more from additional data.

Overall set-valued prediction allows the classifier to provide powerful predictions in the context of uncertainty. The evaluation also showed that set-valued prediction makes the classifier more robust against confusing data elements as we observed in Section 6.6.

7.1 Future Work

Some observations during early stages were not fully understood and might allow further improvements of the classifier's performance.

Pseudo Sentence Splitting Early experiments indicated, that the classifier benefits from clear sentence boundaries. Since MLG provides on reliable boundaries, we could add a preprocessing step that would produce a chunking first and then predict the chunked sentences. Such a chunking could rely on XY tags and punctuation.

Improved Pruning We saw, that the pruning of unrelated data can boost the classifier's accuracy. It would be interesting to investigate if there are settings where also the utility will profit from dataset pruning. This would mean, that the dataset would need to be split in smaller chunks than only the document boundaries and more reliable distance metrics need to be defined. The selection of training samples could be managed by a machine learning algorithm itself.

Valid Tag Constraint The selection of valid tags influences the algorithmic complexity negatively if too many tags are considered. However, considering too little tags will constraint the search space in such a way, that only suboptimal taggings can be selected. In combination with the strong assumptions about open- and closed tags, the classifier could probably gain significant performance or speedup if the selection of valid tags is done in a more principled way. Instead of selecting the valid tags by a static heuristic, a good selection could be learned with a machine learning algorithm. Smaller sets of valid tags would allow the classifier to consider larger tag context windows.

Improved Model Selection Since many configurations of CoreNLP are not valid, the search space for the feature extractor frames was strongly restricted. A better understanding of the effect of those parameters on the performance of the classifier would allow to consider a larger search space, which more runnable configurations.

Bibliography

Arlot, Sylvain, Alain Celisse, et al. (2010). "A survey of cross-validation procedures for model selection". In: *Statistics surveys* 4, pp. 40–79 (cit. on p. 13).

Bergstra, James S, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (2011). "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*, pp. 2546–2554 (cit. on p. 15).

Bollmann, Marcel, Florian Petran, Stefanie Dipper, and Julia Krasselt (2014). "CorA: A web-based annotation tool for historical and other non-standard language data". In: *Proceedings of the 8th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH)*, pp. 86–90 (cit. on p. 19).

Brants, Thorsten (2000). "TnT: a statistical part-of-speech tagger". In: *Proceedings of the sixth conference on Applied natural language processing*. Association for Computational Linguistics, pp. 224–231 (cit. on pp. 6, 19).

Cha, Sung-Hyuk (2007). "Comprehensive survey on distance/similarity measures between probability density functions". In: *City* 1.2, p. 1 (cit. on p. 56).

Chow, C (1970). "On optimum recognition error and reject tradeoff". In: *IEEE Transactions on information theory* 16.1, pp. 41–46 (cit. on p. 20).

Chow, Chi-Keung (1957). "An optimum character recognition system using decision functions". In: *IRE Transactions on Electronic Computers* 4, pp. 247–254 (cit. on p. 20).

Collobert, Ronan, Jason Weston, Léon Bottou, et al. (2011). "Natural language processing (almost) from scratch". In: *Journal of machine learning research* 12.Aug, pp. 2493–2537 (cit. on p. 19).

Derczynski, Leon, Alan Ritter, Sam Clark, and Kalina Bontcheva (2013). "Twitter part-of-speech tagging for all: Overcoming sparse and noisy data". In: *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pp. 198–206 (cit. on pp. 19, 40).

Dipper, Stefanie, Karin Donhauser, Thomas Klein, et al. (2013). "HiTS: ein Tagset für historische Sprachstufen des Deutschen." In: *JLCL* 28.1, pp. 85–137 (cit. on pp. 19, 24).

Echelmeyer, Nora, Nils Reiter, and Sarah Schulz (2017). "Ein PoS-Tagger für" das" Mittelhochdeutsche". In: (cit. on p. 19).

Falkner, Stefan, Aaron Klein, and Frank Hutter (July 2018). "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". In: *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pp. 1436–1445 (cit. on pp. 15, 43).

Geisser, Seymour (1975). "The predictive sample reuse method with applications". In: *Journal of the American statistical Association* 70.350, pp. 320–328 (cit. on p. 13).

Godbole, Shantanu and Sunita Sarawagi (2004). "Discriminative methods for multi-labeled classification". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer, pp. 22–30 (cit. on p. 30).

Huang, Zhiheng, Wei Xu, and Kai Yu (2015). "Bidirectional LSTM-CRF models for sequence tagging". In: arXiv: 1508.01991 (cit. on p. 19).

Hutter, Frank, Holger H Hoos, and Kevin Leyton-Brown (2011). "Sequential model-based optimization for general algorithm configuration". In: *International conference on learning and intelligent optimization*. Springer, pp. 507–523 (cit. on p. 15).

Jurafsky, Daniel and James H. Martin (May 16, 2008). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd ed. Prentice Hall (cit. on pp. 6, 9, 31).

Koleva, Mariya, Melissa Farasyn, Bart Desmet, Anne Breitbarth, and Véronique Hoste (2017). "An automatic part-of-speech tagger for Middle Low German". In: *International Journal of Corpus Linguistics* 22.1, pp. 107–140 (cit. on pp. 1, 9, 19, 23, 29, 31, 40, 47, 55, 56).

Lafferty, John, Andrew McCallum, and Fernando CN Pereira (2001). "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: (cit. on pp. 6, 9, 19).

Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar (2016). "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: arXiv: 1603.06560 (cit. on pp. 15, 43).

Lindauer, Marius, Katharina Eggensperger, Matthias Feurer, et al. (2017). *SMAC v3: Algorithm Configuration in Python*. https://github.com/automl/SMAC3 (cit. on p. 43).

McCallum, Andrew, Dayne Freitag, and Fernando CN Pereira (2000). "Maximum Entropy Markov Models for Information Extraction and Segmentation." In: (cit. on pp. 8, 19).

Mortier, Thomas, Marek Wydmuch, Eyke Hüllermeier, Krzysztof Dembczyński, and Willem Waegeman (2019). "Efficient Algorithms for Set-Valued Prediction in Multi-Class Classification". In: arXiv: 1906.08129 (cit. on pp. 1, 11, 12, 20, 27, 30, 36, 53).

Pearson, Karl (1895). "VII. Note on regression and inheritance in the case of two parents". In: *proceedings of the royal society of London* 58.347-352, pp. 240–242 (cit. on p. 17).

Picard, Richard R and R Dennis Cook (1984). "Cross-validation of regression models". In: *Journal of the American Statistical Association* 79.387, pp. 575–583 (cit. on p. 14).

Platt, John C. (1999). "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods". In: *ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, pp. 61–74 (cit. on p. 8).

Prasad, Rashmi, Nikhil Dinesh, Alan Lee, et al. (2008). "The Penn Discourse TreeBank 2.0." In: *LREC*. Citeseer (cit. on p. 23).

Ramaswamy, Harish G, Ambuj Tewari, Shivani Agarwal, et al. (2018a). "Consistent algorithms for multiclass classification with an abstain option". In: *Electronic Journal of Statistics* 12.1, pp. 530–554 (cit. on p. 11).

– (2018b). "Consistent algorithms for multiclass classification with an abstain option". In: *Electronic Journal of Statistics* 12.1, pp. 530–554 (cit. on p. 20).

Ratnaparkhi, Adwait (1996). "A maximum entropy model for part-of-speech tagging". In: *Conference on Empirical Methods in Natural Language Processing* (cit. on p. 19).

ReN-Team (Aug. 2019). *Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200-1650)*. http://hdl.handle.net/11022/0000-0007-D829-8. Version 1.0 (cit. on p. 22).

Samuelsson, Christer (1993). "Morphological tagging based entirely on Bayesian inference". In: *Proceedings of the 9th Nordic Conference of Computational Linguistics (NODALIDA 1993)*, pp. 225–238 (cit. on p. 6).

Schmid, Helmut (1999). "Improvements in part-of-speech tagging with an application to German". In: *Natural language processing using very large corpora*. Springer, pp. 13–25 (cit. on p. 19).

– (1994). "Probabilistic Part-of-Speech Tagging Using Decision Trees, Intl". In: *Conference on New Methods in Language Processing. Manchester, UK* (cit. on p. 19).

Seemann, Nina, Marie-Luis Merten, Michaela Geierhos, Doris Tophinke, and Eyke Hüllermeier (2017). "Annotation Challenges for Reconstructing the Structural Elaboration of Middle Low German". In: *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pp. 40–45 (cit. on pp. 19, 22).

Shafer, Glenn and Vladimir Vovk (2008). "A tutorial on conformal prediction". In: *Journal of Machine Learning Research* 9.Mar, pp. 371–421 (cit. on p. 20).

Shen, Libin, Giorgio Satta, and Aravind Joshi (2007). "Guided learning for bidirectional sequence classification". In: *Proceedings of the 45th annual meeting of the association of computational linguistics*, pp. 760–767 (cit. on p. 19).

Sheskin, David J (Apr. 2011). *Handbook of parametric and nonparametric statistical procedures*. 5th ed. Chapman and Hall/CRC (cit. on p. 16).

Sokolova, Marina and Guy Lapalme (2009). "A systematic analysis of performance measures for classification tasks". In: *Information processing & management* 45.4, pp. 427–437 (cit. on p. 12).

Tophinke, Doris (2012). "Syntaktischer Ausbau im Mittelniederdeutschen. Theoretisch-methodische Überlegungen und kursorische Analysen". In: *Niederdeutsches Wort* 52, pp. 19–46 (cit. on p. 21).

– (2009). "Vom Vorlesetext zum Lesetext: Zur Syntax mittelniederdeutscher Rechtsverordnungen im Spätmittelalter". In: *Oberfläche und Performanz. Untersuchungen zur Sprache als dynamische Gestalt*, pp. 161–186 (cit. on p. 21).

Toutanova, Kristina, Dan Klein, Christopher D Manning, and Yoram Singer (2003). "Feature-rich part-of-speech tagging with a cyclic dependency network". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for computational Linguistics, pp. 173–180 (cit. on pp. 6, 9, 19, 40).

Toutanova, Kristina and Christopher D Manning (2000). "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger". In: *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*. Association for Computational Linguistics, pp. 63–70 (cit. on p. 19).

El-Yaniv, Ran and Yair Wiener (2010). "On the foundations of noise-free selective classification". In: *Journal of Machine Learning Research* 11.May, pp. 1605–1641 (cit. on p. 20).

Yin, Wenpeng, Katharina Kann, Mo Yu, and Hinrich Schütze (2017). "Comparative study of CNN and RNN for natural language processing". In: arXiv: 1702.01923 (cit. on p. 9).

List of Figures

2.1 The NLP pipeline. The scope of this thesis is limited to part-of-speech tagging . . . 3
2.2 Example sentence with extracted relations using `https://corenlp.run` . . . 3
2.3 Visualization of an exemplary hidden Markov process used in an HMM . . 8
2.4 Visualization of Viterbi path and backpointers . . . 8
2.5 Example of features that could be used by a MEMM . . . 9
2.6 Comparison of numerical and one-hot encoding . . . 10
2.7 Splitting of data for cross-validation and performance evaluation . . . 13

4.1 Scatter plot of document origin and metadata from the corpus. Small jittering has been added to the geographic positions to decrease the overlap of the datapoints. . . . 21
4.2 Distribution of the origin and creation time of the documents. The x-axis is the total number of tokens from documents in the respective category. . 22
4.3 Heat map reporting the tag distribution of each document . . . 23

5.1 Example graph on which CoreNLP calculates the optimal tagging . . . 33
5.2 Visualization of a set-valued context window in CoreNLP . . . 35

6.1 Visualization of the within document train-test split . . . 39
6.2 Performance comparison of the two default models and the two results of our optimization runs . . . 44
6.3 Performance effect of punctuation analyzed on the simpler default model and the final optimization model . . . 45
6.4 Performance effect of data augmentation analyzed on the final optimization model . . . 45
6.5 Performance effect of the open and closed class segmentation analyzed on the final optimization model . . . 46
6.6 Evaluation of in-domain performance. The classifier is trained on the training part of one document and evaluated on the test part of the same document. . . . 46
6.7 Performance of the baseline and the main tagger. The top plot shows the performance in terms of accuracy. The bottom plot shows the utility with $g_{\alpha=1,\beta=1}(\cdot)$. . . . 48
6.8 Visualization the two variants to evaluate cross-document performance . . 49
6.9 Performance of the baseline and the main tagger. The top plot shows the performance in terms of accuracy. The bottom plot shows the utility with $g_{\alpha=1,\beta=1}(\cdot)$. . . . 50
6.10 Training and Test data used for this experiment. The classifier is trained across the whole dataset and tested on the testing parts of single documents. 51
6.11 Accuracy gain in comparison to Section 6.4 across several datasets. . . . 52

6.12 Utility gain in comparison to Section 6.4 across several datasets. 52
6.13 Across corpus performance and set size 53
6.14 ml-acc and relative frequency in the training data for each tag. The relative frequency is normalized to the most frequent tag (NA). Only tags with at least 20 occurrences in the test data are shown. 55
6.15 Classifiers performance and Utility evaluated trained and evaluated on the ST2 dataset. The fraction of the total training data is gradually increased from 135 tokens up to the whole training part of 13509 tokens. 57

List of Tables

2.1 Comparison of different classification problems 11
2.2 Example performance scores and application of the Wilcoxon signed-rank test. 16

4.1 The tagset used in the corpus. 24
4.2 Relative frequency of the multi-label target types in the dataset. 25

5.1 Comparison between token and sentence-level set-valued prediction 28
5.2 Selection of CoreNLP feature extractors 34

6.1 Configuration space for the feature extraction architecture. The blue marked elements are the default values from which the search was started. 42
6.2 Configuration space for the feature extraction architecture. 43
6.3 Results of the in-domain experiment. The performance of the main classifier and the baseline are compared across 9 different datasets with the performance measures accuracy, utility and average set size 47
6.4 Results of the cross-domain robustness experiment. The performance of the main classifier and the baseline are compared across 9 different datasets with the performance measures accuracy, utility and average set size. The training has been conducted on the whole corpus except for the evaluation dataset. 51
6.5 Absolute performance of the classifier on a selection of datasets when trained on the whole training dataset. 52
6.6 Results of the overall corpus performance experiment. The performance of the main classifier and the baseline are compared different choices of the convexity parameter of the utility discount function $g(\cdot)$. 54
6.7 Tagger performance on known and unknown words 54
6.8 Relative tag frequency across the whole dataset of the 10 most frequent tags. 55
6.9 Most common tagging errors 56
6.10 Classifiers accuracy, utility and set size ST2. The training set is gradually increased. The performance is evaluated on the whole test data. 58
6.11 Top ten closest tag pairs by Ruzicka similarity of the induced words. . . . 59

A.1 List of all documents in the corpus with their full name, year (50 year resolution) and their number of Tokens. 71

Appendix

A

A.1 Corpus

Table A.1 provides metadata information for all documents in the Corpus.

Tab. A.1.: List of all documents in the corpus with their full name, year (50 year resolution) and their number of Tokens.

	Name	Year	Tokencount
SOS1	Soester Schrae	1300	7,413
SRR1	Statuarrecht Stadt Rüthen	1300	5,391
St2	Stadtrecht Stade	1250	16,572
WS1	Werler Statuten	1300	1,328
BS1	Stadtrecht Braunschweig	1200	2,376
DSR	Duisburger Stadtrecht	1500	15,514
Ko	Kolberger Kodex	1300	13,249
REN1	Bremer Sachsenspiegel	1350	15,803
REN2	Bremer Stadtrecht	1300	26,076
REN3	Bremer Urkunden 1301-1350	1350	1,869
REN4	Bremer Urkunden 1351-1400	1400	7,531
REN5	Bremer Urkunden 1401-1450	1450	1,739
REN6	Bremer Urkunden 1451-1500	1500	2,632
REN7	Hamburger Urkunden 1301-1350	1350	202
REN8	Hamburger Urkunden 1351-1400	1400	6,589
REN9	Hamburger Urkunden 1401-1450	1450	4,181
REN10	Hamburger Urkunden 1451-1500	1500	3,407
REN11	Ravensberger Urkunden	1300	501
REN12	Werler Urkunden Neheim	1300	255
REN14	Bamberg	1500	19,715
REN15	Rostocker Bürgerspr.	1550	1,169
REN18	Braunschweig Ält. DegB Altst. I	1300	1,711
REN19	Braunschweig Ält. DegB Altst. II	1300	8,243

A.2 Implementation

Most of the implementation of this thesis has been done in `python`. CoreNLP, as well as the preprocessor from the Intergramm Project, are written in `java`. To call them from the python code, wrappers where written, that provided input data as temporary files, called the external programs and the output was read in. In that way, the external programs could be called like normal python functions.

Most of the implementation was done with the use of the `pandas` and `scikit-learn` library. All taggers implement the `BaseEstimator` class. Similarly, all special data

splitting implementations obeyed the sklearns interfaces. This allows to seamlessly use our implementation with the standard functions from `scikit-learn`.

The visualization was mainly done with `seaborn` and `pandas` directly and only modified with `matplotlib` as needed. Most of the tables in this thesis were also generated directly with `pandas`.

Colophon

Most of the figures where created using the python library `matplotlib` or its wrapper `seaborn`.

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties. I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Paderborn, December 2, 2019

Stefan Heid