The Hanover Tagger (Version 1.1.0)

Lemmatization, Morphological Analysis and POS Tagging in Python

Christian Wartena

Hochschule Hannover
Data|H – Institute for Applied Data Science Hannover
Bismarckstraße 2, 30173 Hannover

February 2023

HanTa, or Hanover Tagger, is an open source Python library for lemmatization and part of speech tagging. This document contains a description of the functionality, an introduction to the ideas and techniques used and some information on the annotated training data for Dutch, English and German.

Contents

1 Introduction

HanTa, or Hanover Tagger, is a Python program for lemmatization and part of speech tagging. The motivation for HanTa was to provide a Python program that is easy to install and easy to use for these tasks. In Natural Language Processing Classes for Library and Information Science (LIS) students we found that such a tool would be very helpful. We use the Natural Language Toolkit (NLTK, [Bir06]) to make students familiar with basic concepts of language processing, text and data mining. NLTK does not offer PoS tagging and lemmatization for German, the native language for most of our students. Since students often work on their own devices with different operation systems and settings, installation of other tools often turned out to be very time consuming.

Thus, the main goal of HanTa is neither to be faster, better or more innovative than existing solutions, but just easier to use. Consequently, HanTa is written in pure Python, without dependencies on non-standard libraries and with the possibility for installation from the Python Package Index.

HanTa in fact consists of three main components: a morphological analyzer, a part of speech tagger and a lemma generator. All three components are based on traditional machine learning techniques and have to be learned from training data. The morphological analyzer is the most innovative part of the system, using a Hidden Markov Model (HMM) to find the most likely sequences of morphemes for a given word. The PoS-tagger is a very classical second order HMM as can be found in every text book. There is one main difference to the standard implementation: the probabilities that a PoS tag is realized by a certain word is not computed from the corpus, but instead given by the morphological analysis. This might improve PoS tagging of unknown word forms at the price of less accurate results for common words. Finally, the lemma generator generates an lemma given the morphological analysis of the word. Again from the training data HanTa learns which morphemes are part of the lemma and what suffix eventually has to be added.

The main challenge in the end is to find or construct suited training data. For HanTa we generate training data from corpora with PoS tags and, ideally, lemmata for all words. We have written scripts with rather ad-hoc rules to split words into sequences of morphemes on which the morphological analyzer is trained.

The analysis that HanTa finally gives is based on the training data: it tries to reproduce the labels it has seen in the training data. Since the training data are partially also generated in this project, we will discuss this as well in this documentation.

Initially, HanTa was only build for German; now more languages are available as well. For German we could show that HanTa gives state of the art results for lemmatization and PoS tagging [War19]. Recently, more language models have been added. These have not yet been tested. Testing PoS tagging and lemmatization, beyond cross validation on the training data, is not a trivial task, since we need to find another annotated data set, that uses the same PoS tags, and the same policies for lemmatization as those used for annotating the training data.

HanTa is available as Python package from the Python Package Index (`https://pypi.org/project/HanTa/`) . Source code is also available at GitHub (`https://github.com/wartaal/HanTa`). At GitHub in addition some example files and scripts used to create the annotated training data, that are not part of the end user package, are available.

2 Getting Started

Since HaTa is available from the Python Package Index it can be easily installed on any system by once running the following command:

```
pip install HanTa
```

Now HanTa can be used immediately for all available languages in any Python program. For the use in other programming languages, HanTa can be run in a small server and called through a REST service. See `https://github.com/b310-digital/nlp-word-tagger-server` for details.[1]

When the package is installed, we can import the library, load a morphology model, and analyze a word:

```
from HanTa import HanoverTagger as ht

tagger = ht.HanoverTagger('morphmodel_ger.pgz')
lemma, pos = tagger.analyze('Fachmärkte')
```

The function `analyze` will return the lemma and the part of speech (PoS) of the word *Fachmärkte*.

Many words are ambiguous and can have more than one part of speech and more than one lemma. If we want to see all possibilities we can call the function `tag_word`.

```
tagger.tag_word('Angeln')
```

Now we see that there are five possible PoSs for this word: it can be a form of the verb *angeln* (to fish, to hook), or it can be the dative plural of the noun *Angel* (hinge) or it could be a proper name (In the training data a few names like Miguel Angel Fernandez are found). In case of the verb, the form could be a plural finite verb, the infinitive or the infinitive used as noun (*Das Angeln*), here tagged as NNI, in the TIGER corpus (that was used for training) as NN.

From the context in many cases it is immediately clear what sense is the correct one. Thus, if we want to analyze a whole sentence, or a word in the context of a sentence we should not use the above function, but hand over the whole sentence to the tagger.

```
tagger = ht.HanoverTagger('morphmodel_en.pgz')

words = ['Let', "'s", 'go', 'for', 'a', 'run', 'in', 'the', 'car', '.']
tagged_sent = tagger_en.tag_sent(words)
```

Now the result is a list of triples, each one consisting of the analyzed word, its lemma and its part of speech. In this example, the word *run* is correctly disambiguated as a noun.

[1] Thanks to Jannik Streek for implementing this solution

3 Methods

The HanTa package includes methods to train morphological models, to analyze words, and to tag words in a sentence with their most likely part of speech, The package includes a few trained models. Thus, for most users there is no need to train new models.

3.1 Application

To analyze words and sentences we use the class `HanoverTagger`. The constructor can be called either with a loaded model or with the name of a file that contains a model. For normal use only the latter is needed. Once a model is loaded the following methods can be called.

tag_word(*word, cutoff=5, casesensitive=True*)

returns a list of pairs consisting of a part of speech tag and a number. The number is the natural logarithm of the probability that an arbitrary word in the training data is that word with that PoS tag. Depending on the settings used for training, this is the real observed probability in the training data or it is the estimation based on the observed probabilities of the morphemes the word is composed of. For words that were seen less than 3 times, the probabilities are always estimated. For words found less than 10 times in the training data, observed probabilities are used for PoS tags given to that word in the data, and estimated probabilities for unobserved PoS tags.

The parameter `cutoff` can be used to limit the number of results. If set to 0, only the most likely PoS is returned. Otherwise, if set to a value v, only results are given with a probability of at least 2^v lower than the probability of the most likely word/tag pair. This restriction applies only to estimated probabilities.

The parameter `casesensitive` can be used to include case to predict the probability of a part of speech. Usually this is useful since in most languages upper case is used to indicate proper nouns or common nouns as well. If it is known that a sentence is written with all words capitalized or without capitalization at all, results might be more accurate when this parameter is set to False.

analyze(*word, pos='EMPTY', taglevel=1, casesensitive=True*)

analyzes a word and returns the analysis of the word and the most likely part of speech. If the parameter `pos` is used, it will try to analyze the word as the specified part of speech. E.g. the Dutch word *staat* (state or stands), will be analyzed as one morpheme, if we tell the method it should be a noun. If we require it to be a verb, it is analyzed as a stem *staa* and a suffix *t*. If no PoS is specified (or set to 'EMPTY') or if an invalid PoS is given, the function `tag_word` is called first to determine the most likely PoS.

The parameter `casesensitive` is handed over to the function `tag_word` to determine the PoS, if no PoS is given.

The parameter `taglevel` determines in what form the analysis is presented. It can be one of the integers 0,1,2 or 3. The behavior is as follows:

0 No analysis is given, just the PoS is returned

1 HanTa tries to generate the correct lemma and returns the lemma and PoS.

2 A string of morphemes separated by a '+'-sign and the PoS is returned.

3 A list of morphemes, each consisting of a string and tag is returned, again along with the PoS

Thus, if we analyze the English word *walking* at each of four levels, we get the following results, resp.:

```
VBG
('walk', 'VBG')
('walk+ing', 'VBG')
('walk', [('walk', 'VB'), ('ing', 'SUF_ING')], 'VBG')
```

Here the stem and the lemma are identical. If we do the same for the German word *wirft* (throws) we get:

```
VV(FIN)
('werfen', 'VV(FIN)')
('werf+t', 'VV(FIN)')
('werf', [('wirf', 'VV_VAR'), ('t', 'SUF_FIN')], 'VV(FIN)')
```

Here we see, that with level = 1 we get the correct lemma, with level 3 we get the (normalized) stem of the word. If derivational morphology is used, the stem also can consist of several morphemes. For the word *actualities* we get:

```
NNS
('actuality', 'NNS')
('actual+ity+es', 'NNS')
('actuality',
 [('actual', 'JJ'), ('iti', 'SUF_JJN_VAR'), ('es', 'SUF_NN_S')],
 'NNS')
```

tag_sentence(*sent, taglevel=1, casesensitive = True*)

analyzes a list of words and determines the most likely PoS for each word in the given sentence. The first parameter, sent, should be a list of strings. For the models currently distributed with the software, it is assumed that words are tokenized as is done by the tokenization function from the NLTK-package [Bir06]. Correct tokenization is especially important for the correct analysis of contracted words and apostrophes in English.

The parameter `casesensitive` is used as in the method `tag_word`.

The result of the method is a list with an analysis for each word. If the parameter `taglevel` is set to 0, only the PoS-tag for each word is returned. In all other cases a tuple, consisting of the original word, the analysis and the PoS is given. The analysis is either just the lemma (taglevel = 1), a string of morphemes separated by the '+'-sign (taglevel = 2) or a full list of morphemes exactly as returned by the method `analyze`.

If no lemmata or stems of a word are needed,`taglevel` should be set to 0 since the analysis is much faster if no word analyses have to be produced.

list_postags()

Gives a list of all PoS tags along with a few random chosen examples for each tag.

list_mtags()

Gives a list of all morpheme tags along with a few random chosen examples for each tag.

3.2 Training

For training a model the HanTa package provides the class `TrainHanoverTagger`. A typical script to train a model would look like the following:

```
from HanoverTagger import TrainHanoverTagger

trainer = TrainHanoverTagger()
datafile = open("nl/labeledmorph_dutch.csv", "r","utf-8")
trainer.load(datafile)
datafile.close()
trainer.train_model()
trainer.write_model('morphmodel_dutch.pgz')
```

3.2.1 Methods for training

The class `TrainHanoverTagger` provides the following functions:

load(*file*)

loads the training data. The argument `file` is the opened file.

train_model(*observed_values=True*)

trains the model. The optional parameter `observed_values` can be set to False if the sentence model should only use word probabilities estimated by the morphological model. If set to True, the sentence model uses probabilities observed in the training data and uses estimated probabilities for unseen words only. Usually better results are obtained when the value is set to True.

write_model(*filename*)

creates a file with the given name and writes the trained model to that file.

3.2.2 Format of the training data

The training data has to be a tabulator separated file with 7 columns. Each line represents one word. The following information has to be put in the columns:

1. Number of the sentence, or -1 if the word is not part of a sentence.
2. Word
3. Lemma
4. Stem
5. PoS tag

6. List of morphemes, in the same format as the output of the method `analyze` produces with `taglevel` set to 3. The list is given in square brackets. Each morpheme consists of a pair of the string and the tag of the morpheme. Both strings and morphemes have to be put between quotation marks.

7. Mapping (optional), a triple consisting of the tag and the string of the morpheme that should be replaced to obtain the lemma of the word. The third element is the replacement.

For English a few lines from the trainig data look as follows:

```
53161   I         i         i         PPSS    [('i', 'PPSS')]
53161   answered            answer    answer  VBD     [('answer', 'VB'), ('ed', 'SUF_VB_D')]

53161   readily   readily   readily   RB      [('readi', 'JJ_VAR'), ('ly', 'SUF_RB')] ('
    JJ_VAR', 'readi', 'ready')
53161   enough    enough    enough    QLP     [('enough', 'QLP')]
53161   ,         ,         ,         ,       [(',', ',')]
53161   recalling           recall    recall  VBG     [('recall', 'VB'), ('ing', 'SUF_ING')]

53161   her       her       her       PP$     [('her', 'PP$')]
53161   last      last      last      AP      [('last', 'AP')]
53161   visit     visit     visit     NN      [('visit', 'NN')]
```

In English we usually have no difference between the stem and the lemma. If the stem and the lemma are not the same, a line from the training data looks like the following line from the Dutch training data:

```
6259    afliep    aflopen   aflop     WW(pv,verl,ev)  [('af', 'PTK'), ('liep', 'WW_VAR(verl)'
    ), ('', 'END_WW(pv,verl,ev)')]    ('WW_VAR(verl)', 'liep', 'lop')
```

4 Background

The Hanover Tagger was presented at the Konvens conference in Erlangen in 2019 [War19]. Since then much was improved and optimized, but the core algorithm for morphological analysis is still the same. The following section is based on the text published in the conference proceedings.

4.1 Morphological Analysis

The basic idea underlying HanTa is that a word can be analyzed as a sequence of morphemes of various categories. Each morpheme category can be seen as a state in a Markov model. Now we have probabilities that a state follows a sequence of preceding states and probabilities that each state generates a part of the word. This general idea is illustrated in Figure 4.1.

Given a word $w = a_1 \dots a_n$ we try to find the most likely sequence of morpheme tags $s = t_1 \dots t_k$ that generates w. We cannot use a standard Hidden Markov Model and the Viterbi algorithm to find the most likely sequence s since we do not have a segmented list of output observations. However, the solution presented here is very similar to a Hidden Markov Model and the computation of the most likely tag sequence is almost identical to the Viterbi Algorithm. We define the most likely tag sequence s that generates w as

$$\max_{k, s \in T^k} \prod_{i=3}^{k} p(t_i \mid t_{i-2} t_{i-1}) \cdot p(a_{l_i m_i} | t_i) \tag{4.1}$$

where T is the set of all tags, $0 \leq l_i \leq m_i \leq n$ for each i and $w = a_{l_3 m_3} \cdot \ldots \cdot a_{l_k m_k}$.

Since here every state is dependent on two previous states we call the model a second order model and we have to add two start states to every sequence. We also add a final state that generates the empty string, to each tag sequence. We add one final state for each PoS tag. This will allow us, to compute the most likely tag sequence for each PoS.

Using dynamic programming we can find the optimal tag sequence efficiently. Using a first order model, for a string $w = a_1 \dots a_n$ we define the probability that a prefix $a_1 \dots a_j$ of w is

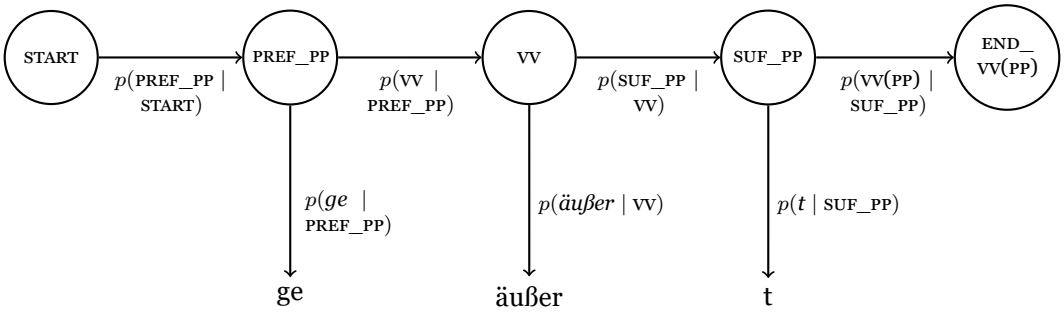

Figure 4.1: Markov chain generating the German word *geäußert* (voiced/expressed) .

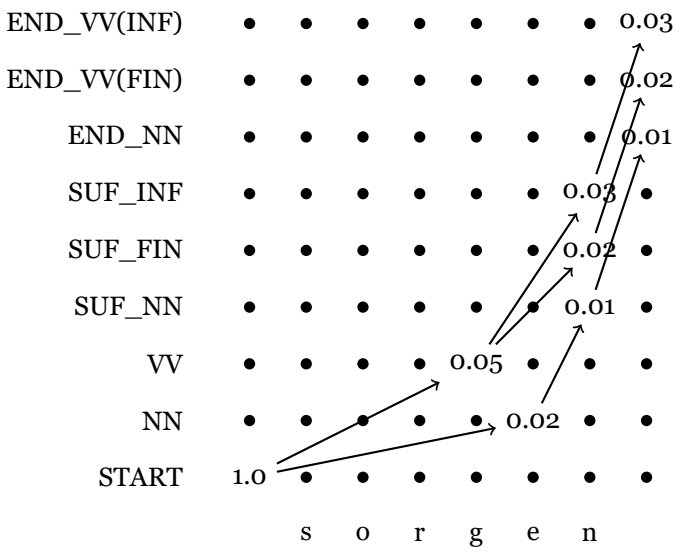

Figure 4.2: Trellis diagram for computing the most likely end state for the German word *sorgen* (worry).

tagged with a tag sequence in which the last tag is t as

$$\vartheta(t, j) = \max_{r,s,i} (\vartheta(s, i) \cdot p(a_{ij} \mid t) \cdot p(t \mid s)) \tag{4.2}$$

for each $t \in T$ and $2 \leq j \leq n$ where $s \in T$, $2 \leq i \leq j$, a_{ij} denotes the substring $a_i \ldots a_j$ from w and

$$\vartheta(t, 1) = p(t \mid \text{START}). \tag{4.3}$$

The equation can easily be extended for a second order model but the becomes slightly more complicated. When we compute $\vartheta(t, j)$ for each tag t and each position j, we get the well-known Trellis diagram (see Figure 4.2). When we extend the algorithm with a backpointer, we can easily find the optimal tag sequence for a given word.

Consider e.g. the word *Sorgen* (worry), that can either be the plural of the noun *Sorge*, the infinitive of the verb *sorgen*, or a finite form of the same verb. Now, for each of the corresponding final states we can compute the most likely tag sequence that generates the word *sorgen*. In our data we thus find the following tag sequences:

$$\begin{aligned} s_{\text{inf}} &= \text{None}, \text{START}, \text{VV}, \text{SUF\_INF}, \text{END\_VV(INF)} \\ s_{\text{fin}} &= \text{None}, \text{START}, \text{VV}, \text{SUF\_FIN}, \text{END\_VV(FIN)} \\ s_{\text{nn}} &= \text{None}, \text{START}, \text{NN}, \text{SUF\_NN}, \text{END\_NN} \end{aligned}$$

The probability that the sequence s_{nn} generates the word is computed as follows: $p(Sorgen, s_{\text{nn}}) = p(\text{NN} \mid \text{None}, \text{START}) \cdot p(\text{SUF\_NN} \mid \text{START}, \text{NN}) \cdot p(\text{END\_NN} \mid \text{NN}, \text{SUF\_NN}) \cdot p(\text{'sorge'} \mid \text{NN}) \cdot p(\text{'n'} \mid \text{SUF\_NN}) = e^{-1.60854} \cdot e^{-1.46985} \cdot e^{0.0} \cdot e^{-7.82129} \cdot e^{-1.43789} = e^{-12.33757}$. Similarly, we find $p(sorgen, s_{\text{fin}}) = e^{-10.77681}$ and $p(sorgen, s_{\text{inf}}) = e^{-10.63024}$.

Since there could be several sequences generating *sorgen* and ending in END_NN, the probability $p(Sorgen, s_{\text{nn}})$ is not the probability that the word is generated by a noun sequence. To compute that probability we would need an equivalent of the Forward Algorithm, that computes the sum of

all probabilities leading to one state instead of the maximum. However, in practice alternative paths turn out to be completely nonsense (since the model is highly over generating) or extremely unlikely (and thus do not change anything).

Given the large amount of training data for words, and the fact that for each substring we can assume that it is generated by one of the open class morphemes, we do not need any interpolation or smoothing and use the trigram probabilities directly.

Finally, we use also case information and multiply the found probability with the probability that a word of the found class is capitalized or not.

4.1.1 Unknown Words

For each string we can assume that it is an unseen instance of an open morpheme class. For each open morpheme class we will find a number of hapax legomena. Given the probability $p_{hap}(t)$ that the tag t is realized by a hapax legomenon, the program assumes that t is the tag of an open class if $p_{hap}(t) > 0.005$

For a morpheme $m = a_1 \dots a_n$ we can estimate the probability that an unseen morpheme is generated by a given morpheme tag using the probability that a morpheme ending on a given suffix is generated by that class. We compute these suffixes probabilities on infrequent morphemes, assuming that morphemes not observed in the test data are more similar to infrequent than frequent to morphemes. If not enough observations are available for suffixes of length n we use the probabilities for suffixes of length $n-1$. To compute the probabilities of the shorter suffixes we exclude all morphemes ending on one of the longer suffixes for which we had enough observations. E.g. if we use the probability $p(\text{noun} \mid ung)$, for bigrams we use $p(\text{noun} \mid ng$ and not $ung)$ rather than $p(\text{noun} \mid ng)$.

For longer unknown words we need to be sure, that an analysis using several known morphemes is preferred over the analysis as one unknown morpheme. Especially long nouns should be much more likely to be a noun compound, consisting of two or more known stems than being a completely unseen stem. Thus we also use the probability $p(n \mid t)$ that a morpheme of length n is generated by t. We compute this probability on infrequent morphemes again. Finally, we use the probability $p_{hap}(t)$ that the tag produces a hapax legomenon. Thus we approximate the probability of an unseen morpheme $m = a_1 \dots a_n$ given a tag t as

$$p(m \mid t) \approx p(a_{n-2}a_{n-1}a_n \mid t) \cdot p(n \mid t) \cdot p_{hap}(t). \tag{4.4}$$

4.2 POS Tagging

The usual way to analyze German or English is to start with part-of-speech tagging and then to analyze each word according to the found PoS. It is now tempting to investigate, whether the other way around works as well: instead of using observed probabilities we could use the probabilities as computed by the morphological analysis. To be precise: we computed $p(w, t)$ for a word and a tag before. In a standard trigram tagger (see e.g. [Bra00]) we need the probability $p(w \mid t)$. This probability can be computed easily by using the fact that $p(w \mid t) = \frac{p(w,t)}{p(t)}$.

The approach has the advantage that we get much better statistics for inflectional variants of infrequent words. On the other hand we lose a lot of information on specific word forms. For some words only certain forms are frequently used, and others are infrequent or even not existent. E.g. the noun *Ärger* (trouble, annoyance) does not have a plural form. Consequently, the form *ärgere* only can be a verb form (from *ärgern*, to annoy). Using the morphological analysis described above, we will nevertheless find an analysis as noun as well.

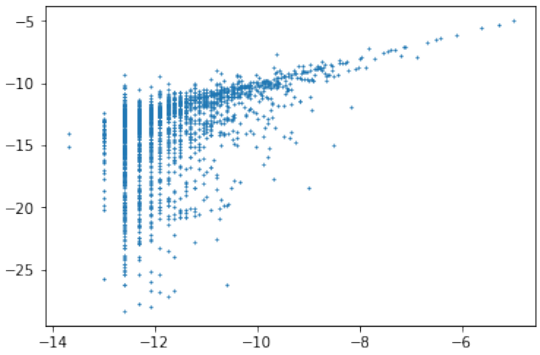

Figure 4.3: Observed (x-axis) versus computed (y-axis) logprobs for each pair (w, t) observed at least 3 times in the corpus.

We base the transition probabilities on trigram statistics over tags. Here we use linear interpolation to avoid zero probabilities and set the smoothed probability $p^*(t_n \mid t_{n-2}t_{n-1}) = 0.95 \cdot p(t_n \mid t_{n-2}t_{n-1}) + 0.04 * p(t_n \mid t_{n-1}) + 0.01 \cdot p(t_n)$.

4.2.1 Caching

In texts a few words cover most of the text. If a whole text has to be analyzed it makes no sense to analyze these words over and over again. HanTa therefore stores the probabilities $p(w, t)$ for the most frequent words and all possible tags.

For PoS tagging the observed values for $p(w \mid t)$ are of course much more accurate than the computed ones. Usually, the observed values for word forms occurring in the corpus are much higher than the computed ones while the computed values for all possible word forms not seen in the corpus is larger than zero. Figure 4.3 shows the relation between computed and observed probabilities. Each point is the logprob of a pair (w, t) observed at least 3 times in the TIGER corpus. The position of the point on the x-axis is the observed value for $p(w, t)$ the position on the y-axis is the computed value.

If not specified otherwise HanTa will store observed values instead of cached values. This makes PoS tagging more accurate but has one major drawback: a word that has two possible parts of speech, only one of which occurring in the training data, cannot be analyzed according to its unseen PoS. For an example see also section 2 on page 17.

As a compromise the following is done: for each word found 10 or more times in the corpus only the observed values are stored. For words observed at least 3 times and at most 9 times, the observed values are used but for every possible PoS that was not observed the computed value is used. For words seen less than 3 times no values are stored and probabilities computed at run time have to be used.

4.3 Lemmatization

The goal of lemmatization is normalization of word forms either for counting words or collecting other statistics or to look up a word in a dictionary. In the first case we can do whatever we want, for the purpose of dictionary lookup we have to follow the same conventions as those followed by the dictionary. We basically find two principles for lemmatization. The first one is that we remove all inflectional affixes but keep derivational affixes. Thus from the word *friendships*

we remove the plural *-s* but keep the derivational suffix *-ship* and obtain the lemma *friendship* which will be found as an entry in any English dictionary. The second principle is that the lemma needs to be an existing word form. For German and Dutch verbs the stem cannot be used without a suffix (except for the uncommon imperative). Thus here the infinitive form is used.

Both principles lead to a number of problems. In Dutch e.g. the formation of diminutives is very productive. Formation of the diminutive is a derivational and not an inflectional process, thus we should keep the diminutive form as lemma. However, for most words, these forms will not be found in a dictionary. In German the lemma of an adjective normally is just its stem. However, a few adjectives cannot be used without inflection (see also section 4.4), like the word *mittler* (middle). In the TIGER corpus the lemma for *mittler* thus is *mittlerer*. Similarly, for the adjective form *viertes* (fourth) the lemma *vierter* is used, since *viert* cannot be used without inflection except in the construction *zu viert* (in a group of four), in which case the lemma *viert* is used.

For the generation of training data we used lemmatized corpora and basically keep the lemmata provided there with a few simplifications as described in chapter 5.

Given the decomposition of a word in its morphemes, lemmatization is done by applying three simple rules that are not completely language independent but might work for a number of inflectional European languages:

1. Remove all morphemes that are not part of the stem given the PoS of the word;
2. Replace the last morpheme by its normalized form if that morpheme is present in the list of morphemes that have to be replaced;
3. Add a suffix to the stem using a list of suffices for each PoS and ending of the stem.

Which morphemes are part of the stem depends on the PoS. E.g. in the German word *ungeeignet* (unsuited), the prefix *ge*, labeled PREF_PP is part of the stem while it is not if the same word is tagged as past participle. Which morphemes are part of the stem is simply decided by counting during training of the model.

Necessary replacements of morphemes are annotated in the training data and simply collected and stored in the model. We assume that replacement is only necessary on the last morpheme. E.g. for a Dutch compound like *zakenreizen* (business trips) the lemma is *zakenreis* where the stem variant *reiz* is substituted by *reis*[1] but *zak* not by *zaak* .

Finally, a suffix might be added. For the stem of a German e.g. verb this can be either *-n* or *-en* depending on the last letters of the stem. In the training process these word ending / suffix combinations are simply learned by starting with a suffix for each word and then removing letters from the beginning of the stem until a conflict arises. In a few cases conflicts cannot be resolved and wrong lemmata will be generated, e.g. both German words *Muse* (plural: *Musen*) and *Museum* (plural: *Museen*) are assumed to have the stem *Muse*. Now it cannot be decided whether to add the suffix *-um* for the lemma or not.

4.4 Common Problems

The simplicity of HanTa's approach to morphological analysis is appealing, but also has a number of limitations. In this section we list some of these limitations along with some ideas how they eventually can be alleviated in future versions.

[1] linguistically speaking, the form *reiz* is here the normal form and the form *reis* is the variant formed by the phonological process of Auslautverhärtung. In HanTa we, nevertheless, always assume that the form used in the lemma is the main form

Overgeneration

The most sever problem is overgeneration. This is not fundamental problem of approach, but one of the training data. However, we do not have enough manually created training data and have to rely on training data that are generated using simple scripts from tagged corpora.

In most cases overgeneration doesn't cause much problems. E.g. in German adjectives can be used attributive and then always get some inflectional suffix. Adjectives also can be used predicatively and adverbially, without any suffices. A few adjectives cannot be used predicatively or adverbially and always need en inflection. An example is the word *mittler* (middle). We can say *Das mittlere Haus* (The middle most house) but not **Das Haus ist mittler* (The house is middle). Using the current model for German, HanTa has no problem analyzing *mittler* as an adjective (ADJD), thus causing the word *Mittler* (Moderator), that actually is a noun, to become artificially ambiguous.

The problem discussed above could be solved by introducing a new morpheme class of adjective stems that need an inflectional morpheme. Thus the model can be improved by refining the training data. However, it should be noted that the amount of training data should be increased if many small morpheme classes are introduced, in order to have enough examples for each class. A further drawback is, that introducing more classes usually also causes more annotation errors.

Another nice example of overgeneration is the formation of compounds in German. Two nouns can be combined to one noun. In many cases a short linking morpheme is inserted and in a few cases the final *e* of the first part is deleted. An example of deletion or elision is the word *Grenze* that appears in compounds without the final *e*, like in *Grenzfall* (border case, borderline) or *Grenzübergang* (border checkpoint). In the current German model this is handled by a second alternative stem for *Grenze*. However, the stem with the final *e* can still be used in the analysis of a compound. Thus HanTa analyzes *Grenzerinnen* (female border guards; actually found in the TIGER corpus) as *Grenze+Rinne+n* (border gutters).

Independence Assumption

Out basic assumption is that we can analyze a word as a sequence of morphemes that represents the states of a hidden Markov model. Now we have probabilities that a state follows a sequence of preceding states and a probability that a state generates a certain observable morpheme. We assume that the probabilities are independent, but this is not the case. Partially this again is a problem of the annotation. E.g. in English the third person singular of a verb is analyzed as a sequence of VB and SUF_VB_S. Now the state VB generates some verb stem and the state SUF_VB_S generate either 's' or 'es'. These two choices are obviously not independent. The problem again could be solved by using more specific states (or morpheme classes).

However, not all cases can be handled by more detailed annotation. Let us look again at German compounds. If we have a sequence of two noun stems, the choice for the realization of each noun is in our model independent. However, if the first noun is e.g. *Bauch* (belly, stomach) now there are a few very likely candidates for the second word (like *Schmerz* (pain) or *Gefühl* (feeling)) and most other nouns are possible but highly unlikely. If the compound *Bauchschmerz* is formed, we can either go to a final state or continue with a plural suffix. Both choices are possible, but the plural ending is much more likely, since the word *Bauchschmerz* is almost only used in plural.

Markov-Assumption

A very fundamental assumption underlying hidden Markov models is the Markov assumption, saying that the probability of a state is only dependent on a fixed (small) number n of preceding states. In HanTa we set $n = 2$. In most cases this is unproblematic, in a few cases it is a bit tricky and in at least one example it is problematic. In Dutch and German a past participle is

usually formed by the combination of a prefix and a suffix. Let us consider the participle of *bremsen* (brake, retard): *gebremst*, consisting of the prefix *ge*, the stem *brems* and the suffix *t*. In HanTa this corresponds to the states PREF_PP, VV, SUF_PP and the final state END_VV(PP). Since we labeled the suffix not just as a verb suffix but as a suffix for the past participle, no problem arises: If we see the suffix *t* it is clear that it is not the third person singular suffix but the past participle suffix. When choosing the final state we cannot see the prefix anymore, but seeing the state SUF_PP is enough to decide that END_VV(PP) has to be the final state.

Past participles are strange words. In a construction with an auxiliary verb they are usually considered as an inflected form of the verb, tagged with VVPP, VV(PP) or something similar and the infinitive of the verb is given as lemma (see e.g. [CHSSZE05, p. 13]). Past participles can also behave like adjectives and get adjectival suffixes. Problematic is the case in which a participle is used as an uninflected predicative adjective. Here we can see no difference between the participle and the derived uninflected adjective . Thus the same word is usually considered to be a past participle in example 1a and an adjective in example 1b[2]

(1) a. Im Oval wird hingegen kaum **gebremst**.
In the oval, on the other hand, there is hardly any braking.

b. Der Franzose fliegt ab [...] und schlägt **gebremst** in die Bande ein.
The Frenchman flies off [...] and hits the railing slowed down.

The correct tag here can be assigned only on the base of the context of the word.

The derived adjective can be preceded by the negating prefix *un-*. The past participle cannot. In the first example we thus cannot replace *gebremst* by *ungebremst*. In the second example the sentence even becomes more fluent when we do so.

(2) a. *Im Oval wird hingegen kaum **ungebremst**.

b. Der Franzose fliegt ab [...] und schlägt **ungebremst** in die Bande ein.

In HanTa *ungebremst* is analyzed as `[('un', 'PREF_NEG'), ('ge', 'PREF_PP'), ('brems', 'VV'), ('t', 'SUF_PP')]`. After the state SUF_PP either the final state END_VV(PP) or the final state textscend_adj(d) can follow. However, if the sequence starts with the state PREF_NEG only the latter one should be possible, but the model cannot look back further than the state PREF_PP.

In many cases HanTa nevertheless will produce the correct analysis, because it is forced by context or the correct PoS is cached for words frequently occurring in the training data. In some cases, however, the negated word is tagged as a pats participle which is completely wrong.

Long words

Often long words that are composed of a long sequence of morphemes turn out to be problematic. Since all probabilities are multiplied, the probability of sequence drops very fast with an increasing number of states. At the same time, we always have to consider the possibility that the word in an unknown word that cannot been analyzed as a sequence of known morphemes. This is especially the case for proper nouns, but also for common nouns. At some point the probability for the latter solution becomes higher than the analytic solution and a wrong decision is made.

It is not a solution to try whether a word can be analyzed without unknown morphemes first and allow the assumption of unknown parts only in a second try, since this would result in many very obscure and unlikely morpheme sequences for many words that were not seen in the training data.

[2] Both examples are taken from the Deutsche Nachrichten-Korpus (`https://corpora.uni-leipzig.de/de/res?corpusId=deu_news_2021&word=gebremst`)

The problem is alleviated somewhat by always allowing unknown morphemes in the forward algorithm to determine the correct PoS. In the Viterbi algorithm that is used to determine the optimal sequence of states, once the PoS is already known, we first try to reach the desired end state without unknown morphemes.

Over fitting

For PoS tagging morphological analysis is a great option for words that were not seen in the training data. However, if a word is present in the training data results are much better when we use the observed probabilities for the possible part of speech tags for that word. Above we considered the past participle *gebremst*. In a corpus we will find only very few examples in which this word is tagged as uninflected adjective and in almost all cases it is tagged as part participle. In another case like *gemischt* (mixed), the adjective use is much more common. These differences cannot be derived from the morphological structure. Thus in the models HanTa uses for words that were observed in the training data, these probabilities are stored and not estimated from the composing morphemes. This also excludes wrong analyses, like the analysis of *ungestört* (undisturbed) as a past participle as we have seen above.

The use of the probabilities from the training data also prevents HanTa from tagging ambiguous words correctly that were only found with one PoS in the training data. In Germany, recently people prefer to us nominalized adjectives, especially present participles, over other nouns to refer to persons, since the plural form of nominalized nouns does not carry gender information and thus includes female, male and diverse people. E.g. the word *Läufer* (runner) refers to a male runner, whereas we can use *Läuferin* for a female athlete. Instead of using both words to refer to a group of male and female runners (*Läuferinnen und Läufer*) or using some artificial form, many people consider using the present participle as an elegant alternative and say *die Laufenden* (the running ones). In HanTa this could in principle be analyzed correctly using the PoS tag NNA for a nominalized adjective:

```
>>> tagger.analyze('Laufenden',taglevel=3,pos = 'NNA')
('laufend', [('lauf', 'VV'), ('end', 'PRESPART'), ('en', 'SUF_ADJ')], 'NNA')
```

In the training data, however, the word *laufenden* is quite frequently and only found as an adjective (being part of some collocations, like *laufenden Kosten* (running costs)). Thus by using the observed values from the training data the model is overfitting on that data and is not able to tag *Die Laufenden* correctly.

Granularity of tagging

In general we would expect that more detailed annotations could help to improve the quality of the learned model and reduce overgeneration. However, detailed annotation leads to small classes for which eventually not enough examples are present to compute reliable statistics. HanTa has no mechanisms to derive properties or statistics from more general classes. E.g. in the German model currently there is only one class of noun stems. HanTa learns that a noun stem always can be followed by a diminutive suffix, which is not very frequent in German. If we introduce three genera and distinguish between countable and non countable nouns, this has to be learned for six different noun classes.

Robustness of lemmatization

While PoS tagging and morphological analysis is quite robust against errors in the annotation of the training data, lemmatization is not. E.g., in order to generate the correct lemma *Zopf* for

the German plural form *Zöpfe* we have to observe this in the training data. In a corpus of 1 million words, many of such replacements occur only once. Thus HanTa assumes that this replacement has to be made for lemmatization if it is observed once, making the algorithm vulnerable for errors in the training data. The algorithm could be made somewhat more robust if we would count how often it is substituted and how often it is not. However, this type of error is not observed much and we preferred to correct the training data in a few cases instead.

5 Tag Sets and Training Data

5.1 Dutch

To train the morphology model for Dutch we used the Sonar-1 Corpus [Oos12, ORHS13], that was automatically tagged and manually corrected.

The tag set used in the Sonar Corpus is the CGN tag set [Eyn04]. This tag set is much more detailed than the Tübingen-Stuttgart tag set used for German. The tags in the CGN tag set are composed from a main tag and a list of feature values. In total over 300 combinations are possible. An overview of all possible tags is e.g. given by `https://universaldependencies.org/tagset-conversion/nl-cgn-uposf.html`.

In HanTa there is no possibility to take any advantage of the feature structure and each combination of a tag and features has to be treated as an atomic tag. Thus the large number of tags becomes a challenge for training the model. On the other hand side the detailed information also makes it possible to easily exclude impossible sequences, reduce the degree of overgeneration of the model and exclude spurious analyses. The problems that remain are: 1) a number of features are purely syntactical and cannot be derived from the morphological structure, like e.g. the feature indicating whether a preposition is used pre- or postnominal, and 2) underspecified features, like the gender of some nouns that can have both genders. The tags for the morphemes are largely determined by the tags of the words.

HanTa mainly deals with inflectional morphology. A few cases of highly productive derivational morphology are also included in order to enable better analysis of word not present in the training data. These cases are:

- Diminutives (of nouns)
- Comparatives (of adjectives)
- Present and past participles (that often are treated like inflectional morphology)
- Noun-noun compounds
- Verb-noun compounds
- Noun-Adjective compounds
- Nominalization of adjectives with the suffix *-heid*
- Nominalization of verbs with the suffix *-ing*
- Negated-adjectives with *on-*

In all cases the training data were generated semi-automatically from the original Sonar-annotations. This means that also strongly lexicalized derivations that are not perceived as such anymore are decomposed. On the other hand side many derived words are not decomposed since not enough evidence for the existence of the base parts could be found in the corpus. Finally, separable and non-separable prefixes of verbs are marked as morphemes in the training data.

In the training data a word is only split in two parts if enough evidence is found in the training data that both parts really exist. Thus, in some cases a word is not split in the training data. E.g. the word *luchtvochtigheid* ((air) humidity) is not split since the noun *vochtigheid* does not occur in the dataset. Thus the whole word is added as a atomic noun to the model and the tagger will not show the morphemes that make up this word. In contrast, if we let the tagger analyze the word *bodemvochtigheid* (soil moisture/humidity of the soil), HanTa will analyze it correctly as bodem+vochtig+heid, since the whole word is not in the training data and HanTa is forced to make a morphological analysis.

5.1.1 Creating training data

The Sonar-1 Corpus cannot be distributed with the HanTa software, but it is available on request from the Instituut voor de Nederlandse taal (`https://taalmaterialen.ivdnt.org/download/tstc-sonar-corpus/`). Furthermore a list of frequent words from subtitles [KBN10] is used, that can be obtained from the University of Gent (`http://crr.ugent.be/programs-data/subtitle-frequencies/subtlex-nl`) .

Since there are some typos and other errors in the Sonar corpus, first run the script `RepairSonar.py` to correct some of these errors. Also sentences containing too many foreign words are removed. The script finally replaces the annotation *deeleigen* for the part of a proper name by `N(eigen,…)` since the distinction between a proper noun and a proper noun that is part of a name is purely syntactical and cannot be derived from the morphology.

The script `create_train_data_nl.py` finally can be used to create training data for HanTa.

5.1.2 Strong verbs, ablaut and irregular words

One of the main assumptions underlying HanTa is that a word can be analyzed as a sequence of morphemes and that both essential morpho-syntactic information and the lemma of the word can be found and reconstructed from that sequence of morphemes. However, phonological processes can change the morphemes in the context of other morphemes. Moreover, in many languages among which German, Durch and English we have vowel changes, also known as Ablaut, that are used to build some word forms. We capture both phenomena by using variants of morphemes and a mapping of the variant to the main form of the morpheme. This mapping is used only to reconstruct the lemma. E.g. the morpheme *liep* is stored as a variant of the verb stem *lop* (stem of to walk). This mechanism is applied automatically when generating the training data and thus enables the tagger to handle any kind of strange deviations between a word form and the given lemma. In the case of Dutch this mechanism also handles the addition of a diaeresis or trema to a vowel. In Dutch the trema is used to mark the border between two morphemes, if a confusing sequence of vowels makes the border hard to find. E.g. if we add the prefix *ge* for a past participle to the verb stem *eer* (praise), a trema is used to mark the beginning of the verb stem: *geëerd*. HanTa now simply stores *ëer* as a variant for the verb stem *eer*.

5.1.3 Orthography

Due to the rules of dutch orthography many stems of nouns, verbs and adjectives have different spellings depending on the subsequent phonemes. E.g. the stem *groot* (big, large) is written *groot* if nothing or a consonant follows (as in the superlative *grootst*) and written as *grot* when an inflectional *e* is following (*grote*). In someway the model needs to capture this phenomenon since it should be able to generate the lemma *groot* for the word *grote*. We decided to use the same mechanism as used to capture Ablaut-phenomena (in Dutch and German). We always take the variant used in the lemma of the word as the main variant. Thus for the adjective *groot* this is

groot , but for the verb *lopen* (to walk) it is *lop* and for the verb *vallen* (to fall) it is *vall*. If another variant is used we store a mapping to the main variant. Though this works in the large majority of all cases, it has a clear drawback: suppose the noun *staak* (infrequent word for pole) is not seen in the training data. If HanTa now successfully analyses the word *staken* as a plural noun composed from an (unkown) stem *stak* and plural suffix *-en* it will not be able to generate the correct lemma. The situation is complicated by the fact that there also might be a vowel change in the plural: the lemma for *dagen* (days) is *dag* (irregular) while the lemma *zagen* (saws) is *zaag* (regular). Here some language specific rules would be needed to handle unknown words correctly.

5.2 English

To train a model for English we use the Brown corpus for PoS tags and Wordnet [Mil95] to obtain lemmata for the words. Since both resources are included in the NLTK library [Bir06] we do not have to download any resources, but directly generate the training data using the NLTK library.

Some changes are made to the tokenization of words with apostrophes and contractions in order to work with the same tokenization as produced by the NLTK tokenizer.

In the training data all inflectional morphology is handled as well as some cases of derivational morphology, especially:

- Comparatives (of adjectives)
- Present and past participles (that often are treated like inflectional morphology)
- Derivation of adverbs from adjectives with the suffix *-ly*
- Derivation of nouns from adjectives with the suffix *-ness*
- Derivation of nouns from adjectives with the suffix *-ity*
- Derivation of adjectives from verbs with the suffix *-able*
- Negated-adjectives with *un-*

5.3 German

The German model was trained on the TIGER corpus and some additional word lists. Hence it uses the STTS tag set [STST99, CHSSZE05] with a few changes. Most notable changes are the introduction of the tags NNA and NNI for nominalized adjectives and infinitives, respectively, both tagged as NN in TIGER.

In the annotation most inflectional morphology is included and some productive cases of derivational morphology are handled as well. These cases are:

- Diminutives (of nouns)
- Comparatives (of adjectives)
- Present and past participles (that often are treated like inflectional morphology)
- Noun-noun compounds
- Verb-noun compounds

- Negated-adjectives with *un-*
- Adjective formation form verbs with the suffix *-bar*

The lemma of comparatives is the base form of the adjective. In all other cases the lemma is the normalized form of the derived word.

Currently, the tags for the morphemes are quite underspecified and do not contain any information on genus, case, number etc. This might be changed in future.

5.3.1 Creating training data

The German model that is distributed together with the software was trained on the words of TIGER corpus. Due to the license of this corpus we cannot publish the training data derived from TIGER. To reconstruct the training data one can download the TIGER corpus from `https://www.ims.uni-stuttgart.de/en/research/resources/corpora/tiger/` . The version used is release 2.2 in the CONLL09 format. The corpus has a number of errors and in a few cases, discussed below, we decided to use different annotations. To repair and change the annotations run the script `repair_tiger.py`. Furthermore we use a list of the 100.000 most frequent words from the DeReKo Corpus that can be downloaded from `https://www.ids-mannheim.de/digspra/kl/projekte/methoden/derewo/` and a list of irregular nouns that is part of the Github repository. (From the DeReKo list only verbs are used since distinction between common nouns and proper nouns in DeReKo is not consistent with TIGER.)

The script `create_train_data_ger.py` now will read these files and use a number of heuristic rules to create the data for training HanTa.

5.3.2 Changes from TIGER annotations and other remarks

Nouns

As mentioned above we introduced two new tags for nouns. In the first place we have a new tag for nominalized infinitives. Infinitives can be used as nouns and then even can get some nominal inflection. Nevertheless, they behave quite different from other nouns. Thus we tag them with the tag NNI.

The second new tag is for nominalized adjectives. Nominalized adjectives are very frequent in German. In TIGER they are tagged as normal nouns (NN) but morphologically they behave completely as adjectives and get weak and strong adjective inflection. For these forms we have introduced the tag NNA.

The lemmatization of nominalized adjectives in the TIGER guidelines [CHSSZE05] is quite complicated. Nouns derived from normal adjectives (i.e. adjectives that are not derived themselves from another word) get as lemma the the weak nominative masculine inflection (*Tote, Grüne, Verantwortliche*). If the adjective is a present participle, the uninflected form is used as lemma (*vorsitzend, studierend, anwesend*). Here we find quite a lot of errors in the annotation. If the nominalized present participle becomes the head of a compound, usually the strong form nominative masculine is used, but sometimes also the weak form is found (*Ausschußvorsitzende, Außenstehender, Handlungsreisender, Asylsuchender)*. For nouns derived from past participles, the strong form nominative masculine is used as lemma but the weak form sometimes is used as well (*Unrasierte,Verlorener,Gefangener,Verborgener,Angeklagter,Unvorhergesehene*). We decided to unify all these cases and use the weak masculine nominative inflection for all nominalized adjectives.

Adjectives

The lemma of an adjective in general is just its stem. However, in German there are a few adjectives that cannot be used only attributively, like *andere*. Attributive adjectives always have an inflectional suffix. Thus the stem of such adjectives cannot be used in isolation. The TIGER annotation guidelines here require to use the nominative masculine form with strong inflection. In the TIGER corpus this turns out to be a source of errors. Sometimes weak inflection is added or inflected forms of adjectives that can be used predicatively are given as lemma.

Most adjectives that cannot be used predicatively, however can be used adverbially, and thus can show up without inflection. These adjectives in TIGER also get strong masculine inflection for the lemma. E.g. the words *Schrödersch*, *nächst* and *zweithöchst* get lemmata *Schröderscher*, *nächster* and *zweithöchster*, respectively, since they cannot be used predicatively:

(3) a. *Die Politik is schrödersch.

b. *Die Aufgabe ist nächst.

c. *Dieser Turm ist zweithöchst.

However, each of these words can be used perfectly adverbially:

(4) a. Darin, dass sich hinter dem schrödersch kalauernden Titel ein zunächst eher locker, zunehmend aber besser verknüpfter Themenabend verbarg, der vor allem Schröders eigene Generation ins Visier nahm. [1]

b. Dann werden die Patienten halt in das nächst mögliche Krankenhaus gefahren [2].

c. Die Schweiz ist nach den USA das zweithöchst versicherte Land. [3]

Since it is hard to decide which adjectives really cannot be used without inflection, we decided to change the lemma of all adjectives to the stem of the adjective. In a few cases this gives slightly strange lemmata (like *ander* (other), but it makes the annotation much more consistent and easier to process.

Verbs

Tagging and lemmatization of finite verbs is quite unproblematic. Some problems arise with infinitives and participles. We discussed the infinitives that can be used as nouns already above. Present participles are always treated as derived adjectives and get as lemma the uninflected form of the present participle. Past participles are seen as verb forms if thy are used together with an auxiliary verb. In that case they get the infinitive of the verb as lemma. In all other cases they are seen as derived adjectives (or nouns derived from the adjective). The lemma than is the uninflected past participle.

[1] From: Neue Westphälische, 2013, febr. 16th, https://www.nw.de/lokal/kreis_guetersloh/rheda_wiedenbrueck/7902230_Der-Comedystar-als-Moralist.html

[2] From www.focus.de retrieved via https://corpora.uni-leipzig.de/de/res?corpusId=deu_news_2021&word=n\%C3\%A4chst

[3] From: Nebelspalter : das Humor- und Satire-Magazin, 106(1980), 47. retrieved from https://www.e-periodica.ch/cntmng?pid=neb-001\%3A1980\%3A106\%3A\%3A3715

Bibliography

[Bir06] Steven Bird. NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, Sydney, Australia, July 2006. Association for Computational Linguistics.

[Bra00] Thorsten Brants. Tnt: A statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, ANLC '00, pages 224–231, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[CHSSZE05] Berthold Crysmann, Silvia Hansen-Schirra, George Smith, and Dorothea Ziegler-Eisele. Tiger morphologie-annotationsschema, 2005.

[Eyn04] Frank van Eynde. Part of speech tagging en lemmatisering van het corpus gesproken nederlands, 2004.

[KBN10] E. Keuleers, M. Brysbaert, and B. New. SUBTLEX-NL: A new frequency measure for Dutch words based on film subtitles. *Behavior Research Methods*, 42(3):643–650, 2010.

[Mil95] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, nov 1995.

[Oos12] Nelleke Oostdijk. Sonar-1. corpus van 1 miljoen woorden verrijkt met diverse semantische annotaties. 2012.

[ORHS13] Nelleke Oostdijk, Martin Reynaert, Véronique Hoste, and Ineke Schuurman. The construction of a 500-million-word reference corpus of contemporary written dutch. In *Essential speech and language technology for Dutch*, pages 219–247. Springer, Berlin, Heidelberg, 2013.

[STST99] Anne Schiller, Simone Teufel, Christine Stöckert, and Christine Thielen. Guidelines für das Tagging deutscher Textkorpora mit STTS, 1999.

[War19] Christian Wartena. A probabilistic morphology model for german lemmatization. In *Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019): Long Papers*, pages 40–49, Erlangen, Germany, 2019. German Society for Computational Linguistics & Language Technology.