# Optimization of the Dutch Corpus of Contemporary and late Modern Periodicals (C-CLAMP)

Julie Nijs

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Speech and Language Technology

**Thesis supervisor:**
Prof. dr. Frank Van Eynde

**Assessor:**
Prof. dr. Vincent Vandeghinste

**Mentors:**
Prof. dr. Freek Van de Velde
Prof. dr. Dirk Speelman
Jozefien Piersoul

Academic year 2024 – 2025

© Copyright KU Leuven

Without written permission of the thesis supervisor and the author it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to the Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 or by email `info@cs.kuleuven.be`.

A written permission of the thesis supervisor is also required to use the methods, products, schematics and programmes described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

# Contents

# Abstract

The Dutch Corpus of Contemporary and late Modern Periodicals (C-CLAMP) is a corpus that consists of a collection of articles from 13 cultural or literary periodicals published in Flanders and The Netherlands. It is a historical corpus, containing texts from the period between 1837 and 1999. The optimization of historical corpora is often hard because of the outdated spelling and the spelling variation that is present in historical texts. The goal of this project is to optimize the C-CLAMP corpus. The end result is a corpus that is completely Part-of-Speech tagged and lemmatized.

First, I made sure that the corpus material is clean and does not contain any doubles. I also performed foreign language recognition on the corpus, using a pre-trained fastText model. The precision of the implementation of this model is 0.947. Second, I used a vector approach combined with a nearest neighbors and edit distance algorithm to normalize the spelling of the corpus. This approach has an accuracy of 0.820. Both the foreign language recognition step and the spelling normalization step are performed because it will help with the last step: PoS-tagging and lemmatization. This last step was done with Frog. The accuracy for the PoS-tagging is 97,42% and the accuracy for the lemmatization is 98,34%.

# Chapter 1

# Introduction

The goal of the project is to optimize the Dutch Corpus of Contemporary and late Modern Periodicals (C-CLAMP). The corpus consists of a collection of articles from 13 cultural or literary periodicals published in Flanders and The Netherlands. The texts stem from the period between 1837 and 1999. The corpus contains a total of 198,067,908 tokens. This project aims to first clean up the corpus and subsequently Part-of-Speech tag (PoS-tag) and lemmatize it as correctly as possible. In order to fulfill these aims, three tasks need to be carried out: 1) cleaning up the material, 2) spelling normalization and 3) PoS-tagging and lemmatization.

The first task consists of cleaning up the material. This pre-processing step is described in section 2. It is important that the corpus does not contain any double texts (section 2.1) and that the material is clean to work with, meaning that the material should contain as little noise as possible (section 2.2). The corpus contains Dutch texts, but these texts may contain words or whole sentences that are not Dutch. This is the case, for example, when authors cite other works written in foreign languages. Users of the corpus will often pose research questions in which the foreign language material is not useful. Furthermore, this foreign language material may pose a problem for the PoS-tagging and lemmatization, because the tagger is trained on Dutch data. It is therefore important to perform foreign language recognition on the corpus prior to PoS-tagging and lemmatizing (section 2.3).

The second task is the normalization of the spelling and is explained in section 3. It is well known that spelling normalization improves the results of the PoS-tagging and lemmatization of historical texts (van der Goot, Plank, & Nissim, 2017). I will use a vector based approach that generates nearest neighbors combined with an edit distance algorithm.

The last task is the PoS-tagging and lemmatization of the corpus. This part is described in section 4. After the necessary preparation the corpus is ready to be tagged and lemmatized. I consider both the Frog pipeline and a BERT model for the PoS-tagging. The lemmatization will be done with Frog.

# Chapter 2

# Cleaning up the material

## 1 Elimination of double articles

An easy way to identify if two texts are the same is by comparing the first couple of lines in the texts. I wrote a Python script that loops through all the .txt-files in the corpus and stores the first 20 lines of each file in a dictionary. I chose 20 lines because sometimes two texts may have the same introductory lines, but otherwise be very different texts. So only comparing the first five lines, for example, returned a lot of false positives. The dictionary then stores the file names as keys and the lines as values. Afterwards the script searches through the dictionary for keys that have the same values. It then prints a list of these keys to the console. It is very important that no useful material gets lost. That's why I decided to check and remove the double articles manually. This way I am certain that the articles, given as output by the script, are real doubles and do not simply have the same opening lines. Sometimes two articles are double, but one is shorter than the other. In this case I deleted the shortest article. In total 151 articles were removed.

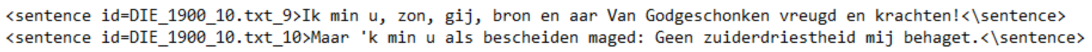
## 2 Eliminating noise

The corpus contained some unwanted noise, such as double punctuation or spaces. I wrote a Python script that loops over all the files in the corpus and cleans up each .txt-file, using regular expressions. More specifically the script does the following things:

- remove lines that only contain a full stop;
- remove existing tags, both sentence tags and foreign language tags;
- remove double punctuation and double spaces;
- standardize the quotation marks;
- remove numbers attached to words;
- place new sentence tags on each line.

Some lines consisted of only a full stop. Such lines contain no relevant information and where removed. Since each line contains one sentence and each line has a numbered sentence tag, the removal of lines causes the numbering to become incorrect. That's why I removed all the sentence tags and added them again. The foreign language material in the corpus has already been tagged using Polyglot. However, the results were unsatisfactory and so the foreign language recognition task has to be carried out again. This means that I had to remove the previous foreign language tags. Next, I removed double punctuation and double spaces and standardized the quotation marks to the same Unicode. Lastly, I removed numbers that are attached to words. When compiling the corpus from DBNL, the footnotes attached to sentences were also adopted. This resulted in numbers that are attached to some words, which now have been removed.

After this step the corpus looks as follows:



```
<sentence id=DIE_1900_10.txt_9>Ik min u, zon, gij, bron en aar Van Godgeschonken vreugd en krachten!<\sentence>
<sentence id=DIE_1900_10.txt_10>Maar 'k min u als bescheiden maged: Geen zuiderdriestheid mij behaget.<\sentence>
```

Each sentence has an opening tag and a closing tag. The opening tag contains the sentence id, which shows an abbreviation of the name of the periodical, the year in which the text was written, the number of the article and the number of the line. The example shows two sentences from the periodical *Dietsche Warande en Belfort*, written in 1900. It is the tenth article from that periodical in that year and the lines displayed are lines nine and ten.

## 3 Automatic recognition of foreign languages

C-CLAMP is a corpus of Dutch texts, but these texts sometimes contain words or sentences written in foreign languages. The reason is that the authors of the texts often cite from texts written in other languages. The foreign language material should not be lemmatized or PoS-tagged like the Dutch material for two reasons. The first reason is that a Dutch lemmatizer or tagger will not work properly on the foreign language material. The second reason is that users of the corpus may not always be interested in the foreign language material. There is a need to easily identify and filter this material. To do this each foreign word will receive a tag: __FL__.

A pre-trained model for foreign language recognition is available through the fastText library. FastText is an open-source library which allows users to train models for text representation and text classification. Some pre-trained models are readily available, for example word vector models for 157 languages. For the task at hand I have used the pre-trained language identification model (Joulin et al., 2016). The model was trained on data from Wikipedia, Tatoeba and SETime and is able to distinguish 176 languages. FastText was trained on UTF-8 input, and can consequently deal readily with C-CLAMP, which uses the same encoding standard. The model takes some text (one or multiple words/sentences) as input and gives as output one or more languages, that the model thinks are the most likely languages, as well as a confidence score for each language. The model can be set to output as many languages as needed. There are two versions of the model: one which is faster and slightly more accurate, but has a file size of 126MB. The other one is a compressed version, with a smaller file size of 917kB, but less accurate. I used the first version, because it is more accurate and faster.

I wrote a script that loads the fastText model and reads the corpus one file at a time. Before making a prediction, the line is first cleaned. Punctuation and brackets are removed so they do not influence the prediction. I let the model predict one language for each line in the file. If the line is recognized as not being Dutch, the script then goes over all the words in the line and predicts for each word three possible languages. If one of these languages is Dutch the word is left alone, otherwise it receives a tag. I chose to implement the model in this way because the model is sometimes confused between languages that are related to each other. Simply letting the model decide for each word separately would have led to a lot of errors. Dutch words incorrectly receiving a tag is a primary concern here. In the way that the model is implemented now there are two ways to mitigate this. The prediction per line is the first buffer: if the model thinks that the line is a Dutch sentence it is left alone and no Dutch words will be tagged incorrectly. This also implies that if only a few foreign words are present in a sentence that otherwise only contains Dutch words, the model is likely to still categorize the sentence as Dutch and so these words will not receive a tag. The second buffer is the prediction of three languages on the word-level. By predicting three languages it is possible to see if the model has doubts whether the word is actually Dutch or something else. If there is some doubt (one of the predicted languages is Dutch), then the word will not receive a tag. Another way to decide which words should receive a tag is to take into account how confident the model is of the predictions it makes. However, this idea was abandoned because the confidence of the model is sometimes very low, even though the output is correct or vice versa.

As already mentioned, an important criterion is that Dutch words should not be tagged. Since it is a Dutch corpus, we are interested in the Dutch words. Therefore it is preferable to have ignored some foreign language words than to tag Dutch words as foreign. In other words, in this case it is better to have false negatives (foreign words that are not tagged) than to have false positives (Dutch words that are incorrectly tagged as foreign). Given this criterion, I have chosen to compute the precision, using the following formula:

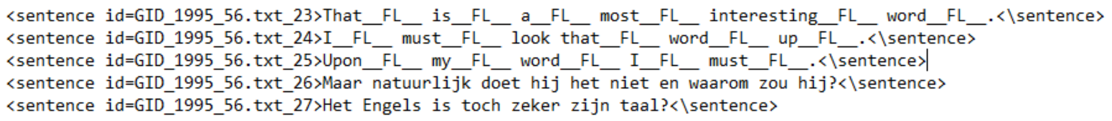
$$Precision = \frac{true\ positives}{(true\ positives\ + false\ positives)}$$

I have validated the model on a small sample of the corpus. The corpus is made up of 13 different Flemish and Dutch periodicals. In order to get an overview of how well the model performs on the whole corpus, I made a test set by randomly picking one text from each periodical. I took into account that there is sufficient variation in the years in which the texts are written. I manually went through the sample to count the true positives and the false positives. Names of people are counted separately, because it is not problematic that they receive a tag but it is not necessary either. There is also a residual group that consists of abbreviations and roman numerals, among others. This group is treated in the same way as the names. Older Dutch words that are tagged are considered to be false positives, because these are still Dutch. It is however not a surprise that the model makes mistakes on them, since the model is trained on contemporary Dutch. Table 1 contains the results of the test set. The tagger performs reasonably well. Some texts have no errors at all, while the worst tagged text has a precision of 0.766. The total precision on the test set is 0.947, which is a good result.

| Text | TP | FP | Precision | Names | Residual |
|---|---|---|---|---|---|
| DIE_1939_75 | 48 | 0 | 1 | 30 | 7 |
| FDL_1995_46 | 11 | 3 | 0.785 | 3 | 1 |
| FOR_1934_201 | 50 | 0 | 1 | 3 | 0 |
| GEM_1935_59 | 185 | 1 | 0.994 | 1 | 0 |
| GID_1845_10 | 41 | 10 | 0.803 | 7 | 1 |
| GRO_1926_61 | 285 | 4 | 0.986 | 4 | 0 |
| JAA_1984_21 | 11 | 0 | 1 | 5 | 1 |
| NIE_1959_40 | 35 | 0 | 1 | 5 | 0 |
| NVT_1946_46 | 3 | 0 | 1 | 3 | 0 |
| ONT_1961_139 | 16 | 2 | 0.888 | 2 | 0 |
| SPI_1983_3 | 102 | 31 | 0.766 | 25 | 17 |
| STR_1940_37 | 129 | 3 | 0.977 | 1 | 9 |
| TAA_1990_49 | 391 | 19 | 0.953 | 38 | 46 |
| Total: | 1307 | 73 | 0.947 | 127 | 82 |

Table 1: Performance of foreign language detection on test set. TP = True Positives, FP = False Positives.

In total the corpus consists of 172,882,529 words, of which 4,870,374 received the foreign language tag. So 2.817% of the corpus has been tagged. Below is an example of what the foreign language tags look like in the corpus. In this example, the full sentences are made up of English words. There are also sentences that are half Dutch and half foreign. If the sentence contains enough words foreign words, then the model will classify the sentence as foreign. In that case the algorithm will continue and classify each word separately, either as Dutch or as foreign.



```
<sentence id=GID_1995_56.txt_23>That__FL__ is__FL__ a__FL__ most__FL__ interesting__FL__ word__FL__.<\sentence>
<sentence id=GID_1995_56.txt_24>I__FL__ must__FL__ look that__FL__ word__FL__ up__FL__.<\sentence>
<sentence id=GID_1995_56.txt_25>Upon__FL__ my__FL__ word__FL__ I__FL__ must__FL__.<\sentence>
<sentence id=GID_1995_56.txt_26>Maar natuurlijk doet hij het niet en waarom zou hij?<\sentence>
<sentence id=GID_1995_56.txt_27>Het Engels is toch zeker zijn taal?<\sentence>
```

# Chapter 3

# Spelling normalization

## 1 Goal

It has been proven that spelling normalization leads to better results in lemmatization and PoS-tagging (van der Goot et al., 2017). It is therefore common to normalize a text prior to lemmatization and tagging. The C-CLAMP corpus contains modern Dutch, which is fairly close to present day Dutch. This means that a lot of words are already spelled correctly. The goal is to correctly normalize the spelling of words with the modern Dutch spelling to the present day Dutch spelling, while leaving those words that are already correct alone. I have focused on normalizing those words that occur at least five times in the corpus and ignored infrequent words that only occurred four times or less. This way it is possible to focus on correctly normalizing words that will actually improve the quality of the corpus. I combined a vector based approach with an algorithm that measures the edit distance between the modern Dutch word and its nearest neighbors.

## 2 Vector approach

It has been proven by Nguyen and Grieve (2020) that, next to semantic and syntactic properties, vectors are also able to encode spelling variation to some extent. Because spelling variation is very common in social media, they used English data from Twitter and Reddit to construct two word embedding models. The first model is a continuous skipgram model with negative sampling (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). The second model is a fastText model (Bojanowski, Grave, Joulin, & Mikolov, 2016), which is an extension of the continuous skipgram model. The big difference between these two models is the fact that the skipgram model does not take the spelling of a word into account, while the fastText does take spelling into account, using character-level n-grams. Each n-gram has a vector representation and a word is represented by the sum of the vector representations of its n-grams. Consider the word *apple*. If a model uses 3-grams, then the 3-grams of *apple* are *app*, *ppl* and *ple*. Both models use the contexts of words to learn the embeddings.

The embeddings were tested on an analogy test, a similarity analysis and a diagnostic classifier test. The goal of the analogy task is to retrieve the correctly spelled word given a spelling variant, based on pairs of spelling variants. For example, if there are two pairs (a: cookin, a*: cooking) and (b: movin, b*: moving) then the goal is to recover b* so that the following holds:

$$a - a* \approx b - b*$$

The similarity analysis consists of computing the average cosine similarity between the embeddings of the non-conventional and the conventional forms of several word pairs. The goal of the diagnostic classifier test is to predict the type of spelling variation. Seven different types of spelling variation are distinguished: lengthening, swapped characters, common misspellings (based on a list of common misspellings), g-dropping (-ing vs. -in), vowel omission, nearby character substitution and British vs. American spelling. The tests have shown that both word embedding models encode spelling variation at least to some extent and that some forms of spelling variation are captured more easily than others (Nguyen & Grieve, 2020). Given these results, it should be possible to use vectors in a spelling normalization task.

## 3 Word lists

Since not all the words in the corpus need to be normalized, I made a list of words that do need to be normalized. First, I used AntConc, which is a freeware corpus analysis toolkit for concordancing and text analysis, to make a frequency list of all the words in the corpus. I used this list to filter out the words that only appear four times or less. Then I used a Python script to filter the remaining words with two Dutch word lists, the OpenTaal word list (OpenTaal, 2020) and a word list that consists of television subtitles (Emmanuel, Marc, & Boris, 2010). The list still contained some foreign language words, despite having tagged them in the previous step. I used an English word list (dwyl, 2020), a French word list (Belgacem, 2017) and a German word list (Enzenberger, 2020) to remove as much of those remaining words as possible. Lastly, some remaining Greek words and rhyme schemes were removed by hand. This results in a list of 99,609 words. The list still contains some words that do not need to be normalized, mainly rare compounds that did not occur in either of the two Dutch word lists. It was impossible to remove these by hand due to the length of the list. The final algorithm will need to account for the fact that some words in the list should not be altered.

## 4 Models

The idea is to create a word embedding model that has the highest cosine similarity between as many modern Dutch word forms and their correctly spelled present day word form as possible. I made nine different models using the Python fastText package (Bojanowski et al., 2016). The models differ in three ways: the training data, the type of the model (skipgram or cbow), the n-grams used and the dimensions. The other settings are the same as the default settings. The default settings for fastText models are a skipgram model with 100 dimensions, a minimum n-gram of three and a maximum n-gram of six and a learning rate of 0.05, trained over five epochs. I used two different training sets: 1) the C-CLAMP corpus (total of 1.4GB data) and 2) the C-CLAMP corpus combined with the newspaper material of the SoNaR corpus (total of 2.49GB data). Both sets were cleaned beforehand, by removing the punctuation and the tags, and concatenated into one big file.

Figure 1 gives an overview of all the models. The names of the models indicate which settings are used. The default settings are used, unless specified otherwise. All models were tested on a set of 30 words, randomly selected from the list of words that need to be normalized, and the mean cosine similarity is calculated (see appendix A for the full dataset). All models show improvements over the default Dutch fastText model (model 1). A big improvement is made by using the C-CLAMP corpus as training data ($p < 0.001$, model 1 vs model 2) and further improvements are made by tweaking the model settings. The original fastText model is trained on Wikipedia data. The Wikipedia data most likely does not contain the spelling variation that is found in the C-CLAMP corpus. This explains the improvement when using the C-CLAMP data. I made three variants on the model trained with C-CLAMP data: a cbow model (model 4), a skipgram model with a minimum n-gram of two and a maximum n-gram of five (model 6) and a cbow model with a minimum n-gram of two and a maximum n-gram of five (model 8). Out of these three models the last one has the highest mean similarity, but the difference between these models is not significant. However, the difference between model 2 and model 8 is significant ($p < 0.05$).

Next, I tried to see if adding more training data would improve the model. I chose to add the newspaper material of the SoNaR corpus to the C-CLAMP data for two reasons. The first reason is that those two corpora fit well together thematically: periodicals and newspaper material. The second reason is that the present day Dutch spelling of some words might be underrepresented in C-CLAMP and SoNaR could mitigate this. The results show that adding the SoNaR data does not improve the default model (model 2 vs model 3, $p > 0.1$) or the skipgram model with a minimum n-gram of two and a maximum n-gram of five (model 6 vs model 7, $p < 0.01$). The mean cosine similarity for the cbow models is almost identical (model 4 vs model 5, $p > 0.5$). The cbow models with a minimum n-gram of two and a maximum n-gram of five are also almost identical (model 8 vs model 9, $p > 0.5$).

Lastly, I have tried to build a model with more dimensions. Increasing the dimensions is only a good idea when there is enough data, which is why I chose to increase the dimensions of the cbow model, trained on C-CLAMP and SoNaR data with a minimum n-gram of two and a maximum n-gram of five (model 9). This model uses both C-CLAMP and SoNaR data and has the highest mean cosine similarity, although the difference with other models is not significant. I tried to make a model with 150 dimensions instead of 100, resulting in model 10. However, this moderate increase didn't improve the mean cosine similarity. Instead, it dropped from 0.7479 to 0.7436, but this drop is not significant ($p > 0.5$).

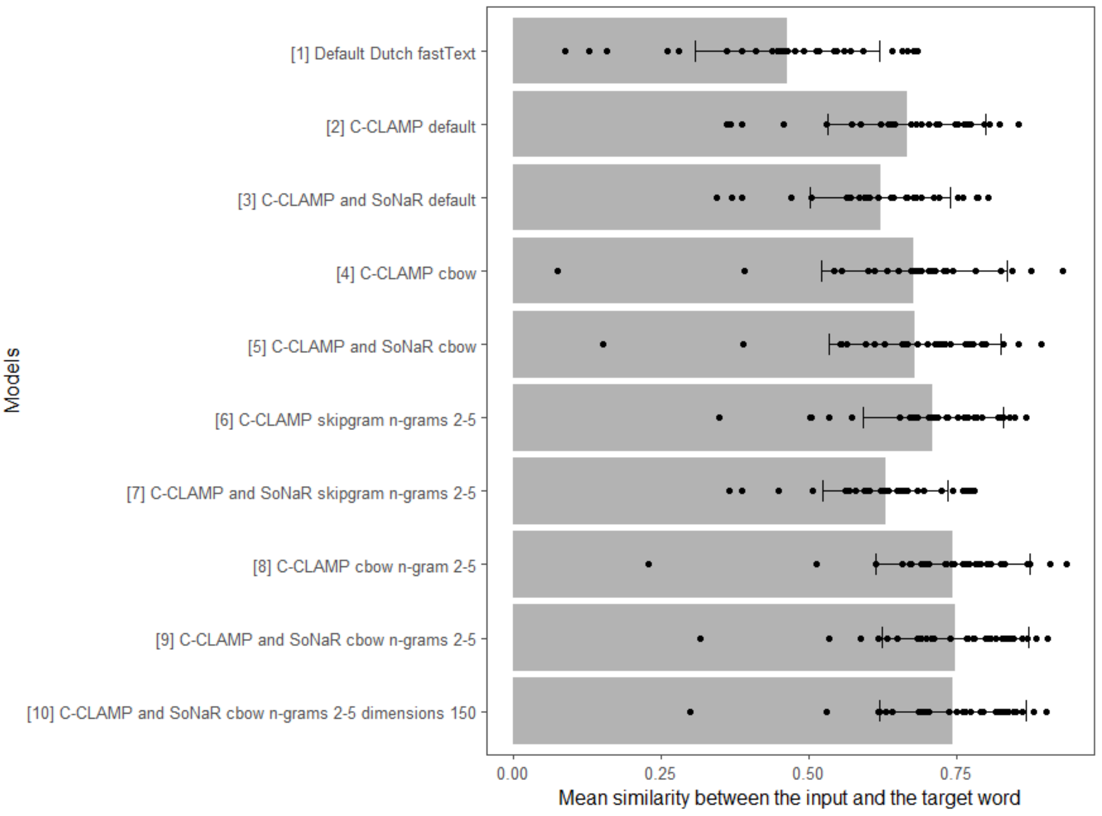


FIGURE 1: Comparison of the models. The bars show the mean similarity and the dots represent the words.

Since the difference in mean cosine similarity between most models is not significant, it is hard to tell which model is the best. Model 9 has the highest mean cosine similarity, but it does not significantly differ from model 8 and model 10. In order to get a better insight into the models, table 2 shows the amount of word pairs that have a cosine similarity greater than 0.8 and less than 0.5. What is needed is a model where the similarity between the input word and the correct word form is as high as possible. From the results I conclude that model 9 is the best model to use in the algorithm, given its mean cosine similarity and the amount of word pairs that have a cosine similarity greater than 0.8.

| Model | Mean CS | CS > 0.8 | CS < 0.5 |
|---|---|---|---|
| 1. Default Dutch fastText | 0.4631 | 0 | 17 |
| 2. C-CLAMP default | 0.6657 | 4 | 4 |
| 3. C-CLAMP & SoNaR default | 0.6207 | 1 | 4 |
| 4. C-CLAMP cbow | 0.6482 | 5 | 2 |
| 5. C-CLAMP & SoNaR cbow | 0.6786 | 4 | 2 |
| 6. C-CLAMP skipgram n-grams 2-5 | 0.7102 | 6 | 1 |
| 7. C-CLAMP & SoNaR skipgram n-grams 2-5 | 0.6293 | 0 | 3 |
| 8. C-CLAMP cbow n-grams 2-5 | 0.7427 | 9 | 1 |
| 9. C-CLAMP & SoNaR cbow n-grams 2-5 | **0.7479** | **13** | 1 |
| 10. C-CLAMP & SoNaR cbow n-grams 2-5 dim 150 | 0.7436 | 12 | 1 |

TABLE 2: This table shows the mean cosine similarity (CS) of each model, the words with a cosine similarity > 0.8 and the words with a cosine similarity < 0.5.

## 5 Algorithm

I wrote an algorithm that combines the vector model with edit distance rules. The algorithm uses the cbow model, trained on C-CLAMP and SoNaR, with a minimum n-gram of two and a maximum n-gram of five to generate 2000 nearest neighbors for each word in the list of words that need to be normalized. The algorithm then picks the nearest neighbor with the smallest edit distance from the input word. The output is a file with on each line the input word, followed by the corresponding output word in the following form:

vrolik: vrolijk

The edit distance is calculated using a set of rules, with a different cost for each rule. There are three kinds of rules: deletion rules, substitution rules and insertion rules. Each edit has a standard cost of 2.0, except those edits that account for a common spelling change, for example the substitution of *k* for *c* in words like *kontakt* that has become *contact*. Initially, all these rules had a cost of 0.25. The cost of the rules has been adapted through trial and error, by generating 5 samples of 100 words and adapting the rules to get as many words correct as possible. As mentioned in section 3.3, not all words in the list need to be normalized: some already have the correct form. The algorithm takes this into account by rejecting nearest neighbors that have an edit cost that is higher than 1.0. If the nearest neighbors with the smallest edit distance has a cost that is higher than 1.0 then the output word will be the same as the input word.

$$Accuracy = \frac{(true\ positives + true\ negatives)}{(true\ positives\ + false\ positives + true\ negatives + false\ negatives)}$$

The accuracy (see formula above) of the algorithm was tested on a sample of 250 words, randomly selected from the list of words that need to be normalized. The result can be seen in table 3. The algorithm has an accuracy of 0.588, which is not yet very high. A big problem is that the edit rules do not take into account the context in which the rules should be applied. An example is the word *heerlike*, which the algorithm turns into *eerlike*, instead of *heerlijke*. The deletion of *h* was meant for words like *dagh*, which has to become *dag*, or *dialectgeographie*, which has to become *dialectgeografie*. In other words, *h* should only be deleted when it follows *g* or *p* and not at the beginning of a word. Adding context to the rules raises the accuracy to 0.820, which is a great improvement.

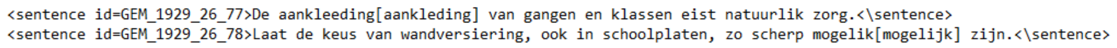
Words like *metaphysika* or *visscherien*, where multiple spelling changes are at play, pose another problem. Since the algorithm searches for the nearest neighbor with the smallest edit distance, it often finds a nearest neighbor that only solves one spelling change. The algorithm with context aware edit rules turns *metaphysika* into *metaphysica* (*ph* still needs to become *f*) and *visscherien* into *visscherijen* (*ch* still needs to be deleted). In order to achieve the correct results for this type of words, I tried to apply the algorithm a second time to the output of the first time. This strategy works for *metaphysika* and *visscherien*: *metaphysika* becomes *metafysica* and *visscherien* becomes *visserijen*. However, the accuracy drops to 0.744, because some rules work in both directions. This means that some words, like *tussenvoegsels*, that were correctly normalized during the first run of the algorithm, revert back to their non-normalized form *tusschenvoegsel*. To remedy this, I tried to run the algorithm a third time on the output of the second run. Indeed, this gives the highest accuracy, 0.828, since most reverted forms have been normalized again. However, the pay-off is not very high: the gain in accuracy is only 0.8% and this method is too unreliable. In the end it was decided to run the algorithm with context aware edit rules once. The rules can be found in appendix B.

| Algorithm variant | Accuracy |
|---|---:|
| Original algorithm | 0.588 |
| Algorithm with context aware edit rules | 0.820 |
| Algorithm with context aware edit rules + input is first output | 0.744 |
| Algorithm with context aware edit rules + input is second output | 0.828 |

Table 3: Accuracy of the algorithm

## 6 Discussion

To implement the list of normalized words into the corpus, I let the algorithm make a list of the old word forms and their corresponding present day word form. This list serves as input for a script that searches for these older forms and adds a tag to them with the present day word form. Once the normalized word forms have been added to the corpus, the corpus is ready to be lemmatized and Part-of-Speech tagged. The normalized corpus looks as follows:



```
<sentence id=GEM_1929_26_77>De aankleeding[aankleding] van gangen en klassen eist natuurlik zorg.<\sentence>
<sentence id=GEM_1929_26_78>Laat de keus van wandversiering, ook in schoolplaten, zo scherp mogelik[mogelijk] zijn.<\sentence>
```

Overall, the vector approach has been successful in normalizing the spelling of the corpus. From the sample of 250 words it has been estimated that the algorithm is correct for 82% of the words that have to be normalized. Combining a vector model with a simple nearest neighbor and minimum edit distance algorithm gives good results for spelling normalization. It is important to observe that the addition of context to the edit rules is crucial to the working of the algorithm, as it prevents the rules from being applied to words for which they are not intended. This project has shown that it is indeed possible to use word embeddings for spelling normalization. This approach can be applied to other languages or time periods by creating a word embedding model based on data for the relevant language and time period and by constructing new context aware rules, applicable to that language and time period.

While these results are good, several improvements could still be made to the algorithm. Words like *metaphysica* are still normalized incorrectly. This can be solved by filtering the output of the algorithm by again applying a word list to the output. The words that are still spelled incorrectly can be put through the algorithm again. This should at least partly avoid the problems mentioned above. A foolproof way to do this would be to manually check the output, but this is of course time consuming. The approach could also be made more efficient and possibly more accurate by splitting the input words into morphemes. It is then possible to normalize a list of morphemes instead of words. The first advantage here is that less frequent compounds for which the right nearest neighbor could not be found by the model, can still be correctly normalized. This is an effective way to deal with possible data sparsity in the model. A second advantage is that the list of morphemes to be normalized is shorter than the list of words to be normalized, which means that less nearest neighbors have to be generated and checked. This means that the algorithm would take less time to execute.

Chapter 4

# Lemmatization and Part-of-Speech tagging

## 1 Taggers

I consider two taggers for the Part-of-Speech tagging task: the Frog pipeline (Hendrickx, van den Bosch, van Gompel, & van der Sloot, 2016) and BERTje (de Vries et al., 2019), a Dutch pre-trained BERT model. For lemmatization Frog will be used.

### Frog: PoS-tagging

Frog is a modular memory-based morphosyntactic tagger and dependency parser for Dutch. Frog integrates several memory-based natural language processing modules developed for Dutch. It includes tokenization, part-of-speech tagging, lemmatization, dependency parsing, named entity recognition and morphological segmentation of word tokens (Hendrickx et al., 2016).

The Frog tagger combines two submodules, one for known words and one for unknown words (Van den Bosch, Busser, Canisius, & Daelemans, 2007). The submodule for the known words uses the classifier engine IGTree (Daelemans, Van den Bosch, & Weijters, 1997), an algorithm that makes decision trees in a top-down manner. The algorithm compresses a database of labeled examples into a decision-tree structure. The decision tree is composed of nodes that each represent a partition of the original example database, and are labeled by the most frequent class of that partition. This submodule for known words uses the following features:

- the focus word and its immediate left and right neighboring words;
- the three preceding predicted tags;
- the two still ambiguous tags on the right.

The unknown words module uses TRIBL. It builds a tree structure for the most informative features and performs k-nearest neighbor classification on the remaining features. The submodule for the unknown words uses the following features:

- the first two letters and the last three letters of the focus word;
- binary features marking whether the word is capitalized, contains a hyphen or one or more numbers;
- the immediate left and right neighboring words of the focus word;
- the three preceding predicted tags;
- the two still ambiguous tags on the right.

The tagger is trained on several manually annotated PoS-tagged Dutch corpora, all tagged with the Corpus Gesproken Nederlands (CGN or Spoken Dutch Corpus) tagset (Van Eynde, 2004): CGN, the ILK corpus, the D-Coi corpus and the Eindhoven corpus. This results in a dataset of 10 million words, of which 90% is used as training data and 10% as test data. Training a tagger on such a substantial training set has the advantage that usually less than 10% of the words in a new text will be unknown. This means that the submodule for the known words will do most of the work, but the remaining work for the unknown words tagger will be harder. The accuracy of Frog on the test set is 98.6% for the main tags (only part of speech) and 96.5% for the full tags (part of speech and morphosyntactic information) (Van den Bosch et al., 2007).

### Frog: lemmatization

The Frog lemmatizer is trained on the e-lex lexicon (TST-centrale, 2007). This lexicon contains 595,664 unique word form (PoS-tag and lemma) combinations and has been manually cleaned. A timbl classifier (Daelemans, Zavrel, der Sloot, & den Bosch, 2004) has been trained to learn to convert word forms to their lemmas. The training instances are taken from the lexicon by extracting the last 20 characters of the word forms. This means that word forms that are longer than 20 characters are abbreviated, for example *consumptiemaatschappijen* becomes *umptiemaatschappijen*. This results in a training set of 402,734 unique word forms. Leaving out the beginning of a word does not hinder the lemma assignment, because in Dutch most morphological changes occur in the word suffix. Each training instance has a class label that is the PoS-tag and a rewrite rule to transform the word form to the lemma form. These rewrite rules are applied to the word form endings and delete or insert one or multiple characters. The lemmatizer depends on the output of the PoS-tagger to disambiguate between different candidate lemmas. For example the word *zakken* can either be used as a noun and will have the lemma *zak* or as a verb and then it will have the lemma *zakken* (Hendrickx et al., 2016).

### BERTje

BERTje is a Dutch pre-trained BERT model and is architecturally equivalent to the BERT model. The difference between BERTje and the standard BERT model lies in the way that BERTje is pre-trained. BERTje is pre-trained on a dataset of 2.4B tokens, containing several Dutch corpora: Books, TwNC, SoNar-500, Web news and Wikipedia. The tasks used for pre-training are the sentence order prediction task and the masked language modeling task (de Vries et al., 2019). In the sentence order prediction task two sentences are either consecutive or swapped. The model has to predict which one is the case, which forces the model to learn semantic coherence between sentences. In the masked language modeling task tokens from the input get masked at random. Here, the model has to predict the original token based on the context. The model thus has to embed each word based on the context words. For BERTje, this last task is implemented by masking consecutive word pieces that belong to the same word. 15% of all tokens were masked. 80% of these are replaced with a mask token, 10% are replaced by a random token and the other 10% are left as-is.

A fine-tuned version of BERTje for PoS-tagging is available via DeepFrog. DeepFrog is the deep learning counterpart of Frog. It is complementary to Frog in the sense that while the various modules in Frog are built on decision trees and k-nearest neighbor classifiers, DeepFrog builds on deep learning techniques and can use a variety of neural transformers. One such neural transformer is BERTje. BERTje is fine-tuned on the same training data as Frog and uses the same CGN tagset. The precision of the model is 0.973 (van Gompel, 2021).

## 2 Results

After trying out both taggers on a few sentences of the corpus, it quickly became clear that BERTje has significantly more trouble with out-of-vocabulary words, most noticeably proper names, compared to Frog. When a word does not occur in the vocabulary of BERTje, the model splits the word into parts that it does know and tries to tag these parts. This poses a problem, because each part can have a different tag and there is no easy and automatic way to always pick the right tag in these situations. Given the genre of texts in C-CLAMP, a lot of proper names are present in these texts: names of authors, names of characters, titles of books, titles of periodicals, etc. For example, the name *Louk E. Roest Crollius* (retrieved from NIE_1949_107) is tagged is tagged correctly by Frog, but not by BERTje, as shown in table 4.

| | Frog | BERTje |
|---|---|---|
| Louk | SPEC(deeleigen) | lo: SPEC(vreemd)<br>u: N(soort, ev, basis, zijd, stan)<br>k: SPEC(vreemd) |
| E. | SPEC(deeleigen) | e: SPEC(afk)<br>.: LET() |
| Roest | SPEC(deeleigen) | roest: N(soort,ev,basis,zijd,stan) |
| Crollius | SPEC(deeleigen) | cro: N(soort,ev,basis,zijd,stan)<br>lli: SPEC(vreemd)<br>us: SPEC(vreemd) |

Table 4: Illustration of the difference between Frog en BERTje for the PoS-tagging of proper names.

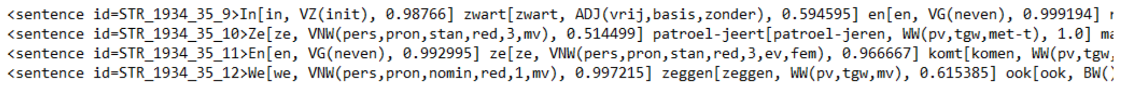
Given this problem with BERTje and the fact that Frog does tag these words correctly, I decided to first try out Frog, both for PoS-tagging and lemmatization, on a test set of 13 texts from the corpus. Each text in the set is from a different periodical. I also made sure to vary the years in which the texts are written, so both older and more recent periods are represented. Table 5 shows that Frog performs really well on both tasks. The average accuracy of Frog on the PoS-tagging task is 97.79% and 98.59% on the lemmatization task. There is one outlier for the PoS-tagging task: text GRO_1910_7. Frog made significantly more errors tagging this text, compared to the other texts. This could be because it is one of the older texts, but the sample contains even older texts that have not been detected as outliers.

| | PoS-tag | | | Lemma | | |
|---|---|---|---|---|---|---|
| Text | Correct | Incorrect | Accuracy | Correct | Incorrect | Accuracy |
| DIE_1900_9 | 363 | 17 | 95.53% | 369 | 11 | 97.11% |
| FDL_1963_5 | 398 | 5 | 98.76% | 401 | 2 | 99.50% |
| FOR_1934_95 | 341 | 8 | 97.71% | 340 | 9 | 97.42% |
| GEM_1925_58 | 154 | 2 | 98.72% | 153 | 3 | 98.08% |
| GID_1838_37 | 443 | 7 | 98.44% | 446 | 4 | 99.11% |
| GRO_1910_7 | 737 | 48 | 93.89% | 750 | 35 | 95.54% |
| JAA_1934_7 | 300 | 11 | 96.46% | 303 | 8 | 97.43% |
| NIE_1949_107 | 246 | 3 | 98.80% | 246 | 3 | 98.80% |
| NVT_1979_38 | 445 | 2 | 99.55% | 446 | 1 | 99.78% |
| ONT_1962_141 | 250 | 7 | 97.28% | 255 | 2 | 99.22% |
| SPI_1979_1 | 337 | 7 | 97.97% | 343 | 1 | 99.71% |
| STR_1934_35 | 299 | 2 | 99.34% | 301 | 0 | 100.00% |
| TAA_1993_20 | 336 | 4 | 98.82% | 340 | 0 | 100.00% |
| Total | 4649 | 123 | 97.42% | 4693 | 79 | 98.34% |

Table 5: POS-tagging and lemmatization results for Frog

I have tried to further improve the incorrect lemmas by comparing each lemma to the subtitle word list (Emmanuel et al., 2010). If the lemma occurs in the word list then it is correct. If it does not occur in the word list, I tried to find a replacement from the word list by picking out a word that only underwent either one substitution, one deletion or one insertion. This attempt was unsuccessful, because it only introduced new errors.

Since Frog performs really well, it was decided to tag and lemmatize the corpus with Frog. If a word has been normalized, Frog only takes into account the normalized form for the PoS-tagging and the lemmatization. The non-normalized forms are afterwards put back in the text. Each word receives a tag between square brackets with the lemma, the PoS-tag and a confidence score. The corpus now has its final form:



```
<sentence id=STR_1934_35_9>In[in, VZ(init), 0.98766] zwart[zwart, ADJ(vrij,basis,zonder), 0.594595] en[en, VG(neven), 0.999194]
<sentence id=STR_1934_35_10>Ze[ze, VNW(pers,pron,stan,red,3,mv), 0.514499] patroel-jeert[patroel-jeren, WW(pv,tgw,met-t), 1.0]
<sentence id=STR_1934_35_11>En[en, VG(neven), 0.992995] ze[ze, VNW(pers,pron,stan,red,3,ev,fem), 0.966667] komt[komen, WW(pv,tgw,
<sentence id=STR_1934_35_12>We[we, VNW(pers,pron,nomin,red,1,mv), 0.997215] zeggen[zeggen, WW(pv,tgw,mv), 0.615385] ook[ook, BW(
```

# Chapter 5

# Conclusion

The goal of this project to clean up, PoS-tag and lemmatize the C-CLAMP corpus, has been completed. This project has led to two important insights.

First, the project confirms that it is indeed possible to not only use vector models to encode spelling variation, but also to use these vector models in a historical spelling normalization task. This method is useful when there is no or not enough annotated data available to carry out statistical or neural network approaches to spelling normalization. However, raw data is required to build a solid vector model. In this project the corpus itself was used to build the model, together with almost the same amount of data from the SoNaR corpus. Table 2 (p. 10) shows that it is also possible to make a good model with only the corpus data. This means that other historical corpora that are about the same size as C-CLAMP can be normalized in a similar fashion as the C-CLAMP corpus.

Secondly, the project demonstrates that memory-based taggers can achieve good results on the PoS-tagging and lemmatization of historical texts, provided that the texts have been normalized. Given the time constraints on the project, I was unable to do an analysis of the performance of BERTje like I did for Frog. However, the fact that BERTje struggles with proper names, which occur a lot in C-CLAMP, is an indication that the performance of BERTje on C-CLAMP would not be as high as the performance of Frog. Furthermore, there is no easy or straightforward solution to find the correct tag among the multiple tags that BERTje generates for one word. For this reason, I come to the conclusion that Frog is the most suitable tagger for the C-CLAMP corpus. This is a rather surprising conclusion, because neural network taggers overall do tend to achieve better results on these tasks than memory-based approaches.

# Bibliography

Belgacem, H. B. (2017). *French-dictionary.* Retrieved from `https://github.com/hbenbel/French-Dictionary`

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. *CoRR*, *abs/1607.04606*. Retrieved from `http://arxiv.org/abs/1607.04606`

Daelemans, W., Van den Bosch, A., & Weijters, A. (1997). Igtree: Using trees for compression and classification in lazy learning algorithms. *Artif. Intell. Rev.*, *11*, 407–423. doi: 10.1023/A:1006506017891

Daelemans, W., Zavrel, J., der Sloot, K. V., & den Bosch, A. V. (2004). Timbl: Tilburg memory based learner, version 6.3, reference manual. technical report ilk research group technical report series 10-01. *ILK, Tilburg University, The Netherlands*.

de Vries, W., van Cranenburgh, A., Bisazza, A., Caselli, T., van Noord, G., & Nissim, M. (2019). Bertje: A dutch BERT model. *CoRR*, *abs/1912.09582*. Retrieved from `http://arxiv.org/abs/1912.09582`

dwyl. (2020). *English-words.* Retrieved from `https://github.com/dwyl/english-words`

Emmanuel, K., Marc, B., & Boris, N. (2010). Subtlex-nl: A new frequency measure for dutch words based on film subtitles. *Behavior Research Methods*, *42(3)*.

Enzenberger, M. (2020). *German-wordlist.* Retrieved from `https://github.com/enz/german-wordlist`

Hendrickx, I., van den Bosch, A., van Gompel, M., & van der Sloot, K. (2016). *Frog, a natural language processing suite for dutch, reference guide* (Tech. Rep.).

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fast-text.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, *abs/1310.4546*. Retrieved from `http://arxiv.org/abs/1310.4546`

Nguyen, D., & Grieve, J. (2020). Do word embeddings capture spelling variation? In *Proceedings of the 28th international conference on computational linguistics* (pp. 870–881). Barcelona, Spain (Online): International Committee on Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/2020.coling-main.75` doi: 10.18653/v1/2020.coling-main.75

OpenTaal. (2020). *Nederlandse woordenlijst.* Retrieved from https://github.com/OpenTaal/opentaal-wordlist

TST-centrale. (2007). E-lex voor taal- en spraaktechnologie, version 1.1. technical report. *Nederlandse TaalUnie*.

Van den Bosch, A., Busser, B., Canisius, S., & Daelemans, W. (2007). An efficient memory-based morphosyntactic tagger and parser for dutch. *Computational Linguistics in the Netherlands: Selected Papers from the Seventeenth CLIN Meeting*, 99-114.

van der Goot, R., Plank, B., & Nissim, M. (2017). To normalize, or not to normalize: The impact of normalization on part-of-speech tagging. *CoRR*, *abs/1707.05116*. Retrieved from http://arxiv.org/abs/1707.05116

Van Eynde, F. (2004). Part of speech tagging en lemmatisering van het corpus gesproken nederlands. technical report. *Centrum voor Computerlingustiek, KU Leuven, Belgium.*.

van Gompel, M. (2021). *Deepfrog - nlp suite.* Retrieved from https://github.com/proycon/deepfrog

# Appendices

## Appendix A: Test set word embedding models

| Original word | Target word | Default Dutch fastText | C-CLAMP default | C-CLAMP cbow |
|---|---|---|---|---|
| visch | vis | 0.460897952 | 0.586541653 | 0.554300547 |
| mensch | mens | 0.591504753 | 0.769569278 | 0.677417755 |
| aadlaar | adelaar | 0.157151744 | 0.702444613 | 0.782132506 |
| eerlik | eerlijk | 0.546418369 | 0.638656914 | 0.714394927 |
| vriendelik | vriendelijk | 0.569211066 | 0.797403395 | 0.844700098 |
| veele | vele | 0.453524262 | 0.571605504 | 0.73387748 |
| aaklig | akelig | 0.128223747 | 0.622637451 | 0.709471822 |
| gedoan | gedaan | 0.27860561 | 0.456648856 | 0.556062281 |
| vleesch | vlees | 0.683875978 | 0.806440115 | 0.71153456 |
| produkt | product | 0.666351438 | 0.821797371 | 0.875310004 |
| konkreet | concreet | 0.445503891 | 0.747620165 | 0.728994071 |
| woar | waar | 0.087485842 | 0.386359185 | 0.073189452 |
| kontakt | contact | 0.677402496 | 0.719252706 | 0.844558716 |
| bestaen | bestaan | 0.260537058 | 0.360086024 | 0.703181982 |
| tusschen | tussen | 0.541885436 | 0.75271517 | 0.61079663 |
| Nederlandsche | Nederlandse | 0.639994085 | 0.702077806 | 0.690092027 |
| zooveel | zoveel | 0.464111 | 0.673758149 | 0.599906027 |
| noodig | nodig | 0.457585245 | 0.690661788 | 0.65261972 |
| geene | geen | 0.386447132 | 0.772897482 | 0.74268955 |
| geheele | gehele | 0.47624892 | 0.634208024 | 0.631546974 |
| elkander | elkaar | 0.657551348 | 0.855712473 | 0.825249791 |
| beteekenis | betekenis | 0.457981944 | 0.644992828 | 0.729741156 |
| algemeene | algemene | 0.513438702 | 0.645801783 | 0.684427202 |
| regeering | regering | 0.517563403 | 0.805556536 | 0.92960459 |
| zoowel | zowel | 0.359519094 | 0.529226303 | 0.389822245 |
| schoone | schone | 0.490437806 | 0.681115925 | 0.712764382 |
| zeide | zei | 0.438540936 | 0.762276828 | 0.673700809 |
| hooger | hoger | 0.559072137 | 0.749040663 | 0.709064186 |
| meening | mening | 0.516613126 | 0.368035883 | 0.54337579 |
| wenschen | wensen | 0.409772724 | 0.716745615 | 0.712496579 |

| Original word | Target word | C-CLAMP skipgram n-gram 2-5 | C-CLAMP CBOW n-gram 2-5 | C-CLAMP Sonar default |
|---|---|---|---|---|
| visch | vis | 0.652803302 | 0.613445342 | 0.710829496 |
| mensch | mens | 0.779480755 | 0.733736634 | 0.783182681 |
| aadlaar | adelaar | 0.769155145 | 0.831040382 | 0.638811767 |
| eerlik | eerlijk | 0.716620445 | 0.781337082 | 0.666268349 |
| vriendelik | vriendelijk | 0.828531861 | 0.874780238 | 0.802444279 |
| veele | vele | 0.731906533 | 0.79050529 | 0.564554751 |
| aaklig | akelig | 0.679810524 | 0.82487148 | 0.60289079 |
| gedoan | gedaan | 0.502901852 | 0.690077007 | 0.385231316 |
| vleesch | vlees | 0.840420485 | 0.782884657 | 0.760089934 |
| produkt | product | 0.822360635 | 0.907588303 | 0.719596148 |
| konkreet | concreet | 0.765873134 | 0.763943315 | 0.585217416 |
| woar | waar | 0.3486332 | 0.227388754 | 0.342919797 |
| kontakt | contact | 0.762965083 | 0.808552921 | 0.681666791 |
| bestaen | bestaan | 0.502456009 | 0.75963974 | 0.369591177 |
| tusschen | tussen | 0.735055268 | 0.659016073 | 0.690541446 |
| Nederlandsche | Nederlandse | 0.734479308 | 0.746301949 | 0.64340198 |
| zooveel | zoveel | 0.684446931 | 0.671838701 | 0.570471942 |
| noodig | nodig | 0.709228694 | 0.703158855 | 0.597025096 |
| geene | geen | 0.820145726 | 0.801697433 | 0.564219415 |
| geheele | gehele | 0.675980508 | 0.731777549 | 0.643088639 |
| elkander | elkaar | 0.866777897 | 0.870263755 | 0.78664875 |
| beteekenis | betekenis | 0.702518284 | 0.803440809 | 0.751225173 |
| algemeene | algemene | 0.671889126 | 0.78766948 | 0.677080095 |
| regeering | regering | 0.848008633 | 0.935441732 | 0.596683562 |
| zoowel | zowel | 0.572289646 | 0.512486875 | 0.503499806 |
| schoone | schone | 0.752954483 | 0.787145734 | 0.618381858 |
| zeide | zei | 0.78354758 | 0.695845723 | 0.664661884 |
| hooger | hoger | 0.793265522 | 0.770761013 | 0.638697326 |
| meening | mening | 0.534353614 | 0.673906922 | 0.469583422 |
| wenschen | wensen | 0.717491031 | 0.740839958 | 0.594402194 |

| Original word | Target word | C-CLAMP Sonar cbow | C-CLAMP Sonar skipgram n-gram 2-5 |
|---|---|---|---|
| visch | vis | 0.628173828 | 0.693738401 |
| mensch | mens | 0.739797711 | 0.773248255 |
| aadlaar | adelaar | 0.797904074 | 0.725133419 |
| eerlik | eerlijk | 0.773880839 | 0.649840713 |
| vriendelik | vriendelijk | 0.89323318 | 0.780819893 |
| veele | vele | 0.777622581 | 0.57785809 |
| aaklig | akelig | 0.730763018 | 0.597202957 |
| gedoan | gedaan | 0.683774292 | 0.385322183 |
| vleesch | vlees | 0.700001478 | 0.77114588 |
| produkt | product | 0.855225563 | 0.74344784 |
| konkreet | concreet | 0.662979007 | 0.603037775 |
| woar | waar | 0.150315225 | 0.364491135 |
| kontakt | contact | 0.658146441 | 0.684651136 |
| bestaen | bestaan | 0.767466187 | 0.448668152 |
| tusschen | tussen | 0.556070089 | 0.667081296 |
| Nederlandsche | Nederlandse | 0.611421108 | 0.629362643 |
| zooveel | zoveel | 0.564637601 | 0.561282814 |
| noodig | nodig | 0.55334276 | 0.579727948 |
| geene | geen | 0.665260553 | 0.593099833 |
| geheele | gehele | 0.714027226 | 0.654501081 |
| elkander | elkaar | 0.765985191 | 0.766427517 |
| beteekenis | betekenis | 0.800171673 | 0.759706557 |
| algemeene | algemene | 0.724727333 | 0.660936177 |
| regeering | regering | 0.828761756 | 0.629023015 |
| zoowel | zowel | 0.38896066 | 0.505070388 |
| schoone | schone | 0.717023075 | 0.620705307 |
| zeide | zei | 0.595427811 | 0.652942896 |
| hooger | hoger | 0.595100522 | 0.599769771 |
| meening | mening | 0.792525113 | 0.633589566 |
| wenschen | wensen | 0.666521251 | 0.567931652 |

| Original word | Target word | C-CLAMP Sonar cbow n-gram 2-5 | C-CLAMP Sonar cbow n-gram 2-5 Dim 150 |
|---|---|---|---|
| visch | vis | 0.689925373 | 0.695520043 |
| mensch | mens | 0.780634105 | 0.764465153 |
| aadlaar | adelaar | 0.842240572 | 0.83833611 |
| eerlik | eerlijk | 0.842886031 | 0.829699039 |
| vriendelik | vriendelijk | 0.905023456 | 0.901616752 |
| veele | vele | 0.827630639 | 0.816223919 |
| aaklig | akelig | 0.847168863 | 0.849794745 |
| gedoan | gedaan | 0.769611418 | 0.736373603 |
| vleesch | vlees | 0.76814872 | 0.773943126 |
| produkt | product | 0.884611726 | 0.879793763 |
| konkreet | concreet | 0.739502192 | 0.750493407 |
| woar | waar | 0.316260457 | 0.29870832 |
| kontakt | contact | 0.699501038 | 0.684738219 |
| bestaen | bestaan | 0.80814147 | 0.821563601 |
| tusschen | tussen | 0.616581976 | 0.629909813 |
| Nederlandsche | Nederlandse | 0.683310032 | 0.689178884 |
| zooveel | zoveel | 0.649872422 | 0.639881611 |
| noodig | nodig | 0.631597936 | 0.617343962 |
| geene | geen | 0.778288662 | 0.761410952 |
| geheele | gehele | 0.802518785 | 0.793873429 |
| elkander | elkaar | 0.81737709 | 0.832238793 |
| beteekenis | betekenis | 0.86214304 | 0.847329617 |
| algemeene | algemene | 0.831198931 | 0.82543987 |
| regeering | regering | 0.868864536 | 0.861071765 |
| zoowel | zowel | 0.533466816 | 0.529345214 |
| schoone | schone | 0.799983799 | 0.790920913 |
| zeide | zei | 0.588028193 | 0.618802667 |
| hooger | hoger | 0.708055913 | 0.703879416 |
| meening | mening | 0.835809231 | 0.826128185 |
| wenschen | wensen | 0.710940123 | 0.701648831 |

## Appendix B: Context aware edit rules

| Rule | Cost | Word example |
|---|---|---|
| insertion of e except at the end of a word | 0.50 | duizlen → duizelen |
| insertion of j | 0.55 | vrolik → vrolijk |
| insertion of c | 0.25 | antarctis → antarctisch |
| + insertion of h | 0.25 | |
| insertion of h except at the start of a word | 0.25 | aanbragt → aanbracht |
| + substitution of g for c | 0.15 | |
| insertion of u | 0.50 | harmoniese → harmonieuze |
| insertion of - | 2.00 | does not occur |
| deletion of a if followed by a | 0.50 | beschaaving → beschaving |
| deletion of e except at the end of a word | 0.50 | arresteeren → arresteren |
| deletion of o | 0.50 | eentoonig → eentonig |
| deletion of u if followed by u | 0.50 | muuren → muren |
| deletion of n except at the end of a word | 0.50 | aantonnen → aantonen |
| deletion of g except before e | 0.50 | koningklijke → koninklijke |
| deletion of d except at end/start of a word | 0.50 | badden → baden |
| deletion of c after s | 0.10 | visch → vis |
| + deletion of h after c | 0.10 | |
| deletion of c before k | 0.10 | donckere → donkere |
| deletion of h after g | 0.10 | dagh → dag |
| deletion of h after t | 0.10 | logarithmen → logaritme |
| substitution of y for i | 0.05 | asylrecht → asielrecht |
| + insertion of e except at the end of a word | 0.50 | |
| substitution of e for a | 0.10 | allemael → allemaal |
| substitution of o for a before a | 0.10 | woar → waar |
| substitution of t for d | 0.10 | goetduncken → goeddunken |
| substitution of s for t | 0.15 | emosies → emoties |
| substitution of j for e after i | 0.15 | fabrijk → fabriek |
| substitution of s for z | 0.15 | aansien → aanzien |
| substitution of p for f | 0.10 | phantasie → fantasie |
| + deletion of h after p | 0.10 | |
| substitution of c for k before k | 0.10 | aantrecken → aantrekken |
| substitution of k for c | 0.05 | diktie → dictie |
| substitution of o for e | 2.00 | does not occur |
| substitution of e for u | 2.00 | does not occur |
| normalization of diacritics | 0.05 | |