Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 3868–3876
Marseille, 11–16 May 2020

Bag & Tag'em - A New Dutch Stemmer

Anne Jonker[1], Corné de Ruijt[1] and Jornt R. de Gruijl[2]
Vrije Universiteit Amsterdam, Faculty of Science [1], Bright & Company[2]
annejonker@gmail.com, c.a.m.de.ruijt@vu.nl, jornt.de.gruijl@brightcompany.nl

Abstract

We propose a novel stemming algorithm that is both robust and accurate compared to state-of-the-art solutions, yet addresses several of the problems that current stemmers face in the Dutch language. The main issue is that most current stemmers cannot handle 3rd person singular forms of verbs and many irregular words and conjugations, unless a (nearly) brute-force approach is used. Our algorithm combines a new tagging module with a stemmer that uses tag-specific sets of rigid rules: the Bag & Tag'em (BT) algorithm. The tagging module is developed and evaluated using three algorithms: Multinomial Logistic Regression (*MLR*), Neural Network (*NN*) and Extreme Gradient Boosting (*XGB*). The stemming module's performance is compared with that of current state-of-the-art stemming algorithms for the Dutch Language. Even though there is still room for improvement, the new BT algorithm performs well in the sense that it is more accurate than the current stemmers and faster than brute-force-like algorithms. The code and data used for this paper can be found at: `https://github.com/Anne-Jonker/Bag-Tag-em`

Keywords: Stemming, PoS tagging, Dutch

1. Introduction

The increased availability of both text data and computational power has resulted in an increase in applications using text mining for various purposes: it has been applied in sentiment analyses to determine the overall reputation of a company (Pang et al., 2008), identifying and analysing cybercrime activities (Kontostathis et al., 2010) and fraud detection (Phua et al., 2010), to name a few.

In order to perform such analyses and develop applications based on unstructured text data, data is commonly preprocessed in a number of ways. One such common preprocessing step in text mining is to reduce the number of variants of a word to a common root in order to reduce noise and enhance the accuracy of the analyses that follow. This process of reducing variants of words to a common *stem* is referred to as *stemming*. Stemming has been studied thoroughly for the English language and considerable success has already been achieved. This is commonly done through use of the Porter algorithm (Porter and others, 1980) that uses a rigid rule-based system to stem a word.

The problem with many other languages, including the Dutch language, is that they have been less of a focal point for research and that many words are irregular when it comes to their inflections. As such, they do not conform to the otherwise established rule sets, forming exceptions. Our aim was to construct a new algorithm that is both more accurate and robust than the state-of-the-art algorithms currently in use for the Dutch language. Our algorithm, the Bag & Tag'em (BT) algorithm, achieves this through the two components it comprises.

The first component is a new tagging model that categorises a word based on bi-grams of characters in singular words. This differs from established tagging algorithms which use large corpora of pre-tagged words to look up a corresponding category. An example of such an algorithm is Frog (Bosch et al., 2007). In theory, our approach decreases the requisite computational resources at the cost of some degree of accuracy. Furthermore, it should increase robustness and allow the algorithm to better process typos and neologisms, for example.

The second component applies pre-determined rules to stem a word based on its assigned tag. This part of the approach is similar to the Porter algorithm, but uses rules specific to the Dutch language instead.

The final algorithm could aid in analysing large quantities of Dutch text documents, due to its accuracy, robustness and computational efficiency compared to current standards. As such, applications of text mining stand to gain from the algorithm described in this article.

We give an overview of stemming methods in Section 2., followed by the methods on how the BT algorithm was built in Section 3. The results of the experiments are shown in Section 4. and discussed in Section 5.2.

2. Literature

2.1. Stemming

Stemming is the process by which words or grammatical forms are reduced to common stems (Jivani and others, 2011). An example of this is to reduce the words *walking* and *walked* to their common stem: *walk*. The purpose of doing so is to decrease the number of occurrences of words that have different forms, yet convey the same meaning.

Several stemming algorithms are applied in the text mining field, and can be categorised as one of three classes (Jivani and others, 2011):

1. Truncating Methods (e.g. Lancaster, Porter, Snowball). These stemming algorithms use affix stripping to reduce a word to its corresponding stem.

2. Statistical Methods (e.g. Hidden Markov Models (HMM)). These stemming algorithms use probabilities to determine what the correct stem should be, often done in forms of Neural Networks.

3. Mixed or Inflectional and Derivational Methods (IDM) (e.g. Part of Speech). These methods combine rule-based approaches and statistical approaches

and therefore tend to be more morphologically accurate. These stemming algorithms rely more on large corpora and context information.

In this paper we will focus on IDM, where we will make use of truncating and PoS tagging algorithms.

2.2. Truncating algorithms

Affix stripping algorithms rely on a set of rules in order to remove certain parts of words, be it at the start (prefix) or at the end (suffix). A few basic rules for the English language (out of many rules) are as follows (Willett, 2006): If the word ends with *-ing*, remove *ing*. If the word ends with *-ed*, remove *ed*. If the word ends with *-s*, remove *s*.
However, problems occur with irregular verbs; for example *run* and *ran*. Both have the same stem - *run* - but this technique would identify two different stems instead. The Dutch language contains many irregular verbs and nouns and is therefore problematic in this regard. Every language has its own characteristics, practically necessitating use of a language-specific stemmer for best results.
There are a few different variants of stemming algorithms that serve a specific purpose. Some require the stripped stem to be a word that is contained in the lexicon of the language (akin to the process of lemmatisation or reducing a word to its canonical form). If this is not the case, a new rule has to be applied to reduce it even further to derive a proper stem, or expand the lexicon itself.
The benefit of rule-based stemming techniques is fast processing of documents (Jivani and others, 2011), but the downside is the potential for large inaccuracies due to prefixes or suffixes of words wrongly triggering rules or preventing rules from being applied. We investigated the feasibility of improving performance through use of specific subsets of rules, depending on the word's structure.
The most common algorithms for affix stripping are: Porter (Willett, 2006), Lancaster (Paice, 1990) and Snowball (Porter, 2001). Snowball is a version of the Porter algorithm adapted to make the set of rules more language-specific, as the affix stripping rules are different for each language.

2.3. Inflectional and Derivational Methods

A different approach to stemming is to make use of Part of Speech Tagging (PoS), in which words are tagged based on their function in the sentence and then accordingly handled further by different sets of rules (Monz and De Rijke, 2001).
The algorithm tags words as a type or "speech tag", e.g. noun, adjective, verb and so on. An approach based on probability was proposed by (Brill, 1992). In this case, the algorithm assigns a PoS tag to each word by using the corresponding tag probability for a word as obtained from a large previously tagged corpus. A word will therefore always be assigned the same specific tag, disregarding context. The example that was presented in the original paper shows that in the following two sentences the word *run* is tagged as a verb, since *run* was most frequently a verb in the tagged corpus, rather than a noun as in the first sentence:

1. The *run* lasted thirty minutes.
2. We *run* three miles every day.

One potential downside of this approach is that there seems to be no direct way to assign a tag to words not contained in the training corpus. This was mitigated by assigning a tag based on the last three letters of such unknown words. Relatively small (parts of) suffixes, and potentially other letter combinations, may therefore be highly indicative of a corresponding tag.
The paper showed that the performance of this relatively simple part-of-speech tagger was roughly the same as that of other taggers, but had several advantages. These include speed, applicability and ease of transfer to different languages.
Two PoS taggers for the Dutch language are Alpino (Van der Beek et al., 2002) and Frog (Bosch et al., 2007). Both use a large pre-tagged corpus from the CELEX database (Van der Wouden, 1990). Frog is generally regarded as highly accurate for the Dutch language and may be considered a benchmark representing the state-of-the-art in terms of performance. Frog determines both tag and stem (or lemma) based on a single word rather than any additional information from the sentence that word was taken from. Since PoS tagging shows promising results in literature and due to the availability of the Frog algorithm as a ground truth, we investigated the feasibility of a novel approach that combines a token-based (single-word) tagging module and a rule-based stemming module.

3. Methods

The scope of this paper limits itself to the Dutch language. We evaluated the following three algorithms for the multinomial classification problem of tagging: Multinomial Logistic Regression (*MLR*), Neural Network (*NN*), Extreme Gradient Boosting (*XGB*). For the actual stemming based on PoS tags, we developed a new rule-based stemmming module as part of the Bag & Tag'em (BT) algorithm. Table 1 shows the used tags combined with their translation and examples. The tags are the common abbreviations in the Dutch language.

Tag	Translation	Example	Dutch
OTT	Present tense	Walk	Loop
ZNW	Noun	Dog	Hond
OVT	Simple past tense	Walked	Liep
VTT	Past perfect tense	Walking	Gelopen
BVNW	Adjective	Big	Groot
BW	Adverb	Other	Ander

Table 1: Tags explained

3.1. Tagging algorithms

To evaluate the performance of MLR, XGB and NN in assigning tags, the F_1 score was used. This is preferable to accuracy and based on both precision and recall, which are otherwise common measures for performance.
The MLR is one of the most basic regression techniques for multinomial classification problems (Schmid, 1994), and was selected as a benchmark for the other algorithms. Due

to the rise and extensive documentation on XGB (Chen and Guestrin, 2016) with very promising results in other fields, we investigated its performance on PoS tagging. To our knowledge, this is uncommon: we are not aware of any other work that uses XGB for PoS tagging purposes. Literature also suggests that NN show promising results in Natural Language Processing (NLP) (Schmid, 1994) and therefore we also investigated a PoS tagging implementation using neural networks.

The algorithms were implemented using packages from Scikit-learn (Pedregosa et al., 2011). To improve the predictions of the algorithms, hyperparameters were tuned. The tuning was performed through grid searches, optimising the F_1 score, which we used as performance measure.

For experimental purposes two datasets (DS_1, DS_2) were created, where in the first dataset (DS_1) no feature selection was implemented, in contrary to the second dataset (DS_2). Both datasets are based on the training set, which will be discussed in more depth in Section 3.3. DS_1 and DS_2 were grid searched separately while using the same train and test words. As the results on DS_1 are similar and lead to the same conclusion, only the results of DS_2 are shown here. Since the results with feature selection outperformed those without feature selection, the final models are based on dataset with feature selection.

To combine the F_1 score for each tag, we consider the weighted F_1 score, which we write as (1)

$$\bar{F} = \sum_{k=1}^{K} \frac{F_1^{(k)} S_k}{\sum_{k=1}^{K} S_k}, \quad (1)$$

where $F_1^{(k)}$ denotes the F_1 score for tag $k \in \{1, \ldots, K\}$ and S_k is the support for tag k. An overview of the found $F_1^{(k)}$, S_k and $\bar{F}_1$-scores is given in Table 2.

DS_2	NN	XGB	MLR	Support
OTT	0.85	0.91	0.85	7313
ZNW	0.68	0.75	0.67	2982
OVT	0.78	0.88	0.80	2458
VTT	0.81	0.65	0.83	1438
BVNW	0.48	0.61	0.48	824
BW	0.21	0.32	0.26	219
$\bar{F}$	0.77	**0.84**	0.77	15234

Table 2: F_1-scores on DS_2 (with feature selection)

3.2. Stemming algorithms

The quality of a stemmer is determined by its ability to accurately reduce words belonging together to a common stem while ensuring that words that do not belong together yield different stems (Moral et al., 2014). There are two main errors that can occur and both relate to the aggressiveness of the stemmer in removing parts of the words. If the stemmer is not aggressive enough, insufficient prefixes and/or suffixes are deleted, which results in an incorrect stem. This type of error is called understemming, since the stripping of characters is under the required level. As a consequence, variants of a word may end up with different stems.

Conversely, the other type of error is overstemming. In this case, the stemmer is too aggressive and removes too much of a word, potentially clearing parts of the morphological root. As a result, different words may be wrongly assigned the same stem.

The BT algorithm classifies the words in the dataset into the six tags introduced in Table 1. Each subcategories represents a tag which follows different stemming rules.

3.2.1. Verbs (OTT, OVT, VTT)

The first step of stemming a verb is to determine if it is a compound verb. This is done by comparing the first three characters of the verb and a list of known compound verb prefixes. Examples of such prefixes are: *af*, *bij*, *in*, *op*, *over*, *uit*.

After testing there were a few verbs that were not stemmed properly due to this way of splitting, these were added to the exception list. The exception list consists of edge cases to which our stemming rules do not apply, e.g. irregular verbs. After the word is initially passed through the stemming algorithm, it is checked against the exception list to see if further lemmatisation is needed.

The Dutch language contains roughly 235 irregular verbs (Haeseryn et al., 1997). These verbs have different vowels and structures in different tenses, although they should stem to the same root. The stemmed version of the irregular verbs are added to the exception list.

For stemming it is important to know if a verb is singular or plural. Verbs that have characteristics of being an infinitive (suffix *-en*) obtain plural, whereas all others obtain singular. There are infinitives where the suffix is *-enen*, for example in *rekenen* (to calculate) and *zegenen* (to bless): the singular form of these verbs do have a suffix of *-en*. To prevent these singular form verbs to be processed as plural, these verbs are added to the exception list.

For readability purposes and stemming of adjectives, the stem that is used as the actual stem is the first-person singular present simple. Therefore, additional rules are implemented to transform the stem to the first-person singular present simple form. The rules are applied in a particular order based on Dutch grammar rules.

In the Dutch language, no word ends with a *v* or a *z*. There are exceptions, but these words originate from other languages. Examples of such words are quiz and jazz and could be stemmed incorrectly. Due to the fact that the number of such words is limited, a few of these are also included in the exception list. After removing the suffixes of a verb, thus reducing the verb to its root, the last letter is examined. If the last letter is a *v*, then it is replaced by the letter *f*, and likewise if the last letter is a *z*, then it is replaced by the letter *s*. These are known in the Dutch language as the *valse s en valse f* (*valse* translates to *false*).

To check if a verb is irregular, the exception list is called after affix removal. This is done in order to keep the size of the exception list to a minimum, as not all forms of a verb need to be stored.

The irregular verbs were obtained from a Dutch dictionary website (Mijnwoordenboek.nl, 2004). This website maintains a large database of almost all verbs, nouns and adjectives with their various forms.

When the verb is stemmed to the root, doubling of vowels is needed to bring it back to the actual stem. The first-person singular present simple form of *lopen* is *loop*. Following the stemming rules, the root of the word *lopen* would become *lop*. Therefore an additional *o* must be placed before the last letter.

This is also the case for the letters *a*, *e* and *u*. To see if a vowel is doubled, the algorithm looks at the last three letters of the stemmed word, which was originally a plural. If the first letter is a non-vowel and the second letter is a vowel, then the second letter gets doubled. To go back to the previous example of *lopen*, which was determined to be a plural verb in the first step, following the stemming rules suffix *-en* is removed. Afterwards the last three letters (*lop*) are examined. "*L*" is the first letter and a non-vowel, *o* is the second letter and therefore is doubled, resulting in *loop*, which indeed is the first-person singular present simple of *lopen*.

The final step in ensuring the correct stem is removing double letters at the end of a verb in case this is a non-vowel. This is due to the fact that these do not exist in the Dutch language.

3.2.2. Adjectives (BVNW), Noun (ZNW) and Adverbs (BW)

The adjectives require fewer rules to bring the adjective back to the root. The root of an adjective is defined as its shortest possible form, e.g. stripping suffixes indicating superlatives or grammatical gender. As a design choice, superlatives were brought back to the normal form. For example *goed* (good), *beter* (better), *best* (best) were all reduced to *goed* through the exception list.

The stem of a noun is defined as the singular form of the noun. Therefore, the first step is to determine if a noun is plural or singular. If the noun ends on suffix: *en*, *jes* or *s*, it is considered to be plural. There is an additional requirement on the suffix *s* to be a plural and that is a reversed doubling of vowel method. In Dutch, a noun that is actually in singular form but ends on *s* will generally have a repeated vowel before the *s*. Since this does not occur in plural form, such a noun is then marked as singular. For example, the noun *moeders* (moms) will be stripped of the suffix *s*, but *vaas* (vase) will not be stripped from the *s* due to the double *aa*.

The plural nouns are then stripped of their respective suffixes, that marked them as plural and checked against the exception list.

Since the adverbs are words that are likely to be removed after stemming due to lack of information gain, these are also not stemmed. Names of countries, people or companies follow many different rules to accurately stem them and are therefore currently tagged as a BW, effectively placing stemming of named entities and related terms out of scope.

3.3. Data Collection

To determine which algorithms to use in the final BT algorithm, documents were collected to conduct analysis and for training purposes. To increase the probability of 'correct' usage of the Dutch language, certain types of publications are preferred over others. An example can be Social Media information, where sentences can contain many grammar mistakes. The probability of correctly constructed Dutch sentences is likely to be higher in legal documents.

The data collection was done by downloading publicly available collective labour agreements (CLA) as PDF files from the website of the largest Dutch trade union: FNV (FNV, 2019). Besides the CLAs, pages from Wikipedia and Dutch children's books were used to diversify the words in our dataset. The PDF documents where parsed using a combination of Tika parser (Mattmann and Zitting, 2011) and Tesseract (Smith, 2007), both of which are open source in Python 3.6. All programming was conducted in the Anaconda environment using Spyder (3.3.4) and Jupyter Notebooks (5.7.8).

In total 252 CLAs, five pages from Wikipedia and three children's book chapters were parsed. The unique words were converted into a Pandas dataframe for further analysis. To prevent overfitting on words due to frequency of occurrence, the unique words were used to ensure assigning each the same weight during training. If the word *de* (the) would frequently be contained in the dataset, it could result in having the feature *de* (also a common OVT suffix in Dutch) more likely to contribute to labelling words as BW (adverb), instead of OVT (simple past tense). The manner in which class imbalance was handled is discussed in more detail in Section 3.4.

To obtain a ground truth for the training set, the dataframe of unique words was processed through a connection to the Frog algorithm using LaMachine (van Gompel and Hendrickx, 2019) via a virtual machine to obtain the tag for each word. Since Frog uses a taxonomy of tags that is more complex than needed for our classification purposes, Frog tags were converted into the six tag categories, shown in Table 1, which are also the categories the BT algorithm is trained to predict.

For purposes of simplification and since Dutch verbs can be assigned to one of three categories (OTT, OVT, VTT), these were the only categories used for verbs. There are differences in how sentences may be constructed, but looking at the prefix and suffix of the main verb is generally sufficient to assign a verb to one of those categories.

The final dataset consisted of 76,167 unique words after cleaning, with the distribution of the tags shown in Figure 1. During writing of the code, all words that were not a verb, noun or adjective were labelled BW. For stemming purposes it was a matter of convenience to list these as BW due to the fact that these words will not be stemmed.

3.4. Feature space

The words were converted into a feature space word. A feature space word is a word with a '_' symbol added to the beginning and end of the word. This ensures that in the next step it is still clear what the first and final letters of the word were. To convert the text to a vector space, we used bi-grams of the characters of each word, which were converted to a vector using one-hot encoding.

The feature space consisted of $27 \cdot 27 - 1 = 728$ different vectors. There are 26 letters in the Dutch alphabet, together with the new _ symbols on the beginning and the end of the

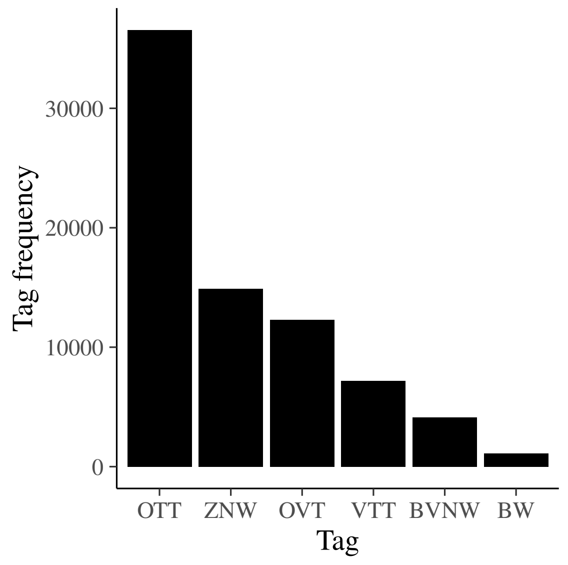

Figure 1: Distribution of tags for unique words in the dataset

word. The one excluded combination is _ _, i.e. the empty word.
The distribution of the features was skewed, which is to be expected in textual data. Some combinations are more common (e.g. *en*) than others, while some do not occur at all (e.g. *xd*) in the Dutch language. Based on manual inspection, features that occurred less than 200 times in the data were removed to decrease possible noise in the algorithms later on. For experimental purposes, all of the models were trained on the entire dataset and on the dataset with the feature selection.
Since the distribution of the different tags (Figure 1) was imbalanced, the different train and test sets were carefully constructed. First a stratified (80/20) sample was obtained from DS_1. In order to keep information while mitigating the risks of over-sampling, we used an approach called SMOTE (Chawla et al., 2002). SMOTE is an over-sampling method in which minority classes are over-sampled by creating synthetic observations.

3.5. Model evaluation

To determine how accurate the entire model (i.e. tagging and stemming modules) is, a new evaluation dataset was used consisting of four documents: ES_n for $n \in \{1,2,3,4\}$. Every ES_n was stemmed manually and independently twice: once by the first author, and once by a group of two native speakers. The Cohen's kappa was calculated as a measure of inter-rater agreement on the document and thus to determine the accuracy of the manually checked stems (Berry and Mielke Jr, 1988). The Cohen's kappa was calculated at 0.96. Given this high inter-rater agreement, we considered the manual tags sufficiently accurate to be used in further evaluation.
Words that were disagreed upon during the check, e.g. due to a typo or ambiguity, were analysed and changed where necessary. The documents that were used were not previously seen by the tagging algorithm.

Since the type of language that the models were trained on mainly focused on CLAs, it is interesting to see how the models perform on other types of text.
The description of the four ES_n are presented below

1. ES_1: A chapter of the Dutch translation of Roald Dahl's book "the BFG" (i.e. "de GVR" in Dutch), to see how the model handles non-existing words (Dahl, 2013).

2. ES_2: A newspaper article (Marcelis, 2019) on the earthquakes happening in the northern part of the Netherlands due to gas extraction. This article contains more technical language.

3. ES_3: A Wikipedia page containing information on the university where this model has been developed (Vrije Universiteit Amsterdam) (Wikipedia, 2019).

4. ES_4: A vacancy for a position at the Dutch football club Ajax, to examine how the model performs on business language text documents (Ajax, 2019).

The stemming algorithms were tested along three dimensions: understemming, overstemming and computational speed. The combinations and results of the various models will be discussed in Section 4..
The documents were manually processed and so Frog was not used to determine the tags taken as ground truth. As a result, evaluation of the performance of the Frog algorithm was made possible.
An overview of ES_n is presented in Table 3. The distribution of tags shows that the OVT and VVT categories are underrepresented in this dataset. The number of words that were unknown to the CELEX database will affect the performance of the Frog algorithm, since Frog uses CELEX to determine the tag. If Frog is unable to determine the tag through CELEX it uses an additional algorithm, but the exact workings of this additional algorithm are unknown to the authors.

ES_n	ES_1	ES_2	ES_3	ES_4	Total
Words	390	340	443	340	1513
Unique Words	230	194	270	201	772
CELEX unknown	27	17	40	19	102
OTT	34	24	41	18	103
ZNW	48	47	88	47	224
OVT	10	8	0	10	24
VTT	3	9	3	7	22
BVNW	33	17	33	31	110
BW	75	72	65	69	187

Table 3: ES_n overview in unique word counts

To calculate understemming and overstemming, the unique stems of the manually checked file were taken as the ground truth, as using Frog for this purpose led to some incorrectly tagged words. Understemming and overstemming were computed as follows: let S be the set of all unique stems found by some algorithm in dataset D, and let $S^{'}$ be the set of all manually found stems in D. Furthermore, let $g(s^{'})$ be a function which maps some $s^{'} \in S^{'}$ to the set of uniquely found algorithm stems, for all words which were manually stemmed s' in D. Likewise, $g^{-1}(s)$ maps some algorithmically found stem s to the set of all manual stems, for words which were algorithmically stemmed s in D. Then understemming and overstemming, written as λ_D^- and λ_D^+ respectively, are computed as

$$\lambda_D^- = \frac{\sum_{s' \in S'} \left(|g(s')|\right)^{-1}}{|S'|}, \tag{2}$$

$$\lambda_D^+ = \frac{\sum_{s \in S} \left(|g^{-1}(s)|\right)^{-1}}{|S|}. \tag{3}$$

Computational speed was determined by using the method "cell magic" (The IPython Development Team, 2019) in Jupyter Notebook. The averages were taken from 100 test runs and 100 loops each. Experiments were conducted on a computer with an Intel i7 processor, using three out of four cores, running at 2904 MHz, using 16 GB of RAM, operating on Windows 10.

4. Results

The ES$_n$ was processed to determine understemming, overstemming and the computational speed of the various stemming algorithms. The *Total* that is mentioned in these sections relates to the concatenated ES$_n$, that is, the ES$_n$ were combined and from these the unique words were used. This is done to ensure that the number of words in a document does not influence the average on multiple documents.

4.1. Performance of the models

Looking at understemming, Table 4 shows that the tagging algorithms outperform Lancaster, Porter and Snowball, while the manual and model tagged algorithms perform similar to the case where tagging is done using the Frog algorithm. This result seems to be consistent over all ES$_n$. This is to be expected, since the pure stemming algorithm has less information than an IDM algorithm on how to accurately stem.

There is some variation in the understemming accuracy among the various ES$_n$, but all tend to a follow the same trend. The dataset where all documents were concatenated (Total), scores the worst. The dataset containing information on the university from the Wikipedia page, scores the best.

Surprising is the fact that the Frog algorithm actually performs well on the datasets where the number of unknown words to the CELEX database (1 and 3), was largest. This high performance suggests that the Frog algorithm, that gets activated if a word is unknown, performs very well.

The overall best scoring method was the BT algorithm using the manual tags to stem the words. This was to be expected, since the BT algorithm was designed to reduce a word to a stem that corresponds with the manual stem.

Table 5 shows similar results as Table 4, although the differences are smaller. The Porter algorithm in particular shows high accuracy in overstemming, indicating that it is conservative in how much of a word is truncated. The BT algorithm performs similar to the Frog tagger, despite not relying on a combined approach of almost brute-force lookup for known words and Frog's well-performing algorithm for handling unknown words.

ES$_n$	ES$_1$	ES$_2$	ES$_3$	ES$_4$	Total
Porter	0.93	0.93	0.97	0.94	0.91
Lancaster	0.93	0.95	0.98	0.94	0.93
Snowball	0.93	0.95	0.97	0.94	0.93
BT + Frog tag	0.98	0.98	0.99	0.98	0.97
BT + Manual tag	0.98	0.98	0.99	0.98	0.99
BT + model tag	0.97	0.98	0.98	0.98	0.97

Table 4: F_1-score on understemming of the algorithms on ES$_n$

ES$_n$	ES$_1$	ES$_2$	ES$_3$	ES$_4$	Total
Porter	0.98	0.98	0.99	0.98	0.98
Lancaster	0.97	0.97	0.96	0.97	0.95
Snowball	0.98	0.99	0.98	0.98	0.97
BT + Frog tag	0.99	0.99	0.97	0.99	0.97
BT + Manual tag	0.98	0.98	0.98	0.99	0.97
BT + model tag	0.99	0.99	0.97	0.99	0.97

Table 5: F_1-score on overstemming of the algorithms on ES$_n$

The third performance measure was computational speed. Table 6 only takes into account to computation time needed to stem the words after a tag was determined by the tagging module. An important note while considering the computation times is that Porter, Lancaster and Snowball have been written in C, which may be considered a low-level programming language compared to Python. The BT algorithm was written in Python 3.6, which is a higher-level programming language. Some optimisation was done in the programming, but there is still room for improvement. Programs written in C are usually faster than programs written in Python (Oliphant, 2007). A comparison study (Prechelt, 2000) among various programming languages showed that the average computation time could be improved by a factor 2 or 3, whereas the memory consumption would decrease similarly.

Table 6 shows that the BT algorithm is considerably slower, in part due to the reasons previously mentioned, but is still able to stem a single document in under one second (1000ms). Through parallelisation more speed improvements could be made, but this was not implemented for this paper.

Table 7 shows the difference in computation time on determining a tag between the Frog algorithm and the BT algorithm. Again an important note while interpreting Table 7 is that Frog runs on a server. Therefore the computation speed is not only determined by how fast the algorithm is able to

ES_n	ES_1	ES_2	ES_3	ES_4	Total
Porter	9.21	9.15	10.60	8.44	30.60
(Std. dev)	(2.04)	(3.13)	(2.22)	(2.84)	(4.71)
Lancaster	9.10	7.83	10.80	7.46	29.40
(Std. dev)	(2.21)	(1.66)	(2.30)	(1.74)	(3.27)
Snowball	6.20	5.67	7.67	5.74	20.60
(Std. dev)	(1.35)	(1.54)	(2.39)	(1.43)	(3.76)
BT	737.14	702.84	754.18	676.38	2,580
(Std. dev)	(18.63)	(17.25)	(19.31)	(17.45)	(65.90)

Table 6: Computation time of stemmers in ms: mean (std. dev.) of 100 runs, 100 loops each

process the information, but also by the internet speed of the user. Before a file can be stemmed, there are some pre-processing steps required and these are taken into account when Table 7 is examined.

ES_n	ES_1	ES_2	ES_3	ES_4	Total
Frog	79.32	49.81	53.12	47.29	136.84
(Std. dev)	(3.23)	(1.10)	(4.14)	(0.98)	(7.38)
BT	2.83	1.79	2.11	1.64	6.21
(Std. dev)	(0.21)	(0.14)	(0.18)	(0.17)	(0.92)

Table 7: Computation time of the Frog and BT tagger in seconds: mean (std. dev.) of 100 runs, 100 loops each

As the Frog tags were considered the ground truth during the training of the various models, a comparison is made between the manual tags and the Frog tags. The confusion matrix of this comparison is presented in Table 8, with the Frog F_1 scores being presented in Table 9. Due to the fact that the Frog tags are taken as the ground truth for the BT training, the F_1 scores shown in Table 9 would ideally have been higher.

Frog \ Manual	OTT	ZNW	OVT	VTT	BVNW	BW	Total
OTT	**154**	10	0	3	2	0	169
ZNW	2	**273**	5	0	17	2	299
OVT	0	0	**36**	0	6	0	42
VTT	0	3	2	**18**	1	0	24
BVNW	0	5	1	2	**131**	0	139
BW	2	65	4	0	30	**454**	555
Total	158	356	48	23	187	456	**1228**

Table 8: Confusion matrix Frog and manual tags

Tag	F_1 - Score
OTT	0.9249
ZNW	0.8235
OVT	0.8000
VTT	0.7660
BVNW	0.8037
BW	0.8447

Table 9: F_1-scores of the Frog tagging algorithm on the manual dataset

5. Conclusion & Discussion

5.1. Conclusion

The goal of this paper was to build a new stemmer that would be more accurate than the state-of-the-art stemmers (i.e. Porter, Lancaster and Snowball), as well as faster than the Frog algorithm, for the Dutch language. Based on the results discussed in Section 4., it seems that the new BT algorithm works well for the stemming and tagging of words in the Dutch language. Our findings indicate that this approach indeed shows promise, while still leaving room for a high degree of technical optimisation.

Out of the three algorithms - *MLR*, *NN*, *XGB* - that were tested, XGB was shown to be the most accurate for PoS tagging purposes. This is a surprising result, since no other instances of XGB being used as a tagging algorithm were found in the literature, despite the fact that it showed promise with other classification problems (Fan et al., 2018).

The number of rules that are implemented in the BT algorithm is relatively small compared to the current state-of-the-art stemming algorithms (i.e. Porter, Lancaster, Snowball). This is mainly due to the fact that the words follow tag-specific rules, obtained from the tagging algorithm.

The combination of the XGB model and new stemming rules manages to perform well with the manually tagged and stemmed database in terms of under- and overstemming, as well as computation speed, and thereby resulted in the new BT algorithm.

As can be expected, the Frog algorithm performed its tasks slower in the case that it encountered words that are not included in the CELEX database (i.e. ES_1 on the BFG). Surprisingly though, its resulting accuracy was fairly high, in some cases even performing better than situations wherein words could be found in the CELEX database.

The newly-built BT algorithm is not yet completely optimised in terms of programming and its rule set. Programming it in a language that is faster than Python 3.6, such as C, should increase the computation speed. Additional experiments should be conducted on larger datasets to see whether the stemming rules need tuning for the accuracy to be further improved.

The overall speed of the BT algorithm is faster than the current Frog algorithm. Combining this with the fact its performance with under- and overstemming was slightly worse than when using the Frog algorithm, yet still better than the truncating algorithms, suggest that this new development can be considered a preliminary success. The entire process of inserting PDF documents into the algorithm, preprocessing, then tagging and stemming them takes an estimated time of around 8 to 10 seconds. Previous literature suggests that it can be inferred that this time can be improved by factor of 2 or 3.

Finally, after having tested the performance of the Frog algorithm, its performance turned out to be less strong than initially thought. This is especially relevant due to the fact that it was considered the ground truth for the training models. The F_1 scores combined with the confusion matrix from Tables 8 and 9 show that 65 nouns were predicted to be an adverb (BW), which suggests that the BT algorithm would not have stemmed these nouns when provided the Frog tag or (potentially) when using its own PoS tagging model.

5.2. Discussion

The new tagging module only differentiates between six possible tags, while there are more that could be stemmed. For example, various pronouns can be stemmed back to their respective personal pronouns. Numerical word types can also be reduced to one of the overarching numerical word types. However, this changes the meaning of the word which is not always desirable. Currently, they are tagged as an adverb, and are therefore not stemmed.

The evaluation dataset, though small, provides an indication of the performance of the algorithms on various types of documents. Extending the dataset could prove worthwhile, e.g. if it is to be a standard approach for benchmarking stemming and/or tagging algorithms for the Dutch language. Extending the evaluation dataset would, however, be a rather tedious job as it requires manual annotation.

Like any other machine learning model, the XGB-classifier can also be enhanced further by adding more training data. By expanding the training set with a larger variety of correctly tagged words, XGB could make more accurate predictions. The grid search space can also be expended, which might result in a better model.

With the promising results presented in this paper and a clear path for the future, the goal of building a new stemming algorithm that is more accurate than the current state-of-the-art stemmers, but faster than the Frog algorithm, has been achieved.

5.3. Further Research

Since there was no available data that was pre-tagged and included full sentences, sequential models could not yet be implemented. Despite the encouraging results achieved in this paper with respect to under- and overstemming, this could be the logical next step, as the literature suggests that this could enhance the tagging performance, and thereby stemming algorithms (Elbayad et al., 2018). The sequential model should not replace the current tagging module but may be used as a source of additional correction after a word is initially tagged by the tagging algorithm.

One suggestion would be to make use of anchor points that are provided to the tagger in order to assist the sequential model. Anchor points in a sentence would be very common words: personal forms, articles, forms of the most used verbs (to have, to shall, to do, to will). In addition, this could be combined with a Named-entity recognition algorithm to identify names and geographic locations (Nadeau and Sekine, 2007).

The sentence information can be used in a pervasive attention model which would not only look at sequences but also anchor points, as described by (Elbayad et al., 2018). Combining the predicted tag with a prefix and suffix model would further enhance the probability of selecting the correct tag.

Further analysis on feature importance has to be conducted to determine whether certain features should be excluded from the BT algorithm to improve the XGB model, due to the possibility of overfitting.

As the BT algorithm works for the Dutch language, this approach could be investigated for similar languages as well. Finally, because of the many exceptions in the Dutch language, further research regarding lemmatisation could yield solutions that are more accurate than our BT algorithm. However, since lemmatisation generally replaces inflected words with their canonical form through looking up known words in a database, this approach will likely come at the expense of robustness, e.g. in coping with neologisms.

6. Bibliographical References

Ajax, P., (2019). *Ajax is op zoek naar een Assistent-Controller*. Retrieved from: `https://www.ajax.nl/club/vacatures.html`, accessed February 7, 2019.

Berry, K. J. and Mielke Jr, P. W. (1988). A generalization of Cohen's kappa agreement measure to interval measurement and multiple raters. *Educational and Psychological Measurement*, 48(4):921–933.

Bosch, A. v. d., Busser, B., Canisius, S., and Daelemans, W. (2007). An efficient memory-based morphosyntactic tagger and parser for Dutch. *LOT Occasional Series*, 7:191–206.

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

Dahl, R. (2013). *De GVR*. de Fontein Jeugd.

Elbayad, M., Besacier, L., and Verbeek, J. (2018). Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction. *arXiv preprint arXiv:1808.03867*.

Fan, J., Wang, X., Wu, L., Zhou, H., Zhang, F., Yu, X., Lu, X., and Xiang, Y. (2018). Comparison of Support Vector Machine and Extreme Gradient Boosting for predicting daily global solar radiation using temperature and precipitation in humid subtropical climates: A case study in China. *Energy conversion and management*, 164:102–111.

FNV, (2019). *Cao Sector*. Retrieved from: `https://www.fnv.nl/cao-sector`, accessed January 8, 2019.

Haeseryn, W. J.-M., Romijn, K., Geerts, G., Rooij, J. d., and Van den Toorn, M. C. (1997). Algemene Nederlandse spraakkunst [2 banden].

Jivani, A. G. et al. (2011). A comparative study of stemming algorithms. *International Journal of Computer Applications in Technology*, 2(6):1930–1938.

Kontostathis, A., Edwards, L., and Leatherman, A. (2010). Text mining and cybercrime. *Text Mining: Applications* *and Theory. John Wiley & Sons, Ltd, Chichester, UK*, pages 149–164.

Marcelis, H., (2019). *Groningen getroffen door vierde aardbeving in twee weken tijd.* Retrieved from: `https://www.ad.nl/binnenland/groningen-getroffen-door-\vierde-aardbeving-in-twee-weken-tijd`, accessed June 13, 2019.

Mattmann, C. and Zitting, J. (2011). *Tika in action.* Manning Publications Co.

Mijnwoordenboek.nl, (2004). *Mijnwoordenboek werkwoorden.* Retrieved from: `https://www.mijnwoordenboek.nl/werkwoorden/NL/`, accessed March 03, 2019.

Monz, C. and De Rijke, M. (2001). Shallow morphological analysis in monolingual information retrieval for Dutch, German, and Italian. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 262–277. Springer.

Moral, C., de Antonio, A., Imbert, R., and Ramírez, J. (2014). A survey of stemming algorithms in information retrieval. *Information Research: An International Electronic Journal*, 19(1):n1.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20.

Paice, C. D. (1990). Another stemmer. In *ACM Sigir Forum*, volume 24, pages 56–61. ACM.

Pang, B., Lee, L., et al. (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Phua, C., Lee, V., Smith, K., and Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*.

Porter, M. F. et al. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Porter, M. F., (2001). *Snowball: A language for stemming algorithms.* Retrieved from: `http://snowball.tartarus.org/algorithms/dutch/stemmer.html`, accessed February 16, 2019.

Prechelt, L. (2000). An empirical comparison of C, C++, java, perl, python, rexx and tcl. *IEEE Computer*, 33(10):23–29.

Schmid, H. (1994). Part-of-speech tagging with neural networks. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 172–176. Association for Computational Linguistics.

Smith, R. (2007). An overview of the Tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE.

The IPython Development Team, t., (2019). *Built-in magic commands.* Retrieved from: `https://ipython.readthedocs.io/en/stable/interactive/magics.html`, accessed June 2, 2019.

Van der Beek, L., Bouma, G., Malouf, R., and Van Noord, G. (2002). The Alpino dependency treebank. In *Computational linguistics in the Netherlands 2001*, pages 8–22. Brill Rodopi.

Van der Wouden, T. (1990). Celex: Building a multifunctional polytheoretical lexical data base. *Proceedings of BudaLex*, 88:363–373.

van Gompel, M. and Hendrickx, I. (2019). LaMachine: A meta-distribution for NLP software. In *Selected papers from the CLARIN Annual Conference 2018, Pisa, 8-10 October 2018*, number 159, pages 209–221. Linköping University Electronic Press.

Wikipedia, (2019). *Vrije Universiteit Amsterdam.* Retrieved from: `https://nl.wikipedia.org/wiki/Vrije_Universiteit_Amsterdam/`, accessed June 12, 2019.

Willett, P. (2006). The Porter stemming algorithm: then and now. *Program*, 40(3):219–223.