Opleiding Informatica

Can a rule-based algorithm accurately correct whitespace errors?

Pim Bax

First supervisor:
Suzan Verberne
Second supervisor:
Hugo de Vos

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

April 22, 2021

Abstract

In this paper we attempt to create a rule-based approach to determine if two words in a Dutch sentence contain a split error (i.e. it is incorrectly split up, as opposed to being compounded). Since compounding is ultimately based on a set of rules, any algorithm that can correctly determine the exact function of a word in a sentence should be able to apply these rules to output a properly compounded word consisting of the two words which were previously written as two split-up words.

The main issues that need to be dealt with in applying such a rule-based approach is correctly parsing a sentence which features a split error (and is therefore not grammatically correct), and applying the proper compounding rules, such as determining which letters should be inserted between the two compounded words to properly compound them.

This paper seeks to produce an algorithm that can be used to reduce the amount of split errors that get approved by existing spell checkers, and hopes to provide a new approach that could be applied to spell checkers to more accurately detect and correct this type of error.

Contents

1 Introduction

The Dutch language, along with several other Germanic languages, heavily features compound words in its grammar. This allows for several situations in which adjacent words, such as noun-noun pairs (for example “fietsbord”) or some adjective-noun compounds (such as “laagsteprijsgarantie”), must be written as one word. However, because these compound words can be constructed from virtually limitless combinations of words, spell checkers that rely on dictionary lookups are unable to correctly mark correctly formatted compound words as being spelled correctly, and commonly flag these words as needing spaces in between them. See Figure 1 for an example in Google Docs.

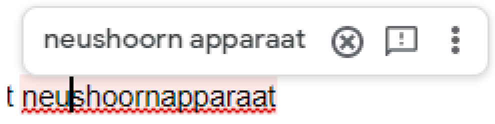
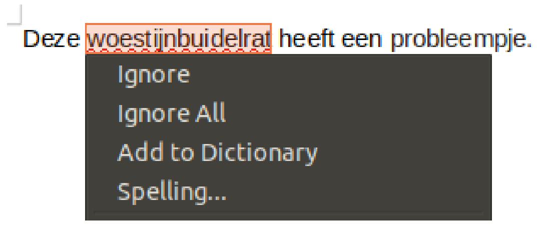


Figure 1: An example of the Google Docs spell checker inserting a split error.

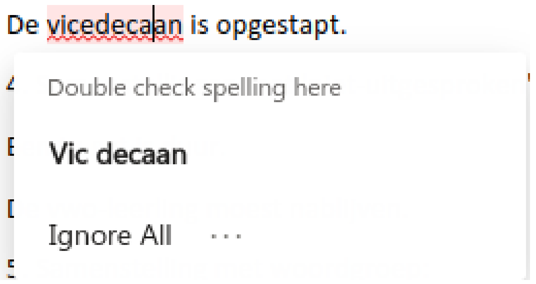
Although most spell checkers are able to handle simple two-word compounds fairly well, they still struggle with many compounds consisting of more than two words. These either do not get recognised, or the spell checker proposes splitting up the compounded word into its constituents. See Figure 1 for an example in LibreOffice, which uses Hunspell’s OpenTaal dictionary.



Deze woestijnbuidelrat heeft een probleempje.

Figure 2: An example of the LibreOffice spell checker (Hunspell) not recognising a word due to compounding.

Even the spell checkers that falsely flag very few compounds still struggle with some types of prefixes or suffixes. See Figure 1 for an example in Microsoft’s OneDrive.



De vicedecaan is opgestapt.

Figure 3: An example of the OneDrive spell checker not recognising a word due to it being a prefix.

Some spell checkers, such as Valkuil[1] and Microsoft's spell checker[2] are relatively advanced in their recognition of compounded words. These virtually never mark compounded words as requiring to be split up, and occasionally do suggest compounding of words. Even these spell checkers are lacking in their correction of existing split errors, however. As an example sentence, we used "Ik heb een vakantie dag met de schoon familie." to test if split errors ("vakantiedag" and "schoonfamilie") would be recognised. Both Google Docs and LibreOffice were unable to recognise either of these split errors. OneDrive and Valkuil were able to recognise the much more common "schoonfamilie", however they both failed to recognise "vakantiedag" having a split error in that sentence. See Figure 1 for an example.

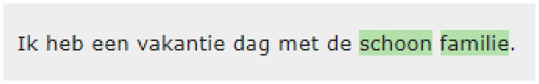

Ik heb een vakantie dag met de schoon familie.

Figure 4: An example of Valkuil failing to recognise one of two split errors in a sentence.

This tendency for spell checkers to commonly introduce split errors, as well as their inability to recognise split errors, led to the research question for this thesis.

1.1 Thesis overview

This thesis aims to answer the question "To what extent can a rule-based algorithm detect and correct Dutch split errors?"

This paper aims to create a rule-based algorithm that follows the Dutch grammatical rules that determine if two words are meant to be compounded. It relies on existing language analysis tools which make use of machine learning – specifically, SpaCy[3] – to interpret sentences and tag each of the words (tokens) in those sentences using their Part-of-Speech tags. These tagged tokens are then fed into the algorithm, which checks a series of grammatical rules in order to determine whether two adjacent tokens should be combined or left separate.

The goal of this paper is not to produce an algorithm that will function as a standalone spell checker. Due to the specific category of grammatical error it will be detecting, it is intended to work in conjunction with other spell checkers.

1.2 Why use a rule-based approach?

Many parts of language are subject to a series of rules. In this project, we try to collect a full list of all applicable compounding rules, as far as this is possible. Given that it is possible to perfectly analyse sentences and correctly tag the grammatical function of all words in a sentence, it should be possible to apply these rules to properly determine if two words should be compounded. This project seeks to show to what extent such an approach would be viable, given the tools that are currently available.

[1] Valkuil.net spellingcorrector, http://valkuil.net/
[2] We used Microsoft OneDrive to test this spell checker: https://onedrive.live.com/
[3] SpaCy, Industrial-Strength Natural Language Processing; https://spacy.io/

2 Related Work

In previous work done by Beeksma et al. (2018), four competing algorithms were created to correct several different types of grammatical errors, among which were split errors. All four competing algorithms made use of different approaches, but all made use of statistical and machine learning based analyses. These algorithms were all focused on making as many accurate corrections of grammatical errors as possible, which means that none of the algorithms had split errors as their primary target. In Beeksma et al. it becomes clear that each of these algorithms performed relatively poorly at detecting split errors when compared to most other types of error. As seen in the results, the two most common errors in the given text were capitalization errors and split errors, and out of the four algorithms only one was able to achieve more true positives than false negatives. The amount of false positives in a specific category were not given.

Another, much older paper, Vosse (1992), references to Dutch spelling correction, also specifically mentions split errors ('split compounds'). Although precise numbers on the efficacy of detecting and correcting split errors using their approach are not given, they do state that "The detection of punctuation errors and split compounds still needs improvement". Their approach of parsing could be modified to better interpret sentences when they contain split errors, however more modern sentence parsing have likely superseded this type of parser in modern systems. Using this approach along with a more modern system like Valkuil could prove useful, however.

In Tijhuis (2014), context-based generic spelling correction is analysed, based on Wikipedia revision history. This paper is also referenced in the description of the 'FRAUNHOFER' team's approach in Beeksma et al. (2018). It builds a database of spelling corrections made in Wikipedia revisions, and uses several methods to find candidate words to replace a spelling error. This paper mainly focuses on single-word errors, however, and shows 25 compound errors in their Table 3.1, with none of them being corrected with this approach. In their introduction, they explicitly state that this paper does not focus on compound words. This again shows that generic spell checkers are typically not well-suited for tackling split errors.

While not all other languages feature split errors like the Dutch language does, many more languages do feature a tangentially related type of error: the real-word spelling error. This is a spelling error that turns a correctly spelled word into another valid word that can be found in the dictionary, which does not fit the sentence in that position. Several approaches exist to correct these types of real-word errors; one approach is that of Hirst and Budanitsky (2005) which uses semantics of a given *suspect* word and compares it to the semantics of other situations that word is used, using the heuristic that "with the exception of a handful of frequently misspelled words, misspellings rarely tend to be repeated in a document". While it may be useful to rely on data that has been verified to be a correctly spelled compound, such as dictionary data of known compounds, this project is more focused on detecting split errors in situations where the compounded word may have never occurred before, making it impossible to compare compounds to existing pieces of text.

Many other algorithms that try to detect real-word spelling errors, make use of n-grams. While using n-grams may be useful in some future modifications to the approach we used in this paper, this approach again has the downside of being limited to corrections to existing combinations of words.

Another paper that focuses on real-word spelling errors (also using n-grams), Verberne (2002) goes into the topic of compounded words more. It dedicates a subsection to "The word boundary problem". While this issue is much less prevalent in English than in Dutch, this issue still exists in English, commonly for certain bound morphemes, such as 'together'. However, while some suggestions to solve this issue are made in that paper, no specific solutions to this problem that would be applicable to the situation in Dutch are mentioned. This paper does state that detecting when a word has been incorrectly split up is very difficult.

While there appear to be few papers focusing on rule-based approaches, Mangu and Brill (1997) takes a rule-based approach, while acquiring the rules that are being applied automatically. While not applied in this paper, the approach used here may be one that would be beneficial in future research into this type of spelling correcting. In particular, they lay out the approach to "Apply [a] transformation to a copy of the corpus" and then "Score the resulting corpus according to the objective function". A similar approach could prove useful in filtering out false positives.

3 Methods

In order to more efficiently process sentences, we do our analysis only on individual sentences. This is mostly due to the analysis tool SpaCy splitting up its analysis into sentences. This means context from surrounding sentences that might disambiguate the semantics of a given sentence is not taken into account.

These individual sentences are then fed into SpaCy, which labels individual words with several of their grammatical functions. For example, it labels their parts-of-speech (POS, e.g. noun, verb, adjective), their grammatical function (e.g. subject, object), as well as their dependencies to other words in the sentence (e.g. adjective modifier, apposition, oblique nominal). Spacy's output is discussed further in Subsection 3.1. Then, each of the words in the sentence is looked up on Wiktionary[4], which is used to consult for several grammatical properties of the word. This is further discussed in Subsection 3.2.

When all necessary data is collected for every word in a sentence, all rules are then applied in a specific order. We determined what order to apply these rules by consulting several different websites that provide details on Dutch grammar rules, as well as relying on personal experience with the Dutch language. The rules that are applied, are then fine-tuned based on experimentation with Wikipedia articles. More information about this approach is described in Subsection 3.3.

In order to create an algorithm that follows grammatical rules and applies them to all applicable combinations of tokens, we first need to structure the grammatical rules into a format understandable by computers. How these grammatical rules were obtained is discussed in Subsection 3.4.

3.1 Interpreting SpaCy output

SpaCy analyzes sentences to extract as much machine-readable information from sentences as it can. It splits sentences into tokens, which typically represent individual words, or special characters such as punctuation. Each of these tokens get tagged with a Part-of-Speech (POS) tag, the grammatical dependencies in the sentence, its grammatical base form, as well as several other features that are less important to this thesis, such as entity labels for proper names and similar constructs.

SpaCy will output a list of tokens, each of which contains information about the POS, the dependencies[5], as well as more details about these properties. These tokens are then looked up on

[4]Dutch Wiktionary: WikiWoordenboek; https://nl.wiktionary.org/
[5]Universal Dependency Relation tags, https://universaldependencies.org/u/dep/

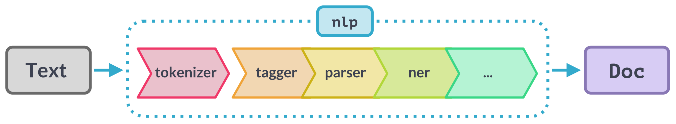


Figure 5: General workflow for SpaCy's sentence processing.
Source: https://spacy.io/usage/processing-pipelines

the Dutch Wiktionary (WikiWoordenboek) to obtain more information about a given word.

The token output from SpaCy is generally fairly accurate when analysing sentences with correct grammar. However, when split errors are present in a sentence, SpaCy regularly produces incorrectly tagged tokens. For example, in the sentence "Het maximum gewicht is tien kilo", "maximum" is tagged as a noun, and "gewicht" is tagged as a verb. Their grammatical dependencies are marked as 'adjectival modifier' and 'nominal subject', however.

Because SpaCy gets to analyse grammatically incorrect sentences (due to the split errors), the output POS might be tagged in a way that is grammatically impossible. In the previous example, 'gewicht' was marked as a verb, even though that word can only ever be used as a noun. To correct this, Wiktionary is also checked to see if a word can be used as either of four types of word: a noun, adjective, verb or adverb. If the Wiktionary page makes no mention of that word being used as one of those four POS categories, but SpaCy does label it with that POS, that word's function in the sentence is marked as being impossible. This data is then compared to the output of SpaCy.

In order to determine what the grammatical function of a word should be corrected to, the following steps are then taken as a heuristic:

1. If Wiktionary agrees the POS SpaCy tagged it as is a possible use of the word, the POS is left unchanged.
2. If Wiktionary disagrees but does list the word as a noun, the SpaCy POS is overwritten to be a noun.
3. If Wiktionary disagrees but does list the word as an adjective, but not a noun, the POS is overwritten to be an adjective.
4. If Wiktionary disagrees but does list the word as a verb, but not as either of the above POS tags, the POS is overwritten to be an adjective.
5. If Wiktionary only lists the word as an adverb, the POS is overwritten to be an adverb.
6. If Wiktionary does not provide the above information, the POS is left unchanged.

In the same example of 'maximum gewicht' which we used earlier, the POS for 'gewicht' is overwritten from verb to noun, following point 2 of this list. Anecdotally, rule #2 is the rule that gets triggered most commonly, especially in overwriting a word that was tagged as an adjective.

3.2 Gathering information from Wiktionary

Wiktionary is used to gather several details about each word in a sentence. Several different properties for every word in a sentence are collected, in order to most accurately analyse each word. These details and reasons for gathering this information are:

- Whether or not the word is marked as an acronym; acronyms always get compounded with a hyphen[6], except in special cases where the acronym is pronounced as a normal word.
- What the numerical value of that word is, if a word represents a number; words representing large numbers (over a thousand) never get compounded with other numerical words[7]. This rule is further described in Subsection 3.4.
- A list of related words, as listed in Wiktionary; can be used to determine how a word is compounded, e.g. with an *–s–* in between the compounded words.
- Different forms of the word:
  - For nouns, the plural and diminutive forms; several rules depend on plural forms to determine compounding rules.
  - For relevant nouns, their feminine or masculine forms; feminine forms of masculine words sometimes determine if an *–e–* needs to be inserted in the compound.
  - For adjectives, the different adjective conjugations; some conjugations do get compounded, whereas other conjugations do not.

Aside from this data, we also attempt to determine which way a word is compounded. Based on compounding rules we will discuss in Subsection 3.4, the left hand part of a compound determines in which way two words are compounded (and which conjugation the word gets before compounding). This means that, if Wiktionary has an entry for an existing compound which starts with a certain word, we can use that to determine how that word should get compounded in general.

To find this 'general' prefix of a word that should be compounded, a process is used which is illustrated in Figure 6. For every word `W` a list of Wiktionary pages is obtained that start with `W`, using a prefix search. This list of pages is then iterated through, until a page `P` is found that explicitly states that page is about a word that is a compound of `W` and another word `V`. Then `V` is removed from the end of `P`, and all trailing hyphens are trimmed from this word. The resulting string `S` will then contain `W` along with its compounding conjugation.

An example of this procedure is the word `W` 'bedrijf'. This word is looked up, and the first page returned that starts with this prefix is `P` 'bedrijfjes'. That word does not have any information about it being a compound of 'bedrijf', so the next page is checked. This is `P` 'bedrijfs-pc-netwerk', which does specify that it is a compound word of `W`, and `V` 'pc-netwerk'. The word `V` is then removed from the end, resulting in the original word needing to be compounded with "bedrijfs-". The hyphen is then trimmed from the end, giving the conjugation prefix `S` 'bedrijfs' for any compound word where the left-hand side is 'bedrijf'.

During development of this algorithm, this was determined to be the most reliable method of obtaining compounding conjugations for words that are compounded with an *–s–*, as well as being a catchall for all types of conjugations, including those that follow clear rules, and those with irregular conjugations such as *–er–* like in '*kinderopvang*'. If no Wiktionary pages about a compound word starting with `W` are found, compounding is done following the rules described in Subsection 3.4.

[6]https://onzetaal.nl/taaladvies/afkorting-in-samenstelling/: "als een van de delen van de samenstelling een afkorting is, zoals tv, is er een streepje nodig."

[7]See also https://woordenlijst.org/leidraad/6/9, a Dutch website describing rules for writing Dutch numbers with words.

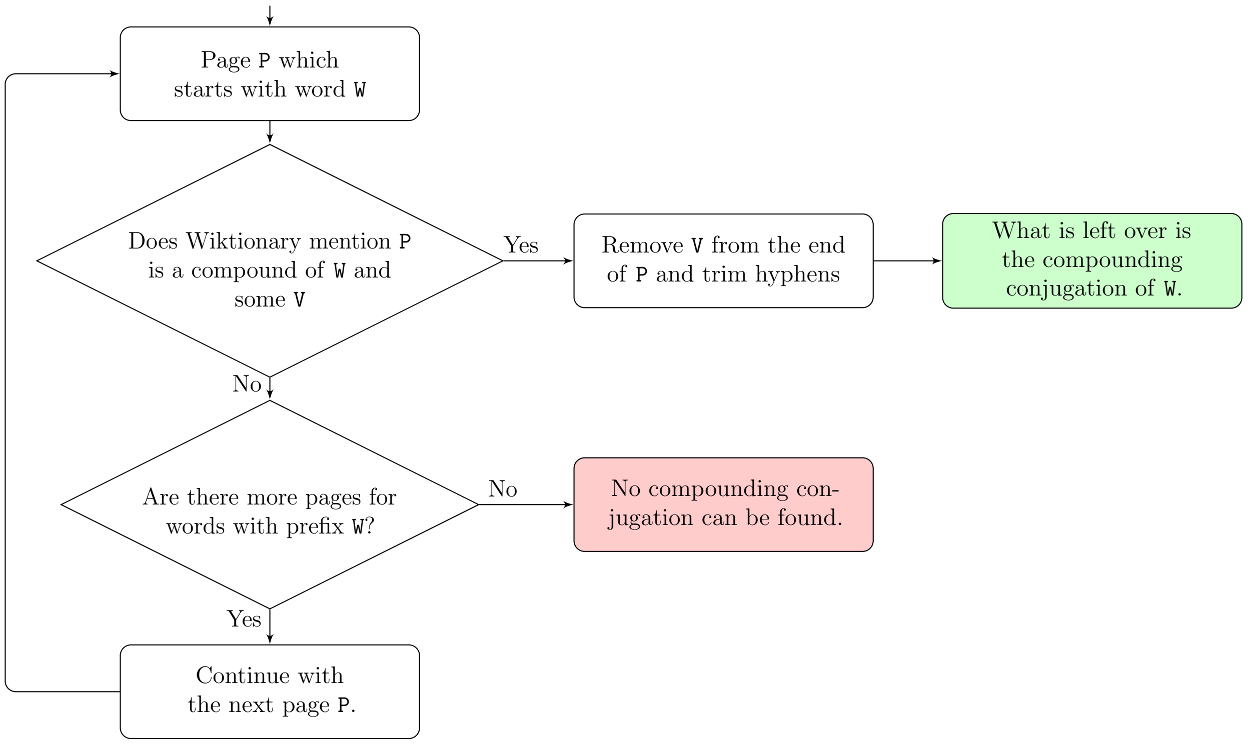

Page P which starts with word W
Does Wiktionary mention P is a compound of W and some V
Yes
Remove V from the end of P and trim hyphens
What is left over is the compounding conjugation of W.
No
Are there more pages for words with prefix W?
No
No compounding conjugation can be found.
Yes
Continue with the next page P.

Figure 6: Flow chart for determining the compounding conjugation of a word `W`, starting with the first Wiktionary page `P` in a list.

3.3 Determining how and when to apply morphological rules

In order to correct split errors, we first need to detect them, using a set of detection rules. Some very specific rules are first handled, to reduce the amount of possible conflicting rules further down the line. This is because different rules may cause words to be compounded in different ways. Making sure more specific rules are applied first will prevent incorrect compounding rules from being applied when more broad rules are checked.

These simple rules include the compounding of small numerical words (e.g. '*drie en twintig*' into '*drieëntwintig*'), larger numerical words (e.g. '*twee duizend drie*' into '*tweeduizend drie*'), and some rules that never get compounded get handled to ensure these types of words don't get falsely compounded (for example an adjective in its *–e* form, like '*witte rijst*').

Sometimes, SpaCy marks a word as being part of a compound (tagging it as "compound:prt")[8]. In these cases, SpaCy already detected the split error, so based on this we compound the marked words.

After these cases are handled, general rules get applied. If two consecutive nouns are tagged as both the subject (e.g. '*ruimte sonde*') or both an (indirect) object (e.g. '*Ik zie de politie agent*'), and their syntactic dependency either is the same, or the former of the two words acts as a modifier (according to SpaCy), they are are compounded. However, due to the fact the variant of the sentences with the split error is grammatically incorrect, SpaCy very commonly does not tag two consecutive 'subjects' in a sentence as both being subjects (commonly matching them as a modifier

[8]https://universaldependencies.org/docs/en/dep/compound-prt.html "The phrasal verb particle relation identifies a phrasal verb, and holds between the verb and its particle. It is a subtype of the compound relation."

word instead).

Because of SpaCy's dependency tagger is not always accurate, our algorithm does perform some compounding on commonly mis-labelled dependencies. Even though this does increase the false positivity rate, not doing so would effectively reduce the amount of corrected split errors to only a handful of specific cases where SpaCy marks a grammatically incorrect sentence accurately.

Next, a few checks are done to see if a compound consists partially of a prefix word or suffix word. If it does, and the POS tags for that prefix or suffix indicate that it is not a standalone word (for example the words in "ex premier" getting tagged as "SPEC/deeleigen" and as a noun, respectively), then the prefix or suffix will be compounded to the word.

Following these checks, several criteria are applied to determine if words need to be compounded, in no particular order, since each of these criteria generally only apply to a single set of word types, so the checked criteria do not overlap with each other. These rules were mainly built based on the output of SpaCy's tagger, based on certain inputs. Either a correctly spelled sentence, or an incorrectly spelled sentence, were fed into SpaCy, and then based on the tags applied to the tokens, the algorithm was tuned to have as many corrections as possible in the right places. See also Appendix B for the exact code that applies these rules.

If two words do not get recognised by any of the criteria for compounding, they are left separated.

3.4 Determining applicable compounding rules

After detecting split errors, we need to follow a set of rules on how to compound the flagged words. If no prefix was found using Wiktionary lookup, described in Subsection 3.2, we need to figure out the compounding conjugation of these words using a set of rules.

3.4.1 Numbers as words

The simplest rules to apply are numbers that are written out as words. They follow very clear rules[9], and can therefore be handled first.

- Numbers below a thousand get written together. Examples are "tweehonderddrieëndertig", "vijfennegentig", "negenhonderdnegenennegentig".
- Multiples of a thousand get written together with the word 'duizend', but separate from the parts smaller than a thousand. An example of this is "driehonderdzestigduizend honderdnegentien".
- Words representing a value of one million or larger get written separately. This means whenever a number above a million is written out, the 'illion' word (italicised in the following examples) will always be a separate word. Examples are: "vijftien *biljoen* driehonderdvier", "zevenhonderddrie *miljard* en veertig"

3.4.2 Unique compound words

Next, a few exceptional word endings are covered individually. These words will get a unique compound when they are the first part of a compound[10]. Other exceptions typically listed are

[9]An overview of these rules can be found on https://onzetaal.nl/taaladvies/getallen-uitschrijven.

[10]See rule 8.E on https://woordenlijst.org/leidraad/8/3 as a reference; rule 8.F are all dictionary words and are therefore ignored in this thesis.

‘*koninginnedag*’ (not ‘koninginnendag’) and some references also list ‘lieveheersbeestje’ (not ‘Lieve Heersbeestje’), but those can be considered dictionary words (or ‘versteende taalvorm’) instead of an exception to the rule. So this algorithm assumes the construct ‘koninginne dag’ would be corrected by dictionary-based spell checkers already.

- Zon (‘*zonne–*’)
- Maan (‘*mane–*’)
- Hel (‘*helle–*’)
- Onze-Lieve-Vrouwe (‘*onzelievevrouwe–*’)

3.4.3 Prefixes and suffixes

Next special case that is handled is prefixes and suffixes. A self-made list of prefixes and suffixes was compiled based on some searching on the internet. Three of these prefixes, namely ‘*ex–*’, ‘*oud–*’ and ‘*anti–*’, are prefixes that always get compounded *with* a hyphen (given that they have the right POS tags, since “Een oud mannetje” should not be compounded). Other than these, the prefixes ‘*vice*’ ‘*pseudo*’, ‘*mega*’, ‘*giga*’, and ‘*super*’ do not always get compounded with a hyphen, being called an ‘afleiding’, so they should be compounded directly to the word[11]. There were also a few suffixes that typically get compounded, given the right combinations of POS tags, namely ‘*vrij(e)*’ and ‘*loos/loze*’.

3.4.4 Simplifications

A few simplifications were then applied to the words. Any diminutive word that is already in its plural form should just be compounded (“*meisjesschoenen*”). Any similar diminutive word should be compounded with –s– put between them, effectively turning them into the diminutive plural.

Another simplification is that if the second part of the compound is classified as a verb (such as “*avondvullend*” or “*koemelkende*”), it should be compounded without anything put between the compound. This prevents the first part of the compound to get a suffix it would otherwise get when following the rules. However, this simplification might not always be appropriate.

3.4.5 –e–, –n– and –en–

Words that get compounded with either an –e–, an –n–, or an –en– in between the compounded words, can be generally be constructed using the properties of the word and its other forms. When words are compounded using ‘*–en–*’ and the first part of the compound already ends in ‘*–e*’, then only an ‘*–n–*’ gets inserted. Otherwise, ‘*–en–*’ is inserted.

The first other form of the word that is checked is the female form of the word. If a word has any feminine form (as listed on Wiktionary) that only differs from the word itself by a suffix of ‘*–e*’ (for example ‘*student*’ and ‘*studente*’), then it should be compounded with ‘*–en–*’ (like ‘*studentenkamer*’)[12].

[11]See https://woordenlijst.org/leidraad/6/2: An ‘afleiding’ gets written directly attached to the other word.
[12]See rule 8.G on https://woordenlijst.org/leidraad/8/3.

The next check is the plural form of the word. If the word does not have a plural form, such as words describing a material like '*water*' or '*goud*', then the compound does not get any letters inserted. The words are combined as-is (for example '*watervlo*' or '*goudsmid*').

If a word does have plural forms, they are checked if the plural forms contain either an *–es* or *–en* form, or both. Examples of those three categories of words are "*dames*", "*honden*", and "*weides/weiden*", respectively.

1. If the word has a plural form ending with *–en*, but no plural with *–es*, (such as '*hondenhok*' or '*boerenvla*') then the word needs to be compounded with its plural form.
2. If the word has a plural form ending in both *–en* and *–es*, then it should simply be compounded without additional compounding characters (for example '*groentebak*', '*gemeenteraad*').
3. In *all* other situations, nothing more can be said about the compounding rules for the word. Each possible situation is given here with an example of different behaviours:
    - No plural of either *–en* or *–es*: '*appelsap*' without additional letters, or '*stationsplein*' with.
    - Only an *–es* plural: '*schedepoets*' without additional letters, or '*damestas*' with.

The additional emphasis on this third point is due to the abundance of websites claiming generalizations of rules that should apply to these situations.[13] [14] Most websites that attempt to explain the rules of required letters between two compounded words, do so by assuming the reader already knows the sound of the compound, and is attempting to derive the spelling from that.[15] This is made additionally obvious when attempting to find any generalizable rules for inserting an *–s–* between a compound, with some sites simply claiming that the rule is "we write –s– when we hear it in its pronunciation".[16]

Due to the lack of formal rules that can be applied to generalizable inputs, the final method for determining how the compound would need to be constructed was chosen to be fully based on Wiktionary whenever that was possible, only falling back to the small amount of grammatical rules that could be relied upon, as detailed above.

4 Experiments

Two text collections were used to test this algorithm, namely the Dutch Wikipedia (http://nl.wikipedia.org/) and a popular Dutch forum site, FOK!forum (https://forum.fok.nl/). We used the former for its high-quality content, which contains relatively few grammatical errors.

[13]https://www.vlaanderen.be/taaladvies/tussenletters-e-en-en-in-samenstellingen – " geen meervoud op -en → zonder -n- (...) het linkerdeel is een zelfstandig naamwoord dat alleen een meervoud op -es heeft." is a rule that is contradicted by '*damestas*'

[14]https://woordenlijst.org/leidraad/8/1 – "Heeft het linkerdeel een meervoud dat eindigt op -en? (...) schrijf -e-" is a rule that only applies to words that already end in '*–e*' in the first place

[15]https://onzetaal.nl/taaladvies/tussen-n/ – "In veel samenstellingen en afleidingen gebruiken we een tussenklank (...) Die klank wordt soms geschreven als een e en soms als en" – The information is based on sounds, not the base parts of a compound

[16]https://vrttaal.net/taaladvies-spelling/samenstellingen-tussenklank-s – "De tussen-s in samenstellingen schrijven we als we die uitspreken."

This makes it a very useful site for testing for false positives. The latter is used for its relatively low-quality content, in a grammatical sense, due to it being a platform where casual writing is by far the most common writing style. Each of these experiments is described in its respective subsection.

4.1 Wikipedia articles

The experiments run on random Wikipedia articles were mainly focused on flagging false positives. After this, we selected a collection of high-quality articles to perform experiments on. Wikipedia has an exposition of high-quality articles located on its *Wikipedia:Etalage* page, where featured articles are listed. These featured articles are all also added to the category *Wikipedia:Etalage-artikelen*. From this category, all articles were processed by the algorithm, and 100 of them were manually checked for their false/true positivity rate. Due to the sheer volume of text that was processed, only the differences between the algorithm's input and output were analysed. It would require a manual review of the full page content to determine the amount of false negatives, so we ignored the parts of the pages that were left untouched by the algorithm in the analysis, which also means we were unable to determine the ratio of false positives to false negatives, in this experiment.

4.2 FOK!forum posts

FOK!forum posts are useful for testing whether the algorithm correctly detects and corrects a split error that occurs in text. Because forum posts are only subject to the author's own sense of spelling and their web browser's spell checker, these posts feature widely varying levels of grammatical correctness.

This also means the algorithm will be given many more errors, which allows for some indication of the ratio of false negative to true positive results. However, due to the sheer amount of data that would need to be processed, it is still unfeasible to tally up all false negatives in all files. Because of that, only a small sample of three thread pages were used for a full analysis of false negatives.

This false negative test was performed on the three forum threads which already featured a large number of true positives. This indicates the threads contain a large number of split errors, which is ideal to test the ratio of true positives to false negatives.

5 Results

In general, the words that were correctly compounded (true positives) were almost always compounded in the correct way, with the proper letters inserted between two words. This includes *–e–*, *–n–*, *–s–*, but also more complicated compound rules such as "*kind fiets*" into "*kinderfiets*". This approach does not work when the first part of a compound does not already have any compounded entries in the dictionary (as described in Subsection 3.2). For example, "*kleinkind fiets*" would get compounded to "*kleinkinderenfiets*", due to having to fall back on the much less reliable approach described in Subsection 3.4.

Table 1: Table of analysed pages. TP is True Positives and #Corrections is the sum of both True and False Positives.

	#Articles	#Words	TP	#Corrections
Total set	244	1,265,822	-	7,361
Total (average)	1	5,187	-	30
Manually verified subset	48	245,840	8	1,543
Manual subset (average)	1	5,121	$^1/_6$	32

Table 2: Table of analysed forum threads. TP is True Positives and #Corrections is the sum of both True and False Positives, and FN is the False Negatives.

	#Threads	#Words	TP	#Corrections	FN
Total set	529	331,387	-	2795	-
Total (average)	1	626	-	5.3	-
Manually verified subset	100	59,440	79	535	-
Manual subset (average)	1	594	0.8	5.3	-
False negative check	3	5,637	26	62	15

5.1 Wikipedia articles

Due to the randomly selected Wikipedia articles only being used in building the algorithm's rules, as a (manual) 'learning' set, the results that are discussed here are only the pages listed in the *etalage* mentioned in Section 4.1. The results of these experiments are shown in Table 1.

From the Wikipedia articles, the overwhelming majority of corrections made by the algorithm are false positives. From a sample of 48 random Wikipedia pages, only 0.5% of corrections made in the manually analysed subset of pages were good corrections. Of course it is to be expected that these articles contain very few true positive spelling mistakes, considering this data set was selected exactly for the reason that this text is of high quality.

In total, the full set of articles in the *etalage*, including the pages that were not manually checked for false positives, closely resembles the manually verified data in both average words per article (around 5,100 words), as well as the total amount of corrections made per article (around 30 corrections). Assuming the rate of true positives is also roughly the same as the manually checked page, this would have resulted in a total of around 40 true corrections that would have been made by the algorithm.

From the manually checked portion of the set, some of the most common false positives are listed in Table 3. Some more tuning of the algorithm would likely result in a significant reduction of such false positive results.

5.2 FOK!forum posts

In the FOK!forum data, the lower quality of the text caused the algorithm to find significantly more true positives. The gathered data is shown in Table 2. Even though there are still a significant amount of false positives here, for these forum threads, about 15% of all corrections were true positives.

A manual check was done for three forum posts which featured the largest number of true positives, as described in Section 4.2, of which the results are shown in the bottom row of Table 2. These threads featured 5, 10 and 11 true positives from the algorithm. After manual checking for false negatives, these had a respective 2, 8 and 5 split errors that were not flagged as such by the algorithm. This means the algorithm had an average detection rate of 26 out of 41 split errors, which is 63%. The individual threads that were checked had a detection rate of 71%, 56% and 69% of split errors, respectively.

6 Discussion

The analyses of the experiments were performed only by the author in order to determine the correctness of a sentence. The only false negatives that were recorded are false negatives spotted by the author, and the validity of positives was only determined by the author, with consultation of some linguistics websites. Even though this was done while attempting to be as accurate as possible in this analysis, this is all subject to some human error. Therefore, these results can only be taken as a rough indication of accuracy of this algorithm, and for more precise results, a more rigorous experiment may be required.

Due to the relatively large amount of false positives, the algorithm cannot be used practically in its current state. However, the algorithm was able to flag several relatively hard to spot split errors throughout the whole set of the analyzed files. This gives the impression that, if the overwhelming majority of false positives can be filtered out, this approach could be practically applicable. Ways to clean up a significant portion of these false positives are discussed in Subsection 6.3. However, a more likely approach to solving the issue of false positives would be to feed the corrected text (including false positives) back into a spell checker that is known to rarely introduce split errors, such as Valkuil or Microsoft's spell checker. Those spell checkers would then be able to re-introduce splits where necessary, while leaving the correct changes. Despite the low likelihood of being able to filter out all false positives though, it would be possible to greatly reduce the amount of false positives by fixing the issues with the errors listed in Table 3. This would result in a significant improvement in the ratio of true versus false positives.

The issue described in the introduction of Section 5, which caused "kleinkind fiets" to be compounded incorrectly, even though "*kind fiets*" was compounded correctly, could be resolved by first determining if the first word of the two is itself a compound. Since it is, the last part of this compounded word (in this case "*kind*") could be used as the word that determines the compounding conjugation. Doing that would result in the conjugation being "*klein**kinder**fiets*", in the example given.

6.1 Wikipedia article results

As expected, the featured articles from Wikipedia were generally of high quality, featuring close to no split errors. The most common true positive that was flagged by the algorithm was the occurrence of *"twee derde"*, but due to the fact both the spelling with a space, and the spelling with a hyphen are correct, these true positives were counted as neither true nor false positives. Other than that, the sampled pages did contain a single detected and corrected error, namely "*Cassini-Huygens project*". The same sentence also featured another, undetected split error, namely "*Huygens ruimtesonde*".

Table 3: A table of some of the most commonly occurring false positives in the data set.

Type of error	**Example**	**Algorithm output**
<SUBJ> die	“De mensen die dit doen”	“De mensendie dit doen”
paar <SUBJ>	“Een paar koeien”	“Een parenkoeien”
vorig jaar	“Die film van vorig jaar”	“Die film van vorigjaar”
<COUNTRY> <SUBJ>	“De Nederlandse tradities”	De Nederlandsetradities
Descriptive nouns	“De stadswijk Zuid-West” “Uit zuidelijk Frankrijk”	“De stadswijkZuid-West” “Uit zuidelijkFrankrijk”
<NUM> en <NUM>	“Tussen 1900 en 2000”	“Tussen 1900en2000”
<ADJ-*en> <SUBJ>	“De zilveren beeldjes zijn weg”	“De zilverbeeldjes zijn weg”

The use of wikipedia articles to find and correct false positives, flagged by the algorithm, can be very beneficial in improving the algorithm. Other than wikipedia articles, news articles from established newspapers and books published by reputable publishers (where high quality of text can be assumed) could also serve as a very good test set for finding the number of false positives. In this thesis, Wikipedia was chosen as the more easily accessible option out of these three categories.

6.2 FOK!forum results

The manually analyzed forum threads generally had true positives that were typically more common Dutch words. In the example with 10 true positives, the correctly fixed compounds were words like “*mega schijf*”, “*usb poorten*”, “*on-demand films*”, “*XBox drive*”, “*hdmi kabel*” and “*HDMI kabel*”, and “*arcade games*”. The false negatives we found manually were words that are either taken directly from English, or are otherwise not typical Dutch words on their own. This makes it harder to detect such split errors, making it more likely for them to be a false negatives. These were words like “*media centertje*”, “*media streamer*”, “*blu ray*”, “*fullHD beamer*”, and “*storage ruimte*”.

6.3 False positives

Several types false positives occurred very frequently in the analysis of the algorithm’s output. Through repeated iteration of tweaking the algorithm, testing it, and tweaking it more, these false positives could be ironed out to result in a much better algorithm. However, the scope of this project did not allow for this kind of extensive feedback process to perfectly optimize the algorithm.

Each of these has a different reason for being detected commonly:

- ‘mensen die’ gets compounded because there were some rules that did not check if the word ‘die’ falls in one of the categories that should be compounded, causing it to be compounded despite being tagged as ‘VNW’.
- Quantifiers such as ‘paar’ get recognised as nouns that act as nominal modifiers, which were marked to be compounded in the same rule as adjectival modifiers such as the first words of compounds ‘maximum gewicht’ and ‘reclame bord’.
- ‘vorig jaar’ gets compounded due to being classified as the exact same combination as a construct like ‘EHBO diploma’ (namely: adjective and noun, with dependencies ‘amod’ and

‘nmod’), which should be compounded. However, with changes to the algorithm that now correct ‘EHBO’ to be recognised as a noun, this rule can be removed.

- ‘Nederlandse tradities’ gets compounded for the exact opposite reason: its base form ‘nederlands’ gets corrected to be a noun instead of an adjective, which causes it to be compounded.
- Descriptive nouns get compounded due to being classified as two consecutive nouns, with the latter word being correctly classified as an oblique nominal. The rule to compound those combinations of words was added due to SpaCy commonly mis-classifying words as oblique nominals.
- The compounding of two numbers larger than 10 is based on the rule added for compounding smaller numbers, when written out. Further restricting this rule to apply to only the combination of a written-out number below 10, ‘en’, and then a number between 20 and 100, would remove these false positives.
- The ‘zilveren beeldjes’ case gets compounded due to SpaCy tagging ‘zilveren’ as a noun in this case.

With improvements to the algorithm these common false positives, as well as many more, can be filtered out.

7 Conclusions and Further Research

We think this approach has potential to be integrated into spell checkers. Even though some of the spell checkers, such as Valkuil and Microsoft’s spell checker, are already able to correctly recognise a large number of correctly compounded words, they are still not good at detecting split errors. Detecting those with a rule-based approach like this, and then verifying the validity of those compounds via those existing spell checkers, would filter out a very large number of false positives, while still being able to flag the true positives as split errors.

For tools like Microsoft’s checker or Valkuil, which were not already introducing some amounts of split errors (as is done by Google’s spell checker, and to a lesser extent, Hunspell), this type of approach would be useful to act as a supplement to these existing checkers. The undetected split errors could be detected through a list of rules such as these, and then the existing systems could verify if a compound like that could exist in that context.

Further research into ways to improve this algorithm may be done using the approach described in Mangu and Brill (1997), which uses the approach to “Apply [a] transformation to a copy of the corpus” and then “Score the resulting corpus according to the objective function”. Using such an approach and then determining which of the two copies of a sentence is grammatically correct in some way might help greatly reduce the amount of false positives found by a rule-based algorithm.

The conclusion to the research question “To what extent can a rule-based algorithm detect and correct Dutch split errors?” we can draw, based on this research, is this. A rule-based algorithm is indeed able to detect a significant amount of split errors in Dutch text. However, the current implementation also brings along with it a large amount of false positives. The approach of using the dictionary’s listed compounds starting with a given word to determine in which way that word should be compounded, as described in Subsection 3.2, proved a good enough strategy to correctly determine the way in which words should be compounded. To avoid the large amount of false positives, either another approach would be needed to resolve this, or this algorithm's output would need to be passed through another spell checker to then filter these false positives again.

References

Beeksma, M., van Gompel, M., Kunneman, F., Onrust, L., Regnerus, B., Vinke, D., Brito, E., Bauckhage, C., and Sifa, R. (2018). Detecting and correcting spelling errors in high-quality dutch wikipedia text. *Computational Linguistics in the Netherlands Journal*, 8:122–137.

Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11(1):87–111.

Mangu, L. and Brill, E. (1997). Automatic rule acquisition for spelling correction. In *ICML*, volume 97, pages 187–194. Citeseer.

Tijhuis, L. (2014). Context-based spelling correction for the dutch language: Applied on spelling errors extracted from the dutch wikipedia revision history.

Verberne, S. (2002). Context-sensitive spell checking based on word trigram probabilities. *Unpublished master's thesis, University of Nijmegen.*

Vosse, T. (1992). Detecting and correcting morpho-syntactic errors in real texts. In *Third Conference on Applied Natural Language Processing*, pages 111–118.

A Git repository

A git repository containing the files used in this thesis can be found at https://github.com/Joeytje50/compound-fixer.

B Function in verbeter.py

This is the main function that determines whether or not two words should be compounded. Code annotations are in Dutch.

```
def isCNOM(n):
    if not n:
        return False
    # is core nominal?
    # obl functioneert vaak hetzelfde als core nominals,
    # dus worden hier ook meegenomen
    return n.split(':')[0] in ['nsubj', 'obj', 'iobj', 'obl']

#prev en cur zijn woord-objecten, nxt is de arr met alle daaropvolgende woorden
def checkCompound(prev, cur, nxt):
    if cur.woord is None:
        # we zijn bij het laatste woord aangekomen.
        return
    # X.dep uitgelegd op https://universaldependencies.org/u/dep/
    if prev.func is None:
        # als in de vorige iteratie nxt[0] is afgehandeld, doe dan niets
        # dit gebeurt bijvoorbeeld bij 'vijf en twintig' -> 'vijfentwintig'.
        return;
    if isPrefix(prev):
        if cur.func == 'N' or cur.func == 'ADJ'
          or prev.functype == cur.functype == 'deeleigen':
            # pseudo-intelligent, ex-coach
            return koppel(prev, cur)
    elif isSuffix(cur):
        if prev.func == 'N':
            return koppel(prev, cur)
    if prev.func == 'TW' and prev.functype == 'hoofd' and prev.dep != 'punct':
        # niet "de eerste drie" en "In het jaar '95" matchen.
        # vorige woord is telwoord
        if cur.func == 'TW':
            # honderddrie, duizendtwee, honderdduizend, 300 000
            if not grootGetal(prev) and not grootGetal(cur)
                and not prev.woord.endswith('duizend'):
                 # behoud nodige spaties bij grote getallen:
                 # vijf_miljoen_en_drie; vijf_miljoen_vierendertig
                 return koppel(prev, cur)
        elif cur.woord == 'en' and nxt[0].func == 'TW' and not grootGetal(prev):
            # drientwintig, eenenveertig
            return koppel(prev, cur, nxt[0])
    if prev.func == 'N':
        if cur.dep == 'compound:prt':
            if prev.dep == 'appos':
                # Spacy vindt dat dit een compound-part is
                return koppelstreep(prev, cur)
        if isSuffix(cur):
            return koppelstreep(prev, cur)
        if (cur.dep == 'ROOT' and cur.func == 'N') or (isCNOM(cur.dep)
           and cur.func in ['N', 'ADJ']):
            if prev.dep in ['amod', 'nmod']:
                # bijv maximumgewicht, oud-coach, reclamebord.
                # Als prev ADJ is niet koppelen (bijvoorbeeld 'een rood huis')
                # soms wordt een znw als ww herkend, zoals
                # 'dit is een lading honden poep'.
                # De syntactische analyse is dan nauwkeuriger.
                koppel(prev, cur)
        if cur.func == 'N':
            if prev.dep == cur.dep and isCNOM(cur.dep):
                # twee ZNWs achter elkaar met dezelfde functie als ow/lv/mwvw
                return koppel(prev, cur)
        if prev.dep == 'obl' and (cur.dep == 'amod' or isCNOM(cur.dep)):
            # avondvullend programma
            koppel(prev, cur)
    if prev.func == 'ADJ':
        if ('met-e' in prev.tag or 'sup' in prev.tag) and not isPrefix(prev):
            # bvnw vervoegd met -e(r) of -st(e) worden niet samengesteld; vb:
            # De groter wordende man, het grootste gewicht, de roodst gloeiende.
            # bvnw in standaardvorm moeten wel samengevoegd; voorbeelden:
            # De grootwordende man, het grootgewicht, de roodgloeiende.
            return
        if cur.func == 'N':
            if prev.dep in ['amod', 'nmod'] and prev.functype == 'prenom':
                if (isCNOM(cur.dep) or cur.dep in ['amod', 'nmod'])
                   and nxt[0].func == 'N'
                   and cur.functype == nxt[0].functype == 'soort' and (
                        nxt[0].dep in ['ROOT', 'xcomp', 'compound:prt']
                        or isCNOM(nxt[0].dep)
                   ):
                    # woorden zoals laagsteprijsgarantie, langeafstandsloper
                    return koppel(prev, cur, nxt[0])
            if prev.dep == cur.dep and isCNOM(cur.dep):
                return koppel(prev, cur)
            if prev.functype == 'prenom':
                if prev.dep in ['nmod', 'amod']:
                    if isCNOM(cur.dep) and isPrefix(prev):
                        # 'De _oud voetballer_ is blij.', maar moet prefix zijn,
                    # anders 'We hebben een avond vullendprogramma gemaakt'.
                    return koppel(prev, cur)
                if cur.dep in ['nmod', 'amod']:
                    # EHBO diploma in voorbeeldzinnen
                    return koppel(prev, cur)
if prev.func == 'BW':
    if cur.func == 'WW':
        if prev.dep == 'compound:prt':
            # weg gesleurd, door gehaald, bij gevoegd, etc.
            return plakvast(prev, cur)
if prev.func == cur.func and prev.func in ['N', 'ADJ']:
    if prev.dep in ['nmod', 'amod'] and cur.dep in ['nmod', 'amod']:
        # twee ZNWs of BVNWs achter elkaar met een 'modifyer'-functie op het
        # volgende woord.
        return koppel(prev, cur)
    # prev is in de standaardvorm zonder -e of -st wegens eerdere filtering
    if prev.func == 'ADJ':
        if isCNOM(prev.dep) and cur.dep == 'acl':
            # roodgloeiende
            return koppel(prev, cur)
        elif prev.dep == 'advmod' and cur.dep == 'acl':
            # de rood gloeiende draad
            return koppel(prev, cur)
        else:
            # anders niet koppelen.
            return
    if prev.dep == cur.dep:
        if isCNOM(cur.dep):
            # zelfde functie in de zin en vervult de rol van ow/lv/mwvw
            return koppel(prev, cur)
        elif cur.dep == 'nmod':
            # 'Ik heb een student kamer gekocht.
            return koppel(prev, cur)
if isCijfer(cur):
    if prev.woord == 'jaren' and nxt[0].func == 'N'
       and (nxt[0].dep == 'appos' or isCNOM(nxt[0].dep)):
        # jaren-80-muziek (witte boekje);
        # groene boekje 'jaren 80-muziek' niet ondersteund.
        koppelstreep(prev, cur)
        koppelstreep(cur, nxt[0])
        return
else:
    # woorden niet samenvoegen
    return
```