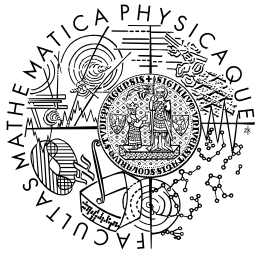

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jonathan Oberländer

Splitting Word Compounds

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Pavel Pecina, Ph.D.
Study programme:	Computer Science
Study branch:	Mathematical Linguistics

Prague 2016

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Splitting Word Compounds

Author: Jonathan Oberländer

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Pavel Pecina, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Unlike the English language, languages such as German, Dutch, the Skandinavian languages or Greek form compounds not as multi-word expressions, but by combining the parts of the compound into a new word without any orthographical separation. This poses problems for a variety of tasks, such as Statistical Machine Translation or Information Retrieval. Most previous work on the subject of splitting compounds into their parts, or "decompounding" has focused on German. In this work, we create a new, simple, unsupervised system for automatic decompounding for three representative compounding languages: German, Swedish, and Hungarian. A multi-lingual evaluation corpus in the medical domain is created from the EMEA corpus, and annotated with regards to compounding. Finally, several variants of our system are evaluated and compared to previous work.

Keywords: decompounding morphology NLP

I want to thank Gergely Morvay and Gero Schmidt-Oberländer for the annotation of the Hungarian and Swedish corpora.

Contents

Introduction

In the English language, when we want to use a specific meaning of a word, we can use prepositions such as "of" or "for": a club *for* sports, a bottle *of* water. Here, the first noun remains the *head* of the entire constituent, and the second one is the *modifier* used for restricting the kind of the first noun (along with other functions, see Chapter *Compounds*). In other words, a bottle of water is still a bottle (but not a water).

Alternatively, the words can be combined without prepositions by merely using them sequentially. In this case, the modifier is the first noun, and the second one is the head: a *sports club*, a *water bottle*.

In some languages (called *compounding languages* here) this process of forming *compounds* results in words that (orthographically) look like proper words themselves. There are a few examples of these kinds of words in English, too, but they are lexicalized and largely come from the German influences of English: A *bookshelf* is a shelf for books, but a *bottleshelf* isn't a shelf for bottles, it's a non-word; the correct form would at best be *bottle shelf*.

In compounding languages sets of nouns can freely be combined into other nouns. In German, a water bottle is called *Wasserflasche* (water + bottle), a shelf for bottles could be called *Flaschenregal* (bottle + shelf). This process can be repeated recursively, such that a shelf for water bottles could be called a *Wasserflaschenregal*, and so on, leading to famous extreme examples such as *Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz* (roughly *beef labeling supervision task delegation law*, a compound with 7 parts). This results in longer word lengths in compounding languages (see Figure 1).

For processing of natural language, this can cause problems: The more complex a compound is, the lower the likelihood of it appearing in corpora, and while some common compounds can be found in dictionaries, a large part of them won't. We can't dismiss these words as made-up words or names and ignore the problem, because any typical speaker of such a language will effortlessly understand their meaning. They also make up a significant part of language: Schiller [2005] find that in a newspaper corpus, 5.5% of all tokens and 43% of all types were compounds.

Consider, for example, the task of Machine Translation from German to English: Compounds that are encountered need to be translated, and without having ever seen them before, this becomes difficult (see for example Koehn and Knight [2002]). In addition, this is problematic for word alignment, because one word in the source language has to be aligned to two or more words in the target language, or vice versa. This difficulty is illustrated in Figure 2.

In Information Retrieval, compounds can also cause problems: A user searching for a rare compound won't get many results, but if the compound can be analyzed and understood, more relevant results (not containing the original compound, but its head and modifier) can be retrieved. This is especially relevant in cross-lingual Information Retrieval.

The solution of these problems of course lies in reversing the process: *decompounding*, splitting a compound into its parts. A system that hasn't seen *Wasserflasche*, but has seen both *Wasser* and *Flasche*, can, once it knows that

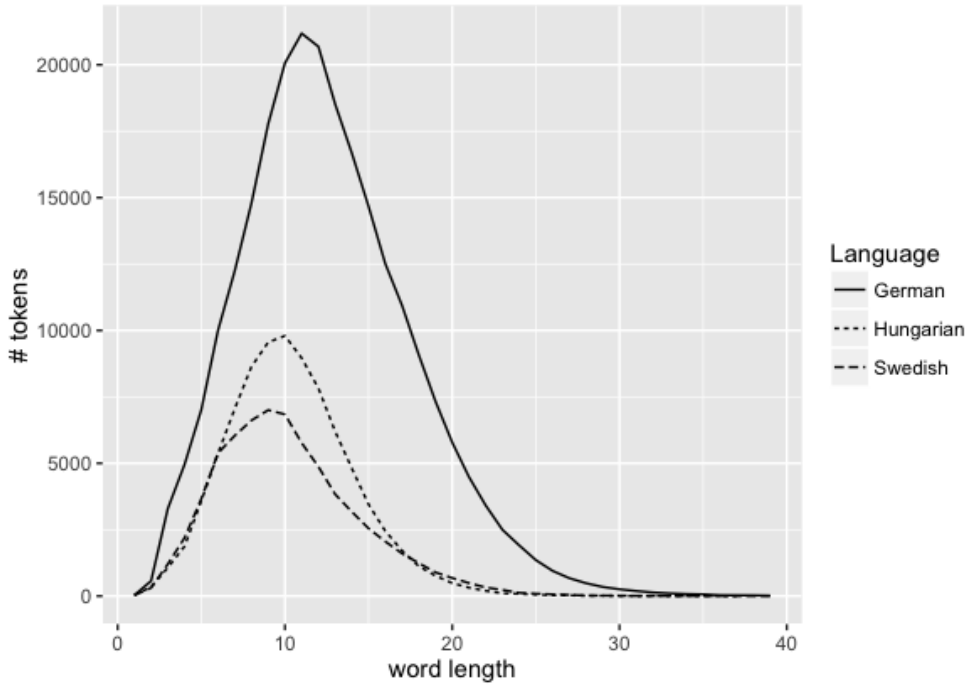

Figure 1: Word lengths of tokens in our evaluation corpus (see Chapter *Corpora*. Compared to English, where the most common word length is 7 [Adda-Decker and Adda, 2000], compounding languages have higher modes of 9 (Swedish), 10 (Hungarian) or even 11 (German).

Die$_1$ **Wasserflasche**$_2$ steht$_3$ auf$_4$ dem$_5$ *Holztisch*$_6$.

The$_1$ **Water**$_2$ **bottle**$_3$ is$_4$ on$_5$ the$_6$ *wooden*$_7$ *table*$_8$.

Figure 2: Word alignment is tricky between compounding and non-compounding languages: **Wasserflasche**$_2$ has to be aligned with **Water**$_2$ **bottle**$_3$, and *Holztisch*$_6$ with *wooden*$_7$ *table*$_8$.

Wasserflasche indeed consists of those two parts infer that the word must mean "water bottle".

Decompounding isn't always as easy as in the previous example. Some compounding languages insert letters between the parts when phonologically required, so called *linking morphemes*: The German word for eye drops is *Augentropfen*, consisting of *Auge* (eye), *tropfen* (drop), and an *n* in the middle.

The process also isn't necessarily unambiguous, for (at least) two reasons: morphosyntactic ambiguity and morphological ambiguity.

Compounds that are truely morphologically ambiguous aren't that common, but they exist, for example in the German word *Fluchtraum* (shelter; safe room): It can either (correctly) be split into *Flucht* (escape) and *Raum* (room), or (incorrectly) into *Fluch* (curse) and *Traum* (dream). There's no systematic reason the former is correct, in fact, in the right context, one could even use it with the second way of splitting it meaning something akin to nightmare. The only reason *shelter* is the more "correct" solution is that it is a somewhat commonly used word.

Morphosyntactic ambiguity is more common, but also less of a problem. In any compound of three or more parts, there is a question of hierarchy, with this problem also arising in languages that don't freely compound: Is a *house door key* a *key* for the *house door*, or is it a *door key* of the *house*? The question is as difficult to answer as it is irrelevant for most purposes.

Most previous work has focused on German, since it's the compounding language with the biggest population by far. In this work, we take a look at more compounding languages, and build a system that is largely independent of any specific language. In the process, we also contribute a multi-lingual evaluation corpus for this task.

There are a few goals for our system:

- **Language agnosticism.** While a few languages are built-in, it should be possible to extend it to new languages in minutes.
- **Low resource friendliness.** Not all languages have as much annotated data and NLP tools as German, so the system should work with as little annotated data as possible. Usage of raw text corpora is fine, since they exist wherever such a system may be needed.
- **Usability.** The system should be easy to set up, not require large dependencies, and be quick to run.

The rest of this thesis is organized as follows: In Chapter *Compounds*, we look deeper into the process of compounding itself, followed by investigating previous work in the area in Chapter *Related Work*. We continue by describing the creation of our corpora in Chapter *Corpora*. In Chapter *Methods* we describe our system and the experiments made on top of it. Its performance is evaluated compared to previous work in Chapter *Evaluation*, and in Chapter *Conclusion* we conclude and discuss possible improvements and related challenges to be addressed in future work.

Compounds

Compounding is an extremely frequent process occurring in many languages. It is the process of combining two or more content words to form a novel one. While in many languages (such as English), these are typically expressed as multi-word expressions, in a few languages, such as German, Dutch, Hungarian, Greek, and the Scandinavian languages the resulting words, or *compounds*, are written as a single word without any special characters or whitespace in-between. In this work, these languages are called *compounding languages*, even though compounding in a broader sense also occurs in languages that write compounds as separate words, such as English.

The most frequent use of compounding by far [Baroni et al., 2002] is compounds consisting of two nouns, but adjectives and verbs form compounds as well. Compounds are also formed from any combination of nouns, verbs, and adjectives.

Languages such as English also have a few examples of compounds that are written as a single word, but these cases are lexicalized, i.e. they have become words of their own. New compounds of this kind cannot be arbitrarily created, whereas in compounding languages they can. In compounding languages, there exists some kind of fluid lexicality, as frequently used compounds are fully lexicalized, newly created ones are not lexicalized, while most compounds are somewhere in-between; not lexicalized in the sense that you would find them in a dictionary, but frequent enough, that many speakers will have heard or even used them.

Looking at the German noun compounds as an example, compounding performs different semantic functions on the compound parts, but in all cases the first part is the modifier and the second part is the head: A *Taschenmesser* (pocket knife), consisting of *Tasche* (pocket) and Messer (knife) is a certain kind of knife, not a certain kind of pocket (see Figure 3).

In some cases, it is no longer true that the resulting compound is still some kind of its head: A *Geldbeutel* (wallet) consists of *Geld* (money) and *Beutel* (bag), and while one probably wouldn't call a wallet a bag, it is still clear that the bag is the head of the compound and the money the modifier. In many of these cases this has historical reasons: At some point in time, a compound got lexicalized, then it later shifted in meaning. According to Lieber and Stekauer [2009], *endocentric compounds* are compounds where the compound is an instance of the head (e.g. a bookshelf is also a shelf), while *exocentric compounds* are those where they aren't (a skinhead isn't a head).

The compounding process can be repeated in a recursive way: A *Taschenmesserklinge* (blade of a pocket knife) consists of the compound *Taschenmesser* (pocket knife) and *Klinge* (blade). This way of combining existing compounds could conceivably be repeated, making compounding a recursive process.

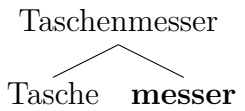

Figure 3: Decompounding *Taschenmesser*. The head is marked in bold.

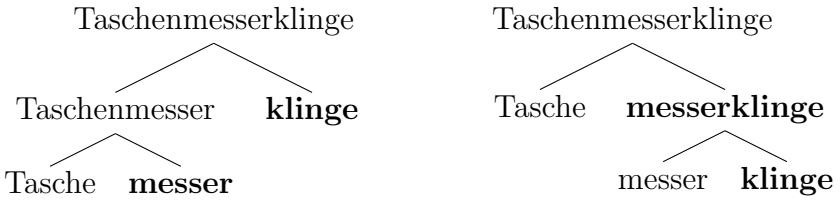

Figure 4: Morphosyntactic ambiguity: The correct (left) and incorrect (right) way of splitting *Taschenmesserklinge*.

As soon as three parts are combined to a compound as above, morphosyntactic ambiguity arises: A *Taschenmesserklinge* could also be constructed from *Tasche* (pocket) and *Messerklinge* (knife blade). For an illustration of this effect, see Figure 4.

In most cases these ambiguities are either not perceivable by humans, or one reading is strongly preferred, other readings not even considered. This is especially true when a compound part is a lexicalized compound itself, like with *Taschenmesser*.

Morphological ambiguity is more subtle, and occurs when a word has several possibilities to be split into compound parts. The already mentioned *Fluchtraum* is one example, and there are even cases of non-compounds that can be analysed as compounds: *Verbrennen* is a verb meaning to burn something, which can (but shouldn't) be analysed as a compound of *Verb* and *Rennen*, or "verb running".

The compounding process is not always a simple concatenation of the compound parts: In *Taschenmesser*, the *n* is neither part of *Tasche* nor of *messer*, it is a so called *linking morpheme* (German: *Fugenmorphem* or *Fugenelement*)[1]. Sometimes, linking morphemes can be interpreted as endings of an inflected form of the modifying compound part (in this case *Taschen* is the plural of *Tasche*), but oftentimes they can't.

Any given language can have several linking morphemes, and some of them can even be "negative": A *Schulbuch* (school book) consists of *Schule* (school) and *Buch* (book), but the final *e* of *Schule* is removed when using it as the first part of a compound.

The choice of a linking morpheme depends entirely on the modifier, never on the head [see Baroni et al., 2002].

During composition, more changes can happen to the modifier than just an attached linking morpheme: In some cases, umlauting is happening to a vowel inside the modifier: Combining *Blatt* and *Wald* creates the compound *Blätterwald*. Due to the unpredictability of this process, especially (for the author) in languages other than German, we simply ignore this, as it doesn't occur very often: In German, less than 0.3% of all compound types experience umlauting, according to Langer [1998].

[1]Langer [1998] criticizes the term, prefering the term "compositional suffix", since it doesn't have much to do with the head of the compound, and must really be seen as just another suffix, one that is used for composition.

Related Work

Wanting to split German compounds, Koehn and Knight [2002] learn splitting rules from both monolingual as well as parallel corpora. They generate all possible splits of a given word, and take the one that maximizes the geometric mean of the word frequencies of its parts, although they find that this process often leads to both oversplitting words into more common parts (e.g. *Freitag* (friday) into *frei* (free) and *tag* (day)), as well as not splitting some words that should be split, because the compound is more frequent than the geometric mean of the frequencies of its parts.

Since their objective is to improve machine translation, they then make use of a parallel corpus: Given a split S of a compound c, if translations of the parts of S occur in the translation of a sentence containing c, the word should probably be broken up: *Freitag* will probably have never beed translated as "free day", so it would not be split. This, of course, can only work for those compounds that actually occur in the parallel corpus, not for completely unseen ones. Furthermore, they constrict parts using a part-of-speech tagger: Words that aren't usually parts of compounds (like determiners), are excluded with this method. Their combined method using frequency information, parallel corpus data and POS tag restriction gives them the highest result, with a recall of about 90.1% and a Precision of about 93.8%[2].

Schiller [2005] use a weighted finite-state transducer to split German compounds based on the output of a morphological analyser, which returns some, but not all possible splits. The necessary weights for the finite-state transducer are obtained from the frequencies of compound parts in manually decompounded word lists. This work is also the source for the claim that the split with the lowest number of (unsplittable) parts is the most likely correct one.

Marek [2006] also use weighted finite-state transducers, but the main contribution of this work for us is the creation of an evaluation corpus and an annotation scheme that was created for annotating it. Our own evaluation corpus is annotated based on this annotation scheme.

Alfonseca et al. [2008] approach the task from an Information Retrieval perspective and wanting to handle noisy data in user input, such as misspellings. They also propose that a split is more likely to be the correct split if its compound parts have a positive *Mutual Information*: If they can be composed, that means there must be some semantic relation between them. For instance, in the compound *Blumenstrauß* (flower bouquet), consisting of the parts *Blume* (flower) and *Strauß* (bunch, bouquet, but also: ostrich), the two parts will co-occur with each other more often than would be expected of two random words of their frequency.

Soricut and Och [2015] use vector representations of words to uncover morphological processes in an unsupervised manner. Their method is language-agnostic, and can be applied to rare words or out-of-vocabulary tokens (OOVs).

Morphological transformations (e.g. *rained* = *rain* + *ed*) are learned from the word vectors themselves, by observing that, for instance, $\overrightarrow{rain}$ is to $\overrightarrow{rained}$

[2]Precision and recall are defined slightly differently than usual, see Chapter *Evaluation* for details.

as $\overrightarrow{walk}$ is to $\overrightarrow{walked}$. To be able to do that, candidate pairs are first extracted by looking at whether it's possible to replace, add, or remove a suffix or prefix of one word to get to the other word. Given that candidate list, they then use word embeddings to judge the quality of candidate rules by trying to find several analogy pairs with the same suffix/prefix in similar relations in the vector space. This filters out spurious suffixes like *-ly* in *only*, or prefixes like *s-* in *scream*.

Their system uses a considerable amount of data to train the word embeddings (they use the skip-gram model of Mikolov et al. [2013b]), something going against our goal of staying low-resource.

Daiber et al. [2015] apply the approach of Soricut and Och [2015] to decompounding. Using word embeddings of compound and head, they learn prototypical vectors representing the modifier. During the splitting process, the ranking occurs based on comparing analogies of the proposed heads with the entire compound.

They achieve good results on a gold standard, and improved translation quality on a standard Phrase-Based Machine Translation setup.

Bretschneider and Zillner [2015] develop a splitting approach relying on a semantic ontology of the medical domain. They first try to disambiguate candidate splits using semantic relations from the ontology (e.g. *Beckenbodenmuskel* (pelvic floor muscle) is split into *Beckenboden* and *muskel* using the *part-of* relation). If that fails, they back off to using a frequency-based strategy.

Erbs et al. [2015] compare the performance of recent work in decompounding using the evaluation corpus from Marek [2006], looking at the correctness scores of the compared systems. They find that the **ASV toolbox** [Biemann et al., 2008] delivers the best results. They then also investigate what benefit decompounding has on the results of a specific task, *keyphrase extraction*.

Corpora

This work made use of several corpora, and produced a new evaluation corpus for evaluation of decompounding systems in German, Swedish, and Hungarian.

EMEA corpus

The EMEA corpus [Tiedemann, 2009] is a parallel corpus based on documents by the European Medicines Agency[3]. It consists of parallel texts for each possible pairing of the numerous languages it contains, including English and all the languages we're investigating (German, Swedish, and Hungarian). For English and Swedish, the texts are also available as parsed corpus files, but the parsed files weren't used by us.

The corpus was chosen as the basis of our evaluation corpus because of our decision to evaluate in the medical domain.

We also use it as the raw text corpus for our word embeddings, since its size is approximately the same for all three languages, meaning we can later exclude different training data size as a cause for differences in the results.

Wikipedia as a raw source

Our system requires some kind of dictionary to serve as a repository of known words. A good source for arbitrary raw text is Wikipedia, because it is relatively big — big enough to contain all kinds of words, in many kinds of inflections. As an encyclopedia, the language use is not necessarily representative, but it was still chosen because in contrast to some other more balanced corpora, it is available for most languages in common use, such as the ones we're investigating here.

Given the restricted domain of our system, and the fact that preliminary experiments indicated that it would be useful to restrict the size of this corpus, we used a subset of Wikipedia concerned with medical articles.

The corpus is a pre-existing one, assembled by Jindřich Libovický[4]. It was created by first extracting titles of Wikipedia articles in the medical domain from the semantic knowledge graph DBpedia [Lehmann et al., 2015], and then extracting the text of those articles from a full Wikipedia dump.

Corpus statistics about the Wikipedia data can be seen in Table 1. The German part is obviously the biggest one (the German Wikipedia is the second-biggest Wikipedia, after the original English one), but what is interesting is that while Swedish has more tokens than Hungarian, Hungarian has more types. This is probably due to the more complex morphology of Hungarian, compared to Swedish.

[3] `http://www.ema.europa.eu/ema/`

[4] `https://ufal.mff.cuni.cz/jindrich-libovicky`

	Types	Tokens
German	245256	2326519
Swedish	67422	426193
Hungarian	86002	354120

Table 1: Corpus statistics for the medical Wikipedia corpus

Marek [2006]

The work of Marek [2006] includes an evaluation corpus that was created based on articles from the German computer magazine *c't — Magazin für Computertechnik*. The corpus itself isn't used, but our own evaluation corpus annotation format is based on it. We drop the annotation of part-of-speech of the compound parts, and concentrate only on where a compound should be split. The annotation scheme is described in more detail below.

Stopwords and affixes

In order for our simple heuristics to filter out obviously wrong splits, we also employ the use of two additional corpora per language: A list of stopwords and a list of suffixes.

The stop word lists[5] contain 232 German words, 199 Hungarian words, and 114 Swedish words.

Because our candidate extraction algorithm goes from left to right (see Chapter *Methods*), it might happen that a word or subword is split into root and suffix, if the root is a known word. Since this is just inflection, not compounding, we want to filter out such cases. For this purpose, we extract a list of suffixes for each language from Wiktionary (a sister project of Wikipedia):

We simply take all page titles in the `Category:`*`language`*`_prefixes` and `Category:`*`language`*`_suffixes`[6] and remove the dash at the beginning of each page title. The resulting suffix list contains 115 suffixes in German, 85 suffixes in Swedish, and 545 suffixes in Hungarian. The prefix lists contain 116 prefixes in German, 48 prefixes in Swedish, and 100 prefixes in Hungarian.

Evaluation Corpus

The creation of the evaluation corpus started with the EMEA corpus, which was first shuffled to ensure that the selection of words wasn't biased. Care was taken however, to shuffle the corpus in exactly the same way for each language, so that randomness didn't introduce a difference between the languages.

Next, the words were filtered based on their length. Since compounds consist of at least two parts, they have a certain minimum length. Any word shorter than seven characters was therefore discarded.

[5] `https://github.com/Alir3z4/stop-words`

[6] For example `https://en.wiktionary.org/wiki/Category:German_suffixes`

	German	Swedish	Hungarian
Words	958	565	3361
# Compounds	193	162	512
% Compounds	20.0%	28.7%	15.2%
% 2 parts	91%	93%	93%
% 3 parts	9%	7%	6%
% >3 parts	<1%	0%	1%

Table 2: Corpus statistics of the evaluation corpora

This list of words was then given to annotators, with the German part being annotated by the author himself. The annotators were given the task to split the words into compound parts using a plus character, when necessary. Non-compounds were to be left untouched.

Word endings of compounds were put in parentheses, linking morphemes split off with a pipe character (|). As in the original annotation scheme in Marek [2006], ablaut and umlaut changes are annotated using capital letters.

When annotating anything there is a choice to be made between accuracy of the annotation process, and the annotation feeling "natural" for native speakers.

A special case of this is how to handle fully lexicalized compounds: They aren't a problem for NLP, since they are frequent enough to be lexicalized. In fact, splitting them could even worsen results in some cases, where the current meaning of a compound isn't directly related to the meaning of its parts anymore:

If a Machine Translation system wants to translate *Geldbeutel*, but translates the decompounded parts instead, the translation might be "money bag" instead of the correct "wallet".

In our case, a decision was made to trust the annotators and let them annotate as compounds what feels like a compound to them.

The German and Swedish parts of the corpus were each annotated by a single person, whereas 10% of the Hungarian corpus was additionally annotated by a second annotator to calculate inter-annotator agreement. Cohen's Kappa was determined to be at $\kappa = 0.95$.

The other part of data required for each language was the list of linking morphemes. The German list is based on Alfonseca et al. [2008], Hungarian and Swedish linking morphemes were extracted from the annotated corpora. They are displayed in Table 3.

Corpus statistics about the evaluation corpora can be seen in Table 2. Most compounds in the evaluation corpora have only two parts, there are very few examples of compounds of three or more parts.

The evaluation corpus in all three languages is included as Attachment 2 (`eval-corpus.tgz`).

Language	Linking morpheme	Example
German	s	Hemd**s**ärmel
	e	Hund**e**hütte
	en	Strahl**en**therapie
	nen	Lehrerin**nen**ausbildung
	ens	Herz**ens**wunsch
	es	Haar**es**breite
	ns	Wille**ns**bildung
	er	Schild**er**wald
Swedish	s	utgångs**s**datum
Hungarian	ó	old**ó**szer
	ő	gyűjt**ő**doboz
	ba	forgalom**ba**hozatali
	ítő	édes**ítő**szerként
	es	kék**es**barna
	s	szürké**s**barna
	i	ízület**i**fájdalom
	a	kor**a**szülött

Table 3: Linking morphemes in German, Swedish, and Hungarian

Methods

This chapter describes the decompounding sytem created for this work. It consists of a script written in Python 3, and is not backwards compatible with Python 2, because it makes heavy use of features missing in Python 2 (and not importable from the `__future__` module).

The script reads a file containing one word per line, and prints a decompounded (if necessary) version of every word. Alternatively, the `Splitter` class can be imported from another module.

The only external requirements are `docopt`[7] for argument parsing, and `ftfy`[8] for repairing broken encoding input.

Basic Principle

When calling the script from the command line or creating a `Splitter` object manually, an ISO 639-1 language code[9] can be specified. If it isn't given, "de" (German) is assumed.

Initialization

Based on the language, the list of linking morphemes is set. This list is adapted from Alfonseca et al. [2008] for German, and based on input from the annotators for the other supported languages.

In addition to the linking morphemes, a dictionary of known words is read in. As a source for the dictionary, a Wikipedia dump of articles in the medical domain is used. The reason for opting for a raw text corpus instead of a real lexicon is that we will find not only base forms, but also most inflected forms in it, and won't have to worry about morphology too much. Since it is our goal to produce a system that is adaptable to any language where it is needed, we want to use as little training data as possible, in fact, the only piece of knowledge we don't obtain from a raw corpus is the list of linking morphemes. This obviously doesn't include the evaluation corpus, which isn't needed for the system itself.

Generation of candidate splits

After this initialization, words can be decompounded by the system. First, a sequence of possible binary splits of a given word is obtained. This includes the "split" where the point of splitting is after the word, meaning nothing is split at all. Starting with the smallest possible split from left to right, we check whether *either*

- the left part of the split is a known word
- the left part of the split is a linking morpheme

[7] `https://github.com/docopt/docopt`

[8] `https://github.com/LuminosoInsight/python-ftfy/`

[9] `http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=22109`

If neither of those conditions are met, the process continues. Otherwise, this process is repeated recursively for the remaining part.

This method has several implications:

- **The right-most part of the proposed split need not be a known word.** This helps alleviate the need for true morphological analysis as endings of the head don't matter so much.
- **The process succeeds for any given word.** It may however return a split which contains only a single word, i.e. isn't a split at all.
- **The process often returns several possible ways of splitting.** Afterwards, a decision needs to be made which of the splits to keep.

Cleaning the splits

As a next step, initial cleaning of the candidate splits is being done, having to do with linking morphemes:

- Linking morphemes can't be at the beginning of a word. If the first part of a split is a linking morpheme, it is merged with the second part.
- Given two consecutive parts, if the right one isn't a known word, the left one is a linking morpheme, and the concatenation of the two parts is a known word, they are merged together. This doesn't need to be checked the other way around, because of the left part *always* being either a known word or a linking morpheme, based on the previous steps.

The list of possible splits is reduced using this process, but there is often still a choice to be made.

Ranking the splits

Next, we find splits where each part is either a known word, or a linking morpheme. If at least one such split exists, we consider only those splits, otherwise we skip this step.

Finally, we take that remaining split with the least amount of parts, following Rackow et al. [1992]. The result is returned either as a tuple, or if requested, in the evaluation file format described in Chapter *Corpora*.

Several variations of the described systems are also available through configuration changes. They are evaluated separately in Chapter *Evaluation*, and described below.

Stop words and affixes

Since compound parts are typically content words, we want to limit the available words the system sees as valid. Since we're building on raw data, part-of-speech tags are not available. We therefore use stop word lists: After the lexicon is read, any words present in the list of stop words is removed from it again.

This prevents splits into pseudo-compounds which supposedly begin with a closed-class word like a determiner. For example, *Dermatologe* (dermatologist) isn't a compound, but without excluding stop words from the list of known words, it might be split into *Der* (nominative masculine singular "the") and *matologe* (spurious).

We also do affix-based filtering here: In the step of candidate extraction, we discard any split that has:

- a prefix as a part anywhere in the split
- a last part that starts with any suffix and is not much longer than it[10]

From manual inspection of the affix lists, we estimate that this has the potential to remove legitimate splits: For example, the German suffix list contains *-zentrisch* ("centrical"), which could conceivably occur as the head of a compound, but it is still our assumption that this filtering will improve more than it will hurt recall.

Frequencies

Frequency information has been shown to be of use in previous work [Koehn and Knight, 2002, Baroni et al., 2002]. Since we're working with a raw text corpus as a lexicon, it is available to us, too.

We use the frequencies for two purposes:

- **filtering out noise:** Any word that only occurs once in the entire corpus is assumed to be either noise or not of use in any case. We therefore discard words that only occur once.
- **frequency-based ranking:** As a primary ranking method, we simply take the split with the highest product of part frequencies. In case of a tie, we take the split with the smallest amount of parts. If there is still a tie, we go on to taking the highest sum of logarithms[11] of the frequencies.

Word embeddings

A relatively recent trend in Natural Language Processing is the use of semantic vectors that represent the meaning of words or other textual entities. Such vectors have been shown to be successful in tasks such as parsing [Socher et al., 2013] or sentiment analysis [Maas et al., 2011].

A fast, memory-efficient way to obtain a semantic vector space is a family of methods called **word2vec** [Mikolov et al., 2013a], of which we use the continuous skip-gram variety: Word and context vectors are initialized randomly,

[10] Because not all inflectional variants of all suffixes are present in the list, we are are looking at whether the last part *starts* with any suffix. However, since we assume just blindly doing this would introduce many false negatives, we only allow the last part to be up to two characters longer than the suffix.

[11] With $\log 0 := 0$. This makes it so that even if a part is zero, the result will still be defined.

then, iterating over all words in a given corpus, the model tries to predict context words from a given word. In the process, word vectors are brought closer to the context vectors of their context words, and, using *negative sampling*, distanced from random "noise" contexts.

Distance between two vectors is measured in terms of *cosine similarity*; the cosine of the angle between the vectors. Using cosine similarity is equivalent to using a euclidean distance between unit vectors, and results in values between 0 and 1, inclusive.

As word embeddings require only raw data, we also investigate its use in our scenario. The idea is to look at the (proposed) parts of a split, and see if adjacent parts are at least a little bit similar to each other, and then returning the split with the highest average similarity. We choose the average of the similarities because a product would massively favour short splits, and a sum would favour longer splits. Comparing only adjacent parts makes sense, because we assume only they share some meaning: In *Taschenmesserklinge* (pocket knife blade), *Tasche* (pocket) and *Messer* (knife) are often seen together, even more so with *Messer* and *Klinge* (blade), but *Tasche* and *Klinge* are likely not to be similar at all.

One of the proposed splits is always the hypothesis that the given word isn't a compound, we also need to assign a value to this case. A default value of 1 would make it such that this ranking method never splits, making it equivalent with the baseline described in Chapter *Evaluation*, a value of 0 would force it to always split, if possible. Since neither of those options are acceptable, and any other value in-between would be a magic number, having to be tuned for each language, we instead use this method as a secondary ranking method if the primary method can't decide between several options.

Typically, lemmatization or stemming is used as a preprocessing step for creating vector spaces. On any given language, we won't necessarily have tools for that available, which is why we use a very primitive stemming method for all languages: From all words, we strip away any characters after the 6th one. We assume that this will be enough to uniquely identify most words, but also to strip away any possible suffixes. Using this process, *goat* remains as is, but *hamster* is transformed to *hamste*.

We use the **gensim** implementation [Řehůřek and Sojka, 2010] of the skip-gram model. As the `gensim` package has quite a few bigger dependencies[12], we save the vector spaces as pickled objects[13], which means that `gensim` will only be loaded when the respective option is enabled.

We use the raw EMEA corpus for training the embeddings, throwing away any tokens that contain non-alphabetic characters. The corpus contains about 23 million tokens in every language.

Forcing decomposition

Finally, we experiment with whether it makes sense to insist on splitting words, whenever possible: This setting causes the ranking system to always produce a

[12] `smart-open`, `six`, `scipy`, and `numpy`

[13] `https://docs.python.org/3/library/pickle.html`

split with more than one part (i.e. the claim that any word is a compound).

We don't expect this to do better than the other methods, but assume it might be useful when it is already known that a word must be a compound.

Interface

The system can be used in two different ways: As an imported module, or using its command-line interface. The latter is described here.

The script takes one required argument, which is the name of a file which contains the candidates to decompound. This file should contain a single word per line. When the decomposition should happen on demand, the file name can be replaced with `/dev/stdin` to instead read from standard input.

In addition, the script can take a number of options:

- A two-letter language code can be specified using the `--lang` option. This defaults to `--lang=de`, German.
- If the list of stop words should not be used, one can pass `--stopwords=no`.
- If the system should try to split words whenever possible, one can pass `--force-split=yes`.
- The system uses the word frequencies from the corpus by default. If it shouldn't do so and instead resort to the more basic method, the switch `--use-counts=no` will change the behavior.
- To specify a minimum word frequency under which words should be discarded from inclusion in the lexicon, the option `--min-freq=...` can be used. By default, a value of 2 is used, meaning that any word occurring less than twice will be discarded.
- By default, the lexicon is read using the `latin-1` encoding. Although incorrectly encoded files will typically still be decoded correctly using `ftfy`, the argument `--encoding=...` can be passed to the script to explicitly set the encoding.Possible values are dependent on Python's supported codecs[14].
- If the vector spaces should be used as a secondary ranking method, the option `--use-vectors=yes` can be provided. The vector spaces can be found in the `vec` directory.

When called, the script first reads in the dictionary and then the list of stop words from two files in the `lex` directory.

The dictionary is expected to contain one word per line, in the format *frequency*\t*word*\n[15]. The stop word list simply consists of one stop word per line.

After this initial setup is done, the system will read words from the input file, one at a time. The decompounded result is then printed to the standard output, in the same format the evaluation corpus uses.

[14] `https://docs.python.org/3/library/codecs.html#standard-encodings`

[15] The system will also accept lines that start with some amount of whitespace. This is such that it can handle output from `cat raw_words.txt | sort | uniq -c | sort -nr`.

Because it doesn't wait for the input file to be read in completely, it is possible to run the decompounding system as a server and use it on-demand, for example using a named pipe created using `mkfifo`.

When used from another Python script, the returned value can alternatively be in tuple form, then the result will be a tuple of compound parts.

The source code, along with all the lexical resources required to get it to run, is included as Attachment 1 (`compound-splitter.tgz`).

Evaluation

Our system is evaluated against other systems and a baseline here. The competing systems are:

- **Baseline:** This is a hypothetical[16] identity system which when given any word, returns the same word. When seen as a decompounding system, it always postulates that any given word isn't a compound. Since most words in the evaluation corpus aren't compounds, its Accuracy will be fairly high, but Recall and F1-score are zero (since it never finds a single compound), and its Precision is undefined.
- **ASV Toolbox:** This is a supervised system described in Biemann et al. [2008].
- **compound-splitter-nl:** This is an apparently unpublished system[17] by Katja Hofmann, Valentin Jijkoun, Jaap Kamps, and Christof Monz (in alphabetical order). The system is primarily designed for the Dutch language, but can be configured to work for German, Finnish, Swedish, and Afrikaans, too.
- **jWordSplitter:** Originally created by Sven Abels, and now maintained by Daniel Naber, this is a rule-based Java library for splitting German compounds[18].

Our system is evaluated in various configurations:

- **basic:** This is the most basic version using the raw words without frequency information. The splits are ranked based on how many parts are known words first, and if there's a tie, based on the amount of (non-linking-morpheme) parts (fewer ranks higher).
- **basic+stop:** The same as **basic**, but with stop words removed.
- **basic+counts:** This is like **basic**, but it ranks the splits based on the geometric means of the frequencies of the parts.
- **basic+counts+stop:** This is like **basic+counts**, but with stop words removed and affix-related filters active.
- **basic+counts+stop+force:** This is like the previous system, but will assume the word is a compound and always return a split with more than one part, if possible. This could be useful when used on top of another system that knows a word is a compound, but doesn't know how to split it.
- **basic+counts+stop+sem:** This builds on top of **basic+counts+stop** to rank the candidate splits, falling back to average semantic similarity of their parts when there's a tie.

[16] It wasn't actually implemented as its performance metrics are evident from the data.
[17] `http://ilps.science.uva.nl/resources/compound-splitter-nl/`
[18] `https://github.com/danielnaber/jwordsplitter`

	Precision	**Recall**	**Accuracy**	**F1**
Baseline	-	.00	.80	.00
ASV Toolbox	.41	.42	.75	.41
compound-splitter-nl	.23	.05	.77	.09
jWordSplitter	**.88**	.57	**.91**	**.69**
basic	.44	.18	.81	.26
basic+stop	.47	.18	.81	.25
basic+counts	.49	.53	.84	.51
basic+counts+stop	*.59*	.59	*.88*	.59
basic+counts+stop+force	.57	.57	.87	.57
basic+counts+stop+sem	*.59*	***.60***	*.88*	*.60*

Table 4: Performance of all systems compared on the German evaluation corpus. The best system for a score is marked in bold, the best of our systems in italic.

Since this isn't a simple binary classification task, the definitions for Precision and Recall (and therefore, F-Score) differ slightly:

$$\text{Precision} = \frac{\text{correct splits}}{\text{correct splits} + \text{superfluous splits} + \text{wrong splits}}$$

$$\text{Recall} = \frac{\text{correct splits}}{\text{correct splits} + \text{wrong splits} + \text{incorrect non-splits}}$$

As accuracy is only dependent on the correctness of the work, it keeps the familiar definition:

$$\text{Accuracy} = \frac{\text{correct splits} + \text{correct non-splits}}{\text{all instances}}$$

The F1-score is defined as a trade-off between Precision and Recall, giving them equal importance:

$$\text{F1-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Whether the F1-score is a fitting evaluation metric depends on the task at hand, on what is more important; Precision or Recall. Since we report both of them here, too, only the F1-score is included here, but any other kinds of F-scores can easily be computed from Precision and Recall.

The results for the German evaluation task can be seen in Table 4. As with all classification systems, the question is which metrics are more important to the user:

Putting an emphasis on Recall means caring more about splitting all compounds than about potential incorrect decompositions. Higher Precision focuses on being more "careful", and prefering to decompound easier compounds correctly, rather than decompounding too much. Since most words in the evaluation corpora aren't compounds, obtaining a high accuracy value is easy: The simple identity baseline obtains 80% accuracy, because 80% of the instances in the German corpus aren't compounds. One could argue for such a system to have perfect Precision, but it is marked here as not having a value. However, a system that has a few cases hard-coded could still easily obtain a precision of 100%.

	Precision	**Recall**	**Accuracy**	**F1**
Baseline	-	.00	.72	.00
compound-splitter-nl	.26	.04	.70	.07
basic	.35	.21	.73	.26
basic+stop	.38	.21	.74	.27
basic+counts	.47	.33	.77	.39
basic+counts+stop	.52	.34	***.78***	.41
basic+counts+stop+force	***.53***	.35	***.78***	.42
basic+counts+stop+sem	.52	***.37***	***.78***	***.43***

Table 5: Performance of all systems compared on the Swedish evaluation corpus. The best system for a score is marked in bold, the best of our systems in italic.

In this task, it isn't as easy to obtain a high Recall. Whereas in binary classification tasks selecting *every instance* yields a recall of 100%, due to the more complex job a decompounding system has, the same result can't be obtained here.

In this case however, the rule-based **jWordSplitter** clearly outperforms all other systems, losing only in Recall to our best systems.

Although forcing decomposition expectedly doesn't improve the basic system, all other improvements seem to do, improving on all performance metrics. **basic+counts+stop+sem** is a special case, it improves Recall (and therefore F-Score) so slightly, that one could argue not to use it at all, because of the work involved in getting vectors and the required large dependencies.

Since the situation is different in the other languages, we still set the configuration (**basic+counts+stop+sem**) as the default configuration for our system.

In the Swedish dataset (see Table 5), the situation is similar, but here the system using word embeddings is the clear winner, and even forceful decompounding works better than just **basic+counts+stop**. Overall, the scores are considerably lower. This could be either due to the smaller amount of data available (see Table 1), or possibly due to the task being more difficult in Swedish. This hypothesis is supported by the accuracy of the baseline being lower here, too.

On the Hungarian corpus, the results are different: Here no clear winner can be seen, although the overall best system is probably **basic+counts+stop**.

The more complicated methods using stop words and forcing decompounding lose in Recall what they gain in Precision. Adding word embeddings as a secondary ranking method retains the same F-Score, while improving Recall and hurting Precision.

Annotation quality discrepancies could also have played a role in the differences between the languages, but since the author speaks neither Swedish nor Hungarian, that is merely another hypothesis.

It should be noted, that the numbers given here don't necessarily say much about the usefulness of the decompounding systems within other tasks (this is why it can be more insightful to evaluate using a task-based method; i.e. measuring how much a system's results on a task improve when using decompounding, this is e.g. how Daiber et al. [2015] and Erbs et al. [2015] evaluate):

A given system using a decompounder is typically not completely unknowl-

	Precision	**Recall**	**Accuracy**	**F1**
Baseline	-	.00	.85	.00
basic	.22	.16	.77	.18
basic+stop	.33	.21	.82	.25
basic+counts	.53	***.30***	.87	.38
basic+counts+stop	.64	.29	***.88***	***.40***
basic+counts+stop+force	***.69***	.21	.87	.32
basic+counts+stop+sem	.59	***.30***	.87	***.40***

Table 6: Performance of all systems compared on the Hungarian evaluation corpus. The best system for a score is marked in bold, the best of our systems in italic.

	German	**Swedish**	**Hungarian**
Baseline	.00	.00	.00
ASV Toolbox	**.95**		
compound-splitter-nl	.08	.06	
jWordSplitter	.62		
basic	.29	.41	***.40***
basic+stop	.27	.39	.36
basic+counts	*.84*	.52	.29
basic+counts+stop	.78	.50	.26
basic+counts+stop+force	.78	.50	.26
basic+counts+stop+sem	.80	***.53***	***.40***

Table 7: Coverage of the various systems on all languages

edgable about what word is a compound: It will often have POS-tagged the source text already, and might only resort to using the decompounding system when a given word can't be found in a dictionary. In such a case, Recall can be more important than Precision.

We therefore also report coverage percentages, defined as the fraction of compounds that are correctly identified as compounds by the various systems (but not necessarily correctly split):

$$\text{Coverage} = \frac{\text{correct splits} + \text{incorrect splits (of compounds)}}{\text{all compounds}}$$

The Coverage values can be seen in Table 7 for all languages. In this case, Coverage can't be seen as a direct measure of system quality, but more of the eagerness of a system to split. From our systems, one would expect **basic+counts+stop+force** to be the one with the highest coverage, because it literally splits whenever it can, but the usage of stop words and affixes will reduce the amount of times the system splits. The highest coverage is achieved in German by a system that doesn't use stop words, and therefore splits more, but it is not the best of our system configurations. In Swedish and Hungarian, the system falling back to using word embeddings has the highest coverage, because like **basic+counts+stop+force**, it splits whenever there's a tie between systems in the first ranking

	system	under-split	over-split	wrongly-split
German	ASV Toolbox	12	214	7
	compound-splitter-nl	176	40	0
	jWordSplitter	76	13	0
	basic	144	30	8
	basic+stop	148	22	8
	basic+counts	31	82	36
	basic+stop+counts	42	35	40
	basic+stop+counts+sem	38	37	40
Swedish	compound-splitter-nl	152	20	0
	basic	102	28	25
	basic+stop	104	16	27
	basic+counts	80	24	28
	basic+stop+counts	84	12	28
	basic+stop+counts+sem	78	16	29
Hungarian	basic	324	437	41
	basic+stop	342	247	32
	basic+counts	373	69	11
	basic+stop+counts	391	25	10
	basic+stop+counts+sem	324	75	19

Table 8: Error analysis. *under-split* are those instances that are split into less parts than they should have been. *over-split* are those instances that are split into more parts than they should have been. *wrongly-split* are those instances, which have the right amount of splits, but are incorrectly split.

method, given that any similarity value between two vectors will be higher than 0.

We also do an error analysis of all systems but **basic+stop+counts+force** (because the mistakes it makes are expectable) in Table 8. Most of the time a system makes a mistake, it is in how many splits a word should be split, rarely is there a case where a system splits the compound in the right amount of pieces, but in the wrong position(s). This is to be expected. From our systems, we can see that once word frequencies are introduced, the amount of under-split compounds is drastically reduced. The only exception is Hungarian, where it, for whatever reason, behaves the other way around: Here, frequency information reduces over-split compounds, increasing the amount of under-split instances slightly. Introducing word embeddings decreases under-splits and increases over-splits, as expected.

Interestingly, while in German we reach almost the same number of over- and under-splits, in the other languages there are more under-splits by quite a bit. This points to too many words not being known; i.e. a corpus that is too small.

Conclusion

This work compared previous work on decompounding applied to the medical domain, and developed a new system that is compatible with German, Swedish, and Hungarian, and can easily be extended to work with other languages as well.

It is clear that as easy as the task might seem at first, it is far from solved. Given a good lexicon, all systems can decompound obvious, clear cases, but more complex compounds, those that are ambiguous, and those being noisy in other ways are still hard to split.

This is especially true in a specific domain such as the medical one, as the language and the used compounds are different from regular language. While some compounds, like *Krankenschwester* (nurse, literally "sick sister", as in "sister for the sick") are in common use in colloquial language, more specific terms, such as *Thrombozytenzahlen* are not. This makes the task more difficult than it would be for non-expert language, and explains the comparably low results of all systems when compared to other evaluations (such as in Erbs et al. [2015]).

It is also apparent that the amount of data used influences the accuracy of the various decompounding systems, as the results for German are consistently higher than for the other languages.

Evidently this kind of language-agnostic word splitting system is one with limited use. When building a system for a specific language, one could employ any number of language-specific improvements. While we did end up using word embeddings to improve the ranking procedure, our stemming method was very primitive in order to stay language-independent. It was nevertheless required: Given a candidate split S consisting of the parts A, B, we need to either lemmatize or stem both A and B to be able to map them to their semantic vectors. One benefit of our basic left-to-right system described in the Chapter *Methods* is that we can largely ignore morphological changes like suffixes, because they typically only happen to the head, apart from the binding morphemes that we do handle. This is why we also chose such a simple stemming method, but better pre-processing tools would probably improve our results.

The downsides of our system are also its benefits: Without any annotated training data, it will likely work for any additional languages with composition systems similar to Germanic, Skandinavian, or Uralic languages. For example, Modern Greek also exhibits compounding, and given a Wikipedia dump or something similar as a source corpus, it is hypothesized that it will also work for Greek.

Like all systems not exclusively concerned with semantics, it can handle both endocentric and exocentric compounds, while some of the systems, like those that are only based on word embeddings can't decompound exocentric compounds (e.g. Daiber et al. [2015]).

For future work we propose the following:

- **Evalutating on other domains.** Our medical evaluation corpus has proven to be a tough one for all systems. A cleaner, balanced evaluation corpus not restricted to a specific domain would be more informative regarding the performance of decompounding systems in naturally occurring language.

- **Learn from easy compounds.** A system could, instead of splitting compounds one by one, require a list of words to decompound, before it will start the decompounding process. It could then first decompound those instances that are less ambiguous, or for other reasons easier to decompound, and then learn from those presumably correct splits. This way, the system would stay unsupervised, but still acquire examples to learn from during execution. In very recent work, Riedl and Biemann [2016] use a similar method to generate a dictionary of "single atomic word units".

- **Evaluation on non-compounding languages.** As a sort of "sanity check", one could run decompounding systems on non-compounding languages. Ideally, very few words would be tagged as compounds and split, except possibly for lexicalized compounds.

- **Knowledge transfer from high-resource languages.** We purposefully don't use methods requiring large amounts of data, such as word embeddings, or tools such as POS-taggers. One way of staying true to our goals, but still using such methods would be to transfer information from high-resource languages to low-resource languages: For example, word vectors could be first trained on German corpora, and then retrained on other (Germanic) languages, or a (more sophisticated) stemmer from German could be blindly applied to other languages.

Looking back at our goals defined in the Chapter *Introduction*, **Language agnosticism** is achieved, because the only knowledge required to extend the system to another language is a list of linking morphemes of that language. Even when not supplying this list, the system will work, although to a lesser degree.

Another big goal was **low resource friendliness**. Corpora of the size we used are available in many languages, not just the few big ones with lots of attention from the NLP community. We also don't require the existence of any other parts of the NLP pipeline (stemmers, lemmatizers, POS-taggers, parsers, ...).

In terms of **usability**, if the word embedding method isn't used[19], the system has only two dependencies, both only a few kilobytes large. Once the dictionary and the lists of stop words and affixes are read in, decomposition takes less than a millisecond even for long, complex examples.

[19]The `gensim` package required for the use of our embeddings requires bigger dependencies, but those (namely `numpy` and `scipy`) are frequently used packages in the scientific Python community.

Bibliography

Martine Adda-Decker and Gilles Adda. Morphological decomposition for ASR in german. In *Workshop on Phonetics and Phonology in ASR, Saarbrücken, Germany*, pages 129–143, 2000.

Enrique Alfonseca, Slaven Bilac, and Stefan Pharies. German Decompounding in a Difficult Corpus. pages 128–139, 2008.

Marco Baroni, Johannes Matiasek, and Harald Trost. Predicting the components of german nominal compounds. In *ECAI*, pages 470–474, 2002.

Chris Biemann, Uwe Quasthoff, Gerhard Heyer, and Florian Holz. ASV toolbox: a modular collection of language exploration tools. *Proceedings of LREC*, (2): 1760–1767, 2008.

Claudia Bretschneider and Sonja Zillner. Semantic splitting of german medical compounds. In *International Conference on Text, Speech, and Dialogue*, pages 207–215. Springer, 2015.

Joachim Daiber, Lautaro Quiroz, Roger Wechsler, and Stella Frank. Splitting compounds by semantic analogy. *arXiv preprint arXiv:1509.04473*, 2015.

Nicolai Erbs, Pedro Bispo Santos, Torsten Zesch, and Iryna Gurevych. Counting What Counts: Decompounding for Keyphrase Extraction. *Acl-Ijcnlp 2015*, pages 10–17, 2015.

Philipp Koehn and Kevin Knight. Empirical Methods for Compound Splitting. 2002.

Stefan Langer. Zur Morphologie und Semantik von Nominalkomposita. In *Tagungsband der 4. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS)*, pages 83–97, 1998.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

Rochelle Lieber and Pavol Stekauer. *The Oxford handbook of compounding*. Oxford University Press, 2009.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

Torsten Marek. Analysis of German Compounds using Weighted Finite State Transducers. *Bachelor thesis, University of Tübingen*, 2006.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

Ulrike Rackow, Ido Dagan, and Ulrike Schwall. Automatic translation of noun compounds. In *Proceedings of the 14th conference on Computational linguistics-Volume 4*, pages 1249–1253. Association for Computational Linguistics, 1992.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

Martin Riedl and Chris Biemann. Unsupervised compound splitting with distributional semantics rivals supervised methods. In *Proceedings of NAACL-HLT*, pages 617–622, 2016.

Anne Schiller. German compound analysis with wfsc. In *International Workshop on Finite-State Methods and Natural Language Processing*, pages 239–246. Springer, 2005.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465, 2013.

Radu Soricut and Franz Och. Unsupervised morphology induction using word embeddings. In *Proc. NAACL*, 2015.

Jörg Tiedemann. News from opus-a collection of multilingual parallel corpora with tools and interfaces. In *Recent advances in natural language processing*, volume 5, pages 237–248, 2009.

Attachments

The following attachments have been uploaded to SIS.

1. **compound-splitter.tgz:** A gzip file containing the source code of the compound splitter, along with all the required lexical resources (frequency information, affix and stop word lists; and vector spaces).

2. **eval-corpus.tgz:** The evaluation corpus in German, Swedish, and Hungarian. Each file contains one word-annotation pair per line, separated by a tab character.