

Pre-processing text for web information retrieval purposes by splitting compounds into their morphemes

Sven Abels, Axel Hahn
Department of Business Information Systems,
University of Oldenburg, Germany
{ abels | hahn } @ wi-ol.de

Abstract

In web information retrieval, the interpretation of text is crucial. In this paper, we describe an approach to ease the interpretation of compound word (i.e. words that consist of other words such as “handshake” or “blackboard”). We argue that in the web information retrieval domain, a fast decomposition of those words is necessary and a way to split as many words as possible, while we believe that on the other side a small error rate is acceptable.

Our approach allows the decomposition of compounds within a very reasonable amount of time. Our approach is language independent and currently available as an open source realization.

1. Motivation

In web information retrieval, it is often necessary to interpret natural text. For example, in the area of web search engines, a large amount of text has to be interpreted and made available for requests. In this context, it is beneficial to not only provide a full text search of the text as it is on the web page but to analyze the text of a website in order to, e.g., provide a classification of web pages in terms of defining its categories (see [1], [2]).

A major problem in this domain is the processing of natural language (NLP; see e.g. [3], [4] for some detailed descriptions). In most cases, text is prepared in terms of being preprocessed before it is analyzed. A very popular method is text stemming, which creates a basic word form out of each word. For example the word “houses” is replaced with “house”, etc. (see [5]). A popular approach for performing text stemming is the porter stemmer [6]. Apart from stemming text, the removal of so called “stop words” is another popular approach for pre-processing text (see [7]). It removes all unnecessary words such as “he”, “well”, “to”, etc. While both approaches are well established and quite easy to implement, the problem of splitting compounds into its morphemes is more difficult and less often implemented. Compounds are words that consist of two or more other words (morphemes). They can be found in many of today’s language. For example, the English word “handshake” is composed of the two

morphemes “hand” and “shake”. In German, the word “Laserdrucker” is composed of the words “Laser” (laser) and “Drucker” (printer). In Spanish we can find “Ferrocarril” (railway) consisting of “ferro” (iron) and “carril” (lane).

Splitting compounds into their morphemes is extremely useful when preparing text for further analysis (see e.g. [10]). This is especially valid for the area of web information retrieval because in those cases, you often have to deal with a huge amount of text information, located on a large number of websites. Splitting compounds helps to detect the meaning of a word easier. For example, when looking for synonyms, a decomposition of compound words will help because it is usually easier to find word-relations and synonyms of morphemes than to look at the compound word.

Another important advantage when splitting compounds is the capability of stemming compounds. Most stemming algorithms are able to stem compounds correctly if their last morpheme differs in its grammatical case or in its grammatical number. For example, “sunglasses” will correctly be stemmed to “sunglass” in most algorithms. However: There are cases, where a stemming does not work correctly for compounds. This appears, whenever the first morpheme changes its case or number. For example the first word in “Götterspeise” (English: jelly) is plural and the compound will therefore not be stemmed correctly. Obviously, this will result in problems when processing the text for performing, e.g., text searches.

A nice side effect when splitting compounds is that you can use the decomposition to get a unified way of processing words that might be spelled differently. For example one text might use “containership”, while another one uses “container ship” (2 words). Using a decomposition will lead to a unified way.

2. Difficulties and Specific Requirements

Splitting compounds in the domain of web information retrieval has some specific requirements. Since web information retrieval tasks usually have to deal with a large amount of text information, an approach for splitting compounds into its morphemes has to provide a

high speed in order to keep the approach applicable in praxis. Additionally, a low failure rate is of course crucial for the success. Hence, it is necessary to use an approach that provides a very high speed with an acceptable amount of errors.

Another specific requirement for applying a decomposition of words in web information retrieval is the high amount of proper names and nouns that are concatenated to proper nouns or figures. For example, we might find words such as “MusicFile128k” or “FootballGame2005”. Because of that, it is necessary to use an approach that can deal with unknown strings at the end or at the beginning of a compound.

The most difficulty in splitting compounds is of course the detection of the morphemes. However, even when detecting a pair of morphemes, it does not mean that we have a single splitting, that is correct for a word. For example, the German word “Wachstube” has two meanings and could be decomposed into “Wachs” and “Tube” (“wax tube”) or into “Wach” (guard) and “Stube” (house), which means “guardhouse”. Obviously, both decompositions are correct but have a different meaning.

In the following section, we will describe an approach for realizing a decomposition of compounds into morphemes, which is designed for dealing with a large amount of text in order to be suited for web information retrieval. In this approach, we focus on providing a high speed with a small failure rate, which we believe is acceptable.

2. Description of the Approach

In our approach, we sequentially split compound words in three phases:

1. direct decomposition of a compound,
2. truncation of the word from left to right and
3. truncation of the word from right to left.

In the first phase, we try to directly split the composed word by using a recursive method `findTupel`, which aims in detecting the morphemes of the word and returns it as an ordered list. In case of not being able to completely decompose the word, we truncate the word by removing characters starting at the left side of the word. After removing a character, we repeat the recursive `findTupel` method. If this does not lead to a successful decomposition, we use the same methodology in the third step to truncate the word from right to left. This enables us, to successfully split the word “HourseboatSeason2005” into the tokens { “House”, “Boat”, “Season”, “2005” } as discussed in the last section.

Before starting with the analysis of the word, all non-alphanumeric characters are removed and the word is transformed into lower case.

The main task of our approach is performed in a recursive way. This is to be realized as a single `findTupel` method with one parameter, which is the current compound that should be decomposed into morphemes. In case that this word is smaller then 3 characters (or null), we simply return the word as it is. In all other cases, it is decomposed into a left part and a right part in a loop. Within each loop, the right part gets one character longer. For example, the word “hourseboat” will be handled like this:

Loop No	Left part	Right part
1	Houseboa	t
2	Housebo	at
3	Houseb	oat
4	House	boat
...

Table 1. Decomposition

Within each loop, it is checked whether the right part is a meaningful word that appears in a language specific wordlist or not. In our implementation, we provided a wordlist of 206.877 words containing different words in singular and plural. In case that the right part represents a word of this wordlist, it is checked if the left part can still be decomposed. In order to do this, `findTupel` method is called again with the left part as a new parameter (recursively). In case that the right part never represents a valid word, the method returns a null value. If the recursive call returns a value, different from null, its result is added to a resulting list, together with the right part. Else the loop continues. This ensures that the shortest decomposition of the compound is returned.

For some languages, compounds are composed by adding a connecting character between the morphemes. For example, in the Germany language, one can find an “s” as a connection between words. In order to consider those connecting characters, they are removed when checking if the right part is a valid word or not.

3. Managing problems

Basically, the success rate is highly influenced by the quality and completeness of the language specific word list. The approach benefits from a large amount of words. In order to ensure a high speed, we use a hash table for managing the wordlist.

A major problem of this approach is the existence of very small words in a language that might lead to a wrong decomposition. For example the word “a” in English or the word “er” (he) in German can lead to results that change the meaning of the word. For example, our approach would decompose the word “Laserdrucker” into “Las”, “Er”, “Druck”, “Er” (read-he-print-he) instead of

“Laser”, “Drucker” (laser printer). In order to avoid this, we use a minimum length of words, which in the current implementation is a length of 4. This made the problem almost disappear in practical scenarios.

4. Related work

There are of course several other approaches that can be used for decomposing compounds. One example is the software Machine Phrase Tagger from Connexor¹, which is a “word tagger” used for identifying the type of a word. It can, however, also be used to identify the morphemes of a word but is quite slow on large texts. Another example is SiSiSi (Si3) as described in [8] and [9]. It was not developed for decomposing compounds but for performing hyphenations. It does, however, identify main hyphenation areas for each word, which is in most cases identical with the morphemes of a compound. More examples can be found in [10] and [11].²

Existing solutions were, however, not developed for the usage in the web information retrieval domain. This means that many of them have a low failure rate but do also need a lot of time compared to our approach. In the following section we will therefore perform an evaluation, analyzing the time and quality of our approach.

5. Realization and Evaluation

The approach was realized in Java with the name jWordSplitter and was published as an open source solution using the GPL license. We used the implementation to perform an evaluation of the approach. We analyzed the implementation based on (i) its speed and (ii) its quality since we think that both is important for web information retrieval.

The speed of our approach was measured in three different cases.

- Using compounds with a large amount of morphemes (i.e. consisting of 5 or more morphemes). In this case, our approach was able to split about 50.000 words per minute.
- Using compounds that consist of 1 or 2 morphemes (e.g. “handwriting”). In This case, our approach has been able to split about 150.000 words per minute.
- Using words that do not make any sense and cannot be decomposed at all (e.g. “Gnuavegsdfweeerr”). In

¹ <http://www.connexor.com>

² Please note that pure hyphenating is not the same as word splitting. It is only equivalent in those cases where each root word has only one syllable.

this test, jWordSplitter was able to process 120.000 words per minute.

In order to test the quality of the approach, we took a list of 200 randomly chosen compounds, consisting of 456 morphemes. The average time for splitting a word took about 80 milliseconds. Within this test set, jWordSplitter has been unable to split about 5% of the words completely. Another 6% have been decomposed incorrectly. Hence, 89% have been decomposed completely and without any errors and about 94% have been either composed correctly or at least not been composed incorrectly.

We performed the same test with SiSiSi, which took about twice as long and which was unable to split 16% of the words. However, their failure rate was a bit less (3%).

7. Conclusion

We have presented an approach, which we argue is suited for using it as a method for preparing text information in web information retrieval scenarios. The approach offers a good compromise between failure rate, speed and ability to split words. We think that in this domain it is most important to split as much words as possible in a short period of time, while we believe that a small amount of incorrect decompositions is acceptable for achieving this.

Our approach can be used to ease the interpretation of text. It could for example be used in search engines and classification methods.

8. Further research and language independence

In order to test the effectiveness of our approach, we intend to integrate it into the “Apricot”-project. It is proposed to offer a complete open source based solution for finding product information on the internet. We therefore provide a way of analyzing product data automatically in order to allow a fast search of products and in order to classify the discovered information. The integration of jWordSplitter in this real-world project will help to evaluate its long-term application and will hopefully also lead to an identification of problematic areas of the approach.

An interesting question is the language independence of the approach. jWordSplitter itself is designed to be fully language independent. It is obvious that its benefit does, however, vary between languages. While word splitting is very important for languages with many compounds, it might lead to fewer advantages in other languages. We therefore intend to extend Lucene³, an open source search engine by preprocessing text with

³ <http://lucene.apache.org>

jWordSplitter. Afterwards, we will rate its search results before and after the integration. We intend to repeat this test for different languages in order to get a language dependent statement about the benefits of jWordSplitter.

9. Acknowledgement and Sources

In the current implementation, it contains a German wordlist. Since the approach itself is language independent, it can be used for other languages as well if a wordlist is provided.

The sources for jWordSplitter are available online as an open source project using the GPL at:

<http://www.wi-ol.de/jWordSplitter>

10. References

- [1] Eliassi-Rad, T.; Shavlik, J. W.: *Using a Trained Text Classifier to Extract Information*, Technical Report, University of Wisconsin, 1999
- [2] Jones, R.; McCallum, A.; Nigam, K.; Riloff, E.: *Bootstrapping for Text Learning Tasks*, In IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications, pp. 52-63. 1999.
- [3] Wilcox A.; Hripcsak G. *Classification Algorithms Applied to Narrative Reports*, in: Proceedings of the AMIA Symposium, 1999
- [4] Harman, D; Schäuble, P.; Smeaton, A.: *Document Processing*, in: Survey of the state of the art in human language technology, Cambridge Univ. Press, 1998
- [5] Jones, S.; Willet, K.; Willet, P.: *Readings in Information Retrieval*, Morgan Kaufmann, 1997
- [6] Porter, M.F.: *An algorithm for suffix stripping*, Program, 14(3), 1980
- [7] Heyer, G.; Quasthoff, U.; Wolff, C.: *Möglichkeiten und Verfahren zur automatischen Gewinnung von Fachbegriffen aus Texten*. In: Proceedings des Innovationsforums Content Management, 2002
- [8] Kodydek, G.: *A Word Analysis System for German Hyphenation, Full Text Search, and Spell Checking, with Regard to the Latest Reform of German Orthography*. In: Proceedings of the Third International Workshop on Text, Speech and Dialogue (TSD 2000), Springer-Verlag, 2000
- [9] Kodydek, G.; Schönhacker, M.: *Si3Trenn and Si3Silb: Using the SiSiSi Word Analysis System for Pre-Hyphenation and Syllable Counting in German Documents*, Proceedings of the 6th Internat. Conference on Text, Speech and Dialogue (TSD 2003), Springer-Verlag, 2003
- [10] Andersson, L.: *Performance of Two Statistical Indexing Methods, with and without Compound-word Analysis*, www.nada.kth.se/kurser/kth/2D1418/uppsatser03/LindaAndersson_compound.pdf, 2003
- [11] Neumann, G.; Piskorski, J.: *A Shallow Text Processing Core Engine*. In: Proceedings of Online 2002, 25th European Congress Fair for Technical Communication, 2002