Selective compound splitting of Swedish queries for Boolean combinations of truncated terms

Rickard Cöster, Magnus Sahlgren and Jussi Karlgren

Swedish Institute of Computer Science, SICS,
Box 1263, SE-164 29 Kista, Sweden

{*rick, mange, jussi*}@*sics.se*

Abstract

Swedish is a compounding language, and therefore it is important to split compound words so that useful word constituents can be found. One of the problems is that it is difficult to find constituents that express a concept similar to that expressed by the compound.

The approach taken in this paper is to look at how the leading constituent of the compound word can be used to expand a search query. The constituent was added to the original query, while still keeping the compound. Every word was then truncated so as to increase recall by hopefully finding other compounds with the leading constituent as prefix. Since this approach increase recall in a rather uncontrolled way, we also used a Boolean quorum-level type of query combination so that documents were ranked according to both the tf-idf factor but also to the number of matching Boolean combinations.

The Boolean combinations performed relatively well, taken into consideration that the queries were very short (maximum five search terms).

Also included in this paper are the results of two other methods we are currently working on in our lab; one for re-ranking search results on the basis of stylistic analysis of documents, and one for dimensionality reduction using Random Indexing.

1 Introduction: Compounds in Swedish

This year, we focused on the Swedish monolingual track. We submitted four runs, of which the first two deal with the problem of using compound word splitting for query expansion.

The other two runs test very different approaches: first, how to re-rank search results based on a stylistic analysis of the retrieved documents and, secondly, the effect of aggressive dimensionality reduction using Random Indexing.

Swedish is a compounding language. This means that new words are often formed by adjoining two or more separate words. Such words are called *closed* compounds. For example, the Swedish word "Diamantgruva" is a closed compound of "Diamant" (Diamond) and "Gruva" (Mine). The other two forms of compound words are the *open* form ("Post office") and the *hyphenated* ("Long-term"). The closed form is common in Swedish (and languages such as German, Finnish and Danish) whereas the open form is more common in English.

Certainly, it is necessary for a retrieval system to split the compounds into constituents, since such words may be too specific but contain constituents that are useful and content-bearing. One of the problems of compound splitting is that it is difficult to find constituents that express a concept [1] that is similar to that expressed by the compound.

For example, the word “Diamantgruva” is important to split so that the content-bearing words “Diamant” and “Gruva” can be found. These two words are good query terms, since they are not ambiguous and express the two separate concepts diamond and mine. On the other hand, if the word “Domstol” (Court) is split into “Dom” (Judgement) and “Stol” (Chair), some errors are introduced. “Stol” is not a good search term in this context, and “Dom” is ambiguous; it also means “Them” in spoken Swedish. Splitting “Domstol” thus make more harm than keeping the compound form.

Another problem that may occur can be exemplified by splitting the compound “Riksdagshus” (Parliament building). The lemmatizer that we used in our experiments split this into the three constituents “Riks”, “Dags” and “Hus”. It did not find the word “Riksdag” (Parliament) which is an important word in this context. Furthermore, the words “Riks” and “Dags” should not have been extracted as constituents, since “Riksdag(s)” is a not a compound. Moreover, it seems that the lemmatizer does not make a morphological analysis of the constituents. This has the effect that sometimes a joining ’s’ is left at the end of words that should not have it.

In summary, some of the problems with using constituents from compound splitting are that they

- may not express a concept similar to that expressed by the compound
- may be ambiguous
- may not always be valid words

2 Selective compound splitting and Boolean combinations

Since compound splitting may not always yield constituents that improve a query, it is desirable to have a method for selecting only a subset of the constituents. Since compounds are often very specific words, one way to improve the query is to find constituents that boost the recall. In our case, we chose to select only the leading constituent for expanding the query.

The leading constituent is sometimes a modifier to the last constituent, i.e. it determines something about the last constituent. For instance, the leading constituent “Ozon” (Ozone) in “Ozonlager” (Ozone layer) determines that the layer mentioned is the ozone layer. The word “Ozon” is useful as a search term, since documents about ozone layers probably also contain the separate word “Ozon”. Such documents might also contain other compounds that begin with “Ozon” such as “Ozonhalt” (Ozone amount) and “Ozonhål” (Ozone hole), so finding these words might also help improve the query.

By expanding a query with the leading constituent of a compound search word will effectively increase the recall, and hopefully in such a way that the new documents that are found are related to the concept expressed by the compound. And if we also expand the query with all words that begin with the leading constituent, we will find new words that again, hopefully, are related as well. But we will also find words that are not related to the concept at all, so there must be some way of narrowing the query as well.

Our general approach in runs `sicsSVtad` and `sicsSVtmd` was to expand the queries with leading constituent. These were added as search words while also keeping the original compound. Each search word was then truncated, i.e. we found all other indexed words that had the search word as prefix. All search words were truncated, not only the leading compounds. For instance, the truncation of “Diamant” found “Diamantexport” (Diamond export), “Diamantföretag” (Diamond corporation) and also “Diamantgruva” mentioned earlier.

We decided to do the compounds splitting at query time, so that we could elaborate with how to select good constituents for improving the queries.

[1]By concept, we mean search term

2.1 Retrieval engine and model

The underlying retrieval engine we used is an experimental system developed at SICS. It currently supports Boolean, Vector Space and structured queries. It is designed to handle a large amount of documents and queries, using algorithms described in [8] and [1] to effectively manage large amounts of data. The system is described in more detail in our CLEF paper [5] from last year.

The Swedish document collection was parsed and normalized using a lemmatizer, but we did not use any compound splitting at indexing time. Also, we used a list of 285 stop words.

For scoring documents, we used *pivoted cosine normalization*, or *Lnu* in Smart notation [7]. We set the slope to 0.3 after some informal experiments, and set the pivot to the average number of unique terms in a document, as suggested in [7].

2.2 Boolean combinations of truncated terms

We used two different approaches to query formulation. For the first run, `sicsSVtad`, we used the selective compound splitting on the *Title* field only. In the second run, `sicsSVtmd`, we again used the *Title* words, but also added some words from the *Desc* field (those with lowest document frequency) so that there was a maximum of 5 words in the query. Each such base word was then truncated, and we performed a ranked Boolean AND between the base words and a ranked Boolean OR between the words found by truncating the base words. In the case where there were less than 1000 documents in a result list, we appended the results of a standard vector space query using all words from the *Desc* field.

To illustrate the Boolean combination procedure, let a, b and c be three query terms, and let $a_1, \ldots, a_n$ be the expanded words from the truncation of a (and $b_1, \ldots, b_m$, $c_1, \ldots, c_r$ for b and c). For each possible Boolean combination of the query terms, we constructed one query. In total this makes 7 queries, displayed in Figure 1.

$$\begin{array}{l}
(a_1 \vee \ldots \vee a_n) \wedge (b_1 \vee \ldots \vee b_m) \wedge (c_1 \vee \ldots \vee c_r) \\
(a_1 \vee \ldots \vee a_n) \wedge (b_1 \vee \ldots \vee b_m) \\
(a_1 \vee \ldots \vee a_n) \wedge (c_1 \vee \ldots \vee c_r) \\
(b_1 \vee \ldots \vee b_m) \wedge (c_1 \vee \ldots \vee c_r) \\
(a_1 \vee \ldots \vee a_n) \\
(b_1 \vee \ldots \vee b_m) \\
(c_1 \vee \ldots \vee c_r)
\end{array}$$

Figure 1: The 7 possible Boolean combinations of three truncated query terms a, b and c

In general, the number of such Boolean query combinations is $\sum_{i=1}^{k} \binom{k}{i}$, where k is the number of terms in the query. This type of Boolean combination is sometimes called *quorum* level search [6], although the standard way of performing the search is to include all combinations of the same size in one and the same query, using the OR operator to combine them. This strategy was not appropriate for this task, since then we would have normalized documents differently and would not have been able to merge the result lists in a simple way.

Since each Boolean combination was a single query, we simply set the RSV for each document to the sum of the RSV values from all queries in that combination. This has the desirable property that documents are not only ranked according to the tf-idf model, but also to the number of combinations of the search words that are spotted in the document. For instance, a document where all search words are found would be at the top of the list, since that document would get positive RSV values from all queries in the combination.

3 Other approaches

Since the Boolean combinations of truncated terms is designed to increase recall, it is important to make sure that we do not add too much noise in this process. One way of doing this is to use stylistic filtering of the retrieved documents to boost news items with animate agents.

3.1 Stylistic filtering to boost news items with animate agents

As the text corpus was composed of news service items with longer more textual pieces, short one-paragraph or one-sentence passages, as well as tables of sports or stock results, a filter to boost the rank of items more likely to be relevant to the textually oriented materials requested for CLEF was designed. The basic assumption of the filter was that items more likely to be relevant would contain more animate agents than others: texts with the personal agents present and with descriptions of actions taken by people or organizations or other animate entities were assumed to be of a higher information value than texts with completely impersonal and non-active constructions.

The style filter was constructed in a multi-step process. First, a set of prototypical animate agents was drawn up. The list used as a seed set can be seen in Table 1. Second, all verbs in the corpus were tabulated by their occurrence with a subject from the seed set of animate agents. Verbs which occurred at least once with one of the prototypically animate agents were noted to be personal verbs - comprising a set of over 2200 verbs.

Third, for each textual item, the number of personal verbs was tabulated. This statistic was used as an animacy score for the text item.

Fourth, the output from other retrieval runs was then reranked using the animacy score as a key. The reranking was done in one pass through the list. If an item has a low animacy score, operationalized as less than 75 per cent of the average animacy score of any text in the retrieved set, and the item just below it has a higher animacy score, their position is swapped. This method avoids large scale movement of items through the ranked list but shifts adjacent items from position to position.

han	he	barn	child
hon	she	ungdom	youth
man	man / impersonal “one”	pojke	boy
kvinna	woman	flicka	girl

Table 1: Seed word set for the prototypical animate agents

Unfortunately, we could see no effect of the reranking in our results, and are currently investigating why this was the case.

3.2 Dimensionality Reduction by Random Indexing

Dimensionality reduction is often important in information retrieval tasks, since the dimensionality of the search space induces constraints on the performance of the retrieval engine; very high-dimensional data will require large amounts of memory and processing time, and will severely limit the efficiency of the system. This is especially important in real-world settings, where the user expects both accurate and, sometimes even more important, *fast* results. A common approach to reduce the dimensionality of the data in information retrieval systems is by using various forms of word filtering techniques, such as stop lists, frequency thresholding and morphological normalization.

An alternative method for dimensionality reduction in the Vector Space Model (VSM) [6] is to combine word filtering with the use of reduced representations for the vocabulary. Assuming the standard definition of the VSM, where the dimensionality of the document vectors is given by the size of the vocabulary, i.e. the number w of unique words in the data (normally after word filtering), we can define a reduced representation as vectors of dimensionality $d \ll w$. One way of producing such reduced representations is to use a random mapping method [4], where words are represented by *nearly* orthogonal random vectors of dimensionality $d \ll w$, and where document and query vectors are defined as the average (i.e. the vector sum) of the vectors of the words in the document or query. The point of this methodology is that the resulting search space will be significantly smaller than the original search space, while still containing approximately the same information.

In one of our CLEF 2003 runs (`sicsSVind`), we used the Random Indexing approach [2], [3] to assign nearly orthogonal sparse random vectors to each unique word in the data. The vectors, which we call *index vectors*, were 1,000-dimensional with 6 randomly distributed non-zero elements (three +1s and three −1s). We then produced 1,000-dimensional document and query vectors by simply summing the index vectors of the words in the documents and the queries (after aggressive word filtering[2]). The resulting 1,000-dimensional document and query vectors are much smaller than the standard VSM vectors that will be 121,545-dimensional for the Swedish data (after word filtering).

The retrieval was then performed by simply calculating the vector similarity between each query vector and all the document vectors. The documents with highest similarity score were ranked as most relevant to the query. As similarity measure, we used the cosine of the angles between the vectors, given by:

$$d_{cos}(x,y) = \frac{\vec{x} \cdot \vec{y}}{\mid \vec{x} \mid\mid \vec{y} \mid} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

The results are somewhat disappointing; only 2 queries are above the median results, 3 are on the median, and 48 below. It is not clear at this point whether the results are an artefact of the dimensionality reduction, or if they depend on the simple and naive query formulation process used in these runs. Future experiments will investigate this matter more fully.

4 Results

A summary of the results of the runs using the Boolean combination queries is displayed in Table 2. The Table shows the number of queries that were above, on, or below the median result as well as the number queries that obtained the maximum or minimum score.

Run	Above	On	Below	Max	Min
`sicsSVtad`	14	10	29	6	5
`sicsSVtmd`	12	10	31	6	2

Table 2: Number of queries above, on or below the median score for each run. The number of queries with max or min score is displayed in the last two columns

The overall results are encouraging but there are also many missed queries. Since we use a maximum of 5 search terms for each query, the misses are easily attributed to the small amount of query terms. However, we believe that it is interesting to evaluate what type of results that can be achieved when using few query terms, since it is well known that many users (especially search engine users) typically use only three or four words to express their query.

There is an interesting difference between the two runs; the number of queries above the median result and the number of queries that obtained the lowest score. Recall that in `sicsSVtad`, only terms from the Title field were used. When we added some terms from the Desc field in `sicsSVtmd` we got fewer (3) queries that obtained lowest score but also fewer queries (2) above the median.

The fewer number of queries that got lowest score is due to the fact that we added more search terms to the query in the `sicsSVtmd`. For instance, the average precision of query 165 was improved from 0.3333 to 1.000. The query in `sicsSVtad` was "GOLDEN GLOBE" whereas in `sicsSVtmd` it was expanded to "GOLDEN GLOBE KATEGORI DRAMA FILM".

We noted that for the queries that were above median in `sicsSVtad` and then below median in `sicsSVtmd`, the difference was very small in terms of average precision. For instance, the average precision of query 147 was changed from 0.0673 to 0.0619, and the median was 0.0639.

[2]We used a stop list based on document frequencies together with ordinary word frequency thresholds (excluding low ($<$ 3 occurrences) and high ($>$ 12000 occurrences) frequency words).

5 Discussion

For a compounding language such as Swedish, it is important to find methods that can effectively manage compound words in information retrieval systems. The approach taken in this paper is to look at how the leading constituent of a compound word can be used to expand the query. The query terms were then truncated to increase recall. To strike a balance between high recall and high precision, we used a Boolean quorum-level type of combination where documents were ranked according to both the tf-idf factor but also according to how many of the Boolean combinations that matched.

The Boolean combinations performed relatively well, taken into consideration that the queries were very short. This was our first attempt to tackle the problem of using compound word splitting for query expansion, and we will continue to pursue this line of research. What we would like to do to next is to use co-occurrence statistics and perhaps also clustering methods to find words that are related to the compound, so that we can have a more principled way of checking the relation between the concept expressed by the constituent and that expressed by the compound.

Acknowledgements The work reported here is partially funded by the European Commission under contracts IST-2000-29452 (DUMAS) and IST-2000-25310 (CLARITY) which is hereby gratefully acknowledged. We thank Tidningarnas TelegrambyråAB, Stockholm, for providing us with the Swedish text collection.

References

[1] M. J. Folk, B. Zoellick, and G. Riccardi. *File Structures: An Object-Oriented Approach with C++*. Addison-Wesley, 3rd edition, 1998.

[2] P. Kanerva, J. Kristofersson, and A. Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, page 1036. Erlbaum, 2000.

[3] J. Karlgren and M. Sahlgren. From words to understanding. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, *Foundations of Real World Intelligence*, pages 294–308. CSLI publications, 2001.

[4] S. Kaski. Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *Proceedings of the IJCNN'98, International Joint Conference on Neural Networks*, pages 413–418. IEEE Service Center, 1998.

[5] M. Sahlgren, J. Karlgren, R. Cöster, and T. Jrvinen. Sics at clef 2002: Automatic query expansion using random indexing. In *The CLEF 2002 Workshop*, September 19-20 2002.

[6] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[7] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Research and Development in Information Retrieval*, pages 21–29, 1996.

[8] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann Publishing, 2nd edition, 1999.