Second ACL Workshop on Multiword Expressions: Integrating Processing, July 2004, pp. 40-47

Multiword Expression Filtering for Building Knowledge Maps

Shailaja Venkatsubramanyan
San Jose State University
Department of MIS
One Washington Square
San Jose, California, 95192-0244
USA
shailaja@acm.org

Jose Perez-Carballo
California State University, Los Angeles
Dept. of Information Systems
5151 State University Drive
Los Angeles, CA 90032
USA
perez-carballo@acm.org

Abstract

This paper describes an algorithm that can be used to improve the quality of multiword expressions extracted from documents. We measure multiword expression quality by the "usefulness" of a multiword expression in helping ontologists build knowledge maps that allow users to search a large document corpus. Our stopword based algorithm takes n-grams extracted from documents, and cleans them up to make them more suitable for building knowledge maps. Running our algorithm on large corpora of documents has shown that it helps to increase the percentage of useful terms from 40% to 70% – with an eight-fold improvement observed in some cases.

1 Introduction

Many real world applications require extraction of word sequences or multiword expressions from text. Examples of such applications include, among others, creation of search engine indexes, knowledge discovery, data mining, machine translation, summarization and term suggestion for either knowledge engineering or query refinement by end users of a search system.

The application of interest to the authors of this paper was that of building knowledge maps that help bridge the gap between searchers and documents. A *knowledge map* for a particular domain is a collection of concepts, relationships between these concepts as well as evidence associated to each concept. A domain *concept* represents an abstraction that can be generalized from instances in the domain. It can be a person, a thing, or an event. An example of a concept in the operating system domain is 'installation guidelines'. *Relationships* between concepts can be either generalization or specialization (such as "is a") as well as different types of association (such as "part-of"). The *evidence* associated to a concept is a set of single or multiword terms such that if any of those terms are found in a document, then it is likely that the document refers to that concept.

The task we were trying to support was to identify multiword expressions in a corpus of documents belonging to a domain that can help ontologists identify the important concepts in the domain as well as their evidence.

Our research was focused on domains where the corpus of documents representing the domain contains a high degree of technical content. The reason for this is that such documents are served on many company web sites to help provide technical support for both employees and customers.

Our research assumes that a term[1] is "useful" when it meets all of the following conditions – (1) it makes sense in context of the domain, (2) it represents an action, some tangible or intangible object, name of a product, or a troubleshooting phrase, and (3) it would be chosen by an ontologist to be incorporated to their knowledge map.

Some examples of multiword expressions that may be considered useful for building knowledge maps about technical content are "how to

[1] In common parlance, the words "term" and "expression" are generally used interchangeably. In this paper, a term refers to a expression with one or more words.

uninstall the program", "Simple Mail Transfer Protocol", and "cannot reboot the computer". Some expressions may not seem useful at first glance but may make sense to an ontologist familiar with that domain. For instance, the occurrence of the number "error 37582" may be an error code, and hence evidence of a particular kind of problem. Similarly, expressions such as "after rebooting the system" may not seem useful but may be good evidence of concepts related to problem identification. Examples of expressions that may be acceptable for some purposes, but not for building knowledge maps are "this software was implemented by" and "and reboot the system to". These expressions however can become useful after undergoing some processing or manipulation by humans.
We extracted n-grams from documents using an algorithm proposed by Tseng [1998], and cleaned them up iteratively using a stopword-based algorithm in order to make them more useful for building knowledge maps. Tseng's algorithm is based on the assumption that documents concentrating on a topic tend to mention a set of words in a specific sequence repeatedly. In other words, repeated multiword expressions are extracted since they will make good evidence candidates.

Our experience with Tseng's algorithm was that it extracts many useful multiword expressions. But, it also extracts multiword expressions that are repeated frequently in the documents but are not useful when viewed independently of the sentences from where they originated. This may not matter for some applications, but puts a lot of burden on librarians or ontologists who want to use those multiword expressions to build knowledge maps. Examples of such expressions are "software was", "computer crashed after", "installed in order to", and so on. Such expressions have to undergo further manipulation or processing by ontologists in order for them to be useful. A good weighting algorithm may eliminate some of these expressions in some cases. However, our experience has shown that in a sufficiently large and homogenous set of documents, occurrence of all of these variations is so high that many of them meet the threshold requirements to qualify as eligible multiword expressions. Setting higher frequency thresholds may be a solution to this problem, but that may result in elimination of other useful multiword expressions.

One of the steps usually undertaken is to eliminate not so useful single word terms extracted from documents. For instance, the word "the" is not considered to be useful for most purposes. If a user were to submit a query such as "the cat", returning documents that contain "cat" would be more useful than looking for documents that contain both "the" and "cat." Terms such as "a", "an", "the" and so on are referred to as "stopwords" or "noise words" or "skipwords", and these are usually ignored by search engines when they occur by themselves when building indexes. There are many common stopword lists useful in various contexts.

Statistical and quantitative techniques using frequency or mutual information scores for multiword expressions as well as syntactic techniques using phrase trees have been used to extract multiword expressions from text. [Choueka 1988, Dias 1999, Lessard 1991, Hamza 2003, Merkel 1994, Paynter 2000] According to Dias et. al. [1999], many multiword units identified using statistical methods can not be considered as terms although it may be useful to identify them. Examples cited by the authors include terms such as "be notified" and "valid for". Less commonly found in literature is work done to "clean" or "filter" the extracted multiword expressions to make them suitable for certain purposes. An example of implementation of a filter is found in work done by Merkel et al. in their FRASSE system [Merkel 2000] where they defined words that should be stripped at the beginning and at the end of multiword expressions as well as requirements on what kinds of characters should be regarded as pairs (quotation marks, parentheses, etc). The reason for identifying characters that should be regarded in pairs is to make sure that multiword expressions that are retained after filtering do not have only one parenthesis character or quotation mark. Their filter was implemented with the use of entropy thresholds and stopwords for the Swedish language. Another example of a proposed filter is found in work by Dias et. al. [1999] in which the authors suggest using a filter that removed stopwords where they occurred either at the beginning or at the end of multiword expressions. Our work uses a standard stopword list used by systems that suggest terms to ontologists and end users, and part of speech information to achieve the same goal. The part of speech information ensures that we treat beginning and end of multilword expressions differently.

Our contribution has been to extend Tseng's algorithm using stopwords and a part of speech based algorithm to reduce the occurrence of expressions that need further processing by ontologists. Our goal was to increase the proportion of expressions extracted that don't have to undergo any more manual processing by ontologists to make them useful. This is very useful in situations such as term suggestion where users can be saved the time and effort involved in going through long lists of terms many of which may not be useful, or may have to be manipulated in some way to make them useful. Running our algorithm on large corpora of documents has shown that it helps to increase the percentage of useful terms from 40% (+-10) to 70% (+-10). In other words, the improvement is at least 20% and could be high as 160%.

The rest of this paper is organized as follows – Section 2 describes our algorithm for extraction of frequently occurring n-grams, and converting them to useful multiword expressions. Sections 3 and 4 describe the results of evaluating our algorithm on large corpora of documents and conclude the paper.

2 Term Extraction and Filtering Algorithms

Researchers have extracted keywords by a simple recognition of fixed patterns that occur frequently in the text. [Choueka 1988, Tseng 1998] We adopted Tseng's algorithm which identifies frequently repeated N-grams because of its efficiency.
We begin by describing Tseng's algorithm and then discuss our modifications to extract useful multiword expressions.

2.1 Tseng's Algorithm

This algorithm consists of three major steps. The first step requires only linear-time complexity for converting the input text into a list. The second step repeats merging tests until no elements remained to be merged. The third step involves filtering out noisy terms using a stop list.

Table 1: Tseng's algorithm [Tseng 1998]

```
1. Convert the input text into a LIST of overlapping 2-grams (or 2-words, see an example below).
2. WHILE LIST is not empty
        2.1. Set MergeList to empty.
        2.2. Push a special token to the end of LIST as a sentinel.
        2.3. FOR each pair of adjacent elements K1 and K2 in LIST,
            IF K1 and K2 are mergeable and both of their occurring frequency are greater than a threshold
            THEN
                Merge K1 and K2 into K and push K into MergeList.
                Accumulate the occurring frequency of K.
            ELSEIF the occurring frequency of K1 is greater than a threshold and K1 did not merge with the element before it in LIST
                Push K1 into the FinalList.
            ENDIF
        ENDFOR
ENDWHILE
2.4. Set LIST to MergeList.
3. Filter out noisy terms in FinalList and sort the result according to some criteria.
```

2.2 Iterative Term Filtering Algorithm

We used Tseng's algorithm to select a set of multiword expressions. Then, we applied an algorithm based on stopwords and part of speech information to filter out the less useful words from the beginning and end of multiword expressions in order to help identify and construct useful multiword expressions. Our algorithm deletes words from the beginning and end of the multiword expression until one of the following conditions are satisfied: (1) the result is a one word term that is not a stopword, or (2) the words at the beginning and end of the multiword expression are deemed acceptable by our algorithm or (3) all words in the expression are deleted.

Our technique uses the stopword list that is used by step 3 of Tseng's algorithm along with rules regarding which of those stopwords are acceptable if found at the beginning and at the end of the extracted n-grams. Stopword lists may be generated by analyzing a corpus of documents, and identifying the most frequently occurring words across the corpus. Many pre-compiled lists are also available, and generally the practice has been to adopt a pre-compiled list and tweak it to meet a specific purpose.

The number of entries in stopword lists can range from approximately 50 to 5000 words. ["Onix" online stopword lists, "freeWAIS-sf", "Seattle

City Clerk Information Services Database", "RDS"] The stopword lists that are used for term suggestion tend to be longer or more aggressive than those used for building standard search engine indexes. The stopword list used by the term suggestion system that we used for our experimentation contained around 600 words. FastNPE, a noun phrase extractor, uses a stopword list with more than 3500 words [Bennett, 1999]. One would find words such as "can", "cannot", should", "approximately", and so on a stopword list for term suggestion even though they may not be present in other stopword lists. The reason for this is that such words are not useful by themselves to end users or to ontologists who are trying to understand the content of documents. But, these words may be useful when found at the beginning or end of multiword expressions. Our algorithm assumes the use of a stopword list generally used by implementers building term suggestion systems. Our part of speech based heuristics determine which of those stopwords would be considered acceptable at the beginning or end of multiword expressions.

2.3 Stopword Analysis

In order to gain an initial insight into what kinds of words are acceptable at the beginning and end of multiword expressions, we built lists of acceptable multiword expressions based on filtering performed by a team of experienced ontologists on a sample set of multiword expressions extracted from hundreds of thousands of documents. Studying words in the multiword expressions that were discarded or retained gave us clues about what words are acceptable at the beginning and end of multiword expressions. This helped us identify patterns that could then be incorporated into a more general algorithm.

Most of the time, stopwords were not useful when they were at the beginning or end of multiword expressions. Some good examples of stopwords that are not useful at the beginning or end or a multiword expression are coordinating conjunctions such as "and", "or", and so on. However, there are exceptions to this rule. The exceptions are described in sections 2.3.1 and 2.3.2.

2.3.1. Stopwords Acceptable at the Beginning of Multiword Expressions

We studied words retained and discarded by ontologists at the beginning of multiword expressions. As expected, many times, these were words that were in the stopword lists. But, there were cases where some stopwords were not discarded. These helped us identify cases or patterns that went into the creation of our algorithm. Those cases are presented below:

Prepositions – Words such as "under", "over", "after", and so on. An example of an expression that has a preposition at the beginning, and is a useful expression is "after installing the operating system". Using the standard stopwords list to eliminate words from the beginning of multiword expressions would have resulted in that expression being reduced to "installing the operating system". The meaning of "after installing the operating system" is quite different from "installing the operating system". The content of documents containing the expression "after installing the operating system" may be quite different from documents containing just the expression "installing the operating system". "After installing the operating system" may be indicative of a document about problems users may run into after installation. Just "installing the operating system" may be indicative of documents about how to install an operating system. The goal here is not to determine whether one multiword expression is really different from another, but to provide the ontologist with all possible information to make those judgment calls.

Auxiliary verbs – Words such as "can", "cannot", "will", "won't", "was", "has", "been" are examples of auxiliary or helping verbs. For example, the expression "can uninstall the program" is quite different from "cannot uninstall the program". Since "can" is both a noun as well as an auxiliary verb, it is usually not on most stopword lists. But, "cannot" is sometimes found in some stopword lists.

Adverbs - Words such as "slowly", "insufficiently", "fast", "late", early", etc. may be found in stopword lists used for term suggestion since these words do not carry much meaning by themselves. But, they are useful when found at the beginning of multiword expressions. Examples of such expressions include "early binding" and "late binding".

Adjectives – Adjectives such as "slow", "fast", "empty", "full", and intermittent" are useful when found in the beginning of multiword expressions. Examples include "slow CPU", "intermittent failures", etc. "All," "any," "each," "few," "many," "nobody," "none," "one," "several," and "some," are some examples of indefinite adjectives. Multiword expressions such as "several users", and "all CDROM drives" may convey more meaning than just "users" and "CDROM drives".

Interrogative pronouns – "How", "why", "when", and so on are not useful by themselves, but are very useful when found at the beginning of multiword expressions. Examples of such expressions include – "how to install an anti-virus package", "when to look for updates", and "how do I fix my computer".

Correlative conjunctions – "Both the computers", and "either freezing or crashing" are examples of expressions that begin with correlative conjunctions. "Both" and "either" are very likely to be found in stopword lists used for term suggestion, but they add meaning to multiword expressions.

2.3.2. Stopwords Acceptable at the End of Multiword Expressions

Similarly we studied words retained and discarded by ontologists at the end of multiword expressions. As expected, many times, these were words that were in the stopword lists. But, there were cases where some stopwords were not discarded. Those cases are presented below:

Numbers – Numbers are generally found on most stopword lists. 0, 1, 2, and so on rarely make sense by themselves, especially in the context of term suggestion. However, when they are found at the end of the multiword expressions in the digit form (0, 1, 2, and so on) rather than in the word form (one, two, three, and so on), they can be useful. Examples of such cases are usually product names with their version numbers – "Microsoft Word version 3.0", "Windows 3.1", and so on.

Closing parentheses – Closing parentheses usually indicates the presence of opening parentheses within the multiword expression. Therefore, retaining the closing parentheses is a good idea. Examples of such expressions are "Manufacturing (UK division)", "Transmission Control Protocol (TCP)", and so on. A nice side effect of this heuristic is the ability for the users to learn about acronyms in the domain.

Adverbs – Words such as "slowly", "quickly", "immediately", and so on are useful at the end of multiword expressions. Examples of these include "computer shuts down slowly", and "uninstall the program immediately".

Table 2 describes Tseng's algorithm modified using our term filtering algorithm

Table 2: Modified Term Filtering Algorithm

```
1. Convert the input text into a LIST of overlapping 2-grams (or 2-words, see an example below).
2. WHILE LIST is not empty
        2.1. Set MergeList to empty.
        2.2. Push a special token to the end of LIST as a sentinel.
        2.3. FOR each pair of adjacent elements KI and K2 in LIST,
            IF Kl and K2 are mergeable and both of their occurring frequency are greater than a threshold
            THEN
                Merge KI and K2 into K and push K into MergeList.
                Accumulate the occurring frequency of K.
            ELSEIF the occurring frequency of KI is greater than a threshold and KI did not merge with the element before it in LIST
                Push KI into the FinalList.
            ENDIF
        ENDFOR
ENDWHILE
2.4. Set LIST to MergeList.
3. Filter out noisy terms in FinalList and sort the result according to some criteria.
4. FOR each expression FL on the FinalList,
        4.1 IF the first word in FL is a stopword and is not: a preposition, an auxiliary verb, an adverb, an adjective, an interrogative pronoun, or a correlative conjunctions
          THEN
            Delete that word
        ENDIF
        4.2 IF the last word in FL is a stopword and
is not: an adverb, a closing parenthesis, or a number,
        THEN
            Delete that word
        ENDIF
UNTIL (the words at the beginning and end of FL are
not on the stopword list OR FL is a one word term
that is a not a stopword or all the words in the expres-
sion are deleted by this algorithm)
5. Push FL into FilteredList.
```

This algorithm can be implemented using either a program that does part of speech tagging or a program that looks up a thesaurus. Our implementation used a list of stopwords that are acceptable at the beginning of the expression, and another list of stopwords that are acceptable at the end of the expression.

3 Evaluation

We implemented this algorithm using Java, and ran it on more than 20 corpora of documents dealing with technology topics. The size of the corpora ranged from 4000 documents to 500,000 documents. The average size of the corpora was around 5-6 MB. The topics discussed include, among others, computer networking, instructions on how to install and use application software, troubleshooting software problems, and so on. Program inputs include documents, and a stopwords list.

Benefits of applying our algorithm to filter expressions include:

Term list size reduction - The result of applying our algorithm to filter expressions extracted from documents is a reduction in number of terms by at least 30%-40%. This translated to an order-of-magnitude reduction in time and effort on the part of ontologists and other users. Without the algorithm, ontologists may have had to study the list manually to eliminate meaningless expressions and manipulate other terms to turn them into useful expressions. Examples of such reduction include:

Expressions such as “Windows 98 operating system”, “Windows 98 operating system was”, “the Windows 98 operating system”, and “Windows 98 operating system is” are reduced to “Windows 98 operating system”.

Expressions such as “the screen flickers”, and “screen flickers and” would be reduced to just “screen flickers”.

Expressions such as “and is a” and “is not” and “and etc.” are eliminated from the list. The individual words in these expressions are in the stop words list, but ordinarily a multiword expression such as “is not” would make it past the stop words filter since it contains more than one word in it.

The reduction in the number of terms translated to a reduction in the number of person-weeks required to create a knowledge map using the terms. We noticed a savings in the number of person-weeks that ranged from 50% to close to 90%. In one particular instance, using our algorithm reduced the time required to create a knowledge map based on extracted n-grams from 4 person-weeks to about 0.5 person /weeks

Higher precision - There is a greater probability that the terms that remain after filtering are useful terms. In other words, the remaining terms are more likely to be considered useful by users. Our experience has shown that the percentage of useful terms prior to filtering ranged from 30% to 50%. Post filtering, the percentage of useful terms ranged from 60% to 80%. In other words, running our algorithm on large corpora of documents has shown that it helps to increase the percentage of useful terms from 40% (±10%) to 70% (±10%) – with an eight-fold improvement observed in some cases.

Domain independence - Pattern extraction from documents involves extracting both domain specific and domain independent terms. Domain specific terms are those that represent the core knowledge in the domain. For example, terms such as “dynamic host control protocol” and “TCP/IP” can be considered to be domain specific terms in the computer networking domain. On the other hand, terms such as “document author” are not domain specific. The technique described in this paper aids in filtering both domain specific and domain independent terms extracted from documents. This ensures domain portability. The tests conducted have been primarily with documents containing technology topics. However, this algorithm worked well with documents related to electronic commerce as well.

The algorithm is, of course, not foolproof, and there are instances where expressions that ought to be modified are not, and expressions are modified more than necessary. For instance, the expression “software was” will be correctly reduced to “software” since “was” is an auxiliary verb. The multiword expression “computer crashed after” will be reduced to “computer crashed” since “after” is a prepositon, but “installed in order to” will be reduced to “installed in order”. “Installed in order” is not a useful expression, but it is one of the expressions that are not processed correctly by our algorithm. On the whole, however, our finding is that applying this algorithm results in a significant savings of time and effort to extract useful multiword expressions from documents.

4 Conclusion and Future Work

We believe that our approach can help tremendously with the task of filtering expressions extracted automatically from documents. The result of applying our approach will be automatic extraction of more useful expressions, and reduction of burden on users who are presented with those expressions.

Future work includes using more sophisticated statistics such as IDF other than just frequency of occurrence of terms to eliminate more terms before they are processed by the multiword term filtering algorithm. Our initial approach was to do something fast and simple that has a significant impact. Our plan is to evaluate various statistical approaches in order to select one that can produce better multiword expressions that can then be fed into the term filtering algorithm. An approach that we experimented with was running the algorithm on just the titles and abstracts of larger documents. We noticed that this approach worked well for extracting concepts for building knowledge maps. However, it needs to undergo further testing. Besides, testing this algorithm on documents from other domains such as medical, pharmaceutical and financial domains, and using syntactic and semantic information to build “positive filters” that identify well formed patterns, instead of stripping away ill-formed patterns are other issues worth researching.

References

Bennett Nuala A., He Qin, Chang, Conrad T. K., and Schatz, Bruce R. Concept Extraction in the Interspace Prototype, UIUCDCS-R-99-2095, April 1999

Choueka, Y. (1988) “Looking for needles in a haystack”. In RIAO 88, User-oriented Content-based Text and Image Handling, Volume 1, 609-623, 1988.

Dias, G, Vintar, Pereira Lopes, G.; Guillore, S. Normalising the IJS-ELAN Slovene-English Parallel Corpus for the Extraction of Multilingual Terminology. In: Monachesi, P. (ed.) Proceedings of the CLIN '99 (Computational Linguistics in the Netherlands).

freeWAIS-sf stopword list at http://www-fog.bio.unipd.it/waishelp/stoplist.html

Hamza, H., Mahdy, A. Fayad, M. E. and Cline, M. Extracting Domain-Specific and Domain-Neutral Patterns Using Software Stability Concepts, OOIS 2003, Geneva, Switzerland, September 2003

Lessard, G., Hamm, Jean-Jacques. (1991) Computer-Aided Analysis of Repeated Structures: the Case of Stendhal's Armance. ACH/ALLC 91, Tempe.

Merkel, Magnus & Andersson, Mikael (2000). Knowledge-lite extraction of multiword units with language filters and entropy thresholds. In Proceedings of RIAO'2000, Collége de France, Paris, France, April 12-14, 2000, Volume 1, pp. 737-746.

Merkel, Magnus, Nilsson, Bernt & Ahrenberg, Lars (1994). A Phrase-Retrieval System Based on Recurrence. In *Proceedings from the Second Annual Workshop on Very Large Corpora*. Kyoto.

Onix Text Retrieval Toolkit stopword list #1 at http://www.lextek.com/manuals/onix/stopwords1.html

Onix Text Retrieval Toolkit stopword list #2 at http://www.lextek.com/manuals/onix/stopwords2.html

Paynter, Gordon W., Witten, Ian H., Cunningham, Sally Jo, Buchanan, George. Scalable browsing for large collections: a case study. June 2000. Proceedings of the fifth ACM conference on Digital libraries.

RDS Business Reference Suite stopword list at http://rdsweb2.rdsinc.com/help/stopword_list.html

Seattle City Clerk's Office Information Services Database Stopword List at http://clerk.ci.seattle.wa.us/~public/stop.htm

Tseng, Yuen-Hsien. August 1998 Multilingual keyword extraction for term suggestion. Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval