Technology Review

Shai Yusov

Keywords provide a compact representation of a document's essential content. Therefore, they provide utility to a variety of applications in text retrieval and analysis. This paper compares three approaches to keyword extraction (TF-IDF, RAKE, and TextRank) in terms of their characteristics, efficiency, and ease of use, and surveys free software that can be used to implement them.

The first approach to keyword extraction depends on TF-IDF. TF-IDF stands for Term Frequency - Inverse Document Frequency and is a way to transform a term according to its frequency in the document as well as among all documents in a collection to achieve a better measure of its importance. This approach involves removing stop words, cleaning and normalizing the text, stemming and lemmatizing the tokens, choosing candidate n-grams based on their part of speech tag pattern, finding the term frequency for each n-gram, and ranking the n-grams according to their TF-IDF score. The top ranked n-grams constitute the extracted keywords.

The TF-IDF approach draws on the collection of documents for context. Hence, it requires a significant enough sample of documents to produce appropriate results. It does not consider the context around the term in the document and will not naturally capture phrases of multiple words. However, it is designed to find terms that are important in a document and filter out common words that are used everywhere as part of natural language.

This approach has a reasonably efficient implementation but it is implemented manually in multiple steps and thus requires some development effort. For general natural language tasks such as removing stop words, part of speech tagging, stemming, lemmatization, and generating n-grams, the following toolkits can be used: *nltk* (python), *spaCy* (python), *Apache OpenNLP* (Java), *Stanford CoreNLP* (Java), and *Lemur* (Java, C++, and C#). And TF-IDF computation is supported in *Scikit-Learn* (python) and *Lemur* (Java, C++, C#). Overall, this approach is reasonably performant and requires more development effort compared to an off-the-shelf solution, but available libraries facilitate implementation and come with useful features and extensions that can be used to customize it.

The second approach to keyword extraction is Rapid Automatic Keyword Extraction (RAKE). Rake was designed with the assumption that keywords tend to contain multiple words but do not typically contain punctuation and stop words, such as *and*, *is*, and *for*, because they are not considered meaningful. First, it splits text into an array of words at word delimiters, and then it splits the array into sequences of contiguous words at stop words and phase delimiters. Each sequence is a candidate keyword. Word co-occurences are represented in a graph where every word contains an undirected edge to every word with which it shares a candidate keyword (including itself). The score of a word is defined as the ratio of the degree of the word's node in the co-occurences graph to the frequency of the word in the document. And the score of a candidate keyword is defined as the sum of scores of its words. Thus, the word score metric favors words that occur largely in long candidate keywords and disfavors words that occur too frequently and in shorter candidate keywords. Finally, a portion (typically a third) of the top scoring candidate keywords are selected as keywords for the document

Rake does not require a collection of documents for context. On the one hand, it does not naturally differentiate between words that are relevant specifically for one document and words that are found among many documents in a collection (although there are extensions that address that). On the other hand, it easily scales to extremely large collections and real-time processing use cases. And since Rake is also simple and efficient - it can be implemented in one pass over the document, it is suitable for use in a wide variety of applications. Finally, Rake naturally considers phrases with multiple words. That can both aid in detecting keywords with multiple words as well as lead to unnecessarily including meaningless words in phrases.

Rake can be used in multiple programming languages with off-the-shelf implementations. In python, there is *python-rake*, which is the original implementation in native python as described in the original paper and is available through *pip; rake-nltk*, which is implemented using *nltk* and is available through *pip;* and more. In Java, there is *RAKE-Java*, which is an implementation of the algorithm as described in the original paper and is available through maven; *rapidrake*, which is a fast implementation of Rake and is available through maven; and more. Rake is also available in other languages through various implementations. Overall, Rake is efficient and can be used with an off-the-shelf dependency.

The third approach to keyword extraction is TextRank. It is a graph based algorithm. First, the document is split into tokens, which are then annotated with part of speech tags and filtered. Next, each token is added as a vertex in the graph, and an undirected edge is drawn between tokens that co-occur within a window of size N words. Then, each vertex is initialized with a value of 1, and an algorithm derived from PageRank is run for several iterations until it converges. Finally, vertices are sorted by their score and a portion (usually a third) of the top scores are selected as the keywords.

TextRank is unsupervised and does not require information other than the text. It is slower in execution compared to TF-IDF and Rake due to its repeated iterations to achieve convergence. In fact, as the number of words in the text increases, the difference in execution time only widens. TextRank does not naturally consider keyword candidates with multiple words; instead, it constructs multi-word keywords in the post-processing phase by collapsing sequences of adjacent keywords into one keyword. Thus, the range of potential multi-word keywords is not comprehensive and some possibilities may not even be considered.

TextRank can be used in multiple programming languages with off-the-shelf implementations. In python, there is *pytextrank*, which is the implementation as described in the original paper with some improvements and is available through *pip; summa*, which is a TextRank implementation with optimizations on the similarity function and is available via *pip; textrank,* which is a TextRank implementation as described in the original paper, only with Levenshtein Distance as the relation between text units; and more. In Java, there is a full source implementation of TextRank in *Apache OpenNLP Sandbox*, which is not released but is available in Apache's *opennlp-sandbox* repository on Github, as well as other full source implementations. Overall, TextRank is slower than the other approaches and can be used with an off-the-shelf dependency.

In conclusion, this paper compared three approaches to keyword extraction; (1) TF-IDF; (2) RAKE; and (3) TextRank. It described each approach and discussed its characteristics, efficiency, and ease of use. Based on this comparison, it is clear that there are differences in the characteristics, efficiency, and ease of use of these approaches. It additionally surveyed free software that can be used to implement these approaches and provided a pragmatic overview of their possible utilization.

References

Mihalcea, R. and Tarau, P. (2004) Textrank: Bringing order into text. In *Proceedings of EMNLP 2004* (ed. Lin D and Wu D), pp. 404-411. Association for Computational Linguistics, Barcelona, Spain. URL: http://aclweb.org/anthology/W04-3252.

Rose, S., Engel, D., Cramer, N., and Coley, W. (2010) Automatic Keyword Extraction from individual documents. In Berrry, M. W. and Kogan, J. (Eds.), *Text Mining: Applications and Theory, pp. 3-19.* John Wiley & Sons. URL: https://www.doi.org/10.1002/9780470689646.