KeyPhrase Extraction with Lexical Chains

Gönenç Ercan
Computer Engineering Dept.
Bilkent University, Ankara, Turkey
ercangu@cs.bilkent.edu.tr

ABSTRACT

Keyphrases have various usages, including indexing, summarizing, labeling, classifying, categorizing, highlighting, browsing and searching. However most of the text available today does not have keyphrases. Since it is a time consuming task to manually assign keyphrases for documents it is more than desirable to automatically extract keyphrases from text documents. Lexical chains have been extensively used for text summarization, however for keyphrase extraction this feature has not been fully investigated. In this paper the effect of using lexical chains as a feature and treating keyphrase extraction problem as a supervised learning algorithm has been issued.

Keywords: Keyphrase Extraction, Lexical Chains, Machine Learning

1. Introduction

Keyphrases can be considered as brief summaries of document's contents. Therefore it is possible to think of them as a set of keywords covering most of the text. With such short and concrete representation of the underlying text it is possible to use them in different applications. While keyphrases enable readers to decide whether or not a document is relevant for them, they can be used as low cost measure of similarity between documents. Even though there are numerous applications for keyphrases it is not likely to obtain keywords for most of the texts available today. Considering the fact that it is hard and time consuming to assign keyphrases to documents, it is desirable to automate this task by artificial intelligence and natural language processing (NLP) techniques.

In this paper the word keyphrases is used, to emphasize that keyphrases are not limited to single words. Keyphrase extraction tries to extract most indicative phrases from the document. So in other words keyphrase extraction is limited to phrases that appear in the document. However a more general problem called keyphrase generation tries to find keyphrases that may or may not be in the document. Obviously keyphrase generation is a harder problem in the sense that either machine understanding or categorization of the document is needed.

With the work of *[Turney 99]* and [*Kea]* keyphrase extraction problem has been treated as a supervised learning task. *[Turney 99]* describes a system called GenEx, which uses a Genetic Algorithm to learn the best parameters for the extractor algorithm to extract keyphrases. Parameter values for the extractor are the population and the fitness function is the precision of the extractor.

Turney also tested the performance of a general purpose decision tree algorithm C4.5 *(Quinlan).* Both of these algorithms are using frequency and first occurrence of phrases in the text as its features. While GenEx has performed better in terms of precision, training time for GenEx is very high. Current computational complexity of the algorithm is discouraging the efforts for integrating features that are more

expensive to obtain. Running time of GenEx makes it impractical to train GenEx with domain specific data. Fortunately Turney has shown that GenEx does generalize well to other domains.

Kea algorithm considers keyphrase extraction as a classification problem and uses Naïve Bayes algorithm. Experiments show that although GenEx is slightly better in general domains, Kea outperforms GenEx when they are both trained with domain specific training data. Computational complexity of Kea's training is much more lower and thus it is possible to train Kea for domain specific tasks. The features used in Kea are similar to features used in Turney's paper. Features used in Kea are frequency and position in text.

If we consider that keyphrases as a set of phrases that covers most of the text, keyphrases should have relationships with more words. In this paper usage of lexical chains are explored for keyphrase extraction. Lexical Chains are already used for text summarization. *[Barzilay and Elhadad]* describes a method for extracting sentences for summaries. A lexical chain is a set of semantically related words in a text. To identify the relationship between two words we can use WordNet lexical database. WordNet can be considered as a dictionary containing definitions of each word

Red : the color of blood or a ruby
Blood : the red liquid that circulates in the heart arteries and veins of animals.

The first thing to note about these definitions is that they reference each other. These references are proofs of some relation between these two words. There are different types of relations between words. Homonymy (same form completely different meaning), Polysemy (same form different meaning which is derived from the original meaning), Synonymy (different words same meanings), Hyponymy (similar meanings, generalization or specification of a concept e.g. Car and vehicle). In WordNet each word and its different senses are stored. The relations between words are stored in WordNet database. Since keyphrases should maximally cover the text, they are expected to be highly related to other words appearing in the document. The algorithm detailed in this paper is based on this idea.

2. Algorithm:

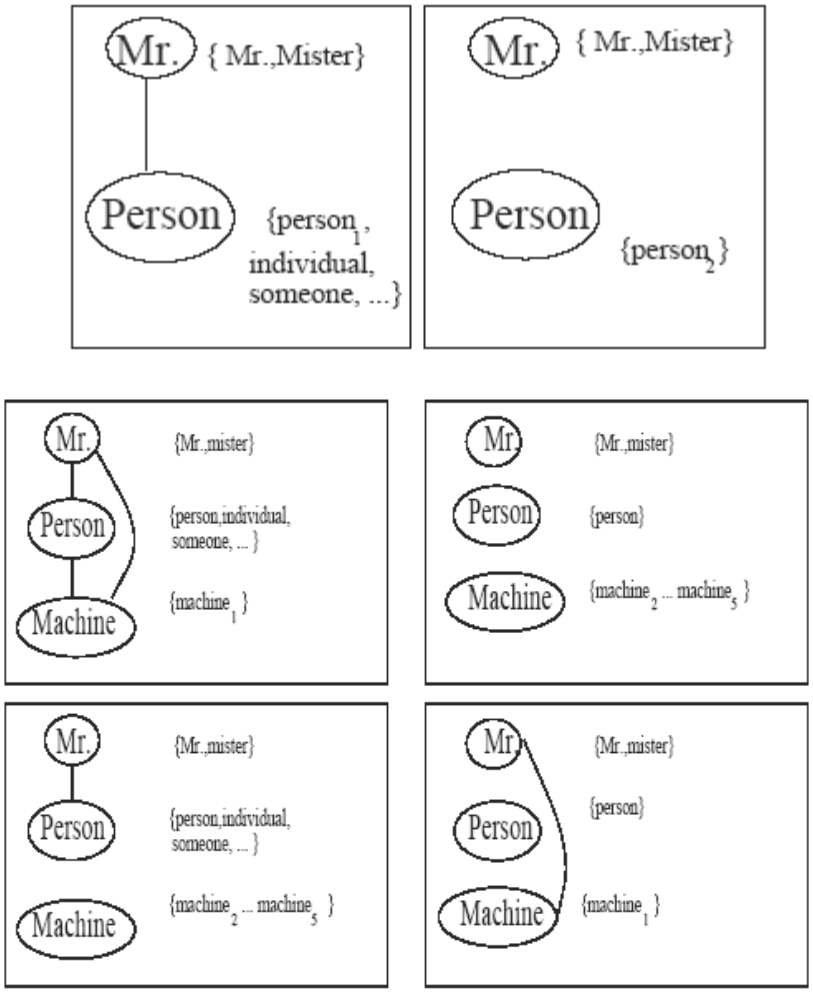
The main difference between previous work and this paper is probably the usage of a more sophisticated NLP based feature. Lexical chains can not be obtained with a surface analysis of the text. Thus the text is passed through morphological analysis and chains are constructed by discovering the semantic relation between words.

Document is cleaned from stop words. To clean the stop words a list of words like "this", "that", "the", "he", "she" is used. Lexical chains in the text are extracted first. To extract lexical chains, for each verb, noun and adjective word encountered, the word is looked up from WordNet and all the possible senses for the word is retrieved. If it is not in WordNet it is simply ignored. Since it is not possible to know the correct sense used in the document the algorithm must try to take into account all possible senses. The chains constructed have different interpretations depending on their sense. All senses of the word is checked with components. A component is an exclusive set of interpretations, where each word in a component has some relation with another word. If a word sense has a relation with another word sense in a component than that word is included in that component. After deciding that a word belongs to a component then all possible interpretations are computed. When number of interpretations is above some threshold the interpretations are pruned and only the best scoring interpretations are stored.

This algorithm with respect to its greedy alternatives is much more convenient for our purposes since it resolves word sense disambiguation problem. If a word sense does not fit into any component than a new component is created. Relations between words are limited by the distance between the words. The distance metric in this paper is number of words in between and it is set to 80 words.

Mr. *Kenny is the* **person** *that invented an anaesthetic* **machine** *which uses* **micro-computers** *to control the rate at which an anaesthetic is pumped into the blood. Such* **machines** *are nothing new. But his* **device** *uses two* **micro-computers** *to achieve much closer monitoring of the* **pump** *feeding the anaesthetic into the patient.*

Below is a simple example execution of this algorithm. Suppose the first word is defined as Mr. Mr. has only one sense. When person is the next word we encounter, since person has 2 senses (person –1 human being and person –2 grammatical category of pronouns and verb forms).



With this algorithm we don't have to decide until we have enough data to disambiguate the correct sense of each word. Scoring of chain sets are done by scoring the relations between the chain elements. Mr. and person in the above example has a relation in between. Lexical chain calculation algorithm is better explained in *[Barzilay and Ehladad] [Barzilay]*.

Along with lexical chains other features are combined. Frequency of a word and first occurrence position of a word is used for this task.

2.1. Frequency Feature:

The number of occurrence for each word is counted. Number of occurrences is normalized by the total number of words in the document. To calculate frequency we need to match two instances of the same base word with different affixes. For example we must match birds with bird. Therefore we need to find the stems of given words. The resources used for lexical chain computations can easily be used to stem given words. Stemming is done with a morphological analyzer. Value of this feature is between 1 and 0.

2.2. Position In text:

First occurrence position of the word in the text is stored. This feature is also normalized by the number of words in the document. Value of this feature is between 1 and 0.

2.3. Lexical Chain Score

The final lexical chains obtained will have set of words and the relations between these words. For each word it is possible to calculate the relation score. To calculate the relation score, each relation type is given a specific weight. For reoccurrence and synonym this weight is 10. For antonym (opposite) this is 7 and for hyponymy 4. The total relation score is stored for each word. Different from scoring lexical chains, all occurrences of the word in different components is considered.

2.4. Keyword Class

This is the class attribute in supervised learning. With the keyphrases we have it is possible to assign either positive or negative labels for each word. What is differs for this phase from other algorithms is that, instead of matching keyphrases for a document, keywords are matched. If a keyphrase is "neural network" than both of the words, "neural" and "network" are marked as positive examples.

3. Learning Algorithm

Since lexical chain calculation is an expensive process it is not possible to consider this feature for algorithms that have high running time. Therefore it is not practical to use lexical chain scores for GenEx algorithm. It is more reasonable to apply these features to more classical machine learning techniques like

Decision Trees or Naïve Bayes algorithm. In this paper C4.5 is used, however it is possible to use these features with Naïve Bayes algorithm.

3.1. Applying C4.5

Feature vectors for each word in the document is calculated. The general setting for C4.5 will be used. The learning algorithm is trained to classify if words appear in a documents keyphrases. Soft thresholds are used to obtain various sized word lists.

4. Post Processing

After obtaining a list of keywords from the decision tree algorithm we must extend these words to keyphrases. When a word is decided to appear in a keyphrase, every occurrence of the word is scanned for obtaining keyphrases. Candidate keyphrases for each word is extracted from the document. A candidate keyphrase should not end with stop words, with adjectives and should not be a proper name based on capitalization. The frequency of selected keyphrases are calculated. If a keyphrase for a word occurs more than 35% of the word than most significant keyphrase is selected as the keyphrase. With this technique it is possible to have stop words in Keyphrase where it was not possible in the other algorithms.

5. Corpus

To be able to compare the results obtained from this algorithm to Kea and GenEx, same testing and training data from journal articles is used. Journal papers are obtained from different journals which guarantee that the results will not overfit to a specific domain.

6. Results

Test results for the algorithm are not available yet it will be placed in later versions of this paper. However preliminary results obtained from lexical chains seems promising. The keywords have high frequency in lexical chains. With this algorithm it is hoped that quality of selecting keywords will be increased and being able to contain stop words in keyphrases will aid in matching keyphrases that was not even considered with other algorithms like keyphrases containing stop words.

7. Conclusion

This paper describes an alternative algorithm for keyphrase extraction. Most important aspect of the algorithm is that keyword selection is based on lexical chains, which potentially describes the word's semantic coverage of the whole document. As future work using these features with a naïve bayes algorithm could be beneficial. Extension from keywords to keyphrases is another interesting contribution to this problems solution.

REFERENCES

1. [Turney, 1999] P.D. Turney. Learning to extract keyphrases from text. Technical Report ERB-1057,National Research Council, Institute for InformationTechnology, 1999.
2.
3. [Quinlan, 1992] J.R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, Los Altos, CA,1992
4.
5. [Barzilay and Elhadad] Regina Barzilay and Michael Elhadad 1997. Using lexical chains for text summarization. In Proceedings of the Intelligent Scalable Text Summarization Workshop (ISTS'97), ACL, Madrid, Spain.
6. [Barzilay 1997] Barzilay, R. 1997. Lexical chains for summarization. MS Thesis
7. [Kea 99] Frank E., Paynter G.W., Witten I.H., Gutwin C. and Nevill-Manning C.G. (1999) "Domain-specific keyphrase extraction" *Proc. Sixteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 668-673.
8. [Kea 99] Witten I.H., Paynter G.W., Frank E., Gutwin C. and Nevill-Manning C.G. (1999) "KEA: Practical automatic keyphrase extraction." *Proc. DL '99*, pp. 254-256. (Poster presentation.)