CMPUT 497 Project Report: RAKE - Key Word Extraction Replication

Shouyang Zhou
University of Alberta
Edmonton, Alberta, Canada
shouyang@ualberta.ca

Sharon Hains
University of Alberta
Edmonton, Alberta, Canada
hains@ualberta.ca

Sharif Bakouny
University of Alberta
Edmonton, Alberta, Canada
albakoun@ualberta.ca

1 Introduction

We replicate the primary evaluation from "Automatic Keyword Extraction from Individual Documents" by Rose et al. (2010). This paper devises an unsupervised method for keyword extraction titled "RAKE" , the authors compare it to an contender unsupervised method called "TextRank" comparing performance on keyword extraction on a data-set of human labeled scientific abstracts. Both methods have similar approaches and are inspired by the same seminal work. RAKE constructs an adjacency matrix to score candidate keywords. TextRank constructs a weighted graph of keywords and scores them based on a recursive voting scheme treating edges as votes to/from candidate keywords.

Keyword extraction is the automated process of extracting important words and phrases from a document. The keyword extraction problem is generating a set of key terms from an input of natural unprocessed text. The importance of keyword extraction is in application, in information retrieval, feature engineering, and assist human labeling tasks. Keyword extraction helps humans process unstructured data in a more efficient and digestible manner. There is widespread discussion on the application of both methods on the web however few attempts to replicate the originating work. It is important to verify the performance of both methods given the amount of discussion.

Additionally we try to replicate a stop-list generation method Rose et al. (2010) suggest that complements RAKE. The input is a corpus and associated human annotated keywords. The output is a list of stop-words which RAKE uses to preprocess texts. This is considered in our replication. In essence, the authors have suggested a supervised complementary stop-list generation that augments the unsupervised method RAKE.

In summary, we cannot replicate Rose et al. (2010)'s salient conclusion, that RAKE using a domain specific generated stop-list, is superior to TextRank and baseline results. We acknowledge difficulties in replicating the stop-list generation method the authors suggest which is crucial to RAKE's functionality. We are able conclude that RAKE, using a generic stop-list, and TextRank perform similarly albeit RAKE has a slight improvement in performance. We replicate Table 1.2 in Rose et al. (2010), in our evaluation. This compares the performance of RAKE and TextRank variants (parameters sets) listing the metrics: extracted keywords (total, mean), correct keywords (total, mean), precision, recall, f-measure summarizing the replication-evaluation.

2 Related Work

2.1 TextRank

Mihalcea and Tarau (2004) propose TextRank, an unsupervised graph-based keyword extraction algorithm, based on PageRank. Their method precedes RAKE, they claim better performance compared to an older seminal work of Hulth (2003). This method generates a graph representing text units, (terms, phrases, or sentences) and generates an importance score per unit. The importance score of a vertex is decided by considering global relatedness of each text unit by recursively computing relations (edge weights) from the entire graph. Each text unit's importance is the sum of its baseline weight and a portion of its neighboring text units'. Intuitively, keywords are those whom recursively have the highest relatedness to other terms in the text, the top scoring units are keywords. TextRank is quite flexible and open to extension, the authors supply a default set of parameters. Implementing this algorithm goes as follows:

1. Select candidate text units and add them as vertices to a graph. (terms, phrases, or sentences) Filter out units if desired.
2. Generate edges of the graph via some relation between text units. (Authors suggest co-occurrence)
3. Compute a recursive importance score of a vertex as some innate value plus the sum of its descendants scores.
4. Iterate importance scoring until convergence using a damping factor to control updates.
5. Rank vertices based on their final scores and select top T vertices. (Authors suggest using 1/3 the size of the graph.)
6. Post-process keywords into key phrases by their adjacency in the text.

2.2 Hulth

Both RAKE and TextRank authors conduct their evaluations by extending the methods of Hulth (2003). Rose et al. (2010) noted difficulty in finding training materials used by Hulth. Hence we not be replicating the results of Hulth. RAKE, TextRank, and Hulth form a linage of keyword extraction methods, each successor claims performance over the others over the same data-set. Rose et al. (2010) describe Hulth's method, "Hulth (2003) compares the effectiveness of three term selection approaches: noun-phrase (NP) chunks, n-grams, and POS tags, with four discriminating features of these terms as inputs for automatic keyword extraction using a supervised machine-learning algorithm." Hulth provides the data-set all three authors use to evaluate their keyword extraction methods. This is a data-set of human keyword annotated scientific paper abstracts divided into a training, dev, and test sets. As RAKE and TextRank are unsupervised methods, ignoring stoplist generation for RAKE, both authors focus on using the test set.

3 Methodology

Automatic Keyword Extraction from Individual Documents by Rose et al. (2010) is the primary article for this replication. As mentioned, they evaluate their method RAKE and compare it with TextRank on human keyword annotated scientific abstracts. RAKE generates an adjacency matrix tracking co-occurrence of text, and define an importance score as the degree of a term divided by the frequency of the term. RAKE requires a list of stop-words. Stop words are punctuation, numbers, conjunctions, and user specified terms which are used to delimit candidate keywords. A brief summary of the RAKE algorithm:

1. Split the text into an array of words using the word delimiters.
2. Split the array into sequences contiguous words using stop words and phrase delimiters.
3. Candidate keywords are words in a sequence that are assigned the same position in the text.
4. Assign scores to each keyword candidate using ratio of degree to word-frequency.
5. Keywords that contain stop words:
   (a) A pair of candidate keywords must be adjoined at least twice in the text in the same order.
   (b) Create a new keyword which contains the pair of keywords with interior stop words between them.
   (c) The new keywords score is the sum of the scores of its keywords components.
6. The keywords of the text are the top T keywords from the keyword candidates list. (Authors suggest using a size of 1/3 of the adjacency matrix.)

To complement RAKE the authors also develop a method for stop-list generation they call "Keyword Adjacency" stop-list generation. While RAKE is an unsupervised method the authors suggest that if training data is available it can be beneficial to create a domain specific stop-list. A brief summary of the stop-list generation algorithm:

1. Given training keywords, find each keyword in the abstract.
2. Look for the adjacent tokens of each keyword. Count these words as 'adjacent words'.
3. Iterate through the adjacent words list.
4. For each abstract, find the count of each adjacency word, called the adjacency frequency.
5. For each keyword, find the count of each adjacency word, called the keyword frequency.
6. If the keyword frequency is higher than the adjacency frequency, remove this word from the adjacent words list.
7. All items remaining in the list is our stop-list.

4 Implementation

Since our project proposal, we found implementations of RAKE and TextRank. Since these are already available to us, we will replicate Rose et al.'s (2010) evaluation of the two using third party libraries. We will implement an evaluation script that feeds a data-set into these third party libraries to extract then aggregate the resultant metrics. This task involves data collection, reviewing the third party implementations, understanding the interface to the RAKE and TextRank implementations, pre-processing data-sets to be piped into the two methods, and analyzing the results.

The data from Hulth is sourced from Boudin (2017), this GitHub repo is hosted by an associate

professor at the University of Nantes researching NLP and information retrieval. The dataset required a python interface to reproduce the inital unprocessed text and human annotated keywords. The file structure was quite odd. The abstracts were stored in XML where each abstract is decomposed into individual tokens-tags groups where each token had been post-proceesed with its POS tag, its stem, and various other qualities. Some effort was required to reconstruct the intital text.

We conducted a review of candidate third party implementations of RAKE and TextRank. The authors of RAKE and TextRank give unit examples of input and output which where used to establish functionality. In this review, we check that the implementation appears to follow the method descriptions as best to our ability. We focus on testing individual steps of the implementations on small examples and the author's examples. Special care was taken on examining the role of tuning parameters, some implementations did not expose an interface for the stated parameter tuning in the papers. In these cases, modifications were taken to expose these parameters. Take for example the reviewal of PyTextRank. We verify that PyTextRank constructs an undirected graph which is used in the networkx's function PageRank, which is an information retrival method that is the basis of TextRank, and establish their tuning of the PageRank settings such as a convergence cofficent accordingly to the paper. We note that PyTextRank does not expose the window parameter used to establish word co-occurance relations, so we had to modify the library to expose this parameter.

We found that many public TextRank implementations did not expose the same parameters as in the paper and that some do not fully implement the method (gensim (2019), summanlp (2019), LIANG (2019)). These implementations generated significantly different keyword/keyphrases than that of the unit example. Furthermore, all three implementations did not implement the keyphrase post-processing as in the paper. TextRank authors suggest that if two keywords/keyphrases are adjacent in the text then they are collapsed into a single keyphrase. In effect, these proved to be only keyword extraction methods.

We decided upon the "PyTextRank" (Nathan (2016)), as our representative implementation of TextRank given its strong similarty to the author's unit examples and its inclusion of the keyphrase postprocessing. The authors of PyTextRank note minor enhancements they make over the author's implementation. PyTextRank leverages spaCy to lemmatize, chunk nouns, and conduct named entity recognition. These improvements serve to prune the graph generated in TextRank for better functionality. RAKE on the otherhand had fewer but common well functioning implementations (Sharma (2017)) that conformed to the examples given by the originiating paper.

Lastly, we implemented the keyword adjacency stoplist generation method as described in Rose et al. (2010) and generated a stoplist from the Hulth training dataset. We found signifcant difficulties replicating the method described in RAKE, there were insufficent details to do so adequately. Rose et al. (2010) does not discuss how the keywords were matched in the abstract, only that they were matched and then looked to its adjacent words to continue on with the stop-list generation algorithm. We used spaCy's PhraseMatcher to find keyword matches in the abstracts, but on individual abstract evaluation, we found that many of the keywords had slightly different wording than what existed in the abstract. This rendered PhraseMatcher unable to find the keyword in the abstract. In addition, many of the keywords did not actually exist in the abstract. We can infer that during Hulth (2003)'s manual process of selecting keywords these abstracts, keywords were chosen that best represented the abstract subjects, not just existing words in the abstract.

5 Evaluation

As our project is based on replicating the results of a paper, we evaluate the replication aggregate measures recorded as per the initial study and conduct an error analysis from samples from the replicating evaluation. First, we summarize the evaluation from Rose et al.'s (2010), and describe the data-sets and code we use. Then we present our findings and conduct an error analysis of the results.

RAKE was compared to TextRank and seminal supervised learning methods (Hulth, 2003) over a data-set of human keyword annotated scientific paper abstracts originating from Hulth (2003). The authors compare two RAKE variants and two TextRank variants. As both RAKE and TextRank are unsupervised methods they evaluate

both methods on Hulth's test set, they largely ignore the test set. One variant of RAKE used a generic stop-list whereas the other variant used a stop-list generated from the training data-set using their stop-list generation method. RAKE was found to outperform all previously used keyword extraction algorithms in precision, efficiency and simplicity when using a domain specific stop-list. Using a generic stop-list, RAKE was found to be no worse performing than TextRank.

5.1 Data Sets & Code Used

To summarize, we have gathered or recreated the following data-sets and libraries:

1. Hulth (2003)'s data-set of human keyword annotated scientific paper abstracts. After postprocessing, this amounts to the test dataset-subset, and the human annotated keywords per abstract.
2. A representative implementation of RAKE via NLTK. Sharma (2017)
3. A representative implementation of TextRank via NetworkX. Nathan (2016)
4. Fox's Stoplist, a generic stop-list used in one RAKE variant.
5. A generated keyword adjacency stop-list, a domain specific stop-list used in one RAKE variant.
6. Various candidate implementations of TextRank. (gensim (2019), summanlp (2019), LIANG (2019))

We will be generating the Keyword Adjacency stop-list, which will be reproduced using the algorithm described in Rose et al. (2010). To complete this, we tokenize both each abstract and its keywords with spaCy's PhraseMatcher, and then completing the stop-list generation algorithm with this tokenized text. We will then test RAKE with our generated stop-list.

5.2 Evaluation Metrics

The measures reported in the original study by method were: number of extracted keywords, correct number of extracted keywords, average precision, average recall, and average f-measure (f1-score). We will compare our recorded measures to that of the original study to what extent are Rose et al.'s (2010) results reproducible. We will consider the results reproducible if the ordinal performance between RAKE and TextRank variants can be verified.

5.3 Results

We present our replication in table 1 on the next page in the same format as Rose et al. (2010). Due to formatting limitations, we will present the results from our experiment then the reference results section wise. Our experimental results are the first and second sections, the latter are for reference.

We are unable to replicate Rose et al. (2010)'s main conclusion that RAKE is a superior alternative, by f-score and precision, when using a generated domain specific stoplist to TextRank. However we recognize that RAKE using a generic stoplist modestly outperforms an enhanced TextRank implementation by a small margin in all three measures. Also, we note that in the reference using a larger window size hampers the significantly performance of TextRank while in our experiments, it provided a minor increase in performance. We find that both methods have generally similar performance profile in precision, recall, and F-score. We recognize two caveats. One, we could not replicate the exact keyword generation process as there was insufficient details to do so. Two, we use an implementation of TextRank which includes minor enhancements, albeit this was the closest implementation found to the original.

Overall, our experiments show that both RAKE and TextRank demonstrate similar levels of performance. By precision, both methods obtain values in the 25-30% region and 40-45% region by recall. Both methods suffer from generating too many false postive keywords/phrases although both methods have moderate abilities to detect relevant keywords/phrases. We are able to demonstrate similar levels of performance of RAKE using the Fox's stop-list and TextRank using a window size of 3 when comparing F-scores to the reference. RAKE using Fox' stop-list also demonstrates similar precision and recall. As the authors of both studies note, it is not possible to achieve perfect precision and recall upon this test data-set. Human annotated keywords may not necessarily be used in the abstract, hence they may not ever be produced via an extractive method.

Lastly, while runtimes were not a focal point for our analysis we could confirm that PyTextRank is slower than RAKE-NLTK via the profiler included in Spyder IDE. If these results are considered representative of RAKE and TextRank, then this confirms the claim in Rose et al. (2010) that RAKE has faster run-times. For pragmatic concerns, RAKE-NLTK is slightly preferable for its speed and marginally better performance.

Table 1: Results of automatic keyword extraction on 500 Inspec test abstracts using a Python implementation of RAKE (Rose et al., 2010) and TextRank (Mihalcea and Tarau, 2004)

	Extracted Keywords		Correct Keywords				
Method	Total	Mean	Total	Mean	Precision	Recall	F-Score
Our RAKE Implementation							
KA stoplist (generated)	8891	17.8	1962	3.9	23.8	40.7	28.7
Fox stoplist	8152	16.3	2125	*4.3	*27.2	*44.3	*32.4
Our TextRank Implementation							
Undirected, co-occ. window = 2	7884	15.8	1973	3.9	26	41.5	30.8
Undirected, co-occ. window = 3	8187	16.4	2064	4.13	25.9	43.0	31.2
RAKE Ref. (Rose et al., 2010)							
KA stoplist ($df > 10$)	6052	12.1	2037	4.1	33.7	41.5	37.2
Fox stoplist	7893	15.8	2054	4.2	26	42.2	32.1
TextRank Ref. (Rose et al., 2010)							
Undirected, co-occ. window = 2	6784	13.6	2116	4.2	31.2	43.1	36.2
Undirected, co-occ. window = 3	6715	13.4	1897	3.8	28.2	38.6	32.6

5.4 Error Analysis

This section elaborates upon four subjects. First, the general drop in performance of our experiments and the reference set. Second, our implementation of keyword adjacency stop-list generation and RAKE. Third, we compare the differences in output of RAKE and TextRank.

Comparing our experimental results and the reference, there are consistently more keywords extracted and lower performance in the experimental results. One confounding factor is the preprocessing/reconstruction of abstracts and the tokenization of the abstracts in both methods. RAKE and TextRank rely upon assigning terms a score after tokenzing the text and processing from there after. It is likely some error was introduced in reconstructing the text from tokens in the XML format of the Hulth dataset and by variation by the tokenizers used by either implementation. There are three confounding sources of error here, the initial XML tokenized format, our reconstruction method, the tokenizer used in the implementations of RAKE and TextRank. Both RAKE and TextRank do not mention the exact tokenizer they used. We note that the mean correct keywords are fairly consistent. Since both methods output keywords based upon the size of their internal representation (eg. size of the TextRank graph), differences in tokenization will alter the amount of keywords/phrases returned. This may also explain the opposite in trends between the differing window sizes between our experiments and the reference of TextRank. A larger co-occurrence window can buffer against noisy tokenization.

As mentioned in the implementation section we had difficulties replicating Rose et al. (2010)'s stop-list generation method. Following the description resulted in a stop-list of 99 words (13%) of the referenced 763 list. For reference, the Fox stop-list is approximately 400 words. RAKE's authors try various degree-frequency thresholds for stop word candidacy in their keyword adjacency scheme, they list their findings in Table 1.4 of their paper. To summarize, they try three thresholds, decreasing in performance with stop-list size in parenthesis:

1. KA df = 10 (Size = 763, F-Score = 37.2)
2. KA df = 25 (Size = 325, F-Score = 35.1)
3. KA df = 35 (Size = 147, F-Score = 32.8).

Decreasing the size of the stop-list had considerable effects on the performance of RAKE. Our stop-list performs starts to perform similarly to the smaller KA stop-lists. This contrast shows the importance of the stop-list in RAKE function and how error in our stop-list generation can confound the performance of RAKE. There appears to be marginally decreasing utility of larger stop-lists, however the marginal decrease is modest. Using a domain specific stop-list leads to better performance, contrasting the moderately sized Fox's stop-list (400 terms) and the moderately size KA stop-list (325 terms) in the reference.

TODO: Finish Section Comparison

6 Conclusion

We find significant but not excessive differences in replication of Rose et al. (2010)'s evaluation, albeit we were unable to replicate their keyword adjacency scheme which is a significant factor in their findings. We are able to validate the performance of RAKE using Fox's stop-list and the results of TextRank to a sufficient extent. We find that RAKE performs similarly to TextRank using a generalized stop-list with a modest increase in performance in precision, recall, and f-score. Our replication notes difficulties that hinder public adoption of these two promising methods. First, RAKE's description of their complementary stop-list generation method hinders its flexibility. RAKE when bundled with a generic stop-list functions is comparable to its peer TextRank as an unsupervised method. RAKE using a domain specific stop-list outperforms its peer, however this method has difficulties to implement and moves the method away from being a truly unsupervised method. Second, many implementations of TextRank do not fully implement the method as described. Many implementations skip the post-processing of keywords into key-phrases which has considerable use. By our evaluation, both methods obtain values in the 25-30% region by precision, 40-45% region by recall and near 30% by F-score. We hope our analysis can act as an anchor point for further analysis of these keyword extraction algorithms.

Improvements to stop-list generation is a natural extension point for our work. We noted difficulties in replicating the complementary stop-list generation method supporting RAKE. Online, we find that few stop-list generation methods are available or even proposed. Developers prefer to bundle their methods with established generic stop-lists such as Fox's stop-list, NLTK's stop-list, MySQL's stop-list, and Google Search's stop-list. Alternatively, some authors implement a simple top-frequency based stop-list. Few authors attempt to devise a method to construct domain specific stop-lists. We believe there is widespread applicability in this endeavor outside of RAKE as well.

7 Acknowledgments

Teamwork Breakdown: Daniel wrote the code for re-creating the abstracts and the measures for testing the output for TextRank and RAKE. Sharon wrote the code for generation the stoplist and found the datasets used from Hulth (2003). Sharif worked with Sharon on the Keyword adjacency stoplist generation code and revised the final report.

References

Florian Boudin. 2017. Preprocessed inspec keyphrase extraction benchmark dataset. https://github.com/boudinfl/hulth-2003-pre.

gensim. 2019. gensim-textrank. https://radimrehurek.com/gensim/summarization/keywords.html.

Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*. pages 216–223. https://www.aclweb.org/anthology/W03-1028.

Xu LIANG. 2019. Understand textrank for keyword extraction by python. https://towardsdatascience.com/textrank-for-keyword-extraction-by-python-c0bae21bcec0.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*. pages 404–411.

Paco Nathan. 2016. Pytextrank, a python implementation of textrank for phrase extraction and summarization of text documents. https://github.com/DerwenAI/pytextrank/.

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text Mining* page 120. https://doi.org/10.1002/9780470689646.ch1.

Vishwas B Sharma. 2017. rake-nltk. https://pypi.org/project/rake-nltk/.

summanlp. 2019. Summa - textrank implementation for python 3. https://github.com/summanlp/textrank.