Unsupervised Glossary Extraction

Deep Inder Mohan (IMT2017013), George Abraham (IMT2017019), and Kaushal Mittal (IMT2017024)

Project for DS/SP 829 Natural Language Processing under the guidance of Prof. G. Srinivasaraghavan

International Institute of Information Technology, Bangalore

July 19, 2020

Abstract

The task of extracting key-terms and definitions from text is a decades old NLP problem with a high potential impact. This technology has been used in multiple applications ranging from automatic medical diagnosis tools to smart question answering textbooks. In this project, we explore the past research in this task and existing approaches to this problem. We then propose a potential system that performs this task in a completely unsupervised manner and without the help of a knowledge base, using highly sophisticated neural network based modern language models.

1 Introduction

Our defined task for this project was titled **"Glossary extraction"**, with the description being: *"To extract a set of glossary terms and their definitions from a given body of text in an unsupervised manner"*. Going into the project, the primary idea we had was to split the glossary extraction task into two serialized sub-tasks:

1. Identifying the keywords in the text (key term extraction)
2. Extracting the definition of those keywords from the text

An important assumption made before approaching this task itself was the fact that we would be working with text from study books/textbooks, as then the hypothesis of the definition of a glossary term being in the text itself would be acceptable.

Formally, the task of key-term extraction can be described as follows:

> Given text T from a collection of documents D, whose vocabulary is V, output a set $K \subseteq N(T)$ that describes the key-terms of T.

In this definition, $N(T)$ is the set of n-grams from T for all $n = 1, ..., \eta$ (η is a maximum term length hyperparameter) where an n-gram is defined as a sequence of n consecutive terms that show up in the initial text. An important note is that the phrase 'key-term' is a general way of stating 'a term of importance.' The actual definition of the key-term will vary from application to application.

2 Previous Work

As a part of the groundwork we did in order to gain a better understanding of the problem at hand, especially in terms of its perceived subtasks, We explored various supervised, semi-supervised, and unsupervised approaches at these subtasks. Seeing why these approaches were explored also gave us an understanding of the potential applications of such an NLP based system.

2.1 Supervised Approaches

2.1.1 Building A Scalable Database Driven Reverse Medical Dictionary Using NLP Techniques[8]

One of the first papers we saw that documented a very good use case for such a system was in a paper titled, *"Building A Scalable Database Driven Reverse Medical Dictionary Using NLP Techniques"*. This paper described a method for mapping a set of symptoms into a possible diagnosis. In the pipeline for dictionary building, the paper described using a Word Sense Disambiguation module followed by a semantic (or topological) similarity module. For the WSD task, the paper uses the **"Lesk Algorithm"**. This algorithm works by comparing the dictionary definition of an ambiguous word with the terms contained in its neighborhood. Although the task of definition extraction is not the same as WSD, and this algorithm uses word definitions from a dictionary, a liberty we cannot afford in the unsupervised approach, the idea of the Lesk algorithm, which is that words in a given *neighborhood* (section of text) will tend to share a common topic, was conducive to our final approach. The idea of semantic similarity was also one we wanted to explore, as it seemed like an appropriate manner in which we could enforce the directive that for any extracted glossary, the terms and their definitions should give the user an idea of the subject of and the information contained in the paper.

Then the other works we explored were in context with the same task of creating glossaries for an intelligent textbook by Stanford under project, 'Inquire'[3], primarily to automate the task. It serves as a quality dataset for the same, as the first few chapter have been manually labeled for key terms and definitions.

2.1.2 Automated Glossary Construction of a Biology Textbook[5]

They discuss how the task of automated term extraction (ATE) has evolved from hand crafted sophisticated linguistics feature and statistical rules towards a machine learning based approach. Early works which use of machine learning make use of naive bayes classifier to identify key terms. They use CountVectorizer, a scikit package, to identify individual words and n-grams and also to vectorize the input. Some pre-processing steps involved are lemmatization and identification and removal of stop words and specific terms such as 'Chapter', 'Figure', and so on. The key idea after pre-processing is adding features such as TF-IDF, location of appearance word (in the sentence and chapter), whether or not the term belongs to top 5000 words, and so on which will help the machine learning models in classification. This idea was an explanation and a major takeaway for our work. They build three different models namely, support vector machines(SVM), guassian naive bayes (GNB), and multi layer perceptron or neural net. To compensate for the very small number of positive instances (<1%), they use techniques such as oversampling and evaluation metrics such as precision and recall. Results show neural net outperforms guassian naive bayes and GNB outperforms support vector machine.

2.2 Semi-supervised and Unsupervised methods

One of the methods we explored for the key-term extraction task was proposed in the paper titled *"Featureless Deep Learning Methods for Automated Key-Term Extraction"*[4]. This paper is also written in the context of the Stanford Inquire project, and in this, the authors perform the task of key term extraction in a semi-supervised manner, with their method being training on an existing knowledge base of glossary terms of a known ontology using a co-training algorithm, gaining two distinct insights into the source text by training both a CNN and a fully connected neural network. While the ideas of this research were fantastic, we ultimately decided to not explore semi-supervised learning algorithms due to the fact that for our task, there would be no access to an existing knowledge base, i.e., relevant textbooks with predefined glossary terms.

Another paper titled "Automatic Extraction of Textbook Glossaries Using Deep Learning"[7] tries to tackle the knowledge base problem, and inspired a solution to the definition extraction problem as well. For term extraction, the paper described the use of the BIOES tagging scheme. They build word representations by concatenating GloVe word embeddings and character-level representations and input them into a BLSTM layer that in turn feeds into a fully connected softmax output layer that produces a predicted tag for each token in the sentence. Figure:1

For definition extraction, the task was approached in this paper as a binary sequence classification task, where sentences from their knowledge base would be tagged as being glossary definition sentences or not. One

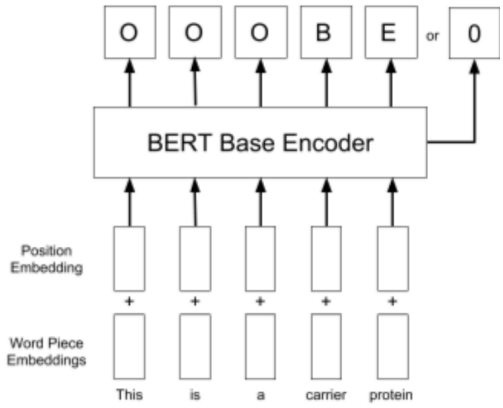


Figure 1: Key term extraction through BIOES tagging

of the ways this was performed was by fine-tuning BERT by adding a sigmoid output layer to its network, and training it on science textbooks scraped from OpenStax. The results produced by this method were quite promising in the task, which made this according to our research, the best possible approach for unsupervised glossary generation. The primary disadvantage is the fact that while the key term extraction is unsupervised, labeled data is again needed for the dictionary extraction task. We ultimately decided upon a completely different methodology, as this method assumes knowledge of the ontology of the text as well. Additionally, fine-tuning BERT for this approach was done in a supervised manner by scraping existing glossaries from OpenStax textbooks. The important relevant conclusion made by this paper in the scope of our task was the fact that BERT outperformed smaller, more specialized modes on both term and definition extraction tasks.

3 Approach

3.1 Glossary Term Extraction

The task of extracting glossary terms from text is performed by a pipeline consisting of three steps (Fig 2):-

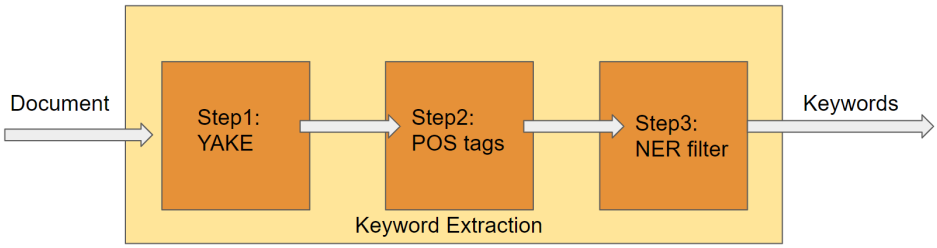


Figure 2: Glossary term extraction pipeline

1. **Step 1: Keyword Extraction**
   Older approaches to the keyword extraction task included generating corpus oriented statistics of individual words. These are, for multiple reasons, not the best algorithms for the task, and most modern approaches use what are known as "document-oriented methods". As a starting point, we tried to understand and implement one of the most popular extraction algorithms, RAKE[6]. Rapid Automatic Keyword Extraction (RAKE) is an unsupervised, domain-independent, and method for extracting keywords from individual documents. The philosophy driving RAKE was to develop a keyword extraction method that was extremely efficient, operated on individual documents to enable application to dynamic collections, was easily applied to new domains, and operated well on multiple types of documents. Here is a brief description of the working of the algorithm algorithm:-

- Identify the candidate keywords in the document by removing 'stopwords', which are words like prepositions, which don't have much lexical meaning, but will occur throughout the text at a high frequency. The set of words between every pair of stopwords becomes one candidate keyword.
- Create a graph (matrix) of word co-occurrences for the text, and score every word in the graph using metrics of frequency and degree. The score of a candidate keyword becomes the sum of the scores of its member words. (Fig 3)
- The top-scoring candidate keywords become the actual keywords for the document.

Figure 3: Example matrix generated by RAKE

	keyword	extraction	difficult	many	libraries	help	rapid	automatic
keyword	3	3	0	0	0	0	1	1
extraction	3	3	0	0	0	0	1	1
difficult	0	0	1	0	0	0	0	0
many	0	0	0	1	1	0	0	0
libraries	0	0	0	1	1	0	0	0
help	0	0	0	0	0	1	0	0
rapid	1	1	0	0	0	0	1	1
automatic	1	1	0	0	0	0	1	1

By isolating words that score highly in degree and frequency in their co-occurrence graph, RAKE is able to isolate the words that, in some sense, dictate the context of the document. Hence, these words in our opinion, are the best potential terms for a glossary that captures the essence of the document.

The disadvantages of RAKE start to show when we try to isolate exclusively unigrams with it. By the very nature of its procedure, i.e., scoring word-groups as the sum of their candidate words, RAKE struggles with unigrams, and hence after appropriate testing, was deemed by us to be unfit for the glossary generation task.

The approach we did end up going with was the unsupervised algorithm called YAKE[1] (Yet Another Keyword Extractor). This algorithm showed better results than RAKE for us. Here is a brief description of the working of YAKE, which involves 4 main subtasks:-

(a) **Text pre-processing:** Split the text into individual terms whenever an empty space or a special character (e.g., line breaks, brackets, comma, period, etc.) is found.

(b) **Feature Extraction:** A set of five features to capture the characteristics of each individual term. These are:

- **Casing:** Particular attention is paid to any word starting with a capital letter (excluding ones at the beginning of sentences) or to any acronym (where all letters of the word are capital) under the assumption that these words tend to be more relevant. Instead of counting them twice we only consider the maximum occurrence within the two of them. The corresponding equation is:

$$W_{\text{case}} = \frac{max(\text{TF}(U(w)), \text{TF}(A(w)))}{log_2(\text{TF}(w))}$$

  Where $\text{TF}(U(w))$ is the number of times word w starts with an uppercase letter, $\text{TF}(A(w))$ is the number of times the candidate word w is marked as an acronym and $\text{TF}(w)$ is the frequency of w. So higher the W_{case} value the more important/relevant it is.
- **Word Positional:** Word Positional values those words occurring at the beginning of a document/chapter of a textbook based on the assumption that relevant keywords often tend to concentrate more at the beginning of a document.

$$W_{\text{position}} = log_2(log_2(2 + Median(Sen_w)))$$

Here, Sen_w indicates the positions of the set of sentences where the word w occurs, and Median of the positions is computed. We get an increasing function,as words that are positioned at the end of the document have higher W_{position} values and conversely, the more often a word occurs at the beginning of a document the less it's W_{position} value. Keywords are extracted chapter wise from an textbook, and important words in a chapter tend to occur towards the start of the chapter.

- **Word Frequency:** Word Frequency indicates the frequency of the word and encapsulates that more frequent words are more important. But for longer documents, the frequencies of most words would have a high value. To counteract this, the frequency of a word w is divided by the mean of the frequencies (MeanTF) plus one time their standard deviation σ.

$$W_{\text{freq}} = \frac{TF(w)}{(MeanTF + 1) \times \sigma}$$

Higher W_{freq} is better. Frequency is divided by the mean as if the document is long, words will have a higher frequency.

- **Word relatedness to context:** This computes the number of different terms that occur to the left (and right) side of the candidate word. The more the number of different terms that co-occur with the candidate word (on either side), the more meaningless the candidate word is likely to be. The more insignificant the candidate word is, the higher the score of this feature will be. Example of such terms would be stop-words.
- **Word DifSentence:** quantifies how often a candidate word appears within different sentences. Similar to Word Frequency, Word DifSentence assigns more value to those words that often occur in different sentences.

$$W_{\text{DifSentence}} = \frac{SF(w)}{\#Sentences}$$

Here, $SF(w)$ is the sentence frequency of w and #Sentences is the total number of sentences in the text. Higher the value of $W_{\text{DifSentence}}$ the more relevant it is.

Note: The features $W_{\text{DifSentence}}$ and W_{freq} are combined with Word Relatedness to Context, meaning that the more they occur in different sentences the better, but should not occur frequently with different words on the right or left side.

(c) **Individual Terms Score:** Individual terms score is computed by heuristically combining all these features into a single measure such that each term is assigned a weight $S(w)$. The smaller the value of $S(w)$, the more important the word w would be.

$$S(w) = \frac{W_{\text{Rel}} \times W_{\text{Position}}}{W_{\text{Case}} + \frac{W_{\text{Freq}}}{W_{\text{Rel}}} + \frac{W_{\text{DifSentence}}}{W_{\text{Rel}}}}$$

(d) **Candidate Keyword List Generation:** We have succeeded in assigning scores for each word. The final step is accounting for n-gram words (uni-, bi-, trigrams).

$$S(kw) = \frac{\Pi_{w \in kw} S(w)}{TF(kw) \times (1 + \Sigma_{w \in kw} S(w))}$$

The score of a candidate keyword is the product of individual contributions of each word, so that if the individual terms are relevant then the produced score is low as well, meaning that it is an important keyword. To prevent unfair advantage to bi and trigrams, this score is divided by the mean of individual component scores. Also, the keyword score is divided by the frequency of that keyword ($TF(kw)$) to penalize less frequent terms.

Given the score of candidate keywords, we rank and take the lowest scoring keywords as our candidates for the glossary term.

YAKE performs very well compared to other approaches and shows promising results (Fig 4) for keyword extraction on different datasets including long medical, computer science and news documents.

Figure 4: Yake scores on SemEval2010, Shutx2008, 500N-KPCrowd and WICC

Method	SemEval2010			Schutz2008			500N-KPCrowd			WICC		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
YAKE!	0.153	0.103	0.123	0.217	0.058	0.091	0.251	0.063	0.101	0.050	0.141	0.073
TextRank	0.101▼	0.067▼	0.081▼	0.198▼	0.052▼	0.082▼	0.265	0.063	0.103	0.018▼	0.058▼	0.027▼
TF.IDF	0.036▼	0.023▼	0.028▼	0.100▼	0.028▼	0.043▼	0.223▼	0.060▼	0.095▼	0.026▼	0.067▼	0.037▼
SingleRank	0.035▼	0.022▼	0.027▼	0.082▼	0.024▼	0.037▼	0.190▼	0.054▼	0.084▼	0.017▼	0.045▼	0.024▼
RAKE	0.007▼	0.004▼	0.005▼	0.013▼	0.004▼	0.006▼	0.120▼	0.038▼	0.058▼	0.004▼	0.012▼	0.006▼

2. **Step 2: Pruning based on POS tagging**
   The keywords extracted in step 1 can possibly belong to any part of speech. However, we hypothesize that glossary terms are generally not verbs, adverbs or prepositions. Therefore we extract only the keywords that correspond to nouns/adjectives as our candidate glossary terms. More specifically keywords with POS tags: ['NN', 'NNP', 'NNS', 'NNPS '] are retained. We have used the POS Tagging Algorithm from **nltk** library to achieve this task.

3. **Pruning based on Named Entities**
   Named entities are generally not included in glossary terms. For the rare cases when this might be false, we are assuming that we would not be able to define these entities in an extractive manner. Therefore all keywords belonging to the categories [People,Dates,Time,Money,Places] are pruned from the candidate set. For example, in science textbooks, Darwin or London may be mentioned many times but we are not interested in defining such entities. We have used the **spaCy** package and the Named Entity Algorithm it provides for this task.

3.2 Glossary term definition extraction

Unlike the previous approaches explored, we extract the definition of a keyword from the given text itself instead of using external resources like a knowledge base. This is done as an attempt to make the system ontology independent, which it wouldn't be in case it was based on data from some extracted knowledge base. Our approach towards this task is centered around two primary assumptions:-

1. The definition of an introduced keyword can be found in that text itself.

2. This definition is expected to be in the vicinity of the keyword.

As stated before, the task has been done keeping in mind the structure of a well-ordered textbook. In such an instance, we can expect the definition of a keyword to be in the vicinity of the keyword itself. The vicinity can vary from a sentence to 2 sentences, etc. We need to note that we do not know the location of the definition across multiple instances of the keyword, i.e the definition can be in the neighborhood of the first time we see the keyword or the second or maybe even the last. But we still expect it to be present within the vicinity.

The intuition is that we need to find a sentence/subsentence across all vicinities such that the definition we choose is semantically similar/carries a similar meaning to the keyword we have chosen. For this purpose, we turn to BERT[2] (Bidirectional Encoder Representations from Transformers). BERT is a state-of-the-art language model that we use to convert potential definition phrases to vectors, and then check for similarity mathematically. The reason for choosing BERT is that even for a newly introduced keyword, it can still produce a vector representation for that by breaking it down into smaller sub-words. The sentence/subphrase vector representation we use is calculated as the mean of individual word component representations.

We take the word representation of the keyword using the BERT model - let these we represented by X. We then use a sliding window approach over the vicinity at every instance of the keyword to scan for a potential definition. The sliding window approach is defined as follows: for the neighbourhood around the keyword, take all possible subregions in that neighbourhood (the subregions are context-independent and are treated as individual sentences since definition should depend on itself) and get the vector representation for

that subregion, represented by R. The sliding window size represents the size of the candidate definition and this is increased from 1 to the size of the neighbourhood. Then the instance of R which is closest to X in the vector space is chosen as it's definition since both carry similar semantic meanings.

The closeness of the glossary term vector and the extracted definition vector is calculated using cosine similarity. Finally, we choose the subregion whose corresponding BERT vector generates the highest cosine similarity value, which implies that the phrase and the keyword refer to very similar things. This subregion is then extracted as the definition of the glossary term.

4 Results

4.1 Baseline implementation results

The Glossary term extraction pipeline works quite well, and is able to isolate relevant terms from textbooks which are appropriate for glossary building. Fig 5 shows some results from bur keyword extraction pipeline after being run on a chapter titled *"Evolution"* from the NCERT biology textbook.

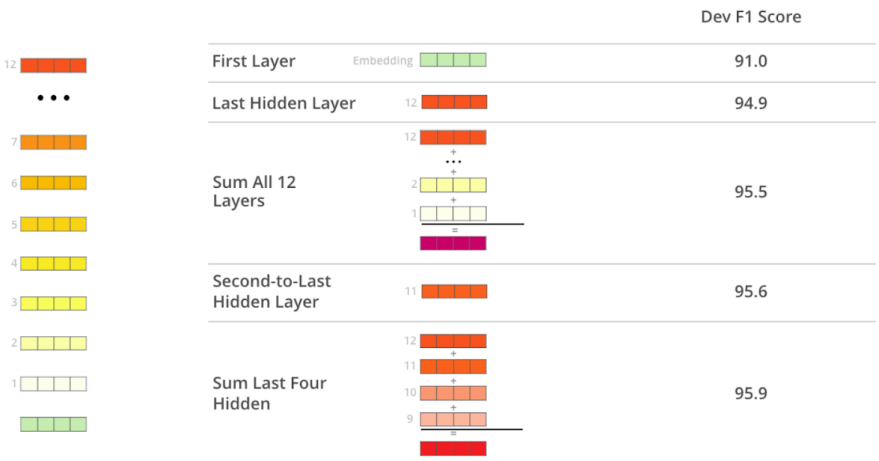
Figure 5: Extracted glossary terms on NCERT chapter

```
[ ] for i in pos_ner_cleaned_keywords:
      print(i)
```

```
biology
evolution
origin
life
forms
theory
radiation
man
history
earth
time
universe
water
form
organisms
figure
population
selection
fossils
mammals
animals
moth
adaptive
natural
frequency
```

In the definition extraction task, for our sliding window, after some trial and error, we found the best size to be around 16, i.e., 8 words in either direction of the glossary term word. We use the base BERT architecture which has 12 transformer encoder layers stacked on top of each other and outputs 768-dimensional vectors from each layer. The versatility of BERT allows us to pull out vector representation for words from any layer and still get decent outputs. But which layer or combination of layers provides the best representation? The BERT authors tested this by feeding different vector combinations as input features for the named entity recognition task and observing the resulting F1 scores (Fig 6).

Figure 6: BERT outputs at different layers



After experimenting on a combination of outputs from different layers, we observed that the mean of the last four layers gave longer definitions, but the definitions extracted did not sufficiently capture the meaning of the target glossary term (the cosine similarity from even the best rated phrases were relatively low). On the other hand, if we consider outputs from the last layer, the extracted definitions tend to be shorter, but they are more semantically similar to the target glossary term.

Although results from this approach are largely hit or miss, in some cases, the definitions generated can be quite accurate to the context. After training on a sample article on "kaggle competitions" of around 200 word, the model extracted keywords such as '*google*' and '*declined*'. The sliding window approach extracted the definitions '*platform*' and '*deny*' for each of them respectively.

However on a majority of the other terms, the definitions seemed quite arbitrary and somewhat random. Our hypothesis is that this is primarily due to our decision of using the mean of the vectors representing individual component words as the representation of the entire definition phrase. Our reasoning in making this hypothesis is as follows:-

- This method fails to neglect the contribution of the stop words, which offer no semantic insight into the extracted phrase, but have an equal contribution to every other word. Removing the contribution of stop words would help in improving the model, but this would increase time complexity in an already time-consuming approach.
- This method does not isolate similar/relevant terms to the candidate glossary term. The more the number of such terms in the decided neighborhood, the better the quality of the extracted definitions.

4.2 Improvements to baseline

One of the main pitfalls of the baseline approach is that the extracted definitions are short and the vector representation of stopwords dilutes the contribution of important terms in a subphrase. For these reasons, we add the following two additional features to our definition extraction process:-

1. We ignore the contribution of stopwords like ('of',' has') etc while calculating the vector representation of the candidate definition.
2. We multiply the similarity score by a function that accounts for the length of the definition so that it does not bias towards smaller definitions. The contribution of the length factor is capped at definitions of length 8.

An example of the results seen after applying these changes is shown in Fig 7.

Figure 7: Results after weighing definition length

```
In [330]: final_dictionary

Out[330]: [[[['of', 'evolution'], ['biology']], 1.3403176410717053],
           [[['biology'], ['evolution']], 1.3403176410717053],
           [[['mechanism'], ['origin']], 1.3140087074620868],
           [[['generation'], ['life']], 1.3162026903422952],
           [[['form'], ['forms']], 1.3766519570113949],
           [[['concepts'], ['theory']], 1.3604724902496064],
           [[['meat'], ['man']], 1.2429505040735014],
           [[['into', 'the', '\x0cevolution'], ['history']], 1.2828314650281325],
           [[['there', 'was', 'no', 'atmosphere'], ['earth']], 1.2928041065455138],
           [[['of', 'geography'], ['radiation']], 1.2659547542442768]]
```

The performance is somewhat decent for terms that consist of composed words, i.e., words that can be broken down into smaller, more meaningful words during tokenization. However, for the shown example of kaggle getting tokenized, the tokenizer splits it into smaller subparts [Kag, gle] which do not carry any particular meaning and therefore the similarity measures fail to work here.

Other than the poor performance, the sliding window approach is also quite slow, having a worst-case time complexity of $\mathcal{O}(k.n.c^2.\lambda)$, where k is the size of the keyword table, n is the number of words in the source text document/book chapter, c is the size of the window on with the definition extraction is done, and λ is the time complexity for calculating the vector representation of a subsection using BERT.

5 Conclusion and Future Work

In comparison to the prior approaches studied for this task, our approach generates lackluster results. Although the glossary term extraction task is consistent and acceptable for an unsupervised, non-ontology dependent approach, the definition extraction task is almost unusable. We hypothesise that continuing along this approach for definition extraction may not be very successful,and hence KB based methods or even semi-supervised methods may be ideal for this task.

In terms of improvements, we can improve the speed of the existing approach by creating a hash table of the extracted glossary terms, as this could allow us to check the status of every word in the text in constant time. The time complexity would then be $\mathcal{O}(h(k).n.c^2.\lambda)$, where $h(k)$ is the look-up time of the hash table.

Additionally, we can also include better and additional pruning methods, which would allow words uncommon in the corpus to receive higher scores or better rankings than some common reoccurring words like '*figure*', '*chapter*', etc. We could also check for word relatedness to already extracted glossary terms to avoid redundancy of similar words, or of words that are derived. For instance, if *archive* is an extracted keyword, then the word *archival* should be ignored.

For improvements to the definition extraction module, we can try to somehow implement a modification in which only relevant words are used for comparison to the original keyword, instead of continuous parts of the neighborhood. Also, some system for adding weight to definitions extracted in the beginning of the text may also be implemented, as important terms in textbooks tent to be defined in the beginning and not in the ending.

References

[1] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. Yake! collection-independent automatic keyword extractor. 02 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[3] Vinay K Chaudhri, Britte Cheng, Adam Overholtzer, Aaron Spaulding Jeremy Roschelle, Peter Clark, Mark Greaves, and Dave Gunning. Inquire biology: A textbook that answers questions. *AI Magazine*, 34(3), September 2013.

[4] Kush Khosla, Robbie Jones, and Nicholas Bowman. Featureless deep learning methods for automated key-term extraction, 2018.

[5] Anita Kulkarni and Rachel Smith. Automated glossary construction of a biology textbook, 2018.

[6] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. *Automatic Keyword Extraction from Individual Documents*, pages 1 – 20. 03 2010.

[7] Manish Singh and Matthew Boggess. Automatic extraction of textbook glossaries using deep learning, 2018.

[8] Pragya Tripathi and Manuja Deshmukh. Building a scalable database driven reverse medical dictionary using nlp techniques, 2017.