ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Natural Language Processing

Graph-Based Keyword Extraction from Scientific Paper Abstracts using Word Embeddings

Author:
Dinno Koluh

Supervisor:
Prof. Paolo Torroni

Co-Supervisor:
Dr. Federico Ruggeri

Bologna,
December 2023.

Abstract

In the era of information overload it became essential to efficiently extract concise, precise and quality information from large texts. One aspect of information extraction is keyword extraction where large texts are represented as sets of tokens i.e. keywords. This prospect of keyword extraction is paramount to researchers as they deal with huge numbers of scientific papers, and having a good and concise representation of those papers is essential for them. This thesis paper addresses that problem in the realm of natural language processing (NLP).

Using core concepts of NLP and modeling texts as graphs, we are going to build a model for the automatic extraction of keywords. This is done in an unsupervised manner as the importance of a word is calculated through the position and weights associated with respective words in the graph. The first metric used to calculate the graph weights are co-occurrence matrices and the other metric are word embeddings. Word embeddings became a crucial way of representing the semantic information of words as dense vectors.

The results of this paper were compared with keywords that were provided by authors of scientific papers in the area of computer science which act as the ground truth, but crucially are not a component in the model construction, but just serve as a verifier of the model's accuracy.

Keywords: NLP, keyword extraction, scientific papers, graphs, word-embeddings

Contents

List of Figures

1 Introduction and Motivation

Natural Language Processing (NLP) is a subfield or Artificial Intelligence (AI) and linguistics that has a focus on the interaction between computers and human languages. This is mostly restricted to written language as other fields (like Speech Processing) deal with spoken language (using audio instead of textual features). In the past decade NLP has seen huge attention with the introduction of some key concepts like *word-embeddings* [1] (which we are going to use) and *transformers* [2]. At the moment of writing of this paper, NLP is the subfield of AI that has the most resources invested into it, mostly in research and development, with new large language models (LLMs) coming out on a daily basis. Our focus will be a bit shifted from Deep Neural Network (DNN) models and more to traditional Machine Learning (ML) models. NLP has many subfields and application areas such as:

- Text classification
- Information retrieval
- Automatic translation
- Speech analysis
- Question answering
- Conversational agents
- Sentiment analysis

The field we are going to be working on will be **information retrieval**, more precisely *keyword extraction*. Keyword (keyphrase) extraction is the automatic selection of important and topical phrases from the body of a document [3]. Scientific papers are usually the area where keywords are frequently used as researchers use them for a quick overview of papers and also sorting papers into different categories. This is the topic we are going to be working on as well. The keywords are going to be extracted from the abstracts of the scientific papers. One detail we should address that will be important later is the distinction between *keywords* and *keyphrases*.

Keywords would be single words or at most MWEs (Multi-Word expressions, i.e. "deep learning") while keyphrases are more complicated entities comprised of several words and they act as a single unit (e.g. the phrase "scientific paper" would be a keyphrase). When doing keyword extraction we might have as an output a combination of both, keywords and keyphrases. Usually keyphrases carry more information than keywords but a combination of both as an output is most representative. From now on, when referring to *keywords* it will also include *keyphrases*, if not explicitly stated otherwise.

The model to be used for keyword extraction is **graph-based**. The motivation behind using graphs is that graphs are a well-studied and understood concept and many practical problems can be represented with graphs. There are also a plethora of algorithms which are going to be useful for the case of instance ranking.

The idea is to model words in a paper abstract as nodes of a graph. Not all the words in the abstract should be included, as keywords are usually composed of a combination of nouns and adjectives (e.g. scientific[ADJ] paper[N]). This means that some preprocessing of the raw text will be required, especially when dealing with MWEs. We will speak about this in the next chapter.

As stated, the nodes of the graph will be modeled as words, but the question comes on how to model the edges of the graph?

It starts from the assumption that words that occur in the same context tend to carry a similar meaning. The idea is to use a **sliding window** of some predefined size and words that are in that window are connected with and edge. In that way the final graph carries semantic (the meaning of words) information of the input text. The number of occurrences when sliding the window over the entire text will give us the initial graph weights. To solidify this relationship another metric that we will use are word embeddings. Word embeddings are essentially a way of representing words as vectors of numbers. We will dive more deeply into how word embeddings are computed ans used but for now we can assume that word embeddings are vectors of numbers and that these vectors carry semantic information of the corresponding word. This means that words that have a similar meaning (whatever that might mean in some general picture) also have similar vector representations and that we can use the usual mathematical tools on these vectors like the dot product which means that we can actually measure the similarity between words.

We now have the complete representation of the input text as a graph. Now we need to somehow rank the nodes according to the graph edges and weights. We can refer to the ranking procedure as to finding the importance of each node. There are several algorithms which can find the importance of nodes, and we will speak in more detail about them later, but for now can assume we have a black box which takes in the graph representation of the abstract as described before and gives us all the nodes (word) ranked by their importance.

We now have a list of the most important words, but there is one more step that we can do to get a more robust output. The given output would be a list of words, but we might want to have phrases instead of keywords (e.g. the phrase "scientific paper" is more representative instead of just "paper"). We can use the words we got out from the graph and traverse the initial text for phrases in which they appear. Then based on those phrases in the initial text and the importance of the words which they are made of, we can get alongside the ranked words also the phrases in which they appear. In this way we can also generalize the problem by letting a phrase be only made up of one keyword. If it is made up of more than one keyword we can average it out, and in the end get the ranking of the specific phrases which appear in the input text.

This was a rough explanation of the procedures which should give us an overview of the steps involved in developing a pipeline for the extraction of keywords from paper abstracts. In the next few chapters we are going to look in more detail in the inner workings of the steps involved. We will start with the NLP preprocessing pipeline.

2 NLP Pipeline

In this chapter we are going to explain the preprocessing steps needed to transform the raw input text into something that can be modeled with a graph. These preprocessing steps are going to be carried out with concepts from NLP like tokenization, sentence splitting, lemmatization, etc. All these NLP concepts used for preprocessing can be stacked into a pipeline hence the name of the chapter, **NLP pipeline**.
We will explain these concepts giving examples and at the end show the whole pipeline as a schema. In the end we will have single instances of processed words which are going to be the graph nodes. Let us start!

2.1 Tokenization

Tokenization is the basic and most important NLP concept. Let us firstly define what is a token. A **token** is a sequence of characters grouped together as a semantic unit used for processing [4]. Depending on the use case, tokens can be equivalent to words, but they don't have to. An example of words and tokens not being equivalent is:

$$\text{I'd} \rightarrow [\text{I, 'd}]$$

In this case the word "I'd" was broken into two tokens. Depending on the specific tokenization method used, words can be broken into lower units (subwords):

$$\text{unlucky} \rightarrow [\text{un, lucky}]$$

Here, we have that the adjective "unlucky" was broken into its prefix "un" and the adjective "lucky". In this example tokenization provided a way to work with text on a granular level, breaking down words into units which might not even be real words, but in this way we are making text processing tasks more cohesive for machines.
The process of tokenization would be the process of breaking down an input text into single tokens. As the input text inherently has whitespaces and newlines, those are usually excluded from the obtained tokens. We can look at an example sentence broken down into tokens:

She didn’t have time. $\rightarrow$ [She, did, n’t, have, time, .]

When dealing with whitespaces, tokenization is a trivial process, but when dealing with punctuations, tokenization becomes a hard problem. The reason is that sometimes punctuations should be kept, but sometimes they should be separated. In the example above, the token “time.” was separated into “time” and “.”, but let’s take a look at this example:

She didn’t have time for Mr. Nobody. $\rightarrow$ [She, did, n’t, have, time, for, Mr., Nobody, .]

In this example the period for the token “Mr.” was not separated while for “Nobody.” it was. This is the reason why tokenization can be ambiguous and hard to compute correctly. And there are tens of other example os ambiguities apart from punctuation marks. One solution to the problem we’ve seen is to keep a **lexicon** of possible ambiguous tokens so we know how to separate them.

2.2 Sentence splitting

Sentence splitting is the process of splitting up a text into sentences. This steps is going to be important when computing co-occurrence matrices with the sliding window. The reason is that when using the sliding window we don’t want to capture the end of one sentence and beginning of the next one in the same context as they might not carry the same context as sentences are naturally boundaries that also we, humans, respect and take into account when reading and processing text.
Sentence splitting has inherently similar problems as does tokenization as punctuation signs can be ambiguous depending on the position and the tokens surrounding the punctuation sign.

2.3 MWE

MWE or Multi-Word Expressions are sequences of words that when grouped together carry a meaning, but when standing alone do not. An example of a MWE is “deep learning”. We see that the word “deep” and “learning” on their own carry other meanings in contrast when putting

the together into a single phrase. For the purposes of this paper we need to keep these expressions together and not split them. The reason is that we are extracting keywords and keyphrases, and the keyphrase to be extracted from a paper that is written on the topic of "deep learning" is "deep learning", not "deep" and "learning".
We expect when dealing with scientific papers, that they are going to have many MWE and possibly new, non-existent MWE. This makes the tokenization problem ever more harder because these tokens should be kept together. One solution is to again keep a MWE lexicon. Another solution, which was found after an inspection of the dataset is that novel MWE usually are written with upper-case letters and they have an abbreviation after the MWE in parenthesis (e.g. "Introduction to Large Language Models (LLMs)") and this fact can be leveraged when dealing with MWE that are not present in the MWE lexicon.
Let us see an example for MWE tokenization:

We are discussing Machine Learning (ML) models in San Francisco. $\rightarrow$
[We, are, discussing, Machine Learning, (, ML,), models, in, San Francisco, .]

2.4 Token removal based on PoS tags

This step is specific for our task, but PoS (Part of Speech) is a regular NLP concept. **Part of Speech** tagging is the process of giving a grammatical category tag to words based on their syntactic and semantic meaning. Basically this step assigns syntactic tags to words i.e. is a word a noun, verb, adjective, etc. The question is why do we need this step?
The reason is the nature of the task we are doing. Namely, keywords are usually a phrase of an adjective and verb or at most a single noun. Taking this into consideration we can substantially reduce the graph size if we just keep nouns and adjective as the possible graph nodes. There are several ways on getting PoS tags of words, like rule-based methods, statistical methods, hybrid methods, deep learning methods, etc. We are not going into the inner workings of these methods, but just use an existent model for getting PoS tags. One of the problems when computing PoS tags is they highly depend on the context in which they occur as we can have words which are the same but carry different PoS tags. For example the word "developed" can

be a verb and an adjective at the same time as show below:

Japan[Noun] developed[Verb] into[Prep.] a[Det.] developed[Adj.] nation[Noun].

This is the reason why context is very important when assigning PoS tags. Deep learning and statistical methods are usually good at these context specific examples but they require quite large training datasets.
An example of this step can be seen below:

Japan developed into a developed nation. $\rightarrow$ [Japan, developed, nation, .]

2.5 Stopword removal

Stopwords are common words occurring in a language that are usually filtered out during NLP tasks as they do not carry much value for the specific task. For our purposes stopword removal is essential as we are going to have many of them, and in the final ranking they will not be present in the keyphrases. The usual method of stopword removal is to build a stopword lexicon and just remove those words from the input text. An example of stopwords are: “the”, “and”, “in”, etc. Let us see an example for stopword removal on a sentence:

The quick brown fox jumped over the lazy dog. $\rightarrow$ [quick, brown, fox, jumped, lazy, dog, .]

This step will come after the token removal based on PoS tags, as stopwords could carry context specific information for PoS tagging. Even though PoS tagging should take care of stopwords, we will still include this step in case some stopword slip thorough the previous step, as stopword removal is a cheap operation.

2.6 Lemmatization

Lemmatization is an NLP technique to reduce words to their base form, also known as their *root* or *lemma*. This means that words that might be morphologically different have the same root and we want to strip the words to their root form. Some examples of lemmatization is show below:

am, are, is $\rightarrow$ be

horses, horse, horses', horse's $\rightarrow$ horse

develops, developed, developing $\rightarrow$ develop

For lemmatization to be accurate, it usually uses PoS as context is important to determine the grammatical role of a word in a sentence. This step is important as the lemma of a word may vary depending on its PoS. Lemmatization will usually transform verbs, adjectives and adverbs to their base form, but if a noun is present among a set of words (e.g. "developer" in the last example) it will keep the noun how it is as nouns are usually lemmas themselves (but it will of course strip possessive or plural forms). With the use of PoS tags and grammatical rules and dictionaries, words are mapped to their lemmas.
One similar concept to lemmatization is stemming, where in the stemming process words are stripped from their suffixes and prefixes using a predefined set of rules. The problem with stemming is that it might produce non-existent words (e.g. "replacement" $\rightarrow$ "replac") which is not practical for our purposes.
The reason we need lemmatization is that words that might have different forms have to be treated as the same word as in the other case we would have two different nodes in the graph that actually represent the same word. Let us see an example of this:

Graph models were used for modeling words. $\rightarrow$ Graph model be use for model word.

2.7 Normalization

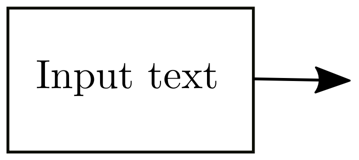

Figure 1: The NLP pipeline

3 Word Embeddings

4 Graph Construction

5 Implementation of Keyword Extraction

6 Testing, Results and Discussion

6.1 Dataset

7 Conclusion

8 Literature

[1] Almeida, Felipe, and Geraldo Xexéo. "*Word embeddings: A survey.*", arXiv preprint arXiv:1901.09069 (2019).
`URL: https://arxiv.org/pdf/1901.09069.pdf`

[2] Lin, Tianyang, et al. "*A survey of transformers.*", AI Open (2022).
`URL: https://www.sciencedirect.com/science/article/pii/S2666651022000146`

[3] Zu, Xian, Fei Xie, and Xiaojian Liu. "*Graph-based keyphrase extraction using word and document embeddings.*", 2020 IEEE International Conference on Knowledge Graph (ICKG). IEEE, 2020.
`URL: https://ieeexplore.ieee.org/abstract/document/9194571`

[4] The Stanford NLP Group
`URL: https://nlp.stanford.edu/`

[5] Jurić Ž. "*Diskretna matematika za studente tehničkih nauka*". ETF Sarajevo, UNSA, 2017.