INF582: INTRODUCTION TO TEXT MINING AND NLP

Lab session 2: Unsupervised Keyword Extraction

Lecture: Prof. Michalis Vazirgiannis
Lab: Antoine Tixier and Kostas Skianis

Friday, January 18, 2019

1 Introduction

In this lab, we will get hands-on experience with the **graph-of-words** representation of text and see how methods from social network analysis can be applied to this representation to **extract keywords**. Keyword extraction is an important NLP task used in many areas such as information retrieval (search engines), summarization, natural language generation, visualization... Today, we will focus on **unsupervised single-document keyword extraction**.

2 Text preprocessing

Before constructing a graph-of-words, the document needs to be cleaned. The standard steps include (1) conversion to lower case, (2) punctuation removal, (3) tokenization, and (4) stopword removal. Additionally, for keyword extraction, research has shown that (5) part-of-speech-based filtering (retaining only nouns and adjectives) and (6) stemming ("winner", "winning", "win" $\rightarrow$ "win") can be useful. These steps are implemented in the `clean_text_simple` function in the `library.py` file.

3 Graph-of-words

There are several ways to represent text as a graph. Today, we will use the classical statistical approach of [6], based on the distributional hypothesis ("We shall know a word by the company it keeps") [3]. This method applies a fixed-size sliding window of size W over the text from start to finish[1]. Each unique term in the document is represented by a node of the graph, and two nodes are linked by an edge if the terms they represent co-occur within the window. Edge weights encode co-occurrence counts. Unlike the vector space model that assumes term independence, word co-occurrence networks capture term dependency, and even term order, if directed edges are used (see Figure 1).

3.1 Your turn

Fill the gaps in the `clean_text_simple` and `terms_to_graph` functions (in `library.py`). Then, use the `gow_toy.py` script to build a graph for the text shown in Figure 1, with a window of size 4. Validate your results by comparing some edges (source, target, and weight) with that of Figure 1. Also, evaluate

[1] interactive demo: `https://safetyapp.shinyapps.io/GoWvis/` [10]

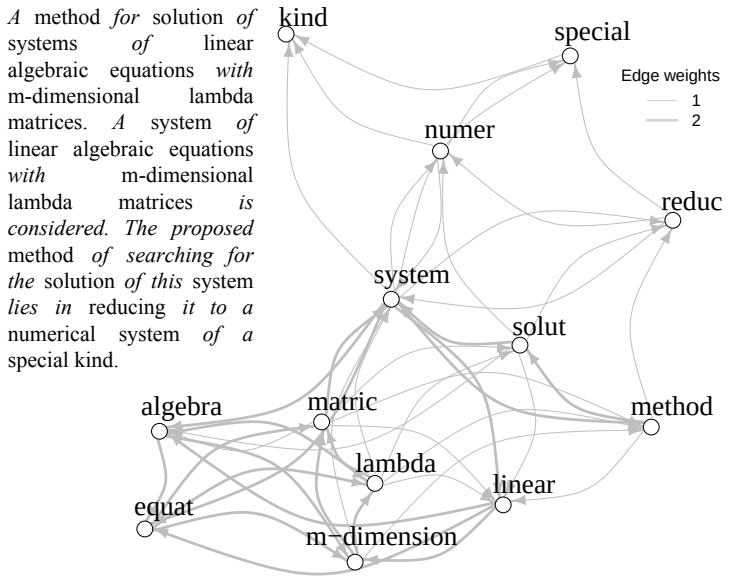

Figure 1: Graph-of-Words representation with POS-based screening, and directed, weighted edges. Non-(nouns and adjectives) in *italic*. Words have been stemmed. $W = 4$

the impact of window size on the density of the graph, defined as $|E|/|V|(|V|-1)$ and accessible via the `.density() igraph` method. What do you observe? Why is the density never reaching 1?

4 Keyword Extraction

4.1 Influential words

In social networks, it was shown that **influential spreaders**, that is, those individuals that can reach the largest part of the network in a given number of steps, are better identified via their **core numbers** rather than through their PageRank scores, betweenness centralities, or degrees [5]. For instance, a less connected person who is strategically placed in the core of a network will be able to disseminate more than a hub located at the periphery of the network, as shown in Figure 2.

Interestingly, taking into account the cohesiveness information captured by graph degeneracy was shown to improve keyword extraction performance [7, 9], meaning that natural language features an important "social" aspect. Keywords can thus be seen as "influential" words of graphs-of-words.

4.2 Graph degeneracy

The concept of graph degeneracy was introduced by [8] with the k-core decomposition technique and was first applied to the study of cohesion in social networks.

A core of order k (or **k-core**) of G is a maximal connected subgraph of G in which every vertex v has at least k incident edges (i.e., neighbors). Edge direction can be taken into account by considering only the incoming or the outgoing edges, but today we will consider all edges.

As shown in Figure 3, the **k-core decomposition** of G is the set of all its cores from 0 (G itself) to k_{max} (its main core). It forms a hierarchy of nested subgraphs whose cohesiveness and size respectively increase and decrease with k. The **core number** of a node is the highest order of a k-core subgraph that contains this node. The main core of G is a coarse approximation of its densest subgraph.

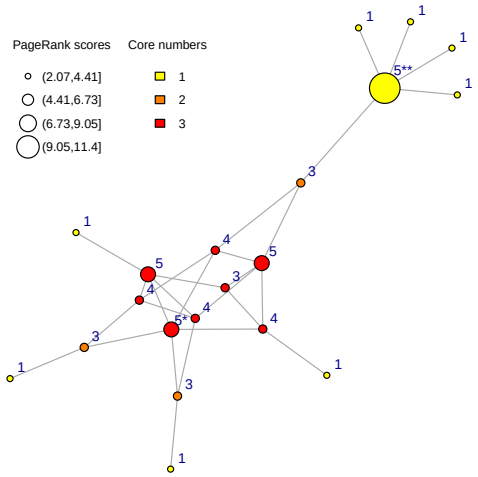

Figure 2: Degree vs PageRank score vs unweighted core number. Node labels indicate degrees. Nodes $*$ and $**$ both have same degree (5) and high PageRank numbers (resp. in $(6.73, 9.05]$ and $(9.05, 11.4]$). However, node $*$ lies in a much more central location and is therefore a much better spreader, which is captured by its higher core number (3 vs 1) but not by degree or PageRank (the PageRank score of node $**$ is even greater than that of node $*$).

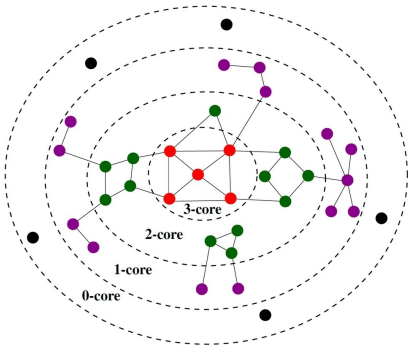

Figure 3: Illustration of the k-core decomposition.

Algorithm 1 show the *unweighted* [8] k-core decomposition algorithm. It implements a pruning process that removes the lowest degree node at each step, and can be implemented with complexity linear in the number of edges [2].

Algorithm 1 unweighted k-core decomposition

Input: graph $G = (V, E)$
Output: core numbers $c(v), \forall v \in V$

$i \leftarrow 0$
while $|V| > 0$ **do**
  while $\exists v : degree(v) \leq i$ **do**
    $c(v) \leftarrow i$
    $V \leftarrow V \setminus \{v\}$
    $E \leftarrow E \setminus \{(u, v) | u \in V\}$
  end while
  $i \leftarrow i + 1$
end while

4.3 Your turn

Fill the gaps in the `unweighted_k_core` function in `library.py` to implement Algorithm 1. Hints: use the `.strength()` and `.delete_vertices()` `igraph` methods. The documentation of all methods is available at `https://igraph.org/python/doc/igraph.GraphBase-class.html`. Note that `.delete_vertices()` automatically removes the edges incident on the node deleted.

Then, go back to the `gow_toy.py` script to decompose the graph shown in Figure 1. Compare your results with that obtained with the `.coreness()` `igraph` method and Figure 4 below.

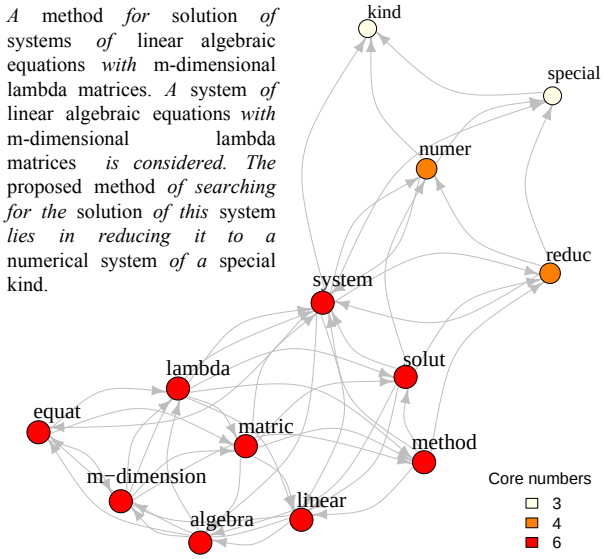

Figure 4: The main core of the graph can be used as the keywords for the document.

5 Keyword extraction

5.1 Data set

We will use the test set of the Hulth 2003 dataset [4], that you can find inside the `data\Hulth2003testing\` folder. This dataset contains 500 scientific paper abstracts. For each abstract in the (`abstracts\` folder), human annotators have provided a set of keywords (`uncontr\` folder), that we will use as ground truth. The keywords were freely chosen by the annotators and some of them may not appear in the original text. Thus, getting a perfect score is impossible on this dataset.

5.2 Baselines

We will evaluate the performance of the k-core-based approach against that of PageRank (applied on the same graph-of-words) and TF-IDF. For each baseline, the top $p = 33\%$ percent nodes will be retained as keywords.

5.3 Performance evaluation

We will use macro-averaged[2] precision, recall and F1 score as our performance metrics. Precision can be seen as the *purity* of retrieval while recall measures the *completeness* of retrieval. Finally, the F-1 score

[2]computed separately for each document and averaged at the collection level.

is the harmonic mean of precision and recall.

$$\text{precision} = \frac{tp}{tp + fp} \qquad \text{recall} = \frac{tp}{tp + fn} \qquad \text{F1-score} = 2\frac{\text{precision.recall}}{\text{precision} + \text{recall}}$$

Where tp, fp and fn respectively denote the number of true positives (the keywords returned by the system which are also part of the gold standard), false positives (the keywords returned by the system which are not part of the gold standard), and false negatives (the keywords from the gold standard that are not returned by the system).

5.4 Your turn

Put everything together by filling the gaps in the `keyword_extraction.py` file and in the `accuracy_metrics` function (in `library.py`). For the baselines, you can use the `.pagerank()`[3] igraph method, and for tfidf, the `TfidfVectorizer`[4] function from the `scikit-learn` library. What can you say about the performance of the different approaches? How could the k-core approach be improved?

References

[1] Batagelj, V., & Zavernik, M. (2002). Generalized cores. arXiv preprint cs/0202039.

[2] Batagelj, Vladimir, and Matjaz Zaversnik. "An O (m) algorithm for cores decomposition of networks." arXiv preprint cs/0310049 (2003).

[3] Harris, Z. S. (1954). Distributional structure. Word, 10(2-3), 146-162.

[4] Hulth, A. (2003, July). Improved automatic keyword extraction given more linguistic knowledge. In Proceedings of the 2003 conference on Empirical methods in natural language processing (pp. 216-223). Association for Computational Linguistics.

[5] Kitsak, M., et al. "Identification of influential spreaders in complex networks." Nature physics 6.11 (2010): 888-893.

[6] Mihalcea, R., & Tarau, P. (2004, July). TextRank: Bringing order into texts. Association for Computational Linguistics.

[7] Rousseau, F., Vazirgiannis, M. "Main core retention on graph-of-words for single-document keyword extraction." European Conference on Information Retrieval. Springer, Cham, 2015.

[8] Seidman, S. B. (1983). Network structure and minimum degree. Social networks, 5(3), 269-287.

[9] Tixier, A., Malliaros, F., and Vazirgiannis, M. "A graph degeneracy-based approach to keyword extraction." Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016.

[10] Tixier, Antoine, Konstantinos Skianis, and Michalis Vazirgiannis. "Gowvis: a web application for graph-of-words-based text visualization and summarization." Proceedings of ACL-2016 System Demonstrations (2016): 151-156.

[3] `http://igraph.org/python/doc/igraph.Graph-class.html#pagerank`

[4] `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`