ADVANCED LEARNING FOR TEXT AND GRAPH DATA

Lab session 1: unsupervised keyword extraction

Lecture: Prof. Michalis Vazirgiannis
Lab: Antoine Tixier and Jean-Baptiste Remy

Monday, October 7, 2019

This lab includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit here a **.zip** file (max 10MB in size) containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet named `firstname_lastname.pdf`, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. Although you are allowed to work in teams, **one submission is required for each student. The deadline for this lab is October 13, 2019 12:59 PM**. No extension will be granted.

---

1 Learning objective

In this lab, you will learn how methods from social network analysis can be applied to word co-occurrence networks to **extract keywords**. Keyword extraction is a fundamental NLP task used in many areas like information retrieval (search engines), summarization, natural language generation, visualization... Today, we will focus on **unsupervised single-document keyword extraction**.

Notation: in what follows, $G(V, E)$ is a graph with $|V|$ nodes (or vertices) and $|E|$ edges (or links). $N(v, U)$ designates the immediate neighbors of v in $U \subset V$.

Igraph: the nodes and edges of a graph `g` can be accessed collectively or individually (through indexing), along with their attributes, such as 'name' or 'weight', via, e.g., `g.vs['name']` or `g.es[index]['weight']`. Documentation of all methods and functions can be found at `http://igraph.org/python/doc/igraph.GraphBase-class.html`.

2 Text preprocessing

Before constructing a word co-occurrence network, 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, (5) part-of-speech-based filtering (e.g., retaining only nouns and adjectives) and (6) stemming ("winner", "winning", and "win" → "win") can be useful. These steps are implemented in the `clean_text_simple` function, found within the `library.py` file.

3 Word co-occurrence networks

There are many 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 are co-occurrence counts. Unlike the vector space model that assumes term independence, this representation captures term dependency, and even term order, if directed edges are used (see Fig. 1).

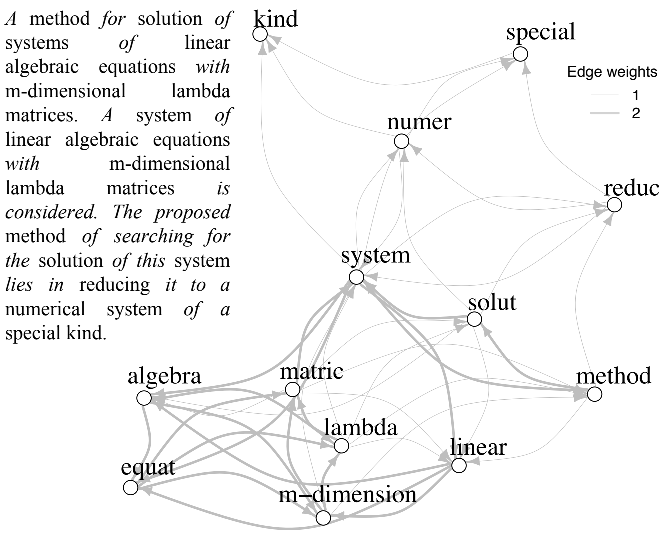

Figure 1: Word co-occurrence network example. Non-(nouns and adjectives) in *italic*. Words have been stemmed. $W = 4$.

> **Task 1**
> Fill the gaps in the `clean_text_simple` and `terms_to_graph` functions (in `library.py`).

> **Task 2**
> Use the `gow_toy.py` script to build a graph for the text of Fig. 1, with a window of size 4. Validate your results by comparing some edges (source, target, and weight) with Fig. 1.

> **Question 1 (2 points)**
> Evaluate the impact of window size on the density of the graph, defined as $\frac{|E|}{|V|(|V|-1)}$ for a directed graph and accessible via the `.density()` `igraph` method. What do you observe?

Note: the density never reaches 1, even when the window includes all terms in the document, because we work with *directed* graphs. To be complete, a directed graph must have two edges between each node (one in each direction). On the other hand, for undirected graphs, the density is equal to $\frac{2\times|E|}{|V|(|V|-1)}$, and there can be at most one edge between two nodes.

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

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 Fig. 2.

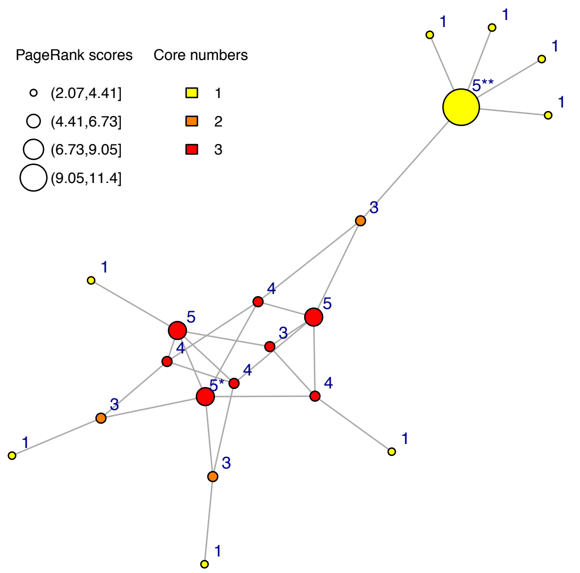

Figure 2: Degree vs. PageRank vs. unweighted core number. Node labels indicate degrees. Nodes $*$ and $**$ both have same degree (5) and high PageRank scores (resp. in $(6.73, 9.05]$ and $(9.05, 11.4]$). However, node $*$ lies in a much more central location and is therefore a much better spreader, which 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 $*$).

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

k-core 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 degree k (i.e., k neighbors).

k-core decomposition As shown in Fig. 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.

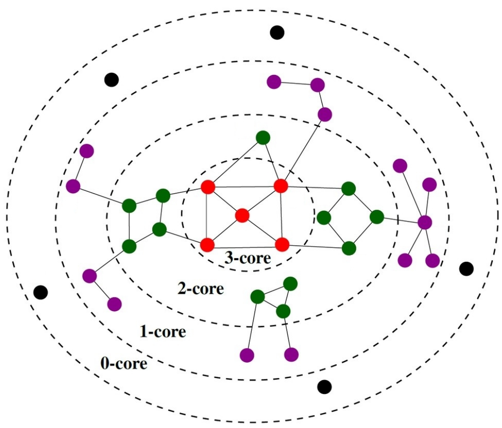

Figure 3: Unweighted k-core.

Algorithm 1 shows the *unweighted* k-core algorithm. It involves a pruning process that removes the lowest degree node at each step, where the degree of a node is simply its number of immediate neighbors. By using the sum of the weights of the incident edges as the degree, we obtain the *weighted* k-core algorithm. The unweighted and weighted k-core algorithms can be implemented with very affordable time complexities $\mathcal{O}(|E|)$ [2] and $\mathcal{O}(|E|\log|V|)$ [1], by using specific strategies and data structures.

Algorithm 1 k-core decomposition

Input: graph $G = (V, E)$
Output: dict of core numbers c

$p \leftarrow \{v : degree(v)\}\ \forall v \in V$
while $|V| > 0$ **do**
  $v \leftarrow$ element of p with lowest value
  $c[v] \leftarrow p[v]$
  neighbors $\leftarrow \mathcal{N}(v, V)$
  $V \leftarrow V \setminus \{v\}$
  $E \leftarrow E \setminus \{(u, v) | u \in V\}$
  for $u \in$ neighbors **do**
    $p[u] \leftarrow \max(c[v], degree(u))$
  end for
end while

Question 2 (10 points)
What is the time complexity of our naive version of the k-core algorithm (Alg. 1), in the unweighted case?

Notes: use big $\mathcal{O}$ notation. Evaluate each line one by one before deriving the overall complexity of the algorithm. You may assume that G is represented by its adjacency list (a list of lists containing the neighbors of each node), available in RAM, and introduce a variable representing the average number of neighbors of a node.

Task 3
Fill the gaps in the `core_dec` function in `library.py` to implement Algorithm 1. Use the `.strength()`[2] and `.delete_vertices()` `igraph` methods.

Note 1: `.delete_vertices()` automatically removes the edges incident on the node(s) deleted.
Note 2: to get weighted degrees, you need to use the `weights` argument of `.strength()`.

Task 4
Go back to the `gow_toy.py` script to decompose the graph shown in Fig. 1. For unweighted k-core, compare your results with the `.coreness()` `igraph` method and Fig. 4 below.

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 consider 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 graphs) and the vector space representation with TF-IDF coefficients. For each baseline, the top $p = 33\%$ percent nodes will be retained as keywords.

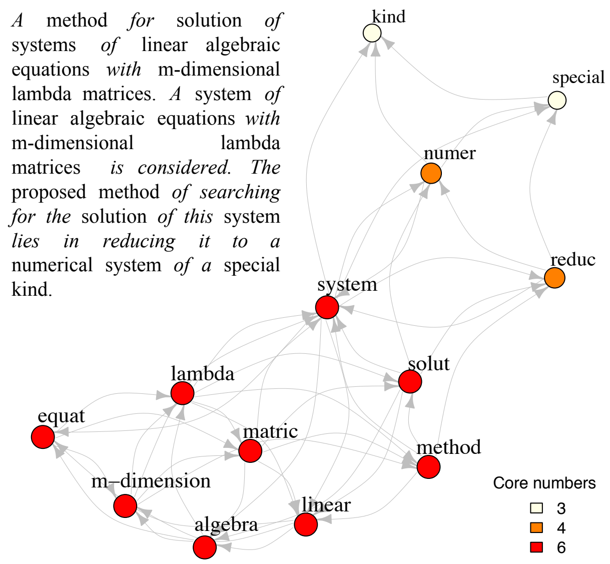

Figure 4: The main core of the graph can be used as the keywords for the document.

Assumption: both for k-core and weighted k-core, we will extract the *main core* of the graph as keywords.

5.3 Performance evaluation

We will evaluate the performance of the different techniques in terms of macro-averaged precision, recall and F1 score. Precision can be seen as the *purity* of retrieval while recall measures the *completeness* of retrieval. Finally, the F-1 score is the harmonic mean of precision and recall. More precisely, these metrics are defined as follows:

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

where tp, fp and fn respectively denote the number of true positives (the keywords returned by the system which are also part of the ground truth), false positives (the keywords returned by the system which are *not* part of the ground truth), and false negatives (the keywords from the ground truth that are *not* returned by the system). Finally, macro-averaging consists in computing the metrics for each document and calculating the means at the collection level.

Task 5
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, use the `.pagerank()`[3] igraph method, and for tfidf, the `TfidfVectorizer`[4] function from the `scikit-learn` library.

Question 3 (2 points)
What can you say about the performance of the different approaches?

Question 4 (4 points)
What are the respective (dis)advantages of (un)weighted k-core?

Question 5 (2 points)
How could these approaches be improved?

References

[1] Vladimir Batagelj and Matjaž Zaveršnik. Generalized cores. *arXiv preprint cs/0202039*, 2002.

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

[3] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[4] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics, 2003.

[5] Maksim Kitsak, Lazaros K Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H Eugene Stanley, and Hernán A Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888, 2010.

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

[7] François Rousseau and Michalis Vazirgiannis. Main core retention on graph-of-words for single-document keyword extraction. In *European Conference on Information Retrieval*, pages 382–393. Springer, 2015.

[8] Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[9] Antoine Tixier, Fragkiskos Malliaros, and Michalis Vazirgiannis. A graph degeneracy-based approach to keyword extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1860–1870, 2016.

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