ADVANCED LEARNING FOR TEXT AND GRAPH DATA

MASTER DATA SCIENCE

Lab 4: Graph-based approaches to NLP

Fragkiskos Malliaros, Antoine Tixier and Michalis Vazirgiannis
February 18, 2016

1 Description

The goal of this lab is to study a graph-based approach to two central tasks in the field of NLP: keyword extraction and document classification. For each task, we briefly present the theory and describe the tasks that need to be performed.

Note 1: the scripts provided under the `code` directory make extensive use of the python igraph library. You should already have installed it, but if not, instructions can be found here.

Note 2: covering both sections in 2 hours might be difficult. Therefore, feel free to start with the section of your choice.

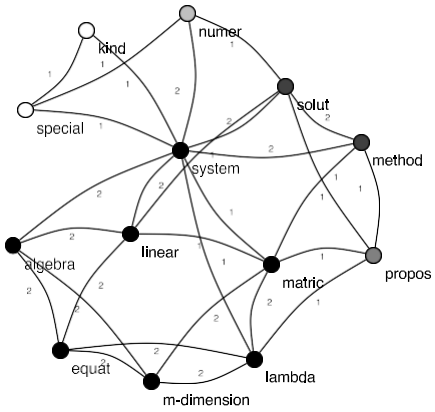
1 Keyword extraction

Keywords are ubiquitous in our everyday life, from looking up information on the Web via a search engine bar, to online ads matching browsing history. Researchers also assign keywords to their papers for indexing but also to express the gist of their content. Indeed, keyword extraction is closely related to text summarization. The task of keyword extraction can be (a) single or multidocument depending on whether the input is from a single document or multiple ones; (b) extractive or abstractive depending on whether the extracted content is restricted to the original text or not; (c) generic or query-based or update depending on whether the extracted keywords are generic, impacted by some user request, or based on prior latent information (e.g., browsing history); and finally (d) unsupervised or supervised depending on whether the extraction process involves labeled data. In today's lab, we focus on *unsupervised generic extractive single-document* keyword extraction.

1.1 Graphs-of-words

Graph representations of textual documents have been proposed for more than a decade (e.g., Mihalcea and Tarau 2004) Unlike earlier approaches assuming term independence, such as the bag-of-words, graphs-of-words offer an information-rich way of encoding text, by capturing for instance term dependence and term order. Recently, graphs-of-words have achieved state-of-the-art performance on various machine learning tasks, such as information retrieval (Rousseau and Vazirgiannis 2013), keyword extraction (Rousseau and Vazirgiannis 2015), and document

categorization (Rousseau et al. 2015). More precisely, a graph-of-words is a graph whose vertices represent unique terms in the document and whose edges capture some meaningful syntactic (grammar), semantic (synonymy), or statistical similarity between terms. Following the aforementioned references, in this lab we assume that two vertices are linked by an edge if and only if the two words they represent co-occur in text within a sliding window of predetermined fixed size. This is a statistical approach, as it links all co-occurring terms without considering their meaning or function. The underlying assumption (similar to the Markov assumption in time series) is that dependence only exists between words found close to each other. Edges may be assigned integer weights based on the number of co-occurrences, as shown in Figure 1. Similarly, edges can be directed to encode term order, forward edges matching the natural flow of the text.



WK-core		PageRank	
system	6	**system**	1.93
matric	6	**matric**	1.27
lambda	6	solut	1.10
linear	6	**lambda**	1.08
equat	6	**linear**	1.08
algebra	6	**equat**	0.90
m-dim…	6	**algebra**	0.90
method	5	**m-dim…**	0.90
solut	5	propos	0.89
propos	4	method	0.88
numer	3	special	0.78
specia	2	**numer**	0.74
kind	2	kind	0.55

Figure 1. Graph-of-words with main core and PageRank keyword extraction. Node color indicates the highest core a vertex belongs to (i.e., its core number), from 2-core (white) to 6-core (black). The graph was built on document #1938 from the Hulth (2003) data set (see below):

```
A method for solution of systems of linear algebraic equations with m-
dimensional lambda matrices A system of linear algebraic equations
with m-dimensional lambda matrices is considered. The proposed method
of searching for the solution of this system lies in reducing it to a
numerical system of a special kind.
```

Note: an interactive web application illustrating graphs-of-words can be found here, and is very useful to develop a good intuition for the concept.

1.2 K-core

The k-core of a graph corresponds to the maximal connected subgraph whose vertices are at least of degree k within the subgraph. The core number of a vertex is the highest order of a core that contains this vertex. The core of maximum order is called the main core. It corresponds to a fast and good (although not perfect) approximation of the densest, or most cohesive connected component of the graph. The set of all the k-cores of a graph (from the 0-core to the main core) forms what is called the k-core decomposition of a graph. Thanks to Batagelj and Zaveršnik (2002), the k-core

decomposition of a weighted graph can be computed in linearithmic time and linear space using a min-oriented binary heap to retrieve the vertex of lowest degree at each iteration (n in total). In the unweighted case, the algorithm is linear in time.

1.3 To-do list

Each of the steps below is implemented either entirely or partially in the `keyword_extraction.py` file that can be found under the `code` directory. The custom functions used can be found in the `library.py` file.

- We will work on a standard, publicly available dataset: Hulth (2003). This data set contains 500 abstracts from the Inspec database, along with manually assigned keywords (gold standard). It can be found under the `data` directory. Since our approach is unsupervised, it is not necessary to split the data into training and testing sets.

- Apply the following standard preprocessing steps to the abstracts: (1) tokenization; (2) part-of-speech annotation and selection (keep only nouns and adjectives); (3) stopwords removal; and (4) stemming. Build a graph-of-words from each processed abstract. Because it was consistently reported to give superior results, we advise using a sliding window of size 4. Note that this corresponds to 3 in the `terms_to_graph()` function provided in the `library.py` file.

- Similarly to what is shown in Figure 1, extract as keywords for each document the main core of its graph-of-words. Experiment with both weighted and unweighted k-core decomposition.

- Evaluate the performance of weighted and unweighted main core extraction against the golden (manually assigned) keywords available for each abstract, in terms of macro-averaged precision, recall and F1 score. Compare their performance to that of two baselines: PageRank and TF-IDF (in each case, retain the top 33% nodes/words). Macro-averaging means that you mean to compute the metrics for each document and average at the collection level. What do you observe?

  Precision, recall and F1 score are defined as follows:

$$\text{precision} = \frac{\text{true positives}}{\text{true positives+false positives}} \qquad \text{recall} = \frac{\text{true positives}}{\text{true positives+false negatives}}$$

$$\text{F1 score} = 2 \,.\, \frac{\text{precision} \,.\, \text{recall}}{\text{precision} + \text{recall}}$$

2 Document classification

Text categorization is one of the most active research areas in NLP. It has a variety of real-world applications such as sentiment analysis, opinion mining, email filtering, etc. Given the current data overflow, especially of textual type, the needs for efficient automated text classification solutions have become more pressing than ever.

The most common pipeline for text classification is the vector space representation followed by TF-IDF term weighting. With this approach, each document is viewed as a point in a sparse space where each dimension (or feature) is a unique term in the corpus. The set of unique terms is called the vocabulary. If we assume that there are m documents $\{d_1 \dots d_m\}$ in the collection and n unique terms $T = \{t_1 \dots t_n\}$ in the corpus (T being the vocabulary), each document can be represented as a vector of n term weights (i.e., the weights are the coordinates of the document in vocabulary space). A classifier is then trained on the available documents (i.e., on a document-term matrix of dimension m by |T|) and subsequently used for classifying new ones. In this lab, we consider the classifier as a black-box, and focus only on improving categorization performance by coming up with better term weights. We will compare two ways of computing these weights.

2.1 TF-IDF

A given document d loads on each dimension of the feature space (each term t) according to the following formula, known as the pivoted normalization weighting (Singhal et al. 1999):

$$\text{tf} - \text{idf}(\text{t}, \text{d}) = \frac{1 + \ln(1 + \ln(\text{tf}(\text{t}, \text{d})))}{1 - \text{b} + \text{b} \times \frac{|\text{d}|}{\text{avgdl}}} \times \text{idf}(\text{t}, \text{D})$$

Where $\text{tf}(\text{t}, \text{d})$ is the number of times term t appear in document d, |d| is the length of document d, avgdl is the average document length across the corpus, $\text{b} = 0.2$, and $\text{idf}(\text{t}, \text{D}) = \log(\text{m} + 1/\text{df}(\text{t}))$, with df(t) the number of documents in which the term t appears.

The intuition behind this scoring function is that frequent words in a document are representative of that document as long as they are not also very frequent at the corpus level. Note that for all the terms that do not appear in d, the weights are null. Since a given document contains only a small fraction of the vocabulary, most of its coordinates in the vector space are null, leading to a very sparse representation, which is a well-known limitation of the vector space model that motivated (among other things) word embeddings, that we will address in the next lab session.

2.1 TW-IDF

We propose to leverage the graph-of-words representation of a document to derive a new scoring function following (Rousseau and Vazirgiannis 2013):

$$\text{tw} - \text{idf}(t, d) = \frac{\text{tw}(t, d)}{1 - b + b \times \frac{|d|}{\text{avgdl}}} \times \text{idf}(t, D)$$

Where $\text{tw}(t, d)$ is some graph-of-words-based score for term t (for the graph-of-words corresponding to document d), and $b = 0.003$ (the remainder of the equation is the same as for TF-IDF).

Various node centrality criteria are intuitively good candidates for tw(t,d):

- Normalized degree centrality:

$$\text{degree(node)} = |\text{neighbors(node)}|/(|\text{vertices in graph}| - 1)$$

  Note that in its weighted version, degree centrality sums up the weights of the edges incident to the node instead of simply counting the number of incident edges. Keep in mind that both igraph implementations are not normalized.

- Closeness centrality:

$$\text{closeness(node)} = (|\text{vertices in graph}| - 1) \Big/ \sum_{node_i \epsilon\, graph} dist(node, node_i)$$

  Closeness centrality is defined as the inverse of the average shortest path distance from the considered node to the other nodes in the graph. As opposed to degree centrality, closeness is a global metric, in that it aggregates information from the entire graph. Note that the igraph implementation has an argument for normalization. If set to True, the function computes exactly the quantity above.

Other centrality criteria may also be highly relevant but will not be explored today due to time issues.

2.3 To-do list

Each of the steps below is implemented either entirely or partially in the `document_classification.py` file that can be found under the `code` directory. The custom functions used can be found in the `library.py` file.

- We will work on a standard, publicly available dataset: WebKB. This data set corresponds to academic webpages belonging to four different categories: (1) project, (2) course, (3) faculty, and (4) students. It contains 2,803 documents for training and 1,396 for testing. The

training and testing sets can be found under the `data` directory. They already have been preprocessed: stopwords and words less than 3 characters in length have been removed, and Porter' stemming has been applied, so that only the root of each term remains.

- Create a TF-IDF and TW-IDF representation for each document in the training set. For TW-IDF, build a graph-of-words for each document (window size of 4, note that this corresponds to 3 in the `terms_to_graph()` function found in `library.py`) and consider both the weighted and unweighted versions of degree and closeness.

- Train a basic classifier (e.g., SVM with linear kernel) for each representation.

- Use the classifiers to predict the labels of the document in the testing set. Note that the documents in the testing set should be represented in the space made of the unique terms of the *training* set only (words in the testing set absent from the training set are disregarded). Similarly, the average document length and the inverse document frequency used should be the ones computed on the *training* set.

- Compare performance….

3 References

Rousseau, F., & Vazirgiannis, M. (2015). Main core retention on graph-of-words for single-document keyword extraction. In Advances in Information Retrieval (pp. 382-393). Springer International Publishing.

Rousseau, F., & Vazirgiannis, M. (2013, October). Graph-of-word and TW-IDF: new approach to ad hoc IR. In Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (pp. 59-68). ACM.

Rousseau, F., Kiagias, E., & Vazirgiannis, M. Text categorization as a graph classification problem. In ACL (Vol. 15, p. 107).

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

Batagelj, V., & Zaveršnik, M. (2002). Generalized cores. arXiv preprint cs/0202039.

Singhal, A., Choi, J., Hindle, D., Lewis, D. D., & Pereira, F. (1999). At&t at TREC-7. *NIST SPECIAL PUBLICATION SP*, 239-252.