Key Words Extraction from News

Aleksandr Safin, Ivan Rodin, Maxim Kaledin

January 15, 2018

Abstract

In this project several techniques are proposed for keywords extraction problem from news. News stories is a very special type of data: they are typically small texts with lots of named entities and there is no much data for supervised methods. In this project we propose methods based on Latent Semantic Analysis, topic modelling, different graph methods using Graph of Words representation of text. The methods are tested on 500KN-Crowd dataset, most of them do not require training dataset, have simple architecture and show very reasonable results during manual and automatic testing.

Introduction

The keywords is a well-known problem in linguistics and natural language processing which has a huge variety of applications. People can use keywords to decide whether the text is relevant, briefly catch the main idea. Machines can use them in recommender system (to recommend you goods based on the reading history), search engines (document indexing).

News stories is a very specific type of texts. They are relatively small (usually no more than 600-700 words) and have a large number of named entities (people, places, organizations,...). Moreover, there is no much (freeware) data for statistics approaches to operate effectively. Here we can propose other unsupervised methods to roughly estimate keywords. The challenge is to use as few number of input texts as possible (usually, only one) and produce results very fast.

The problem of keywords can be solved using modern machine learning approaches and statistical methods. Our goal is to investigate other class of approaches, i.e. heuristic methods arising from graph clustering (known as community detection in social network analysis), random walk models (TextRank) and methods of topic modeling. All code and relevant documents can be found on project's Github [1].

Problem statement

Input texts usually are relatively small (up to 600-700 words), we should produce the *keywords* based on it. The *keywords* are the most important words which could help to roughly estimate the gist of text without reading it. In different approaches the set of keywords is defined differently.

Quality Estimation

We use Jaccard index and 500KN-Crowd dataset [2] with manually extracted keywords to roughly measure the quality of prediction:

$$K_J = \frac{|A \cap B|}{|A \cup B|},$$

where A is set of extracted keywords B are gold standard keywords. But since our test data has about 30-40 keywords on each text, this metrics may be very bad, however the keywords turns out to be reasonable. So we also consider average number of guessed words (in comparison with gold standard) as another quality measure.

Matrix factorization-based approach (A. Safin)

Approximation

Of course, one could propose simple baseline, like consider TF-IDF document-term matrix and then choose the words corresponding to the top highest scores. But the question is could we do better.

Since keywords of the document reflect essential part of the whole document, then we simply consider here that the task of keywords extraction from the documents could be formulated as the task of finding low-rank approximation of the term-document matrix.

Then, to extract keywords for each document simply means to choose k (for instance) words (or bigrams, if they are used it the bag of words vector) corresponding to the highest values in the approximated matrix.

So, the task could be formally described as:

$$\underset{\text{rank}A_r=r}{\text{minimize}} \|A - A_r\|_F^2,$$

where the r is the parameter of the model which could be optimized to provide acceptable results for key word extraction.

One approach to obtain such approximation is to use SVD and then truncate it up to r largest singular values [3].

Another approach utilizes the following idea, that term-document matrix (probabilities of each word to be in the particular document) could be represented is a product of two matrices such that $p(w_i|d_j) = \sum_{k=1} p(w_i|t_k)p(t_k|d_j)$. In other words, one could consider the initial term-document matrix A as $A = FH$, where F is matrix determined by $p(w_i|t_k)$ and $H_{kj} = p(t_k|d_j)$. So, the most weighted elements in the j^{th} column of the matrix F reflects the keywords for the j^{th} topic. And matrix H reflects the probability of each topic for every document. Because we want consider the elements of H and F matrices as a probabilities (or proxy values for probabilities), it is natural to require that F and H would be non-negative.

The task of Non-negative Matrix Factorization [4] (NMF) could be formalized as:

$$\underset{F\geq 0, H\geq 0}{\text{minimize}} \|A - FH\|_F^2$$

Keywords extraction

So, to extract keywords based on NMF or SVD approximation for the whole set of documents we simply choose the words corresponding to the highest scores.

But there could be a pretty simple modification. Let us construct the TF-IDF matrix for each document, namely treating each sentence as document. So, what we obtain is TF-IDF sentence-term matrix. Then after matrix factorization, we choose the words corresponding to the highest scores in the first row of the matrix which reflects topic-word relations.

In the SVD-based factorization this means that we choose the most important topic among all sentences.

Another approach to extract keywords is to utilize the LSA approach, namely using NMF for finding the set of most appropriate keywords for the topics seemed to be in the document (according to NMF). And then select appropriate keywords which are relevant to the document.

PageRank for the graph of words

The idea is to find PageRank for the graph of words and it was proposed in [5] and is referred to as TextRank. Consider a graph $G = (V, E)$ of words, namely V is the set of words and the weight of the edge e_{ij} equals to the number of cases when the word j occurs after the word i within the window of the fixed size w_{size}. Thus, we assign to each word a score corresponding to the PageRank score. Then, we just output the words with the highest PageRank score.

The Table 2 shows that TextRank works more slower than matrix factorization methods. This is simply because of the a lot of preprocessing of natural language text and construction of the graph of words.

Approach	Keywords
SVD-based	'nakamoto' 'use' 'secure' 'satoshi' 'cryptography' 'regulation' 'traditional' 'transactions' 'successfully' 'making'
NMF-based	'19' 'contract' 'price' 'reference' 'set' '18' '805' 'january' 'sponsored' 'exchange'
TextRank	'bitcoin' 'cme' 'future' 'contract' 'exchange' 'price' 'photo' 'margin' 'cboe' 'currency'

Table 1: Comparison of results for different approaches

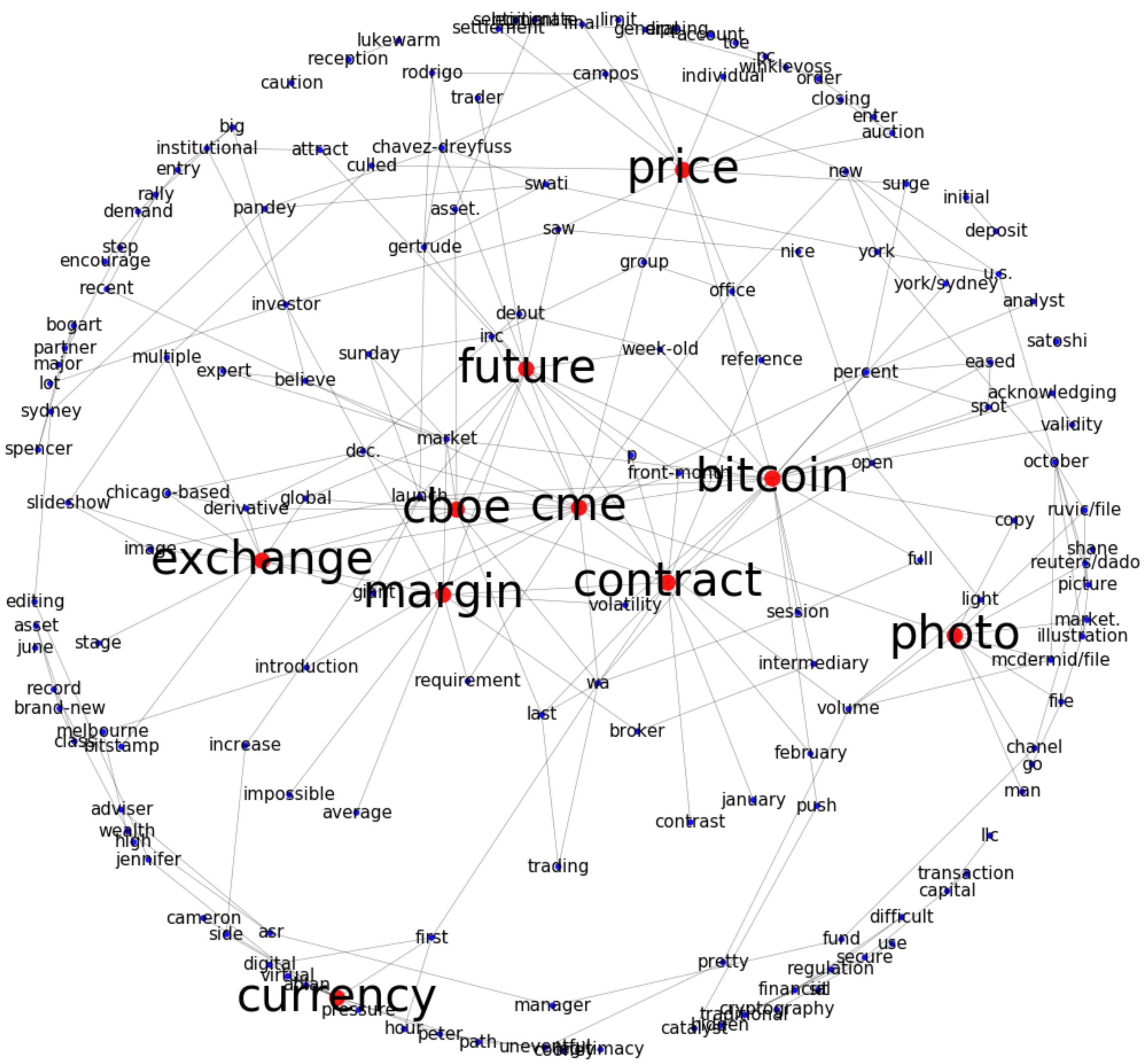

Figure 1: Reuters news story, ”Bitcoin hits bigger stage as exchange giant CME launches futures”, TextRank-based keywords

Approach	Average time (secs / text)
SVD-based	0.0047
NMF-based	0.0237
TextRank	0.418

Table 2: Time performance comparison of the approaches

Extracting Keywords using Optimization of Modularity(M. Kaledin)

Let $G = (V, E)$ be a graph of words which is built from raw text, where V is a set of words (we leave only nouns and adjectives from original text). Two vertices u, v are connected if u is followed by v in the text within sliding window w, this graph can be directed (used for pageRank in this project) or undirected, I consider the latter case. Here is a brief example.

Our aim is to effectively cluster this graph (in social network analysis this process is called *community detection*), then assign keywords by choosing the largest cluster.

Modularity Approaches

The first algorithm suggesting to use modularity was Girvan-Newman's algorithm [6]. It suggests to optimize the *modularity* Q of partition of the graph $G = (V, E)$, $|V| = n$, $|E| = m$. Suppose, we search the best graph partition into two communities (subgraphs with large density inside and less outside edges), let $s_i = 1$ if vertex i belongs to cluster 1 and -1 otherwise. The article suggests to consider the difference between the actual number of edges between two vertices and the expected one if they were randomly generated with uniform distribution. Thus

$$Q = \sum_{u,v \in V:\ u \neq v} \left(A_{ij} - \frac{k_u k_v}{2m} \right) (s_i s_j + 1)$$

defines the modularity of partition. It was noted that this approach can be generalized on case of several communities [7] with

$$Q = \sum_{u,v \in V:\ u \neq v} \left(A_{ij} - \frac{k_u k_v}{2m} \right) \delta(c_u, c_v),$$

where c_u, c_v denote community assignments and

$$\delta(c_u, c_v) = \begin{cases} 0 & \text{if } c_u = c_v \\ 1 & \text{otherwise} \end{cases}.$$

Also it is possible to use the hierarchical approach with 2-community clustering [8]. This can be solved as LP with $O(n^3)$ transitivity constraints:

$$\max_{\delta_{uv}} Q, s.t.$$
$$\delta_{uv} + \delta_{vw} \le 2\delta_{uw}\ \forall u, v, w \in V,$$

which is very hard even if $n \sim 10^2$. Thus direct optimization of modularity is exhausting (NP-hard with ILP, or worse with quadratic IP), so several sub-optimal algorithms are suggested.

Girvan-Newman Algorithm

Algorithm make useof *betweeness centrality* of edge in a graph: it is a number of shortest paths which include this edge. The idea is that the most "betweeness" edges connect communities. Algorithm sequentially delete the most central edge and recalculates centralities until no edges [6]. This method is implemented in NetworkX.

Louvain Method

This is quite effective heuristic algorithm for community detection proposed in [9]. On first step all vertices are assigned to its own community, then iteratively we try to reassign each vertex to its neighbouring community. The community of choice is such that it gives the largest increase of modularity. The second step repeats the first step but each node is a detected community and two communities are connected if they have edges connecting them. This method is implemented in python-louvain library as best_partition.

Other Heuristic Approaches

Fluid-Based Technique for Community Detection

I briefly describe also another heuristic algorithm proposed in [10]. It chooses the number of community fluids k (in our framework I choose it to get maximum modularity), place their start points in random vertices, then run the fluids. Each vertex updates its community with respect to update rule. Denote by $d(c) = 1/|c|$ the density of community c. Compute

$$C_v = \text{Argmax}_c \sum_{w \in v, \Gamma(v)} d(c)\delta(c(w), c),$$

where $c(w)$ is a community assigned to vertex w, $\Gamma(w)$ is the set of neighbours of v. If C_v contains the original community, v stays there, otherwise the new community is chosen among found with uniform distribution. The process is bounded by max_iterations parameter.

This algorithm is implemented in NetworkX library and considered only for comparison with other methods. Unfortunately, it does not work with disjoint graphs, so it is not chosen for the quality estimation. Moreover, it turned out to be very ineffective method on manual experimets.

K-core Technique for Community Detection

Consider also k-core approach for clustering. The maximal subgraph of G of vertices of minimum degree k is called a *k-core* [11]. The k-core corresponding to the largest possible k is called the *main core.* In this approach we should assign keywords as the main core.

Keywords Extraction

Let w denotes window size. Firstly, I tokenize the input text with default tokenizer from NLTK framework. Then simple preprocessing goes:

- Removing punctuation;
- POS-filtering to leave only nouns and adjectives (supplied by NLTK framework);
- Removing stop-words (if any, supplied by NLTK);
- Creating a dictionary from the filtered text, output it and text without punctuation.

Then undirected graph of words is built with window size w (3, by default). After this I apply k-core decomposition to extract the most dense subgraph (known approach, maximal subgraph with all vertices having degree at least k is called a *k-core*, the k-core corresponding to the largest k is called the *main* core). This is done becuse the original graph is very sparse and there are many non-relevant words.

Then we apply community detection algorithm to extract communities and assign keywords as the largest community.

Example of the Keywords extracted

Consider the example with Reuters news story, "Bitcoin hits bigger stage as exchange giant CME launches futures". We have already seen its graph of words for windows size equals 2, here are the community plots. I had to set window size to 5 for fluid algorithm because it does not work with disjoint graphs.

Approach	Keywords
Louvain	'average' 'bitcoin' 'brokers' 'contract' 'contracts' 'contrast' 'experts' 'february' 'front' 'full' 'futures' 'intermediaries' 'january' 'last' 'launch' 'march' 'margin' 'market' 'month' 'old' 'p' 'refers' 'requirement' 'requirements' 'session' 'sunday' 'trader' 'trading' 'validity' 'volatility' 'week'
Girvan-Newman	'validity' 'analysts' 'volatility' 'debut' 'week' 'bigger' 'bitcoin' 'session' 'intermediaries' 'old' 'brokers' 'spot' 'stage' 'launch' 'sunday' 'percent' 'full' 'contracts'
Fluid	'reception' 'analysts' 'reference' 'regulation' 'requirement' 'average' 'big' 'bitstamp' 'secure' 'settlement' 'brokers' 'side' 'step' 'surge' 'trader' 'contract' 'traditional' 'contrast' 'transactions' 'cryptography' 'volatility' 'demand' 'difficult' 'digital' 'entry' 'exchanges' 'february' 'final' 'financial' 'first' 'half' 'hidden' 'higher' 'hour' 'impossible' 'institutional' 'intermediaries' 'investors' 'january' 'last' 'launch' 'light' 'limits' 'lot' 'lukewarm' 'major' 'march' 'margin' 'multiple' 'nice' 'open' 'p' 'percent' 'pressure' 'price'
KCore	'global' 'reuters' 'rodrigo' 'bitcoin' 'brendan' 'swati' 'campos' 'cboe' 'chavez' 'sydney' 'cme' 'contract' 'dado' 'u' 'margin' 'market' 'markets' 'mcdermid' 'volume' 'min' 'month' 'york' 'new' 'dreyfuss' 'october' 'exchange' 'pandey' 'file' 'front' 'photo' 'read' 'futures' 'ruvic' 'gertrude'

Table 3: Comparison of results for different approaches: we can see that the keywords are relevant for all methods except fluid algorithm

The following table and charts show the quality estimates of the algorithms.

Approach	Jaccard's Index	Average number of guessed keywords
Louvain	0.0702	4.01333333333
Girvan-Newman	0.0694	3.94444444444
KCore	0.0948	6.5444

Table 4: Comparison of quality measures for different approaches

EM algorithm (I. Rodin)

For finding Non-negative Matrix factorization for initial term-document matrix A one can apply EM algorithm.

Let W be a dictionary of all words, D is a collection of documents, and n_{dw} be a frequency of word w in document d. Then we want to find model $p(w|d) = \sum_{t \in T} f_{wt} H_{td}$ with parameters $F_{W \times T}$ and $H_{T \times D}$:

$f_{wt} = p(w|t)$ - probability of word w in topic t

$h_{td} = p(t|d)$ - probability of topic t in document d.

Then we have the following optimization task for maximization of log-likelihood:

$$\begin{cases} \sum_{d \in D} \sum_{w \in W} n_{dw} \ln \sum_{t \in T} f_{wt} h_{td} \to \max \\ s.t. \\ f_{wt} \geq 0, h_{td} \geq 0 \\ \sum_w f_{wt} = 1, \sum_t h_{td} = 1 \end{cases}$$

This optimization task can be solved by EM algorithm. EM algorithm is a method of simple iterations constisting of 2 steps:

E-step:

$p_{tdw} = norm_t(f_{wt} h_{td})$

M-step:

$f_{wt} = norm_w(n_{wt})$, $n_{wt} = \sum_d n_{dw} p_{tdw}$

$f_{td} = norm_t(n_{td})$, $n_{td} = \sum_w n_{dw} p_{tdw}$

where $norm_{i \in I}(x_i) = \frac{max(x_i, 0)}{\sum_{j \in I} max(x_j, 0)}$

The main advantage of EM algorithm is that it allows implementation of regularizators for tuning the model and adding sparsity to matrices F and H.

Sparsity of these matrices means that:

1. topics are consisting of small number of words, and words from different topics should not intersect a lot (sparsity of matrix F);
2. each document should correspond to small number of topics (sparsity of matrix H).

For the purpose of finding keywords we would like not to sparse, but smooth matrices, or, in other words, making them more dense.

We will find matrix factorization by means of BigARTM library [12].

After finding dense matrices F and H, we will implement the following approach:

1. For document d_i find topic t_{j*} such that $h_{j*,i} \geq h_{j,i}$
2. Find top k words for topic $j*$ from matrix F. These words are considered as keywords of document d_i.

Core-shell cluster (I. Rodin)

Now let us formulate the following problem:

Let $G = (V, E)$ be a graph of words built from raw text, V is a set of words and two vertices i and j are connected if these words occurs together in sliding window. Let this graph be weighted, and the weight of edge $e_{ij} = e_{ji}$ equals to the number of co-occurence of words i, j withing sliding window in the whole text.The graph is defined by adjacency matrix A.

Then we want to find a core-shell cluster U consisting of two non-overlapping parts $U = S \cup R$, where S is a shell of the cluster with average connection rate $\lambda \geq 0$, and R is a core of the cluster with average connection rate $\lambda + \mu$, where $\mu \geq 0$.

In this cluster R is a set of the most strongly-connected words which may be considered as keywords of the text, then S may be considered as a set of "supporting" keywords.

Optimization task for extraxting such clusters mey be formulated as follows:

$$\begin{cases} \sum_{i,j}^{N} \left(A_i j - \lambda s_i s_j - (\lambda + \mu) r_i r_j \right)^2 \min \\ s.t. \\ \lambda \geq 0, \mu \geq 0 \\ s_i = \{0, 1\}, r_i = \{0, 1\}, i = 1, .., N \end{cases}$$

Or, in other words, we want to find the approximation for matrix A:

$$A = S + R$$

where $S_{ij} = \lambda$ if s_i, s_j are shell elements, and $r_{ij} = \lambda + \mu$ if r_i, r_j are core elements.

The heuristic algorithm for finding sub-optimal solution of this problem is presented in [13].

Spectral clustering

Spectral clustering approach uses the eigenvalue of the adjacency or Laplacian matrix of the graph to apply dimension reduction before clustering in fewer dimensions. It refers to a set of heuristic algorithms, all based on idea of computing the first few singular vectors and then clustering in a low dimensional subspace.

In our implementation we have used library sklearn.cluster.SpectralClustering. It turned out that Spectral clustering approach performs quite good in term of guessed keywords – it has recovered 6.13 words in average, which gives this algorithm second place among all methods.

Results

Here is a table of all achieved results.

Approach	Average Matches	Jaccard Index
SVD-based	3.35	0.069
NMF-based	3.59	0.076
TextRank	3.395	0.125
Louvain	4.0133	0.0702
Girvan-Newman	3.9444	0.0694
KCore	6.5444	0.0948
EM	4.15	0.05
EM regularised	4.475	0.054
Core-shell	0.96	0.023
Spectral	6.131	0.073

Table 5: Comparison of the results for different approaches

Conclusion

In this project we have applied several approaches to the problem of automatic keyword extraction from news. Note that those methods are not very accurate but can be used for rough approximations of keywords. The main advantage of these algorithms is their simplicity. Each algorithm runs relatively fast with one news story given as input and most of them do not require training datasets (except LSA and topic modeling approaches). The proposed algorithms might be useful in recommender systems and estimation of text relevance (search engines, newspaper generation etc.).

References

[1] M. Kaledin, I. Rodin, and A. Safin. (). Key words extraction (github repository), [Online]. Available: `https://github.com/XuMuK1/KeywordsExtraction`.

[2] L. Marujo, A. Gershman, J. Carbonell, R. Frederking, and J. P. Neto, "Supervised topical key phrase extraction of news stories using crowdsourcing, light filtering and co-reference normalization," in *Proceedings of the LREC 2012*, 2012.

[3] D. Martin and M. Berry, "Mathematical foundations behind latent semantic analysis," pp. 35–55, Jan. 2007.

[4] D. Da Kuang, J. Choo, and H. Park, "Nonnegative matrix factorization for interactive topic modeling and document clustering," pp. 215–243, Oct. 2015.

[5] R. Mihalcea and P. Tarau, "TextRank: bringing order into texts," in *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelonaand Spain, Jul. 2004.

[6] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. DOI: `10.1073/pnas.122653799`. eprint: `http://www.pnas.org/content/99/12/7821.full.pdf`. [Online]. Available: `http://www.pnas.org/content/99/12/7821.abstract`.

[7] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, p. 066 111, 6 Dec. 2004. DOI: `10.1103/PhysRevE.70.066111`. [Online]. Available: `https://link.aps.org/doi/10.1103/PhysRevE.70.066111`.

[8] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006. DOI: `10.1073/pnas.0601602103`. eprint: `http://www.pnas.org/content/103/23/8577.full.pdf`. [Online]. Available: `http://www.pnas.org/content/103/23/8577.abstract`.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, P10008, 2008. [Online]. Available: `http://stacks.iop.org/1742-5468/2008/i=10/a=P10008`.

[10] F. Parés, D. Garcia-Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, "Fluid Communities: A Competitive, Scalable and Diverse Community Detection Algorithm," *ArXiv e-prints*, Mar. 2017. arXiv: `1703.09307 [cs.DS]`.

[11] C. Peng, T. G. Kolda, and A. Pinar, "Accelerating Community Detection by Using K-core Subgraphs," *ArXiv e-prints*, Mar. 2014. arXiv: `1403.2226 [physics.soc-ph]`.

[12] K. Vorontsov, "Additive regularization for topic models of text collections," in *Doklady Mathematics*, Springer, vol. 89, 2014, pp. 301–304.

[13] I. Rodin and B. Mirkin, "Supercluster in statics and dynamics: an approximate structure imitating a rough set," in *International Joint Conference on Rough Sets*, Springer, 2017, pp. 576–586.