Web and Social Information Extraction

A.Y 2018/2019

Mining Evolving Topics

Cannistraci Irene

1603090

January 20, 2020

Contents

1 Introduction

The aim of this project was to identify and trace topics over a temporal interval, where a topic is a set of keywords. Hence, one can see changes in the keyword set of topics throughout the specified time period.

1.1 Project structure

The project is written in Python ad has the following structure:

```
Mining−Evolving−Topics
├── Data
│   ├── output_files
│   │   ├── identification
│   │   └── tracing
│   ├── ds1.csv
│   ├── ds2.csv
│   └── projectproposal.pdf
├── src
│   ├── core
│   │   ├── __init__.py
│   │   ├── preprocessing.py
│   │   ├── identification.py
│   │   └── tracing.py
│   ├── test
│   │   ├── __init__.py
│   │   └── main.py
│   ├── util
│   │   ├── __init__.py
│   │   └── functions.py
│   └── __init__.py
├── __init__.py
├── .gitignore
└── README.md
```

Figure 1: Structure

- **Data package** contains:
  - **output_files** directory that contains:
    - **identification** directory that contains the Pagerank and the topic identification results
    - **tracing** directory that contains the timeline of the tracing topic results
  - **ds1.csv** the keyword co-occurrence dataset
  - **ds2.csv** the co-authorship dataset
  - **project proposal.pdf** the project description
- **src package** contains:
  - **core package** that contains:
    - **__init__.py**
    - **preprocessing.py** that contains the preprocessing phase
    - **identification.py** that contains the first task: topic identification
    - **tracing.py** that contains the second task: topic tracing
  - **test package** that contains:
    - **__init__.py**
    - **main.py**
  - **util package** that contains:
    - **__init__.py**
    - **functions.py** that contains utility functions that can be used in all the other modules
- **README.md**

1.2 Execution

You can run the project simply typying the following command: "python3 -m src.test.main"

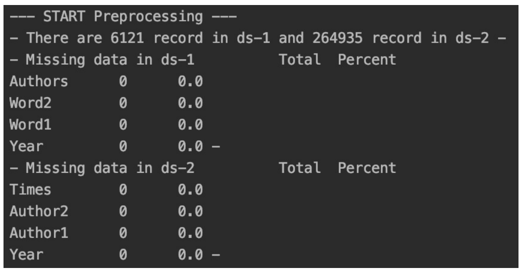
2 Preprocessing

This section explains the "**preprocessing.py**" module contained into the core package.

Data preprocessing is an important step, during this phase the two datasets were inspected in order to check if there were wrong data such as missing values, special characters and so on.. This is very important since wrong data may produce misleading results.

2.1 Missing data

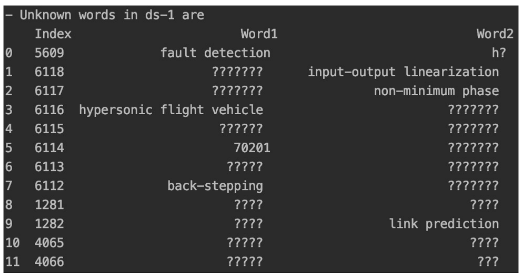
Seemed there were no missing data (nan) in both dataset, but in the keywords co-occurrence dataset



```
--- START Preprocessing ---
- There are 6121 record in ds-1 and 264935 record in ds-2 -
- Missing data in ds-1          Total  Percent
Authors      0      0.0
Word2        0      0.0
Word1        0      0.0
Year         0      0.0 -
- Missing data in ds-2          Total  Percent
Times        0      0.0
Author2      0      0.0
Author1      0      0.0
Year         0      0.0 -
```

Figure 2: Missing data

there were some words that were composed by a sequence of question marker "?", so I decided to consider those words as missing. I also decided to *drop the entire row* since every row represents the relation of two keywords being used by two different articles but if one of them is unknown there was no need to maintain the entire row.



```
- Unknown words in ds-1 are
    Index                     Word1                          Word2
0   5609            fault detection                             h?
1   6118                    ???????    input-output linearization
2   6117                    ???????             non-minimum phase
3   6116  hypersonic flight vehicle                       ???????
4   6115                     ??????                       ???????
5   6114                      70201                       ???????
6   6113                      ?????                       ???????
7   6112               back-stepping                      ???????
8   1281                       ????                          ????
9   1282                       ????               link prediction
10  4065                      ?????                          ????
11  4066                      ?????                           ???
```

Figure 3: Unknown words

2.2 Wrong data

2.2.1 Temporal interval

The project specification was to trace topics over the timeline [2000-2018] so there are no reason to maintain records with a year < 2000 or a year > 2018.

- Dropped rows from ds1: 168
- Dropped rows from ds2: 29830

2.2.2 Words to correct

There were some words like "05.45.vx", "05.45.gg" and "05.45.xt" that I'm not able to understand what they mean, but I decided to not drop those words because since those words derived from articles in literature, maybe they refer to a specific field of studies unknown to me.

3 Topic Identification

This section explains the "**identification.py**" module contained into the core package.

3.1 Top-k keywords

The first task was to select **top-k** keywords for each year, where k is the number of generated topics (k=5,10,20,100), according to a certain metric.

I decided to work in a "step-by-step" way using an iterator over the years, so at every step I generated two graphs. The first graph is the one that contains all the keywords that are in the ds1. I called this graph *current_key_graph* and it is composed as follows: for each row of the dataset I added two *nodes* that are the two keywords and an *edge* between these two nodes. Edges are weighted and after several tries (see the code) I decided to set *weights* using a *custom metric* defined by me.

The metric is the following: I summed up all the times that each word is used by each authors in the dictionary and then I multiplied this value by the number of publications each author did (this information is contained in a dictionary called *weights_dict* that I created). I also normalized the value since in the next step, Spreading of influence, I need values between 0 and 1.

Suppose that one *row* of the dataset was

word1,word2,{a1: times_a1, a2:times_a2, a3:times_a3}

the weight of the edge between *word1* and *word2* was calculated as:

$$(times\_a1 * a1\_pub) + (times\_a2 * a2\_pub) + .. + (times\_an * an\_pub)$$

The second graph is the one that contains all the authors that are in the ds2, I called this graph *current_aut_graph* and it is composed as follows: for each row of the dataset I added two *nodes* that are the two authors and an *edge* between these two nodes that represents the co-authorships. Edges are weighted with the number of co-citations between the two authors. I used this graph only to create the weights_dict.

The *weights_dict* dictionary has the following structure:

- *Key*: node (author)
- *Value*: value between [0,1] that represents the total number of publications he did

After creating the needed graphs I tried to understand what was the structure of these. I noticed that, for each year, the keywords graph was a graph not totally connected, but it had several connected components.

For example the 2002 graph has the following structure:

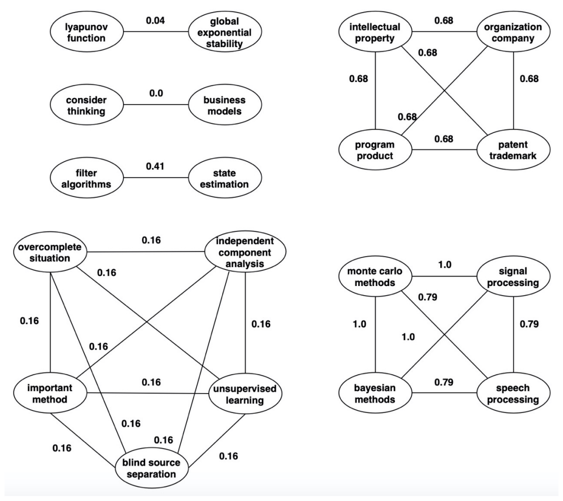


Figure 4: Keywords graph 2002

This is the main reason why I chose to use the **Pagerank** algorithm in order to find the top-k keywords. Pagerank is a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them. The algorithm gives each page a rating of its importance, which is a recursively defined measure whereby a page becomes important if important pages link to it. This definition is recursive because the importance of a page refers back to the importance of other pages that link to it. One way to think about PageRank is to imagine a random surfer on the web, following links from page to page. The page rank of any page is roughly the probability that the random surfer will land on a particular page. Since more links go to the important pages, the surfer is more likely to end up there[1]. In our case we are not dealing with web pages but with kewords but the behaviour is exactly the same.

After the Pagerank execution on the 2002 graph the result is the following:

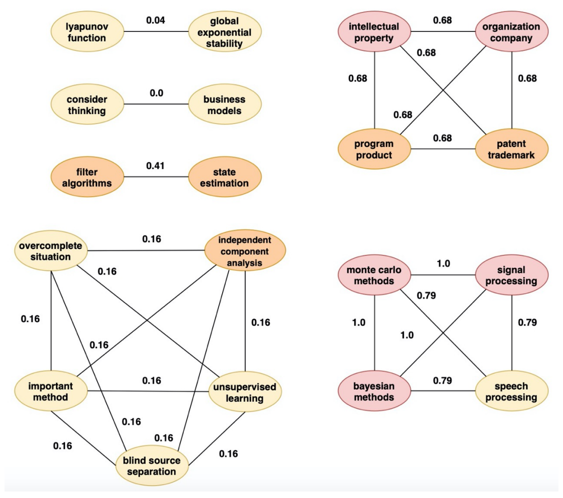


Figure 5: Pagerank on keywords graph 2002

Where:

- **Red**: top-**5** keywords identified
- **Red** and **Orange**: top-**10** keywords identified
- **Red**, **Orange** and **Yellow**: means top-**20** (and top-**100**) keywords identified

Since the number of nodes of this graph is 19, the top-100 keywords and top-20 keywords are the same, the entire graph. The reason why some words with an higher score are not taken into account is that the Pagerank algoritghm use the damping factor in order to 'jump' from one component to another that is not connected to the node that the surfer is currently visiting. This is done in order to not being 'stacked' in the same point forever. In the *output_file\identification* directory there is a *pagerank.csv* file that contains all the pagerank result with the relative score for each node, ordered in an increasing way. Even if I thought that the best choice was to use the Pagerank, I tried to apply other two metrics in order to see what happens:

- *Degree Centrality*: the degree centrality for a node v is the fraction of nodes it is connected to.
- *Betweenness Centrality*: it counts the number of geodesic paths between i and k that actor j resides on; geodesics are defined as the shortest path between points[2].

As I supposed, these two metrics didn't produce good results because the keyword graph is not fully connected, so the result depended only on where the algorithm start, from which component. Another problem with Betweenness is that for weighted graphs the edge weights must be greater than zero because zero edge weights can produce an infinite number of equal length paths between pairs of nodes[3].

3.2 Spreading of Influence

The second task was to apply a **Spreading of Influence algorithm** to report nodes influenced in each iteration of the algorithm. The influenced nodes represent a topic. The algorithm is applied using the top-k as seeds, starting from the top-5 nodes until the top-100.

There are two basic classes of diffusion models: *threshold* and *cascade*. The network is represented as a directed graph and in our case each keyword represents a node. Nodes start either active or inactive, an active node may trigger activation of neighboring nodes.

Since all the algorithms use directed graph and since I had to decide the threshold function, I created a new graph called *current_spread_graph* that is a directed graph where weights on edges are values bewteen 0 and 1 chosen in the following way:

The 'old' weight of the edge multiplied by the degree of the node (where the degree of the node, since the graph is weighted, is the sum of the weight). Then the result is normalized since the threshold is a value between [0,1].

So the main reason why I applied the **Idependent Cascade model** is that it applies threshold on edges while other models, like the Linear Threshold, apply threshold on nodes. Since the 'starting' graph, the current_key_graph is weigthed, when I generated the directed one current_spread_graph I just had to update previous weights according to my metric and in this way I already had weights on edges. After I applied the IndependentCascadesModel[4]. This model starts with an initial set of active nodes A0: the diffusive process unfolds in discrete steps according to the following randomized rule:

- When node v becomes active in step t, it is given a single chance to activate each currently inactive neighbor w; it succeeds with a probability p(v,w).
- If w has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order.
- If v succeeds, then w will become active in step t + 1; but whether or not v succeeds, it cannot make any further attempts to activate w in subsequent rounds.

The influence propagation process terminates at step t, if and only if At=∅[5].

There are three parameters to set:

- *Infected*: represents set of nodes infected at the first step (seed nodes). I used one node of the top-k at time.
- *threshold*: (edge) I simply used edge weigths that were updated as I previously specified.
- *iteration_bunch*: number of iterations to execute. I set this value to 5 since graphs are small and I noticed that after 5 iterations there were no changes, for some graphs 3 iterations are enough.

At each iteration I generated a dictionary containing the number seeds with all the infected words as follows:

- *Key*: year
- *Value*: dictionary with number of seeds as key and the set of topics as values

Since I set just one node at each iteration as infected keywords can be duplicated and it's possible that different node infected same keywords, so different topics contain same keywords. It is important to highlight that since the chosen model is *based on probability*, so everytime you run the program topics can change and it is also possible that top-5 topics will not be included in top-10 topics, top-10 in top-20 and so on.

For example, lets analyze the 2013:

File: *output_file\ranking\pagerank.csv*, top-5 keywords are:

- *simulation*, pagerank score: 0.01997660905332627
- *algorithm*, pagerank score: 0.019891039782059163
- *mathematical optimization*, pagerank score: 0.019749834725123067
- *sampling signal processing*, pagerank score: 0.019053858071798553
- *artificial neural network*, pagerank score: 0.01904089663953379

File: *output_file\identification\topics.json*, top-5 topics are:

- **T1**: simulation, telecommunications link
- **T2**: algorithm, simulation, sampling signal processing, lyapunov fractal, nonlinear system, dynamic programming, artificial neural network, sampling surgical action, throughput, word embedding, social inequality, recurrent neural network
- **T3**: algorithm, mathematical optimization, sampling signal processing, sampling surgical action, data synchronization, linear matrix inequality, optimization problem, artificial neural network, neural network simulation
- **T4**: algorithm, sampling signal processing, data synchronization
- **T5**: lyapunov fractal, artificial neural network, sampling signal processing, algorithm, simulation, numerical analysis, social inequality, neural networks, qlearning,machine learning

For example we can notice that the keyword '*algorithm*' occurs in 3 topic.

3.3 Merge

The third task was to join the **produced topics** in a given year following a merging strategy which takes care of possible overlaps among them.

Since my idea was to run one seed at time, at the end of the Independent Cascades execution I had k set of keywords that represent k topic of a specific year and as I previously said and showed these sets can have common keywords. My idea for merging was to use the **cosine similarity** that measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis[6]. I also tried to use the *Jaccard similarity* and results were more or less the same but I thought Cosine similarity was more suitable for the required task.

For every possible couples of topic in the top-k if the cosine similarity is equal or greater than 0.5 then I merge the two topics, otherwise nothing. For example if we analyze the previous example, one possible case of 2013:

- **T2**: algorithm, artificial neural network, dynamic programming, lyapunov fractal, nonlinear system, recurrent neural network, sampling signal processing, sampling surgical action, simulation, social inequality, throughput, word embedding
- **T5**: algorithm, artificial neural network, lyapunov fractal, machine learning, neural networks, numerical analysis, sampling signal processing, simulation, social inequality, qlearning

Common keywords: algorithm, artificial neural network, lyapunov fractal, sampling signal processing, simulation, social inequality

So these two topics can be *merged* in order to obtain the new topic:

- **T2,5**: algorithm, artificial neural network, dynamic programming, lyapunov fractal, machine learning, neural networks, nonlinear system, numerical analysis, qlearning, recurrent neural network, sampling surgical action, sampling signal processing, simulation, social inequality, throughput, word embedding

All the merged topics are inserted in a new dictionary that is saved in the *output_file\identification* directory that has the same structure of the previous dictionary; for each key are present all the topics, the ones identified in the first step and the one that are the result of the merge of two topics. Since I noticed that for some year, after the first merge, there are some topics with common keywords, I decided to apply the merge again. So after the first merge, there is another merge that instead of trying all the possible couples starting from the top-k topics, it tries to identify similarities starting from the previously merged topics. If two previously merged topics are merged again, they will not appear as single topic in the merged file but only as a 'unique' topic.

The file *output_file\identification\topics.json* will be used for the next task.

4 Topic Tracing

This section explains the "**tracing.py**" module contained into the core package.

4.1 Trace, Analysis and Merge

The last task was to **trace** over the timeline [2000-2018] any topic identified in the previous task and **analyze** the temporal/structural behaviour of a topic and decide if topics identified in consecutive years can be **merged** together. Then create a final list of the merged topics. As I previously said I loaded the *output_file\identification\topics.json* file, produced in the previous step in order to trace topics.

For each year and for each top-k topic I checked if the same topic appears in the following year in the same top-k. For some year and some topics it was possible to find over all the timeline exactly the same topic starting from the year in which it was found until the end. For example the topic containing only the keyword '*artificial neural networks*' is repeated in every year starting from the 2000 until the 2018:

['2000', ['artificial neural network'], '-> 2001', ['artificial neural network'], '-> 2002', ['artificial neural network'], '-> 2003', ['artificial neural network'], '-> 2004', ['artificial neural network'], '-> 2005', ['artificial neural network'], '-> 2006', ['artificial neural network'], '-> 2007', ['artificial neural network'], '-> 2008', ['artificial neural network'], '-> 2009', ['artificial neural network'], '-> 2010', ['artificial neural network'], '-> 2011', ['artificial neural network'], '-> 2012', ['artificial neural network'], '-> 2013', ['artificial neural network'], '-> 2014', ['artificial neural network'], '-> 2015', ['artificial neural network'], '-> 2016', ['artificial neural network'], '-> 2017', ['artificial neural network'], '-> 2018', ['artificial neural network']]

and the topic containing only the keyword '*algorithm*' is repeated in every year from the 2003 until the 2018:

['2003', ['algorithm'], '-> 2004', ['algorithm'], '-> 2005', ['algorithm'], '-> 2006', ['algorithm'], '-> 2007', ['algorithm'], '-> 2008', ['algorithm'], '-> 2009', ['algorithm'], '-> 2010', ['algorithm'], '-> 2011', ['algorithm'], '-> 2012', ['algorithm'], '-> 2013', ['algorithm'], '-> 2014', ['algorithm'], '-> 2015', ['algorithm'], '-> 2016', ['algorithm'], '-> 2017', ['algorithm'], '-> 2018', ['algorithm']]

n.b the timelines are saved in the *output_file\tracing\timeline.txt* file

While for other topics it wasn't possible to find exactly the same topic over all the years.

So, after this task I had two different type of timelines:

- One with always the same topic (the two examples above)
- One with different topics

In the case I tried to find out if it was some topic in the successive year that was similar to the one that I was studying. I decided to establish if two topics are similar using the same metric that I chose for the topic identification task: the *cosine similarity*. For the T1 is used a threshold of 0.5 while in this case I worked in a different way. I started with a highest threshold 0.6, but using this value only I got only *127* possible timelines over all the topics, and all the timelines were composed by more or less the same topic, there were no evolving topics! Then I decided to use the same threshold of the task T1: 0.5. Using this threshold I got *194* possible timelines, for example one of this was:

['2007', ['evolutionary algorithm', 'mathematical optimization'], '-> 2008', ['lyapunov method', 'mathematical optimization'], '-> 2009', ['lyapunov method', 'mathematical optimization'], '-> 2010',

['lyapunov method', 'mathematical optimization'], '-> 2011', ['lyapunov method', 'mathematical optimization'], '-> 2012', ['lyapunov method', 'mathematical optimization'], '-> 2013', ['lyapunov method', 'mathematical optimization'], '-> 2014', ['lyapunov method', 'mathematical optimization'], '-> 2015', ['lyapunov method', 'mathematical optimization'], '-> 2016', ['lyapunov method', 'mathematical optimization'], '-> 2017', ['lyapunov method', 'mathematical optimization'], '-> 2018', ['lyapunov method', 'mathematical optimization']]

The choice of this threshold was also based on the fact that I noticed some changes of some topics during the years but without loosing the meaning of the topic itself. Maybe I could try to test several threshold values but unfortunately I don't have enough time!

The last step was the union of the 'evolving' topics. Since for me it has no sense to merge a timeline that contains only one topic (it is already a topic!) I decided to merge all the timelines that contains at least one different topic from the one that I started. In this way I decided to merge only topics that 'grow' during the years. From the last example above, starting from the topic

['evolutionary algorithm', 'mathematical optimization']

I got the new topic

['mathematical optimization', 'lyapunov method', 'evolutionary algorithm']

Another example is this timeline:

['2012', ['artificial neural network', 'neural network simulation', 'recurrent neural network', 'sampling surgical action', 'sampling signal processing', 'synchronization computer science'], '-> 2013', ['artificial neural network', 'data synchronization', 'experiment', 'linear matrix inequality', 'lyapunov fractal', 'neural network simulation', 'sampling surgical action', 'sampling signal processing', 'simulation', 'vertex'], '-> 2014', ['artificial neural network', 'data synchronization', 'experiment', 'linear matrix inequality', 'lyapunov fractal', 'neural network simulation', 'sampling surgical action', 'sampling signal processing', 'simulation', 'vertex'], '-> 2015', ['artificial neural network', 'data synchronization', 'experiment', 'linear matrix inequality', 'lyapunov fractal', 'neural network simulation', 'sampling surgical action', 'sampling signal processing', 'simulation', 'vertex'], '-> 2016', ['artificial neural network', 'data synchronization', 'experiment', 'linear matrix inequality', 'lyapunov fractal', 'neural network simulation', 'sampling surgical action', 'sampling signal processing', 'simulation', 'vertex'], '-> 2017', ['artificial neural network', 'data synchronization', 'experiment', 'linear matrix inequality', 'lyapunov fractal', 'neural network simulation', 'sampling surgical action', 'sampling signal processing', 'simulation', 'vertex'], '-> 2018', ['artificial neural network', 'data synchronization', 'experiment', 'linear matrix inequality', 'lyapunov fractal', 'neural network simulation', 'sampling surgical action', 'sampling signal processing', 'simulation', 'vertex']]

So starting from the topic

['artificial neural network', 'neural network simulation', 'recurrent neural network', 'sampling surgical action', 'sampling signal processing', 'synchronization computer science']

I got the new topic

['linear matrix inequality', 'simulation', 'lyapunov fractal', 'vertex', 'neural network simulation', 'artificial neural network', 'sampling surgical action', 'sampling signal processing', 'experiment', 'data synchronization', 'synchronization computer science', 'recurrent neural network']

In some cases is only a small evolution, but in any case it's an **evolution**!

References

[1] Eric Roberts: The Google Pagerank Algorithm
`https://web.stanford.edu/class/cs54n/handouts/24-GooglePageRankAlgorithm.pdf`

[2] Paola Velardi: Social Media Analytics
`http://twiki.di.uniroma1.it/pub/Estrinfo/WebHome/9b.SocialMediaAnalyticsA2.pptx`

[3] Networkx library
`https://networkx.github.io/documentation/networkx-2.0/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html#networkx.algorithms.centrality.betweenness_centrality`

[4] Network Diffusion library
`https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/IndependentCascades.html`

[5] Influence Spreading Path and its Application to the Time Constrained Social Influence Maximization Problem and Beyond
`https://www.ntu.edu.sg/home/gaocong/papers/tkde_inf.pdf`

[6] Data Mining: Concepts and Techniques
`https://www.sciencedirect.com/topics/computer-science/cosine-similarity`