WebSocial Information Extraction - Project Report

Antonio Pio Ricciardi

May 2020

1 Introduction

Throughout this report I will explain all the steps performed in order to complete assigned tasks and the reasons behind taken decisions, from data preprocessing to the transformation of datasets into graphs, up to topic merging and topic tracing tasks.
First of all, the chosen programming language is **Python**, which allowed spend more time in problem solving instead of coding, thanks to its easy of use and the large number of libraries available that allowed me to easily perform graph operations and plotting.
For graph analysis and operations **NetworkX** was the chosen Python library.
The following sections describe in detail how both tasks are completed.

2 Preprocessing Data

In this project data understanding is of paramount importance. First thing is to read and analyse data to look for patterns, escape characters or errors in the data structure or organization, then data can be reshaped in order to use it in a more comfortable manner. For this purpose datasets *ds-1.tsv* and *ds-2.tsv* have been analysed looking for flaws in the data.
The findings were for *ds-1.tsv* were:

- **Year 0:** there are some entries having year=0, probably meaning that data could not be fetched for these entries. These will not be taken into account, because we need to compute metrics and merge for each given year and we do not know to which year data in year 0 relate to.
- **Nodes containing only sequences of *?* as keyword:** Again, this is a case of missing information, where data that could not be fetched for some keywords, therefore replaced by question marks. Nodes with these keywords are included in the graph since they are useful for graph connectivity and the creation of cliques, on which my work heavily relies on and topic merges, since there *may be* a pattern in the number of *?*

in a keyword. However they will not be taken into account as starting keywords for topics generation.

For *ds-2.tsv* the only finding was the one relative **Year 0** and, as for *ds-1.tsv*. I decided to ignore this kind of entries.

2.1 Graphs creation

The fundamental structure on which this project relies on are graphs. Datasets ds-1 and ds-2 are converted into a dictionary of graphs,*graphs_ds1* and *graphs_ds2*, respectively. With graphs it has been possible to compute keyword scores, find topics through the Spreading of Influence algorithm and perform topic tracing.

```
2014    artificial neural network   experiment  {'31e7de2233e356105629e9900b3cb20ec9b9bba6': 2}
2014    convolutional neural network    algorithm   {'5ee814c1094d420dc2612fc3453e401b5787f361': 2, 'bf64de56292663ded5487aff7dd904d07a89c394': 2}
2014    algorithm   experiment  {'4ccfbd41bf92960b23a11f59672c28cc08eb0655': 2, 'e1c3721516e2db714cb23e3d71a4231054ba0462': 2}
2018    time complexity numerical analysis  {'6815d26cf8d06fe99e2a14a60171dd60245715b8': 2}
```

Figure 1: ds_1 structure, in tab separated values. An entry is made up by a year, two keywords and a dictionary of authors using these two keywords with an integer value representing the number of times each author used these two keywords together.

Dataset ds_1 ds_1 is converted into a dictionary, where years are the keys and the entries in each line compose the actual graphs. Therefore we have a graph for every year.
The dataset is read line by line and, for all the entries of a given year, keywords become nodes of the graph for that year and there is an edge between them if the keywords are in the same line, meaning that they appear in the same paper at least two times.
Every edge, in turn, contains another dictionary, where keys are paper authors using the two keywords together, and values are the number of times these two keywords are used, for each author.
In this report the set of graphs will be called *graphs_ds_1*, and *graph_ds_1_yyyy* will be used to refer to a specific year.

```
2011    3af456d226ba5dc40b0ec23c4cb7c8a648ee7df2    cd6a69045eeb4f62045185562a0d6a4253344b00    1
2018    f0b8861c53c52330ac1b1b7900fba948c9a03061    93fe78bbb94ab921b4f08acff7d71a911cc501ca    3
2015    85cab93e2f335dd322e9b2f20aa2a2444c9544c3    f20ed04f2dc01b1ed8c2765565f2b9343cab71f0    2
```

Figure 2: ds_2 structure, in tab separated values. A line is made up by a year, two authors and an integer representing the number of co authorship those two authors made that year. Note that the existence of pair **A - B** does not preclude the existence of the pair **B - A**.

Dataset ds_2 To convert ds_2, similarly as what is performed for ds_1, years are the keys of a dictionary with graphs as values. Each graph is made by authors as nodes, with edges between them if they are in the same line in the

dataset, meaning that they are co-authors for a paper. Weight for an edge is the number of co-authorships these two authors have. We will call the set of graphs *graph_ds_2*, while saying *graph_ds_2_yyyy* to refer to a specific year.
Many of the graphs in both graph_ds_1 and graph_ds_2 are usually made of lots of connected components.

3 T1: Topic Identification

Topic Identification means finding a set of keywords influenced by a starting node, over a graph. In order to find influenced nodes with a *Spreading of Influence algorithm*, I decided to use the *Linear Threshold model* over graphs_ds_1. Before running this algorithm it is necessary to assign each keyword a score. Following subsections explain how the scoring system for keywords works.

Spreading of Influence must be performed from a set of *top-k* keywords. To rank keywords, it is necessary to give them a score. Scoring is a combinations of keyword cliqueness score - the number of cliques keywords belong to, in the graph for a certain year - weighted by another keyword weight that involves the pagerank of authors. Following subsections describe in detail the scoring system.

3.1 Keyword Scoring

Given and edge e in graph_ds1_yyyy, we define the score for that edge by the sum of the pagerank of each author multiplied by the value associated to that author:

$$score_e = \sum_{i=0}^{m} pagerank(a_i) * c_i \tag{1}$$

where $pagerank(a_i)$ returns the pagerank value of the author a_i computed on the graph_ds2_yyyy.

First we get authors scores by computing the *pagerank* over the graph_ds2_yyyy, for each year. Subsequently, the score of a keyword is given by the sum, for each of the edges to which it is connected, of the values of each author in that edge, weighted by the pagerank score of each author.
We basically want that the importance of an author directly reflects over the keywords. A pair of keywords used many times by less important authors should be scored lower than keywords used less times by more important authors. More in detail:

- **Cliqueness score:** A clique is an important scoring system because words forming cliques may already represent topics, therefore we want this to matter.
- **Authors Pagerank:** if an author using this word has high pagerank score, then this should positively weight the score for the word.

Author Pagerank: Given an edge e connecting two keywords (kw_0, kw_1), we define $(a_0 : c_0, a_1 : c_1, \ldots, a_n : c_m)$ the list of authors that used these two keywords together in at least two papers. v_i for a_i denotes the exact number of times that author used these two keywords together.
Here we make the assumption to be working on a certain year *yyyy* for *graphs_ds_1* and *graphs_ds_2*.

Keyword weight: The weight for a keyword kw is then given by:

$$weight(kw) = \sum_{e:e \in Edges(kw)} score_e \tag{2}$$

that is simply the sum of the scores of all of the edges connected to kw. This is computed by the method *get_keywords_weight(...)*.

Final Keyword Score: Finally, the final score for a keyword is given by the number of cliques the node of that keyword belongs to and denoted by $cliques(kw)$, weighted by the weighting factor, written as:

$$score(kw) = |cliques(kw) * weight(kw)| \tag{3}$$

This is performed by method *get_top_topics_by_cliqueness(...)*.
Obtained scores are then normalized in the [0,1] range using the normalization formula:

$$\frac{x_i - min(x)}{max(x) - min(x)} \tag{4}$$

Once we have a way to assign a rank to keywords, they can be sorted in order to select the first top k highest scored keywords.

3.2 Topic identification

To identify topics using a *Spreading of Influence algorithm*, starting from keywords in the *top-k*, the *Linear Threshold method* is used. In order to apply this algorithm, it was first necessary to convert all the graphs in graphs_ds1 into directed graphs, by simply replacing every edge with two directed edges, one pointing the opposite direction of the other.
Each value is first normalized in the $[0.01, 1]$ range using the normalization formula:

$$\frac{x_i - min(x)}{max(x) - min(x)} \tag{5}$$

then, using keywords ranks edge values are computed: Therefore the weight for the edge between (kw_j, kw_i) is given by:

$$weight(kw_j, kw_i) = \frac{rank(kw_j)}{total\_incoming\_weight(kw_i)} \tag{6}$$

where $total\_incoming\_weight(kw_i)$ is the sum of the value of all the edges incoming into kw_i:

$$total\_incoming\_weight(kw_i) = \sum_{(kw_j, kw_i) \in in(kw_i)} rank(kw_j) \tag{7}$$

(kw_j, kw_i) is a directed edge going from kw_j to kw_i, $in(kw_i)$ is the set of incoming edges of kw_i. What we do here is just the sum of the rank of all the incoming keywords into kw_i.

By computing this, the sum of the weight for all the incoming edges of a node equals 1.

3.3 Topic generation: Spreading of Influence Algorithm

It is now time to apply the *Linear Threshold* model to compute the Spreading of Influence over a graph in a given year. A simple check on scores is used as an activation function σ_{kw_j} for nodes at each iteration of the algorithm.
A node can be considered active if the sum of the normalised weight of its neighbours (incoming edges) is larger than the score of the node:

$$\sigma_{kw_i} = \begin{cases} \text{True} & \text{if } sum\_incoming(kw_i) \geq score(kw_i) \\ \text{False} & \text{otherwise} \end{cases}$$

where $sum\_incoming(kw_i)$ is given by the sum of the weight of all incoming edges of kw_i, that is:

$$sum\_incoming(kw_i) = \sum_{(kw_j, kw_i) \in in(kw_i)} weight(kw_j, kw_i) \tag{8}$$

The algorithm is started from the top-k highest scored keywords in each year.
The result will be a the set of influenced keywords that we call topic.
Topics are represented as a dictionary, with the starting node - that I call **generating keyword** as a key and the whole topic (that includes the generating keyword, too) as a value.
Figure 3 shows the execution of the spreading of influence algorithm on two graphs for two different years: it can be seen that influenced keywords may really be associated to the generating keyword, maybe representing different contexts, too.

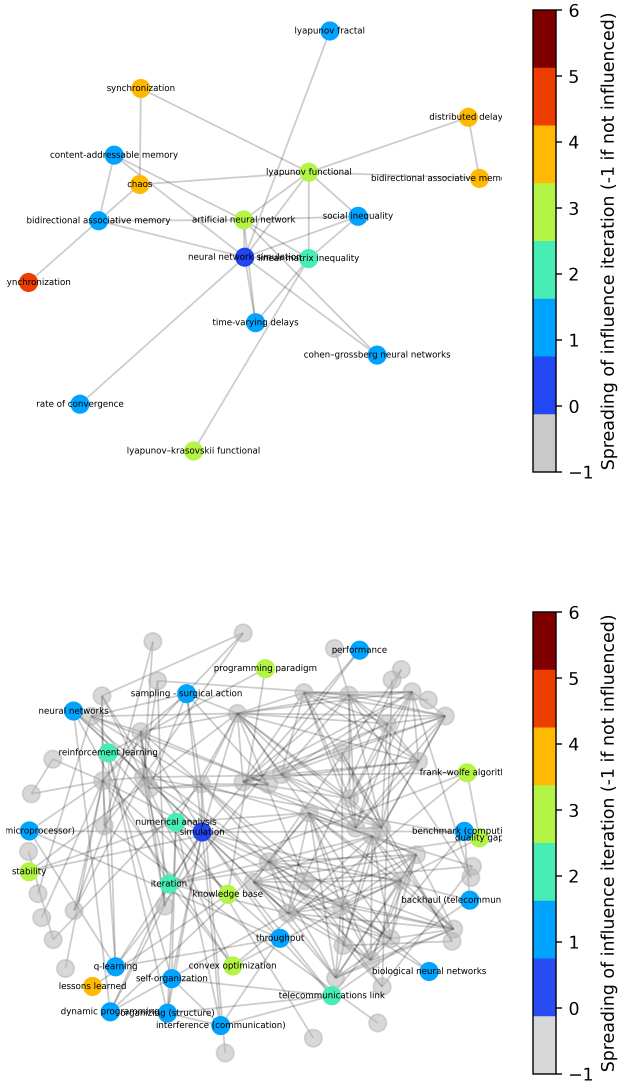

Figure 3: Node coloring represents iterations of the Spreading of Influence algorithm over a graph. 0 is the starting node and each increasing number represent nodes activated in that iteration of the algorithm. -1 is for non-activated nodes (for which labels are hidden, to increase readability). First graph is for year 2007 and its coloring is obtained performing the algorithm starting from keyword *neural network simulation*, Second one is for year 2003, with coloring obtained starting from the keyword *simulation*. Influenced nodes form a topic.

3.4 Topic merging

Once topics have been obtained, topic merging can be performed.
To merge topics in a given year, it is necessary to establish a method to assess whether two topics are similar or not.
Topics are first sorted from larger to smaller, then we start iterating over topics t_i, for $i = 0 \dots n$. What happens is that at iteration i, we compare t_i to all the smaller topics t_j, for $j = i + 1 \dots n$, therefore always having $j > i$ and, since topics are sorted we know that $size(t_i) \geq size(t_j)$.
If the dimension of $t_i \cap t_j$ is at least 0.7 times bigger than the smaller set alone, that is t_j we can merge these two topics. This can be written as:

$$\frac{t_i \cap t_j}{t_j} \geq 0.7 \tag{9}$$

where 0.6 is a tuned hyperparameter.

The reason behind this threshold model is that if the smaller topic is contained in another by at least the 70% of it, then they should be merged together. Subsequent iteration of the algorithm will be skipped for each topic t_i already merged to another topic in a previous iteration (therefore when it was a t_j), whilst if t_i has been merged and there is a smaller topic t_j that has been merged to topic t_k during a previous iteration - therefore $k < i$ and $size(t_k) > size(t_i)$ - with smaller similarity with respect to t_i, then it will be de-merged from t_k and merged to t_i.
Note that actual merging happens only after the whole iteration is complete, that is when $i = n$. During the iterations it is only checkd which topics merge to which. This decision is taken to avoid a continuous grow of a topic which could end up merging topics that were not similar originally. For example if t_1 is similar to t_2 and t_3 but has no keywords in common with t_4, could become similar and then merge t_4 after merging t_2 and t_3. I avoided this.
Graphs depicting the spreading of influence algorithm, generated topics and topic merges can be found in the *results* directory. Figure 4 shows a merged topic.

```
{'content-addressable memory', 'neural network simulation', 'artificial neural network',
'chaos', 'lyapunov fractal', 'rate of convergence', 'lag synchronization', 'cohen–grossberg
neural networks', 'lyapunov functional', 'time-varying delays', 'linear matrix inequality',
'bidirectional associative memory neural networks', 'distributed delays', 'social inequality',
'bidirectional associative memory', 'lyapunov–krasovskii functional', 'synchronization'}
```

Figure 4: Topic *neural network simulation* in year 2007 after topic merging. Many keyword strictly related to that generating keywords can be seen, such as *artificial neural network*, *rate of convergence* and others.

I found interesting that a topic, other than containing keywords representing similar words, can even describe quite well the generating keyword.

4 T2: Topic Tracing

This task requires to follow the evolution of topics over the years, then merge them if they are similar enough. For example, a topic may or not be relevant in consecutive years or be merged into another one.
To trace the temporal behaviour of a topic, a graph structure is again used. Each set of merged topics over the years is represented as a graph, where topics are represented as nodes - with the generating keyword as a label - and an edge connects topics of two consecutive years. Note that each subgraph represent a set of merged topics, meaning that non-merged topics will either appear in subgraphs where they merge, or in a subgraph as a single node if they don't merge at all.
To trace a topic over the years and thus perform merging, the same method used for topic merging in a same year is applied.
Therefore merging is performed whether the similarity between two sets in consecutive years is bigger than a certain threshold, that was set to 0.5.
More in detail:

- First create a graph for every topic in $year = 2000$, where each generating keyword represents a root for the corresponding graph.
- Then, starting from $year = 2000$ and until $year = 2017$, a topic t_i in year $yyyy$, is compared to all the topics t_j in year $yyyy + 1$:
  - If t_i and t_j are considered to be similar, then t_j can be merged to t_i. The actual merging will happen once t_i has been compared to all topics in $yyyy + 1$, to avoid the situation where t_i grows while there are still some topics to compare and thus merging more topics than it should, ending to grow so much to merge to the whole topics set!
  - For every merged node t_j create a node labeled with its generating keyword in the graph to which t_i belongs and draw a directed edge from t_i to t_j
  - If t_j has not been merged to any t_i in $yyyy$, then create a new graph with the generating keyword of t_j as a root. We have a new graph to trace.
  - If t_i does not merge to any topic in $yyyy+1$ then we stop tracing that path and resulting graph represent the evolution of the root topic and all the connected topics. Nodes in the graph can now be merged

Figure 5 shows a tracing graph, while the resulting topic can be seen in Figure 6.

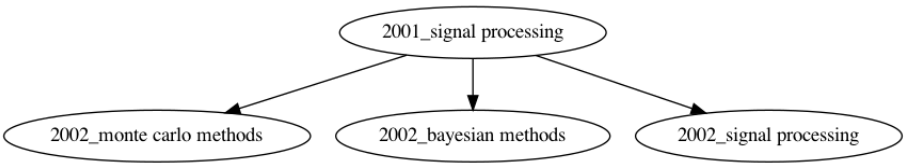

Figure 5: This figure shows a tracing graph. This indicates that the topic identified by the generating keyword *signal processing* in year 2001, merges to topics identified by *monte carlo method*, *bayesian methods* and *signal processing* in year 2002. This graph also indicates that *signal processing* does not merge to any topic in the year 2003. Moreover in this we can also read that all of its child nodes merge to *signal processing* and therefore do not have a tracing graph.

```
{'monte carlo methods', 'fading', 'filtering algorithms', 'laboratories', 'state-space methods', 'filtering', 'bayesian
methods', 'signal processing algorithms', 'particle filters', 'mathematics', 'stochastic processes', 'signal processing'}
```

Figure 6: Topic generated by merging all of the topics in the tracing graph of Figure 5

Topic, spreading of influence image, tracing graphs and resulting merges can be browsed in the *results* directory.

Some findings: A pretty interesting finding is that, when it is set a low threshold for topic tracing merges (e.g. 0.1) and therefore we easily perform merges, the generating keywords are still quite thematic. Figure 7 shows an example of tracing graph obtained with a low threshold merge

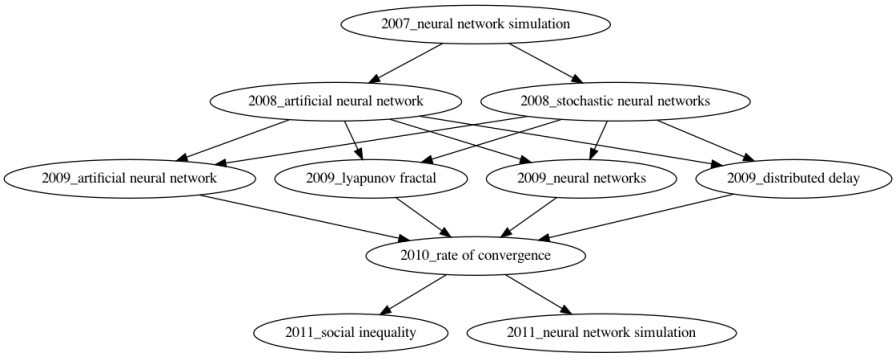

Figure 7: Tracing graph obtained by using a 0.1 merging threshold. This graph start with the generating keyword *neural network simulation* in 2017 and connects similar generating keywords such as *artificial neural network*, *neural networks*, *rate of convergence* and others.