Final Project
Mining Evolving Topics

Web and Social Information Extraction

Bianca Francesco - 1877011

Supervised by Prof. Paola Velardi

Dipartimento di Informatica

M.Sc. in Computer Science

Universitá degli studi di Roma, "La Sapienza"

2018/2019

Contents

1

Introduction

The aim of the Web and Social information extraction course is to identify and track topics over a time interval (from 2000 to 2018 in this specific case). A topic can be viewed as a set of keywords relevant to that topic. To trace we intend to see how this topic varies over time. The project consists of two tasks that we will examine in detail in the subsequent chapters of the report.

1.1 | Technologies

Before going into the development of the project let's briefly introduce the technologies and libraries used during the development:

- **Python 3.7** it was chosen as the programming language for this project as it has already implemented algorithms useful for carrying out the various tasks.
- **NetworkX :** is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It also offers most of the algorithms useful for operations on graphs already fully implemented.
- **Matplotlib :** is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

1.2 | Data

We have been given two noisy datasets in *.tsv* format. So before we could start working, a pre-processing of the data was done to eliminate the problematic rows.

1.2.1 | DS-1: The keyword co-occurrence

This dataset contains one graph for every year [2000-2018]. It contains rows with the following structure:

$$\mathbf{y_q}\text{<tab>}\mathbf{k_i}\text{<tab>}\mathbf{k_j}\text{<tab>}\mathbf{[a_0{:}n_0,\ldots,a_m{:}n_m]}\text{<newline>}$$

where $\mathbf{y_q}$ is the year, $\mathbf{k_i}$ and $\mathbf{k_j}$ two keywords and $\mathbf{[a_0{:}n_0,\ldots,a_m{:}n_m]}$ is the dictionary of authors organized in key-value pairs ***(a,n)*** where ***a*** represents the author while ***n*** represents the number of times the author ***a*** uses $\mathbf{k_i}$ and $\mathbf{k_j}$ in his articles. By grouping the lines by year we obtain a unique graph for each year.

1.2.2 | DS-2: The Co-authorship

This dataset contains one graph for every year [2000-2018]. Collaborations between authors are memorized over the years.
Each row of the dataset is formatted as follows:

$$\mathbf{y_q}\text{<tab>}\mathbf{a_i}\text{<tab>}\mathbf{a_j}\text{<tab>}\mathbf{n}\text{<newline>}$$

where $\mathbf{y_q}$ is the year, $\mathbf{a_i}$ and $\mathbf{a_j}$ are two authors and n is the number of collaborations between the latter (therefore it defines the weight of the edges).

2

T1: Topic Identification

The first task asks us to go and identify the *top-k* keywords in the DS-1 for each year according to some metrics such as pagerank, hits, betweeness, brokerage (k is the number of generated topics). For every node in *top-k* we run a Spreading of Influence Algorithm to report the nodes influenced by them in each iteration of the algorithm (similar to Linear Threshold Model or Independent Cascade). The influenced nodes represent a topic. Finally we must join the topics produced in a given year following a merging strategy.

2.1 | Dataset preprocessing

To carry out the first task we must use the **DS-1** and group the lines by year. After a quick look of the dataset through regex applications I eliminated some corrupt lines (For example, there are entries in the dataset that do not contain the correct keywords and are replaced by a series of '????'. There are also entries concerning a period of time before 2000 but our task is to trace the topics along the timeline [2000,2018] so for simplicity we can avoid considering the previous ones.

2.2 | Creation of a graph per year

Once the **DS-1** dataset was cleaned, it was possible to create a graph for each year of the timeline taken into consideration [2000-2018]. Each node of the graph represents a *keyword* for that year, while the edges indicate whether the two keywords occur simultaneously in an article by the same author. The main problem is the weight we are going to attribute to the edges of these graphs. In fact in the second dataset the weight is ex-

plicitly defined as the number of collaborations between two authors while in the first dataset it is left to our interpretation. To set the weight of the edges of the first dataset I used some information from the second dataset: for each year I obtained the collaborations made by each author. So the weight of each edge $\mathbf{k_i}$ - $\mathbf{k_j}$ is given by the number of times the author uses the two keywords in his articles for the number of collaborations made by the same author.

$$weight(k_i\ ,\ k_j) = \sum_{i=1}^{A} Co - occurences * collaborations_a$$

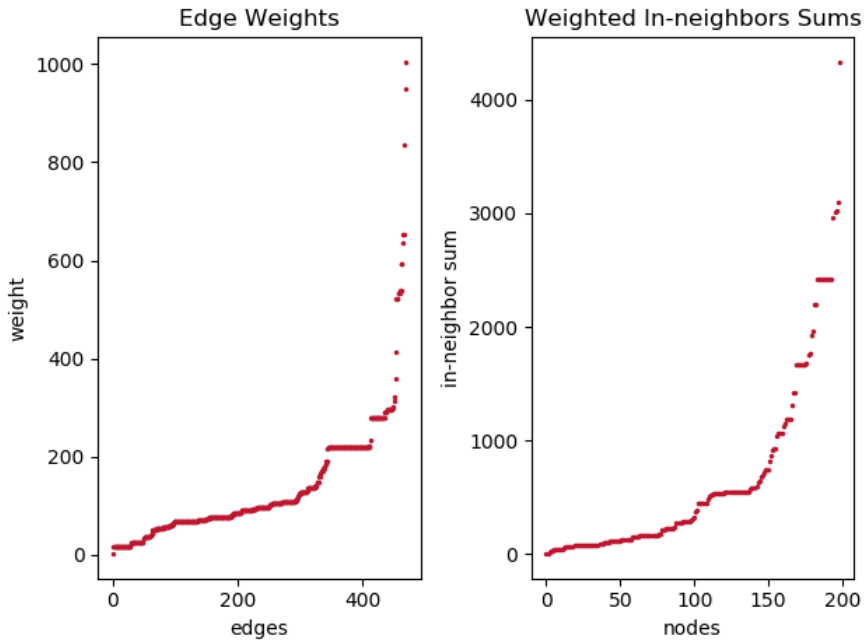

Figure 2.1: Graphs showing the distribution of the edge-weights and of the in-neighbors sums before the normalization step.

One problem that was encountered during the first phase of the project was the presence of outlier edge-weights. Indeed, the weights computed can be unbalanced (ref 2.1), which caused a number of problems when trying to apply the linear threshold algorithm.
weight(k_i , k_j) has been normalized to the interval [0,1] using the *MinMax Normalization*:

$$z_i = \frac{x_i - min(x)}{max(x) min(x)}.$$

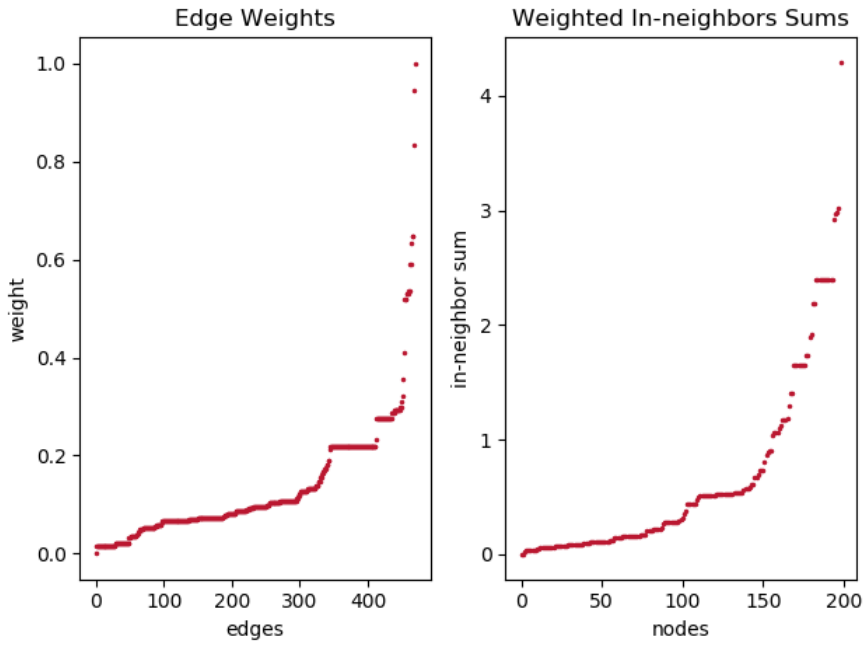

Figure 2.2: Graphs showing the distribution of the edge-weights and of the in-neighbors sums after the normalization step.

The Python *networkx* library was used for the creation of the graphs. Through the method *draw_random* I saved a graphical representation of the graphs. In the figure 2.4 we can see how over the years the number of keywords and connections grows exponentially.

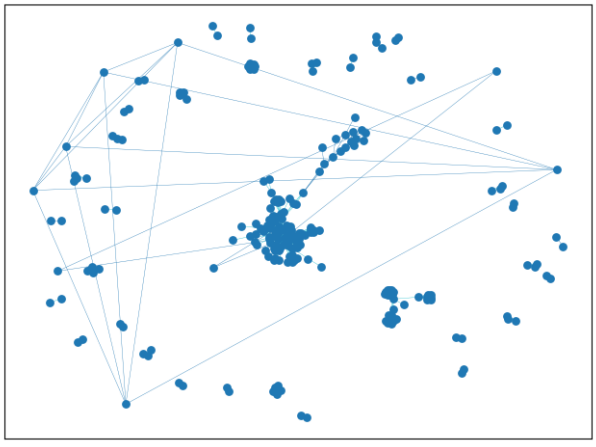

Figure 2.3: Graph comparison [2001 and 2018].

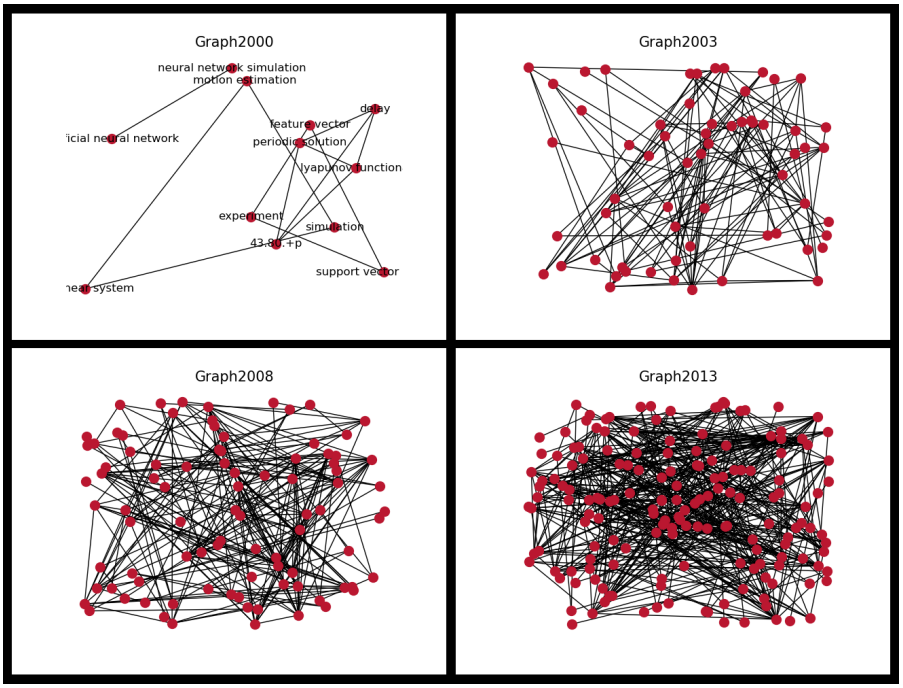

Figure 2.4: Graph evolution over the years.

2.3 | *Top-k* keyword selection

The selection of the top-k keywords was made according to various metrics. In fact, through a function it is possible to pass the desired recovery mode as a parameter. Various tests were carried out to see the different results obtained by the various types of metrics. In the end I opted to use the degree centrality which also takes into account the weight of the edges and this is very important in the graphs that we are going to consider.

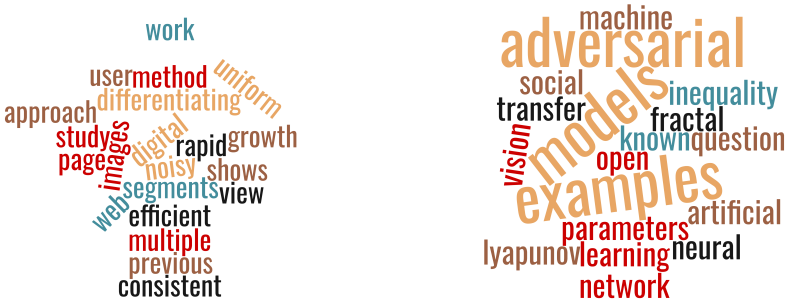

Figure 2.5: Some keywords extracted over the years.

```
Top 5 keywords of 2006 are:  ['linear matrix inequality', 'global asymptotic stability', 'lyapunov-krasovskii functional', 'global exponential stability', 'stochastic systems']
Top 5 keywords of 2007 are:  ['neural network simulation', 'artificial neural network', 'linear matrix inequality', 'lyapunov functional', 'social inequality']
Top 5 keywords of 2008 are:  ['linear matrix inequality', 'query server accesses', 'specified products', 'crawler program', 'scored web pages']
Top 5 keywords of 2009 are:  ['mathematical optimization', 'particle swarm optimization', 'adaptive neuro fuzzy inference system', 'gradient descent', 'recursion']
Top 5 keywords of 2010 are:  ['early termination techniques', 'new techniques', 'existing approaches', 'resulting indexes', 'query processing']
Top 5 keywords of 2011 are:  ['lyapunov fractal', 'artificial neural network', 'linear matrix inequality', 'increasingly inadequate', 'critical role']
Top 5 keywords of 2012 are:  ['approach improves', 'latent factors', 'exploit parallel architectures', 'nips 2011 workshop', 'sparse coding']
Top 5 keywords of 2013 are:  ['christopher d. manning', 'andrew y. ng', 'richard socher', 'tomas mikolov', 'lukás burget']
Top 5 keywords of 2014 are:  ['artificial neural network', 'nonlinear system', 'lyapunov fractal', 'numerical analysis', 'neural network simulation']
```

Figure 2.6: Top-5 keywords [2006-2014].

2.3.1 | PageRank

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages. Page Rank is a good measure of support.

2.3.2 | Authority

Result produced by the application of the HITS algorithm. A authority is a page with many incoming links (in-links, back-links). The idea is that the page may have good or authoritative content on some topic and thus many people trust it and link to it. Authority is a good measure of support

2.3.3 | Hubness

A hub is a page with many out-links. The page serves as an organizer of the information on a particular topic and points to many good authority pages on the topic (e.g. a portal). Hubness is a good measure of influence

2.3.4 | Degree centrality

Centrality refers to (one dimension of) location, identifying where an actor resides in a network. Mostly used for undirected networks. Conceptually, centrality is fairly straight forward: we want to identify which nodes are in the 'center' of the network. The most intuitive notion of centrality focuses on degree. Degree is the number of ties, and the actor with the most ties is the most important

$$C_D = d(n_i) = X_{i+} = \sum_{j}^{d} X_{ij}$$

2.3.5 | Betweenness centrality

Model based on communication flow: A person who lies on communication paths can control communication flow, and is thus important. Betweenness centrality counts the number of geodesic paths between i and k that actor j resides on. Geodesics are defined as the shortest path between points.

$$C_B\,(n_i) = \sum_{j<k} g_{jk}(n_i)\,/\,g_{jk}$$

Where g_{jk} = the number of geodesics (shortest) connecting jk, and g_{jk} (ni) = the number of such paths that node i is on (count also in the start-end nodes of the path).

2.3.6 | Closeness centrality

A second measure of centrality is closeness centrality. An actor is considered important if he/she is relatively close to all other actors. Closeness is based on the inverse of the distance of each actor to every other actor in the network.

$$C_c\,(n_i) = [\sum_{j=1}^{g} d(n_i, n_j)]^{-1}$$

2.4 | Linear Threshold Model

In this model, the edge weights (b_{ij}) denote the influence that node i has on node j. The chance that a node is infected depends on two quantities: the set of infected neighbors at a particular time instant and a random node-specific threshold that remains constant over time. For each $i \;\; V$, we impose the condition:

$$\sum_{j} b_{ji} \leq 1$$

The thresholds $\{\, \theta_i : i \in V \,\}$ are i.i.d. uniform random variables on [0, 1]. Beginning from an initially infected set $A \subseteq V$, the contagion proceeds in discrete time steps, as follows: At every time step, each vertex i computes the total incoming weight from all infected neighbors, i.e., $\sum_{j \text{ is infected}} b_{ji}$. If this quantity exceeds θ_i, vertex i becomes infected. Once a node becomes infected, it remains infected for every succeeding time step.

To carry out the second part of the first task, a variant of the Linear Threshold Model was applied. The nodes affected at the end of the application of the algorithm represent

the various topics. The algorithm takes as input the graph and the *top-k* keywords found in the previous step and outputs the topics for the requested year. The threshold is set to a random number in the range [0.1] to assume that each node has a certain probability of being affected. This is due to the fact that each threshold value leads to different results therefore it is difficult to establish which is correct.

```
Spread of Influence Algorithm for 2018:
Keyword: lyapunov fractal
Influenced set:
        {'nonlinear system', 'approximation', 'neural network simulation', 'numerical analysis', 'lyapunov fractal', 'numerous'}

Keyword: artificial neural network
Influenced set:
        {'embedded system', 'artificial neural network', 'approximation', 'neural network simulation', 'numerical analysis', 'algorithm', 'time complexity', 'numerous'}

Keyword: social inequality
Influenced set:
        {'social inequality'}

Keyword: mathematical optimization
Influenced set:
        {'algorithm', 'mathematical optimization', 'numerous'}

Keyword: adversarial examples
Influenced set:
        {'nips 2017 competition', 'vision models', 'known parameters', 'small changes', 'unknown parameters', 'adversarial examples', 'open question', 'similar mistakes',
        'transfer adversarial examples', 'leveraging recent techniques', 'make mistakes', 'machine learning models', 'school bus'}
```

Figure 2.7: Application of Spread of Influence Algorithm for 2018.

2.5 | Producted topics join

Once the topics have been produced, the last request of task1 is to join the topics produced in a given year following a merger strategy that deals with the possible overlaps between them. The metrics to join that I tested are:

- **Jaccard Similarity** : compares members for two sets to see which members are shared and which are distinct. It's a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations.

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- **Cosine Similarity** : is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

$$\cos(\theta) = \frac{A \cdot B}{||A||\,||B||}$$

- **Overlapping Similarity** : is a similarity measure that measures the overlap between two finite sets. It is related to the Jaccard index and is defined as the size of the intersection divided by the smaller of the size of the two sets.

$$\text{overlap}(X,Y) = \frac{|X \cap Y|}{min(|X|,|Y|)}$$

```
Produced topic join for 2018:
Keyword: lyapunov fractal
Influenced set:
        {'fuzzy control system', 'lyapunov fractal'}

Keyword: social inequality
Influenced set:
        {'social inequality'}

Keyword: adversarial examples
Influenced set:
        {'model generalization', 'training distribution', 'domain shift', 'machine learning models', 'similar mistakes', 'adversarial examples phenomenon', 'known parameters',
        'leveraging recent techniques', 'transfer adversarial examples', 'school bus', 'open question', 'small changes', 'unknown parameters', 'make mistakes', 'deep networks',
        'vision models', 'adversarial examples'}

Keyword: mathematical optimization|artificial neural network
Influenced set:
        {'loss function', 'anatomical layer', 'data compression', 'prime-factor fft algorithm', 'mathematical optimization', 'artificial neural network', 'optimization problem'}
```

Figure 2.8: Topics join for 2018 with overlapping similarity.

3

T2: Topic Tracing

The second task to be performed is directly related to the first task and asks us to trace along the timeline [2000-2018] each topic identified in the previous task. Finally, we need to create a list of merged topics (we will choose whether two topics identified in two consecutive years can be combined).

3.1 | Topic tracing over the timeline [2000-2018]

Task 1 ended with the generation of a topic dictionary for each year. In particular, we now have a dictionary which is composed of pairs (key, value) where the key = year and the value = dictionary of the topics merged in the last point of the previous task (Ref: 2.8). At this point you can apply simple functions to make statistics. The first statistic we are going to perform is to trace a single topic during the timeline [2000-2018] and see how many keywords are related to it. Next we calculate the *top-10* keywords during the time interval [2000-2018]. In the same way we calculate the *top-10* topics during the time interval [2000-2018].

```
Specific topic: linear matrix inequality
In year 2006 : 6 keywords
In year 2007 : 2 keywords
In year 2008 : 12 keywords
In year 2011 : 9 keywords
```

Figure 3.1: Specific topic trace over a year.

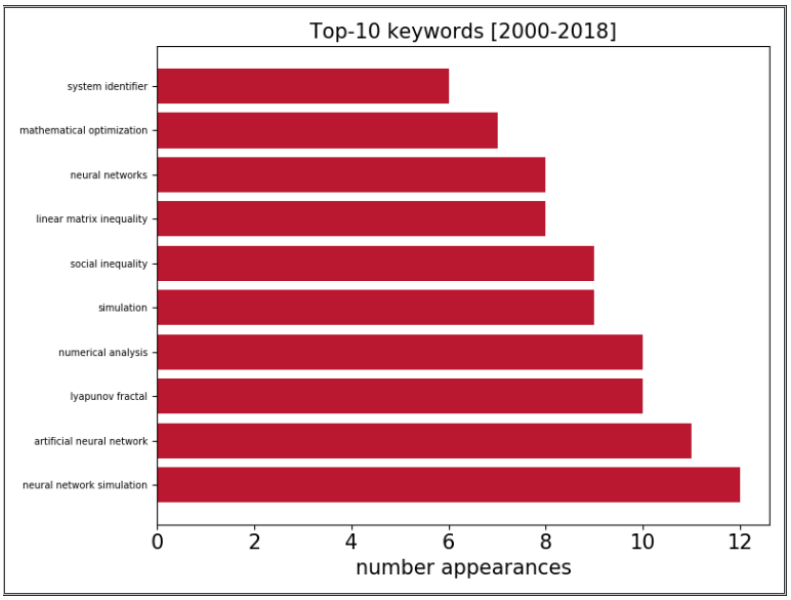

Figure 3.2: Top ten keywords that appeared the most among all topics.

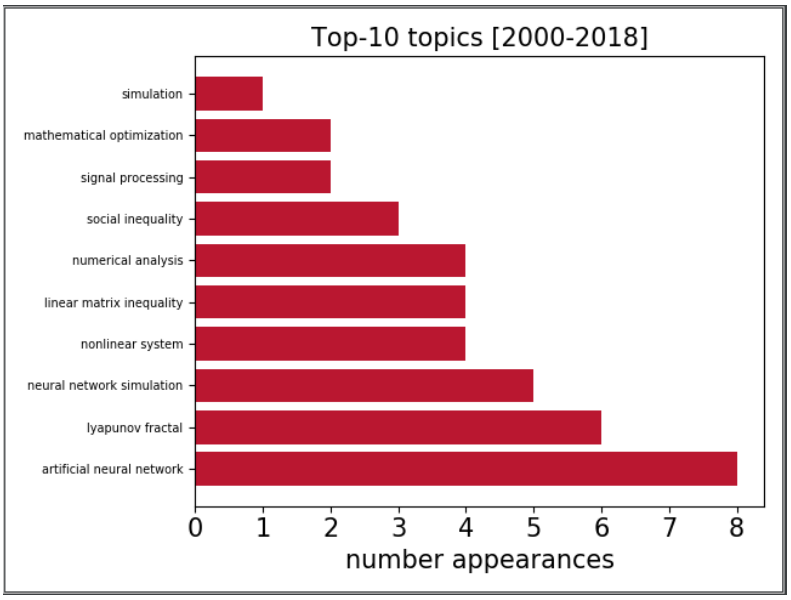

Figure 3.3: Top ten topics that appeared the most.

3.2 | Topics identified in two consecutive years

The second part of Task2 asks us if two topics identified in the two consecutive years can be merged together. Two different approaches to solve the problem have been tried:

3.2.1 | First approach : Overlapping similarity of the keyword sets

For each pair of topics in two consecutive years, I chose the best coupling always based on the Overlapping similarity of the keyword sets and taking only the pairs that had a score greater than zero.

3.2.2 | Second approach : Creating a subgraph of common nodes

For each pair of topics in two consecutive years, I calculated the best Overlapping similarity on the pairs of edges of the two induced subgraphs, created only with the common nodes of the two topics, again taking only values greater than zero.

3.3 | Final list of the merged topics

Once the topics have been identified in consecutive years, merge requires only a small code function. However, considering the two different approaches previously described we can notice some differences in the results. The figure below shows an extract of both of them being processed. For the final list you can find a text file in the project outputs.

```
2015-2016 : lyapunov fractal <--> lyapunov fractal
X: {'lyapunov fractal', 'stabilization'}
Y: {'lyapunov fractal', 'neural networks'}
overlapping similarity: 0.5

2015-2016 : algorithm <--> neural network simulation|artificial neural network
X: {'algorithm', 'markov chain', 'throughput'}
Y: {'algorithm', 'adversary (cryptography)', 'subnetwork', 'particle swarm optimization\xa0', 'neural networks', 'sample complexity', 'memristor',
'numerical linear algebra', 'artificial neural network', 'mathematical optimization', 'feature extraction', 'biogeography-based optimization\xa0',
'reinforcement learning', 'deep learning', 'machine learning', 'q-learning', 'training', 'social inequality', 'long short-term memory',
'neural network simulation', 'will dabney'}
overlapping similarity: 0.3333333333333333

2016-2017 : lyapunov fractal <--> lyapunov fractal
X: {'lyapunov fractal', 'neural networks'}
Y: {'lyapunov fractal'}
overlapping similarity: 1.0

2016-2017 : numerical analysis <--> numerical analysis
X: {'control theory', 'numerical analysis', 'neural networks'}
Y: {'numerical analysis'}
overlapping similarity: 1.0
```

Figure 3.4: Output with first approach.

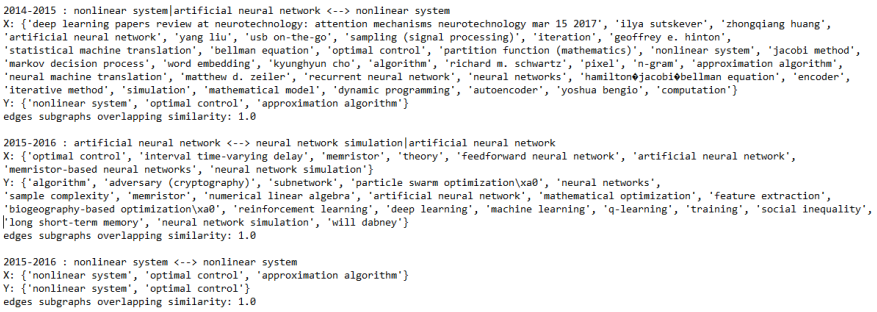

Figure 3.5: Output with second approach.

4

Conclusion

Some considerations on the project and on the results.

4.1 | Considerations

We note that in the graphs with a few nodes (therefore mainly those of the initial years) arguments formed only by a keyword were produced and therefore did not influence any other node; Whereas in graphs with multiple nodes the results are different and, in fact we notice topics made up of multiple keywords. This could also be due to the chosen threshold. For what concerns similarity measures (in addition to the defined ones) an attempt could be made by applying edit distance. The problem is that this dissimilarity measurement has a cost for each operation that performs (ex. 1 for removal, 1 for replacement etc.), and using it on large graphs like the ones we are considering we could have computational problems.

4.2 | Focus on the code

This is the structure of the code:

- **Datasets**: folder in which the datasets are contained.
- **ProjectResults**: folder where the results are stored.
- **sapienza**: main package.
  - **dataset_preprocessing**: package for processing datasets.

- **clean_dataset.py**: cleans the dataset by removing unnecessary and error-prone lines.
- **create_dataset_for_year.py**: for each dataset it creates a file for each year and subsequently a python data structure to manage it.

☐ **first_task**: package to handle the first task.

- **top_keywords_selector.py**: module to select top-keywords according to various metrics.
- **spread_of_influence_algorithm.py**: module to define the algorithm.
- **topics_merging.py**: module for merging topics according to various similarity functions.

☐ **second_task**: package to handle the second task

- **topic_tracing.py**: module to track topics during time intervals producing some statistics.
- **consecutive_year_tracing.py**: module to define similar topics of consecutive years.
- **topic_final_list.py**: module to produce the final list of topics.

☐ **utility**: package to manage some useful operations.

- **graph_builder.py**: module to build graphs through networkx.
- **measurements.py**: module to define the various measurement metrics.
- **visualization.py**: module for the graphic and textual visualization of the results.

☐ **main.py**: performs the entire flow of the project for all the years.

☐ **tasks_execution.py**: performs the flow of the project taking three parameters: <start_year> , <end_year> and <output_dir> which represent the start and end year of the tracking and folder in which to save the results.