MINING EVOLVING TOPICS

Web and Social Information Extraction Project 2018-2019

October 6, 2019

Leonardo Picchiami - 1643888
Università di Roma La sapienza
Department of Computer Science

Contents

1 Introduction **3**
1.1 Project target . . . 3
1.2 Document structure . . . 3
1.3 Technologies and libraries . . . 3
1.4 Tests configuration . . . 4

2 Datasets exploring and modeling **5**
2.1 Ds-1 dataset weight strategies and test . . . 7

3 T1: Topic Identification **9**
3.1 Top-k selection . . . 9
3.1.1 Pagerank . . . 9
3.1.2 Betweenness Centrality . . . 10
3.1.3 Normalized Centrality Degree . . . 10
3.1.4 Considerations on other methods . . . 11
3.1.5 Metrics application results . . . 12
3.1.5.1 Tables year 2000 . . . 12
3.1.5.2 Tables year 2002 . . . 13
3.1.5.3 Comments on all results and choosing the best metric . . . 14
3.2 Spread of Influence . . . 16
3.2.1 Model considerations and choice . . . 16
3.2.1.1 Generalised Threshold Model . . . 16
3.2.1.2 Susceptible-Infected-Recovered (SIR) Model . . . 16
3.2.1.3 Indipendent Cascade Model . . . 16
3.2.1.4 Tipping Model . . . 17
3.2.1.5 Linear Threshold Model . . . 17
3.2.2 Threshold strategies . . . 18
3.2.3 Models results . . . 18
3.3 Overlap merge on year . . . 21
3.3.1 Method results based on spread of influence models . . . 21

4 T2: Topic Tracing **24**
4.1 Method and metric of tracing topics along the timeline . . . 24
4.1.1 Tracing approach . . . 24
4.1.2 Tracing metric . . . 24
4.1.3 Tracing results . . . 25
4.2 Merging during the tracing . . . 27
4.3 Final list creation . . . 27

5 Conclusion **28**

1 Introduction

This document is the report of the Web and Social Information Extraction project: **Mining Evolving Topics**. This section is composed by brief explanation of the project objective, the structure of the document, a brief explanation of the technologies and libraries used to develop the system and a brief explanation of the tests.

1.1 Project target

The scope of the project is to identify and trace topics over a temporal interval. A topic can be seen as a set of keywords; you can see changes in the keyword set during the specified time period. The temporal interval, or timeline, is on [2000 - 2018].
Two dataset have been provided: *ds-1.tsv* and *ds-2.tsv*. The main dataset on which the topics must be tracked and identified is ds-1.tsv, while ds-2.tsv is an additional dataset useful for making some decisions and both can be modeled as graphs:

- *ds-1.tsv*: on the temporal interval [2000 - 2018], the dataset contains a graph for each year where a node represents a different keyword and each edge represents a relationship between two keywords. Each edge is decorated with an ordered dictionary of authors where the key represents an author and the value represents the number of times the author co-cited the keywords in his/her articles.
- *ds-2.tsv*: on the temporal interval [2000 - 2018], the dataset contains a graph for each year where a node represents an author who have published in a given year and each edge represents the co-authorship between two authors. The weight of an edge is the number of collaborations between the two authors.

To T1 task: a metric has been defined for the ranking of the keywords to retrieve the top k keywords for each year; each keyword is the most representative for a topic, selecting the number of main keywords you select the maximum number of topics you want to identify and trace. It was necessary to decide on a Spread of Influence Model to be applied to the top rank nodes (or keywords) and report the nodes influenced by them in each iteration of the algorithm: the influenced nodes represent all topics merged together. In fact, in this way, the all keywords that constitute the topics generated by the ranking algorithm are detected. And finally has been decided a merging strategy to merge the produced all topics in a given year which takes care of possibile overlaps among them, thus extrapolating the effective topics.

To T2 task: it is necessary to identify all the topics along the timeline. Then decide how and with which metric you decide to track topics and, during the tracking, decide whether topics from consecutive years can be merged together. Finally, a list must be created with all the macro-topics identified along the timeline.

1.2 Document structure

The first section, this section, is the introductory section of the report.

The second section is the analysis and study of the two datasets used for the project. After a study phase, two strategies for calculating the ds-1 dataset weights are proposed.

The third section concerns task T1. It contains the subsections concerning the metrics to generate the k topics, the spread of influence models to be applied on the k keywords to recover the all topics and merge the topics retrieved through strategy which takes care of possible overlaps among them.

The fourth section deals with task T2. It contains the subsections that describe the tracing of the topics, if and how they are eventually merged topics of consecutive years and the creation of a final list.

The fifth and last section is that of the conclusions where all the passages are summarized and the final considerations are made.

1.3 Technologies and libraries

The entire project was developed with Python 3.[1]
It was decided to use Python for the best support on graphs and operations on them which guarantees

compared to other languages such as, for example, Java.

In using the python language, the following libraries were used:

- Pandas[2]: to manage the given datasets and perform operations on them.
- Networkx[3]: to model datasets as graphs and perform operations, calculate metrics and features on the resulting graph.
- Matplotlib[4]: to display the graphs and correlations between the data extrapolated from the dataset or obtained.
- Ndlib[5]: to find for influenced nodes by Spread of Influence Algorithm starting from a set of infected nodes.

1.4 Tests configuration

To develop the project, specifically, we chose Python 3.6 and Python 3.7. It is guaranteed to work for these two versions of Python, but not for the previous ones. The tests reported were performed only with Python 3.6, but the system has also been tested for Python 3.7.

It has been developed and tested on both 32bit and 64bit systems. Specifically, Python 3.6 was used on a 32-bit system and Python 3.7 on a 64-bit system. Both systems are Linux systems.

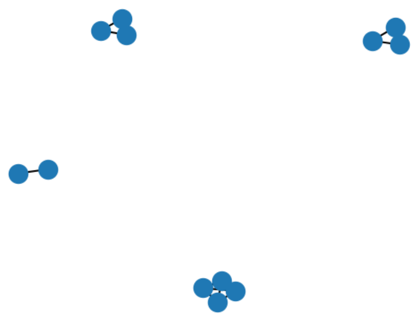
All the tests performed on the ipython netebook file are reported: in each part of the report the path of the tests relative to that part is indicated. Log files (or images in case of the plotting) regarding some tests have been saved in the appropriate folder in `output_sample` path. In the report the relative log files will not be shown in each part: each folder in `output_sample` is related to the analysis of a specific part; for example, the analysis logs of the metrics performed through the `sample_metrics_analysis.py` script in `src/metrics/analysis` are found in `output_sample/metrics_analysis`.

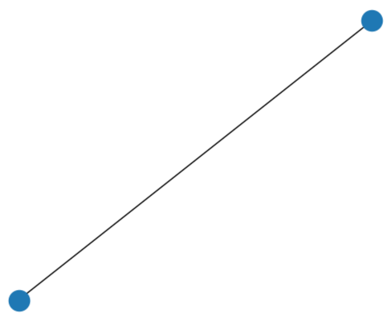
2 Datasets exploring and modeling

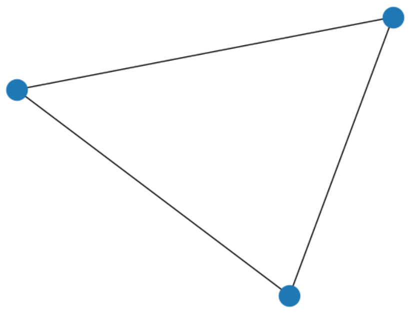
The first step was to study both datasets by modeling their data as graphs. Two datasets were used for this project: *ds-1.tsv* and *ds-2.tsv*; where each record represents an edge between two nodes in a specific year of the timeline.
The meaning of each node and each relationship depends on the dataset. In the *ds-1.tsv* dataset each node k_i represents a keyword used by an author and an edge $e = \{k_i, k_j\}$ represents the co-occurence of two specific keywords (the two nodes) being used by two different articles. Moreoever, all records have, as last field, a dictionary with key-value pairs (a, n): the key a is an author while the value n is the number of times the author uses both keywords in his/her articles. But the dataset does not provide a specific value to be used as weight of the edges, so it was decided, after an initial exploratory phase, how to set the various weights.
In the *ds-2.tsv* dataset, each node a_i reprsents an author that has published in the a specific year (year of timeline). An edge represents the relationship of co-authorship: two nodes a_i and a_j have an edge $e = \{a_i, a_j\}$ if the corrisponding authors have published articles together. In this case, the dataset provides a specific weight value for a given edge that corresponds to the number of collaborations between the two authors modeled as incident nodes.

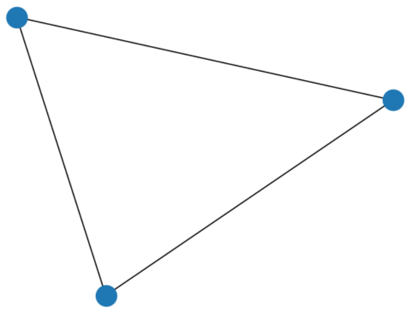
The networkx library was used to model the graphs. Networkx handles both directed graphs and undirected graphs; in addition, the library allows you to apply network analisys metrics on instatiated graphs. Some metrics, such as node betweenness, centrality degree, closeness centrality, betweenness centrality, can be measured in a undirected network while metrics such as pagerank, hits (hub and authorities), ecc, can be measured in a directed network.
However, networkx allows to calculate metrics about directed graphs on undirected graphs by temporarily modeling them in a directed graphs. Therefore, considering also that every relation of the *ds-1* dataset and *ds-2* dataset is biunivocal for each node, every year has been modeled, for both datasets, as an undirected graph.

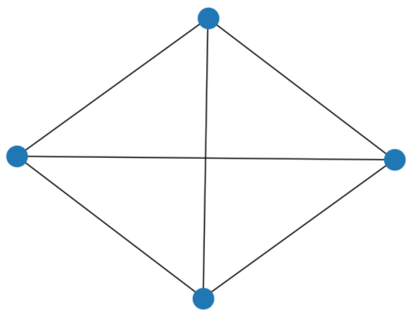
As a first step, the graphs of the *ds-1* and *ds-2* datasets have been modeled for each year of the timeline to be able to make considerations about them. It has been observed that, for every year of timeline, the year graph of *ds-1* is an n-partite graph, with some partitions having a high density of nodes and other partitions with a low density of nodes (approximately). In many cases, there is only one large component connected and the rest of the components are small. Respect to the *ds-2* dataset, the ds-1 dataset is much smaller and consequently the number of nodes is much lower. Each graph of the *ds-1* dataset has been drawn using the python matplotlib library, considering each year of the timeline so being able to make such consideration. Below is an example of a plot of a graph of the *ds-1* dataset and all its connected components, specifically relating to the year 2000:


(a) ds1-2000 all graph

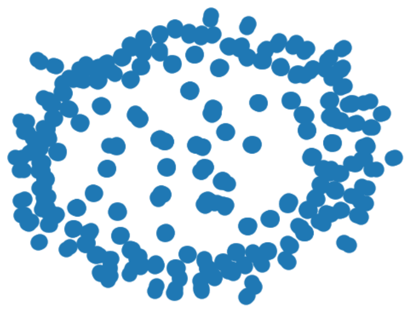

(b) ds1-2000 connected component

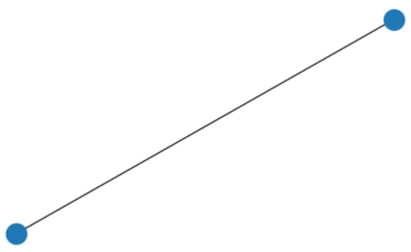

(a) ds1-2000 connected component

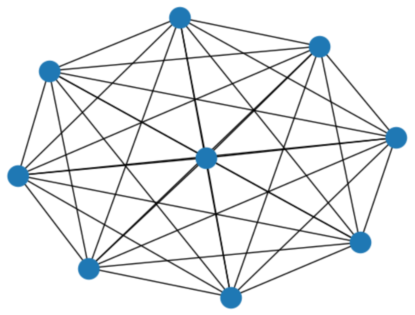

(b) ds1-2000 connected component


(c) ds1-2000 connected component

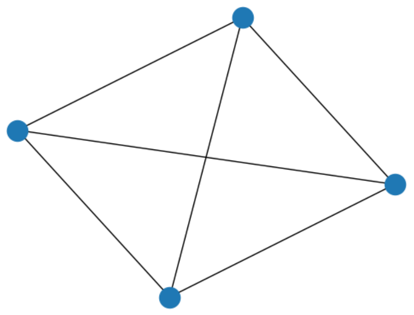
The *ds-2* dataset is also an n-partite graph, but is much bigger than *ds-1*. The dataset has so many connected components formed relatively from a few nodes each; also in this case we have some components with high density of nodes and some components with low density of nodes, however the components are all quite small. All graphs of the *ds-2* dataset has been drawn using the python matplotlib library, so being able to make such considerations. Below is an example of a plot of a graph of the *ds-2* dataset and all its connected components, specifically relating to the year 2000:


(a) ds2-2000 all graph


(b) ds2-2000 connected component



(a) ds2-2000 connected component



(b) ds2-2000 connected component

The implementation of graph modeling through the networkx library and the code that executes the plot of the graphs and connected components is present in the `src/graph` path in the `graph.py` file. In this file there is the YearGraph class whose methods manage the two datasets, extract from the dataset the keywords that represent the nodes and the keywords pairs for which there are co-citated, plot the graphs and the related connected components. Comments and explanations regarding these methods are present on the code.

To analyze the graphs of *ds-1* dataset and *ds-2* dataset, a python script `sample_graph_analisys.py` in `src/graph/exploration` package was created. The explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python sample_graph_analisys.py -h
```

Also, ipython notebook file `datasets_general_exploration.ipynb` (`datasets_general_exploration.html` for html version) has been added in the `src/graph/exploration` package to show the entire structure of all the graphs of the *ds-1* and *ds-2* datasets by plotting. Opening the file you can see the plot of each graph for every timeline year for both datasets.

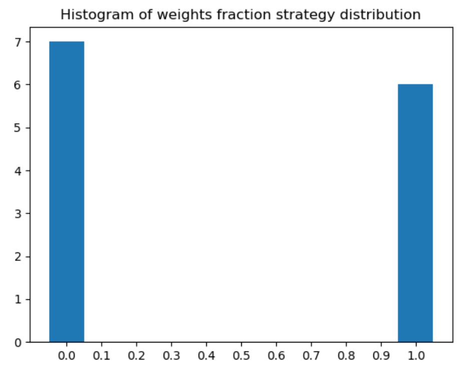
2.1 Ds-1 dataset weight strategies and test

After analyzing the meaning of the relations and the graphs of the *ds-1* and *ds-2* datasets, it was possible to decide how to assign the weights to the graphs of the *ds-1* dataset following two possible strategies:

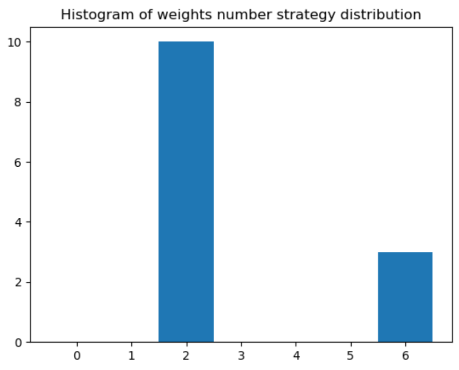
1. For this strategy, the relation between number of co-authored articles between two authors and the authors who co-cited the two keywords (the two nodes) was considered. There is a certain connection: the authors who have co-authored articles togheter could in part have co-cited the same keywords. The weights, in this case, are the fraction of co-citations on a given pair of keywords made by authors who have collaborated at least once together.
2. Alternatively, the weights can be the number of times that the keywords pair are co-cited.

Both strategies have been tested for each year of the timeline and it was observed that for the weights calculation strategy through the count of the co-citations for the relative pair of keywords, the distribution varies according to the year in question; while the weights distribution related to the strategy of calculating the fraction of co-citations between the authors who have all worked togheter respect to the total co-citations of the keywords pair is relatively uniform for each year of the timeline: the values are mainly divided between 1.0 and 0.0 with some value at 0.4, 0.6 and 0.7 depending on the year. The implementation of the two weight calculation strategies is always at the path `src/graph` in `path.py`. In the file there are further methods for calculating the weights and plotting the distributions of these along the edges of the dataset ds-1 graph. Comments and explanations regarding these methods are present on the code.

In the package `src/graph/weight_analysis` is presented a jupyter notebook file, named `all_year_` `weights_analisys.ipynb` (`all_year_weights_analisys.html` for html version), in which is the histogram with the distribution of the resulting weights and the video output of the dictionary whose keys are edges and the values are the associated calculated weight for both strategies. Below is an example of a plot of two histograms relating to both strategies for calculating specific weights at a fixed year without video output, in particular relating to the year 2000:



(a) fraction stategy for year 2000



(b) number stategy for year 2000

In the same package, a python script was created called `sample_weights_analysis.py` for plotting (saving or not) the histograms for both strategies related to a specific year. The explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python sample_weights_analysis.py -h
```

3 T1: Topic Identification

This section concerns the identification of the topics in a year of the timeline: use of a metric to retrieve the top k nodes (generation of k topics), apply a spread of influence algorithm to obtain all the keywords that make up the topics and merge the topics, if possible, with an overlapping-based merging strategy.

3.1 Top-k selection

To select the top-k nodes, three possible metrics were tested: pagerank, hits and betweenness centrality (node betwennees centrality). Having hypothesized to follow two possible strategies of assignment of the weights to the graphs of the dataset ds-1, it has been chosen to use **Pagerank** as a ranking algorithm because, after studying the pagerank algorithm structure and the implementation of the algorithm in networkx, it has been noticed that the weights of the edges of the graphs are used to calculate the weights of the stochastic graphs which are in turn the values of the adjacency matrix. It is interesting to see the effect of choosing the weights calculation. As a second choice for the analysis of the best metric it was decided to choose a metric calculated on undirected graphs that does not use weights for calculating the ranking based on paths of the graph: **Betweenness Centrality**. As third choice it was decided to choose a metric that is calculated on undirected graphs or directed graphs, but that does not use weights for calculating the ranking and is based on degree of node: **Normalized Centrality Degree**. For all three metrics, implementations in the networkx library were studied and used, then compatible with the implemented graphs.
Each of the top k nodes obtained consists of a generated topic, ie each node should be the most representative node of a given topic. Choosing the k value you choose the number of topics to be generated.

3.1.1 Pagerank

Pagerank is an algorithm used by Google Search to rank web pages in their search engine results. It is applied on graphs (in case of the web, it is modeled as a graph); PageRank computes a ranking of the nodes in the graph based on authority (incoming links measure).
Generally, the rank can be calculated in two ways: either algebraically or iteratively. Using the networkx library, the implementation taken into consideration uses the iterative version. The iterative version employs the power method to calculate the rank; to ensure convergence the matrix (the graph) must follow one of the two following conditions:

a. A is square, real and symmetric matrix.

b. A is square, stochastic and irreducible matrix.

A *stochastic matrix* is a matrix whose sum of each column (left stochastic) or each row (right stochastic) is equal to 1. A *irriducible matrix* if the indices not be divided into two disjoint nonempty set (or equivalent to say that the relative graph not has disconnected components).
In the ds-1 dataset each graph consists of many connected components, which means that it is disconnected; to risolve this problem, the pagerank algorithm uses teleporting, so the condition is respected.

The source code of networkx pagerank is avaliable: `https://networkx.github.io/documentation/stable/_modules/networkx/algorithms/link_analysis/pagerank_alg.html`

In this implmentation of pagerank, an undirected graph is transformed into a directed graph and then into a direct stochastic right graph. In this way the graph (hence the matrix) is right stochastic. The source code of the networkx stocastich graph: `https://networkx.github.io/documentation/latest/_modules/networkx/generators/stochastic.html`.

Since, in pagerank algorithm, the graphs of the *ds-1* dataset are temporary modeled as directed graphs, each graph not has a distinction between number of in-links and number of out-links. From this point of view the choice to use pagerank could be senseless, but the different size of the connected components of the graphs can give greater guarantees of an real ranking. If the graphs have many connected components with variable dimensions, the number of out-links between nodes can be quite variable. Furthermore, the weights of the stochastic graph, values of the adjacency matrix, depend on the weights of the input graph, therefore it is useful to analyze the efficiency of this technique also according to the choice of the weights calculation strategy.

3.1.2 Betweenness Centrality

The betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized. This measure for each vertex is the number of these shortest paths that pass through the vertex.
The betweenness centrality of node v is give by:

$$C_b(n_i) = \sum_{j<k} \frac{g_{jk}(n_i)}{g_{jk}}$$

Where:

- n_i is the node i.
- g_{jk} is the number of geodesics paths (shortest paths) connecting the nodes j and k.
- $g_{jk}(n_i)$ is the number of geodesics paths that node i is on (counting also in the start-end nodes of the path).

The source code of networkx implementation: `https://networkx.github.io/documentation/latest/_modules/networkx/algorithms/centrality/betweenness.html`

Studying and analyzing the source code of betweenes centrality, it has been observed that it is possible either to consider the weight of the graph edges or not. Depending on this choice, in fact, the betweenness centrality implementation uses the shortest path algorithm or the dijkstra algorithm to calculate the shortest path beetween two nodes.
In this case study, it was decided to not consider the graph weights and then to consider the graph as unweighted. Ideally the k nodes that are included in many paths could be representative for k different topic to be identified.

3.1.3 Normalized Centrality Degree

The degree centrality is a measure of centrality in a graph based on degree of nodes, it is defined as the number of links incident on a node (i.e., the number of ties that a node has). The degree centrality can be calculated both on directed graphs on undirected graphs: for undirected graphs we have the simple sum of links incident on the nodes, while for directed graphs we have in-degree centrality, number of link directed to the nodes, and out-degree centrality, the number of links that the nodes direct to others.

In this case the degree is the number of keywords with which a given keyword is co-cited. It is interesting to analyze this metric since, on one side, nodes with a high degree can be central to the identification of topics and therefore the first k nodes retrivied could be most rappresentative to k generated topics. On the other side we know that every *ds-1* dataset graph is composed of many connected components and we may not have a real rank because the nodes could be ordered on identical scores. The weights are, obviously, not taken into account for the degree centrality calculation. The degree of centrality for each node n is calculated as follows:

$$C_d(n) = deg(n)$$

The source code of networkx centrality degree: `https://networkx.github.io/documentation/stable/_modules/networkx/algorithms/centrality/degree_alg.html`.

Using the function implemented in the networkx library, the normalized version of the centrality degree calculated as follows was used:

$$C'_d(n) = \frac{C_d(n)}{(N-1)}$$

Where:

- $C_d(n)$ is the degree of node n.
- $(N-1)$ is the number of nodes N in the graphs minus one.

3.1.4 Considerations on other methods

In the choice of methods, other metrics or groups of metrics were also considered, without actually testing and implementing them:

- **HITS**: the algorithm returns the rank by authority (out-going links) or by hub (in-coming links). However, the graphs are modeled as undirected graphs and, although networkx allows to use metrics developed for undirected graphs on directed graphs, we will not have a substantial difference between in-coming edges and out-going edges in temporary modeling as a directed graphs; therefore we will not have a substantial difference between rank by authority and rank by hub.
  The hits algorithm is implemented using the power method as for pagerank in networkx library. The considerations for pagerank and for hits are very similar, if not the same, so it is very likely that the rank presented using pagerank ad rank presented using hits is the same (or most similar). Therefore, it was decided to discard this metric.

  The source code of hits networkx implementation: `https://networkx.github.io/documentation/latest/_modules/networkx/algorithms/link_analysis/hits_alg.html`.

- **Closness Centrality**: the key idea behind this metric is that an actor (the node) is considered important if he/she is relatively close to all other actors. The closeness is based on the inverse of she distance of each actor to every other actor in the network. The formula for calculating this metric is as follows:

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

  where:

  - n_i is the node on which the metric is calculated.
  - d is the distance function.
  - n_j is the other nodes on which the distance is calculated.

  The networkx source code of this metric has been analyzed and has been studied: `https://networkx.github.io/documentation/latest/_modules/networkx/algorithms/centrality/closeness.html`.
  The distance function in networkx is the shortest path calculation function (depending on whether the weights of the graph are considered or not, the function is simple shortest path function for unweighted graph or dijsktra function for weigthed graphs). It is the same way of calculating the distance used in betwenness centrality, so it was decided not to use this metric.

- **Other metrics based on in-links and out-links**: there are other metrics based on in-links and out-links, as pagerank and hits, called prestige. There are other tipes of prestige: based on out-going edges called *influence*, based on in-coming edges called *support*, *influence domain* as number (or proportion) of all other nodes which are connected by a path to this node, and *brokers* based on measure of graph connectivity, for example, in term of reachability as in kpp-neg algorithm (not present in networkx). The main consideration is that the graph is undirected and these metrics are all for directed graphs; as for the previous metrics consiterated for directed graphs, networkx it allows to apply metrics for undirected graphs on directed graphs but an undirected graph modeled as a directed graph has for each node the same in-coming edges and out-going edges. For this reason it was decided not to use these types of metrics any further, not having a different number of in-coming and out-going edges it is not possible to further exploit the actual potentiality. Furthermore, the kpp-neg algorithm is very similar to betweenness centrality which is a metric that has already been used, implementing it based on netwokx graphs could lead to the same result.

3.1.5 Metrics application results

This subsubsection covers the results derived from the three metrics, analyzed on all graphs of the ds-1 dataset along the timeline. In the ds-1 dataset there are small graphs, such as the year 2000, medium-sized graphs, such as the one in 2007, and the large one like that of the year 2011. In this way it is possible to analyze the effect and functionality of the various ranking metrics taken into consideration on different graphs of different sizes with a variety of connected components in terms of size and structure. The possible values of k are [5, 10, 20, 100], so $k = 100$ will always be taken in order to analyze the metrics considering the maximum number of topics that can be generated. Each graph is composed of connected components of various sizes and connections, so it is easy for the chosen metrics to have groups of nodes with the same rank score; therefore it is possible to define that a metric is good if it returns the different score for the rank from the first positions to the last ones as much as possible. This highlights the main nodes that should be most representative for the generated topics. Finding a compromise between the results obtained and the definition given, it was possible to determine whether a metric used is good or not: if it returns a different score for the first 10 or 20 nodes more or less depending on the graph, its size and structure (the first 5 nodes for smaller graphs) and works, possibly, with graphs of any size.

The implementation of metrics modeling through the networkx library is present in the `src/metrics` path in the `metrics.py` file. In this file, the Metrics class is present with methods to obtain the ds-1 dataset graph from the YearGraph class, to apply the algorithms related to the metrics chosen and extrapolate the top k nodes with k in input. Comments and explanations regarding these methods are present on the code.

Furthermore, a python script weas created at the path `src/metrics/analysis/` called `sample_metrics_analysis.py` to test every possible metric described above for a given *year* recovering the *top-k* nodes for a given k between 5, 10, 20 and 100. The explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python sample_metrics_analysis.py -h
```

To show the results for each year of the timeline, an ipython notebook file was created that shows two tables for each year: one table with rank for the metrics of betweennes centrality and degree centrality, while the other table with one rank for the pagerank metric with the weights calculation strategy as a fraction citations, for each keywords pair, between authors who have collaborated at least once together and one rank for the pagerank metric with weights calculation strategy as number of co-citations made on the keywords pair for every keywords pair. The file is present at the path: `src/metrics/analysis/` called `all_year_metrics_analysis.ipynb` (`all_year_metrics_analysis.html` for html version).
The following are examples of output of this part: specifically the tables concerning the year 2000 (reduced dimensions) and those concerning the year 2002 (average dimensions):

3.1.5.1 Tables year 2000

Rank	Node Betweenness	Degree Centrality
1	(artificial neural network, 0.0)	(periodic solution, 0.2727272727272727)
2	(nonlinear system, 0.0)	(43.80.+p, 0.2727272727272727)
3	(simulation, 0.0)	(delay, 0.2727272727272727)
4	(support vector machine, 0.0)	(lyapunov functional, 0.2727272727272727)
5	(feature vector, 0.0)	(nonlinear system, 0.18181818181818182)
6	(periodic solution, 0.0)	(simulation, 0.18181818181818182)
7	(43.80.+p, 0.0)	(support vector machine, 0.18181818181818182)
8	(delay, 0.0)	(feature vector, 0.18181818181818182)
9	(neural network simulation, 0.0)	(motion estimation , 0.18181818181818182)
10	(motion estimation , 0.0)	(experiment, 0.18181818181818182)
11	(experiment, 0.0)	(artificial neural network, 0.09090909090909091)
12	(lyapunov functional, 0.0)	(neural network simulation, 0.09090909090909091)

Rank	Pagerank_fraction_weigthed	Pagerank_number_weighted
1	(nonlinear system, 0.1449271498783759)	(artificial neural network, 0.08333333333333333)
2	(simulation, 0.1449271498783759)	(nonlinear system, 0.08333333333333333)
3	(support vector machine, 0.1449271498783759)	(simulation, 0.08333333333333333)
4	(feature vector, 0.1449271498783759)	(support vector machine, 0.08333333333333333)
5	(motion estimation , 0.1449271498783759)	(feature vector, 0.08333333333333333)
6	(experiment, 0.1449271498783759)	(periodic solution, 0.08333333333333333)
7	(artificial neural network, 0.02173951678829072)	(43.80.+p, 0.08333333333333333)
8	(periodic solution, 0.02173951678829072)	(delay, 0.08333333333333333)
9	(43.80.+p, 0.02173951678829072)	(neural network simulation, 0.08333333333333333)
10	(delay, 0.02173951678829072)	(motion estimation , 0.08333333333333333)
11	(neural network simulation, 0.02173951678829072)	(experiment, 0.08333333333333333)
12	(lyapunov functional, 0.02173951678829072)	(lyapunov functional, 0.08333333333333333)

3.1.5.2 Tables year 2002

Rank	Node Betweenness	Degree Centrality
1	(bayesian methods, 0.0)	(independent component analysis, 0.22222222222...
2	(monte carlo methods, 0.0)	(blind source separation, 0.2222222222222222)
3	(filtering algorithms, 0.0)	(important method, 0.2222222222222222)
4	(lyapunov function, 0.0)	(overcomplete situation, 0.2222222222222222)
5	(independent component analysis, 0.0)	(unsupervised learning, 0.2222222222222222)
6	(blind source separation, 0.0)	(bayesian methods, 0.16666666666666666)
7	(important method, 0.0)	(monte carlo methods, 0.16666666666666666)
8	(overcomplete situation, 0.0)	(speech processing, 0.16666666666666666)
9	(speech processing, 0.0)	(intellectual property, 0.16666666666666666)
10	(business models, 0.0)	(program product, 0.16666666666666666)
11	(intellectual property, 0.0)	(patent trademark, 0.16666666666666666)
12	(program product, 0.0)	(signal processing, 0.16666666666666666)
13	(patent trademark, 0.0)	(organization company, 0.16666666666666666)
14	(signal processing, 0.0)	(filtering algorithms, 0.05555555555555555)
15	(state estimation, 0.0)	(lyapunov function, 0.05555555555555555)
16	(global exponential stability, 0.0)	(business models, 0.05555555555555555)
17	(unsupervised learning, 0.0)	(state estimation, 0.05555555555555555)
18	(consider thinking, 0.0)	(global exponential stability, 0.0555555555555...
19	(organization company, 0.0)	(consider thinking, 0.05555555555555555)

Rank	Pagerank_fraction_weigthed	Pagerank_number_weighted
1	(overcomplete situation, 0.05780343855118794)	(bayesian methods, 0.05434528253466711)
2	(unsupervised learning, 0.05780343855118794)	(monte carlo methods, 0.05434528253466711)
3	(bayesian methods, 0.05780343855118793)	(signal processing, 0.05434528253466711)
4	(monte carlo methods, 0.05780343855118793)	(filtering algorithms, 0.05263157894736842)
5	(filtering algorithms, 0.05780343855118793)	(lyapunov function, 0.05263157894736842)
6	(lyapunov function, 0.05780343855118793)	(independent component analysis, 0.05263157894...
7	(independent component analysis, 0.05780343855...	(blind source separation, 0.05263157894736842)
8	(blind source separation, 0.05780343855118793)	(important method, 0.05263157894736842)
9	(important method, 0.05780343855118793)	(business models, 0.05263157894736842)
10	(speech processing, 0.05780343855118793)	(state estimation, 0.05263157894736842)
11	(intellectual property, 0.05780343855118793)	(global exponential stability, 0.0526315789473...
12	(program product, 0.05780343855118793)	(unsupervised learning, 0.05263157894736842)
13	(patent trademark, 0.05780343855118793)	(consider thinking, 0.05263157894736842)
14	(signal processing, 0.05780343855118793)	(organization company, 0.05263157894736842)
15	(state estimation, 0.05780343855118793)	(overcomplete situation, 0.05263157894736841)
16	(global exponential stability, 0.0578034385511...	(intellectual property, 0.05263157894736841)
17	(organization company, 0.05780343855118793)	(program product, 0.05263157894736841)
18	(business models, 0.008670772314902552)	(patent trademark, 0.05263157894736841)
19	(consider thinking, 0.008670772314902552)	(speech processing, 0.04749046818547229)

3.1.5.3 Comments on all results and choosing the best metric

The comments on the results that have been reported below are related to all the results present in the notebook file.

- **Node Betweenness**: this metric does not work with small and medium-small graphs, in fact for graphs from the years 2000 and 2002 the node betweenness does not returns a ranking (the value of rank is 0 for all node). With medium-sized graphs, like the graph for the year 2007, the metric returns a rank for the first 6-11 nodes (depending on the graph), while for the other nodes the rank value is 0. For the big and medium-big graphs, like the graph for the year 2012, the metric returns a rank for the first 10-20 nodes (depending on the graph) except for the graphs of the last 4 years of the timeline where a rank returns for the first 40-60 nodes while for the other nodes the rank value is 0. For small graphs the metric is absolutely unusable, while it has a poorly acceptable result only for graphs of maximum dataset size. It would therefore be unusable for most graphs; for this reason it was decided to discard this metric.

- **Degree Centrality**: this metric returns one rank score for each node of each graph for each year of the timeline. In any graph of any size (small, medium and large) the degree of centrality, except for the first or the first two nodes of the ranking (depending on the graph), already groups the main nodes in the same score not allowing to understand which ones they are the really representative nodes even with a very small k. For this reason and in accordance with the definition of good metrics given above, it was decided to discard this metric.

- **Pagerank - fraction weights**: this metric returns one rank score for each node of each graph for each year of the timeline. In this version, pagerank returns a good rank for medium and large graphs. It gets worse when the size of the graphs decreases, for example for the year 2002, by grouping the nodes in groups with the same score not highlighting the top k nodes. Since this metric works well on medium and large graphs, and does not work very badly in small graphs, it has not been discarded.

- **Pagerank - number weights**: this metric returns one rank score for each node of each graph for each year of the timeline. In this version, pagerank returns a good rank for medium and large graphs. It gets worse when the size of the graphs decreases, for example for the year 2002, by grouping the nodes in groups with the same score not highlighting the top k nodes. Moreover the score for all the nodes of the year 2000 is identical, therefore it does not return a rank for that graph proving to not work for really small graphs. For this reason and in accordance with the definition of good metrics given above, it was decided to discard this metric.

3 metrics of the 4 proposals were discarded, choosing as metrics for the identification of the top k nodes (k generated topics) *pagerank* with *with the weights calculation strategy as a fraction citations, for each keywords pair, between authors who have collaborated at least once all of them.*

It was interesting to see how on most graphs, undirected graphs, metrics designed on direct graphs have shown a better functioning. This is assumed to also be due to the structure of the graphs and the structure of the connected components.

3.2 Spread of Influence

To identify all the keywords of the topics, some spread of influence models were analyzed: the algorithms presented in the laboratory activities have been studied proposing, after choosing which ones to test, different threshold calculation strategies for each one. The nodes that are obtained iteration by iteration are all the keywords that form the k generated topics. The result is equivalent to the union of all the k topics.
For all the models analyzed, the implementation of ndlib has been studied which allows their application on networkx graphs.

3.2.1 Model considerations and choice

The considerations extend to the 5 models presented in the laboratory activities: Generalised Threshold Model, Susceptible-Infected-Recovered (SIR) Model, Independent Cascade Model, Tipping Model, Linear Threshold Model.

3.2.1.1 Generalised Threshold Model The Generalized Threshold Model is the most general of all: specifically, it is the generalization of both the Linear Threshold Model and the Independent Cascade Model which, in turn, are the generalization of the Tipping Model and the SIR Model.
In fact, given a node v, a monotone function $f_v : 2^{n_v^{indegree}} \rightarrow [0,1]$, a threshold value $\theta_v \in [0,1]$ and an active set A_t at time t, the node v is infected at time step t if $f_v(a_t) \geq \theta_v$.
In our application domain this type of model is too general. Although this spread of influence model would be applicable, it is not necessary to generalize up to this point. The monotone functions defined in the other models are acceptable for our purpose. Consequently discarded this model without carrying out any test.

3.2.1.2 Susceptible-Infected-Recovered (SIR) Model The Susceptible-Infected-Recovered Model, or SIR, is a spread of influence model based on the division of nodes into three categories. The categories are:

- **Susceptible**: node able to be infected.
- **Infected**: node that has been infected.
- **Recovered**: node no longer able to infect or be infected.

The infection procedure is as follows: at time-step t, only infected nodes at time-step *t-1* can infect its susceptible neighbours with probability β (tau in the most popular model implementations). At time step t, the infected nodes are recovered with probability α (gamma in the most popular implementations). The ndlib source code of this model: `https://github.com/GiulioRossetti/ndlib/blob/master/ndlib/models/epidemics/SIRModel.py`.

In our application domain it is only necessary to see which nodes are infected, but not to know which nodes are recovered at the time. Therefore you don't even need to set a threshold to sample the infected nodes that will be recovered. However, it is possible to adapt it to our needs without considering the recovered nodes or to put a probability such that at each step every infected node is recovered but, in the latter case, we would have exactly the Independent Cascade Model. For this reason, this model is too specific and operates at too low a level for the goal of this subtask. As a result, it was decided to discard this model.

3.2.1.3 Indipendent Cascade Model The Indipendent Cascade Model is a spread of influence model based on the division of nodes into three categories:

- **Susceptible**: node able to be infected.
- **Infected**: node that has been infected.
- **Removed**: node no longer able to infect or be infected.

However, the infection procedure is different from the SIR model: at time-step t only the infected nodes at time-step *t-1* can infect its susceptible neighbours with probability β (or edge threshold). At time-step *t*, the infected nodes (nodes that were infected in the previous step) are always recovered (or with probability 1). Therefore, it is possible to see this model as a two-state model:

- **Active**: infected (also removed) nodes.
- **Inactive**: node able to be actived (infected).

When a node is active, it cannot transit to an inactive state (monotonic model). It is considered as a generalization of the SIR model. The algorithm start with an initial set of seeds s_0 and for any inactive node $v_i \in V$ its neighbours try to activate it. It stops when $s_{t-1} = s_t$. All this is visible by analyzing the source code: `https://github.com/GiulioRossetti/ndlib/blob/master/ndlib/models/epidemics/IndependentCascadesModel.py`.
In this way it is possible to have a much simpler model that provides only which nodes have been influenced, which is exactly what this sub-task needs. For this reason it was decided to test this type of model, thus being able to see the efficiency of a simple sampling-based model.

3.2.1.4 Tipping Model The Tipping Model is a spread of influence model that is based on the activation of a node if a portion of its in-coming links is active (or infected). The quantity that indicates the portion (fraction) of active neighbors sufficient to influence a node is called threshold; once a threshold value is set, the model exploits a threshold function that returns the fraction of nodes that are influenced by a given node. The model uses also an activation function, based on the threshold function, which, given as input a set of nodes active (subset of V) at time *t* returns a new set (always subset of V) containing the old active nodes added to the new ones. Therefore, also in this case a seed or target group of nodes already influenced (or active) is associated. The model has two possibily states for the nodes:

- **Active**: node that has been influenced and that can influence other nodes.
- **Inactive**: node able to be influenced.

When a node is active, it cannot transit to an inactive state (monotonic model). The algorithm stops when the set of actives node at time *t+1* is equal at the set of actives nodes at time *t*. The implementation in ndlib library of tipping model: `https://github.com/GiulioRossetti/ndlib/blob/master/ndlib/models/epidemics/ThresholdModel.py`.
The implementation shows that, at each iteration, each inactive node is activated if the fraction of in-coming nodes is at least equal to the threshold. Furthermore, the algorithm takes into account the number of in-coming nodes without considering the weight of each edge; in this way, only the connection of the nodes based on the graph structure is considered. It was decided to test the models.

3.2.1.5 Linear Threshold Model The Linear Threshold Model is a spread of influence model that is based on the nodes activation if the sum of the weights of the in-coming links from active nodes is at least equal to the threshold. This model has a threhsold, a threhsold function and an activation function and can be considered as a generalization of the Tipping Model. The nodes has two possibily states:

- **Active**: node that has been influenced and that can influence other nodes.
- **Inactive**: node able to be actived (infected).

The generalization consists in using graph weights as a measure quantity to determine whether a node can be influenced or not. It is possible to see the Tipping model in terms of weights, considering the weight of each in-coming edge of a node as $\frac{1}{n}$ where n is the number of in-coming edges and, also in this case, the sum of weights of its in-coming edges coming from active neighbor nodes is at least equal to the threshold. The source code for this model on networkx graphs was presented during the course lab activities.
The strategy adopted for calculating the graph weights was designed to highlight the most relevant nodes and could prevent the model from influencing keywords important for topics. For this reason, it was decided to discard this model.

3.2.2 Threshold strategies

In this part, the word threshold means both the threshold of the Tipping Model and the probability of influencing the Independent Cascade Model depending on the model we are talking about.

The spread of influence models consider directed graphs but previously modeled graphs are undirected. The ndlib library supports the application of models also on unirected graphs.
The threshold calculation cannot therefore be applied in terms of in-coming or out-going edges since each node, modeling the graph as a directed graph, would have the same in-coming and out-going edges. The possibility of using the ds-2 dataset or both datasets for the threshold decision for each node (or for each edge) was excluded. From the two datasets it is possible to derive the number of collaborations of two authors, if two or more authors collaborated and how many co-citations etc. But have noticed that these strategies are, or could be, optimal to find the most representative nodes for a given number of topics (each node represents a different topic) not finding a connection between the information they provide and the spread of influence task; consequently, it was decided to focus on the relationships expressed by the structure (connections) of the graph: the degrees of the nodes.
Three different ways have been tested to calculate the threshold for both the Tipping Model and the Independent Cascade Model. Two of these are based on node degrees while the third is fixed at 0.5, used primarily as an intermediate measure.

For the *Tipping Model*:

1. **threhsold**: $\frac{1}{deg(n)}$ where n is the node. The threshold is very low; if at least one node has an active neighbor, then it is activated.
2. **threshold**: 0.5. The threshold is of medium size; if half of the neighbors of a node are active, then that node also becomes active.
3. **threshold**: $1 - \frac{1}{deg(n)}$ where n is the node. The threshold is very high; approximately, a node should be activated if most of its neighbors are active.

The calculation of the threshold linked to the degree of a single node has been done based on the type of implementation that ndlib provides. The configuration of the model required by the Tipping Model is carried out on the nodes, therefore it was decided to implement the threshold calculation by referring to the degree of each single node.

For the *Cascade Model*:

1. **threhsold**: $\frac{1}{mean(e)}$ where $mean(e) = \frac{deg(n)+deg(m)}{2}$, m and n are nodes such that $(m, n) \in E$ and E is set of graph edges. The threshold is very low; the probability of influencing a node is very small.
2. **threshold**: 0.5. The threshold is of medium size; there is an equal probability of influencing and not influencing a node.
3. **threhsold**: $1 - \frac{1}{mean(e)}$ where $mean(e) = \frac{deg(n)+deg(m)}{2}$, m and n are nodes such that $(m, n) \in E$ and E is set of graph edges. The threshold is very high; the probability of influencing a node is very big.

The calculation of the threshold linked to the degrees of the two nodes connected by an edge was made based on the type of implementation provided by ndlib. The configuration of the model required for the Indipendent Cascade Model is carried out on the edges, therefore it was decided to implement the calculation of the threshold by referring to the degrees of the two nodes that are connected by an edge.

3.2.3 Models results

All three strategies of both models were tested for every possible k and all results were observed over the entire timeline.

As already introduced in the previous description, the **Independent Cascade Model** uses the threshold as a probability threshold to sample a node or not, then to influence or not a node. The *threshold*

1 provides a very low threshold, in fact in all the results presented there is a small number of sampled nodes. The larger the graph, the more nodes can be considered, and the more we can see that the number of nodes influenced is low (compared to the total number of nodes).
The *threshold 2* provides of a medium size mass of probability. This case has the same probability of influencing and not influencing a node, consequently the number of nodes is influenced (or activated) are many more than the *threshold 1* having a medium number of keywords for the topics.
The text *threshold 3* provides a high probability mass. It is the probability opposite to the *threshold 1* and being that very low, this is very high. The number of nodes involved, compared to the previous two, is much higher; therefore in terms of maximizing the spread of influence it is a good choice.
There is, however, an intrinsic problem with the independent cascade model due to the nature of the model itself: each node influences another node with a certain probability and with the opposite probability does not influence it. Different executions of this model, with the same set of seed nodes, will sample a different number of keywords. So the topics will have more or less keywords depending on the execution. However, the *threshold 3*, having a very high probability of sampling a node, will probably have the same keywords in different runs.

The **Tipping Model** uses the threshold to determine the minimum fraction of active neighbors to activate a node. The *threshold 1* provides a threshold that maximizes the number of nodes involved in each iteration (compared to other thresholds). With only one neighbor active, the node is activated: the greater the size of the graph, the greater the number of influenced nodes. In terms of maximizing the spread of influence, it is a good choice.
The *threshold 2* provides a medium-sized threshold, which indicates that each node will be influenced if at least half of its neighbors are active. In fact, the number of nodes influenced is less than the *threshold 1*.
The *threshold 3* is the opposite fraction of the *threshold 1*, $1-$ *threshold 1*. The *threshold 1* is very low, so the *threshold 3* value is very high. The number of nodes affected should be very low, the lower than the others. However the number of influenced nodes can be less than the *threshold 1* but greater than the *threshold 2*. This is because, depending on the structure of the graph (when the node has only one neighbor), the threshold in this case could be 0.0 and therefore the node is influenced a priopri. Therefore, the model does not work properly and this threshold calculation strategy must be discarded; for a correct implementation an exception should be launched, but it is interesting to see the result in order to observe this issue.

This model, once the threshold has been set, always returns the same number of influenced nodes for each run. This guarantees depending on the threshold used, that all the nodes that can be influenced for that configuration are recovered. For this reason the Tipping Model is ideally favored at the Independent Cascade Model.

The implementation of the spread of influence models through the ndlib library on networkx graphs is present in the `src/spread_influence` package in the `spread_of_influence.py` file. In this file the SpreadOfInfluece class is present with methods that apply the necessary preprocessing, choose the threshold calculation strategy, calculate the selected treshold and apply the spread of influence model of the ndlib library. Comments and explanations regarding these methods are present in the code.

All tests performed with this model are reported in the ipython notebook present in the path `src/spread_influence/analysis/` called `all_year_spread_of_influence_analysis.ipynb` (`all_year_spread_of_influence_analysis.html` for html version); the test are reported for all possibily *k* value and for all years of timeline. Below is an example of the output reported in the notebook regarding the results obtained with this model:

```
SPREAD OF INFLUENCE RESULT WITH CASCADE MODEL, SEED TOP 5 NODES AND
 HALF THRESHOLD CALCULATION STRATEGY.

iteration 0 : ['state-space methods', 'filtering algorithms',
               'state estimation', 'deconvolution', 'signal processing']
len_iteration: 5

iteration 1 : ['mathematics', 'filtering', 'stochastic processes', 'fading',
               'linear systems', 'monte carlo methods', 'particle filters',
               'smoothing methods']
```

```
len_iteration: 8

iteration 2 : ['computer simulation', 'bayesian methods']
len_iteration: 2
```

The output is therefore the result of the spread of influence iteration by iteration.

In the path `src/spread_influence/analysis/` is present a python script, called `sample_influence_analysis.py`, useful for analyzing the results of both models for a given year by setting a model, k, the strategy for calculating the threshold and the year. More explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python sample_metrics_analysis.py -h
```

The efficiency of both models and the related threshold calculation strategy will be further evaluated in the following subsection by applying the merging method based on overlapping on the nodes influenced by both models for each possible threshold described above. The best model, and the best configuration of the model, will then be decided by adding to the considerations made above other considerations obtained through the application of the merging method based on overlapping.

3.3 Overlap merge on year

The last subtask to identify topics in a given year is the merging of topics (all keywords retrivied) using a strategy based on overlapping. Through these strategies it is possible to detect the effective topics for a given year. To recap: identification of the k nodes, the k most representative keyword to generate k topics are identified, one per topic; subsequently through the diffusion of the models of influence it is possible to identify the keywords that make up all the topics; in the end, through this technique, the arguments are combined to merge similar or related topics identify the actual topics present in a year.

Two methods of orverlap-based merging have been considered: merging topics if they have a certain keyword fraction in common and the merging method of managing communities based on overlapping.

The **common keywords** method is based on the fraction of keywords they have in common. Establishing a minimum threshold, that is the fraction of common keywords, and given a pair of keywords, if the number of keywords in common of at least one of the two with the other is at least equal to the threshold, then they are merged. Two topics in which at least one of the two is composed of a part of keywords in common with the other, show that the area of interest of the two topics should be the same. However, it was decided to reject this strategy because it is believed that a merging strategy of the overlapping communities (community detection), working on the graph, could be more efficient for this subtask.

Then, it was decided to exploit a method based on **clique percolation method**: the clique percolation is a method to find overlapping communities. It takes as input a parameter k, the size of cliques, and the network modeled as a graph on which the method is to be applied. The method procedure is as follows:

1. Find out all cliques of size k in a given network.
2. Construct a clique graph. Two cliques are adjacent if they share $k-1$ nodes.
3. Each connected components in the clique graph form a community.

In the specific case analyzed each community merged through several cliques represents a topic. In the graph given as input to the method each edge is a co-citation between two keywords, therefore a clique represents k keywords all co-cited together. This reinforces the probability that the k keywords taken into consideration can be part of the same topic. Merging cliques that share $k-1$ nodes is equally likely to form a set of keywords that are part of the same area of interest. It was decided, also in this case, to exploit the networkx implementation; the networkx method used based on the clique percolation method is `k_clique_communities`: the variant from the clique percolation method lies in the fact that every possible clique of different size starting by a fixed minimum size is found. in fact, it considers all the cliques at least k size, whose parameter is passed as input to the method, merging all the cliques that have $k-1$ nodes in common. The source code of this method: `https://networkx.github.io/documentation/latest/_modules/networkx/algorithms/community/kclique.html`.

3.3.1 Method results based on spread of influence models

Both the diffusion models of influence selected for all the proposed threshold calculation strategies were tested and, also based on the considerations made in the previous subsection, the spread of influence model and the optimal threshold calculation strategy are selected.
Also in this case the analysis was carried out for every possible k and with a fixed k it was analyzed every year of the timeline. All tests are shown on jupyter notebooks, divided into two files: one to analyze the `k_clique_communities` method on the Independent Cascade Model with all threshold calculation strategies and one to analyze the same method on the Tipping Model with all threshold calculation strategies. Both file are sited in the follow path: `src/overlaps_topics/analysis` and are called, respectively, `all_year_indipendent_cascade_analysis.ipynb` (`all_year_indipendent_cascade_analysis.html` for html version) and `all_year_tipping_model_analysis.ipynb` (`all_year_indipendent_cascade_analysis.html` for html version). Below is an example of a `k_clique_communities` output:

```
OVERLAP MERGING RESULT WITH CLIQUE PERCOLATION METHOD, CASCADE MODEL,
SEED TOP 5 NODES AND DEG THRESHOLD CALCULATION STRATEGY.
```

```
['support vector machine', 'feature vector']
len_topic: 2

['simulation', 'nonlinear system', 'motion estimation\xa0']
len_topic: 3
```

For both model the parameter k of the networkx algorithm is always equal to 2, so the communities (cliques) must have at least one common keyword to be merged. The choice of such a low k is due to the fact that the method merges groups of keywords where each group of keywords consists of at least k keywords all co-cited with each other, so it is very likely that even with a single co-citation in common between the groups, the merged keywords can be part of the same area of interest.
The obtained topics by this method must also meet certain conditions to be valid topics and, by extension, to consider the output of the `k_clique_communities` to be valid:

1. The number of merged topics, fixed a k by which the most significant k-nodes to generate k topics, must be at most k. If this condition is not met, the system will launch the exception: *TopicTooManyException*.
2. At least one of the k top-keywords must be in each generated topic since each keyword is representative for that topic. If this condition is not met, the system will launch the exception: *TopicMalFormedException*.
3. All the keywords retrieved by the adopted spread of influence model must be used to generate the topics, otherwise it means that the nodes influenced are not part of a cliques and therefore it is very likely that it is not inherent with the merged topic that is being generated. If this condition is not met, the system will launch the exception: *TopicNotAllKeywordsUsedException*.

The considerations on the application of `k_clique_communities` on the results of the **Indipendent Cascade Model** can be extended for every possible k between [5, 10, 20, 100]. The application produces inconsistent results: for each year of the timeline this method was tested on the threshold calculation strategies (1), (2) e (3) and all either could work or return an error depending on the execution and, by extension, depending on the nodes that are influenced in that particular execution.
There is no strategy for calculating the threshold applied to the model that brings a correct application of the Clique Percolation Method variant, taking also into consideration the reasoning made on the independent cascade model in the previous subsection is possible to deduce that in no way is applicable this spread of influence algorithm to identify the keywords that make up the topics.

The considerations on the application of `k_clique_communities` on the results of the **Tipping Model** can be extended for every possible k between [5, 10, 20, 100]. For each year of the timeline this method was tested on the threshold calculation strategies (1), (2) and (3) and it was noted that the strategies (2) and (3) depending on the k taken into consideration and the year either work or return *TopicMalFormedException* or *TopicNotAllKeywordsUsedException*. This means respectively that either some topics are formed that do not contain at least one of the top k keywords, i.e. having an invalid topic, or do not use all the keywords previously detected, i.e. having probably detected keywords not related to any topic. Regarding the threshold calculation strategy 3 (reasoning taken from the previous subsection), as mentioned above, some keywords are affected without having infected neighbors, which could favor the exception *TopicNotAllKeywordsUsedException*. While for (1) the method based on Clique Percolation Method works for each k and for each year of the timeline. As previously mentioned, it also maximizes the spread of influence and having graphs that are made up of all connected components to detect topic is ideal. Therefore it was decided to use the Tipping model with the threshold calculation strategy $\frac{1}{n}$.

The implementation of metrics modeling through the networkx library is present in the package `src/overlaps_topics` in the `overlap_topics.py` file. In this file, there is the OverlapTopics class whose methods perform the preprocessing necessary to obtain the subgraph of the dataset ds-1 relative to the nodes affected, apply the clique percolation method and check the conditions in the generated topics described above. Comments and explanations regarding these methods are present on the code.
In the same package, in the `overlap_exception.py` file, there are the implementations of the exceptions used to verify the three conditions. Again, comments and explanations regarding these methods are present on the code.

In the path `src/overlaps_topics/analysis` is present a python script called `sample_overlapmerge_` `analysis.py` to analyze the `k_clique_communities` method on graph for given year, given k, given spread of influence model and given threshold caluclation strategy. The explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python sample_overlapmerge_analysis.py -h
```

4 T2: Topic Tracing

This section describes the topic tracing task; it consists of tracking all the topics identified along the timeline. It describes if, during the tracking, topics of consecutive years can be merged by creating macro-topics identified on the timeline and finally creating the list of identified macro-topics.

4.1 Method and metric of tracing topics along the timeline

The first subtask of the T2 task is the tracing of the topics along the timeline. It can only be done after identifying all the topics of each year between 2000-2018. For tracing, two main decisions must be made: *tracing approach* and *tracing metric.*
A tracing approach consists in the management of the timeline analysis: how much to analyze it in depth, whether to consider all the traced topics to that point in a given chain to trace the next or consider the topics individually etc. A tracing metric is the metric used to decide whether a topic can trace another topic.

4.1.1 Tracing approach

Initially a **linear tracing strategy** was devised analyzing the entire timeline in linear time. This type of tracing scrolls the timeline and tries to trace the topics in the following way: try to trace topics of 2001 through topic of 2000; if it was able to trace using the tracing metric, try tracing topics from the year 2002 starting from the pair of topics 2000-2001. Otherwise topics of 2002 will be traced starting from a topic either of 2000 or 2001. And so on. By adopting this technique we are able to create topic chains from the year 2000 to 2018, but it is not possible to know exactly which topics of the following years is traced by the tracer topic. Since an argument of 2000 is different from the resulting topic from the pair 2000-2001, it is very likely that by adopting this technique chains of topics that could be recovered through a more detailed search of the temporal sequence would be lost. This strategy was discarded without having been implemented or tested.

As an alternative to the *linear tracing strategy*, a **exhaustive tracing strategy** was devised. This new strategy analyzes the topics along the timeline much more deeply than the previous one: for each topic of each year (tracer topic), based on a chosen tracing metric, all the topics of the years following the tracer topic (or topic target), if possible, are traced. For example, it is examined which topics in the years 2001-2018 can be traced individually from each topic of 2000; which topics of the years 2002-2018 can be traced individually from each topic of the year 2001 and so on until the year 2017. This strategy was consequently implemented and tested.

4.1.2 Tracing metric

For the tracing metric, it is necessary to work with the topic content: the keywords. While for the merging strategy based on overlapping, working on a fixed year, it was possible to focus on the graph and its structure, the comparison is now on topics of different years and consequently related to different graphs. This makes it impossible to use algorithms to manage merging communities (whether they are based or not based on overlapping).
The alternative is therefore to decide to trace a topic based on the fraction of words they have in common. Graphs of different years have different sizes and connected components of different dimensions, so the resulting topics have different dimensions. Consequently, it was necessary to adapt the calculation of the fraction of words in common taking into account the possible difference in size of the two topics: if one of the two has at least x words in common with the other then the topic is traced.

The x value is a threshold, the minimum fraction of keywords that one of the two topics must have in common with the other in order to carry out the tracing; the next consideration is therefore how to set the threshold. In different years, not only could there be difference in graphs in terms of size and structure, but also in terms of nodes, ie co-cited keywords. The keywords are the words used by articles in literature in a fixed year; it is possible that in years articles can be of different topics and areas. Therefore they can use (and co-use) only in part the same keywords, all keywords or none. It was necessary to find a balance so as not to have any overlapping topics with not too few or too many keywords in common: the threhsold has been set to 0.5. To recap: a topic can trace another topic if one of the two has at least half of the keywords in common with the other.

4.1.3 Tracing results

The results obtained through topic tracing was observed on each admitted k value. The following considerations are valid for each k:

1. Not all topics are able to trace over the following years, so they can't create a topic chain where the first topic is the tracer topic (target topic) and the following are the topics traced by it. In this case we would have a chain formed only by the target topic itself.
2. Not all topics are able to create a contiguous chain of topics from the year of the target topic to 2018. In some cases they are taken in a sparse way (sequence of topics of years not contiguous to each other) in others you create contiguous chains among some topics but unlinked from the target topics.
3. In some cases, instead, a chain is created by referring to the tracer topic but whose length does not extend to the end of the timeline. It could be interrupted earlier.

These considerations below are acceptable because the metric chosen to trace the topics is the fraction of words that one of the two topics have in common with the other. It is indeed possible to have topics that do not have keywords in common with any other topic identified along the timeline. It is also possible to have topics that do not have keywords in common with a topic of all subsequent years but only with topics of some years.
During some tracing tests it was noted that some topics tend to track more than one topic in the same year. In order to maintain a chain that contains a single topic traced in the same year, it was decided to trace only the first useful topic, i.e. the first topic encountered where he or the tracer topic have at least half of the keywords in common with the other.

The implementation of the tracing is present in the package `src/tracing_merge` in the `tracing_path.py` file. The file contains the class *TraceMergeTimeline* whose methods trace the topics on the timeline (for the years following the year of the tracer topic) for each previously identified topic. It is also possible to decide to print the tracing, write it on the log file or do neither of the two actions. In the latter case, the user does not see any tracing. Comments and explanations regarding these methods are present on the code.

All the tests performed on the tracking are reported in a jupyter notebook file present in the package `src/tracing_merge/analysis` in the `all_view_timeline_tracing.ipynb` file (`all_view_timeline_tracing.html` for html version). Below is an example of a timeline tracing output; specifically an example of a result with $k = 5$:

```
The target topic:

year: 2001
['laboratories', 'state estimation', 'signal processing algorithms',
 'computer simulation', 'linear systems', 'mathematics', 'deconvolution',
 'signal processing', 'stochastic processes', 'fading', 'filtering',
 'bayesian methods', 'smoothing methods', 'state-space methods',
 'particle filters', 'monte carlo methods', 'filtering algorithms']


The traced topics are:


year : 2002
['speech processing', 'bayesian methods', 'monte carlo methods',
 'signal processing']


******************** END THIS TOPIC TRACING **********************************
```

Furthermore, a python script weas created at the path `src/tracing_merge/analysis/` called `sample_timelinetracing_analysis.py` to be able to observe the tracing on the whole timeline for different values of k (considering the values of k allowed). The explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python sample_timelinetracing_analysis.py.py -h
```

4.2 Merging during the tracing

It is necessary to decide whether during the sub-task of tracking the topics along the timeline, two topics identified in two consecutive years can be merged together. The considerations made in the previous section are useful to be able to decide if and how to carry out the merge. The most important consideration to be able to decide is (2), that is the one that examines the nature of the chains created (therefore in the case in which they exist). It is not certain that topic chains have a topic for each year starting from the year of the tracer topic (target topic) until 2018. It is possible that the chains are linked in more possible configurations. Conform to the specifications provided, the merging was designed in the following way: all the topics of contiguous years are merged together starting from the tracer topic until the chain is interrupted; subsequent tracked topics are not taken into account for merging. If there are topics traced but there is no chain of contiguous years from the tracer topic the merge is not carried out.

This happens during the tracing phase, both because required by the specifications and for practical efficiency reasons, and therefore the file and the class that implements this mechanism is the same as the tracing: in the package `src/tracing_merge` in the `tracing_path.py` file. In this case, a jupyter notebook file or a python test script is not provided because the merging result can be seen creating the final macro-topic list.

4.3 Final list creation

As the last subtask there is the creation of the list of macro-topics identified on the timeline. Macro-topics are considered as follows:

- If the tracer topic does not trace any topic, the topic itself is considered as a macro-topic.
- If the tracer topic traces topic but the first topic traced is not related to the year following his, the tracer is considered a macro-topic.
- If the tracer topic traces topics and there are consecutive years starting from the tracer topic, all those that are part of this chain are merged until it is interrupted (the next topic is not the year following the previous topic in the chain) or ends. The new merged topic constitutes a macro-topic and is added instead of the tracer topic.

Whenever a macro-topic is identified, it must be added to the list. The problem you might run into is that the added macro-topic could contain a macro-topic already added or be contained in a macro-topic already present in the list. This problem is verified after identifying all the macro-topics: an exhaustive search is carried out on the whole list analyzing if each topic is contained in another topic, in case one of the two is included in the other, the topic of smaller size is eliminated. In this case, using reduced amount of data, the total traced topics are not many and an exhaustive search does not cause particular problems.

This happens at the end of the tracing phase and therefore the file and the class that implements this mechanism is the same as the tracing: in the package `src/tracing_merge` in the `tracing_path.py` file.

With this last point the whole developed system is completed, so the script that runs the whole system is exactly the `main.py` in the main path `src`. The script allows you to analyze all the macro-topics retrieved along the timeline for different values of k, with fixed k. The explanation of the script, along with the options necessary for use, have been reported in the same and using the help `-h` option:

```
python main.py -h
```

There is no jupyter notebook file for the main since all the decisions necessary to develop the system have been taken. but there are still examples of logs for $k = 5$ and $k = 10$ in `output-sample/main_results`

5 Conclusion

To summarize what has been done:

1. The ds-1 and ds-2 datasets were analyzed, proposing two weighting calculation strategies for ds-1.
2. Pagerank metrics were analyzed (analyzing both weights calculation strategies for this metric), degree and betweenness centrality observing that Pagerank with the weights calculation strategy consisting of the fraction of authors who co-cited the two keywords and who cooperated at least once together is the best metric.
3. The Independent Cascade Model and the Tipping Model were analyzed, proposing for both three threshold calculation strategies.
4. The Clique Percolation Method variant was chosen to obtain the merged topics, then the actual one-year topics. Both spread of influence models were tested, each for all three proposed thresholds. The model (Tipping model) and strategy for calculating the best threshold (1) was therefore chosen.
5. All topics for each year of the timeline have been identified and saved in a dictionary.
6. All the timeline topics were then traced. For each topic of each year we analyze which topics of the following years can trace by creating a topic chain where the first topic is the tracer topic and the others are the topics tracked by the tracer.
7. During the tracing, it was necessary to decide if and how to merge the topics of consecutive years. Taking into consideration the chain created, all those topics of consecutive years starting from the tracer topic (target topic).
8. Finally, the final list of macro-topics was created. If a topic has not traced any topic, the topic has been considered a macro-topic, otherwise the new merged topic is considered through topics of consecutive years in the chain (starting from the tracer topic). Finally an exhaustive search was carried out by eliminating identified macro-topics totally contained in other macro-topics.

We therefore have the final list of macro-topics identified along the timeline through the steps and choices listed above. Trying with increasingly large admitted values of k , the number of macro-topics increases.

The macro-topics are of different sizes and roughly several keywords. It is possible that one of the two topics also has most of the keywords in common with another topic, however there will be no topic completely included in another. This guarantees the diversity between two topics. Obviously many keywords are included in more topics, but thanks to the methodologies and assumptions made, it is possible to say that the topics are fairly homogeneous.

The Clique Percolation Method variant always provides the same topics but often in different order (in the implementation each year has a list of topics, so the order counts). Since it was decided to take the first traceable topic, due to the considerations made previously, it is possible that the traced macro topics are not always the same. For example. after a series of intensive tests it was noticed that, for $k = 5$ and $k = 10$, there is a variation of 1-2 topics in the possible number of topics identified and there is a small variation in the composition of some topics. In `output_sample/main_output` some log files have been reported that show the observations made.

References

[1] Python language. It is avaliable: `https://www.python.org/`

[2] Pandas python library. It is avaliable: `https://pandas.pydata.org/`

[3] Networkx python library. It is avaliable: `https://networkx.github.io/documentation/stable/`

[4] Matplotlib python library. It is avaliable: `https://matplotlib.org/`

[5] Ndlib python library. It is avaliable: `https://ndlib.readthedocs.io/en/latest/`