Sapienza University of Rome

Mining Evolving Topics

Web and Social Information Extraction Assignment

Author:

Giorgio Mariani

Contents

1 Introduction

This document is a report of the **Web and Social Information Extraction Course** project. The project consists of two correlated tasks that need to be solved:

Task 1 requires the extraction of topics for each year between 2000 and 2018, included. These topics should be extracted by processing information contained in several keyword co-occurrence graphs (one per year). Diffusion models have to be utilized in order to determine the nodes inside a topic.

Task 2 asks for the tracking and merging of the previously gathered topics throughout the years between 2000 and 2018. The results of this task is a set of time-fused topics.

1.1 Technologies

Python Interpreter version 3.7. The Language used in the project is **Python** version 3.7, this language was chosen mainly due to the availability of specific computation libraries such as **NumPy** and **NetworkX**. The flexibility of the language also allows for very fast prototyping, which can help in an project like the one proposed, since different approaches were studied during the project progression.

NumPy Module. It is one of the fundamental packages for scientific computing with Python. It contains among other things: a powerful N-dimensional array object, tools for integrating C/C++ code, linear algebra functions, Fourier transform, and random number capabilities.

NetworkX Module. This module is a Python library designed for the creation, manipulation, and analysis of different types of networks. It offers a variety of already implemented graph algorithms, such as PageRank, Betweenness Centrality, etc. It is [..]

igraph-python Module. **igraph** was also used as graph processing library during the early phase of the project, however due to the lack of certain algorithms and the overall less intuitive interface, this package was abandoned for the simpler **NetworkX**. It should be said, although, that **igraph** achieved better performance in the various tasks.

1.2 Data

The datasets used in the project are two: one containing information about keywords co-occurrence, and one containing co-authorship data. Both datasets are domain-specific and contain entries associated only to machine-learning related articles and papers.

Dataset DS-1. As mentioned, the first dataset (denoted as **DS-1**) contains information about keyword co-occurrence throughout some intervals of years, including between 2000 and 2018. Formally, it contains rows with the following structure:

$$\mathbf{y} \text{ <tab> } \mathbf{k_i} \text{ <tab> } \mathbf{k_j} \text{ <tab> } [\mathbf{a_0}{:}n_0 \text{ ,..., } \mathbf{a_m}{:}n_m]$$

with $\mathbf{k_i}$ and $\mathbf{k_j}$ two co-occurring keywords, $\mathbf{y}$ the year in which they co-occur, while n_k is the number of articles published by author $\mathbf{a_k}$ containing both $\mathbf{k_i}$ and $\mathbf{k_j}$. Therefore, this dataset implicitly defines a number of graphs, one per year, describing co-occurrence relationships.

Dataset DS-2. This dataset stores information about co-authorship between authors during several years, including the time-interval from 2000 to 2018. Rows in the dataset are defined as

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

with $\mathbf{y}$ the co-authorship year, $\mathbf{a_i}$ and $\mathbf{a_j}$ the collaborating authors, and $\mathbf{n}$ the number of collaborations between $\mathbf{a_i}$ and $\mathbf{a_j}$ within the year.
Again, this dataset can be seen as a sequence of graphs (one per year), having as nodes authors and as edge information about the co-authorship relation.

2 Diffusion Models

2.1 Linear Threshold Model

The *linear threshold model* [1] is a diffusion model, which are a class of models sometimes adopted in order to represent the rate of diffusion of an information in a given network.
Differently from simpler models like the *tipping model* and the *SIR model*, the linear threshold makes use of information between edges, making it more compelling for this project; Specifically, in the linear threshold model, each directed edge $(u, v) \in E$ has associated a non-negative weight $w_{u,v}$, with the additional constraint that for any node $v \in V$, the total in-neighbor sum[1] must be less than or equal to one

$$\sum_{u \in \eta^{in}(v)} w_{u,v} \leq 1 \tag{1}$$

in order to be consistent.

Given a *Social Network*, represented by the graph $G = (\mathsf{V}, \mathsf{E})$ (with $|\mathsf{V}| = n$ and $|\mathsf{E}| = m$), the linear threshold model computes a set of nodes that might[2] be influenced by an initial set of source nodes (referred to as $A_\theta^{(0)}$). The process works by first assigning (uniformly at random) to each vertex v a threshold value $\theta_v \in (0, 1]$, then it proceeds at iterations, where in iteration i, it expands the set of currently active nodes (indicated by $A_\theta^{(i)}$) until convergence is reached. The active nodes at iteration i are update through the following recurrent equation:

$$A_\theta^{(i+1)} = A_\theta^{(i)} \cup \left\{ v \in \mathsf{V} : \sum_{u \in \eta^{in}(v) \cap A_\theta^{(i)}} w_{u,v} \geq \theta_v \right\}$$

The set of active nodes $A_\theta^{(i)}$ is said to have reached convergence if and only if $A_\theta^{(i)} = A_\theta^{(i+1)}$.

2.2 Average Linear Threshold

As previously stated, the linear threshold model is a randomized process with the output varying depending on the threshold assignments; in order to reduce the output sensitivity, instead of giving a boolean output for each node v (i.e. the node was/wasn't influenced), it is possible to execute the algorithm a number of times[3] (n) and using the average infection rate $\alpha_v \in [0, 1]$ as output:

$$\alpha_v = \frac{1}{n} \sum_{i=0}^{n} b_v^{(i)} \tag{2}$$

with $b_v^{(i)} \in \{0, 1\}$ indicating the result of linear threshold during trial i.

Probabilistic interpretation. This value can be seen as an estimator for the probability of infection of the node v, if the set of nodes S is the infection source:

$$P\{v \text{ is infected} \mid S \text{ as source}\} = \alpha_{v,S}$$

[1] Throughout this document, we refer to the *in-neighbor sum* of a node as the sum of all its incoming edges weights.

[2] Since the algorithm is randomized, this will change depending on the execution.

[3] For this project, the algorithm is invoked 12 times in order to compute the relative frequencies.

Fuzzy Set interpretation. It is possible to interpret the value α as how much the node v is contained in the set of nodes infected by the source set S. So if a fuzzy set [2] is used to describe the nodes infected by S, then $\alpha_{v,S}$ can represent its membership function:

Definition 1 *A Fuzzy Set $\widetilde{F}$ in X is a set of ordered pairs:*

$$\widetilde{F} = \{(x, \mu_{\widetilde{F}}(x)) : x \in X\}$$

The function $\mu_{\widetilde{F}} : X \to [0, 1]$ is called membership function, and describes how much a given element x belongs to the set $\widetilde{F}$.

Using this definition, it is possible to define the set of nodes influenced by the source S as the fuzzy set:

$$\widetilde{S} = \left\{(v, \mu_{\widetilde{S}}(v)) : v \in V, \mu_{\widetilde{S}}(v) = \alpha_{v,S}\right\}$$

Algorithm 1 Linear Threshold Model

Require: graph G, with nodes $V(G)$.
Require: source node $s \in V(G)$, in which the algorithm starts diffusing influence.

procedure linear_threshold_model(G, s, θ)
    visited $\leftarrow$ array with size $|V(G)|$ containing all zeros entries.
    visited[s] $\leftarrow 1$
    $Q \leftarrow$ new queue containing only s.
    while Q is not empty **do**
        $v \leftarrow$ dequeue(Q)
        for $u \in \eta^{out}(v)$ **do**
            if visited[u] $< \theta_u$ **then**
                visited[u] $\leftarrow$ visited[u] $+ w_{v,u}$
                if visited[u] $\geq \theta_u$ **then**
                    enqueue(Q, u)
    result $\leftarrow$ array with size $|V(G)|$ containing all zeros entries.
    for $u \in V(G)$ **do**
        $$\text{result}[u] \leftarrow \begin{cases} 1 & \text{if visited}[u] \geq \theta_u \\ 0 & \text{otherwise} \end{cases}$$
    return result

3 Task 1

3.1 Edge Weight Selection

The edge-weights were selected using information from both datasets **DS-1** and **DS-2**; after selection these weights are then normalized in order to better be exploited by the linear threshold model. The formula used to compute these edge weights, prior the normalization step, is quite intuitive and can be described as the sum of the number of articles containing the both keywords k_1 and k_2, weighted by the rank of the articles' authors. The rank is obtained by computing the *pagerank* of the author graph of the respective year.
The formulation of these pre-normalization edge weights (for example, between keywords k_1 and k_2) is given by

$$w_{k_1,k_2} = \sum_{a_i} \mathsf{rank}(a_i) \cdot \mathsf{count}_{k_1,k_2}(a_i) \tag{3}$$

where a_i varies over the set of all authors, $rank(a)$ is the pagerank of author a, and $count_{k_1,k_2}(a)$ is the number of articles having as author a and containing both k_1 and k_2. The pagerank of an author (for a given year y) is obtained by computing the pagerank of co-authorship graph for the respective year.

Outlier Correction. One problem that was encountered during the first phase of the project was the presence of outlier edge-weights. Indeed, the weights computed using the eq. (3) can be heavily unbalanced (as can be seen in fig. 1), which caused a number of problems when trying to apply the linear threshold algorithm. This problem was approached by using a simple outlier correction strategy; using the *z-score* of the edge-weight distribution, clamping elements with a *z-score* higher than three.

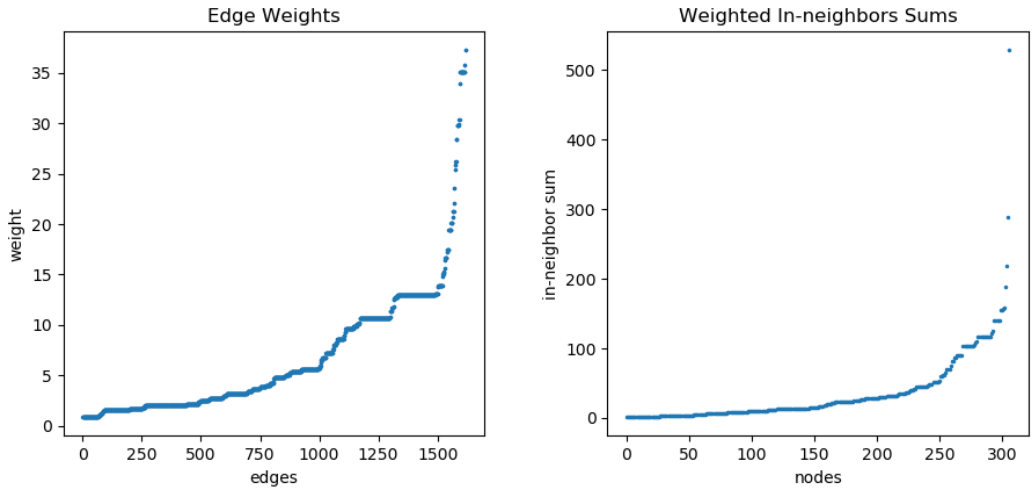

Figure 1: Graphs showing the distribution of the edge-weights and of the in-neighbors sums **before** the normalization step.

Normalization. As previously mentioned, in order to use these calculated edge-weights with the linear threshold algorithm (see section 2.1), a normalization step is necessary. Indeed, remember

that the in-neighbors sums must be less then or equal to one for each node in the input graph, which may not be true for the weights computed using eq. (3). Another problem is that if the in-coming weights of a node are too small then they will not be able infect it, hence blocking the diffusion process.
A number of possible strategies can be used in order to avoid these situation, in particular, for this project three types were considered:

- The simplest normalization strategy is to scale all the weights $w_{u,v}$ such that eq. (1) must be valid for all nodes u. This can be achieved by dividing all the weights by the maximum in-neighbors sum. That is,

$$\hat{w}_{v,u} = \frac{w_{v,u}}{\max_{y} \left\{ \sum_{x \in \eta^{in}(y)} w_{x,y} \right\}}$$

  This normalization, however, caused the edges' weights to be far too unbalanced, favouring nodes with a high number of in-neighbours, and virtually never activating nodes with fewer incoming edges.

- Another strategy initially adopted consisted in forcing the sum of all the in-coming neighbors of all nodes to be 1. This can be accomplished by locally scaling each weight with the formula

$$\hat{w}_{v,u} = \frac{w_{v,u}}{\sum_{v' \in \eta^{in}(u)} w_{v',u}}$$

  The major drawback of this approach is that nodes with a single incoming vertex are always activated with their incoming neighbor. More generally, nodes with a low number of in-coming neighbours are favoured while using this normalization approach.

- The final normalization strategy, which is the one actually adopted in this project, consists in employing a logarithmic balancing of the in-neighbors summations.

$$\hat{w}_{v,u} = \frac{w_{v,u}}{\log\left(1 + \sum_{x \in \eta^{in}(u)} w_{x,u}\right)} \cdot z$$

  with z a normalization factor such that

$$\max_{u} \left\{ \sum_{v \in \eta^{in}\, u} \hat{w}_{v,u} \right\} = 1$$

  The use of the logarithm allows the in-neighbors sums to be well distributed, so that nodes with less nodes can be activated. This method proved to create topics that are somewhat balanced. It is possible to see the difference between before and after this normalization process respectively in fig. 1 and fig. 2.

3.2 Topicness

Following the project's specification, topics are estimated by applying the linear threshold model over some source nodes, which have to be extracted using a metric to define, i.e. pagerank, cluster-coefficient, etc.. During development, different such metrics were tested and refined. At first, the pagerank was utilized in order to pick the best candidates. However, two major

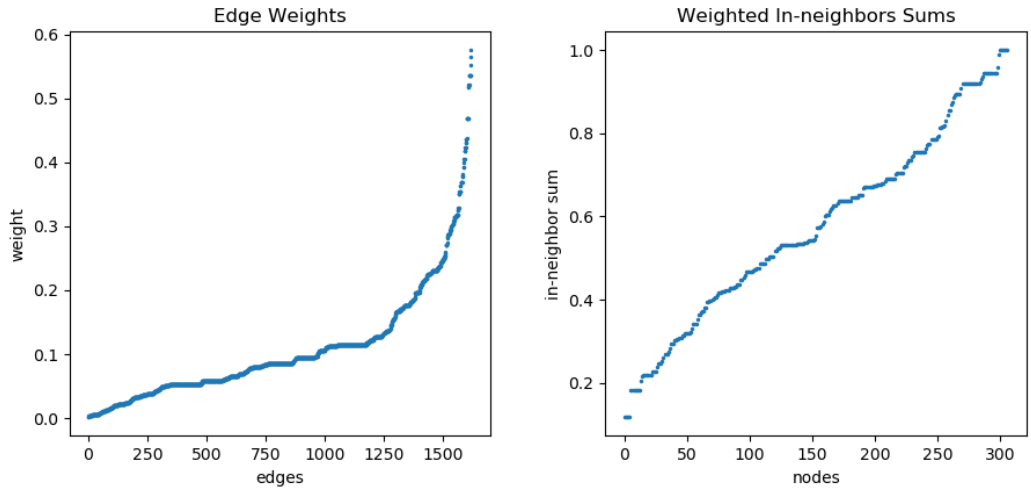

Figure 2: Graphs showing the distribution of the edge-weights and of the in-neighbors sums **after** the normalization step.

problems resulted from using pagerank: a strong bias towards hubs (which are not necessarily good topic sources), and a problem with the "picking" strategy, since simply taking the nodes with highest value will take nodes most likely near to the core of the graph's main component. To solve the first problem a combination of metrics were utilized, specifically, two others besides pagerank: *Local Cluster Coefficient*, and *Betweenness Centrality*. The local cluster coefficient of a node indicates how much the neighbors of a node are interconnected with each-others; it is intuitive that good topics sources have an higher cluster coefficient than normal nodes. For un-directed un-weighted graphs, the cluster coefficient of u is described by

$$\text{cluster coefficient}(u) = \frac{2\left|\left\{\{v,w\} \in \mathsf{E} : v \in \eta(u), w \in \eta(u)\right\}\right|}{d_u \cdot (d_u - 1)}$$

However, the directed weighted variant described in [3] (implemented by **NetworkX**) was used for this project.

It was possible to improve the initial pagerank estimation by adding the information found in the cluster coefficient. This was done by using, instead of standard pagerank, the *personalized-pagerank* algorithm, with personalization vector the cluster-coefficient (after being appropriately scaled to a probability distribution). The difference, using as dumping factor 0.8, can be seen in fig. 3. Despite improving on the initial function, the personalized-pagerank remained yet too inclined towards really big hubs; in order to reduce this bias a new metric was introduced, the aforementioned betweenness centrality. The personalized pagerank values are scaled down proportionally to their betweenness score, which was found to be impacting only nodes that represents big "hubs" in the graph. The resulting metric, denoted as *topicness*, is hence given by the equation:

$$\text{topicness}(u) = \text{personalized-pagerank}(u) \cdot \Big(1 - \text{betweenness}(u)\Big) \tag{4}$$

for each $u \in \mathsf{V}$.

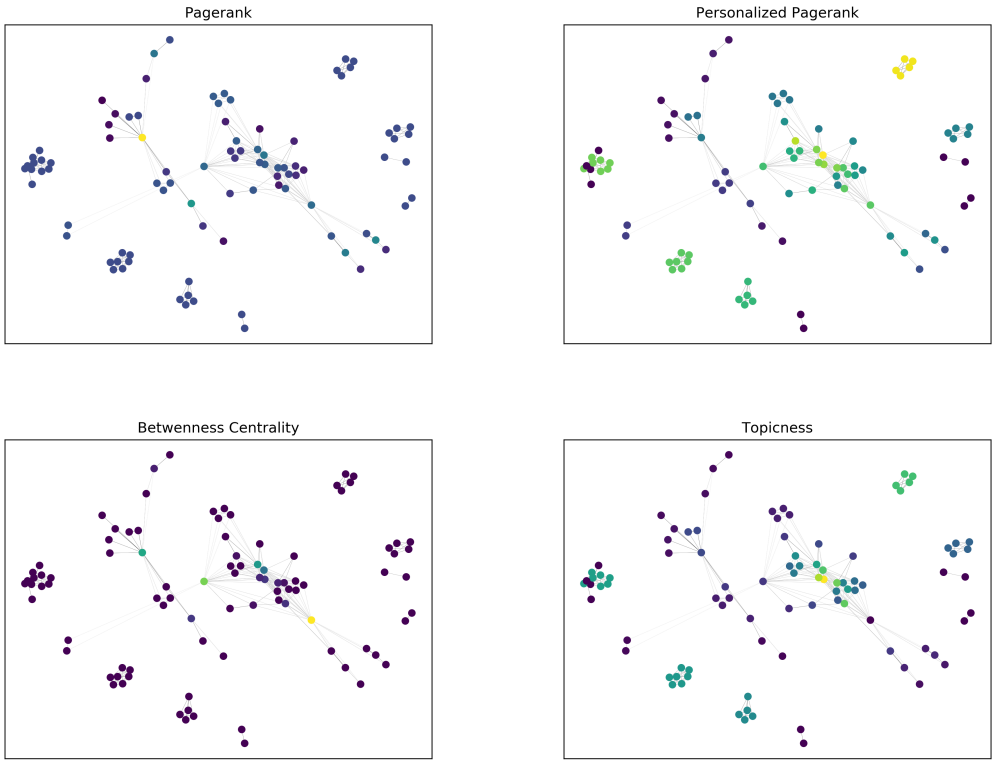

Figure 3: Comparison between standard Pagerank, Personalized Pagerank (using local Cluster-Coefficient), Betwenness Centrality, and Topicness.

3.3 Topic Extraction

Topics are extracted using an iterative method: at each iteration the node associated to the global maximum value[4] of the topicness function is extracted; after that, the influence of said node is used to update the topicness "surface", in order to obtain a different maximum in the next iteration (see algorithm 2).

[4]If there is more then one maximum, then the node is arbitrarily chosen between those.

Algorithm 2 Topic Extraction

Require: graph, keyword co-occurrence graph for a specific year, with n vertices.
Require: `topicness`, array of size n, indicating topicness for each node.

```
1: procedure topic_extraction(graph, topicness)
2:     S ← ∅
3:     while ∃x : topicness[x] > 0 do
4:         maxnode ← argmax_v{topicness[v]}
5:         topic ← node_influence(graph, maxnode)          ▷ topic is an indicator array.
6:         for u = 1 ... n do
7:             topicness[u] ← topicness[u] · (1 − topic[u])
8:         S ← S ∪ {maxnode}
9:     return S
```

Node influence. The `node_influence` is computed using the average linear threshold algorithm, as described in section 2.2; hence the variable topic is a vector of size n with values between 0 and 1, indicating how much a certain node belongs to the topic identified by the chosen source node. During the project development two methodologies were tested for choosing the value of the topic vector:

Fuzzy: At first, the topics were considered as fuzzy topics, using the output of the average linear threshold as description of the topic. However, this caused some problems during the tracking phase, as topics tended to be too unbalanced towards the source (whose membership value is always one).

Crisp: After the problems induced by the fuzzy sets during the second task, it was decided to use standard (i.e. crispy) sets in order to describe topics. Of course, this requires a *defuzzification* procedure, which may loose information.

3.4 Defuzzification of topics

Fuzzy set theory uses several strategies in order to converted fuzzy sets to standard sets. These may range from mean based techniques, to more sophisticated algorithms. In our case, the algorithm is fairly simple; the $k = 6$ nodes with higher membership values (only if they are different from zero) are considered as the crisp representation of the topic. More formally, if x is in the defuzzification S of $\widetilde{F}$, then

$$x \in S \implies \begin{cases} \forall y \notin S : \mu_{\widetilde{F}}(x) \geq \mu_{\widetilde{F}}(y) \\ \mu_{\widetilde{F}}(x) > 0 \end{cases}$$

and the size of S must be less then or equal to k (i.e. $|S| \leq k$). This is done in order to prevent the creation of topics that are either too big or too small, and six was experimentally a good number for topic size. The conversion procedure can be seen in algorithm 3.

Algorithm 3 Defuzzification

Require: F, sequence of length n and containing pairs $(x, \mu_{\widetilde{F}}(x))$. This object represents the fuzzy set $\widetilde{F}$

```
procedure defuzzification(F, k)
    S ← ∅
    Sort array F w.r.t. membership values.
    for i = n − k ... n do
        (x, μ_F̃(x)) ← F[i]
        if μ_F̃(x) > 0 then S ← S ∪ {x}
    return S
```

4 Task2

During the second task, the topics extracted in **Task 1** are tracked throughout the years and fused together. The set of years to be merged is indicated by the sequence $\mathcal{Y} = \{y_i\}_{i=0..18}$. Two approaches were tested for solving this task:

- In the first method, topics between adjacent years are linked together, creating then a *direct acyclic graph* of the topics. From this graph are extracted $K = 20$ paths, representing the topic evolution. These are then merged in order to represent the final fused topics.
- The second approach makes use of more information, and uses the similarity of each pair of topics between all the years in $\mathcal{Y}$. Using this similarity (after being appropriately weighted) chains of topics are computed using a greedy approach. Similarly to the previous approach, these chains are then merged together in order to create the fused topics.

4.1 Keyword Similarity

In order to determine if two topics are related, a metric describing the similarity between two keywords is necessary. This may be useful when trying assess the similarity between topics, especially if said topics do not share a great number of words, but have a lot of similar words in common. The keyword similarities considered for the project are two:

Exact Match: this is a rather simple similarity function, mapping to 1 only pairs of keywords represented by the same exact character sequence. This is a very conservative metric, with high precision but low recall.

$$\text{word-score}(w_1, w_2) = \begin{cases} 1 & \text{if } w_1 = w_2 \\ 0 & \text{otherwise} \end{cases}$$

Jaccard Similarity: the similarity between two keywords is computed using their *bag-of-words* representation; each keyword is seen as a sequence of smaller words (using white-spaces as delimiter), which are then compared utilizing the *Jaccard Similarity*.
For example consider the keywords: **"brown fox"**, and **"quick fox"**, the estimated similarity score is given by

$$\text{word-score}(\textbf{"brown fox"}, \textbf{"quick fox"}) = \frac{|\{\textbf{"fox"}\}|}{|\{\textbf{"brown"}, \textbf{"quick"}, \textbf{"fox"}\}|} = \frac{1}{3}$$

This similarity suffers the opposite problem, since it may give an higher similarity value then it should, particularly if frequent words are in both keywords.

4.2 Topic-Pairs Score

The keyword similarity information described in the previous section is then utilized in order to associate topics from different years. This is performed using a score function, and later on using said score to create a relation between nodes in different years. During the project development, several score-functions were tested, the main one were:

Jaccard Similarity: the jaccard similarity was also used for topics scoring, using the keyword sets associated to each topic. The jaccard score seems to be a simple but effective score, however, it suffers one major drawback: it is not possible to exploit the jaccard keyword similarity previously described in section 4.1.

$$\text{topic-score}(T_i, T_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}$$

Fuzzy Similarity this similarity was an attempt at a generalization of the jaccard similarity, while adapting fuzzy topics and exploiting the keyword similarities described before. Let $\widetilde{T_1}$ and $\widetilde{T_2}$ be two fuzzy topics (with respective membership functions $\mu_{\widetilde{T}_i}$ and $\mu_{\widetilde{T}_j}$)

$$\mathbb{I}(\widetilde{T}_i, \widetilde{T}_j) = \mu_{\widetilde{T}_i}(w_1) \cdot \mu_{\widetilde{T}_j}(w_2) \cdot \text{word-score}(w_1, w_2)$$

$$\mathbb{U}(\widetilde{T}_i, \widetilde{T}_j) = \sum_{w_1, w_2} \mu_{\widetilde{T}_i}(w_1) \cdot \text{word-score}(w_1, w_2) + \sum_{w_1, w_2} \mu_{\widetilde{T}_j}(w_2) \cdot \text{word-score}(w_1, w_2) - \mathbb{I}(\widetilde{T}_i, \widetilde{T}_j)$$

$$\text{topic-score}(\widetilde{T}_i, \widetilde{T}_j) = \frac{\mathbb{I}(\widetilde{T}_i, \widetilde{T}_j)}{\mathbb{U}(\widetilde{T}_i, \widetilde{T}_j)}$$

This approach however yielded poor results, hence was quickly discarded.

Chamfer Similarity: this similarity function is inspired by the *chamfer distance* described in [4]. The score of a topic pair is given by the average keyword similarity of each node in the topic when mapped to the most similar word in the opposing topic. The averages from both topics are then summed, and the resulting value is the final score.

$$\text{topic-score}(T_i, T_j) = \frac{1}{|T_i|} \sum_{w_1 \in T_i} \max_{w_2 \in T_j} \text{word-score}(w_1, w_2) + \frac{1}{|T_j|} \sum_{w_2 \in T_j} \max_{w_1 \in T_i} \text{word-score}(w_1, w_2)$$

A problem with this score function is that it tends to favor smaller topics (especially with one or two nodes), since their score usually vary more than bigger sized topics. This could be addressed by using the square root of the topic cardinality instead of the cardinality itself (however, this approach was not tested).

4.3 Time-Fused Topics (First Attempt)

In this section is described the first attempt at creating time-fused topics. Unfortunately, the results yielded poor accuracy and were heavily unbalanced, creating few topics with a great number of keywords. Nevertheless, the approach used is worth mentioning.

Topic direct acyclic graph. Using one of the previously described topic-pairs score functions, a *Direct Acyclic Graph* (DAG) of topics was created. This graph encoded information about topic evolution throughout the years; it did so by containing several levels of nodes, where each level contained all the topics of a given year, see **??** for a better understanding.
More formally, let G_{DAG} be the topic DAG, then the nodes of this graph are the topics extracted[5] in the previous task; if $\mathcal{T}_y$ is the set containing all the topics estimated for year y, then the set of vertices V_{DAG} of G_{DAG} is defined as

$$\mathsf{V}_{\text{DAG}} = \bigcup_{k=0}^{18} \mathcal{T}_{y_k}$$

on the other hand, the set of edges is computed using the function topic-score; specifically, for a given year y_k, an edge $(T_i, T_j) \in \mathcal{T}_{y_k} \times \mathcal{T}_{y_{k+1}}$ is added to G_{DAG} if:

- T_j is the topic in $\mathcal{T}_{y_{k+1}}$ that maximizes $\mathsf{topic\text{-}score}(T_i, T_j)$,
- or T_i is the topic in $\mathcal{T}_{y_k}$ that maximizes $\mathsf{topic\text{-}score}(T_i, T_j)$.

Hence the set of edges in G_{DAG} (denoted as E_{DAG}) is given by:

$$\mathsf{E}_{\text{DAG}} = \bigcup_{k=0}^{18-1} \left(\overleftarrow{\mathsf{E}}_{y_k} \cup \overrightarrow{\mathsf{E}}_{y_k} \right)$$

with the sets $\overleftarrow{\mathsf{E}}_y$ and $\overrightarrow{\mathsf{E}}_y$ defined in the following manner:

$$\overleftarrow{\mathsf{E}}_{y_k} = \left\{ (T_i, T_j) \in \mathcal{T}_{y_k} \times \mathcal{T}_{y_{k+1}} : T_j = \underset{\dot{T} \in \mathcal{T}_{y_{k+1}}}{\text{argmax}} \{ \mathsf{topic\text{-}score}(T_i, \dot{T}) \} \right\}$$

$$\overrightarrow{\mathsf{E}}_{y_k} = \left\{ (T_i, T_j) \in \mathcal{T}_{y_k} \times \mathcal{T}_{y_{k+1}} : T_i = \underset{\dot{T} \in \mathcal{T}_{y_k}}{\text{argmax}} \{ \mathsf{topic\text{-}score}(\dot{T}, T_j) \} \right\}$$

Topic-chains selection. In order to generate the time-fused topics an intermediate step is necessary; a set of $K = 20$ topic functions mapping years to topics are computed.

Definition 2 *Let $\rho : D \to C$ be a function such that $D \subseteq \mathcal{Y}$ and $C \subseteq \bigcup_{k=0}^{18} \mathcal{T}_k$, then ρ is a* ***topic-chain*** *if and only if:*

- *all years in D are adjacent, i.e.*

$$\forall y_k \in D,\ y_k \neq \max_{\dot{y} \in D} \{\dot{y}\} \implies y_{k+1} \in Y$$

- *All years i D are mapped to topics in their respective years:*

$$\forall y_k \in D,\ \rho(y_k) \in \mathcal{T}_{y_k}$$

Intuitively, these chains should represent the evolution of a topic for a time interval $Y \in \mathcal{Y}$.

The topic-chain selection procedure (see algorithm 4) uses an iterative approach in order to extract a set of topic-chains; At each iteration, the longest path on the topic graph is computed[6]. Then, all the topics in the longest path are removed from the DAG, and the path is add to an output set. The procedure stops after K iterations, and the K path obtained in the process are the output topic-chains.

[5] From the year 2000 to 2018 (included).

[6] This is possible due to the graph being a DAG; the longest path can be found by using shortest path on the negated weights.

Algorithm 4 Chains Estimation (First Attempt)

Require: g, DAG containing the topics for each year.

```
1: procedure chain_extraction(g, K)
2:     P ← ∅
3:     for 1 ... K do
4:         ρ ← longest_path(g)
5:         for u ∈ ρ do
6:             remove_node(g, u)
7:         P ← P ∪ {ρ}
8:     return P
```

Time-Fusion. After computing the chains, the time-fused topics are obtained by computing the union of all the topics in the chain:

$$\text{fused-topic}(\rho) = \bigcup_{T \in \text{Range}(\rho)} T$$

The result is then K fused topics (one per chain). As previously introduced, the results obtained by using this procedure were seemed to be lacking precision and unbalanced. The main drawback of this approach could be that when computing the topic-chains, information only from directly adjacent years is used, which may cause the path to quickly diverge and not maintain a temporally strong consistency.

4.4 Time-Fused Topics (Second Attempt)

Due to the problem associated to the first algorithm, it was necessary to design a new procedure able to take into consideration more than directly adjacent topics, but also topics present in previous years. To do so, a different greedy method was designed; this approach makes use of the topic-score function defined in section 4.2, without creating a DAG. The pseudo-code of this procedure is showed in algorithm 5.

For each topics in all the years, the algorithm creates a candidate topic-chain (see next paragraph), representing the evolution of said topic throughout time. Afterward, K of these topic-chains are selected, using a policy that takes into consideration an overall score of the chain, and the topics inside the already selected chains.

Topic-chain creation. The chain creation algorithm is rather simple, it starts with a given source topic ρ_{y_h} with respective year y_s, then for each year y_i such that $i > s$, it extracts the topic in $\mathcal{T}_{y_i}$ that maximizes a certain heuristics, and adds it to the chain (in position ρ_{y_i}).
The heuristic used to decide which topic in $\mathcal{T}_{y_{i+1}}$ should be added to the chain ρ, is the *Exponential Moving Average* of the topic-score between $T \in \mathcal{T}_{y_{i+1}}$ and all the topics in ρ:

$$\text{EMA}_{\rho_{y_i}}(T) = \begin{cases} \alpha\, \text{topic-score}(\rho_{y_i}, T) + (1-\alpha) \cdot \text{EMA}_{\rho_{y_{i-1}}}(T) & \text{if } i > s \\ \alpha\, \text{topic-score}(\rho_{y_i}, T) & \text{if } i = s \end{cases}$$

with $\alpha = \frac{1}{2}$. This average value is used in order to find what are considered "good topic-chains". The goal here is to create chains that are temporally consistent, not only with the directly previous topic, but also with all antecedent elements in the chain (however, weighted exponentially).

The algorithm used to create these chains is then a greedy one (see `candidate_chain` in algorithm 5): given a topic ρ_{y_s} representing the topic chain start, the procedure iteratively (for each subsequent year) adds the topic T^* that maximizes the moving average, hence:

$$\rho_{y_{s+i}} = \underset{T^* \in \mathcal{T}_{y_{s+i-1}}}{\text{argmax}} \left\{ \text{EMA}_{\rho_{y_{s+i-1}}}(T^*) \right\}$$

with $i \geq 1$ the current iteration in the algorithm. As an additional constraint, the topic T^* must also have a non-zero score with its predecessor, in order to reduce computation intensity and enforce a chain that is more temporally consistent. The final formulation is then

$$\rho_{y_{s+i}} = \underset{T^* \in \mathcal{T}_{y_{s+i-1}}}{\text{argmax}} \left\{ \text{EMA}_{\rho_{y_{s+i-1}}}(T^*) \right\}$$
$$\text{subject to } \text{topic-score}(T^*, \rho_{y_{i-1}}) > 0$$

While chains constructed in this manner may not be maximal, the use of this greedy approach ensured affordable computation.

Topic-chain selection. The chain creation algorithm is invoked for each topic in all the years in $\mathcal{Y}$, resulting in a number of chains equal to:

$$\left| \bigcup_{y \in \mathcal{Y}} \mathcal{T}_y \right|$$

In order to select the best $K = 20$ chains from these, another iterative procedure is employed: all the chains are ranked through a scoring function, then the maximum chain is selected, finally the rank is updated using information from the selected chain. The scoring function is the following:

$$\text{chain-score}(\rho) = \sum_{y_i \in \text{Domain}(\rho)} \text{EMA}_{\rho_{y_{i-1}}}(\rho_{y_i})$$

Initially, all chains use as rank their chain-score, however, after each iteration (in which a topic-chain ρ^* is selected) their rank is scaled down using their similarity with respect to ρ^*; this is done in order to avoid selecting similar chains. The similarity is measured using the set of all the keywords in the chain,

$$\text{keywords}(\rho) = \{w : \exists T \in \text{Range}(\rho) \text{ s.t. } w \in T\} = \bigcup_{T \in \text{Range}(\rho)} T$$

Specifically, given two topic-chains ρ and ρ^*, their similarity is the jaccard similarity between keywords(ρ) and keywords(ρ^*). To see more in detail how the algorithm works, see algorithm 5.

Algorithm 5 Chains Estimation (Second Attempt)

procedure chain_extraction(K)
  $rank \leftarrow$ array with length $|\bigcup_{y\in\mathcal{Y}} \mathcal{T}_y|$, filled with 0.
  $P \leftarrow \emptyset$
  $P^* \leftarrow \emptyset$
  for $T \in \bigcup_{y\in\mathcal{Y}} \mathcal{T}_y$ **do** ▷ iterate over all topics in all years
    $\rho \leftarrow$ candidate_chain(T)
    $rank[\rho] \leftarrow$ chain-score(ρ)
    $P \leftarrow P \cup \{\rho\}$
  for $1, \dots, K$ **do** ▷ select the K optimal topic-chains
    $\rho^* \leftarrow \text{argmax}_{\rho\in P} \left\{ rank[\rho] \right\}$
    $P^* \leftarrow P^* \cup \{\rho^*\}$
    for $\rho \in P$ **do**
      $s \leftarrow \text{jaccard}(\text{keywords}(\rho^*), \text{keywords}(\rho))$
      $rank[\rho] \leftarrow rank[\rho] \cdot (1 - s)$
  return P^*

procedure candidate_chain(T,s) ▷ create new topic-chain, starting with T
  $\rho_{y_s} \leftarrow T$
  $i = 1$
  while $\exists T \in \mathcal{T}_{y_{s+i}}$: topic-score($\rho_{y_{s+i-1}}$, T) > 0 **do**
    $\rho_{y_{s+i}} \leftarrow \text{argmax}_{T\in\mathcal{T}_{y_{s+i}}} \left\{ \text{EMA}_{\rho_{y_{s+i-1}}}(T) \right\}$ subject to topic-score($\rho_{y_{s+i-1}}$, T) > 0
    $i \leftarrow i + 1$
  return ρ

Time-fusion. This step is identical to the one described in section 4.3; some examples of time-fused topics can be seen in **??**.
The main difference is of course the structure of the output topic-chains; this new approach generated far more structurally strong chains, however, this did not come without drawbacks. The increased demand for time-consistency, and the fact that the algorithm used to create the topic-chains is greedy, resulted in far shorter topic-chains, with length that varies around 4-7 topics. The reason for this could be caused by the fact that the greedy approach can easily found itself in dead-ends, where it is not able to progress and is forced to return the estimated chain (see the while condition in the procedure `candidate_chain`). If the problem has optimal sub-structure[7] it may be possible to modify *Dijkstra*'s algorithm in order to find the topic-chain ρ (starting from a source node) that maximizes chain-score(ρ).

[7] Unfortunately, I think this problem does not have optimal substructure, but I do not have time to prove.

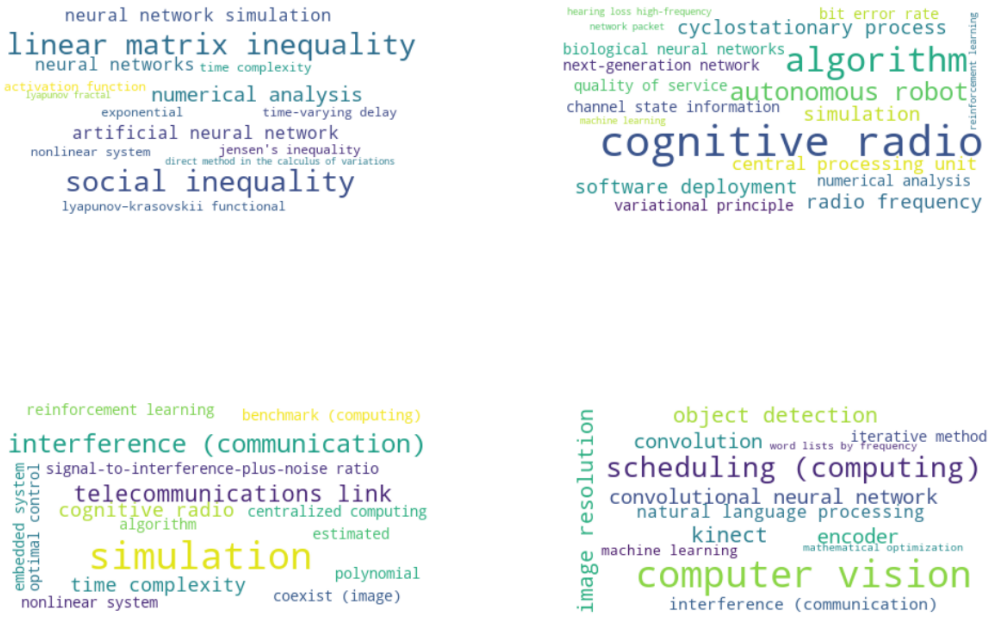

Figure 4: These are some examples of the K time-fused topics, that are computed during **Task 2**, the frequency of the word in the respective topic-chains is used in order to determine the font-size in the word cloud.

References

[1] P. Shakarian, A. Bhatnagar, A. Aleali, E. Shaabani, and R. Guo. *Diffusion in Social Networks*. Springer Briefs in Computer Science. Springer, 2015.

[2] H-J Zimmermann. Fuzzy set theory. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(3):317–332, 2010.

[3] G. Fagiolo. Clustering in complex directed networks. *Physical Review E*, 76(2):026107, 2007.

[4] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.