INF554 Data Challenge Presentation

El Mehdi Oudaoud Victor Fournier Adrien Kahn

December 14, 2021

Table of Contents

Features Engineering : Number of papers

The number of articles written by an author is an upper bound of the h-index. We add the constraint $1 \leq hindex \leq nb_{articles}$ for the authors who wrote less than 5 articles.

$$A_{<5} = \{authors, len(papers[author]) < 5\}$$

$$nb\_articles = \begin{cases} len(papers[author]), & author \in A_{<5} \\ mean_{A_{<5}}(hindex[author]), & author \notin A_{<5} \end{cases}$$

Features Engineering : Text Features

We used the IATE (Inter-Active Terminology for Europe) database as a reference for scientific terms.

One feature is the number of scientific terms used by an author. Another feature represents the "most scientific" word of an author.

$$A_{word} = \{authors | word \in papers[author]\}$$

$$score[word] = mean(\{hindex[author] | author \in A_{word}\})$$

$$best\_word[author] = max(\{score[word] | word \in papers[author]\})$$

Features Engineering : One-Hot Encoding

The one-hot encoding:

The goal is to construct a matrix of size $(nb_{words}, nb_{authors})$ such that:

$$M_{i,j} = 1 \Leftrightarrow word_i \text{ was used by the } author_j$$

The data that we have contains:

- 614718 abstracts
- 217000 authors
- $\approx 10^6$ words

Which leads to a memory problem as well as a runtime one !!

Features Engineering : One-Hot Encoding

We filter the set of words and use the *IATE* terminology dataset[1] to decrease the size of the one-hot encoding matrix.

The matrix now contains **18000** scientific words but we still have **614,718** abstracts, which still poses a problem for memory.

[1] `https://iate.europa.eu`

Features Engineering : Word2Vec

Word2vec

We also thaught about using Word2vec in order to give a numerical representation of the scientific words taking in consideration their context of appearance within the abstracts.

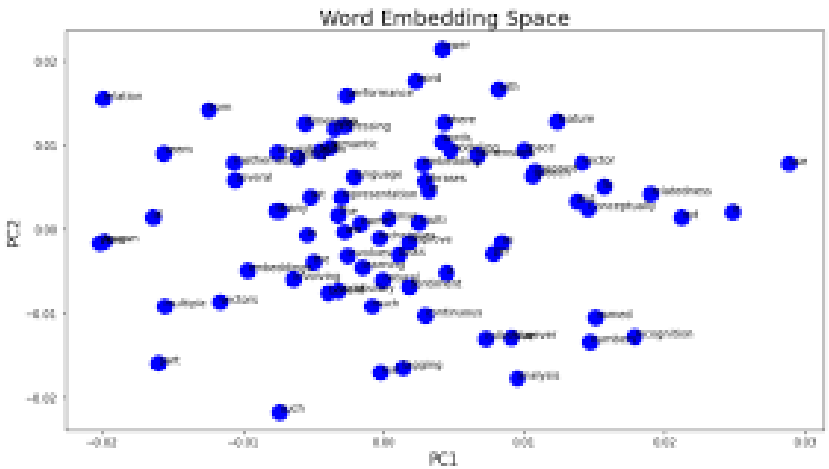


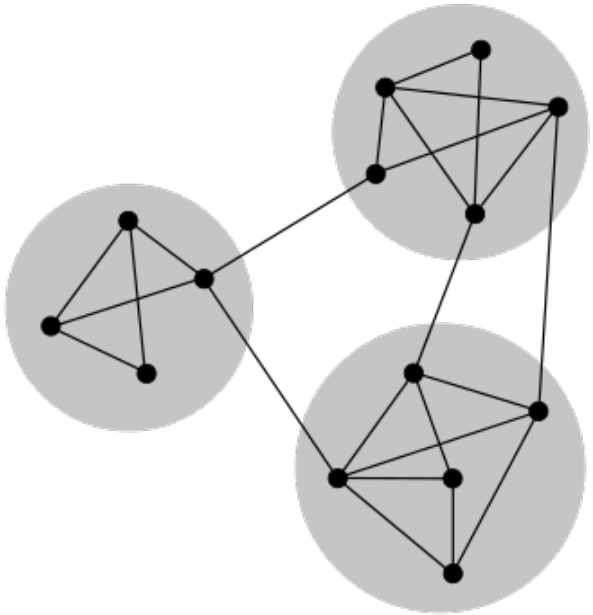
Figure: Pca on Word2vec of some abstracts

The goal is that for each author to take **the mean of the scientific words embedding vectors** that were used in his abstracts.

Community detection

Community structure

A graph has community structure when its nodes can be grouped into communities with few connections between communities.



Modularity

Modularity is a measure of a community partition's performance:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

Why community detection ?

- **Community-based features**: The number of link of a node to communities different than its own...
- **Community-specific features**: If the communities are more or less independent from each other, expensive graph features can be computed at a lower cost in each community subgraph.
- **Community-specific models**: Models might be more relevant if trained for a specific community.

What methods for community detection ?

Many different methods[2]:

- Min-Cut: Too slow, $\mathcal{O}(n^2 \log^3 n)$ with Karger's algorithm.
- Spectral clustering: Too slow as well, `scipy.sparse.linalg.eigsh` stops being reasonable after around 20000 nodes.
- Girvan-Newman algorithm: high performance but slow runtime in $\mathcal{O}(nm^2)$
- Node2vec
- Louvain algorithm

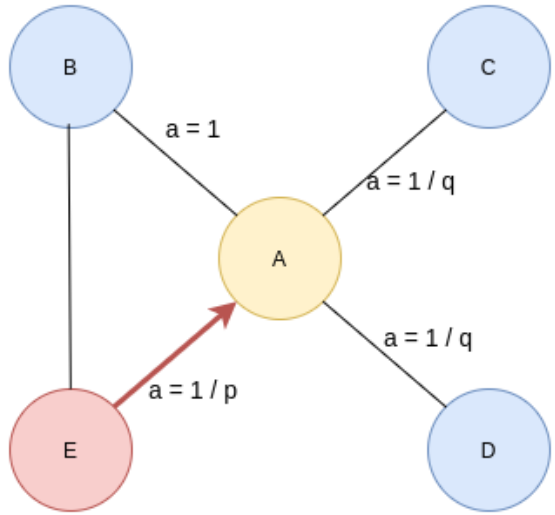
[2] Santo Fortunato. Community detection in graphs `https://arxiv.org/pdf/0906.0612.pdf`

Node2Vec

Second order biased walks

A second-order biased walk parameterized by p and q is walk that determines the probability of transition based on the distance between the previous node and the next node:

$$p \sim \begin{cases} \frac{1}{p} & \text{if } d = 0 \\ 1 & \text{if } d = 1 \\ \frac{1}{q} & \text{if } d = 2 \end{cases}$$



- Node2Vec is just like DeepWalk with second-order biased walks
- Allows for more subtlety in the definition of node similarity (cluster vs. global role)

Node2Vec in practice

- Could not perform clustering on the embeddings
- Lasso model on the embedding: MSE around 145

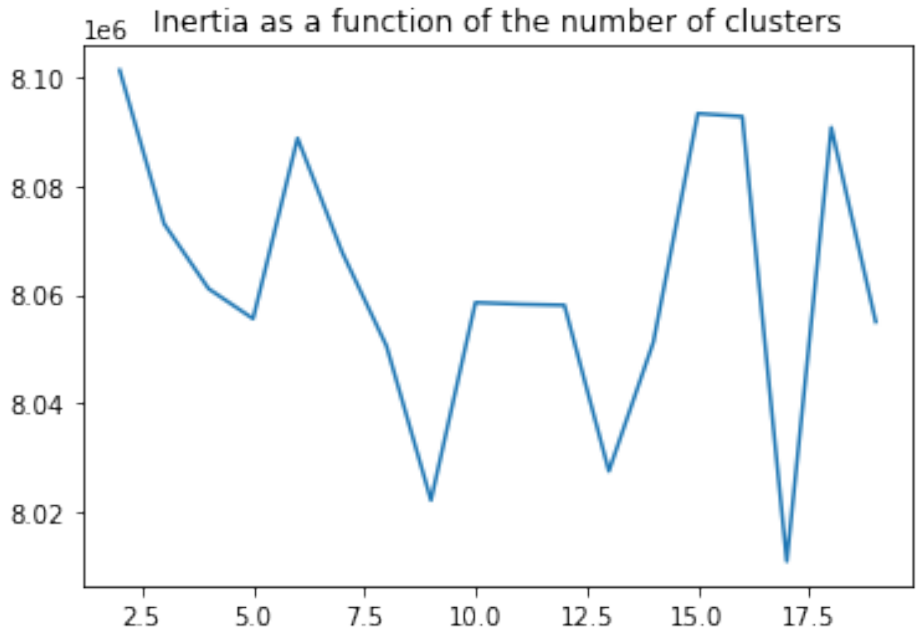


Figure: The k-means inertia as a function of the number of cluster on the node2vec embeddings

Louvain and Leiden algorithms

The Louvain and Leiden algorithms

- Greedy approach to modularity maximization
- $\mathcal{O}(n \log n)$ runtime
- Leiden[a] refines upon Louvain

---

[a] Vincent Traag, Ludo Waltman, Nees Jan van Eck. From Louvain to Leiden: guaranteeing well-connected communities
`https://arxiv.org/pdf/1810.08473.pdf`

Results with Leiden

Results

- Modularity of 0.87
- 247 clusters
- Largest cluster: 15795 elements
- Smallest cluster: 6 elements
- On average: 881 elements

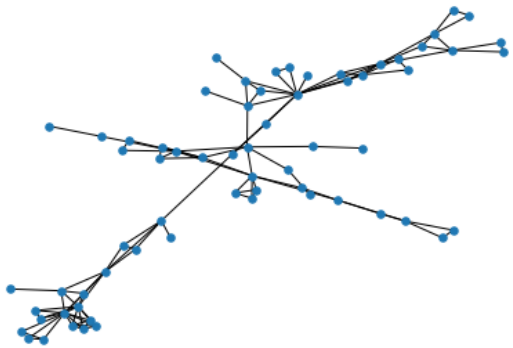


Figure: A community with 66 nodes

Models used for the regression

- Previous research[3] has shown that for *h*-index regression, lasso and neural networks are more than enough
- Lasso is convenient for easy identification of relevant features
- Neural networks are better than lasso but only for normalized data

Method	Text Features		Graph Features		All Features	
	MAE	MSE	MAE	MSE	MAE	MSE
Lasso	4.99	66.91	3.28	29.51	3.28	29.51
SGDRegressor	8.48	112.20	6.20	78.38	8.01	120.91
XGBoost	4.22	64.43	3.04	34.83	2.91	**21.04**
MLP	4.10	59.77	**2.62**	**22.46**	2.59	21.44
GCN	**4.05**	**59.45**	2.68	24.32	**2.57**	21.29
GNN	4.07	60.00	2.66	23.82	2.58	21.85

Figure: Table of the performance of different methods from this article

[3] Giannis Nikolentzos, George Panagopoulos, Iakovos Evdaimon, Michalis Vazirgiannis. Can Author Collaboration Reveal Impact? The Case of h-index https://arxiv.org/abs/2104.05562

MSE per feature group

Standard

	Baseline	Graph Features	Normalized	All Features	Hard Limit	Community
Lasso	132.2	115.6	112.1	99.4	98.29	102.4
NeuralNetwork	None	None	111.9	97.2	97.06	101.1

With Oversampling

	Baseline	Graph Features	Normalized	All Features	Hard Limit	Community
Lasso	130.0	114.0	111.2	98.36	95.84	100.2
NeuralNetwork	None	None	110.4	95.66	95.1	99.64