# INF 554 - Report of the Data Challenge

Team name : **RAKUEN**
Members :
Armel Randy ZEBAZE armel.zebaze-dongmo@polytechnique.edu
Mehdi Aiyad mehdi.aiyad@polytechnique.edu
Amadou Dieye amadou.dieye@polytechnique.edu

December 8, 2021

## 1 Introduction

The aim of the data challenge was to predict the h-index of some researchers based on the content of the abstracts of their most cited papers and the knowledge provided by a co-authorship graph. In order to perform the task, we divided our work into a phase of preprocessing & visualization and a phase of prediction. We use several techniques, some worked and some did not, we will try to be as exhaustive as possible while being short.

## 2 Data pre-processing

### 2.1 How to create text features?

One of the trickiest requirement of the project was the management of the file **abstracts.txt**. That file contains the ID of papers with the content of their abstract in the *Inverted Index* format. The first task we performed was to try to find a numerical representation of these abstracts. We tried several methods

- **Word2vec** & **Doc2vec** : These approaches used pretrained models to determine an embedding of words/sentences that take into account the meaning of each of them.
- **TF-IDF** : This approach is based on the relative frequency of a word into a given corpus. We found that it was more suited for our task since what matter the most when it comes to predicting the h-index is the field in which the author works. For example people working in NLP tend to be more cited than those working in Graph Machine Learning for example.

After having the numerical representation of each paper, we had to use them to find features for a given author. In the training set and the test set, a author is characterised by the list of its most cited papers. This could be handle in 2 ways :

- **Concatenation** : the first approach is to find a numerical representation (using one of the above methods) of the **concatenation** of the abstracts of the most cited papers of a given another.
- **Average** : the second approach is to find a numerical representation of each of the most cited papers of an author, and consider the **average** as the features.

That part of the project required us to process the context of **abstracts.txt**, so we created a method (**preprocess_abstracts.py**)whose the aim was to clearly separate the id of paper from its length and its content in the inverted index format. The rest of the project used the processed file (called **abstracts_bar.csv** in the notebook) rather than the original one.

### 2.2 How to create numerical features?

For a given author, we did not have a clear numerical information. But we had the potential to create some without using the co-authorship graph. These are some we used :

- **n_papers**: it was the number of the papers we had for a given author (between 1 or 5 in theory even though there were some articles mentioned that did not belong to the file **abstracts.txt**). the intuition behind that feature is the fact that prolific researchers (those with a high h-index) tend to have many high-impact papers.
- **length**: it was the average length (number of words) of the papers we had for a given author. The intuition is that a long abstract tend to be correlate with the degree of activity in the fields and the amount of results presented in the paper itself.

### 2.3 How to extract knowledge from the graph?

The graph could be use to retrieve a lot of information about authors and their collaborations. This could be used at the "model" level using models like **Graph Neural Networks**. Unfortunately it did not work because of RAM issues. We decided to rely on extracting pertinent features in the graph and add them to our text features to do the job. These features are :

- **page_rank**: That feature allowed to assess how important is a vertex for the overall graph. it relies on the number of edges going through that vertex.
- **core_number**
- **degree**: a prolific researcher tends to have a huge network of collaborators
- **average_degree**: a prolific researcher tends to collaborate with other great researchers to write papers. There are some exceptions in the case where a famous researcher co-author a paper with his/her students.
- **triangles**
- **clustering**

We had other features in mind such as the **closeness_centrality** and the **betweenness_centrality** but we could not use them because of the complexity of the algorithms that compute them. Another technique that we used but that did not perform well was to use **node embeddings** via Deepwalk or node2vec.

## 3 Prediction

### 3.1 Feature selection

Among all the features we extracted from the graph, we had to choose the optimal subset of features that would guarantee the best performance. We used the correlation with the output and we also checked the correlation between features to avoid redundancy.

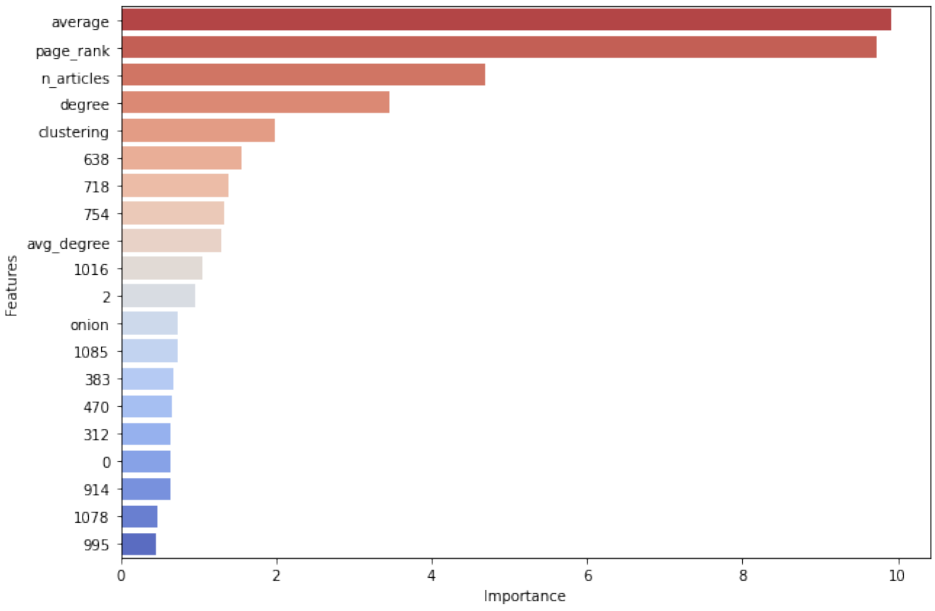


Figure 1: Feature importance as given by catboost.

| feature | Correlation with hindex |
|---|---|
| hindex | 1.000000 |
| onion | 0.459289 |
| degree | 0.406326 |
| page_rank | 0.402164 |
| average | 0.363316 |
| core_number | 0.359067 |
| avg_degree | 0.358154 |
| n_articles | 0.323564 |
| triangles | 0.306164 |
| length | 0.161609 |
| clustering | 0.125795 |
| author | -0.078556 |
| index | -0.193269 |

For the ideal submission we used **TF-IDF** with **1200** features, and we added all the numerical features and some of the graph features. These choices were guided on many tests performed on the dataset. The dimension of TF-IDF was constrained by the capacity of the RAM, a big RAM can reasonably allow a higher dimension, and it tends to improve the performance of our models. The feature importance gave us an indication on which features keep. The best subset was (onion, degree, page_rank, average, avg_degree, length, clustering, and n_articles).

## 3.2 Models

In order to make sure that we would have the best performances we decided to go with boosting regressors. The main regressors we used were **Xgboost** and **Catboost**. They had some parameters, which we chose using cross validation.

- **learning rate** : 0.05
- **depth** : 8
- **n_iterations**: 30000

With catboost and our best parameters we achieved a mean squared error around **44.0**.