# INF 554 Data Challenge - Kaggle - TEAM : Good To Go

Guillaume BARTHE[a], Temur MALISHAVA[b], and Gurami KERETCHASHVILI[c]

[a]*M2 Data AI, Guillaume.barthe@telecom-paris.fr*
[b]*M2 Data AI, temur.malishava@ip-paris.fr*
[c]*M2 Data AI, g.kerechashvili@gmail.com*

December 8, 2021

***Abstract*—The H_index of an author can be defined as the maximum value of h such that the given author has published h papers that have each been cited at least h times. The objective of this kaggle challenge is to use machine learning models to create an algorithm capable of predicting the h_index of a certain scientific author. This challenge was done as part of the course X-INF554 Machine and Deep Learning at école Polytechnique. Keywords - H_index, Regression, Neural Network, Embedding, Vocabulary, Stopwords.**

## I. Introduction

### A. Overview

The objective of this report is to summarize our approach to this data challenge which is hosted on **Kaggle** and you can find our code with this github link [1]. We are dealing with a regression problem using text data and graph data. Text data represents the abstracts of each scientific paper considered and the graph data describes the co-authorship in the data meaning that nodes are linked if they co-authored on a paper. The h_index we are predicting enables to measure the performances and reputation of an author regarding the number of citations he received for his publications. We will present how we **cleaned and processed** the data, how we combined both graph and text data and finally the architecture of our model.

The graph dataset we are using here contains 217, 801 vertices and 1,718, 164 edges in total while the text data consists of two different files : The list of most cited papers for each author (with a maximum of 5 papers) and the abstract of each paper in the dataset

### B. Evaluation Metric

The performance of our models will be assessed using the **Mean Squared Error (MSE)**. This metric is defined as the arithmetic average of the squared absolute errors. It is commonly used for regression tasks. It is defined as follows :

Therefore the objective of our work was to minimize this quantity in order to get the most accurate model.

## II. Data Analysis

### A. Graph Data

The graph used to train our model is an **undirected and unweighted graph** made with the networkx library from python. Vertices of the graph correspon to the 217, 801 authors in our dataset and an edge between author i and j means that they co-authored on a paper at least once.

$$\text{MSE} = \frac{1}{N}\sum_{1}^{N}(\hat{y}_i - y_i)^2$$

Fig. 1. Formula for MSE where N is the number of authors here, $y_i$ is the h-index of the ith author of the test set and $\hat{y}_i$ is our predicted h-index for that author.

In Annex A. you can see a subgraph of this co-authorship graph in order to have a better idea of what it looks like for some authors chosen randomly. (It is impossible to have an image of the whole graph since the number of node and edges is too large)

In order to have an even more precise idea of our graph, we decided to plot some statistics . Therefore, on Fig. 2 you can find **the histogram** representing the distribution of node degrees in the graph and on Fig. 4 a table summarizing different statistics for our graph.

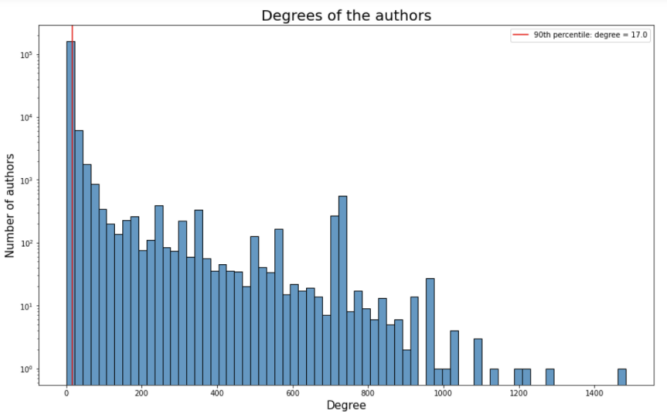


Fig. 2. Histogram representing the distribution of degree in the graph. 90% of the data is below the degree of 17

| | Number of nodes | Number of edges | Average Degree | Max degree | Min degree | Standard deviation |
|---|---|---|---|---|---|---|
| **Graph G** | 217801 | 1718164 | 15.871345 | 1483 | 1 | 68.7507 |

Fig. 3. Statistics for graph data. Very high standard deviation due to outliers

[1] https://github.com/gurokeretcha/H-index-Prediction

### B. Text Data

For our text data, we have at our disposal - in the file abstracts.txt - a set of papers with their abstracts, there is a total of 624181 papers. Most of these abstracts are written in english but is worth mentioning that a little bit less than 10% of the papers are written in french, german, italian and many more languages.

## III. Data Processing

### A. Graph Data - Feature Selection

The objective of this section is find the different graph **properties that are the most representative** for the prediction of the h_index. We looked through the networkx documentation and created a heatmap to find the correlation between the h_index and the different features accessible from the graph.

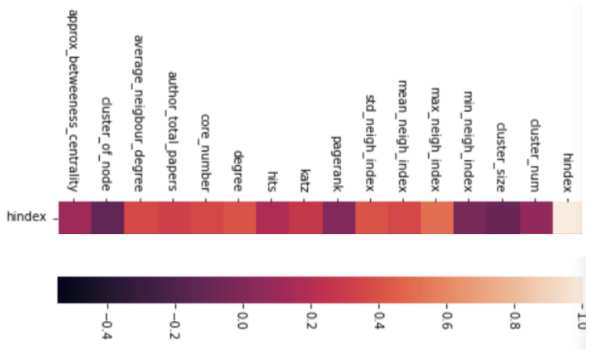


Fig. 4. Heatmap that represents the correlation between the h_index and all the graph features available . Created with seaborn library.

From Fig. 2 one can observe that the most interesting features seem to be [degree, core_number, author_total_papers, average_neigbour_degree, std_neigh_index, mean_neigh_index, max_neigh_index]. The features we selected and why are :

- **degree :** The importance of this feature is obvious, the number of co-authors is decisive for the calculation of h_index
- **Core number :** This represents the largest value k of a k-core containing that node. This enables to capture the structure in the graph and some partitioning of the authors in different groups
- **author_total_papers :** This is the number of papers written by each author. It cannot be higher than 5 but it allows better predictions when the number is lower than 5.
- **average_neigbour_degree :** The average degree of the neighbors of each node, which helps understand if an author is part of a large cluster.
- **mean_neigh_index :** The average index of the neighbors of a node is an important factor that is usually very representative of the h_index of that node.
- **max_neigh_index :** The best h_index among the neighbors of each node is also relevant for the predictions.
- **std_neigh_index :** Since we are using mean and max, it is also interesting to keep the standard deviation so that the model knows about the dispersion of the data

**Custom features :** In addition, we also tried to add a clustering feature. The intuition is that usually papers can be classified in different domains and some domains are more likely to be cited a lot, therefore it might be relevant to look at the clusters in the graph and add some features based on this clustering so that the model can better understand the structure. To divide the nodes into clusters we used The **Louvain method**. It is used for community detection and is a method to extract communities from large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network. The Louvain algorithm is a hierarchical clustering algorithm, that recursively merges communities into a single node and executes the modularity clustering on the condensed graphs.

Please note that we have also tried implementing a **cluster_size** feature in the following way: each author was assigned the size of the cluster it belongs to. This solution did seem to show promise at first, although unfortunately we did not have enough time to fully test it out in our final model.

Finally for feature selection we also tried **permutation importance** technique which is defined to be the decrease in a model score when a single feature value is randomly shuffled. this procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature.

### B. Text Data - Preprocessing and embedding

As we are dealing with paper abstracts, we decided to filter the text in order to use only the most important words from each abstract. We preprocessed the texts in the following way :

- **Step 1 :** Detect the language of each abstract and translate it to english when needed because the embeddings will be based on english words.
- **Step 2 :** Remove the stopwords by using NLTK library
- **Step 3 :** Use stemming in order to have a lower size vocabulary by grouping together all the words with the same stem.
- **Step 4 :** Lemmatization which is grouping together the different inflected forms of a word so they can be analyzed as a single item. This is similar to stemming but it uses a postagging parameter which is why we use both.

By doing this preprocessing we ensure that we have a robust vocabulary of the abstracts and reduce the noise in the text representations.

After cleaning the texts, we had to **embed it into numerical vectors** in order to be able to combine those vectors with the graph features. In order to do so, we combined all the abstracts

of each author and decided to apply different embeddings on those concatenated abstracts.

In order to convert the abstracts to vector of numbers we used different embeddings :

**TF - IDF**

The first embedding we decided to use was TF-IDF. This is the product of the frequency of the term (TF) and its inverse frequency in documents (IDF). This method is usually used to measure the importance of a term $i$ in a document $j$ relative to the rest of the corpus. By using this we were trying to have a model capable of finding the keywords of each abstract which could be very useful for the prediction of the h_index.

**Custom Embedding**

In order to have an efficient model we had to create a **vocabulary** representative of our text data. After the pre-processing described above we reduced the vocabulary from 600.000 words to 200.000. Then, we decided to discard all the words that would not appear in more than 10 papers in order to have a vocabulary size around 100.000 words. With the help of convolutional 1D layer and Global average pooling we managed to get a custom feature vector for each author and then we let the network learn the weights by itself.

**Universal Sentence Encoder**

The USE encodes text into high dimensional vectors that are commonly used for text classification, semantic similarity, clustering or other natural language tasks. The pre-trained Universal Sentence Encoder is publicly available in Tensorflow-hub.

The intuition here is that this encoder summarizes any given sentence to a **n-dimensional sentence embedding**. Since the same embedding has to work on multiple generic tasks, it is able to capture only the most informative features and discard noise. Therefore, this will result in a generic embedding that generalizes to wide variety of NLP tasks, this is why we decided to try it on our data.

## IV. Models

### A. Supervised models

The first models we created were based on the baseline code we were given and we tried to use Lasso Regression with our new features and embedded texts. In addition to Lasso, we also used some common **scikit learn models** like Random forests or linear regression. You can look at the comparison of those models with our final model further in this report in Table 1.

### B. Deep Learning Model - Combine the features

Since the constructed data can be divided in two parts, the numerical part and the textual part, we decided that it would be better to use different models for each of the cases and then combine the results.

For the numerical part we went with a simple neural network with one **dense layer** (We have tried adding more layers but this was slowing the training too much and not improving the model as much as we would have liked it to). The output vector of this layer is combined with the output vector from the textual part.

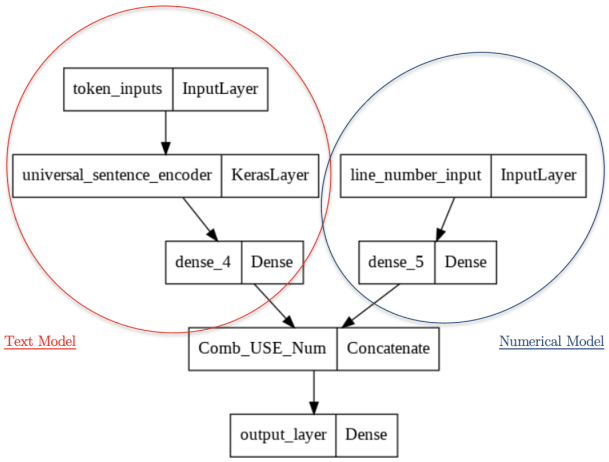


Fig. 5. Visualization of the model used for predictions. It combines the output weight vector of two different neural networks used on textual data and numerical data

As shown in Fig. 5, for the textual data our final model uses Universal Sentence Encoder which architecture was described above, and generated 512 feature vector for each sentence. It is worth mentioning that we unfreeze the USE layer to let the algorithm learn weights, and not just import pretrained weights.

Since we have huge number of trainable parameters we need to use techniques to **reduce overfitting**. First we used an exponential learning rate decay, in order to ensure that the change of parameters will be little over time. In addition, we used l1 and l2 regularization in the dense layers, because they disperse the error terms in all the weights which leads to more accurate final models. Finally we used early stopping callback to stop the algorithm when the validation is not improving anymore based on the validation loss.

Finally, we concatenate these two models and output the final result through a dense layer with linear activation function.

## V. Experiments and results

This final section presents the comparison between the different models presented in the previous section. Please note that the dataset we are using for this project is **imbalanced** - as you can see on the histogram in Appendix C. - some authors have a really high h_index (around 80) while more than 90% of the author have a relatively low index (less than 24). In order to tackle this issue we tried to use a log transformation on the h_index of the train set. Therefore in Table 1. you can find the final accuracy for all the models when training on the real dataset or the one transformed with log. Furthermore, you can find the histogram representing the distribution of the h_indexes after log transformation in the Appendix C. By applying this transformation, the data is more centered and we might reduce the impact of outliers.

There are multiple interepretations for those results, first of all the **log transformation did not improve the predictions** at all and the best score was achieved without it. Then, we

| Model used | Embedding | Score (raw/cleaned) | Score (log transform) |
|---|---|---|---|
| **Lasso** | **Custom** | 120.5 / **95.3** | 110.3 / 100.0 |
| Lasso | TF-IDF | 110.1 / 101.0 | 114.0 / 99.0 |
| Lasso | USE | 99.0 / 104.3 | 105.8 / 101.9 |
| Random Forest | TF-IDF | 140.1 / 95.7 | 110.1 / 101.0 |
| Random Forest | Custom | 125.1 / 79.3 | 107.3 / 80.1 |
| **Random Forest** | **USE** | **60.7** / 95.1 | 83.4 / 99.2 |
| Lin Regression | TF-IDF | 105.0 100.3 | 103.6 / 110.4 |
| Lin Regression | Custom | 120.5 95.3 | 110.5 / 105.5 |
| **Lin Regression** | **USE** | **85.9** 95.1 | 98.6 / 92.3 |
| NN architecure | TF-IDF | 105.1 75.3 | 120.3 / 95.0 |
| NN architecure | Custom | 89.3 63.1 | 87.3 /90.1 |
| **NN architecure** | **USE** | **43.3** 59.1 | 55.1 / 62.0 |

TABLE I
PERFORMANCES OF MULTIPLE MODELS REGARDING THE TRAINING DATASET (RAW ON OR PREPROCESSED ONE) AND THE EMBEDDING

realized that for the USE embedding it was interesting to consider the raw text data because it was trained to capture the context of each sentence and it was harder to do it when we removed words.

We observe that our Neural Network architecture - presented in the previous section - is a lot better than the prediction models from scikit learn and this is the implementation that gave us **an MSE of 43.3 which corresponds to the 3rd place on the Kaggle leaderboard.**

## VI. Conclusion and Future work

In this project we studied how to predict the h_index of an author which is an important measure in the scientific community. We mainly focused here on how to embed the textual data from the abstracts and on the best way of combining text data and graph data. The performances of our final model seem promising but please note that this work has been done in a **limited amount of time** and that this report is a only a brief summary of our work. You can find our code through this github link[2] which is well documented and might help to understand some parts of this report.

If we had more time we would have tried implementing already existing architectures for graph data such as **Graph Convolutional Network or Deep Walk**. Then, we struggled a lot with the fact that the data was imbalanced, normalization and log transformation proved to be inefficient but with more time we could find a way to deal with those imbalances. Finally the most critical part in this work is the embedding and we figured that one could create a more representative vocabulary that might lead to better results.

[2]https://github.com/gurokeretcha/H-index-Prediction

## VII. Appendix A : Graph Visualization

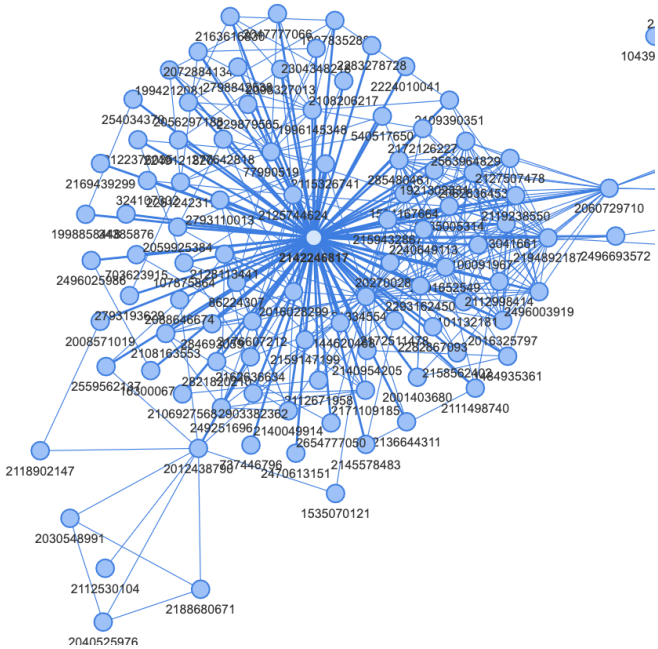


Fig. 6. Subgraph of the dataset. Here the author '2142246817' has a degree of 105.

## VIII. Appendix B : Universal Sentence encoder

We want to learn a model that can map a sentence to a fixed-length vector representation. This vector encodes the meaning of the sentence and thus can be used for downstream tasks such as searching for similar documents.

On a high level, the idea is to design an encoder that summarizes any given sentence to a 512-dimensional sentence embedding. We use this same embedding to solve multiple tasks and based on the mistakes it makes on those, we update the sentence embedding. Since the same embedding has to work on multiple generic tasks, it will capture only the most informative features and discard noise. The intuition is that this will result in an generic embedding that transfers universally to wide variety of NLP tasks such as relatedness, clustering, paraphrase detection and text classification.

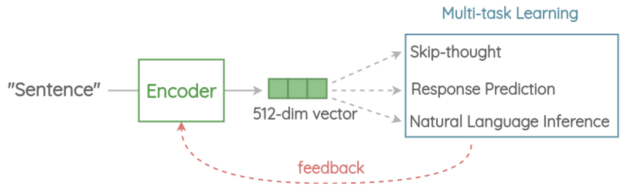


Fig. 7. Visual explanation of the USE embedding layer.

Each of the components of the USE has its purpose:

- **Tokenization** - the sentences are converted to lowercase and tokenized into tokens

- **Encoder** - encodes a sentence into fixed-length 512-dimension embedding. We will be using Deep Averaging Network (DAN). First, the embeddings for word and bi-grams present in a sentence are averaged together. Then, they are passed through 4-layer feed-forward deep DNN to get 512-dimensional sentence embedding as output. The embeddings for word and bi-grams are learned during training.
- **Multi-task Learning** - To learn the sentence embeddings, the encoder is shared and trained across a range of unsupervised tasks along with supervised training on the SNLI corpus like, use the current sentence to predict the previous and next sentence, predict the correct response for a given input among a list of correct responses and other randomly sampled responses, and predict if a hypothesis entails, contradicts, or is neutral to a premise.
- **Inference** - Once the model is trained using the above tasks, we can use it to map any sentence into fixed-length 512 dimension sentence embedding.

## IX. Appendix C : Imbalanced dataset

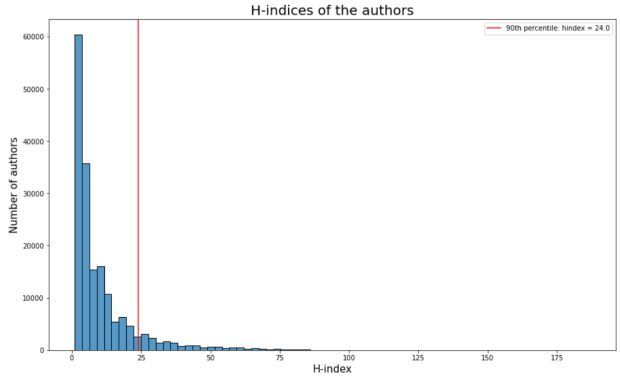


Fig. 8. Histogram representing the distribution of h_index in the dataset. 90% of the authors have an index lower than 24

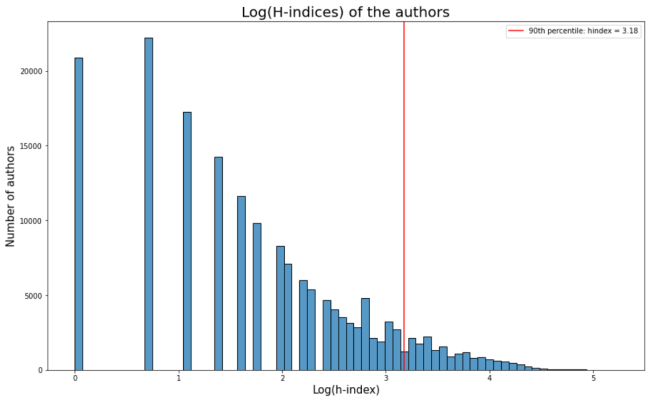


Fig. 9. Histogram representing the distribution of h_index in the dataset when using log transformation to remove imbalances. 90% of the authors have a log(index) lower than 3.18