ALTEGRAD Challenge: Predicting the h-index of Authors

Mohamed El Khames BOUMAIZA , Yassine FILALI
Institut Polytechnique de Paris
M2 Data Science
Team name: Mkaouar rak weldi
mohamed-el-khames.boumaiza@ensta-paris.fr, yassine.filali@ensta-paris.fr

Abstract

In this report, we present our approach for the task of predicting the h-index of authors, based on textual data and a connection graph between authors. We will present the different approaches used during this challenge, from graph embeddings, documents embeddings, and other aspects, in order to extract features from the available data to use it for this regression task.

1 Introduction

Graphs are powerful structures which are present in different problems, in which they add the connectivity information, and display the different relations between the observations, which are modeled by the nodes, in a problem. In our problem the connection between different authors is modeled by a graph. We will also deal with textual data, which will be used to add even more features to the observations. We will first start by presenting the problem and the data available in more detailed way, then we will present graph and text embedding approach used for this problem, and finally how we put them to use in order to perform the task.

2 Problem presentation

The task for this challenge is to predict the $h-$Index of an author, which are mainly authors of research paper, which is a metric used to measure his/her productivity. It's exact definition is as follows : It is 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 data provided for this task are a graph and a lists of abstracts for articles published by the different authors. In total, we have 5 files that are provided to perform the task:

- **An undirected graph**, which represents the co-authorship between authors, where the nodes represent the different authors (231239 nodes) and the edge represent whether two authors have co-authored an article or not (1777338 edges). This illustrates the collaboration between different authors.
- A text file, named **author papers** which contains the different authors and a list of their top-cited articles they have written or co-authored.
- A text file, with a list of articles, and the inverted index of their abstracts, which consists in a dictionnary where the keys are the different words, and the value is a list of it's positions in the abstract.The **abstracts** file is then processed in order to get a textual form.
- A file with the authorID of the training sample, and their respective $h-$Index which will be used to train the model. The repartition of 23 124 $h-$Indexes within the training data is as follows.

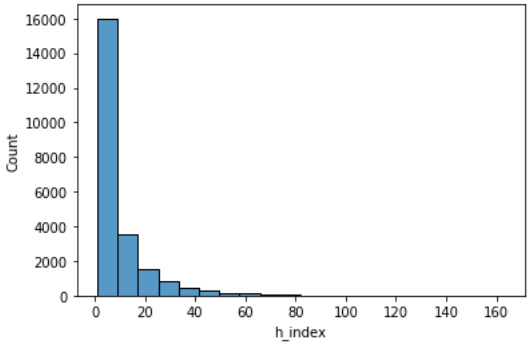


Figure 1: Repartition of authors w.r.t h-index in the training set

- A file with 208 115 authorIDs with the goal of predicting the h-index of each one of these authors..

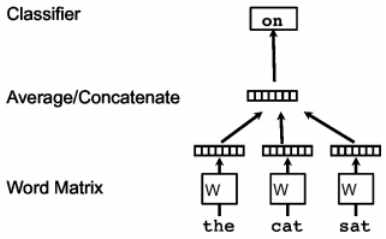
The main metric used for the evaluation of the prediction is the **Mean Absolute Error** = $\frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t|$, where N is the cardinality of the testing set, y_t the true value of the $h-$Index and $\hat{y}_t$ the predicted value.

3 Paper embedding

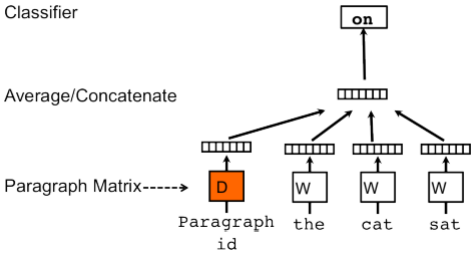
For the Text embedding part, we mainly tested two different NLP embeddings tools:

3.1 Doc2Vec

As a generalization of Word2Vec which can generate vectors for words, Doc2Vec uses an unsupervised learning approach to learn the document representation via the distributed bag of words models: where a text or a sentence is represented as a multi-set of its words. We are talking about generalization



(a) Word2Vec model



(b) Doc2Vec model

Figure 2: Word2vec VS Doc2vec structures

since we use the same word2vac model. However, another vector is added which is for instance the paragraph ID. Therefore, the document vector is trained to represent numerically the document.[3]
In our case, Word2Vec learns to project the processed abstracts into a latent 256-dimensional space.

3.2 RoBERTa

RoBERTa(Robustly Optimized BERT Pretraining Approach) model, introduced by Facebook, is mailnly based on the **bert** model (Bidirectional Encoder Representations from Transformers) through modifying its key hyperparameters, removing BERT's next-sentence pretraining objective, and training with much larger mini-batches and learning rates. [6]

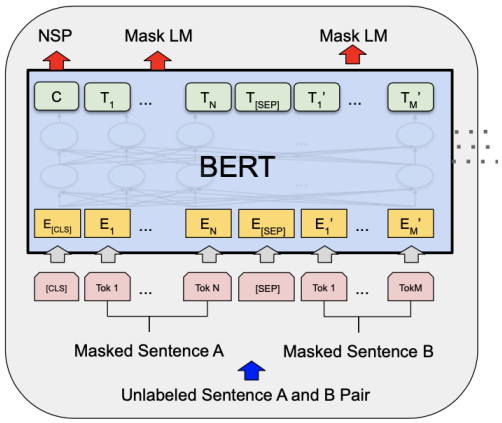


Figure 3: The illustrated Bert structure

The researchers used a new larger dataset for training : 160GB of text instead of the 16GB dataset originally used to train BERT and then trained the model over far more iterations. Eventually, RoBERTa outperforms BERT in all individual tasks on the General Language Understanding Evaluation and thath made it a very promising NLP pretraining tool.

The dimension of the embedding using roBERTa is 768.
The testing phase has proven that the Doc2Vec approach outperforms the RoBERTa since this latter is like a sentence-level embeddings and therefore it will not perform well on a document-level case.

3.3 Author embedding

After generating an embedding for the different articles, by embedding their abstracts, we proceed to generating an **author representation** from the embedding of his articles. Therefore, for one author, the vector describing an author will be the concatenation of different statistics over the embedding of his articles, such as the mean, the sum, the standard deviation, the minimum and the maximum over each dimension of an article embedding. After trying out different combinations of these embeddings, the ones that yielded the best results were the mean, the sum and the standard deviation over all the dimensions of the embedding. Therefore, we have a vector of dimension $256 \times 3 = 768$ representing each author.

4 Graph embedding

For the graph embedding, we mainly proceed in two ways: One with automatic, non supervised generation of graph embeddings using different models based on random walks like DeepWalk and Node2Vec. The other being the manual extraction of some node features in the graph.

4.1 Unsupervised embedding

For this part, we test out several unsupervised models for graph embeddings.

Deepwalk

We first start by generating an embedding using the DeepWalk algorithm, which prcoceeds in a similar manner to Skipgram[4]. The algorithm is able to learn a latent space nodes representations by encoding some important characteristics such that community membership and neighborhood similarity through random walks.
A random walk is a stochastic process, where we do k steps starting from a node $\nu = w_0$ and progressing through nodes $w_1, w_2, ..., w_k$ where w_t is chosen randomly (and uniformly) from the neighbors of w_{t-1}. The DeepWalk algorithm proceeds by generating n_{walks} random walks of length k for each node in the graph. Then, these walks can be seen as short sentences in some special language, and can be passed on to the Skipgram model. The core idea is to find an embedding with a certain mapping $\phi : V \rightarrow R^d$ that maximizes the likelihood of nodes in the walk, with regards of the mapping of the central node in the walk, where V is the set of nodes in the graph.

The problem is formulated as following :

$$\text{minimize}_{\phi} -\log \prod_{\substack{j=i-w \\ j\neq i}}^{i+w} P\left(v_j \mid \phi\left(v_i\right)\right)$$

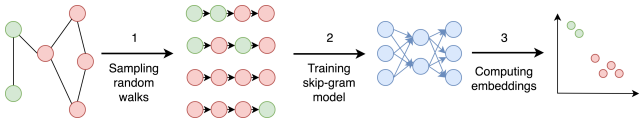


Figure 4: DeepWalk algorithm steps

Node2Vec

Node2Vec is another random walks based nodes embedding algorithm, but the main difference from DeepWalk algorithm is that in Node2Vec, when developing a random walk, there is a certain probability to go back to the previous node. The main features captured are homophily (same communities) and structural equivalence. In a more formal way, the transition probability from one node to another is no longer uniform, but follows this rule: for a random walk that has just traversed edge (t, ν), and now at the node ν, the transition probability from node ν to node v, $\pi_{\nu,v} = \alpha_{pq}(t, v)$ (In our case of an unweighted graph, so all edge's weights are equal to 1), with [1]:

$$\alpha_{pq}(t,v) = \begin{cases} \frac{1}{p} & \text{if } d_{tv} = 0 \\ 1 & \text{if } d_{tv} = 1 \\ \frac{1}{q} & \text{if } d_{tv} = 2 \end{cases}$$

where p, q are parameters to be tuned (for the exploration strategy), and d_{tv} is length of the shortest path from t to v.

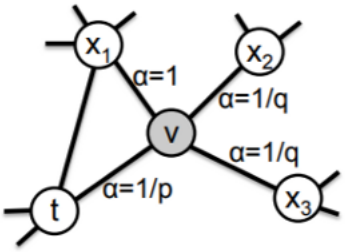


Figure 5: Transition probability in Node2Vec algorithm

ProNE

The main problem with Node2Vec and Deepwalk is that it takes a lot of time to compute. Instead, we try and another alogorithm, which was proved to be more efficient in terms of speed. It is shown to achieve good results in terms of speed compared to other embedding algorithms.[7].

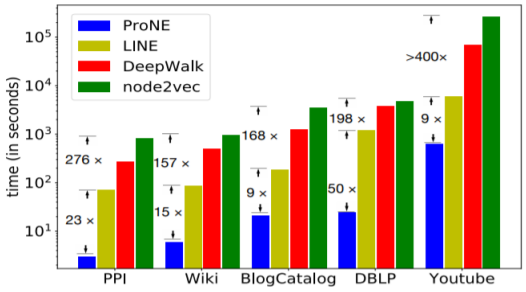


Figure 6: Speed comparison between different unsupervised algorithms[7]

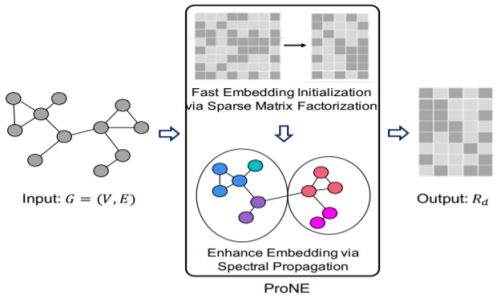


Figure 7: ProNE algorithm main idea[7]

In terms of results, the Node2Vec algorithm generated an embedding that gave better results than DeepWalk, and ProNE gave slightly better results than Node2Vec, but with a faster execution time compared to both other algorithms.

4.2 Manual feature extraction

For this part, We will extract a certain number of features, related to one node and it's neighbors. We will in most cases extract the feature of the node that we want to describe, and some statistics for this feature on the neighbors of the central node, which are the minimum, maximum value of the feature within the neighor nodes, the mean, the sum and the standard deviation. The features extracted are as follows:

- The degree of the nodes, which represent the number of neighbors of the nodes.
- The core number of a node, which is the largest value k of a k-core containing that node, where a k-core is a maximal subgraph that contains nodes of degree k or more.
- The number of article authored or co-authored by the author, which are extracted from the file containing the articles for each author (Therefore not graph based but we include it here since we have nowhere to put it).
- The betweenness centrality of a node ν, which is computed by the following formula [2]:

$$c_B(v) = \sum_{s,t \in V, s \neq v \neq t} \frac{\sigma(s,t \mid v)}{\sigma(s,t)}$$

where $\sigma(s,t)$ is the number of shortest paths from s to t, V the set of vertices of the graph, and $\sigma(s,t|v)$ the number of shortest paths from s to t passing from v.

- The clustering coefficient, which in our case, using an unweighted graph, the clustering of a node ν is the fraction of possible triangles through ν over the number of all possible triangles. This quantity is computed by the following formula [2]:

$$c(\nu) = \frac{2T(\nu)}{\deg(\nu)(\deg(\nu)-1)}$$

where $T(\nu)$ is the number of triangle passing through ν, and $\deg(\nu)$ is the degree of ν.

- The pagerank of a node, which is a metric used by Google to evaluate the importance of webpages. The pagerank can be evaluated iteratively, by the following recurrence formula [5]:

$$PR(p_i;0) = \frac{1}{N}$$

$$PR(p_i;t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j;t)}{L(p_j)}$$

Where $(p_i;t)$ is page i at time t, N The total number of pages,$M(p_i)$ the set of pages that link to p_i, $L(p_j)$ the number of links out of page p_j, and d a damping factor. We can assimilate the webpage to the node of our graph, and since this metric is used for webpages, therefore oriented graph, our graph is transformed into an oriented graph, by duplicating one undirected edge from our original graph into two edges, where each edge goes from a node to the other and vice versa.

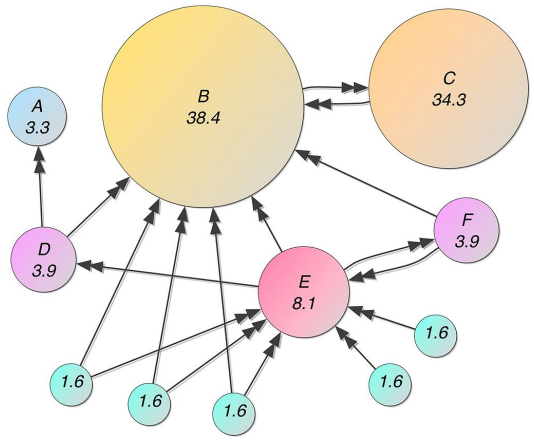


Figure 8: An example of the pagerank of certain pages in percentage. We can see that page C has a right pagerank, even though it is only linked to page B, because B is an important page.

Therefore, with all these features (minus the one with the number of publications for one author) and the statistics computed on the neighbors, we have a total of 35 feature extracted manually from the graph. It is to be noted that some of these features take a lot of time to compute on the whole graph, since the graph is large and some of them have an algorithmic complexity of $\mathcal{O}(nm)$ where n is the number of nodes and m the number of edges. Therefore, some approximations were made and the results were stored in intermediate files in order to avoid recomputing them over and over again.

In terms of results and after trying out different combination between the features extracted either manually or by an unsupervised algorithm, the use of only the features extracted manually gave better results than any other combination of features.

5 Preprocessing

5.1 Abstract preprocessing

As defined before, the input of the Doc2Vec should consist of the document it self as well as its ID.
Having the abstract of each paper which contains the ID and a dictionary where the keys are words and the values are lists of the positions of the corresponding words in the abstract, we would process the dictionary in order to alleviate this representation. The final format would be a text without its non-essential words as long as its ID.

5.2 Principal Component Analysis

Due to the big dimension of our features domain (the embedded space precisely), we opted for using a Principal Component Analysis for dimensionality reduction in order to find directions which data are highly distributed in order to eliminate correlated features. In fact, a decomposition with 60 as the number of components to keep among the 256 dimensions gave us a very consistent result.

6 Model

We tested different models after extracting the different features from texts and graphs, such as random forests, fully connected neural networks, and gradient boosting models. The model mainly used was a gradient boosting model through the framework **LightGBM** which parameters were tuned using a grid-search, with some manual modifications. The main parameters used related to this model are a number of estimator equal to 15000, a maximum depth of dree equal to 14, and a learning rate equal to 0.01. Out of all the tested models, this one offered the best trade-off between precision and execution time.

7 Conclusion

In this project, we had the opportunity to learn a lot of technics that a data scientist needs.Basically those technics are related to data engineering, feature engineering, fine tuning and document embeddings especially with the document to vector approach.
In this report, we presented the different tested approaches to perform the main task of our project . The final and best result obtained on the testing data is a mean absolute error of 3.32 which in terms of scale is not an absurd, and is not far away from the top ranked submission which is 2.99.

References

[1] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. 2016.

[2] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[3] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents.

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug 2014.

[5] Wikipedia. Pagerank.

[6] Naman Goyal Jingfei Du Mandar Joshi Danqi Chen Omer Levy Mike Lewis Luke Zettlemoyer Veselin Stoyanov Yinhan Liu, Myle Ott. Roberta: A robustly optimized bert pretraining approach.

[7] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. Prone: Fast and scalable network representation learning. pages 4278–4284, 7 2019.