INF554 Data Challenge: H-index Prediction

Michael Fotso Fotso, Tristan Francois, Christian Kotait

michael.fotso-fotso@polytechnique.edu, tristan.francois@polytechnique.edu,
christian.kotait@polytechnique.edu
Group name : **CMT**

December 8, 2021

1 Introduction

In this regression problem, each sample corresponds to an author and the goal is to build a model that can accurately predict the h-index of each author. Our pipeline contains features extracted from the graph (centrality, neighborhood features, PageRank, etc.) as well as features extracted from text using `fastText` [2][4][3], `Doc2Vec` and `TF-IDF` [7].

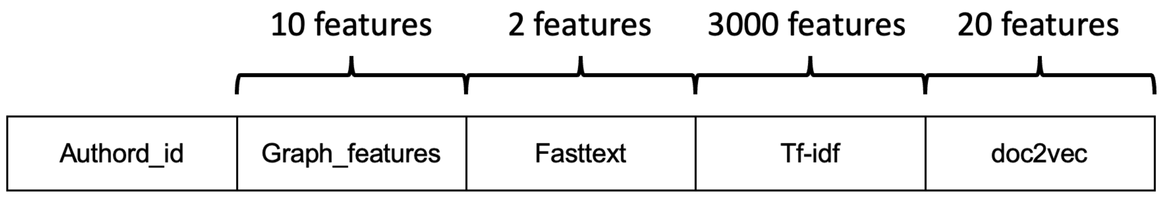

Figure 1: Summary of the selected features

We will also discuss the methods we used to select and extract features as well as the different models we implemented in our pipeline. Finally, we will show why we still have a lot of room to improve our score if we had a little more time or a more processing power.

2 Graph features

We first started our analysis by looking at different graph features. We tried to consider as many features as possible including different types of centralities (eigenvector, closeness, betweenness, etc.) as well as PageRank, clustering coefficients, authority and h-index neighborhood features (maximum h-index of coauthors, average h-index of coauthors, etc.)

We have drawn many conclusions on the graph features. First, we first noticed that neighborhood features were causing a huge overfit in our model when we included neighbors of neighbors features, especially the maximum h-index at distance 2. We were able to detect this dependance after studying the correlation of features and the value returned by the `mutual_info_regression` from `sklearn`. only select neighborhood features at distance 1.

Moreover, we were also confronted with computational difficulties when we tried to compute different centralities including closeness and betweenness centrality. In fact, these different algorithms have very high complexity. We decided to use `NetworKit`, which is a growing open-source toolkit for large-scale network analysis. Its aim is to provide tools for the analysis of large networks in the size range from thousands to billions of edges[1].

Ultimately, we selected the following 10 graph features : "Number of papers", "Core number", "PageRank", "Authority", "Clustering coefficient", "Number of coauthors", "Minimum h-index of coauthors", "Mean h-index of coauthors", "Maximum h-index of coauthors", "Closeness Centrality", "Triangles".

3 Text features

We have implemented different models to analyze and extract features from the authors' abstracts. In every subsection, we explain the advantage of using a specific model.

3.1 Pre-proccessing

As in all NLP tasks, pre-processing the text is a crucial step. That's why we have tested several differents pre-processing in order to be sure to have the most convenient approach to the problem. For all the abstracts, we transformed all the text to lowercase, removed the punctuation and deleted the numbers. We also tested two other variants with `NLTK` allowing to remove the stop-words or to stem the text. However, the simple `Gensim` pre-processing allowed us to ultimately achieve the best scores. Moreover, it is also important to note that we concatenated all the articles for each author in a single text. We also added "Number of articles", which takes into account the number of articles of each author. In addition, some authors do not have paper abstracts. In all our model, these missing values have been replaced by the mean vector over all the authors.

3.2 FastText

`FastText` was chosen for two main reasons. It processes data really fast and it has an auto-tune function that can be used to optimize the input parameters.

Indeed, we decided to train a `fastText` model for a text classification task and then compute a vector representation of the text thanks to the model. Indeed, knowing that the classification of `fastText` is done by the formula $softmax(ABx)$, the representation of the text x is simply ABx. As mentioned earlier, the shape of the matrix A and B have been obtained using the auto-tune feature. This helped us conclude that only two output features were sufficient to classify the text with an accuracy of almost 0.6 and a recall of 0.6.

To label our text into classes, we started by determining the number of classes. We chose 6 classes because it is the median of the hindex distribution. We have then trained fasttext in a neighborhood of 6 and it turned out that 6 was indeed the best number of classes. To determine how to assign a text to a class, we used K-Means on the set of h-indexes to have a homogeneous distribution in h-indexes (for example between 1 and 3 "Class 1", between 3 and 6 "Class 2").

3.3 Doc2Vec

Knowing that `fastText` could not take into consideration the complete semantic of our text, we decided to use `Doc2Vec` from the `Gensim` library [8].

That's why we decided to add to our features `Doc2Vec` vectorized text (which is actually an extension of `Word2Vec` that was seen in class). However, the fine tune of `Doc2Vec` costed a lot of time. Indeed, it was necessary to choose parameters, to train the model and finally to vectorize the text. We decided to vectorize in dimension 20,50 and 100 and it turned out that dimension 20 improved our score much more than the other dimensions. Thus, we added 20 features to the matrix, using `Doc2Vec`.

3.4 TF-IDF

The third model we used is `TF-IDF` [6]. We believed that the h-index was strongly dependent on the field of activity of the authors. In addition, we also thought that the relative frequency of words in the documents allowed to have a better vision of the document's domain. This is why we decided to use the `TF-IDF` vectorization on the abstracts.W progressively went from 1000 features to 5000 features and each time there was an improvement. It goes without saying that the more words we have the better we can identify the domains of the documents. However, we were not able to go beyond 5000 features due to time limitations. This is clearly where our Kaggle score can be improved if we had more time. In fact, our best score was achieved with 5,000 features and 100,000 iterations. The learning curve shows that having a higher number of iterations can drastically improve the performance of the model.

4 Pipeline creation and regressor choice

Once we had computed the feature vector of size 5033 for each author, we had to select the best regressor and find the optimal parameters from training and prediction. We have tested the different models seen in class: Rigde/Lasso regression, SVR, MLP, CNN and Random Forests. We also implemented a train-test split (0.2 for the test) on the complete training dataset.

We immediately noticed that Random Forests achieved lower MSE scores than the others on the whole dataset. This is why we chose to focus our attention on the models using decision trees, in particular those using gradient boosting methods. We found that the models `CatBoost`, `XGBoost` and `LightGBM` performed much better than all the others and had similar performance. We finally chose `CatBoost` [5] because we were seduced by the ease of implementation of fine-tuning that it offered, and also by the possibility of using the GPU.

Regarding the fine-tune, we performed a Grid Search on the depth of the tree and the Border Count with a fixed number of iterations. The learning rate was calculated automatically by `CatBoost` according to the size of the dataset. Regarding the number of iterations, we realized that the performance on the test set was growing strictly with the number of iterations, so we finally chose parameter as large as possible, about 100000.

5 Honorable attempts

During this project, we have invested a lot of effort in attempts we thought could be really fruitful and were unfortunately vain. We would like in this short paragraph to provide a list of honorable mentions for this methods.

The first attempt we would like to discuss was trying to create a new graph using Latent Semantic Indexing (LSI) of `Gensim`. The objective was to train an LSI model with 30 topics and then compute a matrix of similarity between all the authors. Using the values of similarities between authors, we were able to create a new graph and compute the same features discussed in Section 2 (PageRank, centralities, etc.) Unfortunately, we couldn't achieve a better score using these new features.

The second attempt was trying to create a cosine similarity matrix between all the `TF-IDF` feature vectors in order to perform a spectral clustering. These classes would serve as new features that would be concatenated to the previous 5033. The simple construction of the similarity matrix was a big problem because the memory did not allow us to store a matrix of size (200000,200000) even in sparse format. We tried other ways of clustering such as agglomerative clustering or DBSCAN. However, each time we were limited by our computing power.

We also tried to tackle the problem of imbalanced dataset. We knew that a lot of authors had low h-indexes and very few authors had high h-indexes. Thus, we tried over-sampling and under-sampling methods. Our trials were also unfruitful. We also tried to assign a high weight to the largest h-index during training; this also did not really improve our score.

6 Improvement opportunities and conclusion

To conclude, in this project, we have used graph features and text features (`fastText`, `Doc2Vec`, `TF-IDF`) to compute feature vectors for all our authors. We then used gradient boosting method to train our model and perform prediction. We believe that opportunities to improve our score are centered around `TF-IDF` : we would need to increase the number of iterations as well as the number of features in our vectorized text.

References

[1] Eugenio Angriman, Maria Predari, Alexander van der Grinten, and Henning Meyerhenke. Approximation of the diagonal of a laplacian's pseudoinverse for complex network analysis. *CoRR*, abs/2006.13679, 2020.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[3] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[5] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features., 2018.

[6] Juan Ramos. Using tf-idf to determine word relevance in document queries.

[7] Juan Ramos. Using tf-idf to determine word relevance in document queries, 1999.

[8] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.