INF554 – Data Challenge 2021

Group: B3 (on Kaggle)
Members: Bichot Lilian, Bienvenu Julien, Bienvenu Marie

Section 1 - Feature selection/extraction

First of all, we needed to pre-process the data, especially to remove from the texts the punctuation and some numbers.

Our main predictor is based on a regression model, like the one presented in the given baseline file.
We tried several regression algorithms: linear regression, lasso, ridge, and empirically the most efficient was the gradient boosting regressor, so we have kept this one.
The extra-parameter alpha has been chosen after having tested several values.
In terms of features, we have added to the degree and the core-number of the co-authoring graph the number of abstracts of the author in the author_papers file, the page-rank, the eigenvector centrality, and some predictors we have tried to use as features instead of predictors, like the average of the indices of the neighbors whose indices we know, an analytical words-based predictor, and a deep learning predictor based on words tokenizing.
All the features have been normalized, in order to follow a more rigorous approach.

Following the recommendation of a research paper [], we have started by adding the eigenvector centrality, but the values were fluctuating between 10^-10 and 10^-50, so the regression model was unable to fit this coefficient. We have chosen to log-scale it and to normalize it, but it has not been very efficient.
Therefore, we have tried to add as features many functions from networkX, and the one which we have found more relevant is the page-rank.
On the other hand, we have chosen to add as a feature the number of abstracts in the author_papers file. This choice is empirically quite relevant, as we can see in the following table. Indeed, an author whose number of papers is under -for instance- 3 cannot have a H-index above 3.

Later, after having plotted the H-indexes of the training set, we have tried to post-process the prediction file in order to prevent it from predicting above the number of papers the H-index of an author who published 4 or less papers. But, since we had already used the number of abstracts as a feature, this preprocessing was not very efficient and only improved the MSE by 1.

We have also built a NLP approach based on the Gensim GloVe unsupervised algorithm, in order to have a vector representation of the texts (concatenation of abstracts) of the authors. We have then used these embeddings as features in our main regression algorithm, and it gave us a Kaggle score of 70.
We have then plotted the words most similar to the vectorial representations of the authors (as a mean of the vectors of the words he uses), and we have noted that the words that appear the most are [that with this from which data paper using system based have used such results also].
So we choose to weight the barycenter of the words used by each author with the unsimilarity with the word the more similar to the author. Unfortunately, we started too late and our implementation, although complete, is still running ... We will present the results during our oral presentation.

Section 2 - Model tuning and comparison

In terms of extra-parameters, we chose alpha after having tested several values.

We have also developed alternative methods.

- Mean of neighbors predictor :
  - A simple graph-based predictor we built is to associate to every author of the test set the result of the application of a function on the H-indexes of this author's neighbors in the co-authoring graph that are in the training set (for instance the mean of these H-indexes).
  - Since this predictor had a bad MSE (147), we used it as a feature in our main regression algorithm instead of as a predictor in its own right.

- We also implemented several word-based predictors

- First, we wanted to have a clear approach on capturing text information. We thus built a word-based predictor with the following method : we associate to every word of the dataset a partial H-index, and then the predictor would associate to an author we do not know the H-index the result of the application of a function on the partial H-indexes of the words he uses (for instance the mean of these partial H-indexes). The partial H-indexes are fit by browsing all the authors of the training set and all the words they use, and by setting the partial H-index to the result of the application of a function on the

H-indexes of the authors of the training set whu use this word (for instance the mean of these H-indexes). With two means as the two functions mentioned above, we obtain a Kaggle score of 140.

- If you find it more understandable, we want a function which associates to an author of the test set a prediction based on the H-indexes of all the authors of the training set that use words that are used by the query author (taking into account the multiplicity.
- We did not have time to try more, but we could use other functions, and we also could fit the word partial H-indexes with an algorithm. Since there are a lot of words, it seems quite difficult with only a simple regression algorithm, but we could use for instance a stochastic approach on the partial H-indexes, with simulated annealing or alternative approaches.
    - We also tried to use this predictor not as a complete predictor but as a feature of the regression.

- Another approach, using the provided abstracts, was to tokenize the words using Google BERT's tokenizer and then to feed those tokens to a neural network (NN) using tensorflow. The NN has the tokenized concatenation of an author's abstracts (variable size) as input, and gives a unique float number as output. This float number is considered as the prediction of the author's H-index. The model is fitted using mean squared error as the loss function.
- Our NN's architecture is the following : first, some convolution layers ; then, a max pooling operation ; and finally, a few dense fully-connected layers, with some dropout. This architecture is heavily inspired by the articles [2] and [3] related to NLP and NN for regression.

- In order to prevent overfitting, for our neural network, we increased the dropout rate from 0.2 to 0.4, and it improved the MSE by about 20. Indeed, increasing the dropout higher than 0.5 would kill too many neurons.

- A general principle we have followed is to also allow our implementations to be run on a basic laptop, in order to force us to always keep a smart approach regarding the features. Our Kaggle score would maybe be a little bit better if we used all the methods of networkX, or if we trained a super powerful neural network with greats computers from Kaggle or somewhere else, but we have chosen to keep an understandable approach and to avoid using black boxes as far as possible.
- For instance, in order to keep a good comprehension of our algorithms, we have limited the number of features we use at the same time.
- Another example is that, except for the last submission, we have applied a PCA on the word tokenizing in order to keep a limited amount of features.

Section 3 – Extensions

In this section we discuss algorithms and extensions that we would implement if we had more time.

First of all, instead of using a gradient boosting regressor or other simple regression algorithms, we could use more powerful approaches (although less transparent) like neural networks, random forests, SVM, ...

Another way to predict an author's H-index could have been designed through clustering on the co-authorship graph, for instance using spectral clustering. Then, the prediction of a given author's H-index would have been a function (say, the mean) of the H-indexes of the authors from the same cluster.
In the same topic, but using the abstracts, we could have created another graph, where the nodes are words. Two words would have been connected if they were used by the same author, or if they are used in the same abstract. Then, two approaches are possible : with and without clustering.
With clustering, we could have associated to each author a cluster as the cluster of words containing the most words used by the author in his abstract. Then, a prediction of an author's H-index would have been a function (say, the mean) of the H-indexes of all the authors of the same cluster.
Without clustering, we could have computed for each node some key parameters such as the degree, the core-number, and other networkX features ; then, a feature (for each author) in the regression would have been a function (say, the mean) of the key parameters of the words used by the author.

Appendix- Bibliography and more

[1] McCarty, C., Jawitz, J.W., Hopkins, A. et al. (2013), Predicting author h-index using characteristics of the co-author network (https://doi.org/10.1007/s11192-012-0933-0)
[2] Usman Malik (2020), Text Classification with BERT Tokenizer and TF 2.0 in Python (https://stackabuse.com/text-classification-with-bert-tokenizer-and-tf-2-0-in-python/)
[3] Usman Malik (2020), Tensorflow 2.0: Solving Classification and Regression Problems (https://stackabuse.com/tensorflow-2-0-solving-classification-and-regression-problems/)

Degree, core-number	NoP	Mean of neighbors	Log-scaled eigenvector centrality	Word-based predictor	Page rank	Word tokenizing	MSE
X	X						116
X		X					119
X	X		X				116
X		X	X				114
X	X			X			116
X			X		X		103
X	X		X		X		94
X	X		X		X	X	77
Mean for all (similar to the dummy baseline)							157
Mean of neighbors							147
Word-based predictor, with pre-processing							138
Bert + NN							168

Keep in mind that, among all the features used in the lasso regression or gradient boosting regressor, only the "word based predictor" and the word tokenizing uses the words. The other ones only use the co-authoring graph and the number of papers. We see that the maximal Kaggle score we obtain while using only these features is 94.