H-INDEX REPORT INF554

THIBAUT VACEK, THOMAS LESPARGOT, PAUL-ADRIEN NICOLE

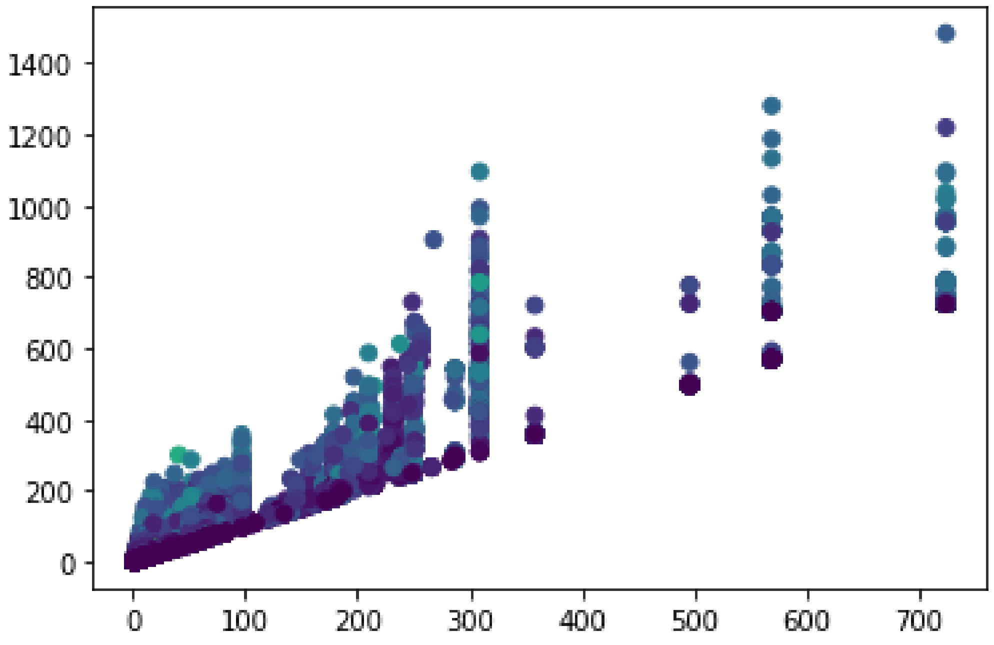
1. introduction

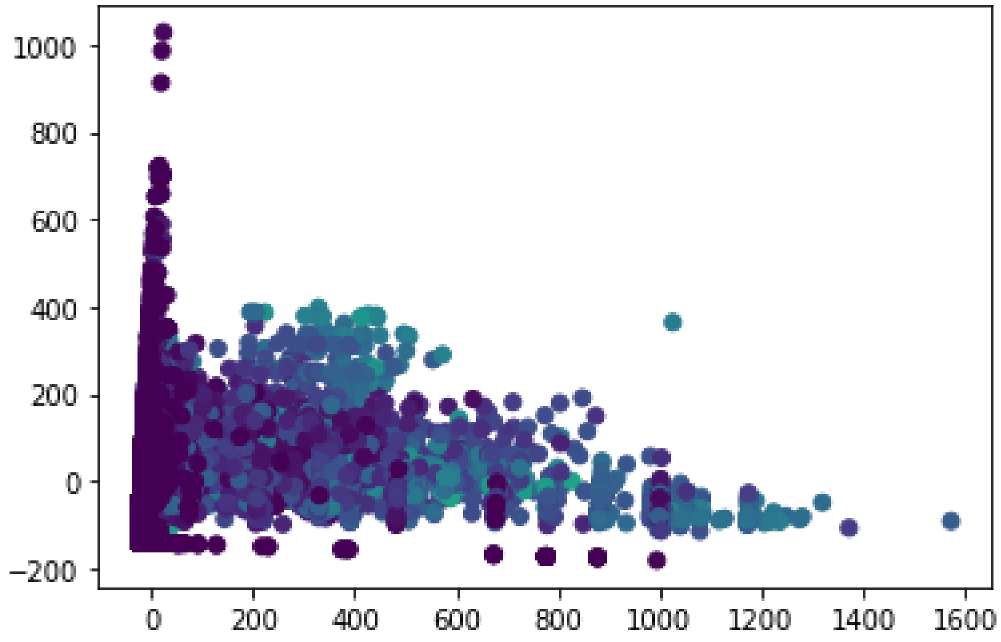
For this challenge, we implemented several models to predict the h-index of a Data set (it is a classification problem). In this data challenge we obtained the score of 68,77. The present report will explain the implementation of the different models and the choice of the different features. The score was obtained thanks to Scikit-learn and PyTorch. The main algorithm is a random forest with several features, that will be described. The choice of the hyperparameters was made thanks to GridSearchCV.

2. First analysis

Our first step was to look at the baseline solution. This solution uses two features from the graph: the core number and the degree for each node. In order to get an insight on the problem, we made the following graph where each point is an author (the lighter the color, the higher the hindex).

Date: 08 Décembre 2021.





Then, we added one more feature (the average number of word per article) and we made a PCA in order to plot a graph. We can clearly see that this is indeed a regression problem.

3. Interesting features

The first step was choosing the best features for the model. On the graph, we used the core number, the degree, the pagerank and the neighbor's average degree. On the text, we used NLP techniques which we'll describe later.

3.1. **Principal component analysis from the paper's abstracts data.** Once we had our new feature coming from the abstracts, extracted thanks to the Tfidf vectorizer (a NLP technique), we sought to reduce the dimension of this new feature (initially equal to 2000, absurdly big with respect to the batch size, and the number of other features dimensions). We decided to compute a principal component analysis to pursue our data processing. We took 20 components from the initial data, via the PCA, which is a dimension pretty much more reasonable. The data was then ready to be put in a model, as new features.

4. Neural Network implementation

Beside the random forest and the NLP, we wanted to test the efficiency of a neural network to predict the h-index. We implement our neural network thanks to the PyTorch library and tried to optimize it through the numerous parameters allowed in this type of problem. We decided to mainly use linear layers and ReLU activation functions on the outputs of the layers. The weights of the network were updated thanks to backpropagation with the Adam optimizer (torch.optim.Adam()), and of course with respect to an mse loss criterion. Once those basic choice were made, the model had to be optimized to improve the training and the general performance, with respect to the features, inputs of the NN.

The training of the model was improved thanks to:
- the choice of the batch size (depending on the training dataset size), important to update the weights the right number of times.
- the choice of the learning rate, which is an important parameter for the optimizer. Different trainings led us to choose the batch size equal to 1024, and the learning rate equal to 0.0001.

The general performance of our neural network was optimized thanks to the choice of the hidden dimension (size of the layers between the first and the last layers). We eventually chose $nhidden = 10$.

5. Random forest implementation

We also try the random forest algorithm. For this algorithm, we use one of Scikit-learn : the Random Forest Regressor algorithm. They are two hyperparameters to optimize for this algorithm : max_depth and n_estimators. n_estimators is the number of threes in the forest, and max_depth is the maximum depth of the tree. To find those two parameters, we use GridSearchCV. We give to the algorithm 4 values for max_depth and 4 value for n_estimators, and it finds the best combination thanks to a cross validation. We choose 5 as cross-validation strategic, that means, the train data set is split in 5 groups, 4 is used to train and one is used to test. The best estimators are found thanks to the function neg_root_mean_squared_error, which computes the loss as described in the Challenge.

Thanks to GridSearchCv we succeed to reduce the score from 110 to 105 with only Random Forest.

6. Building several models

The last idea was to separate the different features in 5 groups, depending on the number of top articles in the author_paper. On each group, we trained a random forest to be more performant. Thanks to this method, we arrived to the score of 68,77.

Email address: `thibaut.vacek@polytechnique.edu`, `paul-adrien.nicole@polytechnique.edu`, `thomas.lespargot@polytechnique.edu`