INF554 DATA CHALLENGE REPORT

December 2021

The eternal donkeys of ML

Josselin Somerville, Michael Pilcer, Matthieu Meunier

1 INTRODUCTION

This challenge's goal is to investigate and apply machine learning and artificial intelligence approaches to a real-world regression problem.
Each sample in this regression problem corresponds to a research article author, and the aim is to create a model that can reliably predict each author's h-index. An author's h-index, more specifically, evaluates his productivity and the influence of his works' citations. It's the highest value of h for which a certain author has authored h articles, each of which has been referenced at least h times.
We use two forms of data to develop the model: a graph that depicts the level of collaboration between two people and the abstracts of each author's five most cited works.
We had to combine large-scale data from many sources for this assignment. This presented a number of hurdles, including data preprocessing, model creation, and hyperparameter tuning. The mean square error (MSE) served as the basis for the loss function and evaluation.

2 FEATURE SELECTION/EXTRACTION

2.1 Co-authorship graph

2.1.1 • First baby steps

We started our work by tackling the issue of extracting features of the co-authorship graph. Since nodes represent authors (and not articles, like in the Cora dataset), it was not obvious to choose the features we would use to embed the nodes.

After a bit of collaborative thinking, we first tried to hand-pick features using the baseline notebook as a starting point. These engineered features contained (but were not limited to) quantiles, number of neighbors, average of the h-indexes of the authors encountered in random walks over the graph. We implemented this method ourselves and obtain an encouraging result of 86.2 on the test set.

2.1.2 • node2vec

Although our score was improving as we added more and more hand-picked features, we felt that it would be very tedious to reach the top of the leaderboard with this method. Therefore we tried a different approach: node representation learning. We browsed through some documentation and research papers, and thought that **node2vec** [6] could be a good solution. The idea is simple

- First, execute finite biased random walks on the graph, starting from each node
- By converting each author to a word, we have a corpus of sentences (each sentence corresponding to a random walk), on which we apply word2vec with skip-gram (see section 2.2.4 below for further details)
- Finally, we get a vector representation of each author

This algorithm is easy to implement, scalable, and allows to encompass structural similarity (similarity between nodes who have the same 'role' in different subareas of the graph) and/or nodes proximity (within the whole graph) depending on the choice of hyperparameters. The main hyperparameters we looked at that for this model were the probabilities in the random walk, and the dimension of the output vectors.

After having a look at the graph, we thought that it would be more relevant to focus on structural similarity. Indeed, in the graph, two neighboring nodes (i.e co-authors) can have very different h-indexes, but we can see some

structural similarities, such as the fact that authors with large h-indexes tend to be 'hubs' in the graph (they have co-authored a lot papers, which is directly linked with their popularity). Hence we chose to follow the recommendations of Leskovec and Grover (authors of the orginial paper for node2vec) and set $p = 1$ (return parameter, the bigger it is relatively to 1 and q the more likely it is that the random walk will explore far from the starting point) and $q = 2$ (in-out parameter, the bigger it is, the more the random walk will behave like BFS). As for the dimension of the output vectors, we decided to keep it relatively large (so that we do not risk loosing to much information), but still smaller than the dimension of the features extracted from the abstracts.

2.2 Abstracts

To extract features from the abstract, we needed to use natural language processing (NLP) algorithms. Our main heuristic is to determine the field and specificity (is the topic very niche or more general) of the paper, and deduce its popularity.

2.2.1 • Preprocessing

The raw text is important, but is too big and can't fit inside the memory, therefore, we need to preprocess it and keep only the important features. We can firstly remove *Stop Words* that don't contain any information. Then, we remove special characters and finally lemmatize the different texts. Lemmatization [4] is a process of determining a base or dictionary form (lemma) for a given surface form. Especially for languages with rich morphology it is important to be able to normalize words into their base forms to better support for example search engines and linguistic studies. We also need to remove some outlyers to train the model : most of the texts contain less than 1000 words, the other ones can therefore be reduced to this size. A few texts also only contain special characters and need to be removed from the training. To deal with the case when it happens with the test data, we can simply output the median of the algorithm.

2.2.2 • TF-IDF

We then tried a *TF-IDF* [1] : term frequency–inverse document frequency algorithm, that is intended to reflect how important a word is to a document in a corpus according to its frequency.

This algorithm weights a term's frequency (TF) and its inverse document frequency (IDF). Each word or term in the text has a TF and IDF score attached. The TF-IDF weight of a phrase is equal to the product of its TF and IDF ratings. Simply put, the greater the TF-IDF score (weight), the more uncommon the phrase in a particular document is, and vice versa.

$$\text{TF} - \text{IDF} = f_d(t) \times \log(\frac{N}{N(t)})$$

With $f_d(t)$ the occurence frequency of t in the document d, N the number of documents and $N(t)$ the number of documents containing t.

However, there are some limitations to this algorithm : in such a corpus, the vocabulary is very large, which can be a problem for learning. TF-IDF also doesn't take into account semantics and semantic similarity between words. Finally it assumes that the counts of different words provide independent evidence of similarity.

2.2.3 • GloVe

Another way to get word embeddings is GloVe [5]. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. GloVe Embeddings are a type of word embedding that encode the co-occurrence probability ratio between two words as vector differences. GloVe uses a weighted least squares objective that minimizes the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences.

2.2.4 • Word2Vec

To take semantic into account, we used a Word2Vec algorithm [2], pretrained from the *lexvec* repository. Word2Vec is a shallow, two-layer neural network that has been trained to recover word linguistic contexts. It takes a huge corpus

of words as input and generates a vector space with several hundred dimensions, with each unique word in the corpus assigned a matching vector in the space. Word vectors are positioned in the vector space so that words in the corpus that have similar contexts are close to one another in the space. Word2Vec is a computationally efficient prediction model that learns word embeddings from raw text. We loaded it thanks to *gensim* implementation of the algorithm. We firstly remove irrelevant words and lemmatize the abstract and then put every words as the input of this algorithm and average the outputs to get a vector representing the abstract. It can be useful to characterize the meaning of the entire abstract and can help draw conclusion on how popular an article can be.

3
MODEL TUNING AND COMPARISON

3.1 First experiments

We had quite a hard time determining which regressor would be best to use here. We tried simple linear regression, k-nn, Gradient Boosting Regressor, SVM and feed-forward neural network. The first models weren't convincing, k-nn and Gradient Boosting Regressor worked only with our first few features, but were too costly when dimensions got bigger. SVM worked but didn't give the results expected, so we chose the feed-forward neural network for this regression.

3.2 The final choice

Finally, we used a simple feed-forward neural network to take into account the different features. This Neural Network has an input of 300 (features from abstracts, using word2vec) + 128 (features from the graph, using node2vec) + 33 (engineered by ourselves) = 461 features and is composed of 3 hidden layers, the first two with 128 neurons, one with 64 and then outputs the h-index with the last layer. The activation function chosen is tanh and a linear activation for the output since it is a regression. The loss of this model is showed on figure 1.

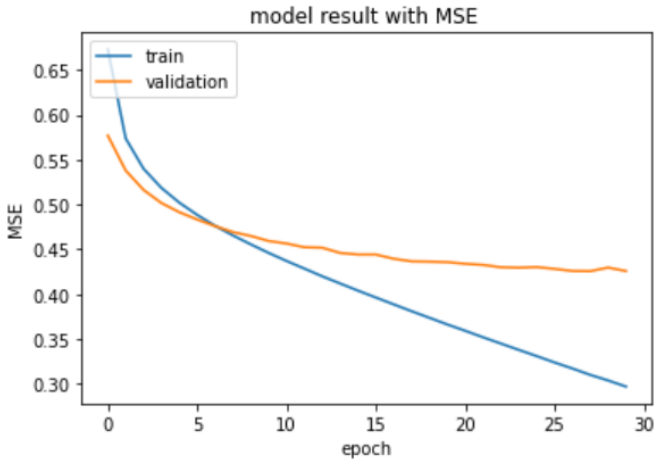

Figure 1: MSE of the neural network as a function of the number of epochs.

We evaluated the different models on a normalized validation set, which was a random split from the original train set. It is important to note that we have used a regression framework, but that it really is classification that we are trying to do (since the h-index is a natural integer). Therefore we had to clean the prediction made by the neural network: rounding to the closest integer and setting all predicted values < 1 to 1. The h-index was normalized in order to get more stability for the model. In the end, we got a score of 76.02 on the test set.

4
APPENDIX

4.1 BERT

To get meaningful features, we tried using *BERT* [3] (with DistilBERT and RoBERTa) which are open source machine learning framework for NLP. BERT is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context. It is used for translation and classification of texts and that's why it seemed a good idea to trying using it in our case.

It returns a vector of size 768 to represent the abstract, we then use a feed-forward neural network to reduce the number of features. Unfortunately, the model takes too much memory space and running it on the entire abstract dataset was too costly on Kaggle, so we didn't manage to use it next to our other feature extraction models.

REFERENCES

[1] (2011) TF–IDF. Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_832*

[2] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean (2013) *Efficient Estimation of Word Representations in Vector Space*

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*

[4] Jenna Kanerva, Filip Ginter, Tapio Salakoski *Universal Lemmatizer: A Sequence to Sequence Model for Lemmatizing Universal Dependencies Treebanks*

[5] Jeffrey Pennington, Richard Socher, Christopher Manning *GloVe: Global Vectors for Word Representation*

[6] Aditya Grover, Jure Leskovec (2016) *node2vec: Scalable Feature Learning for Networks*