ALTEGRAD challenge Fall 2020: h-index Prediction

Aymane Berradi, Taoufik Aghris, Badr Laajaj
Team Name: **Challengers**
M2 Data Science @ École Polytechnique
aymane.berradi@polytechnique.edu, taoufik.aghris@polytechnique.edu, badr.laajaj@polytechnique.edu

Abstract

During 2021's challenge in Advanced Learning for Text and Graph Data (ALTEGRAD) class taught by Michaelis Vazirgiannis, we worked on a regression problem of predicting the h-index of authors. This challenge was hosted on Kaggle and our code is available on Github.

1 Introduction

1.1 Context and motivation

In this report, we summarize the different research efforts and experiments that we conducted on ALTEGRAD challenge, which is hosted on Kaggle, concerning the prediction of the h-index of research papers authors. We are dealing with a regression problem using texts of articles abstracts and a graph network that matches the authors that co-authored the same papers. The target variable of this problem is the h-index which can be defined as the maximum number of published papers h that have each been cited at least h times. This metric measures the performance of an author with regards to the number of citations of his publications. During the challenge, we worked on a large-scale dataset, and followed the regression problem pipeline, that is we went from cleaning and preprocessing the data, features engineering to choosing the right regression models and tuning their parameters, and finally measuring their performances to choose the best ones for the competition.

1.2 Evaluation Metric

When it comes to measuring the performance of our models, the challenge asses the performance of our models using the **mean absolute error (MAE)**, which is the average of the absolute errors over the dataset. We can express it as:

$$MAE = \frac{1}{N}\sum_{1}^{N}|\hat{y}_i - y_i| \qquad (1)$$

Where N is the number of observations in the evaluated dataset, y_i^* the predicted value (h-index) for the ith observation and y_i its actual values. During this competition, we are trying to reduce the errors our prediction, thus we are looking for small MAE values. Finally, after evaluation our model on our train and validation sets, we predict on tests dataset, which we do not know its h-index values, and then submit the csv file into Kaggle the get a real evaluation of our performance.

2 Data Analysis

2.1 Graph Data

We are dealing with unweighted and undirected graph that models the co-authorship network, such that vertices correspond to authors where two authors (vertices) are linked by an edge if they have co-authored at least one paper, we have 231 239 nodes and 1 777 338 edges.
Let's see an example of what our graph looks like for a particular author:

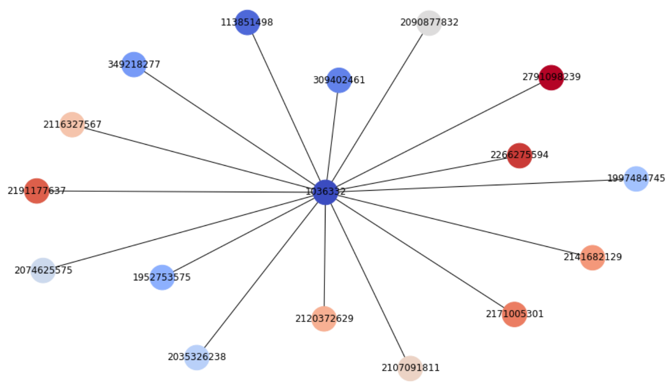


Figure 1: Subgraph of the co-authorship network

In the above example, we notice that the author with $id = 1036332$ has co-authored with 16 authors.
We also observe that there are no isolated nodes in the graph, which means that the author co-authored with at least one author. Moreover, when we examine the `author_papers.txt` document, we realize that even if there are two connected nodes by an edge, we didn't find any single common paper between them and this is due to the fact that the papers set per author contain the top cited papers.
To illustrate what is stated before, we take two nodes with degree 1, which means we should find at least one co-authored

paper between them in the `author_papers.txt`.

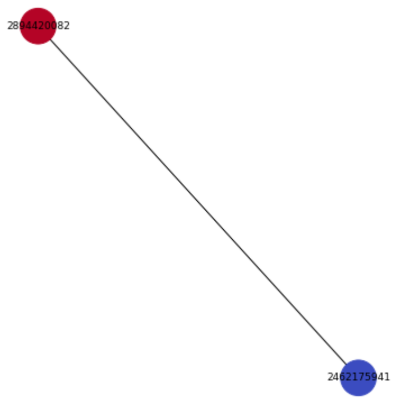
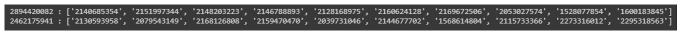


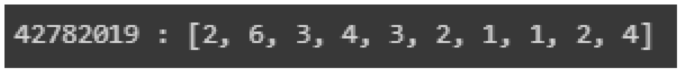
Figure 2: Example of nodes with $degree = 2$

Let's have a look on the papers set for each author:



```
2894420082 : ['2140685354', '2151997344', '2148203223', '2146788893', '2128168975', '2160624128', '2169672506', '2053027574', '1528077854', '1600183845']
2462175941 : ['2130593958', '2079543149', '2168126808', '2159470470', '2039731046', '2144677702', '1568614804', '2115733366', '2273316012', '2295318563']
```

We can figure out easily that there is no common paper between these 2 nodes, another observation is that the papers set for each author in `author_papers.txt` is made of single and co-authored papers.



```
42782019 : [2, 6, 3, 4, 3, 2, 1, 1, 2, 4]
```

In the above example, the author that have $id = 42782019$ wrote 2 papers by his own and co-authored in 8 papers. We are interested now in looking at the distribution of the nodes degree.

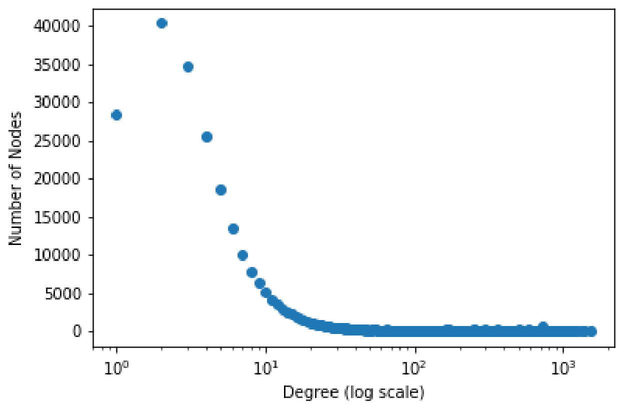


Figure 3: Distribution of node linkages

The figure 3 highlights that the degrees repartition within the graph is not uniform. In fact, there are 28409 nodes with degree 1, which represents 12.28% of the total nodes.
The histogram 4 shows that the range of nodes that goes from

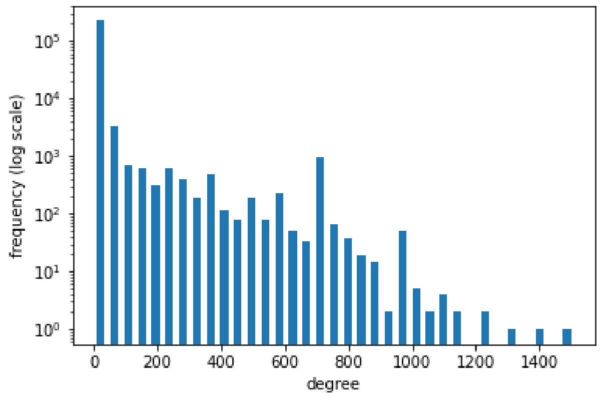


Figure 4: Histogram of nodes

1 to 1508 with the nodes count in log-scale in the y-axis, we notice that most of the nodes have small degrees.

2.2 Text Data

For the text data, we have at our disposal a set of papers with their abstracts, there is a total of 1 056 539 papers that are present in `abstract.txt` file. However we have only 1 056 298 papers that have abstracts so, 241 papers have missing abstracts. We notice also that there is more than one language in the abstracts set. In fact, there are 44 different languages dominated by English with 95.87% of the total languages.

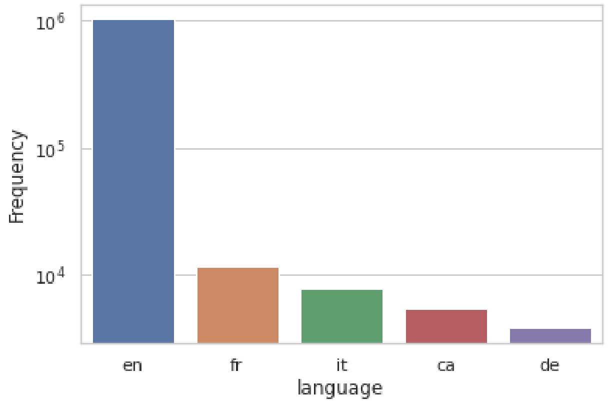


Figure 5: Bar plot of Top 5 languages

We present in figure 5 the top 5 languages used in the papers abstracts, as expected we notice that english and french are most frequent languages.

2.3 Train and test data

The training data contained 23 124 labeled authors, each sample have the author ID with the corresponding h-index, each author has his top-cited papers in `author_papers.txt`. For the test data, we have 208 115 unlabeled authors. The goal is to use the feature extracted from the graph and textual information to predict their corresponding h-index.

3 Text Preprocessing

As we are dealing with papers abstracts which represent text data, we need to clean it from white spaces, English stops words and to preprocess it so we can have same representation of the same word, as example for the verb "write" we can find many formats like "writes", "writing" so we need to normalize it to the same original word and thus have at the end the same vectorial representation of that word. This helps us build a strong corpus that represents main vocabularies of the abstracts and reduce the noise in the text representations. The provided data, `abstracts.txt`, is a .txt file where each paper ID is matched with an inverted index which is a dictionary where the keys are the words, and their values are lists of their corresponding positions in the abstracts. We can summarize this part of data preparation in the following steps:

- Read text from files using `utf-8` encoding to get almost all words present in the abstracts ;
- Loop over each paper id and extract the words in its inverted index ;
- We check if the lowercase of each word is alphabetic and neither a stop word nor empty ;
- If true, we lemmatize the word which consists on changing the inflected form of a word to its original format (as explained in example above with the verb write) so it can be analyzed as a single vocabulary. To do so, we used the **WordNetLemmatizer** from the **nltk** package. This lemmatizer is a pretrained model that inputs the word and its tagging (verb, noun, adjective and adverb) so it can find a meaningful base form based on the context. To get the tag of a word, we built a function `get_wordnet_pos()` that uses **nltk.pos_tag**, a pretrained tagging model from **nltk** package, and output a wordnet tagging as a parameter for our **WordNetLemmatizer** ;
- Now that we have our lemmatized word, we put it in its right positions in our abstract list using the inverted index values ;
- Finally, we output a dictionary with papers ID as keys and list of processed words in their right positions in the abstract.

4 Feature Extraction

In order to describe the structures inherent in our data and explain the problem at hand, we update the dependent variable using logarithm transformation and we extract some meaningful features from the text and authors data.

4.1 Features Engineering

To better understand our new features, we define the following elements:

- **Corpus**: It is a list of the unique tokens cited in all documents;
- **Co-authorship_1**: It describes the number of papers shared between every two authors;
- **Co-authorship_2**: For each author we count the occurrences of his papers in the whole file `authors_papers.txt`;
- **Abstract_ratio**: It is the ratio of tokens set (unique items) of each author's abstract to its length. (formula);
- **Corpus_ratio**: It describes the ratio of the tokens set in an abstract to the corpus's length for each author;
- **Author_freq_abs**: The frequency of each token that is repeated at least two time in the abstract over the abstract length;
- **Author_freq_crps**: The frequency of each token that is repeated at least two time in each abstract over the corpus length;

Based on the elements above we create the following variables for each author:

- **One_co_authorship_2**: It is the number of documents that are not co-authored for each author;
- **Docs_n**: It is the number of documents written by each author;
- **Max**, **mean** and **min** of **Co-authorship_2**;
- **Max**, **mean** and **min** of **Corpus_ratio**;
- **Max**, **mean** and **min** of **Abstract_ratio**;
- **Co_authored_sum**: The sum of Co-authorship_2 where there exist a co-authorship (occurrence paper is different to 1);
- **Max** and **mean** of **Author_freq_abs**;
- **Max** and **mean** of **uthor_freq_crps**;
- **Author_lang**: It is the number of languages each author uses to write their articles.

4.2 Graph Structure Features

We generate many node-level features based on the graph properties to allow our model learns more about the relation.

- **Node degree (nd_degr)**: It is the number of edges connected to the node,
- **Average degree (avg_ngbr)** : The average number of edges per node in the graph,
- **The core number of a node (core_n)**: It is the largest value k of a k-core containing that node,
- **Cluster** : The clustering of a node u is the fraction of possible triangles $T(u)$ through that node that exist (2).

$$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)} \tag{2}$$

We used the NetworkX library to generate the graph features above.

4.3 Log transformation

We use the function $y \rightarrow log(y+1)$ to guarantee that h_index is strictly positive. This transformation also helps to handle skewed data, reduce the effect of outliers. The 1 is added to avoid infinite values for h-index equals to 0.

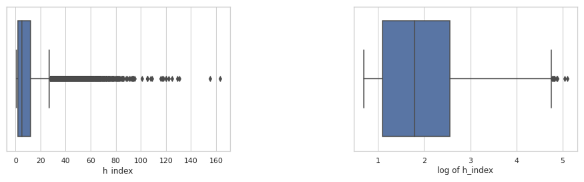


Figure 6: Box plots of h-index before (left figure) and after (right figure) log transformation

5 Models

In this section, we will showcase the global approach in the modeling phase. We started by combining the Doc2Vec with structural features of the graph in order to have an idea of the performance of two combined baselines provided by the challenge. We used two approaches for embedding text data: Doc2Vec and BERT, combined with PCA to escape the curse of dimensionality. As for the graph data, we applied deep walk for weighted and unweighted graph.
For the prediction part, we worked with the boosting family, namely: XGBoost, CatBoost and LightGBM and a stacked version of them using the average of their predictions and StackingRegressor

5.1 Doc2Vec

Doc2Vec is a paragraph vectorizer supposed to be an extension of Word2Vec that learns to project words into a latent d-dimensional space whereas Doc2Vec aims to learn how to project a document into a latent d-dimensional space. However, documents don't have the same length and don't have the same logical structure of words, that's why [Mikilov and Le, 2014] come up in their paper with a new concept that consists on adding a paragraph Id to Word2Vec structure as in fig 7:

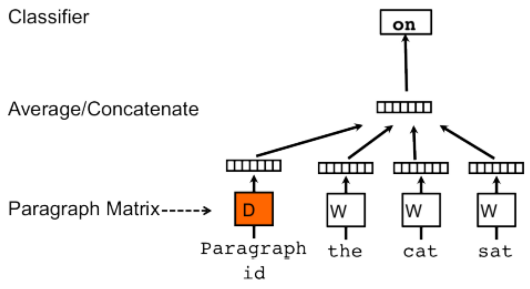


Figure 7: PV-DM

As for Word2Vec, we differentiate for 2Vec between PV-DM (Distributed Memory version of Paragraph Vector) similar to CBOW, and PV-DBOW (Distributed Bag of Words version of Paragraph Vector) like skip-gram.

- PV-DM (fig 7): In Word2Vec, CBOW tries to predict a center of word knowing the context. In Doc2Vec, PV-DM tries to sample randomly a set of consecutive words from a paragraph Id and predict the center word. It acts like a memory that tries to remember the missing word from the given context/topic in the paragraph.
- PV-DBOW: Like skip-gram from Word2Vec, PV-DBOW tries to predict the context given randomly sampled words from the paragraph ID as shown in the fig 8 below.

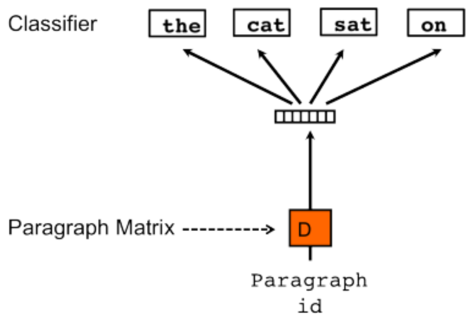


Figure 8: PV-DBOW

Using one of these two variants depends on the project requirement, that is the length of sentences where PV-DBOW is faster to train. However, PV-DM is known to perform better than PV-DBOW but takes longer time to train. In our case, we choose to work with PV-DBOW because we have a big dataset and small computational resources.

5.2 SPECTER: Document-level Representation

The recent Transformer language models (e.g. BERT) shows significant improvements in learning the latent representation of a text. However, these models use tokens and sentences as input to be trained, which limits their performance in a document level representation. The paper [Cohan *et al.*, 2020] comes out with a new method called SPECTER to create the embedding of scientific documents in a document-level based on the citation graph. This approach can be easily extended to different downstream applications without the need for fine-tuning. SPECTER is based on a pretrained language model for scientific text called SciBERT and incorporates citations. This model uses similar Transformer model architecture as BERT but it is pretrained on scientific text. SPECTRER takes as input the title and the abstract of a document, separated by [SEP] token and ended by [CLS] token, as an inter-document context to encode a given paper and generate its embedding. The following figure summaries the workflow of the SPECTRER and how was trained [Cohan *et al.*, 2020]:

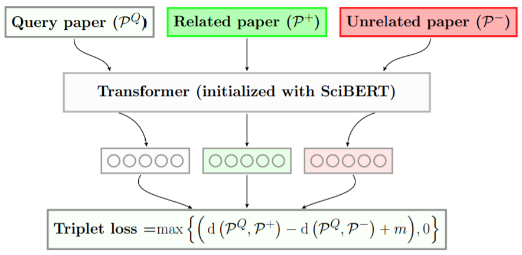

Query paper ($\mathcal{P}^Q$)
Related paper ($\mathcal{P}^+$)
Unrelated paper ($\mathcal{P}^-$)
Transformer (initialized with SciBERT)
Triplet loss $= \max\left\{\left(\mathrm{d}(\mathcal{P}^Q, \mathcal{P}^+) - \mathrm{d}(\mathcal{P}^Q, \mathcal{P}^-) + m\right), 0\right\}$

Figure 9: The workflow of SPECTER

The input is a triplet of papers: a query paper P^Q, related paper P^+ which is cited in P^Q and unrelated paper P^-. The negative paper is not cited in P^Q but to make the model more robust, P^- is randomly selected from the set of papers cited on P^+. In other word, P^- is cited in P^+, P^+ is cited in P^Q but P^- is not cited in P^Q. Then the model is trained using the triplet margin loss function where the L2 norm distance. Many tasks are used to evaluate the effectiveness of `SPECTER` by plug in the embeddings as features for each task but without fine tuning. So, this new model is presented to reproduce embedding for a new input documents without any citation information about it. `SPECTER` is accessible through HuggingFace's transformers library with which we generate the embedding vector of dimension 768 of the papers written in English (the Not-English papers is not considered) using only the abstracts. Then, to get the author embedding we simply average their corresponding documents embedding.

5.3 Deep Walk

`DeepWalk` is a type of graph neural network used to learn the latent representations of vertices in a network. This model operates in two stages. First, we create training data by traversing the network using random walk technique, where we start from a given vertex and go randomly to one of its neighbors until the path length (length of the random walk) is reached. This method allows to transform the graph into a new structure like the corpus for a text, that will be used in the second stage to train the `Skip-Gram model` to generate the embedding of each node. To implement Deep walk model we used the same algorithm described by [Perozzi *et al.*, 2014], but instead of using uniform distribution to go from a node to one of its neighbors we used the probability of co authorship, which is simply the weights of the edges. To train our model we set the following parameters values:

- `Word2vec`: $size$ = 256, $window$ = 8, min_count = 0, sg = 1, $workers$ = 8.
- `DeepWalk`: n_walks = 10, $walk\_length$ = 20.

5.4 The Principal Component Analysis

Due to the limited performance of our machines (12Gb RAM, CPU i7-9th) and the time of training which takes around 20 hours we were not able to test different configurations to generate the node embedding and the author embedding (with `Doc2vec`). To overcome this issue, we use the principal component analysis as a solution to eliminate the features that don't contribute in the prediction of h-index. So, the dimension of node embedding and document embedding are reduced from 256, 256 to 25, 64 respectively and these are the best two dimensions we found after testing different values. However, the proportion of variance explained with the two dimensions is respectively 28%, 41%.

5.5 Graph Convolutional Network (GCN)

`GCN` is inspired from Convolutional Neural Network (`CNN`), where we apply convolution on graphs to learn the features by inspecting neighboring nodes (like neighboring pixels in `CNN`). `GCN` is a generalized version of `CNN` because it does not operate only on regular (Euclidean) structured data (images), but also on irregular unordered structures where the number of nodes and connections vary. We have two categories of `GCNs` which are Spatial Graph Convolutional Networks and Spectral Graph Convolutional Networks, we focus in our case on the second type. `Spectral GCN` has similar idea as signal propagation that is the information propagate along nodes like signals. This model is based on eigen-decomposition of graph Laplacian matrix to make use of this information propagation between nodes. This `GCN` can be considered as message passing network where the information is propagated along the neighboring nodes within the graph. We used in our case `GCN` layer from the paper [Kipf and Welling, 2017] implemented within the `Spektral API`, which aims to learn hidden relationship of graph structure and features of each node, in addition this model was tested in the original paper on a semi-supervised (labels available only for a small subset of nodes) citations classification problem. This `GCN` layer computes:

$$X' = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}XW + b)$$

Where:
* $\hat{A} = A + I$: The adjacency matrix of the graph G with added self-loops
* $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$: degree matrix of $\hat{A}$
* W: Trainable weight matrix, b: biais
* $\sigma(.)$: activation function

We used 3 hidden `GCN` layers with 512, 256 and 128 units respectively and `ReLu` as activation function. And the last layer is a `GCN` layer with one unit since we are dealing with a regression problem. To avoid overfitting, we used `dropout` layers of 0.5 between hidden layers. During our tests, `GCN` gave us an MAE = 4.81 with features (`X_new.pkl` in `data` folder): structure features, Deep Walk and `Dov2Vec` in 2400 epochs. However, it failed to give good results (MAE = 5.47) when we added `SPECTER` (`BERT` embedding) which added around 700 features (`X_final_data.pkl`), which the model couldn't handle.

5.6 Boosting Algorithms

Boosting is a sequentiel ensembling algorithm that is designed to improve the prediction performance by training a sequence of weak models as shown in fig 10 where each model is compensating the weaknesses of its predecessors by giving important weights to the wrongfully predicted points. So, we need to specify the weak model.

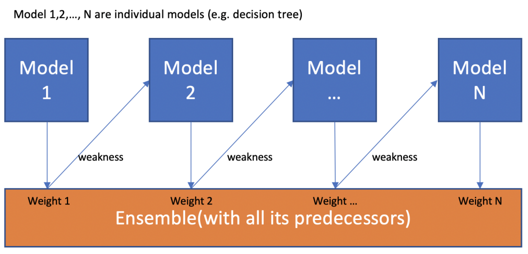

Model 1,2,..., N are individual models (e.g. decision tree)
Model 1
Model 2
Model ...
Model N
weakness
weakness
weakness
Weight 1
Weight 2
Weight ...
Weight N
Ensemble(with all its predecessors)

Figure 10: Example of boosting using decision tree

XGBoost
XGBoost, stands for eXtreme Gradient Boosting, is a gradient boosting algorithm that uses decision trees as its weak predictors. We limit ourselves to most important hyper-parameters used in our case:

- booster: is the boosting algorithm, we choose to work with **gbtree**, a tree based model.
- num_estimators: is equal to the number of boosted trees to use.
- max_depth: is the maximum depth of the decision trees.
- colsample_bytree: is the subsample ratio of columns when constructing each tree.
- learning_rate: is the step size shrinkage used in update to prevents overfitting.
- alpha: L_1 regularization term on weights.

CatBoost
CatBoost means 'categorical' boosting, is an open-source machine learning algorithm, developed in 2017 by a russian company named Yandex, it was designed to integrate a variety of different data types, such as images, audio, or text features into one framework. We used almost the hyper-parameters used for XGBoost.

LightGBM
LightGBM is a gradient boosting framework that makes use of tree based learning algorithms, it takes less memory to run and is able to deal with large amounts of data, and it is suited for large scale data. We choose **dart** for the boosting_type and num_leaves which is defined as the maximum tree leaves for base learners.

5.7 Stacking Models

In order to enhance the accuracy of our predictions, we used stacking technique which is a way to ensemble multiple classification or regression models. The idea behind is to tackle the learning problem with different types of models which are capable to learn some part of the problem.

Averaging method
We simply take the average of the predictions of XGBoost, CatBoost and LightGBM

Stacking Regressor
We build multiple different learners and we use them to build an intermediate prediction, one prediction for each learned model. Then we add a new model (linear regression in our case) which learns from the intermediate predictions the same target, we worked with Stacking_Regressor from Sklearn.

6 Results

In this section, we present the results of our experiments using different configurations of data with same predictive model (LGBM), and then applying all predictive models discussed earlier on the best data configuration.
The first step is to compare different data versions to pick the one that minimizes the mean absolute error on the validation data (33% of the whole data).

Data Configuration[a]	MAE
Doc2Vec + Deep-Walk	3.326
BERT + Deep-Walk	3.164
Doc2vec + Bert + Deep-Walk	3.116

Table 1: Different data configurations results

[a]In addition to features generated in features engineering section

We notice that the best data configuration is the last one Doc2Vec + BERT + Deep-Walk in addition to features generated in feature engineering section, so we will use this model to compare our predictive model, and choose the best model for submission.

Predictive Model	MAE
LGBM	3.116
CatBoost	3.279
XGBoost	3.255
Averaging Models	3.161
Stacking Models	3.114
GCN	5.473

Table 2: Different predictive models results

We conclude that stacking models gives the best result with the final configuration of features, however we can explain that GCN may need a lot of computational resources to be trained and give decent results. We submitted our final result in kaggle using stacking models and we got the 2.99 on the public leaderboard.

References

[Cohan *et al.*, 2020] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. Specter: Document-level representation learning using citation-informed transformers. In *https://arxiv.org/pdf/2004.07180.pdf*, 2020.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *https://arxiv.org/pdf/1609.02907.pdf*, 2017.

[Mikilov and Le, 2014] Tomas Mikilov and Quoc V. Le. Distributed representations of sentences and documents. In *https://arxiv.org/pdf/1405.4053.pdf*, 2014.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *https://arxiv.org/pdf/1403.6652.pdf*, 2014.