ALTEGRAD Challenge : *Predicting the h-index of Authors*

Team P-π

Maxime Lepeytre, Soumaya Sabry, Alexandre Zajac
IPParis - M2 Data science

Abstract–
The aim of this data challenge is the prediction of the h-index of scientific authors with limited supervision. For these authors, a graph is created where each node correspond to an author, and edges between two nodes indicate a coauthorship on a paper. Our method was based on a set of graph and hand-crafted features, with 2 different forms of representation for the authors, one created with the abstracts, and the other with their node-level embeddings. The problem was challenging because of the lack of training data, with the number of authors in the training set being 10x lower than in the testing set. Our approach achieved a score (MAE) of 3.26174 and ranked #5 on the Kaggle public and private leaderboard. The documented code is made publicly available on GitHub.[1]

1 Introduction

The problem studied in this challenge is a regression one, which a goal of building a model that can predict accurately the *h-index* of each author of research papers. We have a graph of co-authorship where each edge correspond to (at least one) paper(s) co-written between the 2 author, and we have the abstracts of the top-cited papers of each author, that can add a useful information to the prediction.

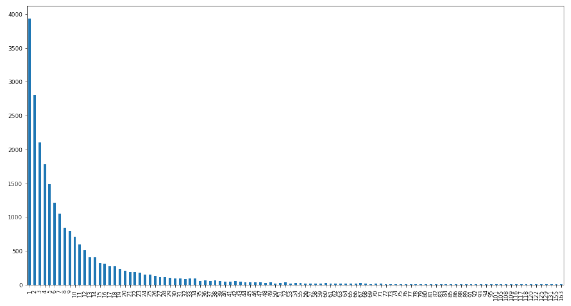
Figure 1: Distribution h_indexes

Our approach is 3-fold : First, we used state of the art transformers with finetuning to extract powerful features on the abstracts and represent the authors in a latent space. Second, we extract node embeddings via random walks in the graph of author nodes, yielding another authors representation. Finally, we combine these embeddings alongside with features generated from the graph structure of our data to generate the final predictions. The distribution of h-index values in the training data is represented figure 1. We can see that the h-index is a non-negative integer, and that it's largely distributed for values below 20, with only a few authors having very high h-index values.

[1]Github link :https://github.com/alexZajac/altegrad_kaggle_2021

2 Data

The training dataset is quite large with around 23,000 authors (rows). It is thus difficult to generalize the prediction on the test set, as it has 10 times more rows (around 200,000). While taking a look at the graph, we can see that it is one giant component without any separation, which indicates that all the authors in our dataset have at least one connection to another author.

2.1 Abstract description

Because of the diversity of authors and their words in the abstracts, we sensed that using the abstract's text to add more information about each author could give meaningful information to the final model.

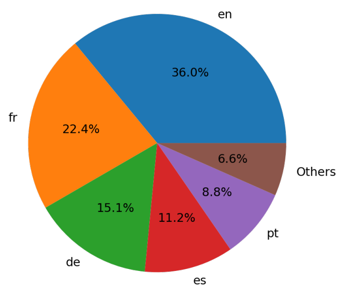

Figure 2: Languges percentage

Concerning the abstract languages, we found more than 40 different languages in the abstracts. As seen in fig.2, the dominant language is English with 36%, then French with 22%, and so on. For better results, we unified our data and translated all the abstracts to English. While this was a long task because of the data volume, we knew it will be worth it the long run because monolingual models on text are on average better than multilingual ones.

Some authors are missing their top-cited article's abstract. Which result on an empty abstract embedding in this case, and this looses the value of their row. An approach that we used, is to take the mean of the abstract embedding from authors linked to him in the graph.

So we started by cleaning the text of each abstract. The following processing pipeline was applied :

- Remove stopwords like: *and, to, in, with, by*...etc. using the utilities in *nltk* library.
- Translate using *GoogleTranslator()* class from the deep_translator python libary. We actually tried a lot of them including the official Google Translate API, but this one was the most efficient to avoid timeouts and authentification errors.
- Put all sentences to lowercase and remove punctuation marks

After this cleaning, we joined the text to get our final abstracts dataset that will be the input of the transformer model, as seen in the next table.

paper_id	abstract
1281918	article,describes,state,art,cloud,focusing,fea...
1461187	paper,describes,elements,inspection,allocation...
1695099	activating,communities,internet,catching,atten...

2.2 Graph weighting

Another data preparation we have done is modifying the input graph. We noticed that some authors that had coauthored a paper in the file authors-papers.txt weren't linked in the given graph, and that we could exploit that same file to make edges weighted. What we did is simple: we added a weight between 2 authors to represent the number of papers they have co-authored. As a side note, the minimum was obviously 0, and the maximum was 23. One other preprocessing step we applied only to compute structural features on the graph, is removing all the edges that were the only link between 2 authors (1-links): the number of connected components went from 1 to 115284.

3 Feature Extracting

3.1 Embedding the authors with the abstracts proxy

The transformations we applied to our text were exclusively designed for this part of the architecture. When we were reflecting on possible solutions to the h-index prediction problem, we knew that we could improve the representation of a given author by getting a correct proxy of his work and style only with his abstracts. In this context, we are describing here the first big choice in our final model, which is finding a precise way of representing the author solely with his abstracts.

Transformers is all you need

While we already obtained decent results with the provided Doc2Vec embeddings, we wanted to have a more meaningful representation of the authors via their abstracts. As stated above, a state of the art method in representing textual data in a latent space is the use of transformers, based on attention mechanisms.

We first tested to fine-tune a vanilla BERT model, as presented in 3, on our regression task, but it was difficult to converge to a loss value below what we established with the Doc2Vec embeddings, so we took 2 decisions to enhance its performance, choosing a variation of BERT and fine-tuning the classification head as well as the layers concatenation strategies.

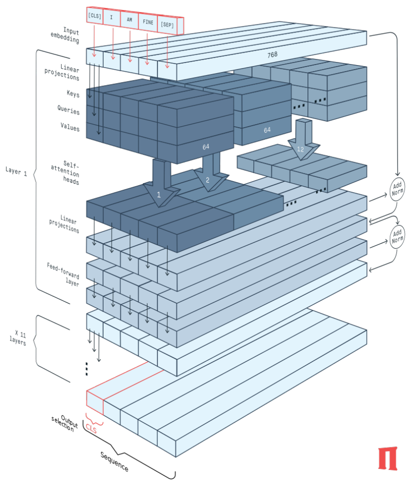

Figure 3: Overview of BERT's layers

SciBERT and Finetuning

We started to test variants of mono and multilingual models for different size such as bert-base-uncased, bert-large-uncased, roberta-base, roberta-large. We then searched for a variant of BERT that was adapted for scientific text: meet SciBERT[2]. It is a pretrained language model based on BERT [3] made to address the lack of high-quality, large-scale labeled scientific data. SciBERT leverages unsupervised pretraining on a large multi-domain corpus of scientific publications to improve performance on downstream scientific NLP tasks. In our case, the downstream task is the prediction of the h-index of the authors in the test set. We used the library Transformers provided by HuggingFace as an abstraction to interact with the SciBERT pretrained model. Its training corpus was papers taken from Semantic Scholar, in the size of 1.14M papers, 3.1B tokens, with the full text of the papers being used in training, not just the abstracts. Note that SciBERT has its own word-piece vocabulary (scivocab) that's built to best match the training corpus.

For our training corpus, given that we used the HuggingFace library, we didn't have to build a custom tokenizer, but we only needed to choose the token's max length.

For a an abstract, its tokenized version is represented fig.4. The tokens are then converted to their ids in the vocabulary and fed to the embedding.

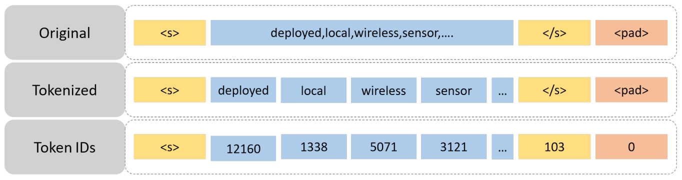

Figure 4: Tokenized sentence

To perform augmentation, we can choose to duplicate rows where authors have more than one abstract. Which give us in the end each rows with only one abstract and the same h_index for the authors.

For finetuning the base model, it was hard at first to know precisely which hyperparameter we had to tweak sensibly, and which parts of the networks we could safely retrain. from the hundreds of combinations of training setups we have tried, here are the general observations and parameter choices we retained:

- We could first either train only the top custom head we have placed on SciBERT, or retrain both this head and the whole parameters from the model. Even if the latter choice was more costly and dangerous since Transformers are highly sensitive to learning rates, it provided best results in the long run.
- Choosing this dual re-training strategy, we had to define hyperparameters for both retraining loops. Obviously, we used a higher learning rate, epochs and batch size for the top head training compared to the retraining of the whole body ($1e-4$, 3, 32 versus $5e-5$, 2 and 6).
- Speaking about batch size, we have chosen to use a dynamic mini-batch trimming to improve the training speed meaning we group batches of sentences with similar length and trim what is not necessary (zero padding), as taken from here.
- One major influence choice was the blending of hidden states at the end of the network, before being passed to the pooler for regression. This variety of choice is illustrated in figure 5. After a lot of trial and error, we finally settled on a concatenation of the last 4 hidden states with the embedding first layer of the network.
- The last notable choice was the top layer network for regression. Once again a large number of possibilities were offered to us for trial, But we settled with a rather simple version:
  - A linear layer with 512 as hidden dimension, with an Tanh() activation
  - A dropout layer of 0.4
  - A linear layer with 256 as hidden dimension, with an Tanh() activation
  - A dropout layer of 0.2
  - A linear layer with 1 final dimension, and no activation function

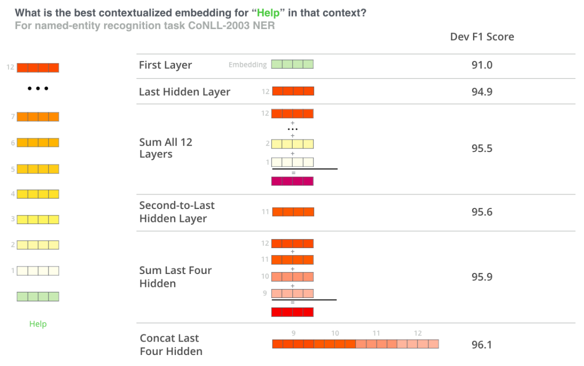
What is the best contextualized embedding for "Help" in that context?
For named-entity recognition task CoNLL-2003 NER
Dev F1 Score
First Layer
Embedding
91.0
Last Hidden Layer
94.9
Sum All 12 Layers
95.5
Second-to-Last Hidden Layer
95.6
Sum Last Four Hidden
95.9
Concat Last Four Hidden
96.1
Help

Figure 5: Overview of possible hidden state combinations with their precision influence on a downstream task

Obtaining the abstracts embeddings efficiently

Training was a long process but was definitely converging towards a lower loss (MAE) as we fine tuned the parameters. The last problem was the inference time for more than 2M abstracts. To tackle this issue, we used BERT as a service to benefit from better parallelism and reduce drastically the inference time, from a whole day to approximately 6 hours. Having no prior knowledge of the tool, we had to transfer our PyTorch model to a Tensorflow one with ONNX, but it was worth the shot.

3.2 Embedding the authors with the nodes proxy

A second big step towards representing and comparing authors is being able to represent them in a latent space, based on the embedding of their nodes in the graph of coauthorship.

GAE

We thus implemented a graph auto-encoder(GAE)[8] with an objective to learn a low-dimensional Z representation for each node. GAE is an unsupervised way to extract node embedding of the graph in which the target is the adjacency matrix A of our graph G, and X its feature matrix with a fixed dimension which we generated randomly. We tested to input features to the nodes obtained through the abstract embeddings, but allowing the network to learn from random initialization worked better.

Deepwalk

Deepwalk [10] is an approach for learning latent representations of vertices in a network, it uses local information obtained from truncated random walks to learn representations by treating walks as the equivalent of sentences. In our case, since we wanted to learn the social relationships between the authors, we decided to try this approach as well.

Others

We also tested different approaches such as Node2Vec [6] and LINE[11]. Each of them have different strategies to extract the node embedding. The Node2Vec model is based on sampling random walks of a fixed length in a given graph,

and node embedding are learned via negative sampling optimization. The LINE based on optimizing a designed objective function that preserves both the local and global network structures for large scale graph.
To choose the best suitable method to our approach, we tested each of the embeddings generated with cross-validation. As a side note, we tested a small Deepwalk, and the results were already more promising than with the embeddings obtained through GAE. The reconstruction of the adjacency matrix A might not represent as well the community structure of the authors as the random walks are. Deepwalk is thus more efficient at a local scale, and authors might be better represented locally than globally Tthe detailed results on the comparison of node embeddings are shown in the *Result* section.

3.3 Node Features

Enriching the model with node-level features from the graphs was a no-brainer since we saw that it was definitely adding potential to the baseline model alone. We have leaved the core number and average neighbor degree and added a few others from the networkx library:

- The onion layers factor, which defined the layer of each vertex in an onion decomposition of the graph: it refines the k-core decomposition by providing information on the internal organization of each k-shell.
- We added the degree centrality, which is the fraction of nodes connected to a specific node.
- We also added the clustering coefficient, as a measure of local density of connections for a specific node.

Here are the other structural features we have added that helped the model: *degree, clustering, eigenvector centrality, pagerank, triangles and greedy color.*

One step of improvement for the final model was the addition of handcrafted features based on logic and estimation of what can influence an h-index:

- We began by simply adding the number of papers per author, which reveals as being the most important variable in the decision process.
- Following that direction, we added statistical coefficients (mean, median, quantiles) on the average number of papers made by the k-nearest neighbors of an author (node in graph).
- We also added the weighted number of collaborators from an author, which stands for the number of coauthors of all his papers divided by his number of papers.
- Finally, we added the count of papers from an author divided by the count of its 1-neighbors, to represent influence at a small scale.

4 Modelling

4.1 XGBoost

XGBoost (or eXtreme Gradient Boosting) is a boosted tree algorithm.

It is important to understand these notions:

1. Boosting:
   The idea is to correct the errors made by a model by adding new models.
2. Gradient boosting:
   The idea is to add new model that predict the errors of prior models.

We first try to train with XGBoost because it is fast compared to other implementations of gradient boosting, can automatically do parallel computation on Windows and Linux and also it is the state of the art for a lot of kaggle challenges.

Our XGBoost implementation combines all previous output (node embeddings, abstract embeddings and features engineering (Graph & text)), but we found that it's predictive power alone could be enchanced, so we started to consider other regression techniques as well.

We have considered to use 2 strong Gradient Boosting algorithms first, which were LightGBM [7], which was quite adapted to our use case because the efficiency and scalability of XGBoost is less satisfactory when the feature dimension is high and data size is large, which was our case since we had at least 800 different features for each author. We also experimented with CatBoost [4], even if it is designed for categorical variables, it still provides meaningful feature learning for this types of regression problems. We also tested "weeker learners" such as Random Forest Regressors and AdaBoost from scikit learn, but they weren't more performant than the booting algorithms

Combining the optimization power and boosting logic of XGBoost, a search on the best parameters for each set of features allowed us to break below the 4 score on the leaderboard, with a local score of 3.9124 and a leaderboard score of 3.81601. We also observed that the higher number of features we imputed, the lower MAE score was.

4.2 Graph Neural Networks

We wanted to try using Graph Neural Networks for the task because we felt that the structural features of the graph could be a better predictor than the embeddings of abstratcs alone. We found an outstanding library for using GNNs called Pytorch-Geometric [5], which allowed us to experiment quickly on our dataset. Most of the approach we tried were based on Graph convolutions or Graph label propagation, because they seemed to be the ones that achieved state of the art performance on co-authorship node classification tasks. Label Propagation (LPA) and Graph Convolutional Neural Networks (GCN) [12] are both message passing algorithms on graphs. Both solve tasks at a node level but LPA propagates node label information across the edges of the graph, while GCN propagates and transforms node feature information. It was interesting to try both approaches to analyze how the feature/label of one node is diffused over its neighbors. After a sparsifying the input adjacency matrix and normalizing it, we tested a lot of configurations with regards to graph size, node features and network configuration to predict the h indices. However, we couldn't converge to a loss below 4 on our testing dataset, so we did not continue with this approaches.

4.3 Multi Layer Perceptron

Because we were still achieving correct results with Graph Neural Networks, we decided to try simpler networks such as MLPs for out task. We started simply with a single hidden layer with 16 neurons, taking as input the abstracts embeddings, the node embeddings and hand-crafted graph features and outperformed our Kaggle leaderboard score, so we decided to dig deeper into this architecture and the choice of parameters used. The features we gaved to the network were the following: a concatenation of the Doc2Vec embeddings of authors' abstracts, the embeddings learned with SciBERT and the graph structural features, as shown in figure 6

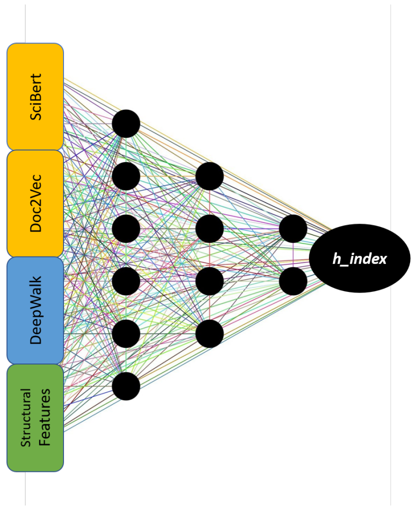

Figure 6: The Multi Layer Perceptron network

4.4 Fine-tuning

For the fine-tuning we started with the use a RandomizedSearchCV and GridSearchCV, but the former was taking steps randomly in the search space, and the latter was difficult to converge due to the the parameter combinatorial complexity. We thus opted for an hyperparameter optimization called Optuna [1], which allow for to construct the parameter search space dynamically, and having a more efficient implementation of both searching and pruning strategies for this space, based on bayesian optimization. We choose to search on the following parameters for Gradient Boosted Models:

1. n_estimators: It is the number of trees (or rounds) in an XGBoost model.
2. max_depth: Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.
3. colsample_bytree: It is the subsample ratio of columns when constructing each tree.
4. eta: Step size shrinkage used in update to prevent overfitting.
5. gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree.

We tried stacking several of this learners with a final meta-learner but it wasn't better than the performance of the MLP.

And we searched on these parameters for the MLP, which helped us to find the best parameters for out top Kaggle submission:

1. n_layers:This is controlling how deep our network is
2. n_units: The number of hidden units varying for all the hidden layers.
3. dropout and batchnorm: the value of the dropout between 0.2 and 0.7 and whether to use batchnorm or not to tackle overfitting issues.
4. Optimizer parameters: What optimizer to use (Adam or SGD), learning rate and weight decay parameters.

We tried ensembling for the MLP but the loss wasn't better than a single fine tuned MLP. One last thing that we applied to fine tune our predictions a post-hoc correction step between the distribution of the predicted h_index for the whole testing set and the ones coming from the train set. This idea cam from the observation of predictions distribution between our 2 best models on the leaderboard and the distribution of the train h-indices (in green) 7

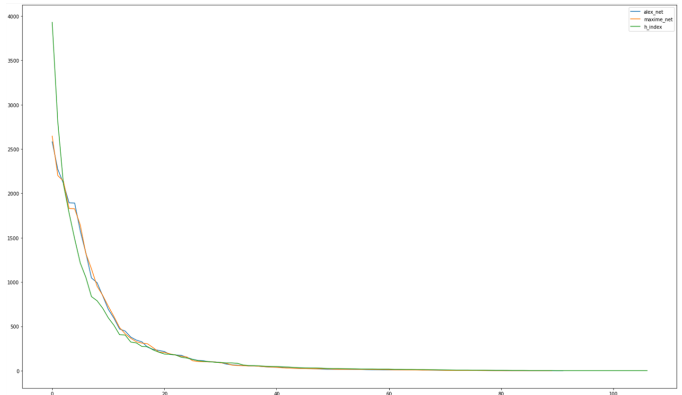

Figure 7: H index distribution for the 2 best models compared to the ones from the training set

We observed a spike around 5 of h index, so we decided to apply a smoothing strategy. We iterated through the predicted h_indices from lowest count to highest count, and when the count was higher in the training set than what we predicted, we lowered by 1 the h_index of authors with the least number of papers and degree (important features found with feature importance methods).

5 Results

5.1 Best Embedding

We tried different configurations for all the methods of embedding., and we used the UMAP [9] algorithm to reduce dimensionality when the computing time of embeddings was too long (ex: SciBERT). We plugged the embeddings on a checkpoints of models to compare them. As seen in the next table, the loss of all the tests decrease a lot while applying a SciBERT embedding with dimension 768.The same comparison is applicable on node embeddings: the Deep Walk method exceeds the scores of the other node embedding methods.

Node emb.	dim	Abst. emb.	dim	loss
GAE	32	Scibert	128	3.98
GAE	64	Scibert	128	4.03
DeepWalk	128	Scibert	128	3.87
DeepWalk	**128**	**Scibert**	**768**	**3.69**
DeepWalk	64	Scibert	128	3.84
DeepWalk	64	Scibert	768	3.70
Node2Vec	128	Scibert	128	4.03
Node2Vec	128	Scibert	768	3.82
LINE	128	Scibert	128	3.99

Then while fixing the node embedding size, we tested to concatenate the Doc2Vec Abstract embedding, which ended up adding useful information, and SciBERT 768 was the best size.

Embedding	dim	loss
Doc2Vec(128) + Scibert	128	3.42
Doc2Vec(128) + Scibert	768	**3.37**

5.2 Best Model

We tested different approaches to predict the final h_index.

Model	Fine Tuned	Loss	
		Local	Leaderbord
XGboost	False	4.01	3.8
XGboost	True	3.87	3.62
Stacking XGbosst	False	3.95	3.82
MLP	False	3.93	3.75
MLP	True	**3.19**	**3.25**
Ensembling MLP	True	3.41	3.45

We can see here that one MLP model is considered to be the best. In the next table, we compare the different number of layers of the MLP model, on 2000 epochs, each with the local MAE loss.

Nb. layer	Local Loss	Leaderbord Loss
1	**3.19**	**3.25**
2	3.26	3.30
3	3.25	3.43

We concluded that an simple neural network model can performer better alone without adding multiple layers and ensembling, as they said *"the simpler the better" (Occam's razor).*

5.3 Our Final Submissions

Therfore, In our best submission, we choose to take the following:

- *Node embedding* : Deep walk with a walk length of 30 and 100 number of walk for each node with the dimension of 256.
- *Abstract embedding*: the concatenation of both 128 Doc2Vec and 786 Sci-bert fine-tuned embeddings.
- *Predictive Model* : MLP model with 1 hidden layer of size 1082 to 495, with dropouts of around 0.35, Batchnorm1D, Relu activations and an Adam optimizer with 0.0007 learning rate (all were find with optuna and corrected by hand).

To train the model for the final prediction, we took on all the training data without splitting and then predicted on the test set, with a final correction method described above: it gave us **3.23734** in the public leaderboard and **3.26174** on the private leaderboard.

6 Future work

There is still room for improvement on different axes. Scraping more data about the authors from Google scholar, in order to find deeper relationships between the h indices and their caracteristics, would have been a nice out-of-the box improvement for our model. Another area of interest was investigating more the border effects of our predictions, maybe the predictions would have been more stable if we would have applied a log transformation on the h_indices, given their skewed distribution.

Another huge room for improvement is the use of proper graph neural networks for the task, we strongly believe that an adapted version of GCN-LPA on smaller graphs of authors, created on features from their abstracts, would add much diversity to the model. Processing the adjacency matrix in sparse batches and fine tuning such a GNN would align with the fact that authors are well represented locally rather that on the whole global graph. This makes sense because authors that are linked to another author 200 nodes away are unlikely to generate co-information that is as valuable as authors that are a few nodes away.

Another dimension of improvement is purely on the performance with ensembling, blending and stacking models between gradient boosted and deep learning techniques. As it was our first Kaggle competition ever, we aren't aware of the best practices and a lot of improvement might come from this.

As a whole, we are quite satisfied with our results. We trained and tested several different strategies on both NLP state of the art models, with a fine grained work on SciBERT embeddings, and we got our hands dirty with structural graph techniques and testing deep learning methods applied to graphs, which represents most of the techniques learned during these 3 months, but we were able to applied them on a real problem.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pages 2623–2631, 2019.

[2] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. arXiv preprint arXiv:1903.10676, 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[4] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363, 2018.

[5] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428, 2019.

[6] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864, 2016.

[7] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30:3146–3154, 2017.

[8] Thomas N Kipf and Max Welling. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308, 2016.

[9] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:1802.03426, 2018.

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 701–710, 2014.

[11] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In Proceedings of the 24th international conference on world wide web, pages 1067–1077, 2015.

[12] Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. arXiv preprint arXiv:2002.06755, 2020.