INF554 Machine and Deep Learning
Data Challenge: H-index Prediction

AHAW team: Alexandre Hirsch, Antonin Wattel
alexandre.hirsch@polytechnique.edu, antonin.wattel@polytechnique.edu
Ecole Polytechnique

1 Introduction

The goal of this project is to correctly estimate the h-index of an author given the ID of his most cited papers, the abstracts of those papers, and the IDs of the authors he has co-authored a paper with.
In our project, we extracted text features from the abstracts, graph centrality features and node embeddings from the graph generated by the co-authorship relationships, and trained them with a Bagging Regressor of an MLP regressor. The source code from this report can be obtained from https://github.com/antonin-wattel/H-index-prediction

2 Method

2.1 Graph features

We have a fairly big graph, consisting of 217,801 vertices and 1,718,164 edges. Our goal is to extract features for each node, corresponding to each author.

Centrality features
We first extract centrality based features for each node. Roughly speaking, centrality features try to characterize the 'importance' of nodes, where importance can have several meanings depending on the centrality measure. We use the Networkx library, which provides several built-in methods to compute centrality measures of nodes. We compute the following features:

- degree: number of edges connected to the node.
- core number: largest value k of a k-core containing that node, where a k-core is the maximal subgraph with minimum degree at least k.

The graph being large, most centrality measures (closeness centrality, katz centrality...) could not be computed in a reasonable time, or did not improve our results (eigenvalue centrality).

Node embeddings
The idea of node embeddings is to map vertices of a graph into a low-dimensional space, such that similar nodes are embedded close to each other in this space. There are different types of methods that can be used to compute node embeddings. As seen in the lecture, they can be categorized as follows.

1. Random walk based methods: employ random walks to capture structural relationships between nodes
2. Edge modeling methods: directly learn node embeddings using structural information from the graph
3. Matrix factorization methods: generate a matrix that represents the relationships between vertices and use matrix factorization to obtain embeddings.
4. Deep learning methods: apply deep learning techniques to learn highly non-linear node representations

The library karateclub [17] provides implementation for a great variety of node embedding techniques. We computed embeddings with most of the techniques the library proposes. We choose default parameters to compute most of them for time and practical reasons. We chose from the following:

- Random walk based methods: *Node2Vec* [6], *Deepwalk*, [4], *Diff2Vec* [18], *Role2Vec* [10], *Walklets* [9],
- Matrix factorization methods: *BoostNE* [12], *NetMF* [13], *RandNE* [14], *NMF-ADMM* [5], *LaplacianEigenmaps* [1], *HOPE* 3 [7], *Sociodim* [8]

Some of these embeddings required quite some time to compute, making it tedious to properly tune the parameters for all of them. We first used default parameters of the library in each case.

The first step was to test these embeddings on our data, with different regressors, as shown in Table 3. The best performance was achieved by the Walklets algorithm, using an MLP regressor. We then decided to compute other node embeddings using this algorithm, with different parameters. The best accuracy was achieved using ($walknumber = 20$, $walklength = 120$, $dimensions = 100$). Considering the fact that computing these embedding was relatively long (23 minutes for default Walklets, 241 minutes for Walklets (2)), we could not perform a rigorous parameter tuning.

2.2 Text features

We have a 624,181 total abstracts and 217,801 authors with between 1 and 5 papers credited to their name. Our goal is to extract and calculate text features for each author.

Choosing the model

Since we were given the set of words that appear in each abstracts in an inverted index, there were two main ways we could have proceeded.

1. Reconstruct the abstract from the inverted index and calculate its embedding using state-of-the-art models, such as BERT [11], BART [15], GPT-3 [16], or simply Doc2Vec [3].
2. Extract the important words from the inverted index and calculate an overall embedding on those words using a state-of-the-art word embedding model known as Word2Vec [2], developed by Google in 2013.

We chose the second method, because a variety of pre-trained models were available. We experimented with some of them: Google News, Glove Wiki Gigaword and a Word2Vec model trained on the abstracts. Ultimately, we chose Google News because it performed the best, as shown in Table 2.

Preprocessing

Following this method, we chose to limit the amount of memory used to store the abstracts to the strict necessary by doing the following:

- We use a dictionary mapping each paper_ID to the set of its inverted index dictionary's keys which means that we disregard the number of times a word appears and its position in the text. We experimented with weighing a word's importance by the number of times it appeared in the text but it was not as effective as counting it only once.
- We remove stopwords, which are words that do not hold a significant semantic meaning over the whole sentence, such as "the", "it", "that", or "is" among many others. The importance of their presence is shown in Figure 1

Word2Vec

After preprocessing the words of the abstracts, and keeping the most important content, we can apply the Word2Vec model on them.

Word2Vec is a neural model that associates a vector to common words, such that the cosine similarity between the vectors of two words represents its semantic similarity. This model is known for the following formula which explains the relation between its vectors:

$$V(\text{“}king\text{”}) - V(\text{“}man\text{”}) + V(\text{“}woman\text{”}) = V(\text{“}queen\text{”})$$

As such, we can see that this model preserves logical meaning under mathematical additions and subtractions and that the model holds the meaning of a sentence when adding the vectors of each of its words together. This is the "Bag-of-Words" use of Word2Vec.

Calculating the text features

Thus, we can roughly attribute the value of an abstract as the sum of each of its words' vectors, which is summarized in the following formula:

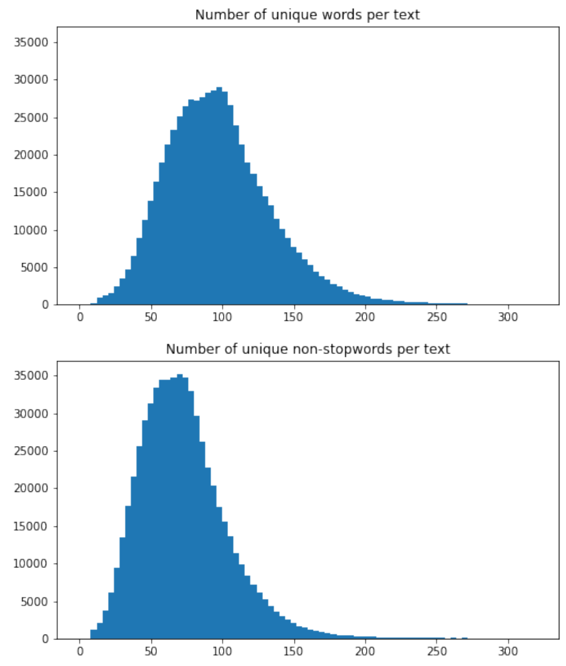

Figure 1: Presence of stop-words in text

$$V(paper) = \sum_{\substack{word \in paper \\ word \notin stopwords}} V(word)$$

We are given the unordered list of the top 5 most cited papers of each author. Using the text embeddings of each paper, we can combine them to get the value of the author's overall text features. Since there can be less than 5 papers in this list, we represent text features by averaging the text representations of the papers, and since the list is unordered, we cannot attribute weights to the papers. This is summarized in the following formula:

$$V(author) = \frac{1}{\#author\_papers} \sum_{paper \in author\_papers} V(paper)$$

2.3 Combining features

Now that we have features from the abstracts and the graph, we want to combine them. We chose to simply concatenate the different features, after normalizing them. To avoid working with too high dimensions, we only choose to add the node embeddings from Walklet [9], that yielded the best results among all node embedding methods used alone.

Another method consists in combining node embeddings in a smarter way, for example by concatenating them and performing dimensionality reduction. We tried this method using PCA. Other dimensionality reduction techniques implemented in scikit learn, such as gaussian random projection were not applicable due to memory allocation issues.

We test these combinations with different estimators, as shown in Table 4. As expected, the combination yielding the best results is when taking all features (text features + graph centrality features + node embedding features).

2.4 Regression

In order to avoid overfitting and have an idea of the loss resulting from our models, we split the given training set into a training set and a validation set, with respective proportions of 0.8 and 0.2.

Models

To make predictions, We tried different regression models available on scikit learn (MLP, Lasso, ARD, BayesianRidge, KNN). We tried these regressors for different word embedding models, different node embedding models, and several combination of features. The MLP with one hidden layer always outperformed the other estimators, even though it generally demanded more time. Depending on the combination of features used and the number of dimensions of the features, we adjusted the number of neurons in the hidden layer to get the best results with a reasonable training time. To reduce computation time and avoid overfitting, we used validation-based early stopping.

Bagging

To improve our results, we use an ensemble method, i.e by combing several models. We did not try to use stacking, as the MLP was the best estimator everywhere, and we did not need several heterogeneous weak learners. We tried AdaBoost, and it did not improve our results. Bagging was an appropriate method to use here.
Bootstrap aggregating or Bagging, is the process of dividing a dataset into multiple subdatasets drawn uniformly and with replacement, and fitting each of them to a regression model, and combining them by averaging the output. We use bagging with the combination of features and estimator that yielded the best result, that is the concatenation of text features, simple graph features, and Walklets node embeddings, with an MLP estimator that gave best results when used alone. The best results were obtained when increasing the number of estimators in the bagging algorithm.

3 Results

Our data is post-processed by zeroing negative values, as they are technically impossible.

Our final result is based on varying the number of estimators in a Bagging regressor with an MLP base estimator with a hidden layer of 200 dimensions and validation-based early stopping. The features used are the text features calculated with the Google News pre-trained model, the centrality graph features and the best-performing node embedding (Walklets 2). Our results are found in Table 1.

Estimators	Bagging MSE	Runtime
8	50.86616	16 mins
16	49.10514	49 mins
48	48.01164	120 mins

Table 1: Bagging results according to number of estimators

4 Conclusion

Our final model was decided after a lot of thorough experimentation on each step of the pipeline: feature extraction and selection, choice of model and its parameters, and how to apply Boosting or Bagging to our best models. It allowed us to reach an error of 48.012.

5 Discussion and improvements

In this section we discuss possible improvements to our methods.

5.1 Improving text features

Word position, frequency, and importance

Even though we found that in general multiplying a word's embedding by the number of times it appears in the text wasn't effective, we could experiment with doing so on specific words that greatly inform the subject of the paper, such as "clustering", "supervised", or "biosequence". The position of a word in the abstract may also inform its overall importance, however we did not find an intuitive way to find this importance.

RNNs

By just doing a sum of word embeddings, we lose the sequential nature of the text data. To improve this we could have used a recurrent neural network architecture such as an LSTM.

Translation

Most abstracts were written in English, but a few were written in other languages, such as French, German or Mandarin. We attempted to solve this problem in the following way:

Obviously, these words were not recognized by the Word2Vec model, so they were translated to English. Before being translated, the language had to be identified. This was done by detecting the language of the first few words in the abstract, rather than all of them, to save computation time. Once the language was identified, we couldn't just translate the text word by word, as some semantic meaning would've been lost, so we reconstructed the whole abstract, translated it, and extracted the set of individual non-stopwords in the resulting text.

This functionality couldn't be implemented because most language detection and translation libraries, even open-source ones, have a limit of translation requests per month that were quickly exhausted when testing the functions that used them. We have left the code for this functionality in for consideration, as we are sure that it would have increased overall accuracy.

5.2 Deep learning for node embeddings

To compute node embeddings, we only used unsupervised methods, not based on deep learning. The use of deep learning methods on the graph such as GCA could have probably improved our node embedding and yielded better results.

References

[1] Mikhail Belkin and Partha Niyogi. “Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering”. In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper/2001/file/f106b7f99d2cb30c3db1c3cc0fde9ccb-Paper.pdf.

[2] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[3] Quoc V. Le and Tomás Mikolov. “Distributed Representations of Sentences and Documents”. In: *CoRR* abs/1405.4053 (2014). arXiv: 1405.4053. URL: http://arxiv.org/abs/1405.4053.

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (Aug. 2014). DOI: 10.1145/2623330.2623732. URL: http://dx.doi.org/10.1145/2623330.2623732.

[5] Dennis L. Sun and Cédric Févotte. “Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 6201–6205. DOI: 10.1109/ICASSP.2014.6854796.

[6] Aditya Grover and Jure Leskovec. “node2vec: Scalable Feature Learning for Networks”. In: *CoRR* abs/1607.00653 (2016). arXiv: 1607.00653. URL: http://arxiv.org/abs/1607.00653.

[7] Mingdong Ou et al. “Asymmetric Transitivity Preserving Graph Embedding”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1105–1114. ISBN: 9781450342322. DOI: 10.1145/2939672.2939751. URL: https://doi.org/10.1145/2939672.2939751.

[8] Mingdong Ou et al. “Asymmetric Transitivity Preserving Graph Embedding”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1105–1114. ISBN: 9781450342322. DOI: 10.1145/2939672.2939751. URL: https://doi.org/10.1145/2939672.2939751.

[9] Bryan Perozzi et al. *Don’t Walk, Skip! Online Learning of Multi-scale Network Embeddings*. 2017. arXiv: 1605.02115 [cs.SI].

[10] Nesreen K. Ahmed et al. *Learning Role-based Graph Embeddings*. 2018. arXiv: 1802.02896 [stat.ML].

[11] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[12] Jundong Li, Liang Wu, and Huan Liu. “Multi-Level Network Embedding with Boosted Low-Rank Matrix Approximation”. In: *CoRR* abs/1808.08627 (2018). arXiv: 1808.08627. URL: http://arxiv.org/abs/1808.08627.

[13] Jiezhong Qiu et al. “Network Embedding as Matrix Factorization”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (Feb. 2018). DOI: 10.1145/3159652.3159706. URL: http://dx.doi.org/10.1145/3159652.3159706.

[14] Ziwei Zhang et al. *Billion-scale Network Embedding with Iterative Random Projection*. 2018. arXiv: 1805.02396 [cs.SI].

[15] Mike Lewis et al. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *CoRR* abs/1910.13461 (2019). arXiv: 1910.13461. URL: http://arxiv.org/abs/1910.13461.

[16] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[17] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. “Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs”. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM ’20)*. ACM. 2020, pp. 3125–3132.

[18] Benedek Rozemberczki and Rik Sarkar. *Fast Sequence-Based Embedding with Diffusion Graphs*. 2020. arXiv: 2001.07463 [cs.LG].

Appendix A Results

	MLP	Lasso	ARD	BayesianRidge	KNN
Glove-wiki-gigaword-300d	84.26	120.85	109	108.97	98.22
Word2vec-google-news-300d	82.13	123.44	108.87	108.85	98.35
Conceptnet-numberbatch-17-06-300d	160.34	160.34	160.34	160.34	175.78
Word2vec-trained-on-abstracts-100d	84.92	116.03	114.87	114.81	93.22

Table 2: Mean Squared Error of different regression models using different word embedding models

	MLP	Lasso	ARD	BayesianRidge	KNN
Node2Vec	118.84	162.29	145.59	145.59	338.6
BoostNE	160.52	160.65	160.55	160.54	135.5
NetMF	120.23	133.45	128.41	128.41	116.14
RandNE	120.24	159.22	158.96	158.91	568.73
Deepwalk	114.34	154.55	139.41	139.4	320.74
Diff2Vec	122.9	160.03	157.03	157	139.15
Role2Vec	109.7	156.36	144.01	143.96	127.4
Walklets	91.58	147.92	117.16	117.16	109.89
Walkets (2)	88.3	153.47	112.14	112.12	110.64
Walklets (3)	86.48	159.73	112.51	112.45	110.77
NEU + Walklets	102.04	156.45	131.46	131.46	112.16
NMFADMM	146.22	159.03	151.11	159.03	167.34
Laplacian Eigenmaps	105.29	156.53	110.45	110.44	97.55
HOPE	130.11	146.7	133.92	133.92	123.67
concatenation + pca (n=150)	104.15	160.90	123.13	123.10	129.49

Table 3: Mean Squared Error of different regression models using only node embeddings

	MLP	Lasso	ARD	BayesianRidge	KNN
T	77.23	115.93	101.02	100.96	93.36
G	109.97	131.09	116.18	131.02	108.43
NE	88.42	157.71	114.74	114.73	113.94
T+G	58.37	107.15	96.17	96.16	74.24
T+NE	57.14	118.6	82.06	82.03	84.49
G+NE	83.4	123.91	100.95	100.94	92.71
T+G+NE	56.71	107.63	80.9	80.89	73.43

Table 4: Mean Squared Error of different feature combinations using different regression models (T : text features, G : simple graph features, NE : Walklets node embeddings)