Predicting the *h*-index of Authors using multi-faceted data and limited supervision

Souaidi Omar[1]*, **Riou Auriane**[2] and **Abbahaddou Yassine**[3]
[1]CentraleSupelec / MVA ENS
[2]ENPC / MVA ENS
[3]CentraleSupelec / MVA ENS
omar.souaidi@student.ecp.fr, auriane.riou@eleves.enpc.fr, yassine.abbahaddou@student.ecp.fr

Abstract

Graphs can be represented as documents, and documents can be represented as graphs, so methods from graph theory can be applied to NLP and inversely. In this paper, we study a real-world regression problem by utilizing the combination of these two modalities.

1 Introduction

The objective of this data challenge is to build a model capable of predicting the **h-index** of authors. The dataset we were supplied with contains $231,239$ authors, each of which has up to 1,500 papers. The data is non-Euclidian and represented in 2 type : **graph** and **text**. The given graph indicates whether two scientists have co-authored any papers while the textual data was the abstracts of the top-cited papers of each author.

The key contributions of this work are as follows : ***i.*** Create the authors embedding using the structural embedding technique **Node2Vec** ***ii.*** Use the Transformer based biderctional pre-trained model for abstract embeddings **RoBERTa** and compare it with a more simple approach **Doc2vec**. ***iii.*** Use model relying on the attention mechanism that combine graph and text embeddings.***vi.*** Compare the two approaches based on thier performance.

2 Methodology

3 Abstract Embedding

We have the abstracts of the top-cited papers of each author. As we can not feed the abstracts directly to a Machine Learning/Deep Learning model, we need first to convert the abstract to vectors of numbers.
Let $A = (A_1, A_2, ..., A_n)$ be the set of authors (n is the number of authors) and $f(A_i)$ a function that assigns to each author the list of the corresponded abstracts.
For i in $[1 : n]$ we have that $f(A_i) = (X_i^1, X_i^2, ..., X_i^k)$ where X_i^j corresponds to the j th abstract of the author. k can be in range $[0 : 10]$ where $k = 0$ means that the author has no abstracts.
Let $G(X)$ be the function that maps one abstract X to a dense vector of numbers (the embedding).
We come up with two different functions $G(X)$.

Sentence Transformers
Sentence Transformers [Reimers and Gurevych, 2019] is a framework that provides an easy method to compute dense vector representations for sentences and paragraphs (also known as sentence embeddings). The models are based on transformer networks like BERT / RoBERTa / XLM-RoBERTa etc. and are tuned specificially meaningul sentence embeddings such that sentences with similar meanings are close in vector space. To embed each abstract into a dense vector, we used the RoBERTa Base model optimized for Semantic Textual Similarity (STS).
We will denote by $G_{ST}(X)$ the dense vector obtain by passing one abstract X to the sentence transformers model.

Doc2Vec
Doc2Vec [Le and Mikolov, 2014] is a Model that represents each Document as a Vector. We train a Doc2Vec using the Gensim Library [Řehůřek and Sojka, 2010] on the whole corpus of text (all the abstracts from the training and validation set).
We will denote by $G_{DV}(X)$ the dense vector obtain by passing one obstract X to the Doc2Vec.

Now for each author A_i, we associated the vectors : $G(f(A_i)) = (G(X_i^1), G(X_i^2), ..., G(X_i^k))$.
If $k < 10$, we complete by adding vectors of zeros that have the same embedding dimension until $k = 10$.

4 Graph Embedding

With the lack of authors features, we used the embedding techniques that relies only on the structure of the network. Several techniques can be used, but the use of most of them is limited to small graphs due to the complexity and resources. The challenge in learning embeddings is that the graph is big :it consists of $231,239$ vertices and $1,777,338$ edges in total. At this large scale, we can use the **Node2vec** algorithm.

Node2Vec
The node2vec algorithm is a Deep Learning based model inspired from the **Word2vec** algorithm: It takes a list of sentences as input and produces a vector or an embedding for

*Contact Author

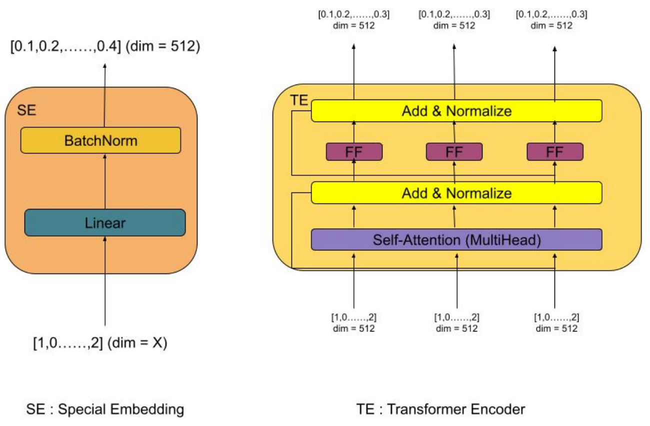


Figure 1: Special Embedding and Encoder Architecture

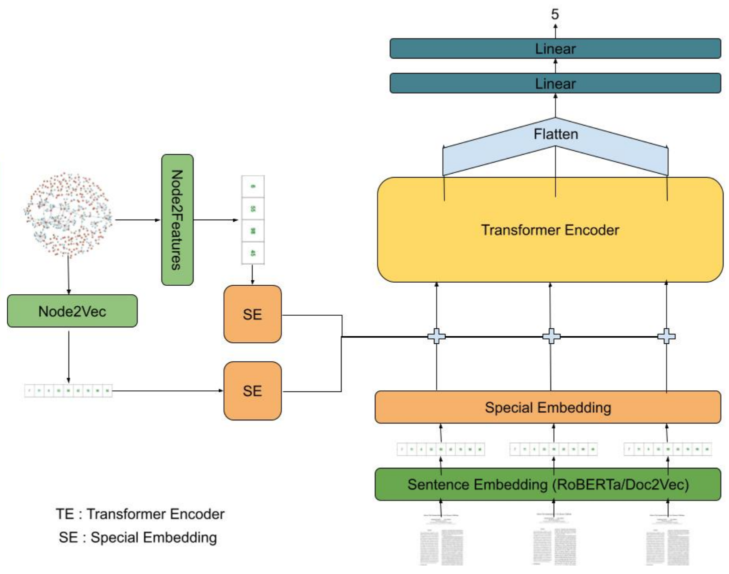


Figure 2: Our Model Architecture

each word that appears in the text corpus. Instead of using a list of sentences as input, we use a list of random walks. We generate the embedding of dimension 128 using $4,624,780 = 20 \times 231,239$ walks (20 walks sarting from each node).
Now for each author A_i, we associated the vectors : $NE(A_i)$ which represent the embedding of the node (obtained by our Node2Vec model) corresponding to that author.

Node2Features
As a node embedding technique, **node2vec** has a limitation : it disregards important structural properties about nodes like the node degree, the core number and the average neighbor degree degree. In one approach, we combined the node2vec features with these information. With the additional textual data, we also added the number of top cited papers.
Now for each author A_i, we associated the vectors : $NF(A_i)$ which represent the additional information about the node corresponding to that author.

5 Approaches

5.1 Transformer Encoder Based Model

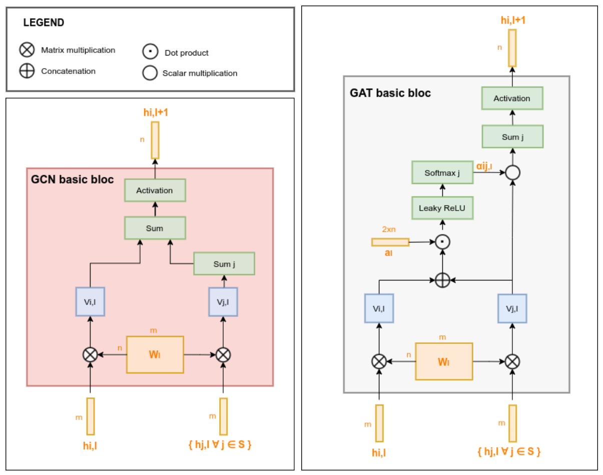
Before presenting our model architecture, we will first introduce some neural network blocks as we can see if Figure 1.
The first one is SE (Special Embedding) that take as input a vector of dimension X and return a vector of dimension 512. It composed by a Linear layer that embed the input vector into a vector of dimension 512 and followed by a batch normalisation. We will denote by $SE(V)$ the function that represent the Special Embedding (V is a vector).
The second one is the encoder part of the Transformer architecture [Vaswani *et al.*, 2017].

Our Model Architecture is presented in Figure 2.
To explain how our model works, we are going to take a certain author A_j. First we retrieve the node embedding $NE_{A_j} = NE(A_j)$ and the node features $NF_{A_j} = NF(A_j)$, we do the same thing for the abstract embedding $AE_{A_j} = G(f(A_j)) = (G(X_j^1), G(X_j^2), ..., G(X_j^10))$ with the corresponding masks MAE_{A_j}. We pass each one of them to a Special Embedding Layer.
Finally we obtain two vectors of shape (512) :
$SE(NE_{A_j})$ and $SE(NF_{A_j})$
One vector of shape (10,512) :
$SE(AE_{A_j})$.
For this latter, we add in each coordinates the vector
$N_{A_j} = SE(NE_{A_j}) + SE(NF_{A_j})$.
At the end, we obtain the vector :
$V_{A_j} = (G(X_j^1) + N_{A_j}, G(X_j^2) + N_{A_j}, ..., G(X_j^10) + N_{A_j})$
We feed this vector the Transformer Encoder, we flatten the output to obtain a vector of dimension (5120), we feed it to a Linear layer followed by a ReLU Activation and dropout, and then by a Linear layer that output the predicted h-index $\hat{\mathbf{y}}$
The parameters of the Transformer Encoder :

- Number of heads : 8
- Number of Encoder Layer : 2
- Encoder Dropout : 0.1

5.2 Graph Neural Networks

We then decided to try another approach based on Graph Neural Networks (GNNs). Indeed, GNNs constitute an effective framework for diverse machine learning tasks on graph data structures. We decided to implement two GNN variants: a Graph Convolutional Network (GCN) as well as a Graph Attention Network (GAT). Both structures are based on a neighborhood aggregation scheme. We aimed at measuring how well these two architectures would perform both for a supervised regression task (by trying to directly predict an h-index value for each node), as well as for a node embedding task.



(a) Basic GCN layer (b) Basic GAT layer

Figure 3: Diagram of elementary blocs for the GCN and GAT architectures

Graph Convolutional Network

Graph convolutional Networks (GCNs) were introduced in [Kipf and Welling, 2017]. This structure is inspired from convolutional layers that are commonly used in Convolutional Neural Networks (CNNs). The major difference between CNNs and GCNs lies in the data structure. Indeed, GCNs can be seen as a generalized version of CNN that can work on data with underlying non-regular structures. More precisely, the convolution operation on a graph consists of computing the representation vector of a node by recursively aggregating and transforming the representation vectors of its neighboring nodes. The layer-wise propagation rule is defined by:

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^lW^l)$$

Where

- $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix
- σ is an activation function
- H^l is the matrix of activations in the l^{th} layer
- W^l is a trainable weight matrix

This vector form equation can can be re-written under the following form, for a specific node i:

$$h_i^{l+1} = \sigma(\sum_{j\in N(i)} \frac{1}{c_{ij}} W^l h_j^l)$$

Where $N(i)$ is the set of one-hop neighbors of a node and i, $c_{ij} = \sqrt{|N(i)|}\sqrt{|N(j)|}$ is a normalization factor based on graph structure. We summarize this aggregation mechanism Figure 3a.
Although GCNs can very efficient for diverse machine learning tasks, the statically normalized convolution operation is structure-dependent, which can hurt its generalizability.
In order to face this issue, we decided to implement another architecture based on attention, using a GAT. The key difference lies in the way the information is aggregated from the one-hop neighbourhood. For GCNs, the convolution operation produces the normalized sum of the node features of neighbours. GATs introduce instead an attention mechanism in order to remove dependency on graph structure.

Graph Attention Network

GATs were introduced in [Veličković *et al.*, 2017]. They rely on an attention mechanism used to aggregate the neighborhood features by specifying different importance values to different neighbors. More specifically, the attention layer can be broken down in the following four main operations:

- **Simple linear transformation:** It consists of transforming the input features into higher level ones. This is performed using a shared linear transformation, parameterized by a weight matrix W^l:

  $$v_i^l = W^l h_i^l$$

- **Computing attention coefficients**: We then compute a pairwise unnormalized attention score between two neighbours. To do so, the two embeddings of the neighbours are concatenated. We then take the dot product of this vector with a learnable weight vector a^l, and apply a LeakyReLU to the result:

  $$e_{ij}^l = LeakyReLU(a^T(v_i^l||v_j^l))$$

  .

- **Softmax operation**: To compare the computed coefficients across different nodes, we normalize them across all choices of j using a softmax function:

  $$\alpha_{ij}^l = \frac{\exp e_{ij}^l}{\sum_{k\in N(i)} \exp e_{ik}^l}$$

- **Aggregation**: This step is similar to GCN. The embeddings from neighbours are aggregated together, scaled by the attention scores:

  $$h_i^{l+1} = \sigma(\sum_{j\in N(i)} \alpha_{ij}^l v_j^l)$$

We summarize this attention mechanism Figure 3b. We also use multi-head attention to enrich the model's capacity as well as to increase stability in the learning process.

6 Training

In the context of this challenge, the performance of our model will be assessed using the *mean absolute error* (MAE). This metric is defined as the arithmetic average of the absolute errors.

$$MAE = \frac{1}{N}\sum_i^N |\hat{y}_i - y_i| \tag{1}$$

Where N is the number of samples (i.e., authors), y_i is the true h-index of the i^{th} author and $\hat{y}_i$ is the predicted h-index of that author.

6.1 Transformer Encoder Based Model

The model performance is evaluated by the MAE value. However, this function is not smooth so we cannot guarantee smooth derivatives. The loss that we considered instead for the training is the Mean Squared Loss which is a fairly good approximation of the MAE:

$$MSE = \frac{1}{N} \sum_{i}^{N} |\hat{y}_i - y_i|^2 \tag{2}$$

We updated the model weights using ADAM Optimizer. We choosed the following hyperparameters :

- starting learning rate : 0.0007
- batch batch size : 64
- epochs : 60

As models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates, we use the module ReduceLROnPlateau to reduce learning rate when the metric (MAE) on the validation set has stopped improving. We choose as parameters :

- patience : 1 epoch
- factor : 0.8

Each 150 iterations (correspond approximately to half an epoch), we evaluate our model on the validation set and we save the one that performed best on the validation set.

Ensemble Model

We notice that our model tend to overfit (a MAE difference of 1.5 between the training set and the validation set), we decided to go with an ensemble models.
The goal of ensemble model is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability/robustness over a single estimator.
To build several estimators, we use the same model (and same hyperparameters) but we use five different training/validation split. At the end we had 5 models for each abstract embedding method (Sentence Transformers and Doc2Vec), a total of 10 models.
If we denote by $\hat{y}_i$ the prediction given by the i^{th} model, then the final prediction $\hat{y}$ is given by :

$$\hat{y} = \frac{1}{N} \sum_{i}^{N} \hat{y}_i \tag{3}$$

Here N=10

6.2 Graph Neural Networks

For our Graph Neural Networks we also used a Mean Squared Loss and ADAM Optimizer. We used as well ReLU activations for both models.

We tried several architecture designs and sets of hyperparameters. Figure 4 shows one example of an overall architecture for our GCN consisting of 3 hidden layers.

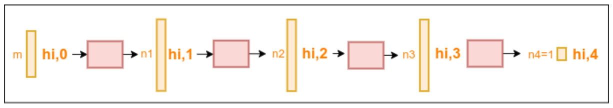


Figure 4: Example of global GCN architecture (built upon basic bloc from figure 3a)

Figure 5 shows an example of our GAT architecture consisting of three layers. The first two layers consist of $K = 3$ attention heads with an output dimension of n_1 and n_2 features for each respectively (for a total of $n_1 \times 3$ and $n_2 \times 3$ features as we use concatenation between the different heads). The final layer is used for regression: it consists of $K = 6$ attention heads, computing 1 value each, that are averaged and followed by our ReLU activation function.

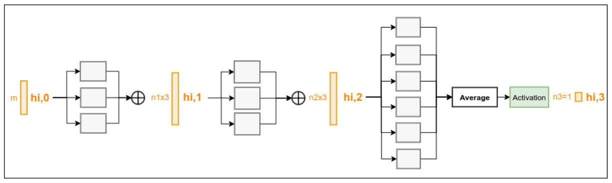


Figure 5: Example of global GAT architecture (built upon basic bloc from figure 3b)

We tried to evaluate the models for both their capacity to directly predict an h-index value for each node (by using author embeddings as initial node features), as well as to generate embeddings for nodes (using one-hot embeddings as initial features). Unfortunately, in both cases we were not able to achieve satisfying results as these models require a lot of memory. We were therefore limited in the depths of the networks as well as regarding the dimensions of intermediate aggregated features which hindered us for achieving good performances. Hence we did not make any submissions with this approach.

7 Results

Table 1: TEBM refers to the Transformer Encoder Based Model. D2V refers to the ensemble model (5 models) made with the abstract embedding by Doc2Vec and ST to the one by the SentenceTransformer framework. ST + D2V refers to the final model that combines both of them.

Name	Public Set	Private Set
TEBM D2V	3.65	3.69
TEBM ST	3.61	3.65
TEBM ST+D2V	3.46	3.50

8 Conclusion

In this project, we looked at how to predict h-index of authors. We mainly focused on the attention mechanism in both the embedding techniques (RoBERTa) and the model itself (transformer encoder blocks & GAT). For the nodes embedding, we limited ourselves to Node2Vec due to the large scale graph. Models are evaluated by the MAE loss but we train them using the smooth loss function MSE; this gives more stable results. Performance varies across models and embedding techniques, but but they have all demonstrated the effectiveness of the attention mechanism.

9 Further Work

We think that the thing that limits most the model performance is the embedding. We could try to have a better text cleaning (we notice at the end that we wanted to filter some special character on the abstract, we also filtered some words), but most importantly a better node embedding.
Another approach will be to make an end-to-end training and can be based on [Saha *et al.*, 2020] where they jointly use multi modal data (text, image, video).

References

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[Le and Mikolov, 2014] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[Řehůřek and Sojka, 2010] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[Reimers and Gurevych, 2019] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[Saha *et al.*, 2020] Tulika Saha, Aditya Patra, Sriparna Saha, and Pushpak Bhattacharyya. Towards emotion-aided multi-modal dialogue act classification. 07 2020.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.