Link prediction on citation networks
Final Report : Machine Learning for Network Science
Team Name: Student Embeddings

Oussama Boussif
CentraleSupélec
oussama.boussif@student.ecp.fr

Omar Souaidi
CentraleSupélec
omar.souaidi@student.ecp.fr

Léna Ezzine
CentraleSupélec
lena.ezzine@student.ecp.fr

Théo Moutakanni
CentraleSupélec
theo.moutakanni@student.ecp.fr

Abstract

*Link prediction is a machine learning task where a model has to predict whether an edge is missing or would form in the future between two particular nodes. In this report we leverage features from the graph structure and from the node attributes in order to predict links in a citation network. Our approach achieves **0.97549** on the public leaderboard and **0.97312** on the private leaderboard and we rank 9th on the final leaderboard.*

1. Introduction

Link prediction is the task of predicting the existence of a link between two entities in a network. Formally, it is the task of finding an edge between two nodes in a graph. Examples of link prediction include predicting friendship links among users in a social network, predicting co-authorship links in a citation network, and predicting interactions between genes and proteins in a biological network.

2. Related Work

Historically, link prediction used hand-crafted graph heuristics. These heuristics can be classified as low-order and high-order heuristics. For example Jaccard Index and Preferential Attachment are low-order heuristics that operate on direct neighbors of nodes while Katz Index and PageRank are high-order and operate on the entire graph. Table 1 summarizes different heuristics.

Hanwen et al [4] used a weighed version of Common Neighbors and Jaccard Index where the weights depend on the authors in common, keywords in common and the publish year. In [3], Hassan et al use several hand-crafted features derived from the paper attributes, moreover, they use topological features such as shortest distance between nodes and the clustering index which measures the localized density and that is the fraction of pairs of a person's collaborators who have also collaborated with one another. Those features are then fed to several classifiers such as SVM and K-nearest neighbors.

Table 1. Graph heuristics and their formulas. $\Gamma(x)$ denotes the neighbor set of vertex x. $\beta < 1$ is a damping factor. $|\text{walks}^{(l)}(x, y)|$ counts the number of length-l walks between x and y. $[\pi_x]_y$ is the stationary distribution probability of y under the random walk from x with restart, see [6]

Name	**Formula**
Common Neighbours	$\lvert\Gamma(x) \cap \Gamma(y)\rvert$
Preferential Attachment	$\lvert\Gamma(x)\rvert\lvert\Gamma(y)\rvert$
Jaccard Index	$\frac{\lvert\Gamma(x)\cap\Gamma(y)\rvert}{\lvert\Gamma(x)\cup\Gamma(y)\rvert}$
Adamic Adar	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\log\lvert\Gamma(z)\rvert}$
Ressource Allocation Index	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\lvert\Gamma(z)\rvert}$
Katz centrality	$\sum_{l=1}^{\infty} \beta^l \lvert\text{walks}^{(l)}(x, y)\rvert$
PageRank	$[\pi_x]_y + [\pi_y]_x$

Recently with the advent of Graph Neural Networks, several papers tackle the link prediction problem by creating graph embedding and generating node features based on the graph topology, Node2Vec [2] and DeepWalk [7] are such approaches where for each node a number of random walks are generated and form "words" that are then fed to the Word2Vec [5] algorithm used in NLP to generate node embeddings. In [9], Muhan Zang and Yixin Chen propose a GNN architecture named **SEAL** that aims to learn better graph heuristics. In **SEAL**, a local enclosing subgraph is extracted around each target link, then the nodes in each enclosing subgraph are labeled differently according to their

distances to the source and target nodes. Finally a GNN is applied to each enclosing subgraph to learn a link representation.

3. Methodology

3.1. Dataset

The directed graph is a citation network that is comprised of 27 770 nodes and 335 130 edges. If there's an edge between node A and node B, then it means that node A cited node B. Each node represents a paper that has many attributes such as author and published year. Table 2 summarizes all the attributes.

Table 2. Summary of the node attributes.

Attributes
Paper ID
Publication Year
Authors
Journal
Tile
Abstract

The training set is split into three subsets that make up 6% of the dataset each and they represent the validation sets, the rest is taken as training set splits. Each algorithm is applied on all three splits and tested against the validation splits.

3.2. Heuristics Approach

In this section, we are going to present the heuristic-based approach. We start by first forming the training graphs that are constructed from the training splits and compute 128-dimensional node embeddings using Node2Vec [2]. Next, we compute embeddings for the title and abstract. Title and abstract are concatenated and fed to a pre-trained transformers model called SciBERT [1] to generate a 768-dimensional embedding that we are going to call: *paper embedding*.

Next, for each edge in the training split, we calculate all heuristics from Table 1. We aggregate the node embeddings using a function f_{AGG} and compute the cosine similarity between the *paper embeddings*. We also compute the number of common authors, the number of common terms in the journal name and the difference in the date.

All the above features are concatenated to form a final feature vector and everything is fed to the XGBoost model. We fine-tune the XGBoost model using grid search on GPU and use a 300 round on early stopping following the AUC metric. Table 3 summarizes the hyperparameter space.

After training each classifier for each split, we make the final prediction on each fine-tuned classifier and ensemble the predictions using probability averaging, next we generate predictions using a threshold that is generated from the ROC curve.

Table 3. Summary of the hyperparameter space. XGBoost is trained on GPU with a 300 round early-stopping on the AUC metric.

Name	**Space**
Maximum Number of estimators	2000
Maximum Depth	$[3, 5, 7, 10, 15, 20]$
Learning rate	$[0.01, 0.05, 0.1, 0.5]$
Minimum Child Weights	$[1, 3, 4, 5]$

3.3. Transformers Approach

Before presenting our approach, we will first introduce some neural network blocks as we can see in 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 is composed of a Linear Layer that embed the input vector into a vector of dimension 512 and then followed by a Batch Normalization. We will denote by SE the function that represent the Special Embedding.
The second one is the encoder part of the Transformer architecture [8]. We will denote by TE the function that represents the transformer encoder block.

Our model architecture is presented in Figure 2. To explain how our model works, we are going to take a certain couple of papers (A_j, A_i). We retrieve the node embeddings of these papers using Node2Vec that we will denote by N_{A_j} and N_{A_i}, and we will also retrieve the abstract and title embeddings using SciBERT (as explained in the previous subsection) that we will denote by T_{A_j} and T_{A_i}. We will denote by $G_{A_{ji}}$ the vector that result from the concatenation of the graph heuristic features and denote by $E_{A_{ji}}$ the vector that result from the concatenation of the nodes attributes (i.e the number of common authors, the number of common terms in the journal name and the difference in the date).
We will denote by MLP the final layers of the model (as it is shown in Figure 2 : One Linear Layer followed by a dropout and a ReLU activation and a final layer for the classification). If we denote by $\hat{y}$ the label predicted, we have that :

$$GE = SE(G_{A_{ji}}) \tag{1}$$

$$EE = SE(E_{A_{ji}}) \tag{2}$$

$$NEnc = TE(SE([N_{A_j}, N_{A_i}])) \tag{3}$$

$$TEnc = TE(SE([T_{A_j}, T_{A_i}])) \tag{4}$$

$$\hat{y} = \sigma(MLP(TE([GE, EE, Nenc, TEnc]))) \tag{5}$$

Where σ represent the sigmoid activation.
Each SE (Four) and TE (Three) have their own weights. The parameters of the Transformer Encoder are :

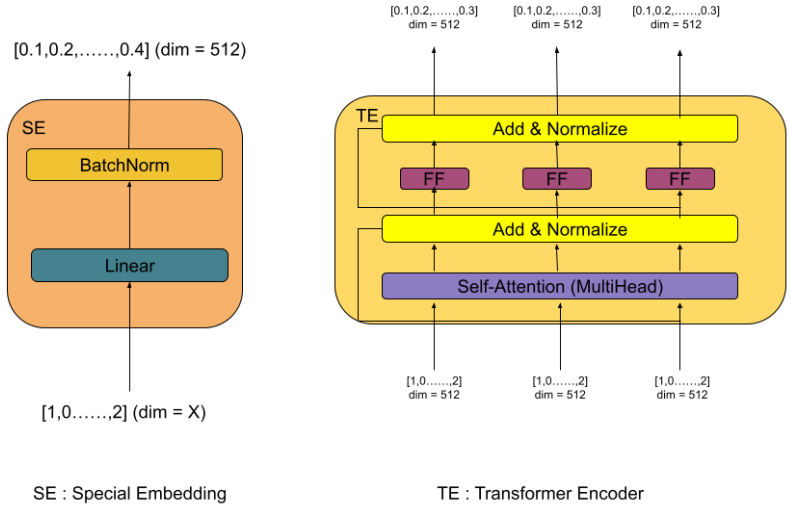

Figure 1. Model Blocks

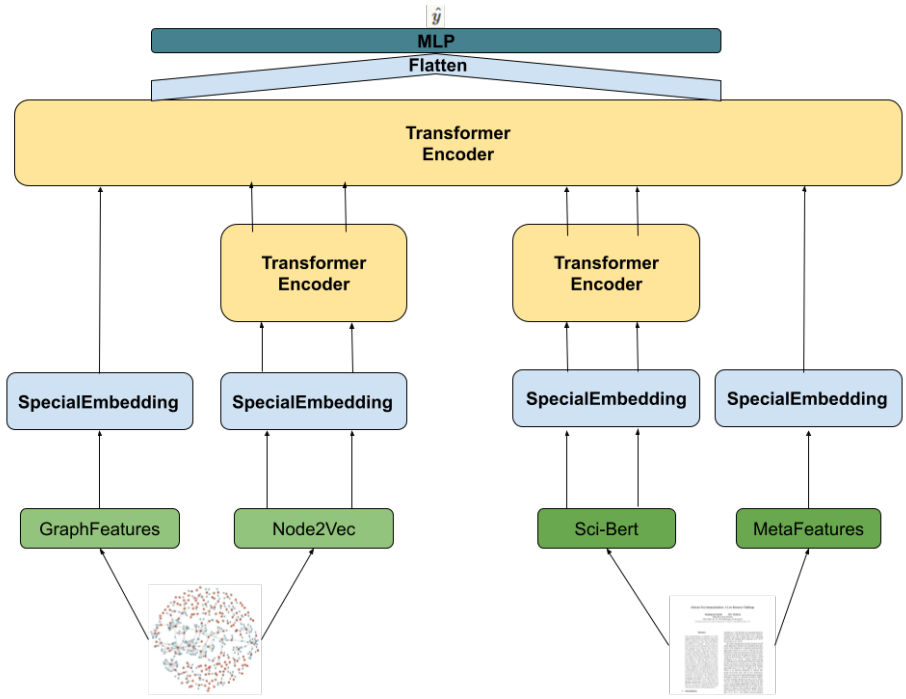

Figure 2. Model's Architecture

- Number of heads : 8
- Number of Encoder Layer : 1
- Encoder Dropout : 0.2

Experimental Setup : The model performance is evaluated by the F1-Score on the validation set. The loss that we considered is the Binary Cross-Entropy Loss and we updated the model weights using ADAM Optimizer, and we chose the following hyperparameters :

- Starting Learning Rate : 0.0008
- Batch Size : 64

- Epoch : 40

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 (F1-Score) on the validation set has stopped improving. Each 1/3 epochs, we evaluate our model on the validation set and we saved the one that performed best on the validation set.

4. Results

In this section we report the results of our approach. We tried several aggregating functions for node-embeddings but two gave us good results: Taking the difference between the embeddings for each edge and concatenating the embeddings. The latter gave us a big boost in the performance as reflected in Table 4. The performance is measured using the f1-score.

Table 4. Results of different approaches.

Approach	**Public Leaderboard**	**Private Leaderboard**
$f_{AGG} = x - y$	0.97064	0.96851
$f_{AGG} = CAT(x, y)$	**0.97549**	**0.97312**
Transformer Approach	0.96544	0.96129

5. Discussion

First of all we can notice a big jump in the score between concatenating the embeddings and taking the difference. That is due to the fact that by giving the concatenation, the model is free to choose and use all features from both node embeddings while the difference imposes a prior that might not be suitable for the problem.

Concerning the heuristics, not all of them performed well. According to the XGBoost model, the best performing heuristics were: Page Rank, Preferential Attachment and Katz centrality along with the hand-crafted feature date difference and cosine similarity between paper embeddings. We see that the date difference plays a big role since a paper can not cite another one that has been published afterwards [1], moreover, paper embeddings also play a big role since similar abstracts and titles can increase the chance of citing. The Katz Centrality and Page Rank proved to be powerful since these are heuristics that are computed not only for immediate neighbors of nodes but iteratively until it combines the entire graph and that gives much more information with the only exception to this being the Preferential Attachment that is just the product of the degrees of nodes, however, it is meaningful because papers with high citations are more probably going to be cited more.

[1] Although we found around 2000 edges that violate this observation and that's a flaw that comes from papers having different versions

On the other hand, the features that did not contribute much to the classifier are: Ressource Allocation Index, Jaccard Index, Adamic Adar, Common neighbors and features concerning the authors and journals. It was surprising that the number of common authors and Journal didn't perform well since intuitively, authors tend to cite their own papers and researchers from the same journal have a higher chance of citing each other compared to different journals.

Concerning the Transformer Approach, it has not outperformed the Xgboost approach. However we did not have enough time to finetune both the hyperparameters (number of layers, dropout, batch size, learning rate ...) and the architecture (try different fusion strategies, different way to combine the features (instead of just flattening the output), but we think that there is still room for improvement.

6. Conclusion

In this report we explored various techniques for link prediction, from heuristics based approaches to Transformers. While the transformers did not outperform the basic machine learning approach, we are confident that with more work we can beat our baseline. The code can be found here : https://github.com/omarsou/link_prediction_challenge.

References

[1] I. Beltagy, K. Lo, and A. Cohan. Scibert: Pretrained language model for scientific text. In *EMNLP*, 2019.

[2] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks, 2016.

[3] M. Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. 01 2006.

[4] H. Liu, H. Kou, C. Yan, and L. Qi. Link prediction in paper citation network to construct paper correlation graph. *EURASIP Journal on Wireless Communications and Networking*, 2019:1–12, 2019.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking : Bringing order to the web. In *WWW 1999*, 1999.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. 2014.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[9] M. Zhang and Y. Chen. Link prediction based on graph neural networks, 2018.