CITATION PREDICTION CHALLENGE
ALTEGRAD : PROJECT REPORT

Kevin Assobo Bagueka
MSc. Data Science
Institut Polytechnique de Paris
Email: kevin.assobo-bagueka@polytechnique.edu

02/21/22

Link prediction aims to infer missing links or predicting the future ones based on currently observed partial networks, it is a fundamental problem in network science with tremendous real-world applications. In academia, the citation network is essential to analyze relationships between researchers or papers. In this article, we build a model to predict the existence of citations between papers by formulating link prediction for a large citation network. The model takes as inputs semantic features from the abstracts of the papers and structural features from the citation graph, and predict whether two nodes are linked by an edge or not.

Introduction

When writing a paper, researchers tend to cite other papers as references. However there might be thousands of related papers for their work, thus there must be some specific rules that they follow in order to find the ideal references. We want to investigate the factors that might affect the existence of a citation between two papers.

The conventional link prediction methods can be divided into several categories. Local link prediction approaches make prediction based on the assumption that two nodes are more likely to be connected if they have many common neighbors [13]. These local similarity based methods are fast and highly parallel since they only consider local network structure. But their prediction accuracy are very low especially when networks are sparse and large. The global link prediction approaches take the whole network's structural similarity into consideration [9], those methods have higher link prediction accuracy compared with the local ones, but they have high computational complexity problem which prevent them to be applied on graphs that contain million and billion nodes. There are also some probabilistic and statistical based approaches that assuming there is a known prior structure of the network, like hierarchical structures [3]. But those methods can not get over the problem of low link prediction accuracy. Besides, we can hardly extract the hidden network structure and node properties from above mentioned conventional link prediction approaches.

Recently, there has been a surge of algorithms that seek to make link prediction through network representation learning which automatically decoding local and global structural information from graphs. The idea behind these algorithms is to learn a mapping function that embedding nodes as points in a low-dimensional space $\mathbb{R}^d$ with nodes vector represent the information extracted from the original graph. One of such representation learning algorithm is *node2vec* [5] which we used in our work. Those algorithms are task agnostic, the learned representations then used to perform graph-based downstream machine learning tasks, such as node classification, node ranking as well as link prediction. Compared with the conventional ones those representation based methods have achieved a much higher link prediction accuracy.

More recently, deep learning techniques based on neural networks have achieved triumphs in image processing and natural language processing. This stimulates the extensions of the methods on graph structures to perform node classification and link prediction tasks by converting the network structures into a low dimensional representations. For example, Kipf and Welling proposed graph convolutional network (GCN) [6] which borrows the concept of convolution from the convolutional neural network (CNN) and convolve the graph directly according to the connectivity structure of the graph; after that they further proposed *graph auto-encoder* (GAE) and *variational graph auto-encoder* (VGAE) [7] which in contrast to most existing models for unsupervised learning on graph-structured data and link prediction can incorporate node features and achieve competitive results.

There are even more advanced deep learning algorithms for graphs but due to limitations in time and resources we were not able to explore them. Our final model uses local and global structural features from the citation graph as well as node embeddings obtained with the *node2vec* algorithm.

1 Processing

1.1 Paper citation graph

The citation network we used is represented by an undirected graph and there is an edge between nodes v and u, then either paper v cites paper u or paper u cites paper v.

In order to evaluate the algorithms while avoiding data leakage, we have to carefully split the dataset. If we consider the initial graph available as the *full graph* with $n = 138,499$ nodes and $m = 1,091,955$ edges, the initial dataset consists of $2m$ node pairs (m edges and m randomly chosen non-edges). Then we create a *train* and *test* datasets in the following way:

- First we randomly sample a fraction $p = 0.1$ of the edges of the *full graph* and the same number of non-edges. These samples of node pairs constitute our *test set*. We then remove the sampled edges from the *full graph* and we obtain our *test graph*.

- We repeat the same process with the *test graph* to obtain the *train set* and the *train graph*.

We used the `EdgeSplitter` class from the python library `StellarGraph` [4] in order to carry the above splittings. At each step we constrain the `EdgeSplitter` to sample edges to remove while insuring that the resulting graph keep the same number of nodes and connected components. Indeed this allows us to avoid having unknown nodes in a dataset for which we cannot get topological features.

The node pairs for which we are supposed to submit the scores on Kaggle will be called the *challenge set*.

1.2 Abstracts

Since we have the abstracts of the papers, one approach is to represent them by vectors. Therefore we need to learn the embeddings of each abstract. In order to do that, we used the *Doc2Vec* [8] algorithm with the Distributed Memory Model of Paragraph Vectors (PV-DM) model implemented in `gensim`. The training took approximately 1 hour to complete 100 epochs.

1.3 Authors

Although we are analyzing the paper citation network, it is the authors who make the decision of citations. Therefore, we think it is also useful to incorporate the authors' properties in our model. For each paper, the list of its authors is available. We started by encoding the authors names into integer identifiers for easier manipulation. Then two more networks were constructed.

1.3.1 Author collaboration network

We want to create a weighted undirected graph of authors, where two authors are connected by an edge if they co-authored at least one paper. Note that this graph is independent from the paper citation graphs (*full, test, train*) and since the three of them contain all the papers, there will not be any data leakage from the collaboration information.

We build the adjacency matrix of this graph as a weighted collaboration matrix $W \in \mathbb{R}^{n \times n}$ (n := number of authors) such that for two authors i and j:

$$W_{ij} = \sum_{p \,\in\, papers} \frac{\delta_i^p \delta_j^p}{n_p - 1} \quad \text{if} \quad i \neq j \quad \text{and} \quad W_{ii} = 0 \qquad (1)$$

where n_p is the number of authors of paper p and $\delta_i^p = \mathbf{1}(i \in \text{authors of } p)$.

This formula is used in [1] so that the weight of a collaboration in a paper is correlated to the number of authors of the paper.

1.3.2 Author citation network

We want to create a weighted undirected graph of authors, where two authors are connected by an edge if there is at least one time when one of them cited the other in a paper. Thus we need the paper citation graph. Since we have three paper citation graph, we will build three author citation graph to avoid data leakage.

Consider a paper citation graph $G = (A, E)$. We build the adjacency matrix of the corresponding author citation graph as a weighted citation matrix $W \in \mathbb{R}^{n \times n}$ such that for two authors i and j:

$$W_{ij} = \sum_{(p_1, p_2) \,\in\, E} \frac{\delta_i^{p_1} \delta_j^{p_2}}{n_{p_1} n_{p_2}} \qquad (2)$$

where n_p is the number of authors of paper p and $\delta_i^p = \mathbf{1}(i \in \text{authors of } p)$.

We use this formula so that the weight of a citation is correlated to the number of authors of both papers. Note that we did not put $W_{ii} = 0$ because we consider self-citations.

2 Features

2.1 Attribute features

The only attribute feature we used is the **number of common authors** between two papers. If more information were available we could have used for example the *difference in publication year* or a boolean value equal to 1 if the two papers are published in the same journal/conference.

2.2 Semantic features

Once we obtain the embedding vectors of the abstracts we can compute the **cosine similarity of the abstract embeddings** vectors of two papers. We hypothesize that the more similar the topics of two papers are, the more likely that a citation exists.

2.3 Topological features

We compute the following features for the *train set* by using the *train graph*, for the *test set* by using the *test graph* and for the *challenge set* by using the *full graph*.

2.3.1 Local based features

Sum of degrees and **absolute difference in degrees** between two nodes.

Jaccard coefficient (JC): number of common neighbors between two nodes u and v, normalized by considering the total set of shared and non-shared neighbors.

$$JC(u,v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \tag{3}$$

where $\Gamma(\cdot)$ represents the set of adjacent nodes and $|\cdot|$ the cardinality of a set.

Preferential attachment (PA): The idea behind this metric is that new nodes joining the network are more likely to connect with the nodes with higher connections (hub) than the nodes with lower degrees.

$$PA(u,v) = |\Gamma(u)| \times |\Gamma(v)| \tag{4}$$

Adamic-Addar index (AA): this metric has similarities to the number of common neighbors but the vital difference is that a logarithm term penalizes the shared neighbors of the two corresponding nodes.

$$AA(u,v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|} \tag{5}$$

2.3.2 Global based features

Shortest path length: If there is no path between the two nodes we assign a value -1.

Absolute difference in betweenness centrality : the betweenness centrality represents the extent to which a node lies on a path between other nodes and can also be interpreted as measuring the influence a node has over the spread of information through the network. We can compute betweenness centrality of a node u in a graph $G = (V, E)$ as the sum of the fraction of all-pairs shortest paths that pass through u :

$$c_B(u) = \sum_{\substack{s,t \in V \setminus \{u\} \\ s \neq t}} \frac{\sigma(s,t|u)}{\sigma(s,t)} \tag{6}$$

where $\sigma(s,t)$ is the number of shortest paths between s and t and $\sigma(s,t|u)$ is the number of such shortest paths passing through node u.

Absolute difference in closeness centrality : closeness centrality of a node u is the reciprocal of the average shortest path distance to u over all reachable nodes. If we note G_u the connected component of the graph G that contains u, $d(u,v)$ the shortest-path distance between u and v, and $|\cdot|$ the number of nodes, then the closeness centrality of u is computed by :

$$c_C(u) = \frac{|G_u| - 1}{|G| - 1} \frac{|G_u| - 1}{\sum_{v \in G_u \setminus \{u\}} d(u,v)} \tag{7}$$

Absolute difference in eigenvector centrality : The eigenvector centrality for node i is the i-th element of the vector x defined by the equation $Ax = \lambda x$ where A is the adjacency matrix of the graph with eigenvalue λ. By virtue of the Perron–Frobenius theorem, there is a unique solution x, all of whose entries are positive, if λ is the largest eigenvalue of the adjacency matrix [11].

Absolute difference in pagerank: The PageRank of a node is a ranking of the node based on the structure of the incoming relationships and the importance of the corresponding source nodes. Therefore it measures the importance of the node within the graph. Finally, the PageRank might be an important feature for our task because the more influence a paper has in the citation network, the more likely it is to be cited by future papers.

Most of the features above were taken from [10] and where computed with `networkx` within few minutes, except for the **betweenness centrality** and **closeness centrality** which were computed with `GraPE` [2] for parallel computation. It took almost 10 hours to compute only these two features for all three paper citation graphs. This is due to the fact that they use a large number of shortest-paths.

2.3.3 Features from the author networks

Average weight of author collaborations: For two papers $p1$ and $p2$ and their respective list of authors $A1$ and $A2$, we consider all the pairs $(a1, a2) \in A1 \times A2$ which are edges in the *author collaboration graph* and we computed the average of their weights. If none of the pairs of authors is an edge then we assign a value 0. This give us the average "power" of collaboration between authors of $p1$ and authors of $p2$.

Average weight of author citations: We use the same method as above but with the *author citation graph*. Remember that there are three citation graph so for each set

of data, we use the corresponding graph. This feature can help us determine if an author of $p1$ ($p2$ resp.) has cited an author of $p2$ ($p1$ resp.) in the previous graph.

3 Node embeddings

3.1 *node2vec*

As mentioned in the introduction, we used the *node2vec* algorithm in order to compute node embeddings for the paper citation network. The *node2vec* framework simulates a family of biased random walks which efficiently explores diverse neighborhoods and treats these walks as sentences [5]. The "sentences" are then fed into a natural language processing model such as *Word2Vec* to learn representations of the "words"/nodes.

Again, since we have three of such graphs (*full, train, test*) we computed the embeddings of the nodes for each of them independently. The training of the model took approximately 30 minutes to completely for each graph. The parameters values were 15 for the random walk length and 100 for the number of walks starting at each nodes.

After that, we have to create features for a node pair because the instances of our datasets are node pairs and not single nodes. Similarly to what we did for the abstracts, we choose to compute the **cosine similarity of the node embeddings** for each node pair.

3.2 Graph Auto-encoder (GAE)

We also wanted to explore GAE [7]to compare the node representations obtained with those of *node2vec*. We use the same implementation as in the course lab 6 and as node features we gave the abstracts embeddings. Then we also computed the cosine similarity for each node pair.

4 Classification algorithm

With all our features ready, the last part was the choice of the machine learning classifier. We chose the XGBoost classifier for two main reasons : computation speed and model performance. Indeed given the challenging context, we want a model with high performance and low time processing.

Logistic regression is an interesting linear method but the performance is lower than tree based methods which in turn are very slow for large datasets. XGBoost fix this issue with parallelized tree building and can even use GPUs. It also comes with many regularization parameters to avoid overfitting.

There are many parts we define for evaluating our model. We split the previous training set into a new **training set** and **validation set**. We also split the test set into a new **test set** and a **calibration set**. Each of these plays an important role in the next stages. We also scaled the data using `StandardScaler` from scikit-learn.

4.1 Hyper parameter tuning

We used `RandomizedSearchCV` to find a good set of hyper parameters for the model. In contrast to `GridSearchCV`, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. This means that the resulting set is not necessary the best one. We fitted this method with our **training set** and it uses cross-validation on this set with an objective of minimizing the logarithmic loss. We used GPU to fit each estimator since XGBoost allows it.

Once we have the best estimator we train it on the **training set** again, this time by allowing "early stopping" once the logarithmic loss starts to increase on the **validation set**. This allows to avoid overfitting.

4.2 Calibration

We decided to use isotonic regression [12] implemented in `CalibratedClassifierCV` for probability calibration. The purpose is to decrease the logarithmic loss and we use the **calibration set** to fit the class.

5 Results

With the use of Google collab and the available GPU, the random search for hyper parameter is done in less than 5 minutes and after that, training the best estimator takes less than 2 minutes. We did not try to do it with CPU but surely it would have lasted for a couple of hours.

5.1 Performance

Here is the performance of our model on the test set :

	log loss	ROC AUC	F1 score
Before calibration	**0.102**	0.993	0.963
After calibration	**0.100**	0.993	0.963

Table 1: Performance of the model on the test set

The logarithmic loss calculated with 50% of challenge set is approximately equal to **0.10522** on Kaggle. This value may be subject to slight variations depending on the result of the random search of hyper parameters.

5.2 Interpretation

5.2.1 Feature importance

Figure 1 shows the importance of each of the features we used in the model. We can see for example that the cosine similarity of the node2vec embeddings is the second most

important feature while the cosine similarity of the GAE embeddings is the second least important feature. There is probably an issue here with our implementation of the GAE. It may come from the fact that many abstracts were not available and the initial Doc2Vec model used empty lists to compute the embeddings of the corresponding abstracts. Then we used these embeddings as features for the GAE. We did not have time to further explore this issue.

Another observation from figure 1 is that the absolute differences in centrality measures have an insignifiant importance even though they were most time expensive to compute. These measures give insights on how global information spread through a node and taking the difference might not a good idea for a node pair. If we had more time we could have explore the effects of other operators instead of the difference.

5.2.2 Shap values

Figure 2 shows how the values of each feature in the test set influence the output of the model. For example, the two most important features have an opposite influence. The larger the shortest-path length between two nodes, the lower the predicted probability of the existence of an edge between them. Whereas the higher the cosine similarity of their *node2vec* embeddings, the higher the predicted probability of a link.

6 Conclusion

The performance of our model can definitively be improved with better representations of the abstracts and the nodes of the paper citation graph. As mentioned in the introduction, more advanced deep learning models exist, such as the variational graph auto-encoder (VGAE) [7], and can be used to compute the representations. The very first improvement could be done on the abstract embeddings which would then be fed as features to the GAE or VGAE. Concerning the challenge, our logarithmic loss on the public leaderboard is currently the sixth best score with a value of 0.1052 but there are a few hours left before the end.

Finally this project was very challenging but also very rewarding, as we were able to apply some of the methods we learned during the ALTEGRAD course and explore other techniques from academia.

References

[1] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. In *PNAS*, volume 101, pages 3747–3752, 2004.

[2] Luca Cappelletti, Tommaso Fontana, Elena Casiraghi, Vida Ravanmehr, Tiffany J. Callahan, Marcin P. Joachimiak, Christopher J. Mungall, Peter N. Robinson, Justin Reese, and Giorgio Valentini. Grape: fast and scalable graph processing and embedding, 2021.

[3] Aaron Clauset, Christopher Moore, and M. E.J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008.

[4] CSIRO's Data61. Stellargraph machine learning library. `https://github.com/stellargraph/stellargraph`, 2018.

[5] 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*, page 855–864, 2016.

[6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

[8] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[9] Haifeng Liu, Zheng Hu, and Hamed Haddadi. Hidden link prediction based on node centrality and weak ties. *EPL (Europhysics Letters)*, 101:18004, 2013.

[10] Ece C. Mutlu, Toktam A. Oghaz, Amirarsalan Rajabi, and Ivan Garibay. Review on learning and extracting graph features for link prediction. *arXiv preprint arXiv:1901.03425*, 2019.

[11] Mark E. J. Newman. Networks: An introduction. *Oxford University Press*, 2010.

[12] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *KDD*, volume 101, pages 694–699, 2002.

[13] Tao Zhou, Linyuan Lu, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal*, B(71):623–630, 2009.

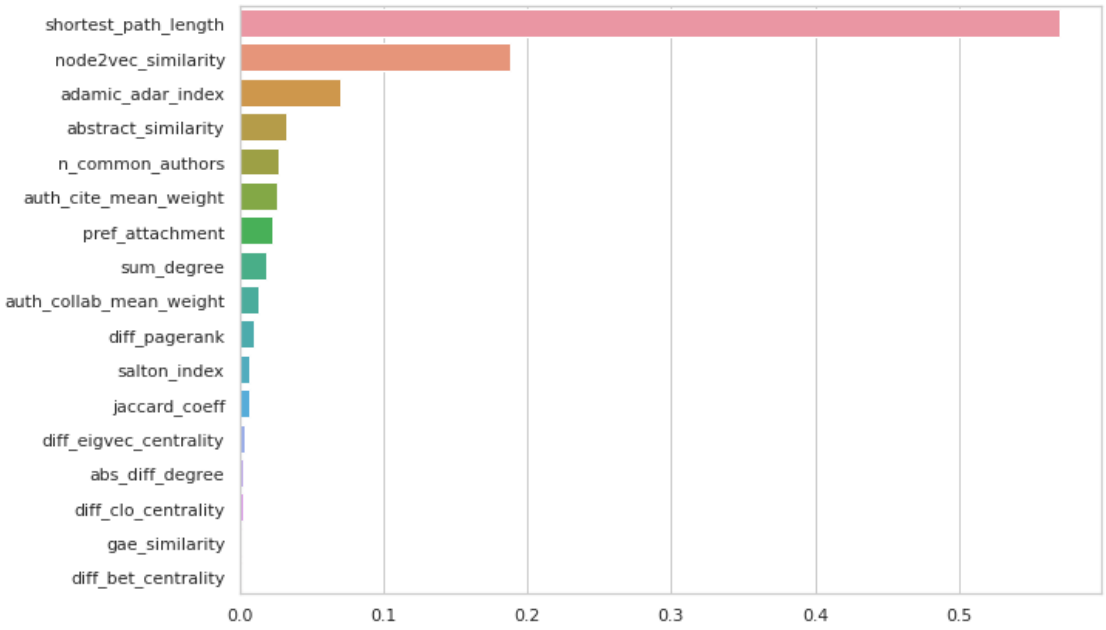

Figure 1: Feature importance of the model

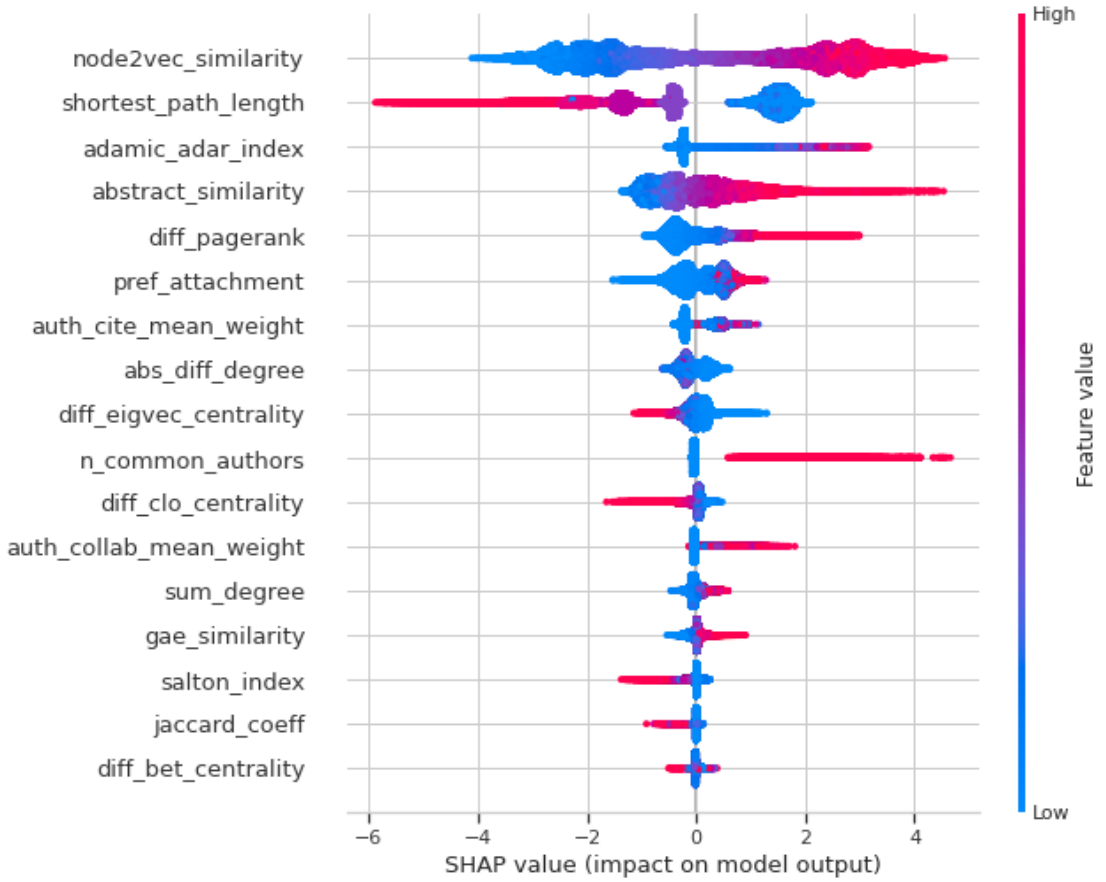

Figure 2: Impact of the feature values of the test set on the model output