Link Prediction in Citation Networks

Yunguan FU, Cheng ZHANG, Yunxiang ZHANG

1 Introduction

Most phenomena in the real world are complex and dynamic. New nodes and links are added so frequently that the structures of the entire network change and grow quickly over time. Therefore, it is of great interest to predict the appearance of a new links or to identify links which may exist but are not represented in the data. Typical applications are commonly seen in recommendation engines, such as new connections on social networks or related products on shopping sites. In this competition, we aim to perform link prediction in a paper citation network. More precisely, given a citation network whose edges have been removed partially, our mission is to reconstruct the initial network by using graph-theoretical, textual, and other information.

The report is organized as follows. Section 2 introduces the problem setting and dataset. Section 3 describes the methodologies we used in problem solving. Section 4 presents the results of different experiments we conducted throughout the competition. Finally, Section 5 presents our conclusions.

2 Problem Set

2.1 Description

Consider a citation network as a directed graph $G(V, E)$ where V is the set of nodes and E is the set of edges (links). Each node corresponds to a paper and the existence of an edge between two nodes means that the source paper cites the target paper. Moreover, each node is associated with information such as the title of the paper, publication year and journal, author names and the abstract. Our mission is to predict if there exists an edge (link) between two nodes (papers) given the above information and we used F1 score as our evaluation metric.

2.2 Dataset

The given dataset contains three files: a training set which contains 615512 entries which indicates whether an edge exists between two nodes; a node information file which provides the title, authors, publication year, journal and abstract for 27770 papers; and a testing set which contains 32648 pairs of nodes that we should predict if a link exists. Besides these data, we also used fastText's [1][2] pretrained word vectors, which is trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset.

3 General Pipeline

3.1 Data Preprocessing

Before extracting features, it is essential to perform data preprocessing to ensure the consistency and completeness of data. Here, we mainly focused on those text-based features in node information.

For author names, we removed all the numbers and replaced "()&@" by comma because sometimes the author name contains his/her institution and we would like to treat these institutions as authors too, in order to provide more information. Similarly, we removed all non-alphabeta characters in journal names.

Concerning the titles and abstracts, we first used "smart stopwords", which was provided during the course, to remove more stopwords. Then we replaced hyphen and quotation marks by space and removed all digital numbers. After dividing the text by space, we stored only tokens which have at least three characters. We also stored all the processed titles and abstracts together as a corpus for further training.

Due to the limitation of memory, we have to extract pretrained word vectors from the wiki news dataset instead of loading one million vectors every time. However, not all the words are contained in this dataset, so we apply the following strategy to extract vectors for as many words as possible: if a word w itself doesn't have pretrained vector, we calculate its stemmed token and check if there are any other word u_i, which has pretrained vector and the same stemmed token as w. If such words exist, we calculate the mean of $\{u_i\}$'s pretrained vectors as w's word vector. Surprisingly, by this way we were able to find a pretrained vector for every word.

3.2 Feature Engineering

Generally, the link prediction problem is mainly studied from two angles: network structure and attributes of nodes and connections. Structure refers to the way in which nodes that compose the network are interconnected. It reflects the information about network topology. The attribute information refers to description of nodes' features, which is difficult to show directly in the network graph.

In this problem, we mainly studied the directed graph of papers as well as paper's attributes such as title, abstract and authors. We also studied the undirected graph of authors. Two authors are linked if they have collaborated and published a paper together. The weight of their edge is the number of collaborations. Similarly, an undirected graph of journals could be created. Two journals are linked if some authors have publications for both of them. The weight of the their edge is the number of such authors.

Based on these graphs and node information, we have three types of features: structure-based, node-based and mixed.

3.2.1 Structure-based features

3.2.1.1 Neighbourhood-based metrics

The area of structure-based prediction has been widely studied and there are many well-known metrics [3]. Here we selected and used five commonly used metrics which are neighbourhood-based: Number of Common Neighbours, Jaccard's Index, Preferential Attachment, Adamic/Adar Index and Salton Similarity (Appendix A.1).

Since these metrics all depend on neighbour's definition, we could adapt different definitions to create more features. More precisely, for the directed graph of papers, there are three ways to determine if node y and node x are adjacent: (1) if there is an edge from x to y (2) if there is an edge from y to x (3) if there is an edge between x and y. Then we can define the neighbourhood using either one-hop neighbour set or two-hop neighbour set: (1) two nodes are neighbours if they are adjacent (2) two nodes are neighbours if they are adjacent or they are all adjacent to another node.

By combining different metrics and neighbourhood definitions, we created 30 features for the directed graph of papers and 10 features for the undirected graphs of authors and journals.

3.2.1.2 Node2vec

Another way to extract useful representations from graphs is to learn them directly by optimizing a neighborhood preserving objective. One of such methods is the node2vec framework [4], which simulates biased random walks (Appendix A.2) to create a context for each node. The contexts can be regarded as documents which consist of words (nodes). Therefore, we can use Word2Vec [5] to calculate an embedding vector for each node. Then, given two nodes, we can simply calculated their vectors' cosine similarity or use a more complex method such as neural networks.

3.2.2 Node based features

For each paper, we have its title and abstract. Apart from using tf-idf vectors, we would also like to extract an embedding vector from these text information. We could first calculate the embedding vector for each word and then calculate a document vector for title and abstract. Or, we could calculate their document vectors directly.

3.2.2.1 Word2vec

For calculating word vectors, we combined all titles and abstracts as the corpus. There are multiple choices for training the word vectors. We could utilize the pretrained vector or use fastText to train our own vectors with the help of corpus. We could also use the pretrained vector as initial values and use fastText to adapt the vectors to our corpus.

Then, to calculate an embedding vector for each title and abstract, we could perform keyword selection and calculate the mean vector of all key words or we could use convolutional layer to extract more sophisticated features.

3.2.2.2 Doc2vec

Instead of training word vectors and document vectors separately, we can also train them at the same time or directly train the document vectors. Two such models are PV-DM and PV-DBOW [6] (Appendix A.3). Since titles and abstracts have many common words, we treat them equally as documents and train their vectors together.

3.2.3 Mixed features

Both the graph structure and node information are important. So we would like to combine them to extract some more useful features. One solution is to use node2vec to generate random walks as training corpus, then use fastText to train the document vectors by using the results of doc2vec as initial values. Another solution is to calculate the mean of neighbours' vectors as a feature, based on the assumption that if the neighbourhood of two nodes have similar vectors, these two nodes should be connected.

3.3 Classifier

After feature engineering, we mainly have two types of features: pair-based and node-based. For each pair of papers, we calculated their neighbourhood-based features, tf-idf similarity of title and abstract, and the difference of publication year. For each paper, we calculated its embedding matrix of title and abstract and different embedding vectors (doc2vec, node2vec, etc.).

Concerning the classifier, we mainly used two types of classifiers: decision tree based algorithms (LightGBM, XGBoost) and neural networks.

3.3.1 Decision tree based algorithms

To reduce the dimension of input, we calculated the mean embedding vectors of title/abstract as paper's new embedding vector for title/abstract. Then, for each pair of nodes and each type of embedding vector, we calculated the cosine similarity of their vectors. At last, we combined all the features as input for classifiers.

3.3.2 Neural Network

Our neural network contains multiple sub-modules, such as Convolutional Neural Network models (CNN) and Multi-Layer Perceptrons (MLP).

For title/abstract, the CNN model takes two matrix as inputs, which correspond to the embedding matrices of two nodes, and uses 1d-convolutional filters of different sizes and strides to extract features. After a maxpooling layer, the results are concatenated and passed into a Multi-Layer Perceptron (MLP). The output is a low-dimensional vector.

For each embedding method, its MLP model takes two vectors as inputs, which correspond to two nodes' vectors. Then it concatenate these two vectors and pass the result into a MLP. The output is a low-dimensional vector.

Then we concatenate the precalculated pair-based features and all the output vectors of sub-modules into a single vector. At last, we pass it into another MLP whose last layer uses a sigmoid activation and the output is the probability of the citation between two papers. The visualization of the structure is presented in Figure 4.

4 Experiments

Our first model only added tf-idf similarity of title and abstract and used LightGBM as classifier. This simple model gave us a F1 score around 0.87, which proved the usefulness of text information for link prediction. Inspired by this, we calculated word embedding vectors and the mean of vectors for each title and abstract for every paper. For each pair of papers, we used the cosine similarity as a new feature. The F1 score was then improved to 0.88. Here, F1 score is referred to the score of public leaderboard unless otherwise stated.

We also calculated the cosine similarity of some other embedding vectors, such as using node2vec for graphs of papers, authors and journals. However, in spite of many experiments, the F1 score remained unchanged while LightGBM showed that these features have considerable importance. We thought that the computation of cosine similarity might loss too much information. Therefore we applied CNN models to extract more complicated features, which improved the F1 score into 0.91.

We noticed that while concentrating on node-based features, we ignored the structure of the citation network. With the assumption that if two papers cite many common papers, it's likely to have a link between them, we added some structure features based on the one-hop out neighbours, i.e. paper y is a neighbour of paper x if x cites y. By only using LightGBM, the F1 score is improved directly from 0.88 to 0.93. Impressed by the improvements, we used different neighbourhood definitions to create more features as described in section 3.2.1.1 and by using XGBoost, the F1 score was increased to 0.97478.

We continued to experiment some other techniques such as feature engineering (such as calculating mean vector of neighbours, using different word vectors, etc.) and parameter tuning. We also compared multiple classifiers (Appendix A.5) and it turned out that XGBoost was the best. Despite of all the efforts, we didn't succeed to find more valuable features and the public score wasn't improved.

It's worth mentioning that we also tried the node2vec features based on the graph of papers. However, we observed an overfitting phenomena: even though this is the most important feature (according to the classifier) and the local validation F1 score is over 0.99, the public F1 score is only around 0.94. Here we used very few estimators and the original training set was divided by two into training/validation set. We believe that this is due to the leak of information via graph structures, since we trained those node2vec vectors by using the network constructed by the whole original training set. However, due to the long training time of node2vec, we didn't investigate more on this issue.

Finally, we constructed a new neural networks as described in section 3.3.2 to combine everything together with the aime to extract more abstract features. Due to the complexity of the network and the low training speed with laptops, we only managed to have a score of 0.96712 after parameter tuning.

From these experiments, we observed that: both the structure-based and node-based features are important. For instance, in Table 2 of Appendix A.6, we list the importance of some features of our best model (private leaderboard score 0.97715). We can observe that neighbourhood-based features played an essential role in the model. Concerning those different embedding vector-based features, they are helpful. Especially, the mean vector of neighbours is in fact very important, since it combines both structure information and node attributes. Also, the graph of authors and journals that we created can do provide some helpful information.

5 Conclusion

In this competition, we performed link prediction in a paper citation network by extracting both structure-based features and node-based features. We experimented multiple classifiers including LightGBM, XGBoost and Convolutional Neural Network (CNN). Our final (private) F1 score is 0.97667 while our best score in history submissions is in fact 0.97715. Through these experiments, we found that both two types of features are necessary to the link prediction problem and those tree-based models are more powerful than other simple classifiers. Concerning the text information, a CNN is able to extract more semantic information than simply calculating the cosine similarity. Finally, there are some other methods we thought might be useful while we did not have enough time to implement. For example, we could do keyword extraction instead of using the whole abstract and we could also utilize other types of neural network like recurrent neural network.

References

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[2] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[3] Fei Gao, Katarzyna Musial, Colin Cooper, and Sophia Tsoka. Link prediction methods and their accuracy for different social networks and network metrics. *Scientific Programming*, 2015:1, 2015.

[4] 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. ACM, 2016.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[6] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

A Appendix

A.1 Neighbourhood-based metrics

Throughout this section, the symbols x and y denote nodes, $\Gamma(x)$ and $\Gamma(y)$ denotes their neighbour sets, respectively.

Common Neighbours This method is based on the assumption that two nodes with many common neighbours will be connected in the future.

$$|\Gamma(x) \cap \Gamma(y)|$$

Jaccard's Index This method is used for comparing similarity of neighbours sets and its value is between 0 and 1.

$$\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

Preferential Attachment This method is based on the assumption that the node with high degree is more likely to get new links.

$$|\Gamma(x)||\Gamma(y)|$$

Adamic/Adar Index This method was initially designed to measure the friendships and the assumption is that it's more likely for a common acquaintance of two people to introduce them to each other when he has only few friends.

$$\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log(\Gamma(z))}$$

Salton Similarity (Cosine Similarity) The idea of this method is based on the dot product of two vectors.

$$\frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{|\Gamma(x)||\Gamma(y)|}}$$

A.2 Random walk of node2vec

We mainly consider a simple random walk, where at each step the location jumps to a neighbour node according to some probability distribution. The distribution is calculated as follows. For a random walk that just traversed edge (t, v) and now resides at node v (Figure 1), the walk now needs to decide where to go on the next step so it evaluates the transition probabilities π_{vx} on edges (v, x) leading from v. We set the unnormalized transition probability to $\pi_{vx} = \alpha(t, x; p, q)\ w_{vx}$, where w_{vx} is the weight of edge,

$$\alpha(t, x; p, q) = \begin{cases} \frac{1}{p} & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ \frac{1}{q} & d_{tx} = 2 \end{cases}$$

and d_{tx} denotes the shortest path distance between nodes t and x. p, q are hyper-parameters that balance the exploration-exploitation. In our experiments, they are all set to 1.

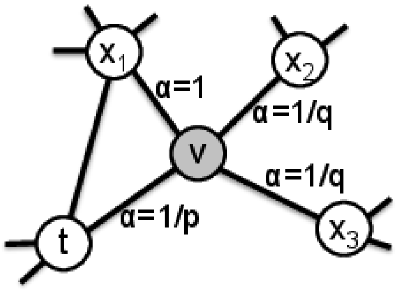

Figure 1: Random walk procedure in node2vec

A.3 Doc2vec

PV-DM Distributed Memory version of Paragraph Vector model is similar to the CBOW model, but instead of using just words to predict the next word, it adds another document-unique feature vector (Figure 2) so that during the training of the word matrix W, the paragraph matrix D is trained as well.

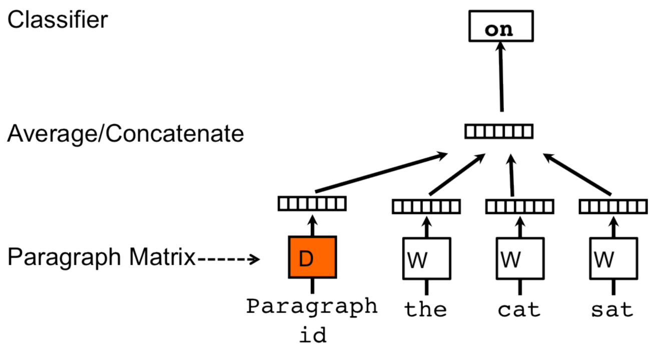

Figure 2: PV-DM

PV-DBOW Distributed Bag of Words version of Paragraph Vector model is similar to the skip-gram model, but instead of using a word to predict the context, it used the document-unique feature vector (Figure 3) so that we have no more need to to save the word vectors.

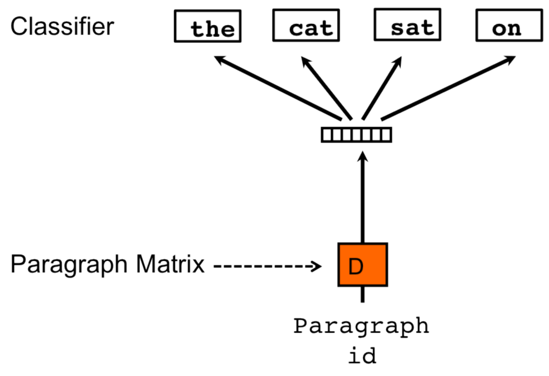

Figure 3: PV-DBOW

A.4 Neural network structure

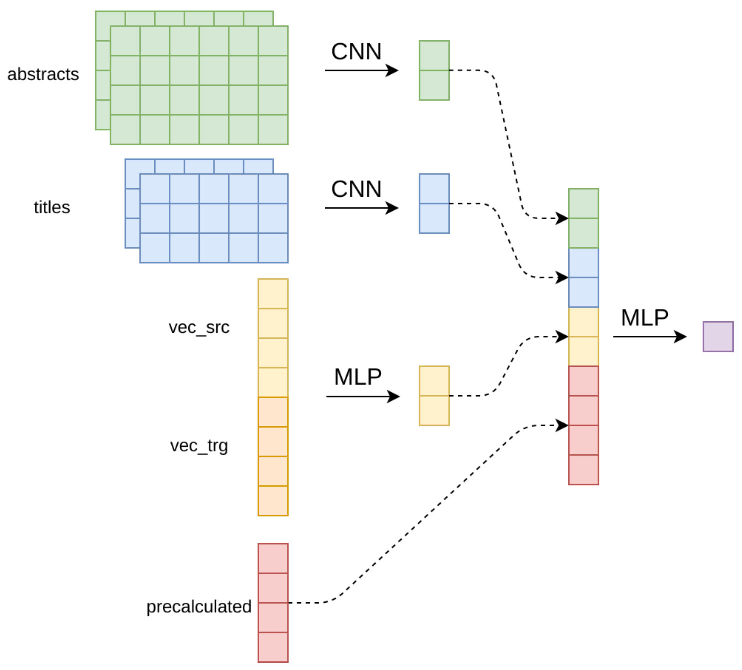

Figure 4: Structure of neural network

Figure 4 demonstrates the structure of our convolutional neural network. Each title was truncated and padded into a word list of length 10 and each abstract was truncated and padded into a word list of length 150. Then, for title, we used 16 filters of size 4 with strides 2 and 16 filters of size 2 with strides 1. The output dimension is 16. For abstract, we used 32 filters of size 8 with strides 4, 64 filters of size 4 with strides 2, and 32 filters of size 1 with strides 1. The output dimension is 32. For each type of embedding vector, the MLP only contains one layer of 64 units. The MLP after concatenation contains one hidden layer of 128 units and the output is a real number. All the activation functions are set to rectified linear unit (ReLU) except the output layer of the last MLP whose activation is sigmoid. The learning rate is 0.005.

A.5 Classifier comparison

We compared the following classifiers with same inputs: Logistic Regression, Linear SVM, Non-linear SVM, Multi-layer Perceptron, Random Forest, LightGBM and XGBoost. After parameter tunning, their performances are shown in Table 1. For non-linear SVM model, the training is so slow that it didn't finish after one hour. Therefore the results aren't available. From the table, we can observe that XGBoost has the best performance and Linear SVM has the poorest performance, but they all have a quite good performance which proves the high quality of our features. Considering the difference of performances, it's quite reasonable since the data must be inseparable and those tree based models are more powerful than Linear SVM.

Classifier	**F1 Score**
Logistic Regression	0.96477
Linear SVM	0.95548
Non-linear SVM	/
Multi-layer Perceptron	0.96885
Random Forest	0.96947
LightGBM	0.97171
XGBoost	0.97631

Table 1: Model comparison

A.6 Feature importance

Table 2 lists part of feature importance derived from our best model (private F1 score 0.97715).

order	feature	importance
1	year_diff	1.27e-01
2	art_aai_all	1.11e-01
3	nbrs_d2v_mean	9.82e-02
4	art_salton_contrib_hop2	7.63e-02
5	art_pa_all_hop2	5.76e-02
6	art_pa_all	5.52e-02
7	art_salton_all_hop2	4.63e-02
8	art_pa_ref	4.63e-02
11	tfidf_abs	3.41e-02
15	d2v	1.95e-02
18	tfidf_title	1.14e-02
19	n2v_author	1.14e-02
20	comm_auth	1.14e-02
22	n2v_journal	1.06e-02
29	abs_w2v_mean_wiki	4.87e-03
39	title_w2v_mean_wiki	8.12e-04
40	overlap_title	8.12e-04
44	comm_journal	0.00e+00

Table 2: Feature importance