ALTEGRAD Challenge

LACOMBE Yoach
MVA ENS Paris Saclay

MERCIER Marine
MVA ENS Paris Saclay

Abstract

Link prediction is predicting the existence of a link between two entities in a network. This problem has recently attracted a lot of attention in many domains. For instance, in social networks, one may be interested in predicting friendship links among users, while in biology, predicting interactions between genes and proteins is of paramount importance. Here, we focus on citation network, we want to determine whether a research paper cites another research paper. We used a typical pipeline of classification problem, the goal is to use edge information to learn the parameters of a classifier and then to use the classifier to predict whether two nodes are linked by an edge or not.

1. Introduction

We are interested in predicting whether a research paper cites another research paper. To do so, we were given papers published at ML, AI, data mining and NLP journals, with their abstracts, their authors and a sub-part of the citations. We were asked to infer missing citations between research papers. We identified the challenge as a graph missing link prediction tasks. Two main difficulties in this task are the sparsity and huge size of the target networks.

1.1. Dataset description

The graph nodes studied here are research papers, and two nodes are linked if a paper quotes on another. Please also note that the citation network was presented as an undirected graph, meaning that we lose the information on which paper cites the other in a pair of nodes. Usually citation networks are directed.

1. **Edge list:** a citation network represented by a graph: 138,499 papers as nodes and 1,091,955 citations as edges.
2. **Abstracts list:** The abstracts associated to the 138,499 papers.
3. **Authors list:** The authors associated to the 138,499 papers.
4. **Test list:** A list of 106,692 pair of nodes. We're asked to predict if there is an edge between the two elements of each pair of nodes.

In order to train a classifier on edges of a graph, we need negative examples. Therefore we create a **Negative edge list**, a list of pair of nodes which are not connected. Since the graph has 138,499 nodes, we cannot enumerate all the negative node pairs, therefore we pick randomly some edges which do not exist in the graph.

The choice of the number of negative examples to use is important for the performance of our model. Ideally we would like to reach a ratio between positive and negative edges similar to the one of the test set. Given that we don't know the composition of the test set, we would like to have as many negative edges as possible, so that our model can train on more data.

We tested different sizes of **Negative edge list**, the largest being 3 275 865 edges, which is three times the number of edges in the positive graph. However, it seems that we get better results with fewer negative edges. This means that the test set is probably balanced between positive and negative edges. We also tried to train our models using weights per class, but the performance of the algorithm wasn't significantly better, while the computational time was considerably longer.

1.2. Evaluation of the performance

The performance of our models are evaluated on the binary-cross entropy (BCE) loss, where the label is 1 if there is an edge between a given pair of papers and 0 otherwise.

1.3. First intuitions

Intuitively, a paper quotes another paper if both papers talk about the same specific area. In a broader point of view, one paper has more chances to cite another paper if :

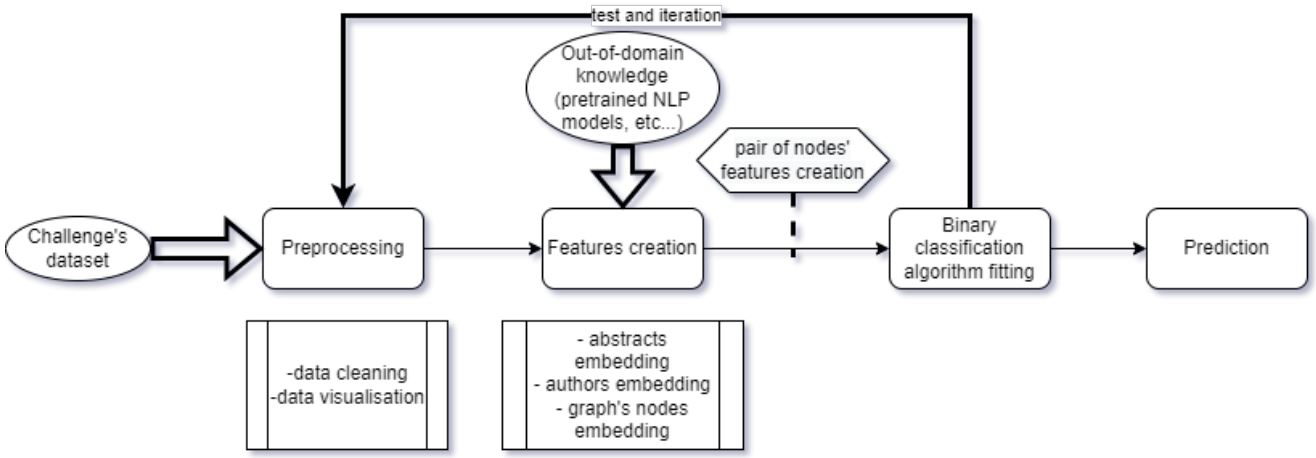

Figure 1. Challenge's pipeline.

- The subjects of the two research papers are similar or related, the subject is defined in the abstract;
- If they quote similar authors or papers;
- If a paper is a reference in the field and is widely used and cited;

1.4. Our pipeline

We implemented a classic challenge pipeline to deal with this challenge. See figure 1 for a resume, the main steps being the preprocessing of the dataset, the per-node features creation, the per-edges features creation and the fitting of a binary classification algorithm.

The rest of this paper will principally explain the full pipeline, with a special focus on features creation. This features creation mixes knoweldge and algorithms learned on the ALTEGRAD's course, with NLP and graph-based techniques, as well as classic data science concepts.

2. Abstract embedding

Abstract embedding is an important part of our pipeline. Its interest is two-fold. Indeed, it uniquely distinguish a paper and it embeds a notion of similarity between papers. So, it can be used as a node label/embedding as well as a direct similarity measure between two papers. The abstract embedding task falls in the NLP framework of document/paragraph embedding, the goal being to represent a text document with a numerical fixed-size vector, regardless of the document's length.

Apart from simple abstract features that were given as a baseline (number of common words, length of the abstract), we used two document embedding techniques to capture the semantic and meaning of the abstracts.

2.1. Doc2Vec

Doc2Vec [4] is a simple extension of the famous Word2Vec [6] technique. Word2Vec is a word-embedding algorithm that learns a word embedding thanks to the context surrounding the word in real-life sentences. In Doc2Vec, the document is also represented as a vector and given to the context of the word.

To be more precise, we used an extension of CBOW (where previous or surrounding words' embeddings are used to predict the current word) called PV-DM 2, in which the document embedding is trained alongside the words' embeddings.

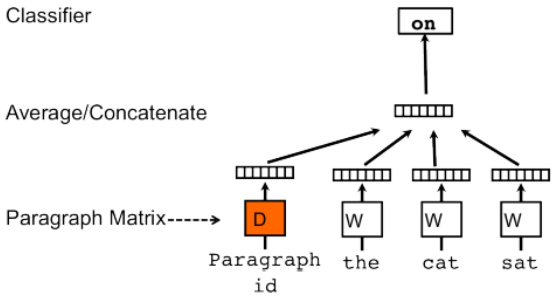

Figure 2. Visual illustration of the doc2vec's PV-DM approach. From https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

Here, Doc2Vec was trained with the Gensim Python library directly on the abstracts.

2.2. Universal Sentence Embedding (USE)

Universal Sentence Embedding [1] is a more recent technique (2018) than Doc2Vec (2014). USE is a Transformer-based encoding method, which relies on context aware rep-

resentations of word in a sentence. Here USE encode the documents onto a 512 dimensional vector that can be used for a variety of NLP tasks such as supervised classification, question answering or documents similarity.

The main advantages of USE are its ease of use (thanks to Tensorflow Hub) and its pretraining on a variety of web sources such as Wikipedia or web news. We decided not to fine-tuning it, as it would require too much computational power and as its performances were satisfactory enough, thanks to the out-of-domain knowledge and semantic expertise USE has acquired through its learning.

3. Authors embedding

We tried different authors embeddings. The first naive approach was using an hot encoder, which performed badly. Moreover, given the number of authors, this methods uses lot of memory.

Then we built an author graph, were nodes would be authors. There is an edge between two authors if a paper of an author quotes the paper of the other. The weight of the edge is equal to the number of citation between the two authors. Once again, this graph is undirected. We tried two methods on this author graph:

- We used a Node2Vec embeddings. First we generate sequences of nodes with a random walk, starting from each nodes of the graph. Then a Word2Vec model is trained with these sequences. We use the output of Word2Vec as node embeddings.
- We also tried to identify the field of study of each author by adding features to each nodes. We chose authors features to be the average of the doc2vec embedding of abstracts of all papers the author wrote. Then we trained a GraphSAGE model (see details later) on a graph using these node features. The GraphSAGE model then gives author embeddings.

Both methods were not conclusive, so we chose simpler algorithm, such as the number of common authors, the weighted number of common authors (weighted by number of papers in which the author appears), or the popularity of all the authors appearing in both articles.

4. Node embedding

Node embedding is a way of representing nodes as vectors, which can capture the topology of the network and give structure information. These embeddings are also useful for link prediction, given that an edge is a pair of nodes. We would like two linked nodes to have a high similarity of their embedding vectors. We tried different nodes embeddings based on the previous experiences with abstracts and authors.

4.1. Simple nodes features

The first naive node embedding would be to concatenate all information we could get about each node, including information extracted from the abstract and from the authors graph. Then we added some additional features, such as the number of neighbors and the number of authors.

4.2. Embedding with unlabeled graph techniques

We thoroughly studied unlabeled graph techniques specially used for missing link prediction tasks thanks to benchmarks, notably [5] and [2], with techniques such as VERSE [8] or LINE [7].

However, we faced some serious difficulties in evaluating these techniques with regard to train/test-split, as we found the output embeddings of such models heavily relied on the edges distribution of the train graph. For example, most of nodes of the citation network are connected and splitting the edges between train and test edges are making the graph highly unconnected, which makes the embedding more difficults. Moreover, the computational and time expenses were too high to be able to iterate quickly. We found that the best way of dealing with such difficulties was to use a labeled graph algorithm instead, as it also relies on the nodes' labels to predict a link.

4.3. GraphSAGE

GraphSAGE [3] is a framework for node embedding on large graph. this method works best when nodes that reside in the same neighborhood have similar embeddings. GraphSAGE learns the parameters of K aggregator functions, which get information from a sample of the node close neighbors. At each iteration, or search depth, the features from local neighbors are aggregated, and as this process iterates, nodes incrementally gain more and more information from further reaches of the graph.

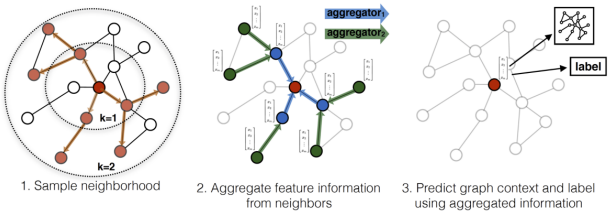

Figure 3. Visual illustration of the GraphSAGE sample and aggregate approach.

GraphSAGE is inductive, contrary to most embedding frameworks, which means that once the aggregator weights are learned, the embedding of an unseen node can be generated from its features and neighborhood 3. In this specific challenge, a transductive approach would have worked, but in a real world situation, citation networks keep evolving over time as new research papers are continuously published. Transductive frameworks do not efficiently generalize to unseen nodes and evolving graphs.

We used the DGL library which proposes an implementation of the GraphSAGE layer. Our model has two GraphSAGE layers, and provides embeddings of nodes. We train the GraphSage model with a multi-layer perceptron predictor. However, we can use the nodes embeddings obtained with other classifiers.

5. Edges Embedding

Each of the previously mentioned embedding methods were thus used to produce a node embedding by concatenating the different embeddings (USE, Doc2Vec, SAGE and more simple features).

Once we had these node embeddings, we computed the edges embedding in two manners. For the MLP classifier, we simply passed concatenated the embedding of the pair of nodes and passed it to the classifier. For the XGB classifier, we computed similarity measures (L2 distance, L1 distance and cosine similarity) between the two embeddings, we also added some direct similarity measure between edges of an unlabeled graph, such as Ressource Allocation Coefficient, Jaccard Index, or Adamic Adar.

6. Classifier

Now that we have an edge-embedding pipeline, we can build a binary classification dataset with positive (when there is a link) and negative (when there isn't) labels. Note the special treatment for negative edges, as we randomly sample pairs of edges and treat them as negative samples. We then trained classifiers to sort positive and negative edges. Remark that the whole goal of the pipeline was to build a tabular dataset, for which a lot of classifiers have been developed.

We compared two classifiers, multilayer perceptron (MLP) and an XGBoost classifier. The MLP is composed of three linear layers, with dropout. The output is transformed into a probability with Sigmoid. XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. This algorithm uses bagging, random forests and gradient boosting. It is specially suited for tabular datasets, since it is both really fast and efficient. We also studied Logistic Regression as a baseline at the beginning of our study.

7. Tests and results

7.1. Testing procedure

To test our method, we first created a negative edges list by randomly sampling pairs of nodes and by discarding them if there were already in the positive edges list. The ratio of negative to positive number of edges was set to around 1, as we found our SAGE embedding quality was downgraded with an higher ratio.

We then tested and iterated our method on a 70 percent train/test split of edges (keeping the same ratio of negative to positive edges).

7.2. Results and comments

Table 1 summarizes our progresses. Note that the best model here had a public score of 0.19360 which is quite impressive, especially since it was trained on 70 percent of the dataset.

We hadn't had any conclusive results by using second-order similarity measures such as Adamic Adar and Ressource Allocation Index on our testing pipeline, however we found it gave use excellent results. Our interpretation is that these are measures that heavily relies on the graph structure, and that the graph structure is impacted by the train/test split. Indeed, for example, we found that the number of connected components of the full graph was 57 whereas it was 1598 with the test graph, meaning testing on a subpart of the graph with second-order similarity measures could only lead to overfitting.

In figure 4, you'll find the features importance, given thanks to XGBoost API. As expected, the similarities given by the SAGE algorithm are the most importance, surely because they encompass both the semantic similarity of the abstracts and the graph neighborhood similarity. The popularity of the authors, i.e the sum of the number of times each author appears in other papers, seems also important. This fact can be explained by the existence of central papers, i.e popular papers that are cited hundred/thousands of times.

All in all, our best model, trained on the whole dataset, led to a public score of 0.12626.

8. Conclusion

In this paper, we proposed a pipeline addressing a link prediction task. We relied on NLP and graph-based paradigms in order to solve it. It is important to point out that our solution, whilst achieving really satisfactory performances, is scalable and should be compatible with a grow-

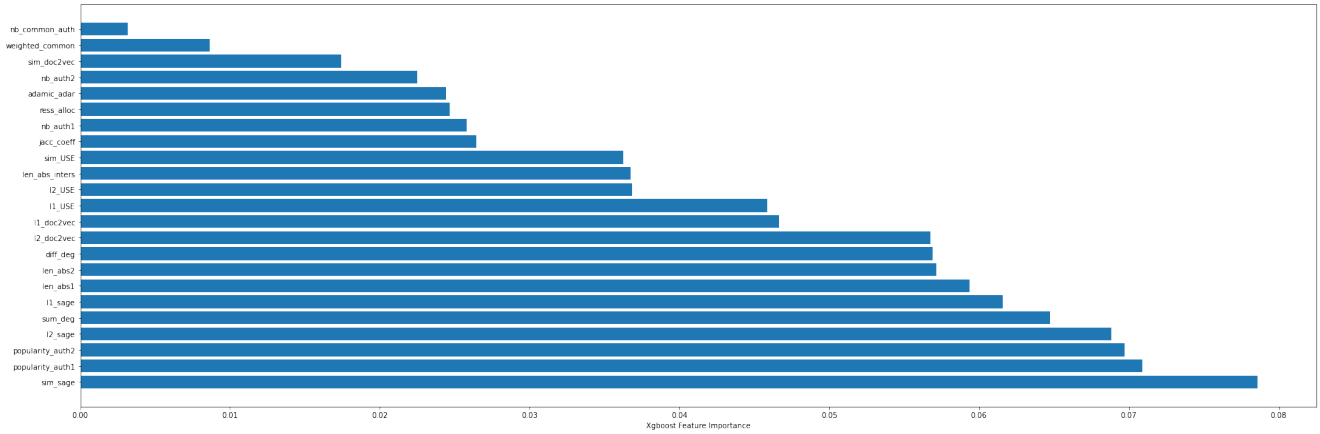

Figure 4. Features importance by "weight".

Features used	Accuracy	Log Loss	AUC
Authors based features	0.776	0.501	0.629
+ baseline features	0.898	0.253	0.940
+ USE embedding similarity	0.919	0.208	0.960
+ Doc2Vec embedding similarity	0.931	0.178	0.971
+ SAGE embedding similarity	0.961	0.110	0.989

Table 1. Results on train/test split with 70 percent of the edges on the train set.

ing graph with new unseen nodes, thanks to the GraphSAGE algorithm. Our next line of work could have been to improve the abstract embedding and to work on a better integration of the authors list.

References

[1] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018. 2

[2] Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, Vedang Patel, Balaraman Ravindran, and Srinivasan Parthasarathy. Network representation learning: Consolidation and renewed bearing. *CoRR*, abs/1905.00987, 2019. 3

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. 3

[4] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014. 2

[5] Alexandru Mara, Jefrey Lijffijt, and Tijl De Bie. Network representation learning for link prediction: Are we improving upon simple heuristics? *CoRR*, abs/2002.11522, 2020. 3

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. 2

[7] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. *CoRR*, abs/1503.03578, 2015. 3

[8] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: versatile graph embeddings from similarity measures. *CoRR*, abs/1803.04742, 2018. 3