Kaggle Competition: Predicting Missing Citations

Assignment 2: Network Science Analytics Course - Fall 2018 - Ecole CentraleSupelec, Paris

Team Ben_Dev_Rai_Saa

Adrien BENAMIRA
Ecole CentraleSupelec
Paris, France
adrien.benamira@supelec.fr

Benjamin DEVILLERS
Ecole CentraleSupelec
Paris, France
benjamin.devillers@supelec.fr

Ayush K. RAI
Ecole CentraleSupelec
Paris, France
ayush.rai2512@student-cs.fr

Manal SAADI
Ecole CentraleSupelec
Paris, France
manal.Saadi@supelec.fr

In this work we address the task of link prediction in a citation network. This work is also a part of an in-class Kaggle Competition for Network Science Analytics Course offered at Ecole CentraleSupelec, Paris in Fall 2018-2019. Our final F-score is 0.973 on the public test set and we are currently ranked 3 / 46. You can find our work on github.

1 FEATURE ENGINEERING

In the recent years, the research on link prediction problem in complex networks has captured much attention of researchers [2] and [6] from various disciplines principally because many available real-world networks are incomplete. The problem of Link Prediction falls into the category of supervised machine learning, however a critical ingredient of designing such models is feature engineering. Our approach extensively captures textual features, graph-theoretical features, unsupervised graph neighborhood similarity based features and meta information based features. In the following section we perform a thorough and detailed analysis of the various features we used to tackle the problem.

1.1 Analysis of Core Features

In this section, we will analyze the core features used to reach our best performance on the Kaggle Platform.

- **Title Overlap**: This feature aims at finding number of overlapping words in the titles of the source and target publication nodes in the graph. This captures the notion that two publications that have a highly overlapping title might belong to the same research domain and therefore have a high likelihood of being linked to each other in the citation network.
- **Common Authors**: This feature calculates the number of common authors between the source node and the target node. A high number of common authors would intuitively indicate that the source and target research publication have been published by authors working in the same research field or maybe working in the same research lab itself and thus there is a high chance of having a citation link between source and target publications.
- **Temporal Difference**: This feature represents the difference in the publication year of the source and target publication. On its own this feature is not very informative however it becomes highly important in the presence of the above two features Title Overlap, Common Authors and other features. For example source and target publication with highly overlapping title and high number of common authors with lesser difference in the publication year have a high chance of citing each other.
- **Number of Inclusive Edges**: The purpose of this feature is to calculate the total number of times a target node has been cited in the training dataset. This idea of measuring target node citation frequency is equivalent to total in-degree for that target node. In other words through this feature we aim to measure the importance of a node in the citation network through in degree centrality criteria.
- **TFIDF_Distance_Corpus**: In order to extract the textual features, we first computed the term frequency inverse document frequency matrix for the entire corpus of the abstract using [3] and subsequently extract tf-idf features corresponding to source and target abstracts. Linear Kernel methods was used to estimate the closeness between source and target tf-idf abstract vectors.
- **TFIDF_Distance_Titles**: This feature is identical to the TFIDF_Distance_Corpus feature, where tf-idf matrix is created for corpus of title in the dataset instead of abstracts. And linear kernel based distance is calculated between the source title tf-idf vector and target tf-idf vector.
- **Distance Abstract** : Here we used a pretrained word embedding model GloVe [4], to estimate the vector representation for the entire source and target abstracts using average of word level GloVe embeddings over all the words in the abstract. Subsequently euclidean distance is computed between source abstract vector and target abstract vector. Lesser the euclidean distance is, higher is the chance of presence of link between source and target node.
- **Shortest_Path_Dijkstra**: This features determines the shortest path between source and target node in the directed version of the citation network using Dijkstra's algorithm. The

intuition behind capturing this information is that it measures the minimum geodesic distance between source and target node. It is worth noting that while computing this feature, if there already exists a directed connection between the source and target node then that connection is deleted before computing the shortest path.

- **Shortest_Path_Dijkstra_Undirected**: Here we consider the undirected version of the citation network and measure the shortest path (geodesic distance) between the source and target nodes of publication. Here also it is worth noting that while computing this feature, if there already exists an undirected connection between source and target node then that connection is deleted before computing the shortest path.
- **Common Neighbors**: The objective of this feature is to capture the number of common neighbors (node) in the citation graph for the source and target publication. The intuition behind this feature is that the nodes which share common neighborhood in the graph eventually become connected over a period of time.
- **Edge_Disjoint_Path**: This feature captures the number of edge disjoint paths between the source and target. Edge disjoint paths are paths that do not share any edge. The number of edge disjoint paths between source and target is equal to their edge connectivity.
- **Jaccard_Similarity_Undirected**: The Jaccard similarity coefficient of two nodes is the number of their common neighbors divided by the number of nodes that are adjacent to at least one of them. In other words, in order to calculate Jaccard Similarity we simply divide the number of common neighbors by the number of total neighbors.
- **Resource_allocation**: For each given node pair u and v, we compute the resource allocation index between them using the following formula:

$$\sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(w)|}$$

where $\Gamma(u)$ denotes the set of neighbors of u.

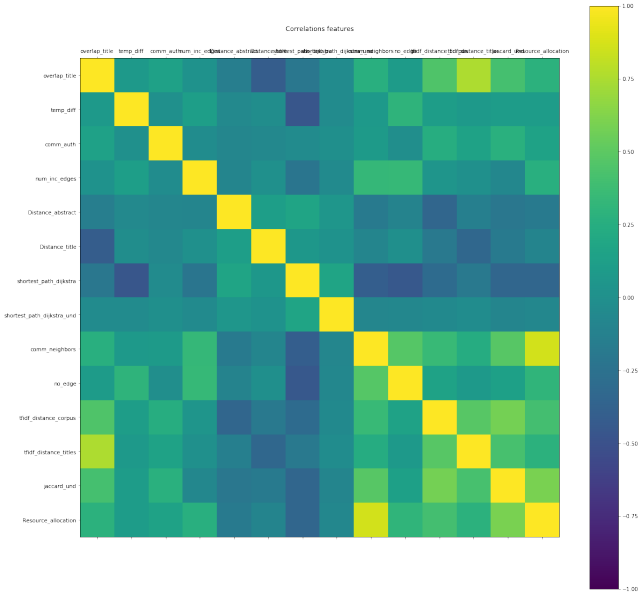

Figure 1: Covariance matrix

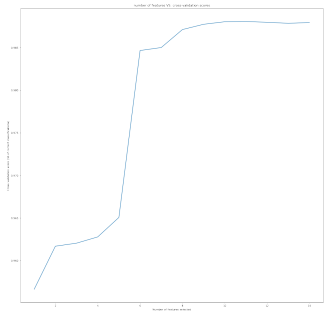

Figure 2: Number of features VS. cross-validation scores

1.2 Features selection

Next, we needed to select the best features to achieve maximum accuracy. For that we compute first the covariance matrix (fig. 1). Then, to know how many features to use, we used the recursive feature elimination with Random Forest (fig. 2). Then, we plot the most important features for Random Forest (fig. 4) and Gradient Boosting (fig. 3).

After doing this, we decide to use the features: Title Overlap, Common Authors, Distance abstract, Shortest_Path_Dijkstra, Shortest_Path_Dijkstra_Undirected, Common Neighbors,Edge_Disjoint_Path, TFIDF_Distance_Corpus, TFIDF_Distance_Titles. It leads to a 91.76 score. After we gradually removed or added features, one by one and it lead to these optimal features:

Temporal Difference, **Common Authors**, **Distance Abstract**, **Shortest_Path_Dijkstra**, **Shortest_Path_Dijkstra_Undirected**, **Common Neighbors**, **TFIDF_Distance_Corpus**, **TFIDF_Distance_Titles**, **Resource_allocation**, **Number of Inclusive Edges**

2 MODEL TUNING AND COMPARISON

First, we computed an overall training set of 10% of the total links available in the graph and normalize it.

2.1 Model Comparison

We use a Gradient Boosting Regressor from the lightgbm library (best model), and we use Scikit-Learn SVM, Random Forest, Regularized Logistic Regression, implementation with the standard hinge loss as suggested by the baseline.

For the training part, we train on 99% of the dataset and eval on the rest. We made the cross validation 25 times. Each time we evaluate the model on the train and val set after convergence and we do the mean. Results are in fig. 2.1.

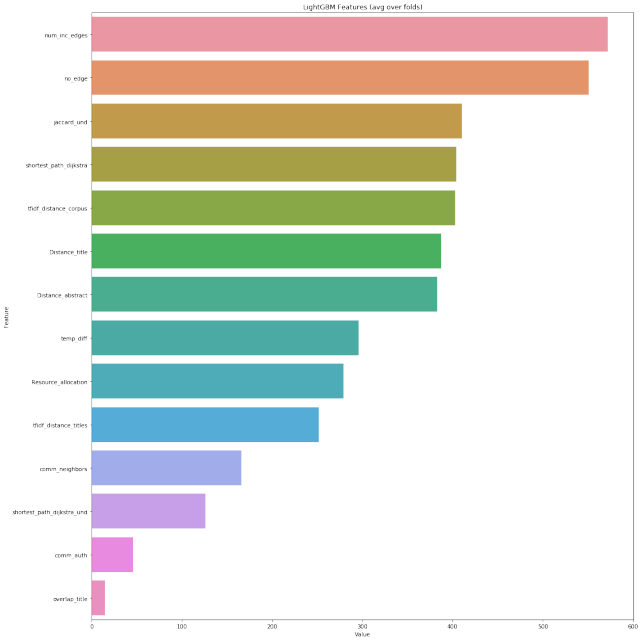

Figure 3: Gradient Boosting Feature importance

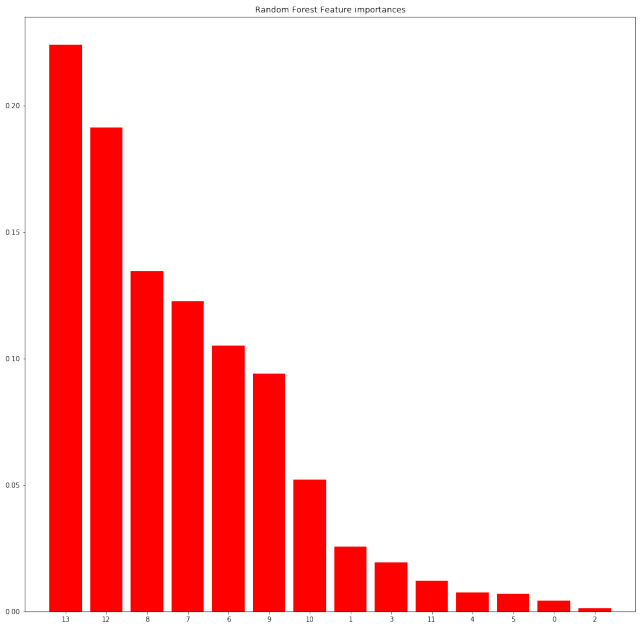

Figure 4: Random Forest Feature importance

Model	Train	Validation
Gradient Boosting	0.979	0.976
Random Forest	1	0.975
SVM	0.964	0.964
Linear	0.966	0.966

The best model is Gradient boosting, despite the optimization parameters done in Model_tunning.ipynb

2.2 Parameter tuning and Preventing over-fitting (V2)

When we trained the four models, we tuned their parameters and toke in consideration the over fitting problem.
For linear SVC and logistic regression, we tuned dual parameter for selecting if we wanted to either solve the dual or primal optimization problem. The tolerance parameter for stopping criteria is among [1e-4,1e-3,1e-5] and C the penalty parameter among 'C': [0.1, 1, 10], we selected the best parameters using Gridsearch. In order to avoid over fitting, we opted for an L2 regularization for the penalty parameter and a good selection of the penalty parameter criteria C, in addition to the stopping criteria.
For Random Forest model, we used the default parameters and selected the best number of estimators among [100, 500, 1000] with Gridsearch. Considering that a high number of estimators can reduce the over fitting by averaging among an important number decision tress, we can say that the number of estimators was used as a way for avoiding over fitting.
For the LightGBM model, as the first model showed strong over-fitting, which we eventually reduced by using L2-regularization (grid search for hyperparameter tuning), and early-stopping. The number of estimators used is very high (10,000), but the training is stopped as soon as the validation score does not improve for 50 iterations (final results used âĹij130 estimators).
Each time, the cross validation was used in the grid search and it helps to limit the over fitting.

2.3 Going Further

Analyse where we get wrong and add new features. We also aim to explore representation learning based methods for graphs like Node2Vec[1] and DeepWalk[5].

REFERENCES

[1] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[2] Longjie Li, Shenshen Bai, Mingwei Leng, Lu Wang, and Xiaoyun Chen. 2018. Finding Missing Links in Complex Networks: A Multiple-Attribute Decision-Making Method. *Complexity* 2018 (2018).

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[4] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[5] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[6] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. *arXiv preprint arXiv:1802.09691* (2018).