MLNS - Kaggle Challenge Report
Link prediction in information networks

Gladys Roch / Aurélien Houdbert
Team name : Gladys$_A urlien$

March 2021

1 Introduction

We participated in a private Kaggle competition on link prediction in information networks.

1.1 Challenge presentation

The network of interest is a citation network. It is a graph where nodes are research papers and there is an edge between two nodes if one of the two papers cite the other. There is a total of 27,770 nodes.

In addition to the graph structure, each node (i.e., article) is also associated with information such as the title of the paper, publication year, author names and a short abstract.

The objective is to accurately predict missing edges in this network using graph-theory and node information. In particular, for any given node pair/pair of papers, we need to tell whether they are linked (1) or not (0). This task is therefore a classification task.

1.2 Evaluation metric

The evaluation metric of the competition is the F1 score, defined by the geometrical average of recall and precision :

$$F1 = 2\frac{p \cdot r}{p + r} \quad \text{where} \quad p = \frac{tp}{tp + fp}, \quad r = \frac{tp}{tp + fn}$$

1.3 Data

To train our model we have access to a subset of 615,512 node-pairs from the network, with the information on whether the two nodes of each pair are linked by an edge or not. There are 335,130 linked pairs (54.4% of the set) and 280,382 un-linked pairs.

We also have a csv file containing the information on all the 27,770 nodes of the network. The textual information (title, abstract) is already prepossessed to lowercase, and punctuation marks as well as common English stopwords have been removed.

The test set comprises 32,648 pairs from a subset of 23,402 nodes from the network.

2 Methodology

We split the training set into training and validation sets with the ratio 0.2 (492,409 nodes-pairs for training and 123,103 for testing). The whole set was already balanced (54.4%), therefore we used *stratify* to preserve this ratio in the training set.

We first created multiple features from the node information and the graph structure was built using *networkx* and all the data available (27,770 nodes and 335,130 edges). Then, we analysed the engineered features to select the most suitable. Finally we tested and fine-tuned several machine learning models to get the best performances possible.

To evaluate our models we used the following metrics :
- accuracy
- specificity
- sensitivity
- f1 score

3 Feature engineering

We used the baseline features and created additional ones to leverage the graph structure.

3.1 Features based on node information

– Difference in years of publication

This features computes the temporal distance between papers. In particular, a paper cannot cite a paper not yet published. Although, in the network there are 440 instances of two papers referencing each other.

– Number of common words in the title and in the abstract

These 2 features bring together papers dealing with similar topics. Number of common words in the abstracts is computed after preprocessing to remove stopwords and accents.

– Number of common authors

Authors know their papers and tend to write new papers in the line the of previous work, therefore referencing it.

3.2 Features based on graph theory

Let $\Gamma(x)$ be the set of neighbors of a node x in the graph.

– Number of common neighbors

$$CN(x, y) = |\Gamma(x) \cap \Gamma(y)|$$

– Number of common in and out neighbors

Same definition as the CN but taking into account the directions of the edges. $CN_{in}(x, y) + CN_{out}(x, y) = CN(x, y)$

– Jaccard Coefficient

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

gives the probability that x and y have common neighbors.

– Adamic/Adar

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

gives larger weight to common neighbors of low degree.

– Preferencial Attachement

$$PA(x, y) = deg(x).deg(y)$$

– in Degree of target node

$$inD(x, y) = in\_deg(y)$$

– Shortest Path

$$SP(x, y) = -\text{Dijkstra(source=x, target=y)}$$

– Personalized Katz measure

Because nor the *networkx* implementation of Katz centricity, nor the dual form of the Katz measure converged on the training set, We defined a simpler Katz measure accounting only for paths of length 2 to 4 between two nodes.

$$Katz(x, y) = \sum_{l=2}^{4} \beta^l |paths_{x,y}^{<l>}|$$

In practice, we used the adjacency matrix (A) of the graph to compute this Katz measure : $Katz(x, y) = \sum_{l=2}^{4} \beta^l |A^l[x, y]|$.

The graph at hand is a directed graph, however some feature give interesting information when computed on the equivalent undirected graph.

3.3 Features selection

3.3.1 Correlations

To choose the right features to feed to our models we first look at the correlation between the features and with the target (existence of an edge). The correlation matrix shown in figure 1 clearly shows that Common Neighbors and Adamic/Adar metrics are completely correlated. We need to make sure to include only one of the two features in order to avoid over-fitting and poor model training. Moreover, as can be seen in the last column of the matrix, some features are more correlated to the label (existence of an edge) than others. In particular the shortest-path metric is fully correlated to the label, this is because the feature was build from the graph including all the edges label 1 corresponds to shortest path of length 1... Because shortest path of length more than 1 are accounted for in the Katz measure we did not rebuild the shortest path metric but discarded it from our feature pool.

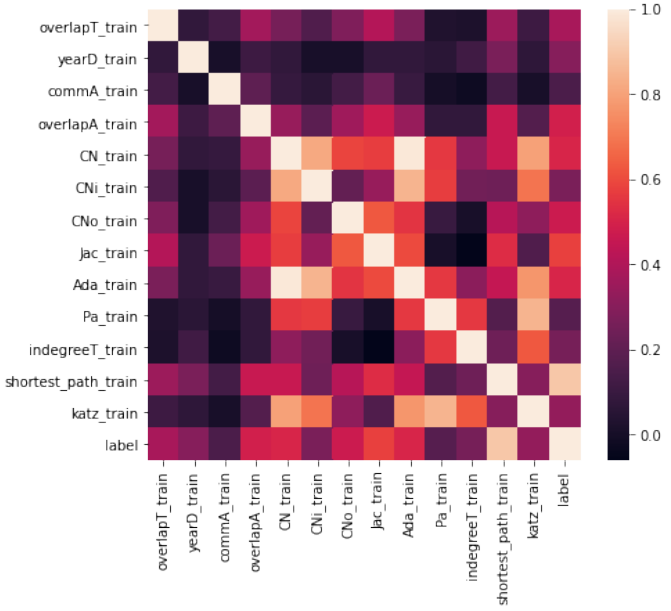

FIGURE 1 – Correlation matrix of the features computed on the training set of edges and the full graph (27770 nodes, 335130 edges

3.3.2 Feature importance

The importance of each variable can be measured through multiple metrics. We combined feature-selection algorithms based on :

– the Pearson correlation,

We compute the Pearson's correlation between the label and the features and we keep the features with highest correlation.

– the chi-squared metric,

We compute the chi-squared metric between the label and the features and we keep the feature with highest score.

– Recursive feature elimination (RFE),

This method uses and estimator, e.g. a logistic regression. The best features are select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained through a coefficient attribute. Then, the least important features are discarded from current set of features. That procedure is recursively repeated until we reached the desired number of features.

– Embedded feature-selection.

Some classifier have built-in feature selection methods. We use RandomForest to select the best features for us. Because these algorithms are fast, we allow ourselves to use all of them to have a more comprehensive feature selection method. We used this scheme to choose between two correlated variable : Number of common neighbors and Adamic-Adar. The results are presented in the table bellow :

Feature	Pearson	Chi-2	RFE	RF	Total
Comparing the 3 Common Neighbors features					
CN	True	False	True	True	3
CN In	False	False	False	False	0
CN Out	False	True	False	False	1
Comparing CN and Adamic-Adar					
CN	True	True	True	False	3
Adamic-Adar	False	False	False	True	1

Table 1 – Feature selection

Given these results we choose to include the Common Neighbors metrics over the other common neighbors features and the Adamic-Adar metric. Therefore we use a total of 9 features to train our models. The list of the feature included is :

– Number of common words in the titles
– Number of common words in the abstracts
– Number of common authors
– Difference in year of publication
– Number of common neighbors
– Jaccard coefficient
– Preferential Attachement
– in-degree of target node
– Katz mesure

4 Models implemented and fine tuning

4.1 Classifiers

We implemented a range of machine learning classifiers using the *sklearn* library. Bellow is the list of classifier studied.

– Linear SVM
– Naive Bayes
– Linear Regression
– Random Forest
– Gradient Boosting

Because of the large number of training samples, it is preferable (for convergence) to use the LinearSVC implementation of *sklearn* over SVC with linear kernel. Moreover, gaussian kernel resulted in convergence failure.

To manage NaN values from the Katz measure we built a sklearn Pipeline that imputes missing value with the mean of the feature vector.

4.2 Fine Tuning

To fine-tuned the models we use Grid Search on the parameters.

LinearSVC is best suited for large dataset. We grid-searched over penalty : ['none', 'l1', 'l2'], and C : [1, 5, 10, 20]. The best estimator was obtained for C=5 and l1 penalty.

Naive Bayes does not have tuning parameters.

For Linear Regression we followed the recommendation of the sklearn documentation on large datasets : we used the 'saga' solver and grid search the best penalty (['none', 'l1', 'l2', 'elasticnet']) and the C value (np.logspace(-2,2, num=10)) with a maximum of 1000 iterations. Several combinations of parameters failed to converge. The best estimator was obtained for C=0.01, penalty = l1.

For Random Forest we focused on two parameters, maximal depth of the tree and number of estimators and we set a random state to ensure reproducible results. max_depth : [50, 100, 200] n_estimators : [2, 5, 8, 15] The best estimator was obtained for max_depth=15 and n_estimators=50.

For Gradient Boosting, we followed the same grid-search as the RF tuning : n_estimators : [50, 100], and

max_depth : [2, 15, 30] The best estimator was obtained for max_depth=15 and n_estimators=100

5 Training and Kaggle Results

In table 2 are presented the performance on the testing set of each model after fine-tuning.

Two models stand out : Random Forest and Gradient Boosting.

Model	Acc	Bal Acc	f1 score	Prec	Recall
Lin SVC	0.956	0.956	0.958	0.935	**0.983**
NB	0.866	0.880	0.861	0.766	0.982
Lin Reg	0.928	0.935	0.937	0.978	0.899
RForest	**0.977**	**0.977**	**0.979**	0.979	0.979
Boosting	0.977	0.977	0.979	**0.980**	0.977

TABLE 2 – Validation metrics of the best estimators resulting from grid search with the 9 features selected (Section 3.3.2)

We compared our best two models, Random Forest and Gradient Boosting, on the Kaggle set of node-pairs. The final Kaggle results for these two models are shown in the following table. The difference of performance of the 2 models are not significant but we give the advantage to our Random Forest model due to a much faster training than Gradient Boosting (60sec vs 12min on full training set)

Model	F1 score (MLNS 2021-2)	
	trained on 492,409 nodes pairs	trained on all nodes pairs (615,512)
Random Forest	0.96731	**0.96777**
Gradient Boosting	0.96563	0.96513

6 Conclusion

During this kaggle competition we built a link prediction algorithm based on a random classifier. We obtained a f1 score of 0.968 on the challenge test set. To build our solution, we engineered and used features based on the graph structure as well as on the node information / content. We tested multiple classic machine learning classifier and compared their results in order to identify the best one and use it for our final submission.