MLNS Course - Kaggle Report
Team : Ramamonjy_Dargier_Douilly

Antoine Dargier
antoine.dargier@student-cs.fr
SDI mention - CentraleSupélec

Thomas Douilly
thomas.douilly@student-cs.fr
SDI mention - CentraleSupélec

Johary Ramamonjy
johary.ramamonjy@student-cs.fr
SDI mention - CentraleSupélec

ACM Reference Format:
Antoine Dargier, Thomas Douilly, and Johary Ramamonjy. 2023. MLNS Course - Kaggle Report Team : Ramamonjy_Dargier_Douilly. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

This report relates about the work we achieved during the Kaggle challenge, which was part of the MLNS course. This challenge consists in a link prediction problem on a leaky graph given as input data. This graph contains a list of actors (as nodes) linked by a bunch of edges designed to represent co-occurence of two of them in a single Wikipedia page. Edges were randomly removed, and the goal of the challenge is to find them.

Practically, this input data consists in :

- A file "train.txt" containing a list of couple of nodes caracterized by a 1 if an edge exists between these ones and 0 otherwise.
- A file "node_information.txt" containing, for each node of the graph, a list of 932 features encoding the information of the actor's wikipedia page.

2 GRAPH EXPLORATION

To have a good comprehension of the graph we are working on, we have created the graph from the training set, and here are the most relevant pieces of information about this graph:

- Number of nodes: 3597
- Number of edges: 5248
- Clustering coefficient (measures how closely connected the nodes in a graph are to each other): 0.020
- Diameter (the longest shortest path between any two nodes): 19
- Number of connected components: 1

We can see here that we have one connected graph, where the nodes in the graph are not very closely connected to each other. This means that the graph is relatively sparse, and there are not many triangles or other small subgraphs in the graph. It can be now interesting to look at the degree distribution to see how the nodes are connected:

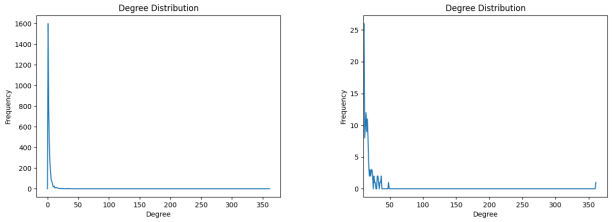

Figure 1: Degree Distribution

We can see from the degree distribution that 44% of the nodes have only one relation, and 97% of the nodes have less than 10 relations. Moreover, all nodes have less than 37 relations, except one central node that has 361 links.

One other important thing to look at is whether nodes in the test set are part of the graph (from the train set) or not. In fact, if they were isolated during the cuts, we can't use our similarity properties anymore. We need to use other techniques, such as kNN for example using the nodes' information to classify them. After a quick view, we have seen that all nodes in the test set are part of the graph, so we will not have this problem.

3 FEATURE ENGINEERING

The easiest way to compute link prediction on graphs is to implement similarity measures between nodes or edges and use them in a classifier. We will give you here a list of the features [2], [3] we used or tested and what they represent. After, we will explain how we have chosen them:

- Degree centrality: the importance or centrality of a node in a graph. In undirected graphs, degree centrality is usually just referred to as degree, and it measures the number of edges that are connected to the node
- Betweenness centrality measure: for a node, it is defined as the number of shortest paths between pairs of nodes in the graph that pass through that node. Here we will compare this measure between the two nodes of the edge
- Preferential attachment: it is computed as the number of neighbours of u times the number of neighbours of v. It encoded that nodes that are already well-connected are more likely to receive new connections, while nodes with few connections may struggle to attract new connections.
- Adamic Adar index: it is a measure of the similarity between two nodes in a network based on the number of shared neighbours they have. It takes into account the importance of each shared neighbour, based on their degree in the network.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

- Jaccard coefficient: it is used to measure the similarity between the sets of neighbours of two nodes, dividing the intersection of neighbours sets by the set union.
- Length of the shortest path: It is of course the length of the shortest path. In the case where two points are linked, we remove the link and look for the shortest path if it exists.
- Node information: for that features, we have decided to use different similarity measures and compare their performances, to see whether it is relevant to use this information or not, and to choose the best measure. Among all possibilities, we have chosen to study the mean average error, cosine similarity, Person correlation, euclidean distance and Manhattan distance.

To choose the best features, we compute the correlation matrix:

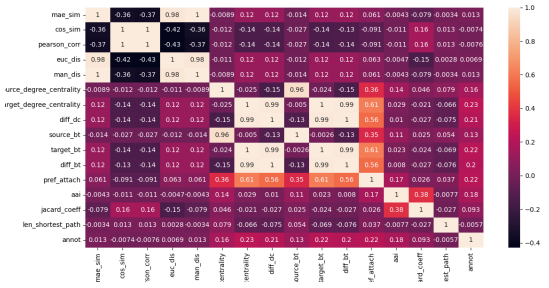

Figure 2: Correlation matrix

From that matrix, we can see that the nodes' information doesn't have any correlation with the annotation: it seems so not very useful for our prediction task. For the other features, the length of the shortest path is not relevant for the classification. Betweenness centrality and degree centrality are very correlated, and as we have an undirected graph, we need not have source and target nodes. One idea to overcome this problem can be to use the minimum and maximum degree centrality, and the same for betweenness centrality. Trying that, we didn't achieve better results so we keep our features as degree centrality for source and target nodes and the difference of betweenness centrality. To overcome the problem of undirected edges, we compute the features in the two ways (for a node (u,v): from u to v and v to u).

4 CONSTRUCTION OF OUR SOLUTION

During this challenge, we tested a wide range of model designs in order to get the best output accuracy. In particular, we can distinguish two main areas of research that will be considered separately in the following sub-sections: Model Tuning and Parameters fine-tuning.

4.1 Model tuning

. Considering that taking into account the environment of the graph (in particular node features) and all the particularities of the given data, we decided to focus only on supervised methods. Indeed, unsupervised methods seemed for us a little useless in our case, because it would have forgotten all the "real" part of this problem. We try to implement Deepwalk and Skipgram method [1], but both didn't achieve very good results. They are two unsupervised learning methods coming from NLP, and they learn from the patterns and relationships present in the input data. When we saw that the results were not as good, we chose to focus on machine learning methods, where we could better control the features and parameters.

. For this purpose, we tested five different models of supervised learning, namely :

- **Logistic Regression**
- **Random Forest**
- **SGD Classifier**
- **k-Neighbors**: We compute the number of predictions for each class (True or False), the final prediction is the predominant one.
- **Gradient Boosting Classifier**

However, with all these methods, it appears that a small part of the points was predicted with difficulties. Indeed, with quasi-50/50 probabilities, the different methods did not seem to be able to decide whether there is a link between two nodes or not.
Therefore, in order to increase the accuracy of the whole algorithm, we decided to combine two methods among the ones presented before. We introduced a threshold: we first start to apply the first method and compute both predictions and the associated probability estimates. For each pair of nodes, we will keep the prediction if the probability is upon the threshold. Otherwise, we will apply the second method.

4.2 Parameters tuning

. For each of these models, we identified a set of parameters for which we achieved parameters tuning. To do this, we shuffled our training dataset (both data and link annotations) into a new training and a validation set (The size of the validation set is $\frac{1}{10}$-th of the size of the given dataset). Then, for each of the tested parameters, we tried a range of values and calculate for each of them the number of links of the validation dataset that have been badly predicted.

. Here are the parameters we tested for each of the supervised models :

- For the Logistic Regression, we tuned the maximum number of iterations, as well as the threshold of tolerance : as we could have expected, a lower value of the maximum number of iterations or a higher value of the threshold lead to worse performances (the regression has not enough time to converge). On the contrary, a higher value of the maximum number of iterations or a lower value of the threshold (compared to the pre-initilized values of the scikit-learn package) does not change the final accuracy on the validation set, as the convergence is already reached initially.
- For the Random Forest method, we tried to tune the number of estimators. Similarly to logistic regression, an increase of the value of this parameter did not seem to have any impact on the overall accuracy of the model.
- For the SGD Classifier, we tried to tune the maximum number of iterations, as well as the value of alpha, the constant

linked to the regularization term. Neither of these parameters seemed to have a real impact on the overall accuracy, so that we decided to only keep the pre-initialized ones.
- For the K-Neighbors algorithm, we chose to tune the number of neighbors. Increasing this number often lead to a small improvement of the accuracy of the whole model, which is not so significant.
- For Gradient Boosting, the n_estimators from sklearn.ensemble. GradientBoostingClassifier parameters was changed. Due to lack of submissions and very small improvement when tuning parameters, it would have been more relevant to focus on the feature extraction than on the model parameter tuning. However, it gave already similar result compared to other classifier without tuning the parameters.

In addition to these parameters, we also tried to tune the threshold presented in the last section. This threshold allowed us to mix two different algorithms. Similarly to the previous experiments, we tried different values of this parameter. Considering Logistic Regression as the main method and k-Neighbours method as our secondary one (ie. the one used when the prediction probabilities of the first one are lower to the threshold), we were able to find that an optimal value at 0,62, which split our data with a 81%/19% rate. Practically speaking, it means that around 81% of the training data was classified using Logistic Regression, the remaining being classified using k-Neighbors. This technic was also tested with other combinations of methods but never succeeded as well as it did in this case.

Finally, this method of fine-tuning allowed us to reach a pretty good overall accuracy, improving the results we could have obtained with the pre-defined values of parameters included in the basic implementation of each of the methods in Python (in particular from the scikit-learn package). Even though it has been quite time-consuming and we could not have tested all the possible values for each parameter, the results are nevertheless pretty good and should not be linked to any potential phenomenon of over-fitting, as we confirmed it on different random shuffling of our dataset.

5 RESULTS

. Using the different methods and hyper-parameters determined in the previous sections, we were able to state that - except the SGD classifier - all our methods reached a respectable accuracy set between 0.75 and 0.76 in Kaggle, which is a clear improvement with respect to the random baseline, which lies around 0.5.

. As explained in the previous part, testing different values for each parameter, did not allow us to improve significantly our score. However, a 0.62 - threshold allowing us to use Logistic Regression as a basis and k-Neighbours if the Regression did not work properly allowed us to reach our final best score of 76.29%.

6 CONCLUSION

. Finally, this project was very interesting and challenging because it was quite new to work on graphs. Through our efforts, we were able to identify the most relevant features for predicting links on graphs and select the best machine learning models for the task. This project has taught us the importance of feature engineering and model selection in machine learning. It is crucial to carefully choose the features that are most relevant to the task at hand and to experiment with different models to identify the one that performs the best. Then we experimented with different models and optimized the models' performance by fine-tuning their parameters and achieved promising results with an accuracy of about 76%.

This project shows us particular problems that we encounter when we deal with graphs which are very particular. We need to understand well the structure of the graph and the nature of the links to model our graph well. We see that we very specific graphs, it can be very difficult to have very good results because some links can be unpredictable, or information can be lost if e.g part of the graph is no more part of the same connected component.

About the models, it could have been interesting to test new methods with Deep Learning, which performs very well for a few years, GAT (Graph Attention Network) [4] in particular. This state-of-the-art model has very impressive results. However, we have seen that the performances of our models are always between 74 and 76%, so we can think that a part of links is very difficultly predictable and Deep Learning methods will not perform much better.

REFERENCES

[1] Pedro Almagro-Blanco and Fernando Sancho-Caparrini. 2019. Improving skip-gram based graph embeddings via centrality-weighted sampling. (2019). DOI: 10.48550/ARXIV.1907.08793.
[2] Ece C. Mutlu and Toktam A. Oghaz. 2019. Review on graph feature learning and feature extraction techniques for link prediction. *CoRR*, abs/1901.03425. http://arxiv.org/abs/1901.03425 arXiv: 1901.03425.
[3] Mohammad G. Raeini. 2020. Link prediction using supervised machine learning based on aggregated and topological features. *CoRR*, abs/2006.16327. https://arxiv.org/abs/2006.16327 arXiv: 2006.16327.
[4] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks, (Feb. 2018).