NGSA: Kaggle Challenge

Arthur CLAUDE, Armand MARGERIN, Louis TERNON,
Thomas DELRUE

Team Name on Kaggle: AAA

March 2020

1 Introduction

The goal of the challenge was to predict missing links in a citation network of research articles. A citation network is represented as a graph G = (V, E), where the nodes correspond to scientific articles and the existence of a directed edge between nodes u and v, indicates that paper u cites paper v. 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. A number of edges have been randomly removed from the original citation network. Our goal was to accurately reconstruct the initial network using graph-theoretical and textual features, and possibly other information.

In this report, we will present the work performed for this challenge and the finally proposed solution with its results.

2 Feature Engineering

As explained before, the goal is to predict, for a couple of nodes (articles), if there is an edge between those nodes. The first step for the completion of the challenge was to create several features associated to a couple of nodes. These features have been built from the elements studied in class and after reading several research papers [1] [2] [3] [4] [5] [6].

2.1 Textual Features

In the data studied, each article is associated with textual information as the title of the paper, the authors names and a short abstract. From this information, we can extract several features.

- **title_commonwords**: The number of common words in the titles of the two articles. Stopwords are not taken into account because they don't

provide any information. It is a basic but efficient measure of similarity between the subjets of two articles.

- **abstract_commonwords**: The number of common words in the abstracts of the two articles.
- **abstract_similarity** [4]: The cosine similarity of the TF-IDF vector representations of the abstracts. We use TF-IDF instead of Bag-of-Words to give more weight to rare terms.

2.2 Graph Features

Then, it is possible to represent our data in a graph where each article is a node and to extract graph-based features.

- **degree_source** [1]: The degree of the source node. We believe that "popular" nodes are more likely to be linked to other nodes.
- **degree_target** [1]: The degree of the target node.
- **triangle_source** [5]: The triangles of the source node.
- **triangle_target** [5]: The triangles of the target node.
- **clustering_source** [1] [2]: The clustering coefficient of the source node.
- **clustering_target** [1] [2]: The clustering coefficient of the target node.
- **common_neigh** [1] [4] [5]: The number of common neighbors of the nodes.
- **pr_source** [1] [6]: The pagerank of the source node.
- **pr_target** [1] [6]: The pagerank of the target node.
- **adamicadar_index** [1] [3] [5]: The Adamic Adar index of the nodes. Very efficient measure of similarity based on the number of common neighbours, but assigns more weight to common neighbors that are rare.
- **ressourceallocation_index** [3] [5]: The Ressource Allocation index of the nodes. Mesure of similarity similar to the Adamic Adar index.
- **jaccard_coef** [1] [3] [4] [5]: The Jaccard coefficient of the nodes. The metric not only takes number of common nodes into account, but also normalizes it by considering the total set of number of shared and non-shared neighbors.
- **pref_attachment** [3] [5]: The preferential attachment score of the nodes: node prefer to form ties with popular nodes.

We tried to implement other features, which, according to the articles, had a strong discriminative power. These features are:

- **simrank** [1] [3] [6]: The Simrank similarity of the nodes.
- **katz** [1] [3] [6]: The Katz similarity of the nodes.
- **cosine_sim** [1]: The cosine similarity of the nodes.

However, their computation was much too long (several days), so we left out these features.

2.3 Other Features

Finally, we can add two other features:

- **year_difference** [4]: The difference between the source year of publication and the target year of publication. Usually researchers tend to cite recent articles.
- **common_authors** [4]: The number of common common authors of the two articles.

3 Model Tuning and Comparison

After extracting all the previously introduced features, we looked for the model best suited to the problem.

3.1 Random Forest

First, we tried a **Random Forest Classifier** without parameter tuning. It gave a score of 0.98619 on Kaggle.

We then searched for the optimal parameters using a method of grid search cross-validation with **GridSearchCV**. In order to make the search reasonably long in time, we realized it on a subset of 10000 observations.

The tested parameters are:

```
parameters = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Figure 1: Tested parameters for Random Forest

The best parameters found are:

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 400}
```

Figure 2: Best parameters for Random Forest

They provide the following results:

split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	r
0.969406	0.975998	0.978698	0.9747	0.003903	

Figure 3: Results for Random Forest

The fact that the variance is close to zero allows us to make sure that we won't have overfitting problems.

The selected model gave a score of 0.98646 on Kaggle.

For this model the importance of the different features are: They provide the following results:

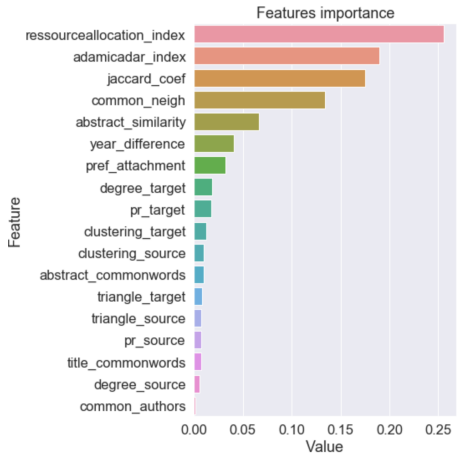

Figure 4: Features Importance for Random Forest

3.2 Gradient Boosting

We then tried to train a **Gradient Boosting** model, with the **LightGBM** library.

The tested parameters are:

```
parametersLGBM = {
    'learning_rate': [0.005, 0.01],
    'n_estimators': [8,16,24],
    'num_leaves': [6,8,12,16],
    'boosting_type' : ['gbdt', 'dart'],
    'objective' : ['binary'],
    'max_bin':[255, 510],
    'random_state' : [500],
    'colsample_bytree' : [0.64, 0.65, 0.66],
    'subsample' : [0.7,0.75],
    'reg_alpha' : [1,1.2],
    'reg_lambda' : [1,1.2,1.4],
    }
```

Figure 5: Tested parameters for LightGBM

The best parameters found are:

```
{'boosting_type': 'dart',
 'colsample_bytree': 0.64,
 'learning_rate': 0.01,
 'max_bin': 510,
 'n_estimators': 24,
 'num_leaves': 8,
 'objective': 'binary',
 'random_state': 500,
 'reg_alpha': 1,
 'reg_lambda': 1.2,
 'subsample': 0.7}
```

Figure 6: Best parameters for LightGBM

They provide the following results:

split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score
0.958008	0.969697	0.973297	0.967	0.006527

Figure 7: Results for LightGBM

The fact that the variance is close to zero allows us to make sure that we won't have overfitting problems.

The selected model gave a score of 0.97506 on Kaggle.

For this model the importance of the different features are:

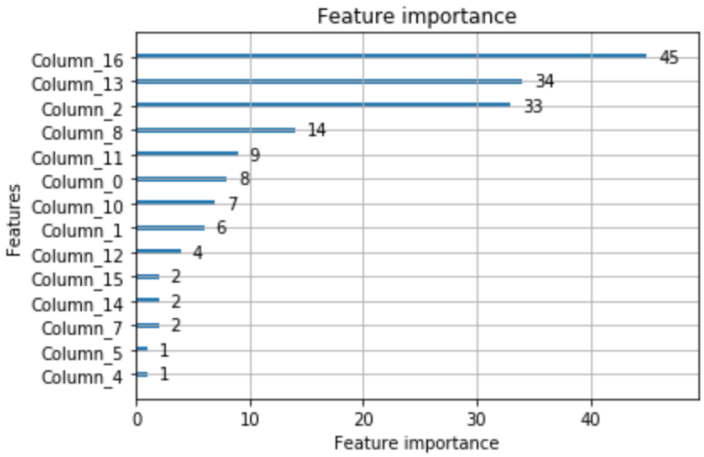

Figure 8: Features importance LightGBM

Where Colum_16 = year_difference, Column_13 = ressourceallocation_index, Column_2 = abstract_similarity, Column_8 = clustering_target, Column_11 = common_neigh.

3.3 Support Vector Machine

We then tried to train a **Support Vector Machine** model.

The tested parameters are: For this model the importance of the different features are:

```
parametersSVM = {'C':[0.1,1,10,100,1000,10000],
                 'gamma':[100,10,1,0.1,0.001,0.0001],
                 'kernel':['linear','rbf']}
```

Figure 9: Tested parameters SVM

The best parameters found are:

```
{'C': 10000, 'gamma': 0.001, 'kernel': 'rbf'}
```

Figure 10: Best parameters SVM

They provide the following results:

split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score
0.967007	0.973297	0.972097	0.9708	0.002727

Figure 11: Results for SVM

The fact that the variance is close to zero allows us to make sure that we won't have overfitting problems.

The selected model gave a score of 0.98731 on Kaggle.

For this model, the importance of the different features is not accessible.

3.4 Logistic Regression

We then tried to train a **Logistic Regression** model.

The tested parameters are:

```
parameterslogreg = {'penalty':['l1', 'l2'], 'C':np.logspace(-3,3,7)}
```

Figure 12: Tested parameters for Logistic Regression

The best parameters found are:

```
{'C': 100.0, 'penalty': 'l1'}
```

Figure 13: Best parameters for Logistic Regression

They provide the following results:

split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score
0.963107	0.971197	0.969397	0.9679	0.003468

Figure 14: Results for Logistic Regression

The fact that the variance is close to zero allows us to make sure that we won't have overfitting problems.

The selected model gave a score of 0.98256 on Kaggle.

3.5 Conclusion

We finally obtain the following results with our different models:

Classifier	F1 score on Kaggle
Random Forest Classifier	0.98646
LightGBM	0.97506
SVM	0.98731
Logistic Regression	0.98256

The best model for the F1-score is therefore the **non-linear SVM** with the parameters selected by **GridSearchCV**. However, this model takes a long time to adjust to the training data, thus, the **Random Forest Classifier** appears to be the best time/precision compromise.

References

[1] W. Cukierski, B. Hamner, and B.Yang. Graph-based features for supervised link prediction. *Proceedings of International Joint Conference on Neural Networks*, pages 1237–1244, 2011.

[2] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. 2006.

[3] E. Mutlu and T. Oghaz. Review on graph feature learning and feature extraction techniques for link prediction. *Proceedings of ACM*, 2018.

[4] N. Shibata, Y. Kajikawa, and I. Sakata. Link prediction in citation networks. *Journal of the American Society for Information Science and Technology*, 2012.

[5] Z. Wu, Y. Lin, J. Wang, and S. Gregory. Efficient link prediction with node clustering coefficient. *Physica A: Statistical Mechanics and its Applications*, 2015.

[6] M. Zhang and Y. Chen. Link prediction based on graph neural networks. *32nd Conference on Neural Information Processing Systems*, 2018.