**COMP90051 Statistical Machine Learning**

**Project 1 Report**

**Willon Chun Yin Lai 1075581**

**Introduction to problem**

The purpose of this project is to predict missing edges whether they exist among test node (author) pairs in an *academic co-authorship network*. They are often missing edges between nodes in a network caused by deficiency or bug in data collection. This project targets to train a machine learning (ML) model with suitable feature selection and approaches to predict future co-authorships. The model predict test edges with probability that represents the confidence that the edge is real, and the test will be used to generate an Area Under the Curve (AUC) of Receiver Operating Characteristics (ROC) for measuring the performance.

**Description of features**

Feature selection is the most critical part of any ML algorithm. In the link prediction problem, we have to choose features that are able to represent similarity of edges or the node pairs, and features can be divided into three categories:

(1) *Proximity features* are measuring how closely related of two nodes in the graph, the “nodes.json” file has provided several features of each node, we extracted five features as node pairs, including (i) difference in first publication year, (ii) difference in last publication year, (iii) difference in number of papers, (iv) same keyword count, (v) same venue count. Hasan et.al [1] suggested that, the larger the size of the intersection in proximity features, the more likely they are, to work in related areas and higher chance to be a future co-author pair.

(2) *Aggregated features* are used to convert two aggregate scores of single nodes into a pair of nodes [2]. For each aggregation, the summation of aggregation function *score(x, y) = aggregateScore(x) + aggregateScore(y*). Three features can be extracted from “nodes.json”, such as (vi) sum of keywords reflects likelihood to work with new authors, (vii) sum of papers and (viii) sum of venues reflects likelihood of co-authoring a paper in the future based on the prolific of the author pairs.

(3) *Topological features* are generated from graph structure itself for future link prediction. We generated the co-authorship graph by Python library NetworkX, which shows the graph has 4016 nodes and 80621 edges. (ix) Shortest path computes the smallest number of steps between two nodes, (x) common neighbours counts number of mutual neighbours in the graph, (xi) sum of neighbours is the summation of neighbours in both nodes, (xii) Jaccard’s coefficient is a commonly used similarity metric measures the probability that both *x* and *y* have some feature given that either *x* or *y* has the feature [2]. (xiii) Adamic Adar measures similarity between two nodes by weighing “rarer” common neighbours more heavily [3], (xiv) resource allocation similarity index is defined as the amount of resource node *x* receives from node *y* through indirect links [4], (xv) neighbourhood distance reflect the likelihood of link formation between the nodes *x* and *y* [5], (xvi) preferential attachment indicates new links will be more likely to connect higher degree nodes than lower degree nodes. (xvii) maximum page rank and (xviii) minimum page rank are the same core algorithm used by Google to rank search results, page rank of a node in the graph is proportional to the probability that the node will be reached through a random walk on the graph [3], (xix) hub score predicts the node value based on outgoing links and (xx) authority score predicts the node value based on the incoming links in HITS algorithm [6].

**ML approaches**

1. *Support Vector Classifier (SVC)*

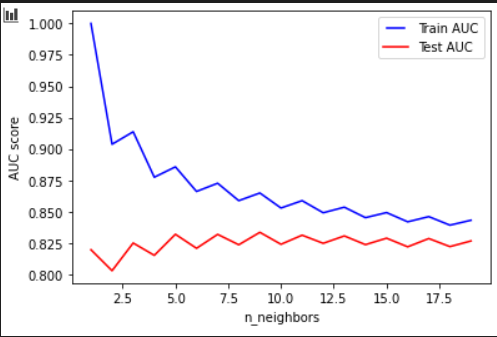
It is a classification method that tries to find a hyperplane in an N-dimensional space (classes) by maximizing the distance between samples and hyperplane. We have implemented the SVC with rbf kernel, where rbf is a kernel commonly used in nonlinear hyper-plane and it is suitable for multiple features and classes. It has more choices of penalties and loss functions. In scikit-learn library, it is handled according to a one-vs-one scheme.

1. *Logistic Regression*

It is a simple ML classifier, which used to predict the probability of a categorical dependent variable. In a link prediction problem, the dependent variable is a binary variable that contains result coded as 1 (totally confident of a real edge) or 0 (zero confidence of a real edge), so it requires only binary classification in logistic regression. Regarding to the scikit-learn library, the parameter solver ‘liblinear’ is applied as it supports both L1 and L2 regularization to avoid overfitting, and one-vs-rest (OvR) is applied then a binary problem is fit for each label.

1. *K-Nearest Neighbours (KNN) Algorithm*

KNN is an instance-based learning algorithm, which is easy to train and require less training time compared to other algorithms. The KNN classifier is implemented in the scikit-learn library. The k value (i.e. the parameter of n\_neighbors) has been tested to examine the optimal AUC, increasing the number of neighbours improves the test scores.



1. *Multinomial Naive Bayes classifier (*MultinomialNB)

The MultinomialNB is suitable for classification with discrete features. It requires integer feature counts vectors for generating multinomial distribution, all features must be converted to absolute value. We can use the scikit-learn library to build this classifier, with tuning the parameter ‘alpha’ to additive smoothing.

1. *Random Forest Classifier*

A random forest is a meta estimator that fits a number of decision tree classifiers on a number of slice of samples in the dataset. In the model parameters, n\_estimators represents the number of trees in the forest, while more trees may have better result. Meanwhile, min\_samples\_split and min\_samples\_leaf are important features that control the minimum number of samples at the leafs and samples required to split in node. Max\_depth represents the depth of each tree in the forest, The deeper the tree, the more splits it has and it captures more information about the data. Using the GridSearchCV in the scikit-learn libaray, *max\_depth = 50* is the optimal value in this dataset.

1. *Decision Tree Classifier*

Decision tree classifier is similar to random forest classifiers, it breaks down a data set into smaller and smaller subsets, while both models have the same parameters. Decision Tree is relatively faster in running time as the structure is simpler. By applying GridSearchCV, we can find the optimal parameters with *max\_depth=50, min\_samples\_leaf= 10, min\_samples\_split = 100*.

**Model selection**

The selection is based on the ease of training of the model and tuning hyperparameters in each model. Although SVC can result in a relatively high accuracy, model accuracy is 0.93, the AUC for test split data is 0.96, average precision and recall are both 0.92, but it is computationally expensive for large dataset. It took over 5 hours for training 100 thousands samples, the complexity is O(n2 \* m) where n is number of samples and m is number of features. Therefore, it is not relatively efficient than other ML model.

Next, KNN and MultinomialNB are not able to achieve a high accuracy score, which are peaked at 0.82 and 0.74, the AUC of test split data of these two models are 0.91 and 0.86 respectively, during the submission on Kaggle, the AUC score dropped to below 0.7. Although the ‘K’ value in KNN was tried on different values ranging from 1 to 30, it failed to get a good performance in AUC, which it does not try to optimize any effectiveness measure, the average precision is 0.83. For MultinomialNB, the average precision is the lowest among all models, which score 0.8. Multiple features often contain similar signals in the dataset, such as aggregation features, the classifier’s conditional independence assumption results in its behaviour of features as distinct signals, so it produce classification descent accuracy but not able to classify the confidence of the test data. They are probably underfitting and these models are not capable to train this dataset very well.

For logistic regression, the result on Kaggle is better than KNN and MultinomailNB, reaching above the AUC score of 0.88. The training took a reasonable time at around 3 minutes, the average precision of test split data is 0.93 and the model accuracy is also 0.93. This is a possible candidate for final choice. The performance is not excellent but relatively stable.

Finally, random forest classifier and decision tree are both using the same hyperparameters after the optimal was found as mentioned above. Decision tree classifier can score above 0.89 on Kaggle, while Random forest classifier is able to pass 0.9. It is interesting that although both models can score the same in model accuracy, random forest classifier is able to predict more correct edges, the precision of it is 0.94 compared to 0.93 in decision tree classifier. Therefore, although decision tree classifier is easier to train, the complexity of random tree classifier is able improve the predictive accuracy and control over-fitting. As a result, random forest classifier is chosen as the final approach.

Problem encountered

One of the biggest challenge of this project is to collect negative samples, which is not provided in the dataset. During the first 10 attempts on Kaggle, all possible edges were collected, the AUC is below 0.7. Since the dataset had serious imbalance samples, there were 53800 positive samples but more than 500 thousands negative samples. Afterwards, the way of reducing negative samples was to randomly pick node pairs that are not connected in the graph and the shortest path between both nodes are larger than 2, and negative samples are restricted to the number of positive samples. This solution drastically improve the performance, the AUC increased to above 0.8. This highlighted the importance of balanced data and soling the imbalance problem.

Another factor is the train test split process in cross validation stage, it requires numerous training with different percentage of test split to improve the AUC. Since there are 100 thousands samples, the test size of 0.3 reached the highest AUC score of 0.91 on Kaggle with the final choice of model.

Furthermore, more topological features are required to improve the AUC score, especially in a network graph. Without topological features, the AUC score is not able to reach 0.75 with random forest classifier. When more topological features, particularly shortest path, it increased the AUC and model accuracy.

**Reference**

[1] M. Hasan et.al. *Link Prediction using Supervised Learning*. [Online]. Available at: <https://archive.siam.org/meetings/sdm06/workproceed/Link%20Analysis/12.pdf> [Accessed 23 April 2020]

[2] Q.Lim et.al. *Social Network Analysis: Supervised Link Prediction*, 9 December , 2017.

[3] Pavlov et.al. *Finding Experts by Link Prediction in*

*Co-authorship Networks*.

[4] T. Zhou, L. Lu, Y.-C. Zhang. *Predicting missing links via local information*. Eur. Phys. J. B 71 (2009) 623. <https://arxiv.org/pdf/0901.0553.pdf>

[5] Yang, J. & Zhang, X.-D. *Predicting missing links in complex networks based on common neighbors and distance*. Sci. Rep. 6, 38208 (2016).

[6] Jon Kleinberg, *Authoritative sources in a hyperlinked environment Journal of the ACM,* 46 (5): 604-32, 1999. doi:10.1145/324133.324140. http://www.cs.cornell.edu/home/kleinber/auth.pdf.