COMP90051 Statistical Machine Learning - Project 1 - Team DNS

Diego Aranda (992038) Nandita Susarla (104447) Shaik Anisuzzaman (1060370)

1 Introduction

This report provides an analysis of 4 classification models applied to the link prediction problem in the context of academic paper co-authorship networks. We built Logistic Regression, Artificial Neural Network (ANN), XGBoost, and LightGBM classifiers, all using a combination of node similarity metrics and node embeddings as features. The aim was to compare these models in terms of AUC performance. All model configurations were run on an Apple M1 machine with 8 CPU cores, 8 GPU cores and 16 GB of RAM.

2 Data

Data was downloaded from COMP90051's Kaggle competition page. The data was provided as an undirected hyperedge structure where each row consists of co-authors of an academic paper. The hyperedge data from each row was processed e.g. as seen below, row 2 shows the edges resulting from the collaboration of the authors in row 1. The entire dataset was translated into 16,087 edges representing a relationship between a pair of co-authors. 3,816 nodes or authors were identified. The weight for each edge was also calculated based on how often the edge was encountered in the dataset.

Row from original dataset: 0 1 2 3 4
Our representation of the data: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

3 Data Processing

We provide an overview of the data sampling procedure we used to generate positive and negative edges and the feature generation process used to build the overall feature set. The result of this data processing stage was used as input to train and test the models evaluated in this report.

3.1 Edge sampling

In order to address the link prediction problem as a binary classification problem, the data needs to be sampled in order to construct training data. The training data needs to consist of a combination of *positive edges* (edges in the graph) and *negative edges* (edges not in the graph). There are various methods to choose these edges as explained below.

3.1.1 Random Sampling

This method involves randomly choosing nodes from the graph that share a link, to form the set of positive edges. Similarly, the negative edges are formed by randomly choosing nodes from the graph that do not share a link.

3.1.2 Entire dataset

We also tried using the entire graph for positive edges and randomly sampled negative edges. This resulted in better performance as the classifiers had the full dataset of positive edges for training. However, this approach most certainly led to overfitting in some of the classifiers.

3.1.3 Sampling to retain graph structure

Instead of randomly selecting positive edges a better approach is to remove edges that do not affect the number of connected components and number of nodes in the graph to maintain graph structure. The best model performance was achieved by using this strategy.

3.2 Feature Generation

3.2.1 Node similarity measures

We considered four node similarity measures [1] as shown in Table 1, to be used as features in order to predict the likelihood of future links between co-authors (x,y), where Γ(x) represents the neighbours of node x:

Common Neighbours	Jaccard's Coefficient	Adamic-Adar	Resource Allocation
$CN = \Gamma(x) \cap \Gamma(y)$	$JC = \frac{\lvert\Gamma(x) \cup \Gamma(y)\rvert}{\lvert\Gamma(x) \cap \Gamma(y)\rvert}$	$AA = \sum_{k \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log \lvert\Gamma(k)\rvert}$	$RA = \sum_{k \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\lvert\Gamma(k)\rvert}$

Table 1: Formulae for node similarity metrics

We experimented with different combinations of these metrics on the implemented models, and it was found that the use of Common Neighbours and Resource Allocation led to overfitting. Thus, we finally used Jaccard's Coefficient and Adamic-Adar measures for our models. The graph metrics were calculated using Python's NetworkX package [2].

3.2.2 Graph structure and Node embeddings

To enrich the feature space we implemented node embeddings. The Node2Vec library [3] creates vector representations of the neighbourhood of each node using a random walk algorithm. The node embeddings resulted in a further 30 features. Thus, we used 32 features in total for the machine learning task.

3.3 Training and Validation sets

The best training performance was obtained by generating a subgraph from the original graph. A small number of the positive edges from the original graph was used for the validation set. These edges were chosen according to Section 3.1.3. These removed edges formed the positive samples for our validation set and these edges were dropped from the original graph, thus leaving a subgraph of the data. The edges within the subgraph were then all used as the positive edges of the training dataset. The subgraph was used to generate the node embeddings and the graph metrics as features. We consider this to be a superior method to construct a validation set since it retains the original graph structure with minimal impact on training performance. In addition, we did not use the entire original graph, but rather the subgraph to train the classifier and hence reduce the likelihood of overfitting.

4. Algorithms

A summary of the algorithms and parameters used along with the AUC results obtained can be seen in Table 2.

4.1 Logistic regression

This model is a classification algorithm that predicts a binary variable based on one or more independent variables [4]. We used a Grid Search algorithm to obtain the best parameters for this baseline model, which resulted in a L2 regularizer and C = 100 for inverse regularization strength. Although this model did not apply a high amount of regularization, we did not obtain a high result as the linear nature of the model does not handle the sparsity of the data hence leading to overfitting.

4.2 Artificial Neural Network

ANN mimics the behaviour of human neurons, which can be activated by receiving signals from other neurons and then passing them to others. We performed a parameter optimization algorithm by leveraging the Talos [5] framework, and we arrived at 4 hidden layers, all with 512 neurons activated with ReLU, using Stochastic Gradient Descent optimizer and early stopping. This model obtained better results compared to the baseline, however, overfitting was still a problem. We consider that these results might be improved by a combination of early stopping and dropout, in addition to optimizing an expanded parameter space.

4.3 XGboost

XGBoost [6] is an ensemble learning algorithm that leverages the predictions of several gradient boosting tree machines, and then combines its estimates to predict the outcome variable class. We optimized its parameters by using the Optuna package [7], which iterates through the parameter space and automatically prunes unpromising model configurations. Thus, we achieved a slightly better result by carefully parameterizing its estimators, learning rate and regularizers (lambda and alpha). However, we still faced overfitting as the number of estimators was too high (690) compared to the default number (100), thus leading to a complex model.

4.4 LightGBM

LightGBM [8] is also a gradient boosting algorithm and can be considered an upgrade of XGBoost. While XGBoost performs level-wise splits when creating the trees, LightGBM does a greedy leaf-wise search when attempting to reduce its loss function. This tends to yield better accuracy by introducing more complex trees. The Optuna framework was used for parameter tuning, which resulted in low values for lambda parameters for regularization to prevent overfitting, and moderate number of leaves and max depth to yield a good AUC score.

Model	Parameters	AUC score
LogisticRegression	C: 100, penalty: l2, solver: newton-cg	0.78041
Artificial Neural Network	hidden_layers: 4, activation: relu, last_activation = sigmoid, neurons: 512, optimizer: SGD, epochs: 100, early_stopping = True, loss = binary_crossentropy	0.83981
XGBoost	n_estimators: 690, learning_rate: 0.391, booster: gbtree, reg_lambda: 1.90e-05, alpha: 2.59e-08, subsample: 0.997, max_depth: 9, min_child_weight: 2, eta: 0.037, gamma: 0.013	0.84351
LightGBM	boosting: gbdt, lambda_l1: 5.95e-07, lambda_l2: 0.063, num_leaves: 161, feature_fraction: 0.542, bagging_fraction: 0.896, bagging_freq: 4, subsample: 0.475, max_depth: 25, learning_rate: 0.406	0.85302

Table 2 - AUC scores obtained for each model in Kaggle competition

5 Model Evaluation

We considered a Logistic Regression model to establish a baseline for our comparative study. Since Logistic Regression is a linear model, we expected ANN, XGBoost and LightGBM to perform better as these models are capable of complex modelling of the data through their parameters and provide a better representation of the graph's sparsity. The driving factor for choosing the LightGBM model over the other models, besides the superior performance in the AUC score is that LightGBM is capable of producing a more complex tree structure by following a leaf wise split rather than a level-wise split as is the case in XGBoost. However, the increased accuracy of LightGBM can be offset by an increase in overfitting of the model. We tried to reduce this effect by experimenting with various max_depth parameters and also by carefully creating a subgraph from which we extracted node embeddings. This coupled with selection of a validation set as described previously led us to choose this model overall.

The overall performance of the models described above leads us to the conclusion that to get state-of-the-art results on the link prediction problem, it is crucial to extract richer features about the nodes and edges of the full graph. A Graph Convolutional Network, if implemented, would have been capable of representing the graph topology and local node information in a highly accurate manner.

6 Conclusion

In this project, we have implemented and shown the performance of various classifiers for the task of link prediction in graph networks. We implemented Logistic Regression as a baseline model that achieved an AUC of 78%. This was improved on using ANN, XGBoost and LGBM classifiers. The LightGBM model we implemented scored the highest AUC of 83% in the final Kaggle competition. This model utilized parameter optimization, and combined graph metrics and node embeddings as features. In order to prevent overfitting, the model also sampled a set of edges and removed those edges from the graph before training the classifier.

Future avenues for research into Graph Convolutional Networks and other Deep Learning models would certainly improve the performance of the link prediction task, by leveraging node and edge features as well as using the node embeddings available from such frameworks.

References

[1] Mutlu, E. C., Oghaz, T., Rajabi, A., & Garibay, I. (2020). Review on Learning and Extracting Graph Features for Link Prediction. *Machine Learning and Knowledge Extraction*, *2*(4), 672-704.

[2] Hagberg, A., Swart, P., & S Chult, D. (2008). *Exploring network structure, dynamics, and function using NetworkX* (No. LA-UR-08-05495; LA-UR-08-5495). Los Alamos National Lab. (LANL), Los Alamos, NM (United States).

[3] Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864).

[4] Wright, R. E. (1995). Logistic regression.

[5] Talos (2021). *Talos: Hyperparameter Optimization for Keras* [Online]. Retrieved from: https://autonomio.github.io/docs_talos.

[6] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., & Cho, H. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, *1*(4).

[7] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623-2631).

[8] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, *30*, 3146-3154.