CP 318: Data Science for Smart City Applications

Project 1 Report

Team Jupiter: Suraj Kumar, GVS Mothish, Naveen Reddi

Introduction: Pairwise relationships are prevalent in everyday life and are often analysed using networks consisting of nodes and edges. Real-world datasets frequently contain missing edges between nodes. In this project, we aim to accurately predict these missing edges between the links by training a machine learning model. The training network is a partial crawl of the Twitter social network.

Phases of Model Development Cycle:

Phase 1: Data Pre-processing and visualization:

The training network input is initially structured as an adjacency list in a CSV file. We converted this data into a pairwise edge format, where each row represents a directed edge from a source node (first column) to a destination node (second column). For instance, the input row containing **"58, 42, 3, 7"** was transformed into three rows: **(58-42)**, **(58-3)**, and **(58-7)**. Nodes without connections were disregarded as they had no edges to or from them.

The resultant data is loaded into the **'NetworkX'** python package. We studied the graph and **descriptive statistics** is presented below.

Number of nodes	4,867,136			
Number of edges	23,945,602			
Statistics	**Mean**	**Std**	**Min**	**Max**
In-Degree	4.91	21.50	1	4840
Out-degree	4.91	642.4	0	761793

Table 1 Descriptive Statistics

Data Summary:

- While the mean in-degree of 4.92 suggests a moderate level of followers per user, the vast majority have a low in-degree.
- Only few users have exceptionally high in-degree values, showcasing influential nodes in the network.
- The out-degree distribution is highly skewed, with a median of 0, indicating that most users do not follow others actively
- The presence of outliers is significant, as reflected in the maximum out-degree of 761793. These extreme values highlight a small group of users with an extensive reach, potentially acting as hubs in the network.

Phase 2: Sampling and Feature Generation:

Direct training on full graph is impractical considering its size. So, we employ sampling strategies to sample positive and negative edges from the network. We employed two strategies: **random sampling**- where pairs of nodes with and without edges were chosen randomly, and **strategic sampling**- where data was sorted by centrality degree of its nodes and then sampling from probability distribution which gives more probability to nodes with more degree of centrality. In this way, we can effectively ignore isolated nodes in the graph. Table 2 lists the feature sets considered for the training

Phase 3: Machine learning model:

We trained the network primarily with three machine learning models namely logistic regression, ensemble method Adaboost and biggraph. We used the python scikit-learn package to set up the training. We now describe the experiments carried out with each of the models and the results of training. We summarize the feature set in table 2 [2]. We arrived at these feature sets while experimenting with the models. The details of how the features are selected will be described later.

Feature Set 1	Adar, Jaccard, common neighbors and resource allocation
Feature Set 2	Adar, Jaccard, common neighbors and resource allocation, Preferential attachment.
Feature Set 3	Adar, Jaccard, common neighbors and resource allocation, Preferential attachment, No of followers and followees for source and destination, in-degree and out-degree.

Table 2 Feature Sets

1. Logistic Regression

Logistic regression is a basic binary classification model which requires minimal hyper-parameter tuning and works fairly well making it a reliable and well-understood choice for many applications. In what follows, we present the details of setting up the model using scikit-learn, hyper-parameter values and results.

- **Data Preparation**: Feature matrix and training labels were generated, and an **80-20** split created training and holdout sets. Features were standardized using **StandardScaler**. **5-fold cross-validation** on the training set assessed model performance, measured by AUC. Testing was done on the holdout set.
- **Model Evaluation:** Experiment 1 with commonly studied features (Feature set 1) resulted in promising AUC values as shown in Table 3 for various sample size.

	Random Sampling		Strategic Sampling	
Total Samples	**AUC (Train)**	**AUC(hold out)**	**AUC (Train)**	**AUC (hold out)**
20000	0.9007	0.8969	0.8795	0.8832
50000	0.8989	0.9014	0.8810	0.8777
100000	0.9001	0.8965	0.8799	0.8840
200000	0.9002	0.8977	0.8805	0.8722

Table 3 Logistic Regression: Sample size vs AUC

SUMMARY:

- Sample size of 100000 is found to be good for learning the model as further increase in sample size gave no appreciable improvement in model accuracy in the hold out set as evident from Table 3.
- Since we already had 200000 sampled data, we went ahead with further experiments with 200000 data (100000 positive and negative samples each)

- **Additional Features Exploration:** To enhance AUC, additional features were iteratively added (Feature sets 2 and 3) along with regularization (C represents inverse regularization parameter). Augmenting features improved AUC in both training set and hold out set from 0.88 to 0.93 and 0.97 as shown in Figure 1. Improvement in the hold out set is indicative of no appreciable overfitting in the data. Strategic sampling showed similar improvements but we observed an interesting result that AUC approached 0.99 both in train and hold out set. Does it mean the model has fully captured the underlying graph relation? Well AUC was **reduced** for this experiment with Kaggle data. We believe this can be explained as follows. Since both training and hold-out accuracy improved by augmenting features, overfitting is unlikely (We verified it by studying variations in AUC as a function of regularization). One possible reason for AUC reduction in Kaggle data can also be attributed to the Kaggle test data which could **potentially include corner cases** and handpicked in the graph not sampled by our sampling strategy and thus the model does not accurately capture the structure of those corner cases.
- **Nonlinear Boundary Exploration**: Considering the possibility of a nonlinear boundary, nonlinear features of degree 2 and 3 with feature set 1 were tested with regularization. Despite these efforts, the mean AUC from 5-fold cross-validation did not significantly improve compared to Experiment 1. AUC for both high regularization and low regularization values are shown in Figure 1 (**C = 1**, high regularization, **C = 1e4**, low regularization)

In Summary, we achieved ~0.88 AUC maximum with logistic regression. As stated earlier, the graph is sparse and has a number of outliers evident from descriptive statistics shown in Table 1. So, we believe accurately predicting such complex interactions may not be possible beyond ~0.88 AUC using logistic regression as it finally learns a single decision boundary. A global decision boundary to represent such complex interactions may not be feasible. Instead, we then thought of using multiple local decision boundaries and ensemble them together. So, we then implemented Adaboost with decision trees as base classifiers. Combining multiple decision trees can help in understanding intricate patterns in the data and we hope for better link prediction.

2. Adaboost

AdaBoost is an ensemble method that combines weak estimators to build a strong classifier. We used the Decision Tree classifier as the weak estimator.

Hyperparameter selection

We first employed RandomizedSearchCV, a hyperparameter optimization technique, to fine-tune the classifier. The search for optimal hyperparameters involved exploring a predefined search space: 'n_estimators' (# base estimators) ranging from 50 to 200, and 'learning rate' varying between 0.001 and 1 on a logarithmic scale. We also used a 5-fold cross validation setup to evaluate. After the search, the best hyperparameters were determined: 'n_estimators' = 100 and 'learning_rate' ≈ 0.0398. These params were chosen to train the model. The results of our experiments from Adaboost are

summarized in Table 4.

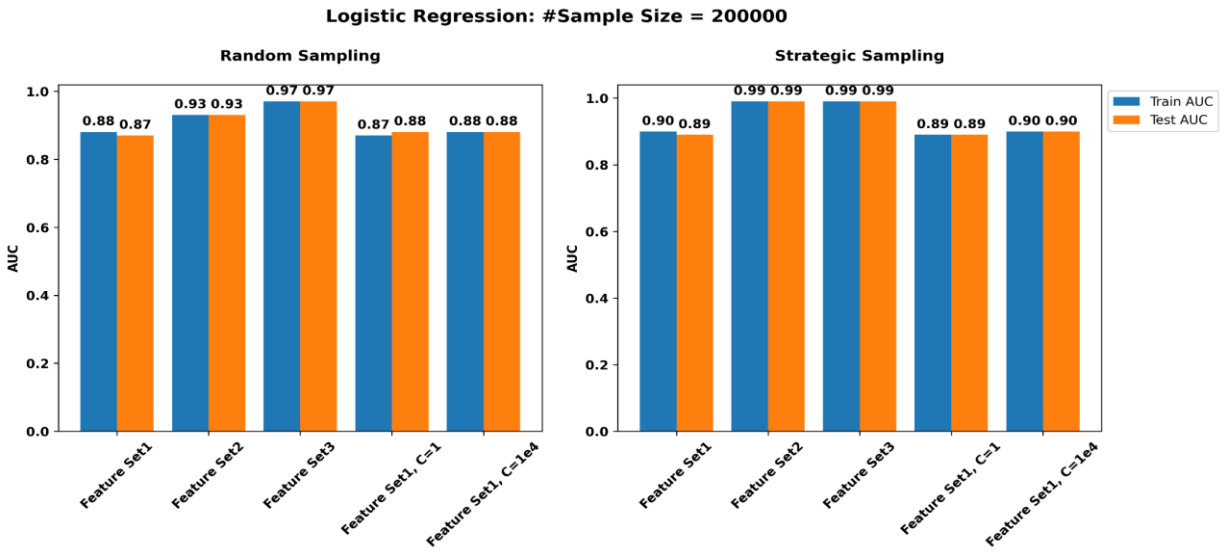

Figure 1 : AUC comparison for logistic regression

As we hypothesized, the optimized Adaboost classifier with Feature set 3 gives 0.99 AUC in training and hold out set shown in Table 4 and 0.92 AUC in Kaggle test data set.

Feature Set	Random Sampling		Strategic Sampling	
	AUC (Train)	AUC(hold out)	AUC (Train)	AUC(hold out)
1	0.8934	0.8927	0.8571	0.8573
2	0.9858	0.9828	0.8906	0.8909
3	0.9976	0.9977	0.9902	0.9910

Table 4 : AUC Adaboost

Parameters used in training:

- Logistic regression: C=1e4, penalty='l2', solver='liblinear'
- Adaboost: 5 folds, 'n_estimators': 100, 'learning_rate': 0.0398
- Big graph: dimension=1024, global_emb=False, lr=0.01, num_epochs=50, hogwild_delay=2 comparator='dot',loss_fn='logistic',

Graph Neural Network - Big graph (bonus marks):

In both the above methods, features were selected manually. We can alternatively first learn the features and then use those features for classification. Since features would be learnt from the data itself, non-trivial feature combinations which are important for data representation can be learnt in contrast to manual selection which requires domain knowledge and engineering insights and sometimes can be difficult.

TorchBigGraph is an open-source framework developed by Facebook Research for training large-scale graph embeddings using Neural Networks. It is built on top of **PyTorch** and is specifically designed for handling massive graphs. We have achieved **0.9086 AUC** on Kaggle by training this model. We hoped for higher AUC than 0.9086. We believed further hyper parameter tuning is necessary for the same.

Conclusion:

Initially, logistic regression yielded a maximum AUC of 0.89 using both linear and non-linear decision boundaries for 20,000 data samples. Efforts to enhance performance failed due to the complexity of edge relations not captured by the global boundary. Subsequently, Adaboost, combining multiple decision tree classifiers, was employed, resulting in an improved AUC of 0.92.

References:

1. Fire, Michael, Lena Tenenboim-Chekina, Rami Puzis, Ofrit Lesser, Lior Rokach, and Yuval Elovici. "Computationally efficient link prediction in a variety of social networks." ACM Transactions on Intelligent Systems and Technology (TIST) 5, no. 1 (2014): 1-25.
2. Cukierski, William, Benjamin Hamner, and Bo Yang. "Graph-based features for supervised link prediction." In The 2011 International joint conference on neural networks, pp. 1237-1244. IEEE, 2011.
3. https://vgnshiyer.medium.com/link-prediction-in-a-social-network-df230c3d85e6.
4. https://github.com/Bachfischer/COMP90051-StatML-Assignment-1/tree/master.