DA 218 Project1 Report: TEAM **Infinite Loopers**

Members: Abhinav Bhargava, Pratik Prabhakar Singh, Pinnamaraju Raviraj Sitaram

1) Description of the Problem:

The problem given is link prediction in a social network; the data set is a partial crawl of the Twitter network where a node represents each user, and an edge between the two users defines two users following each other, as Twitter has a two-way following system thus we need to need to use a directed graph to represent both user A following user B and user B following user A.

The Task is to evaluate from the given 2000 edges which of the edges are real connections and which are fake. For this, we are supposed to develop an ML model to predict the reality of the connection. We are provided with a training dataset with data in adjacency edge list format to train the ML model. We are provided with the test dataset in edge list format to test the model.

2) Description of the approach

a. Data Preparation

The training data is parsed and converted from the adjacency list format into a list of tuples where each element is a real edge in the graph. Similarly, the test edge list is parsed as a list of tuples; we extract the list of all nodes in the graph from these lists.
We have extracted features related to nodes and edges in the graph.

b. Feature Selections

We have extracted a total of 13 node features and 8 edge features; as computations for the entire graph were taking extremely long, we employed multiprocessing to divide the workload, and for faster access, we used the pickle module to store the features for nodes and edges as binary files for later reference.

i. Node Features:

1. Neighborhood: This is defined as the list of all nodes u such that an edge exists between node v in the graph. Then node u is a neighbor of v.
$$\boldsymbol{\Gamma(v) := \{u | (u,v) \epsilon\, E\ or\ (v,u) \epsilon E\}}$$
2. In-Neighborhood: This is defined as the list of all nodes u such that an edge exists and comes to node v from node u in the graph. Then node u is an in-neighbor of v.
$$\boldsymbol{\Gamma_{in}(v) := \{u | (u,v) \epsilon\, E\}}$$
3. Out-Neighborhood: This is defined as the list of all nodes u such that an edge exists and comes to node u from node v in the graph. Then node u is an out-neighbor of v.
$$\boldsymbol{\Gamma_{out}(v) := \{u | (u,v) \epsilon\, E\ or\ (v,u) \epsilon E\}}$$
4. Inclusive-Neighborhood: This is the neighborhood of v along with node v included.
$$\boldsymbol{\Gamma^{+}(v) := \{u | (u,v) \epsilon\, E\ or\ (v,u) \epsilon E\}}$$
5. Degree: We have calculated 4 degree parameters based on the above neighborhood definitions. They are total degree ($d(v) = |\Gamma(v)|$), in-degree ($d_{in}(v) = |\Gamma_{in}(v)|$), our degree ($d_{out}(v) = |\Gamma_{out}(v)|$) and bi-degree ($d_b(v) = |\Gamma(v)| \cap |\Gamma(v)|$), each defined for each node.
6. Degree Densities: Based on the degree features we have computed the densities respectively for in-degree, out-degree, and bi-degree as corresponding degree value/total degree.
7. Subgraph Link Number: We have defined the subgraph for node v as the set of all edges for which both source and target node belong in the neighborhood of v. We calculate the number of such edges as subgraph link number.

8. Subgraph Link Number Inclusive: We have defined the subgraph for node v as the set of all edges for which both source and target node belong in the inclusive neighborhood of v. We calculate the number of such edges as subgraph link number.

ii. Link Features

1. Common Friends: We have defined 4 types of common friends; common friends, in common friends, out common friends, and bi common friends.

$$common - friends(u, v) = |\Gamma(u) \cap \Gamma(v)|$$
$$common - friends - in(u, v) = |\Gamma_{in}(u) \cap \Gamma_{in}(v)|$$
$$common - friends - out(u, v) = |\Gamma_{out}(u) \cap \Gamma_{out}(v)|$$
$$common - friends - bi(u, v) = |\Gamma_{bi}(u) \cap \Gamma_{bi}(v)|$$

2. Total Friends: We have defined this as the total number of distinct neighbors of u and v.

$$total - friends(u, v) = |\Gamma(u) \cup \Gamma(v)|$$

3. Jaccard's Coefficient: The Jaccard's coefficient, which measures the similarity between sample sets, is denoted as the size of the intersection divided by the size of the union of the sample sets.

$$jaccards - coefficient(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

4. Transitive Friends: We have defined this as the number of distinct interaction in out-neighborhood of u and in-neighborhood of v.

$$transitive - friends(u, v) = |\Gamma_{out}(u) \cap \Gamma_{in}(v)|$$

5. Preferential attachment score(pas): One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks.

$$pas(u, v) = |\Gamma(u)|.|\Gamma(v)|$$

c. **Model Selection:** We have tried multiple approaches, using Logistic Regression, DecisionTrees, Random Forests and Naïve Bayes Algorithm.

i. **Description**

1. **Logistic Regression:** Logistic Regression is commonly used to estimate the probability that an instance belongs to a particular class. If the estimated probability is greater than the threshold, then the model predicts that the instance belongs to that class else predicts it does not. For the given problem, this model is a perfect candidate since it solves a binary classification problem and the output before the final prediction is a float value between 0-1.
2. **Bagging and Random Forests:** Random Forest is an ensemble learning classifier which ensembles multiple decision trees. It introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node (of a Decision tree), it searches for the best feature among a random subset of features. Hence, to tackle unseen data in testing RF could have been one of the best options for this problem.
3. **Naïve Bayes:** It is a family of algorithms that all work on Bayes theorem and are based on a few naïve assumptions like the independence of features and equality among features. Here, the underlying principle in forming the edges is a dependency of features. A person follows another person based on certain qualities(features).
4. **Adaptive Boost:** Boosting combines several weak learners into a strong learner. Adaptive Boosting (Adaboost) trains predictors sequentially, trying to correct its

predecessor on the training instances that it underfit. Thus helping the model to be adept at tackling unseen instances. The base classifier used is Decision Trees.

5. **Graph Convolution Networks:** GCN (Graph Neural Networks): It extends to Convolutional Neural Networks applied to Graphs Structured Data. The model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes.

ii. **Model Variations Implement:** We worked on multiple classifiers parallel to be able to choose the best-performing classifier.

1. Bagging, Random Forests: We implemented a soft voting classifier based on two ensemble learning methods, both variations of decision trees. We abandoned this due to the severe overfitting we had observed. We decided to approach the problem with naïve Bayes instead.
2. Naïve Bayes: We saw good results with an accuracy of ~93% on the validation data set.
3. Logistic Regression: We saw good results with an accuracy of ~98% on the validation data set with logistic regression. We used grid search CV to search for the best values of parameters to use, with 10000 iterations for the logistic regression.
4. Naïve Bayes + Logistic Regression: As we were observing good accuracy on both naïve Bayes and logistic regression, we decided to move ahead with both methods parallelly, and we also implemented a voting classifier-based model.
5. Graph Convolution Networks: We tried a few runs with GCNs but due to the complicated setup and a lack of required compute resources we pursued GCNs with low interest. Our Model was built with 2 convolution layers using ReLu as the activation function with final layer using Sigmoid.
6. Adaptive Boost Classifier: We tried to use an adaptive boost classifier with a learning rate of 0.05 with 1000 estimators.

d. **Conclusions on Performance:** We primarily focused on three approaches one using the Voting Classifier with Naïve Bayes and Logistic Regression, Ada Boost, and GCNs(Low Priority)

i. We observed that Naïve Bayes and Logistic Regression based Voting Classifier was working well though it required further fine tuning.

ii. GCNs: Due a lack in compute resources we were not able to scale the model by a lot and we had to use very limited subset of the data. Due to the limited amount of data and the limit on compute resources we were not able to any substantial performance. Though even with limits it proved to be a good method achieving ~83% AUC on validation data set but only 43% accuracy on test data set.

We have evaluated multiple methods during the project and some of them have considerable promise in solving the link prediction problem, our final approach was to pursue AdaBoost, Logistic Regression and GCNs.