Machine Learning in Network Science

Assignment 2
–
Kaggle Challenge Report

Team: ZZCCMXTP

Priyam DASGUPTA | 2201624
Shamir MOHAMED | 2100730

Section 1: Feature engineering

We used both Pandas, NetworkX and Numpy libraries to work on this challenge.
NetworkX is a Python package for the creation, manipulation, and study of complex networks that we use a lot in Machine Learning in Network Science.

Features used:

-*iloc() function* : 'iloc()' is a method used in pandas to select data based on integer indices. In the code, the function is used to select data from the 'node_information.csv' file.

-*iterrows() method* : 'iterrows()' is a method used in pandas to iterate over the rows of a dataframe as (index, Series) pairs. In the code, this method is used to iterate over the rows of the dataframes and extract node features.

add_node() method : is used in the NetworkX library to add nodes to a graph object. In the code, the method is used to add nodes to the graph with their features.

add_edge() method : is used in the NetworkX library to add edges to a graph object. In the code, the method is used to add edges to the graph with label = 1.

compute_network_characteristics() function: This function takes a graph object as input and returns a dictionary containing various network properties such as number of nodes, number of edges, minimum degree, maximum degree, mean degree, median degree, and density.

get_gcc() function : takes a graph object as input and returns the greatest connected component (GCC) of the graph. If the graph is connected, it returns the original graph. If the graph is not connected, it returns the GCC and prints the number of connected components, fraction of nodes in the GCC, and fraction of edges in the GCC.

compute_degree_centrality() function: This function takes a graph object as input and returns a dictionary containing degree centrality values for all nodes in the graph.

compute_closeness_centrality() function: This function takes a graph object as input and returns a dictionary containing closeness centrality values for all nodes in the graph.

assert statement: this statement that checks if a condition is True. In the code, it is used to check if the implementation of compute_closeness_centrality() function returns the correct values.

DataFrame object: DataFrame is a 2-dimensional labeled data structure in pandas. In the code, DataFrame objects are used to store degree centrality and closeness centrality values for all nodes in the graph.

Features Performances:

In this project, we use different graph-based features to train our models and predict the existence of edges between nodes. These features are designed to capture different aspects of the graph structure that could be indicative of whether an edge exists between two nodes. Below we explain the motivation and intuition behind each feature:

1. Degree Centrality: Degree centrality measures the number of edges that are connected to a node. The intuition behind this feature is that nodes that are connected to many other nodes in a network are more likely to be important and influential, and thus more likely to be connected to other nodes.
2. Betweenness Centrality: Betweenness centrality measures the number of times a node lies on the shortest path between other nodes. The intuition behind this feature is that nodes that are on many shortest paths between other nodes may have more control over the flow of information in the network and are therefore more likely to be connected to other nodes.
3. PageRank Centrality: PageRank centrality is a measure of the importance of a node in a network. It is based on the idea that nodes that are linked to by many other important nodes are themselves likely to be important. This feature is designed to capture the importance of nodes in a network.
4. Preferential Attachment: Preferential attachment measures the likelihood that two nodes will be connected based on the number of edges they already have. The intuition behind this feature is that nodes that have many edges are more likely to attract new edges, which may lead to the formation of new connections.
5. Jaccard Coefficient: The Jaccard coefficient measures the similarity between the neighbors of two nodes. The intuition behind this feature is that nodes that have similar neighbors are more likely to be connected to each other.

These features were chosen based on their potential relevance to the problem of edge prediction and prior research in the field. Each feature captures a different aspect of the graph structure that may be indicative of edge existence. By using a combination of these features, we trained our models to make accurate predictions on the “test” text file.

Section 2: Model tuning and comparison

We used a Logistic Regression Classifier with the clf function and trained it on the training features and labels.

In the prediction function, the logistic regression classifier is created with its default hyperparameters. The default hyperparameters are used to train the model and make predictions.

In this project, we experimented with different network metrics to predict the likelihood of a link between pairs of nodes in a social network. We used the Degree centrality, Betweeness Centrality, Preferential attachment, PageRank centrality and Jaccard coefficient as additional features to improve the accuracy of our model.

PageRank is a measure of the importance of a node in a network, based on the idea that a node is important if it is linked to by other important nodes. We calculated the PageRank score for each node in the network and used it as a feature in our model.

We trained our models on a set of labeled edges in the network, using the extracted features as input. We then evaluated the performance of the models using a separate set of test edges, calculating the area under the receiver operating characteristic curve (ROC AUC) as a metric of performance.

We wanted to see if we could improve the performance of the model by using a more complex machine learning model.

To this end, we attempted to use a random forest classifier to predict the likelihood of a link between two nodes. The random forest model is an ensemble of decision trees that combines the predictions of multiple decision trees to make a final prediction. The idea is that by using multiple trees, we can reduce overfitting and increase the accuracy of the predictions.

We used the same set of features that we had used for the logistic regression model, including the degree centrality, betweenness centrality, PageRank centrality, preferential attachment and Jaccard similarity coefficient. We used the scikit-learn library to build and train the random forest classifier.

Unfortunately, the results of using the random forest classifier were not significantly different from the results we obtained with the logistic regression model. The difference in the submission score was negligible. While the random forest model did have slightly better performance on the training set, it did not generalize well to the test set. This suggests that the model may be overfitting the data and not capturing the underlying patterns in the graph or the right combination of hyperparameters were not tested.

Overall, we found that the logistic regression model performed almost at par with the random forest model for this task. It is possible that with more tuning and optimization, the random forest model could perform better, but we did not see significant improvement in our experimentation.