EECS 298 Network Science Final Report GitHub Social Networks

Eesha tur razia babar
Yuhang Yao
Shengqi Li
Yinong Tian

I. Introduction

Network structures, or graphs, are ubiquitous in both nature and society, from the intricate connections between neurons in our brains to the complex web of relationships between individuals on social media platforms. The structure of a graph can have a profound impact on the behavior of its nodes, and different types of networks can exhibit vastly different properties and dynamics. Moreover, the features of individual nodes, such as their age, gender, location, or interests, can play a crucial role in shaping the network's overall behavior. For this reason, many tasks in network analysis, such as classification, clustering, and link prediction, need to take into consideration both the graph structure and the node features in order to achieve accurate and meaningful results. By leveraging the rich information contained within a network's topology and node attributes, researchers and practitioners can gain a deeper understanding of complex systems and develop effective strategies for a wide range of applications.

A. Motivation

To gain a deep understanding of the fundamental notions and tools used in network science, it is often helpful to focus on a specific data set that can serve as a concrete example. In this regard, the data set we used in this project is the GitHub Social Network data set provided in [1]. This data set provides a wealth of information about the connections between a large number of nodes, allowing researchers to explore a wide range of network properties and phenomena. Specifically, the nodes are sampled from GitHub developers and edges are mutual follower relationships between them. The data set also provides features for each node. By studying the structure and behavior of the GitHub Social Network data set, we hope to gain practical experience with important tools and techniques in network analysis, such as node classification methods, community detection algorithms, and link prediction models.

B. Data set Description

Github social network data set contains an undirected graph. Nodes are basically the developers who have starred at least 10 repositories and edges are mutual follower relationships between nodes. Each node of the graph has multiple features which are extracted based on the location of the developer, repositories starred, employer, and e-mail address of the node. Each feature is coded into a 128-dim vector.

An illustration of the structure of the dataset is shown below. Each node is given a binary label indicating the node

Node	Feature (128-dim)	Connection	Label
Node 1	[0.22,0.43,$\cdots$]	[Node 2, Node 8, $\cdots$]	1
Node 2	[0.87,1.23,$\cdots$]	[Node 1, Node 5, $\cdots$]	0
Node 3	[0.11,0.55,$\cdots$]	[Node 9, Node 15, $\cdots$]	1
$\cdots$	$\cdots$	$\cdots$	

TABLE I
An illustration of the dataset structure

(a GitHub user) is a machine learning programmer or a web programmer. The data set is biased such that approximately 74.2% nodes have label 0 and the other 25.8% nodes have label 1. The data set contains $N_v = 37700$ nodes and $N_e = 289003$ undirected edges. Throughout this paper, we will use N_v and N_e to denote the number of nodes and edges of this data set, respectively. In addition, we will use $\mathbf{A}$ to denote the adjacency matrix of the graph and A_{ij} to denote its entries. One possible task that can be done on this data set is the binary node classification. Other possible tests include (unsupervised) community detection, edge prediction, etc.

C. Related Works

The problems of node classification and the community detection are closely related, which are among the fundamental problems in network analysis. In this work, we consider binary node classification and binary community detection. Since our data set contains node features, we will focus on the node-attributed node classification methods. There are several ways to deal with the node-attributed node classification problem, most of which pertains to embedding-based approaches. Node embedding is the process of mapping each node in the graph to a continuous feature representation, such as an element in an n-dimensional vector space while keeping the graph structure as much as possible. The classification process can be done after one has the embeddings of the nodes. Broadly speaking, there are three types of approaches to generating node embeddings: factorization-based, random walk-based, and deep learning based. Factorization-based algorithms represent the connections between nodes in the form of a matrix and factorize this matrix to obtain the embedding [2], [3]. One commonly used approach is local linear embedding, where the embedding of a node is a linear combination of the embeddings of its neighbors. Random walk-based methods use a walking approach to generate sample network neighborhoods for nodes, and the embedding of a node is generated based on the sample network generated from it [4], [5] . Two nodes have similar embeddings if the sample network generated from them is similar. Deep learning-based methods leverage the capacity of deep neural networks to build autoencoders to learn the representation of a graph [6], [7]. In fact, these methods are not exclusive and a complete solution usually contains all of the ideas. For example, Deepwalk implicitly factorizes a PMI matrix that has been shown to correspond to the mean of a set of normalized adjacency matrix powers (up to a given order) reflecting different path lengths of a first-order Markov process [8].

In contrast to the aforementioned node classification problem, which is mostly a supervised task, community detection on the other hand is a semi-supervised or unsupervised task. Several approaches on this task include the Girvan-Newmann's method [9], which divides the graph into two parts by repeatedly removing the edge that has the highest betweenness centrality. This process is also known as the graph bisection. Most approaches to solving graph bisection are spectral methods which solve the problem by finding eigenvectors of some graph-related matrices (e.g., modularity matrix or Laplacian matrix), for example, spectral modularity maximization [10], minimum-cut [11], Radio cut [12], Normalized cut [13], to name a few.

Unlike the previous two tasks where nodes are mostly concerned, link prediction is a problem in network science that focuses on the existence of edges. The task of link prediction involves predicting the likelihood of a connection between two nodes in a network. This problem is also relevant in social networks, where predicting future connections between individuals can provide valuable insights into the dynamics of the network and help in tasks such as recommending new friends or identifying potential collaborations. Useful algorithms on the link prediction can be found in, e.g. [14], [15].

In the following subsections, we will give a more detailed formulation of the three main tasks that we will solve in this work.

D. Binary Node Classification

Node classification is the process of predicting the label of the node. Speaking of this data set, we are interested in predicting the binary label provided by the data set. Suppose we are predicting the labels of N nodes in a test, then the accuracy of the prediction is defined as

$$\text{Accuracy} \triangleq \frac{N'}{N} \in [0, 1], \tag{1}$$

where N' denotes the number of nodes of which the label is correctly predicted.

E. Community Detection

In constrast to the aforementioned task of binary node classification, the task of community detection considered in this paper is an unsupervised task, which aims to predict the label of each node without knowing any true label from the data set. This is a more challenging task because the algorithm cannot learn from the true labels, but it needs to figure out the rules and patterns behind the graph structure and the features. The accuracy is defined similarly as in Eq. (1).

F. Link Prediction

The task of edge prediction is to predict the existence of an edge given the nodes of its two ends. Suppose we are predicting the existence of N possible edges, and the number of correct prediction is N', then the accuracy is defined similarly as in Eq. (1).

II. Binary Node Classification

The first problem we explored on the data set is the problem of binary node classification. Since we are given the ground truth labels of each node, it is feasible to apply supervised approaches. Specifically, in our work, we implemented, tested and compared the following four methods, Node2Vec, GNN (Graph Nerual Network) and SVM (Support Vector Machine). The details of each of the three methods we used will be discussed in the following subsections.

Given the fact that our data is based on graph structure and most of the classical machine learning algorithms work with numerical data only, it is essential to generate the numerical representation of our graph. The way of representing the nodes as numerical vector such that the graph structure is preserved is called node embeddings.

A. Node Embeddings and Shallow enconding

Over time researchers have introduced multiple algorithms to extract node embeddings from graphs. The basic intuition behind these algorithms is that to encode nodes in a way that similarity in the embedding space (e.g., dot product) is almost equal to the similarity in the original network [16]. The simplest encoding approach is shallow encoding where the encoder is just an embedding lockup. In shallow encoding, each node is assigned a unique embedding vector. There are different kinds of shallow encoding-based algorithms based on how these algorithms define similarity. These similarity types are adjacency-based similarity [2], multiple hop-based similarity [17], and random walk-based approaches [4] We chose to work with random walk-based approaches as these are generally considered more efficient. Out of random-walk-based approaches we selected node2vec [18] as it is generally considered to work better for binary classification [19].

We used built-in module of node2vec in python3 for embedding generation. The choice of embedding dimension is a compromise between time/space complexity and large enough embedding size to achieve best classification accuracy. After performing multiple experiments we chose 20 as the dimension of the embedding which is a good compromise between time/space complexity and accuracy.

One thing to note here is that, node2vec does not take into account node features but only graph structure. Hence, our embeddings matrix only has information about graph structure but not any information about node features.

After getting embeddings for our graph we applied multiple classification algorithm on it. We kept the size of train/test split as 0.3. Which means that 70 percent of data used for training and rest 30 percent used for testing.

We tested algorithms like gradient boosting, KNN, random forest, and multi-layer perceptron for classification purposes.

1) Classification Algorithms: The first algorithm which we applied for classification purposes is gradient boosting. After tuning multiple hyperparameters the best accuracy we got through this method as 0.8559. The precision of the model is 0.5766 and recall is 0.8025. The second model we used was a multilayer perceptron. The best results we obtained using hidden layer sizes as 200, alpha as 0.05, activation function as tanh. The best accuracy we got using MLP is 0.8599. The precision of the model is 0.6280 and recall is 0.7795. Using the random forest the best results we obtained using n_estimators as 100, max depth as 180 and max features as 11. The best testing accuracy we got as 0.8555, precision as 0.5693 and recall as 0.8067. Using KNN we obtained best accuracy using k as 26. And the best accuracy value is 0.8535. The precision as 0.5593 and the recall as 0.8199.

Model	Accuracy	Precision
Gradient Boasting	0.8559	0.5766
MLP	0.8599	0.6280
Random Forest	0.8555	0.5693
KNN	0.8535	0.5593

TABLE II
TABLE SHOWING ACCURACY AND PRECISION OF DIFFERENT MODELS

Looking only at the accuracy column of TABLE II it looks like that we are getting good accuracy and our algorithms are working well. However, we also have to consider the column containing the precision of each model especially when our data set contains unbalanced labels.

Fig 1 is the graph showing class distribution in our data set which shows that it is a fairly unbalanced distribution.

Hence, in this situation accuracy itself does not provide any useful information. We need to consider precision and have to look at the confusion matrix too. The precision with the best possible model is only 0.63. Looking at the confusion matrix, we realized that most of the data points in the minority classes are being predicted wrong.

We looked into many techniques to resolve the issue of unbalanced data and decided to use the synthetic minority over-sampling technique (SMOTE) for this purpose. SMOTE up-samples the number of minority class in data set. SMOTE chooses an example from minority class for example a and then searches for k nearest minority class neighbors of that example. The synthetic example is formed by selecting one of the k nearest neighbour b at random. Then it connects a line between a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b [20].

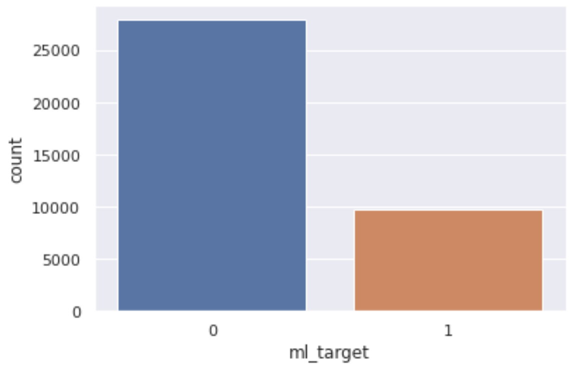

Fig. 1. Class distribution

We applied SMOTE on training data and then reapplied the classification algorithms on the test data.

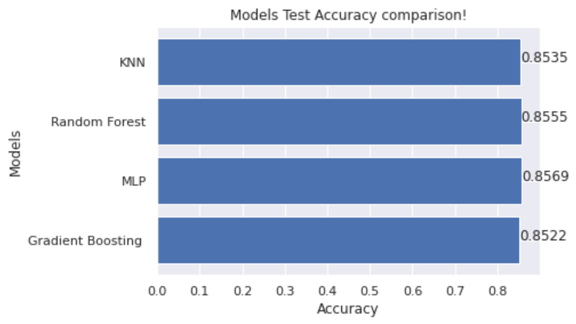

Fig. 2. Test accuracy comparison of different models

2) SMOTE and Classification Algorithms: As discussed above, after applying SMOTE the minority class has been up-sampled in the training data. The model is seeing equal number of 0s and 1s in the the training data. And then we trained multiple models again and tested these on same test set. Using the gradient boosting algorithm, we got best accuracy as 0.820. The precision as 0.7630 and recall as 0.6196. After using MLP with same parameters as used before SMOTE, the best accuracy we got is 0.8217. The precision as 0.7369 and recall as 0.6280. Using random forest the best accuracy we got as 0.8419. Precision as 0.6828 and recall as 0.6924. Using the KNN algorithm the best accuracy we obtained as 0.7809, precision as 0.8132 and recall as 0.5534.

Although looking at accuracy only, it looks like that it decreased after applying SMOTE as best accuracy value decreased from 0.85 to 0.84 but the precision associated with model providing the highest accuracy increased from 0.63 to

Model	Accuracy	Precision
Gradient Boasting	0.820	0.7630
MLP	0.8217	0.7369
Random Forest	0.8419	0.6828
KNN	0.7809	0.8132

TABLE III
TABLE SHOWING ACCURACY AND PRECISION OF DIFFERENT MODELS AFTER APPLYING SMOTE

0.68. Looking at the precision column, it can be concluded that the value of precision increased for almost all algorithms. The higher value of precision shows that now algorithms are more general instead of just being specific to the majority class. Finally we also applied cross-validation to confirm our results and we obtained similar results using cross-validation too.

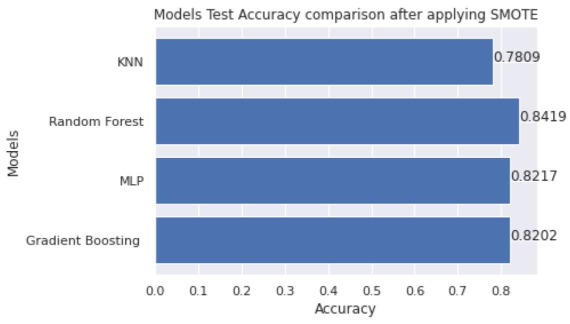

Fig. 3. Test accuracy comparison of different models

Even though the combination of embeddings generated through node2vec and the classification algorithms gave us reasonable test accuracy and precision, however shallow algorithms like node2vec has many limitations.

For shallow embeddings method we need $O|V|$ number of parameters as there are no parameter sharing and every node has its own unique vector. The shallow embeddings are transductive inherently it means its not possible to generate embeddings for the nodes which have not been seen during training phase. Also as mentioned earlier node2vec does not use features of the nodes. In order to avoid these limitations of shallow encoding method, and to incorporate the feature matrix of nodes we decided to explore other algorithms that can use both features and graph structure for learning and prediction. That is why we shifted towards deeper encoding methods.

B. Deep Encoding

In graphs, nodes have different neighborhood sizes. Deep encoder works by producing embeddings based on the local neighbourhood. The nodes aggregate information from their neighbours using the neural network. Key distinction between different deep algorithms are that how different approaches aggregate information across layers. The basic approach is to average the information from neighbours and apply a neural network. Graph convolution is a little variation of neighbourhood aggregation idea. The difference of the approach is shown in Fig. 4. The GCN uses same matrix for self and neighbours embeddings while basic aggregation does not use same matrix for both. Additionally in GCN the function is normalized per neighbour while the same is not true for basic neighbourhood aggregation [16].

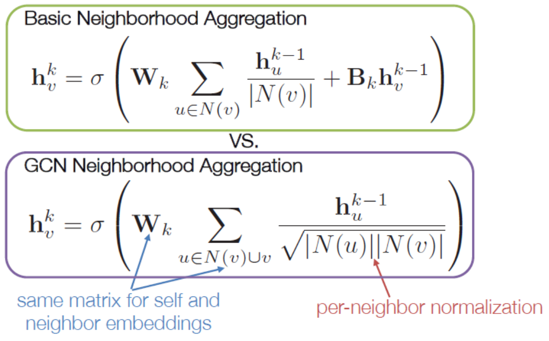
Basic Neighborhood Aggregation
$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u\in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k\mathbf{h}_v^{k-1}\right)$
VS.
GCN Neighborhood Aggregation
$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u\in N(v)\cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$
same matrix for self and neighbor embeddings
per-neighbor normalization

Fig. 4. GCN Approach

In our project, we used the graph convolutional network similar to one introduced by Kipf et al. [21]

In our experiment, we used pytorch geometric library for implementing a graph convolutional neural network. The feature dimension is 128. Other parameters including number of classes, number of nodes and number of edges are the same as we introduced previously. We used the random split function to split data into train set size of 26390 (70%) , validation set size of 3770 (10%) and test set size of 7540 (20%) data points. To get a baseline model for this implementation we used multilayer perception. The output embedding of both neural networks is a two dimension vector, denoted as $v = [v(0), v(1)]$. In the training process, we used CrossEntropyLoss as the criterion for computing the loss, and the predicted class is given as

$$c_{pred} = \arg\max_{c\in\{0,1\}} v(c). \tag{2}$$

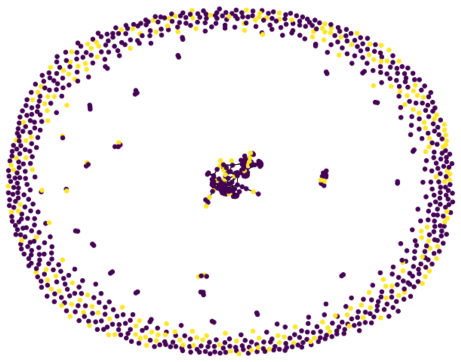

Fig. 5. Graph drawn using 1000 random points.

The multilayer perceptron only incorporates nodes features. After several experiments we acheived best testing accuracy values as 0.855. For this we used a four layer MLP with variable number of neurons in each layers (64, 32, 16), learning rate of 0.1, adam optimizers and cross entropy loss. After the multilayer perceptron (Feed-forward neural network) we applied graph convolutional network.

The graph convolutional network incorporates both graph structure and node features. We performed multiple experiments using graph convolution network. By balancing the time/space complexity and accuracy we got the optimum model using two convolutional layers, 16 neurons per layer, adam optimizer, learning rate of 0.01. The best test accuracy we obtained as 0.872.

The accuracy obtained using GCN is better than the accuracy we got using node embeddings from node2vec and classification algorithms and also using multilayer perceptron. Additionally, for GCN, we did not use the upsampling technique in order to avoid increasing time/space complexity. Instead we are using precision to find out how general our algorithm is and we got best precision of 0.811 which is much better than the precision we got after SMOTE, hence we can conclude that our GCN is working better than all the previously applied classification algorithms.

The yellow points in the figure shows that how many points are predicted right after GNN. As shown from figure the reasonable number of points predicted right after applying GCN.

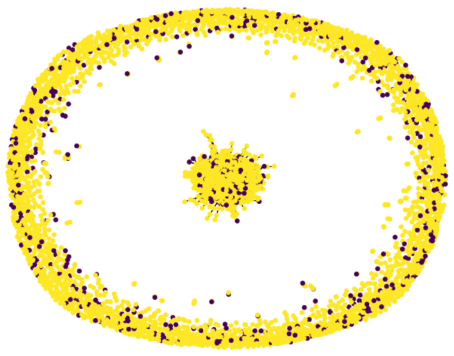
Fig. 6. Graph showing the predicted right points.

The accuracy and loss curves in training the GCN and MLP are shown in Fig. 7. Note that the accuracy we got from GCN is better than that we got from the baseline MLP. Since the MLP does not take information of the graph structure, while the GCN contains the graph structure information, it shows that the graph structure information helps the network to perform better in the binary classification task.

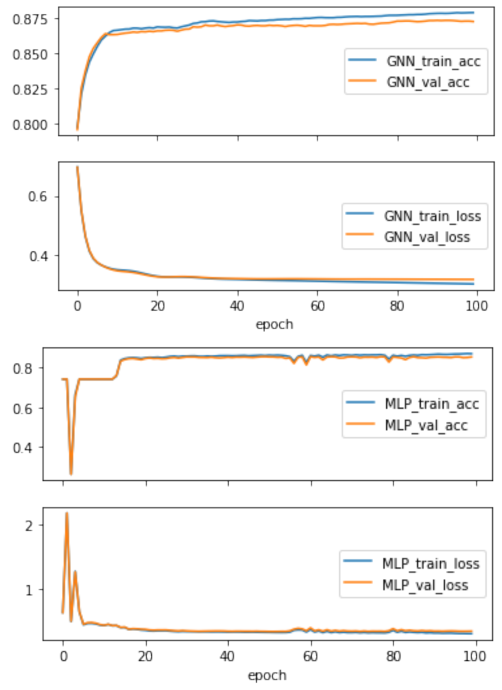

Fig. 7. Training curves (accuracy and loss) of GCN and MLP.

C. SVM

Support Vector Machine (SVM) is a kind of generalized linear classifier for binary classification of data according to supervised learning, and its decision boundary is the maximum margin hyperplane for solving the learning samples SVM uses the hinge loss function to calculate the empirical risk and adds a regularization term to the solution system to optimize the structural risk. It is a classifier with sparsity and robustness. SVM can perform nonlinear classification through the kernel method, which is one of the common kernels learning methods. In our project, the kernel function we use is linear, so we won't get in too far for other kernel function.

linear classification linear separability Given input data and a learning objective in a classification problem, each sample of the input data contains multiple features and thus constitutes the feature space. The learning objectives are binary variables representing negative class and positive class. If there is a hyperplane as a decision boundary in the feature space where the input data is located, the learning target is separated into positive and negative classes, and the point-to-plane distance of any sample is greater than or equal to 1 [22].

Decision boundary :

$$w^T X + b = 0$$

Point to plane distance

$$y_i(w^T X + b) \geq 0$$

Then the classification problem is said to be linearly separable, and the parameters w and b are the normal vector and intercept of the hyperplane respectively.

The decision boundary satisfying this condition actually constructs two parallel hyperplanes as interval boundaries to distinguish the classification of samples:

$$w^T X_i + b \geq +1 \Rightarrow y_i = +1$$

$$w^T X_i + b \leq -1 \Rightarrow y_i = -1$$

All samples above the upper interval boundary belong to the positive class, and those below the lower interval boundary belong to the negative class. distance between two interval boundaries $d = \frac{2}{||w||}$ is defined as the margin, and the positive and negative samples located on the boundary of the interval are the support vectors [23].

So, base on these we implement the SVM tool in MATLAB, using the existing libsvm tool to generate our result, and we first use 40% of the feature matrix as traning data set and 60% as testing, and we got the result as below.

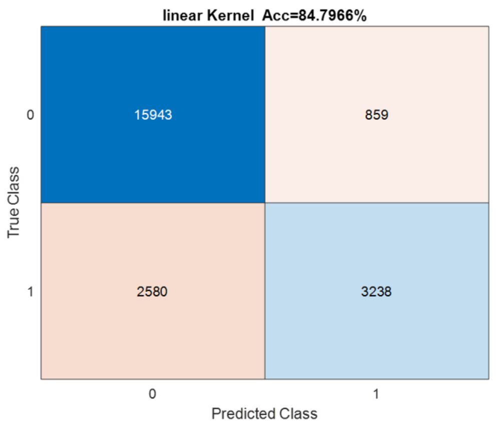

Fig. 8. Confusion matrix for SVM

At the same time, we also use the feature after applying SMOTE for our SVM model, and the final accuracy has slightly increased to 86%.

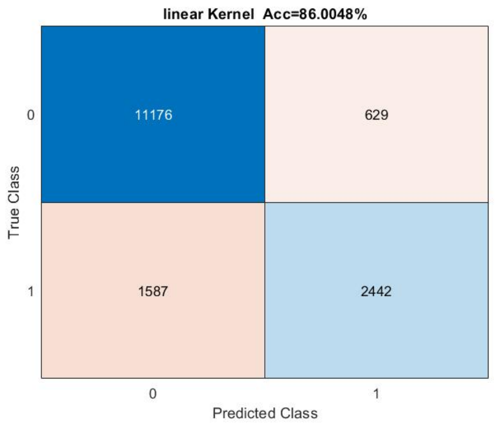

Fig. 9. Confusion matrix for SVM (applying SMOTE)

For binary node classification, node2vec algorithm had highest time complexity while all the other classification algorithm which uses node2vec based embeddings (Gradient boosting, KNN, RF, MLP) were very fast. The GCN and baseline MLP were less expensive as compared to node2vec algorithm. Hence node2vec was the slowest algorithms in binary node classification.

III. Community Detection

A. Girvan-Newman Method

A community is a subgraph of a graph that contains denser nodes than the rest of the graph, or equivalently, if the number of links within a subgraph of a graph is higher than the number of links between these subgraphs, we It can be said that this graph contains communities. As shown in the figure below, the network in the figure contains three community structures, corresponding to the nodes in the three dotted circles in the figure, and the nodes in the same community are closely connected to each other, while the connections between communities are relatively sparse. In other words, the community structure of a complex network is a characteristic that the connections between communities in a complex network are relatively sparse, while the connections within the communities are relatively dense.

Talking about an edge between communities, it can be quantified by many methods, and a community discovery algorithm can be implemented based on it. Newman and Girvan's paper proposed a method using betweenness centrality, which is called the Girvan-Newman algorithm [24]. In graph theory, betweenness centrality is a measure of the center of a graph based on the shortest path. For each pair of vertices in a connected graph, there is at least one shortest path between the vertices, and the betweenness of a node generally refers to the number of shortest paths passing through the node.

Betweenness centrality is defined as:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma st(v)}{\sigma st}$$

Newman and Girvan proposed a modularity function Q to quantitatively measure the results of community division while aiming at the problem of community discovery. Since the metric was proposed, it has gradually been accepted by most researchers, and many new algorithms with modularity as the objective function have been developed. Modularity refers to the difference between the proportion of edges connecting the internal nodes of the community in a complex network and the expected value of the proportion of edges connecting the internal nodes of the community in another random network. Among them, the construction method of this random network is: keep the community attribute of each node unchanged, and randomly connect the edges between nodes according to the node degree. If the community structure is obvious, the proportion of edges connected within the community should be higher than the expected value of the random network. Therefore, the modularity function Q can be used to quantitatively describe the modularity level of community division. Assuming that the community structure of the complex network has been discovered, M is the number of discovered communities, L is the number of edges in the network, and ls is the mutual relationship between nodes in the community s. The number of connections, ds is the sum of the number of interconnections of all nodes in the community s, then the function expression of Q is as follows:

$$Q = \sum_{s}^{M} [\frac{l_s}{L} - (\frac{d_s}{2L})^2]$$

The main idea of such a community discovery algorithm is: calculate the edge betweenness of all edges in the network, and then find the edge with the largest edge betweenness and remove it. When removed, the betweenness of some edges is changed. The shortest path that used to go through the removed edge now goes through other edges, so the edge betweenness must be recomputed, then the edge with the highest value is searched again and removed, and so on. As one edge after another is removed, the initially connected network is eventually split into two parts, three parts, etc., until each of the network becomes a community, a splitting algorithm.

The execution process of the algorithm can be represented by a tree diagram, as shown in the figure below. The algorithm generates a dendrogram from the top, starting with a single connected network, and then divides it until each branch has only a single node. During the running of the algorithm, each independent network structure state corresponds to a horizontal slice of the dendrogram, denoted by the red line in Fig. 2. Each branch that intersects the red line represents a set of nodes, a community, of which all nodes down the branch to the bottom are members. Therefore, for a connected network, the dendrogram can obtain the community structure of the network division in each stage of the algorithm execution process.

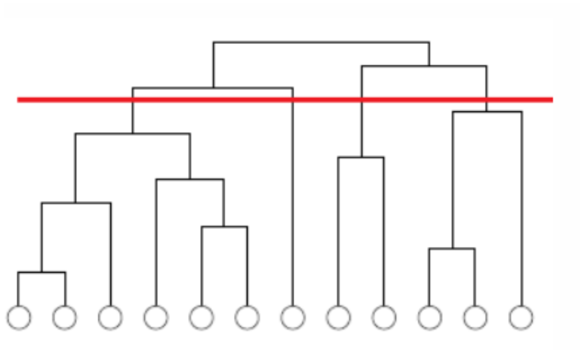

Fig. 10. Tree

In this way we can see that this algorithm does not give a unique division of the network into communities, but gives a variety of different degrees of division. The coarser division divides the network into several large communities, and the finer division divides the network into many smaller communities. Such different degrees of division can reveal information about the hierarchical network structure. At the same time, users can decide which divisions to make according to the number of target communities, or use the modularity function Q to give the optimal division of the algorithm.

As mentioned earlier, intuitively, the edge density between the nodes in the community is relatively large. Therefore, it is a more intuitive community to identify the community through the edge betweenness (the edge betweenness of the edges connecting the community is large). Discovery algorithms. The Girvan-Newman algorithm is a community discovery algorithm based on betweenness. Its basic idea is to continuously remove edges from the network according to the order of edge betweenness centrality from large to small until the entire network is decomposed into various communities. Therefore, the Girvan-Newman algorithm is actually a split method.

The basic flow of the Girvan-Newman algorithm is as follows: (1) Calculate the edge betweenness of all edges in the network; (2) Find the edge with the highest betweenness and remove it from the network; (3) Repeat step 2 until each node becomes an independent community, that is, no edge exists in the network.

The Girvan-Newman algorithm gives how to remove the edges to get the community structure. But before getting the final number of communities, there is still a problem that has not been solved, that is, how to determine the appropriate number of communities to make the results of community division optimal. At this time, we need to use the previously mentioned measure - the modularity function Q. In this way, the Q value can be used to find the appropriate number of communities. Every time an edge is removed in the Girvan-Newman algorithm, the Q value of the resulting community structure is calculated, and the number of communities with the maximum Q value is found.

In our project, we used a classic dataset called Zachary's karate club as example. The Zachary network is a classic real unweighted network to test different community discovery algorithms, with a total of 34 nodes and 78 edges. The network was obtained by Zachary in the 1970s by observing and studying the relationship between 34 members of the karate club for two years, as shown in the figure below. During the investigation, the club director and principal had a dispute and split into two groups. The real community is as follows: [[1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 17, 18, 20, 22], [32, 33, 34, 9, 10, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31]]

Here we use Python's NetworkX package and Matplotlib package to complete the demonstration of the Girvan-Newman algorithm, and Gephi, a JVM-based visualization platform. The result is shown below.

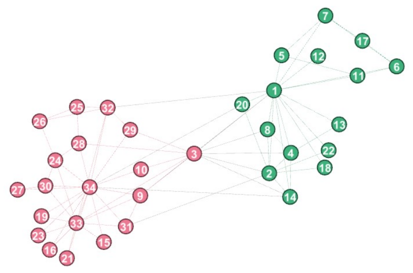

Fig. 11. Community detection on Zachary's karate club

The number of Communites 2 Communites: [[1, 2, 4, 5, 6, 7, 8, 11, 12, 13, 14, 17, 18, 20, 22], [3, 9, 10, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34]]

From here we can see the model did well when generating the community detection, however, when we use the same model for our dataset, the file could not excute, which lead us to think about the reason. We think it is because when calculating the edge betweenness, there may be a lot of repeated calculations of the shortest path, and the time complexity is too high. So we will optimize this in the future.

B. Spectral Modularity Maximization

Since the Girvan-Newman method does not work for our data set, we have tried some other methods, e.g. spectral modularity maximization proposed by Newman in 2006. [10] Spectral modularity maximization has been widely applied to community detection tasks as a classic unsupervised method, especially for bisection tasks due to simplicity.

Suppose we have a graph G and a partion into groups $s \in S$, and modulariy can be defined as follows:

$$Q(G,S) = \frac{1}{2Ne}\sum_{s\in S}\sum_{i,j\in s}(A_{ij} - \frac{d_i d_j}{2Ne})$$

Let g_i be the group membership of vertex i,and $Q(G,S)$ can be rewritten as:

$$Q(G,S) = \frac{1}{2Ne}\sum_{i,j\in V}(A_{ij} - \frac{d_i d_j}{2Ne})\Pi\{g_i = g_j\}$$

For bisection tasks, we can set vector s with binary elements showing community membership for each vertex as follows:

$$s_i = \begin{cases} +1, & vertex\ i\ belongs\ to\ group\ 1 \\ -1, & vertex\ i\ belongs\ to\ group\ 2 \end{cases}$$

To simplify the expression, we can create a modularity matrix B, which satisfies $B_{ij} = A_{ij} - \frac{d_i d_j}{2Ne}$. With the definition of s_i, we can see that $\Pi\{g_i = g_j\} = \frac{1}{2}(s_i s_j + 1)$,. Since $\sum_{i,j\in V} B_{ij} = 0$, $Q(G,S)$ turns into:

$$Q(G,S) = \frac{1}{4Ne}\sum_{i,j\in V} B_{ij}s_i s_j = \frac{1}{4Ne}s^T B s$$

After relaxing $s \in \{\pm 1\}$ to $s \in R^{N_v}$, $||s||_2 = \sqrt{N_v}$, the modularity maximization can be changed into an easier optimization problem as follows:

$$\hat{s} = arg\ \underset{s}{max}\, s^T B s \quad s.\,to\ s^T s = N_v$$

By including a Langrange multiplier λ, we can get $Bs = \lambda s$ and $s^T Bs = \lambda$, meaning that s is the dominant eigenvector of B.

From the above analysis, it can be obtained that the flow of the modularity maximization algorithm is as follows: (1) Compute modularity matrix B which satisfies $B_{ij} = A_{ij} - \frac{d_i d_j}{2Ne}$; (2) Find the dominant eigenvector u_1 of B; (3) Classify vertex i according to $s_i = sign([u_1]_i)$.

In our project, we use Matlab to implement the spectral modularity maximization algorithm, and the accuracy turns out to be 0.5453. Since the size of the matrices is larger than 30000×30000 and the calculation requires huge memory, we use 'single' format to perform the matrix calculation.

C. Ratio-cut Minimization

For bisection tasks, another popular method is to minimize the number of edges between groups. Here a concept 'cut' can be defined to measure the number between groups V_1 and $V_2 = V - V_1$ as follows:

$$C = cut(V_1, V_2) = \sum_{i\in V_1, j\in V_2} A_{ij}$$

Let g_i be the group membership of vertex i,and C can be rewritten as:

$$C = \sum_{i,j\in V} A_{ij}\Pi\{g_i \neq g_j\}$$

We can set vector s similarly as follows:

$$s_i = \begin{cases} +1, & vertex\ i\ belongs\ to\ V_1 \\ -1, & vertex\ i\ belongs\ to\ V_2 \end{cases}$$

$$C(s) = \frac{1}{2}s^T L s, \quad s := [s_1, \ldots, s_{Nv}]^T$$

Minimizing $C(s)$ can then be changed into an easier optimizing problem as follows:

$$\hat{s} = arg \quad \underset{s\in\{\pm 1\}^{N_v}}{min}\ s^T L s \quad s.\,to\ 1^T s = |V_1| - |V_2|$$

However, simply minimizing the cut may lead to some exceptions like the figure shows below:

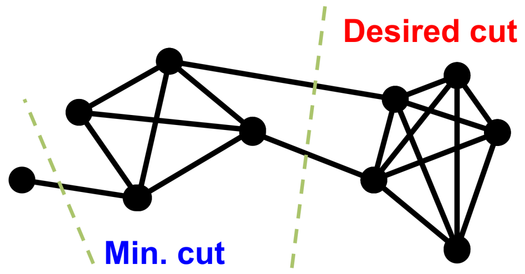

Fig. 12. An exception for minimizing the cut

In order to avoid such exceptions as much as possible, we need to maintain a relative balance of the partion. Therefore, our optimizing goal changes from C to $R = \frac{C}{|V_1|} + \frac{C}{|V_2|}$ to penalize small community, which is called 'ratio-cut minimization'. [12]

By setting a vector f as follows, the ratio-cut minimization can then be changed into an easier optimizing problem as follows:

$$f_i = \begin{cases} \sqrt{\frac{V_2}{V_1}}, & vertex\ i\ belongs\ to\ V_1 \\ -\sqrt{\frac{V_1}{V_2}}, & vertex\ i\ belongs\ to\ V_2 \end{cases}$$

$$\min_f f^T L f, \quad s.\,to\; 1^T f = 0 \; and \; f^T f = N_v$$

Similarly, from the above analysis, it can be obtained that the flow of the ratio-cut minimization algorithm is as follows: (1) Compute Laplacian matrix L which satisfies $L_{ij} = D_{ij} - A_{ij}$; (2) Find the eigenvector v_2 corresponding to the second smallest eigenvalue of L; (3) Classify vertex i according to $s_i = sign([v_2]_i)$.

In our porject, we use matlab to implement the ratio-cut minimization algorithm, and the accuracy turns out to be 0.7395. Similarly, we use 'single' format to perform the matrix calculation.

Both spectral maximization and ratio-cut minimization seem to get poor results. The possible reason is that these two methods are unsupervised, while the task is supervised, so they probably cannot learn some graph structure features of our dataset as supervised methods does.

IV. Link Prediction

Link prediction task is more difficult than node classification. For link prediction we need a encoder-decoder setting as introduced in [25] The researchers in this paper introduced graph auto encoders. The graph auto-encoders (GAEs) are neural network models that are used for unsupervised learning, clustering and link prediction on graphs. In the link prediction task an encoder creates node embeddings by using two convolutional layers. Then some random negative links are added to the original graph. Therefore the task is converted into binary classification where one class in positive links which are existing links in the original graph and the second class is negative links which are links added by us. Finally a decoder performs a binary classification task which is basically link prediction about both positive and negative edges. It calculates the dot product of the embeddings of nodes present on both corners of edge. Then it aggregates the values across the dimensions of encoding and then computes the probability of edge existence between each node pair. Train, validation and test set in this task are splitted using RandonLinkSplit. The train, validation and test set have two new features (edge label and edge label index). They are the labels of each edge and the indices of each edge corresponding to each split. The decoder uses the edge label index in order to make predictions and label of edge is used for evaluation of our model. Finally negative links are added to both validation data and test data with the same number as positive links. These negative links are added to edge label and edge label index features. However these are not added to edge index as it is not desired to add negative link on the encoder and also the negative links are not being added to training set too. These will be added during the training loop in train link predictor mentioned in link prediction section. The model is expected to become more robust through this randomization during training process [26].

The best accuracy we obtained using link prediction as 95.5 percent.

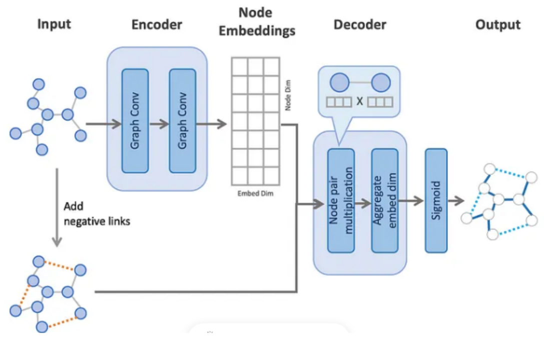

Fig. 13. Encoder-decoder architecture [25]

V. Conclusion and Future Direction

Through this project, we experimented with different classical and state of the art machine learning and graph based algorithms. We focused on three concrete tasks, binary node classification, binary community detection, and link prediction. For the binary classification problem we achieved maximum accuracy as 87.2 percent using GNN. Other approaches including classification using embeddings obtained from node2vec, and using SVM as classifier achieve accuracy ranging from 78 to 85 percent. For the community detection problem, we tested the spectral modularity maximization as well as the ratio-cut minimization. The accuracy of prediction is expected to be not as good as what we achieved in the supervised binary node classification, because the methods we used are unsupervised in the community detection problem. Lastly, for link prediction, we obtained based accuracy as 95.5 using auto-encoders. Possible future directions include generalization of the methods to n-ary ($n > 2$) node classification and community detection tasks also the network visualization.

Individual Contributions

Eesha tur razia babar studied different shallow encoding methods and worked to get embeddings from node2vec. She then studied and applied classification algorithms on the embeddings generated from node2vec for binary node classification. She worked on SMOTE, and applied classification algorithms after SMOTE, and worked on cross-validation. She implemented GCN and baseline MLP for comparison. She worked on link prediction and wrote all the corresponding sections of the report including conclusion.

Yuhang Yao worked in the literature review and the implementation of several modules in the node classification task. He collaborated with Eesha to test and tune the deep embedding methods in node classification. He helped in the preprocessing of the data and generated the features that is used in the SVM.

Shengqi Li worked on the SVM model using the feature matrix we got and generated the result for SVM. Also, he experimented with the Girvan-Newman Method on the Zachary's karate club data set. He will optimize the algorithm and apply it to the GitHub data set in the future.

Yinong Tian worked on the community detection task. He studied, implemented and tested the two modularity methods on the GitHub data set used throughout the project. He compared the results got from the spectral modularity maximization and the ratio-cut minimization.

References

[1] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-Scale Attributed Node Embedding," *Journal of Complex Networks*, vol. 9, no. 2, 2021.

[2] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 37–48.

[3] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 1476–1481.

[4] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[5] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[6] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[7] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 459–467.

[8] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, p. cnab014, 2021.

[9] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[10] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[11] M. Fiedler, "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory," *Czechoslovak mathematical journal*, vol. 25, no. 4, pp. 619–633, 1975.

[12] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 11, no. 9, pp. 1074–1085, 1992.

[13] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[14] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[15] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *SDM06: workshop on link analysis, counter-terrorism and security*, vol. 30, 2006, pp. 798–805.

[16] J. Leskovec, W. L. Hamilton, R. Ying, and R. Sosic, "Representation learning on networks."

[17] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[19] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705118301540

[20] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[22] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[23] T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization," *The Annals of Statistics*, vol. 32, no. 1, pp. 56–85, 2004.

[24] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[25] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[26] T. Masui, "Results from an exam in inferential statistics," 2022. [Online]. Available: https://towardsdatascience.com/graph-neural-networks-with-pyg-on-node-classification-link-prediction-and-anomaly-detection-14aa38fe1275%7D