Citation Network Classification with Graph Neural Network

Team Orange

Shangwen Yan

Jennie Sun

Michael Tang

Yi Feng

ABSTRACT

This project uses the Cora dataset to examine the applicability of Graph Convolutional Neural Network (GCN) which distinguishes itself from other machine learning models by taking into account the intrinsic interconnections between nodes. After comparing the model performance results of GCN and Logistic Regression Classifier, Naïve Bayes Multinomial Classifier, and K-Nearest Neighbors Classifier, we find that GCN gains enhanced predictive power from the interconnectivity within the network. This is corroborated by the significantly better test performance of GCN on the Cora dataset compared with the three other methods.

INTRODUCTION

In recent years, a variety of new methods have emerged in the machine learning field to improve prediction performance on data with different attributes. In particular, Graph Neural Networks (GNN) have attracted lots of attention from practitioners in this field. This project uses one variant of GNN--GCN to extract information from structured scientific publications in the Cora dataset. Two types of predictions are possible for doing machine learning on this dataset: label prediction and edge prediction. Whereas the latter predicts the papers that should be cited given node data, the former predicts the subject attributes, namely the label. Specifically, this project aims to train a graph neural networks model that does label prediction. These subjects are one of seven academic categories. To fully understand the advantages of GCN, this project also integrates other machine learning techniques, including Naïve Bayes Multinomial Classifier, Logistic Regression Classifier, K-Nearest Neighbors Classifier, and compares the differences in performance among these classification methods. Our analysis shows that on this citation network data, GCN has the highest mean accuracy of 0.80 on the testing set, creating a huge advantage gap compared to other methods, which have an accuracy of 0.59 (Logistic Regression), 0.58 (Naïve Bayes), and 0.33 (KNN). Based on the results, we are confident that GCN is a superior label prediction model on highly interconnected data.

BACKGROUND

The core of most machine learning techniques is to use properly trained models to make accurate predictions. In most datasets where each observation is independent of one another, regular ML algorithms and classifiers are close to being the silver bullet to related problems. However, they fall short of problems concerning data that have a network structure, namely, data where independence of each observation does not hold. Sen et al. thus point out that traditional techniques ignore the correlations represented by the intrinsic interconnections within the data.[1]

[1] Sen et al., *Collective Classification in Network Data*, 2008.(http://eliassi.org/papers/ai-mag-tr08.pdf)

Thus, to efficiently and accurately represent the relationship between the observations, we need to consider approaches that solve this problem from different perspectives.

Compared to the performance of GCN that takes into account relationships between nodes and the strength of such relationships, the performance of traditional classification approaches is eclipsed. Another advantage of GCN is its capability of processing non-Euclidean data. According to Inneke Mayachita, while a lot of the real-world data are in graph structures that are non-Euclidean, traditional techniques can only be implemented using Euclidean data, such as images and time-series data. To address this problem, different variants of GNN are being developed in the past few years and GCN is one of them that can deal with the non-regularity of data structures.[2]

METHODOLOGY

As previously mentioned, the power of graph networks lies in its ability to represent relational information in a non-Euclidean space. This is extremely useful in mapping real-world objects or anything that exhibits a network-like structure, such as chemical compounds or a traffic system. What this convenience implies is lower interpretability and visualizability as graphs are generally not defined by dimensional levels. Instead, data representation of graph structure relies on two fundamental components: vertices (V) and edges (E). This idea is illustrated in the figure below:

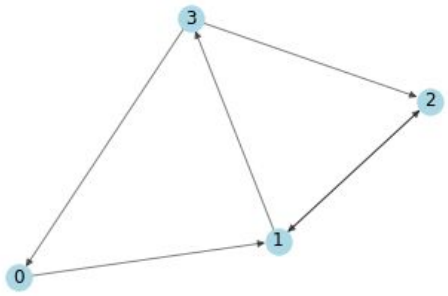


Figure 1. Simple Directed Graph

In figure 1, a vertex is also called a node. It is used to denote features or attributes associated with a particular vertex. For instance, a vertex in a social network like Twitter could represent recent tweets from an account. Note that all tweets should be preprocessed using methods such as LDA or word2vev so that the texts are projected into a high-dimensional feature space. On the other side, an edge is used to embed the relational information between two vertices[3]. This

[2] Inneke Mayachita, *towards data science,* 2020.(https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b)
[3] LINQS Statistical Relational Learning Group. (https://linqs.soe.ucsc.edu/data)

generally contains directional data, which could also be weighted. Using the Twitter example, an edge could refer to the account interactions in the form of retweets, comments, or likes. In summary, a graph convolutional network (GCN) is a neural network that operates on graphs. Given a graph G = (V, E), a GCN takes

1) a feature matrix, **X**, of size $N \times F^0$, where N is the number of nodes and F^0 is the number of input features for each node, and

2) an adjacency matrix[4], **A**, of size $N \times N$, that represents the graph structure. [1]

Graph Convolutional Network (GCN)

Unlike traditional Neural Networks, GCN leverages the idea of convolution in Convolutional Neural Network (CNN) while preserving the structural information within the original data. In a simple CNN, a set of kernels are convolved across an image to extract key information of contiguous pixels. This process is repeated for every hidden layer with kernels of different sizes. The goal then is to train these kernels so they could generalize to other images to perform tasks such as classification and recognition. This process is illustrated in the following figure:

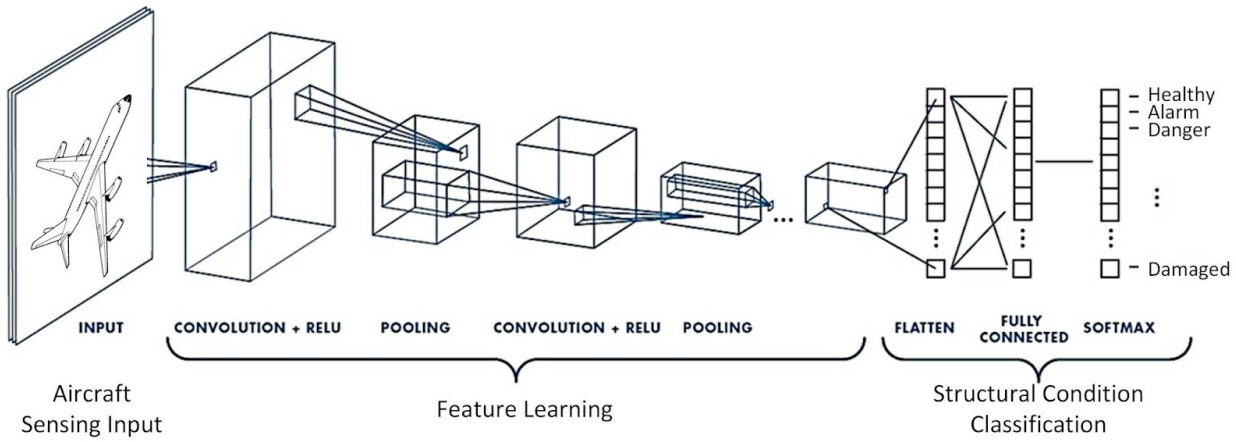


Figure 2. Convolutional Neural Network Architecture

In a simple GCN that contains only 1 hidden layer, convolution is done between a graph's feature matrix and adjacency matrix. This can be written as $H^l = f(H^{l-1}, A)$, where H^0 is the input feature matrix X of shape $N \times F^0$, and f is the propagation rule from input to output. For a GCN that contains multiple hidden layers, the features of the current layer are convolved to form the next layer's features following f. Generally, the number of features decreases as the network

[4] An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

becomes deeper so they could extract high-level information from each consecutive layer. For a classification problem, the number of features in the output layer should equal the number of classes.

The formula above can be further generalized into the following form:

$$f(H^l, A) = \sigma(AH^l W^l)$$

Where σ denotes an activation function such as ReLUand W^l is the layer-specific trainable weight matrix of the size $F^l \times F^{l+1}$. In other words, F^{l+1} determines the number of features in the next layer.

To better understand the role of the adjacency matrix and how it could be used to aggregate feature information, we can make a few assumptions to simplify this computation process[5]. More specifically, we assume an architecture with just 1 hidden layer so $f(H^l, A) = f(X, A)$, and use the identity function as the non-linear activation function σ. Additionally, the weight matrix is chosen to be a square matrix of 1s such that $AXW^0 = AX$. In the end, we get $f(H^l, A) = AX$ as the simplified propagation rule.

Using the simple network in figure 1, the following adjacency matrix (4 × 4) is obtained:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 3. Adjacency Matrix

We also create node-specific features for every node based on its index, which would be a 4×2 matrix shown below:

$$\begin{bmatrix} 0. & 0. \\ 1. & -1. \\ 2. & -2. \\ 3. & -3. \end{bmatrix}$$

Figure 4. Pre-Propagation Feature Matrix

[5] Jepsen, Tobias. https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780

We then apply the simplified propagation rule to our network, the final product would be a 4 × 2 matrix where each row represents post-propagation node features. And values in each row are a sum of neighboring nodes' features, similar to the algorithmic goal of a layer in CNN. Note that because W^l is deliberately set to a square matrix of 1s, the output shape remains the same:

$$\begin{bmatrix} 1. & -1. \\ 5. & -5. \\ 1. & -1. \\ 2. & -2. \end{bmatrix}$$

Figure 5. Post-Propagation Feature Matrix

The simple scenario described above demonstrates how the adjacency matrix is incorporated into the calculation of forward propagation. However, additional steps are needed to make GCN more robust. First of all, we need to add a self-loop to take each node's own feature into consideration when doing feature aggregation. This could be achieved by adding the Identity matrix to the adjacency matrix. On top of that, feature aggregation would automatically amplify the post-propagation feature values for nodes with larger degrees than the nodes with small degrees. This could lead to vanishing/exploding gradients which can be very problematic during training. To account for this, the feature matrix should be normalized by calculating the Degree Matrix and multiplying its inverse by the adjacency matrix. The overall process can be summarized in the formula below:

$$H^{(l+1)} = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

In the formula above, $\widehat{A} = A + I_N$ where I_N is the Identity matrix. $\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}$ depicts feature normalization, where $\widehat{D}$ represents the Degree matrix. (Note that symmetric normalization is used here instead of regular normalization, which intends to make dynamics more interesting)[6]

Data

The project uses the Cora dataset 2, which consists of 2708 academic publications. The Cora dataset is a type of citation network, where publications are interconnected through a citational relationship (if publication A cites publication B, then the graph has a link from A to B). The publications are encoded using bag-of-words without word counts, and there are a total of 1433 unique words in the vocabulary. Therefore, each publication is represented by a vector where 1 and 0 are used to indicate word presence and absence, respectively. The nodes are classified into

[6] Kipf, Thomas., and Welling, Max. *Semi-Supervised Classification with Graph Convolutional Networks*, (https://arxiv.org/pdf/1609.02907.pdf)

7 subjects (labels): Theory, Reinforcement Learning, Genetic Algorithm, Neural Networks, Probabilistic Methods, Case-Based, and Rule Learning. In conclusion, our feature matrix (document by term) is of size 2708 × 1433, and the adjacency matrix is of size 2708 × 2708.

Experiment Set-up

We first load the Cora network from *Spektral*, a Python library for graph deep learning. The Cora dataset contains three attributes: an adjacency matrix, a feature matrix, and 7 labels. To begin with, we convert the 7 labels from one-hot-encoding to single value encoding. Next, we create a color mapping for the 7 labels and convert the adjacency matrix to a graph. This plot demonstrates a high-level structure of the Cora dataset with each color indicating a label in one of the 7 categories. All plots can be found in the appendix.

After performing some basic introspection, we take a subset of the nodes for training and use the rest for validation and testing. We construct a 3-layered-GCN and define features and adjacency inputs. Our 3 GCN layers reduce the features to 64, 32, and 7 classes, respectively. To minimize the possibility of overfitting, we introduce a Dropout parameter to the network, which is a form of regularization that randomly drops out (by setting the activation to zero) a certain portion of output units from the layer in the training process. During this regularization process, we use a Dropout rate of 0.5 to reduce overfitting. Once the number of units is reduced, we use the Keras functional API to construct a GCN model. We also preprocess the adjacency matrix by adding self-loops and scaling edge weights. Since the training set has a class imbalance that might need to be compensated, we specify a weighted cross-entropy loss in the model, with class weights inversely proportional to class support. Additionally, we choose the *Adam* optimizer, a common optimizer to use in the early stages of setting baselines as it is more forgiving to hyperparameters. It is then used to measure how our model is updated based on the data it sees and its loss function. Last, we set the number of epochs to be 50 and convert the training data to a numeric array to prepare the training data to train the model.

Other Methodologies

In addition to GCN, we apply a few other common types of classification algorithms, including Multinomial Naïve Bayes Classifier, Logistic Regression Classifier, and K-Nearest Neighbors Classifier on the Cora dataset to predict the label. This also aims to examine how traditional classification algorithms differ from GCN in performance.

It is important to note that even though we attempt to compare the performance of GCN with other machine learning methodologies, such a comparison is not methodologically valid because of the fact that the objectives of these techniques are intrinsically different. Most traditional classifiers are supervised, which means that the model is trained on a large amount of data where the true labels are also provided, and consequently could be deployed to classify new observations. However, GCN is a semi-supervised learning method, meaning that the model is

trained on partially labeled data within a large network. More concretely, what distinguishes GCN from other techniques is that it takes advantage of the network nature of the data and the relational information, which is represented mathematically by an adjacency matrix. GCN then leverages this relational information and uses just a small portion of labeled data in this network to classify the remaining unlabeled data. However, this also marks the shortcoming of GCN which is that it cannot identify observations beyond the given network, of which the adjacency matrix has not been shown to GCN. In essence, what we are really doing here is controlling for the effect of this relational information and the algorithm.

The data preprocessing regarding the three types of classification techniques are similar to that of GCN. We first instantiate a classifier class from *scikit-learn*. For Naïve Bayes Classifier, we convert the labels from One-Hot Encoding back to integer and use ~5% of the original data for training. This ensures a training size of 140 observations and a test size of 2568 observations, which is the same as GCN. Keep in mind that this train-test split ratio is chosen solely to establish equivalence for comparison, because we are trying to isolate the advantage that GCN offers. The classifier is then fitted on training data, and evaluated on testing data. The same approach applies to the Logistic Regression and K-Nearest Neighbors Classifiers.

RESULTS

Our GCN model reaches an accuracy of 0.92 on the training data and achieves a test accuracy of 0.80, which we will use to label the nodes. It turns out that the accuracy on the test dataset is a little less than the accuracy on the training dataset. This gap between training accuracy and test accuracy represents overfitting. We apply the same color code as mentioned in the methodology section to plot the structure of the nodes as a visual representation of the prediction result. To better assess the performance of the model, we visualize false predictions by color coding the falsely predicted labels red. Red nodes mostly appear towards the tip of a network branch. This makes intuitive sense since these nodes have much less connections and their relational information would be less robust in comparison with nodes located at the center.

Among the results from all four types of classification algorithms, GCN has the highest test accuracy - 0.80, followed by Logistic Regression Classifier - 0.59, Naïve Bayes Classifier - 0.58, and K-Nearest Neighbors Classifier - 0.33. It is interesting to note that in figure 6, the training accuracy is higher when using traditional classifiers. One possible explanation is that the words used for each class of publications are very distinctive in the 140 training data, so patterns could easily be captured by these classifiers. However, this could inevitably lead to overfitting, which is reflected by the much lower testing accuracies. On the other hand, despite a lower training accuracy, GCN distills structural information in addition to native features of each observation that could generalize better to testing data.

Method	Result
Graph Convolutional Networks (GCN)	Mean accuracy on training set: 0.92 Mean accuracy on testing set: 0.80
Naïve Bayes Classifier (Multinomial)	Mean accuracy on training set: 0.99 Mean accuracy on testing set: 0.58
Logistic Regression Classifier	Mean accuracy on training set: 1.00 Mean accuracy on testing set: 0.59
K-Nearest Neighbors Classifier	Mean accuracy on training set: 1.00 Mean accuracy on testing set: 0.33

Figure 6. Performance Results Summary

CONCLUSION

Our analysis has revealed that GCN is indeed a powerful machine learning model for data with a network structure. The power of GCN lies in its ability to infer correct labels with a small training set and a network's relational information. By including the interconnected edges as an adjacency matrix in the model, GCN successfully captures the high-level relationships between nodes and offers valuable insights for nodes that have not been exposed to the model. In comparison, traditional methodologies are built using much larger training sets to avoid underfitting, which could be expensive to collect and train on. Therefore, it is promising to envision the opportunities this method can create in fields such as Natural Language Processing, recommendation systems and social network analysis.

SUMMARY OF TEAM ROLES

Shangwen Yan: Presenter/checker/coordinator/writer

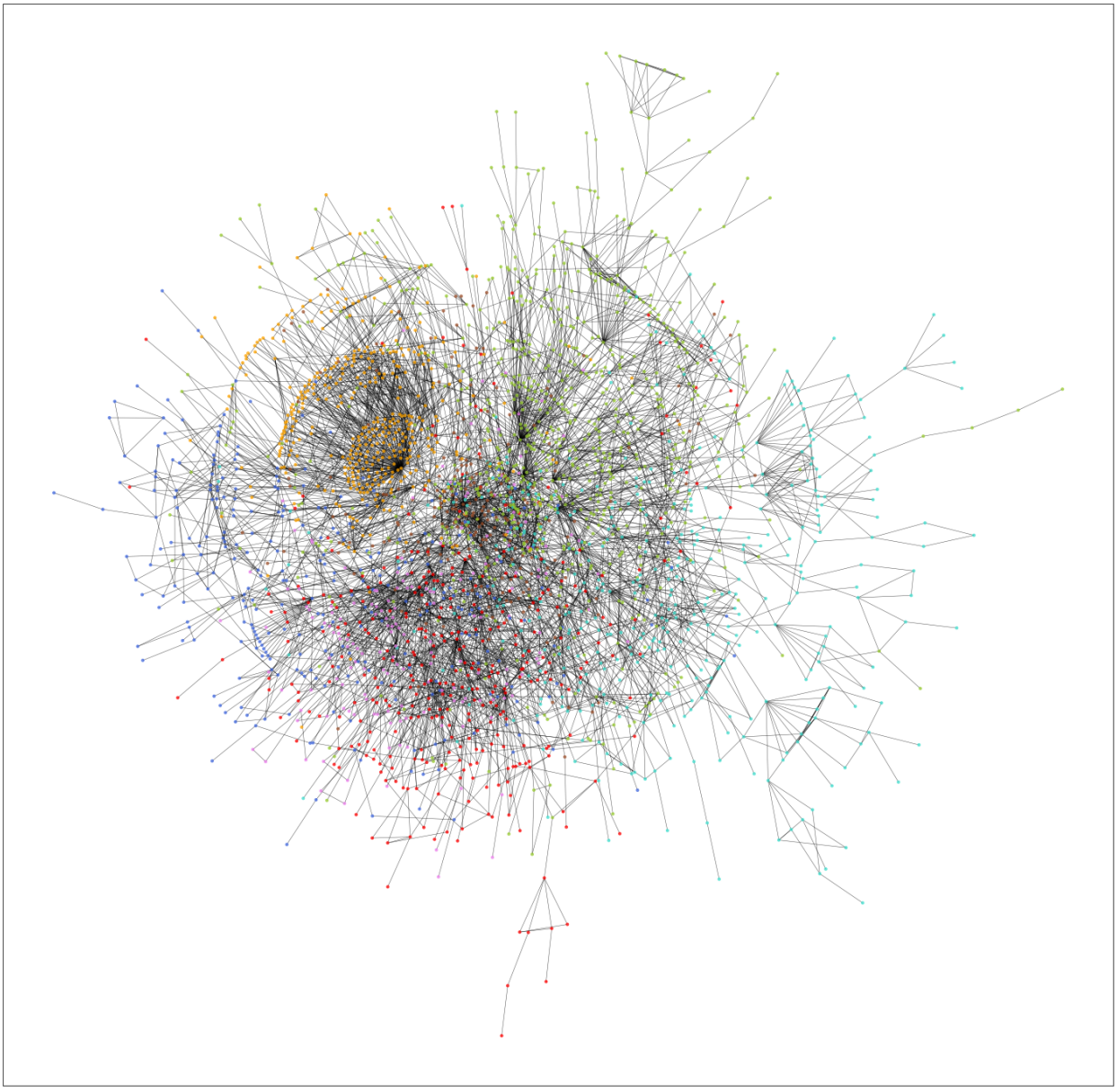
Jennie Sun: Presenter/checker/coordinator/writer

Michael Tang: Programmer/presenter/checker/coordinator/writer

Yi Feng: Presenter/checker/coordinator/writer

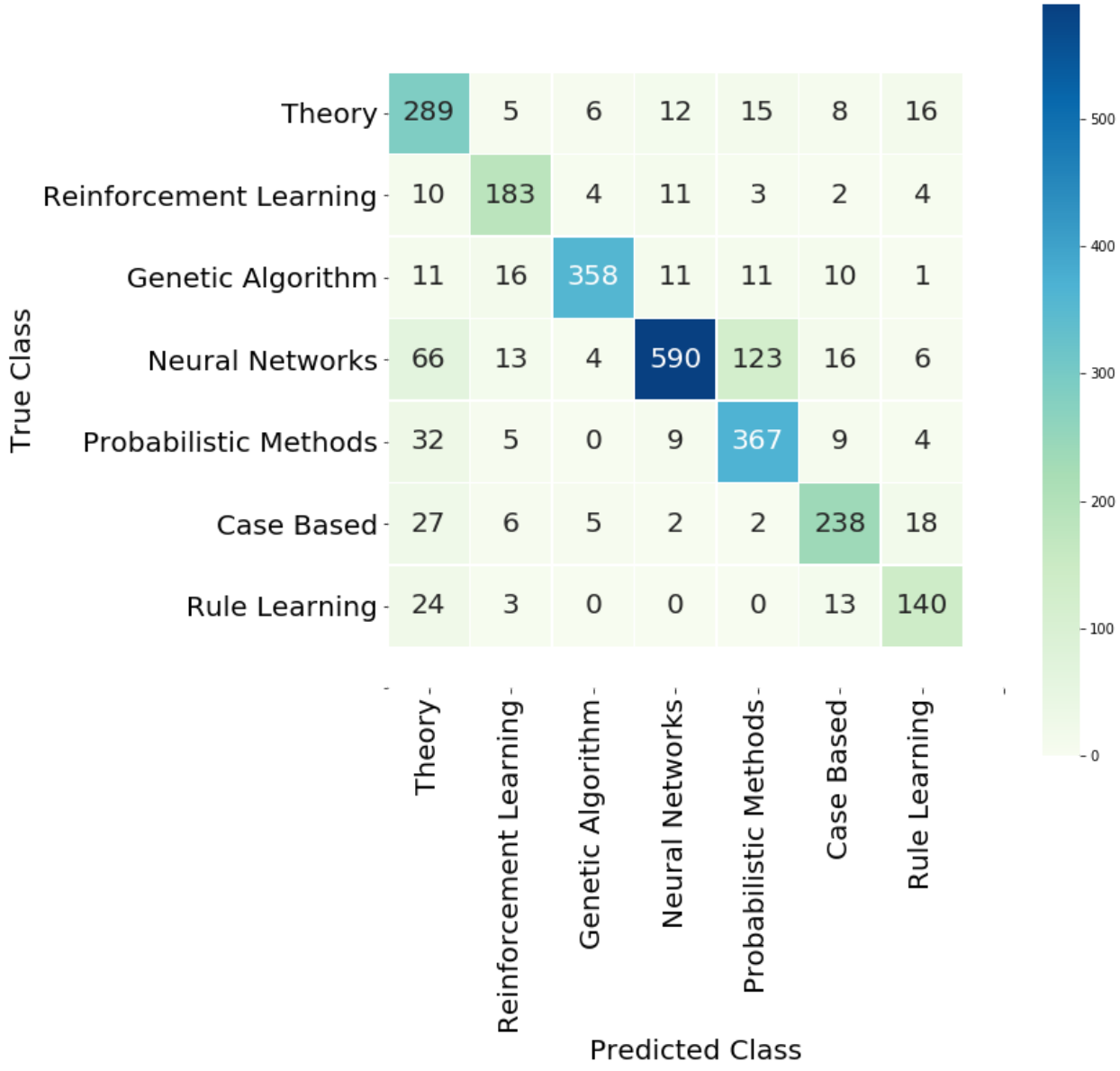
APPENDIX

Cora Dataset with True Labels

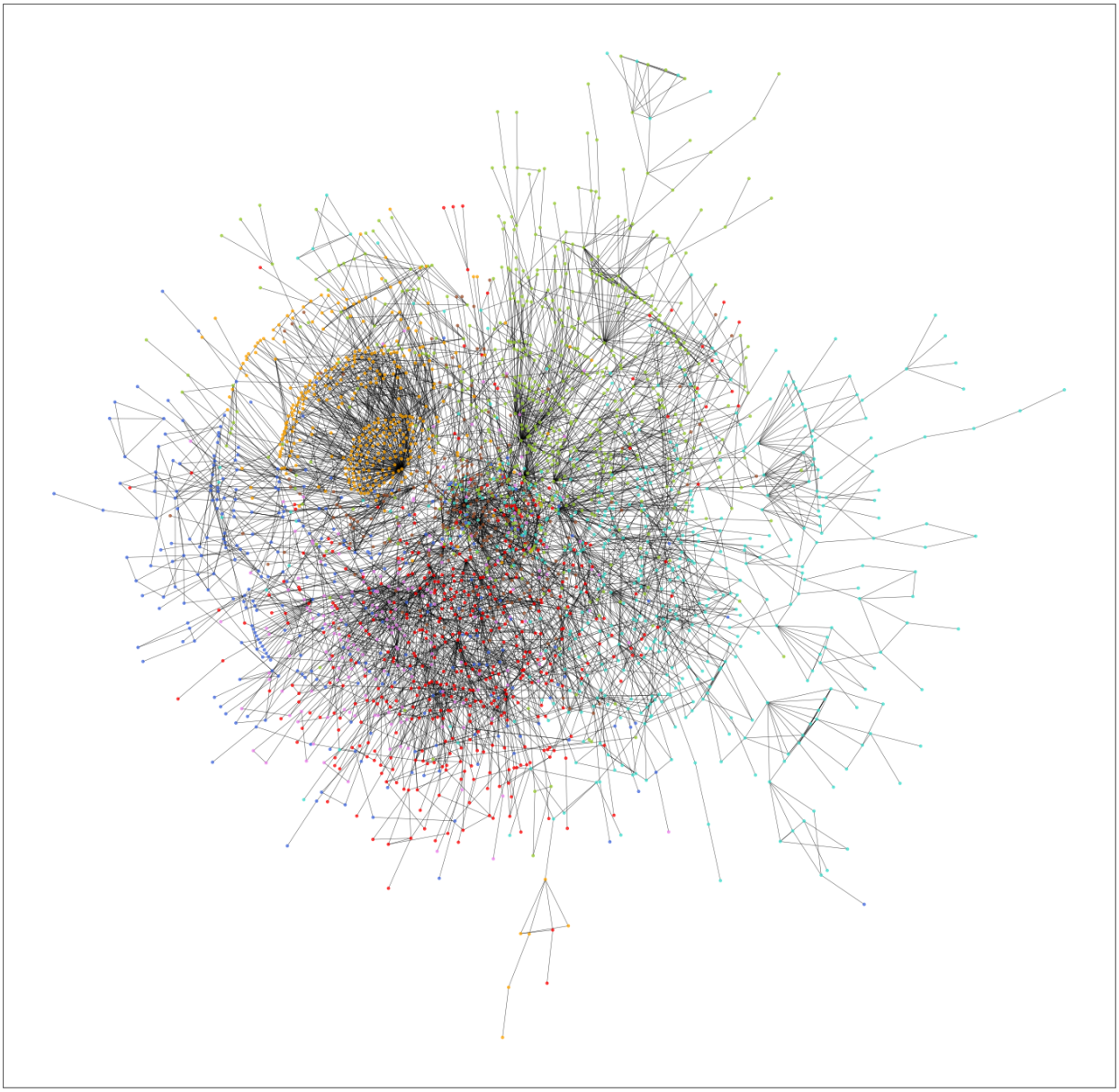


GCN

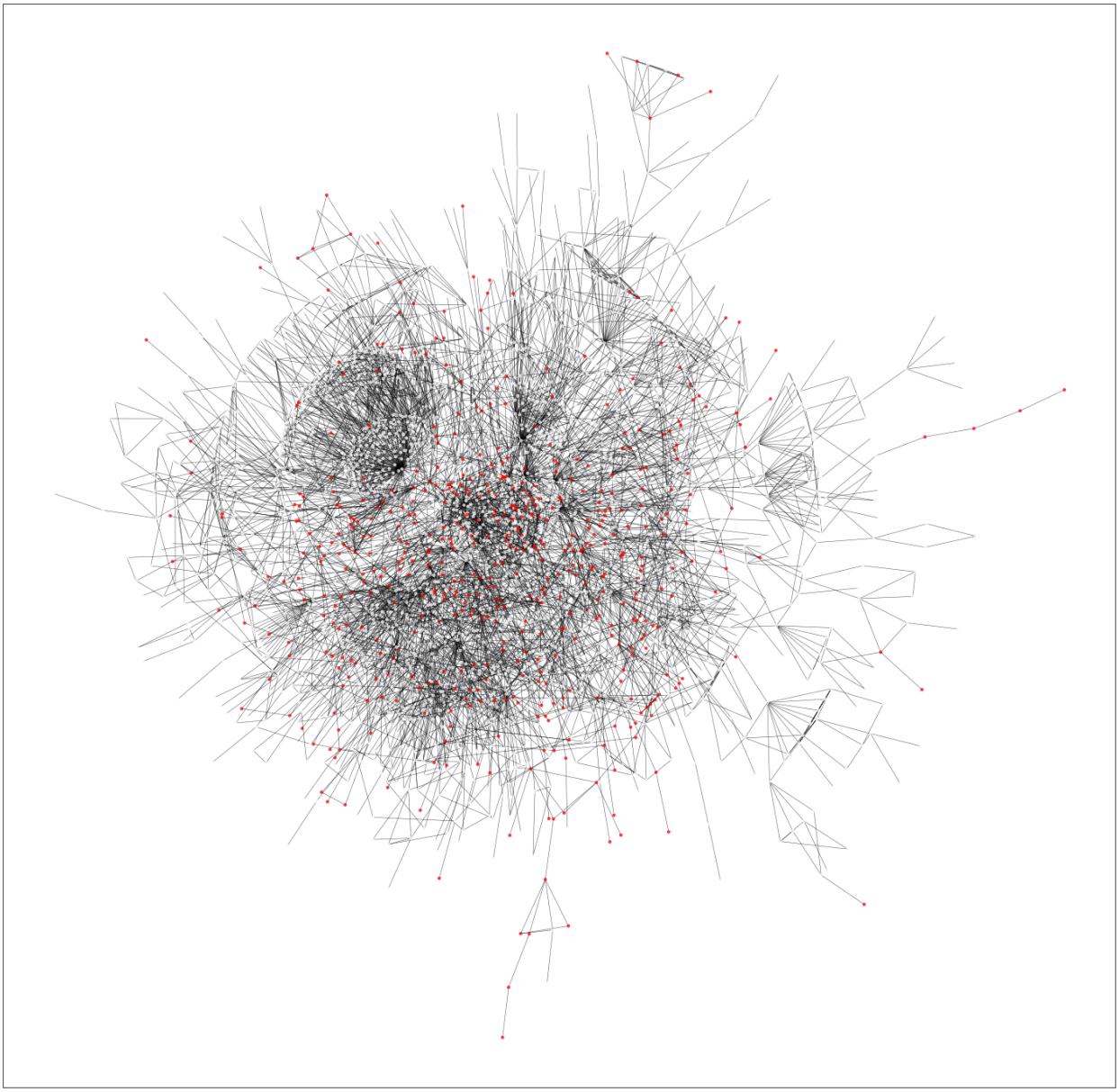
GCN Model Prediction Confusion Matrix Heat Map



GCN Predictions Plot

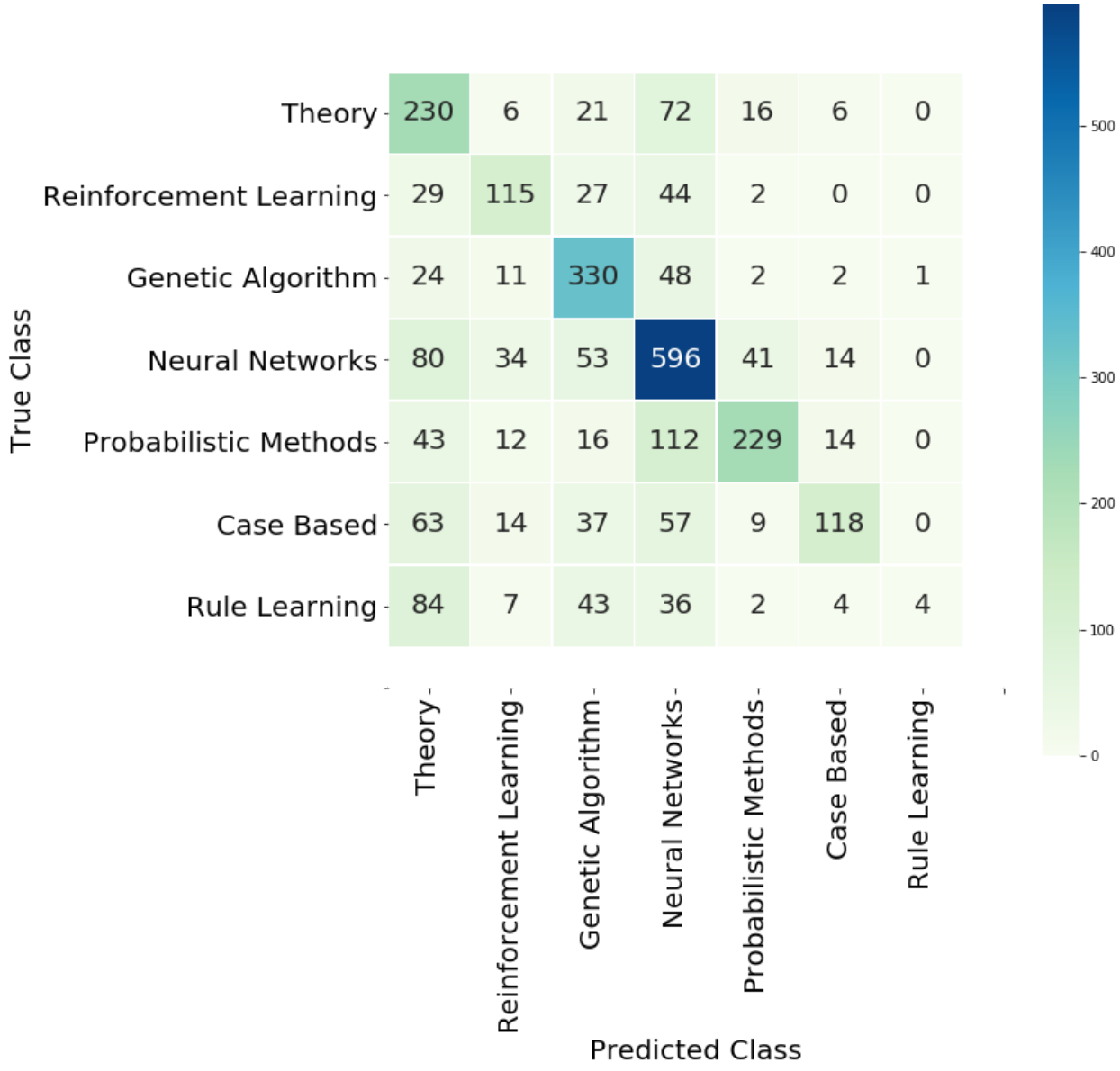


Visualize False Predictions in GCN Model

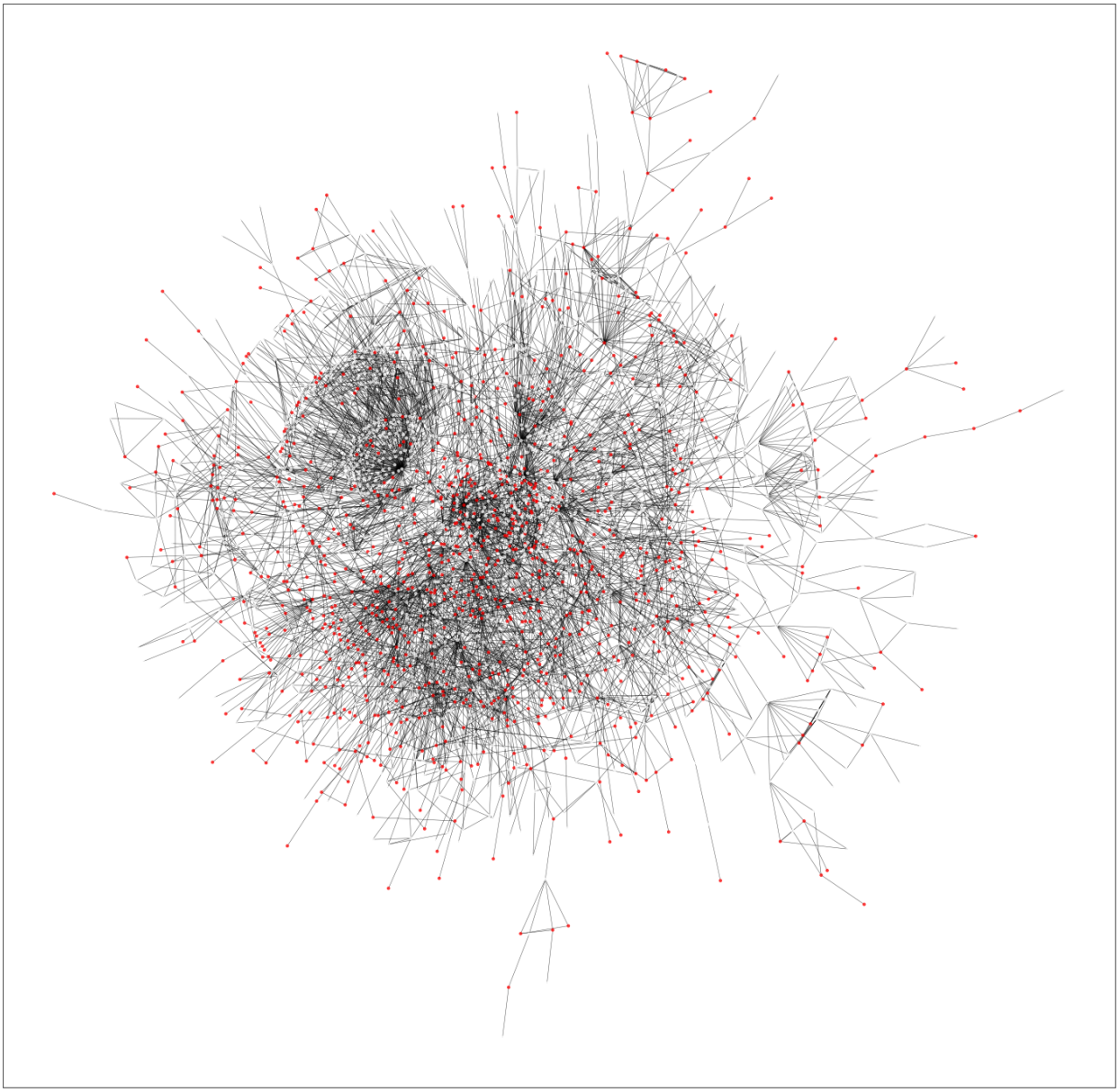


Naive Bayes Classifier

Naive Bayes Classifier Model Predictions Confusion Matrix Heat Map

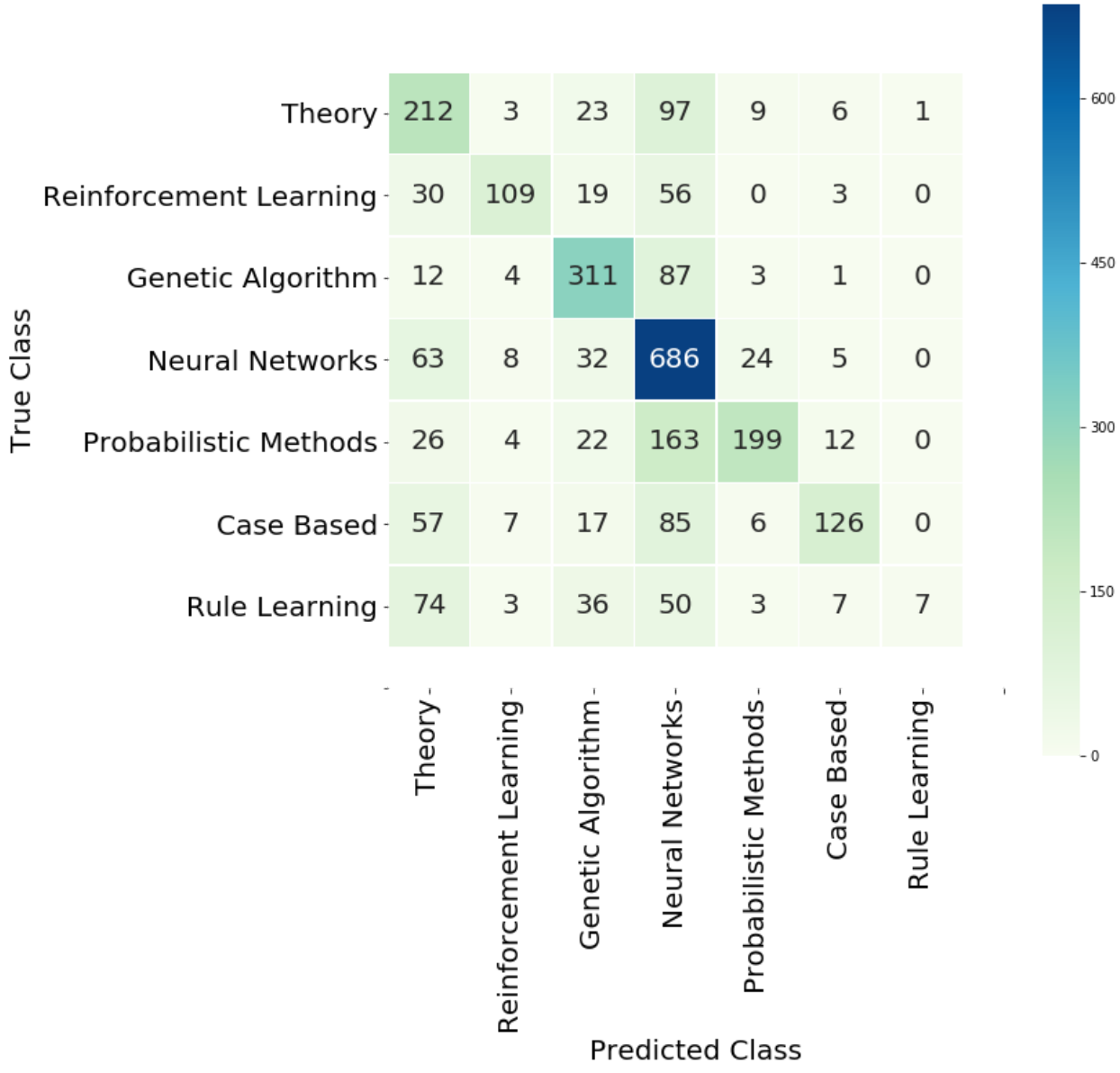


Visualize False Predictions in Naive Bayes Classifier Model

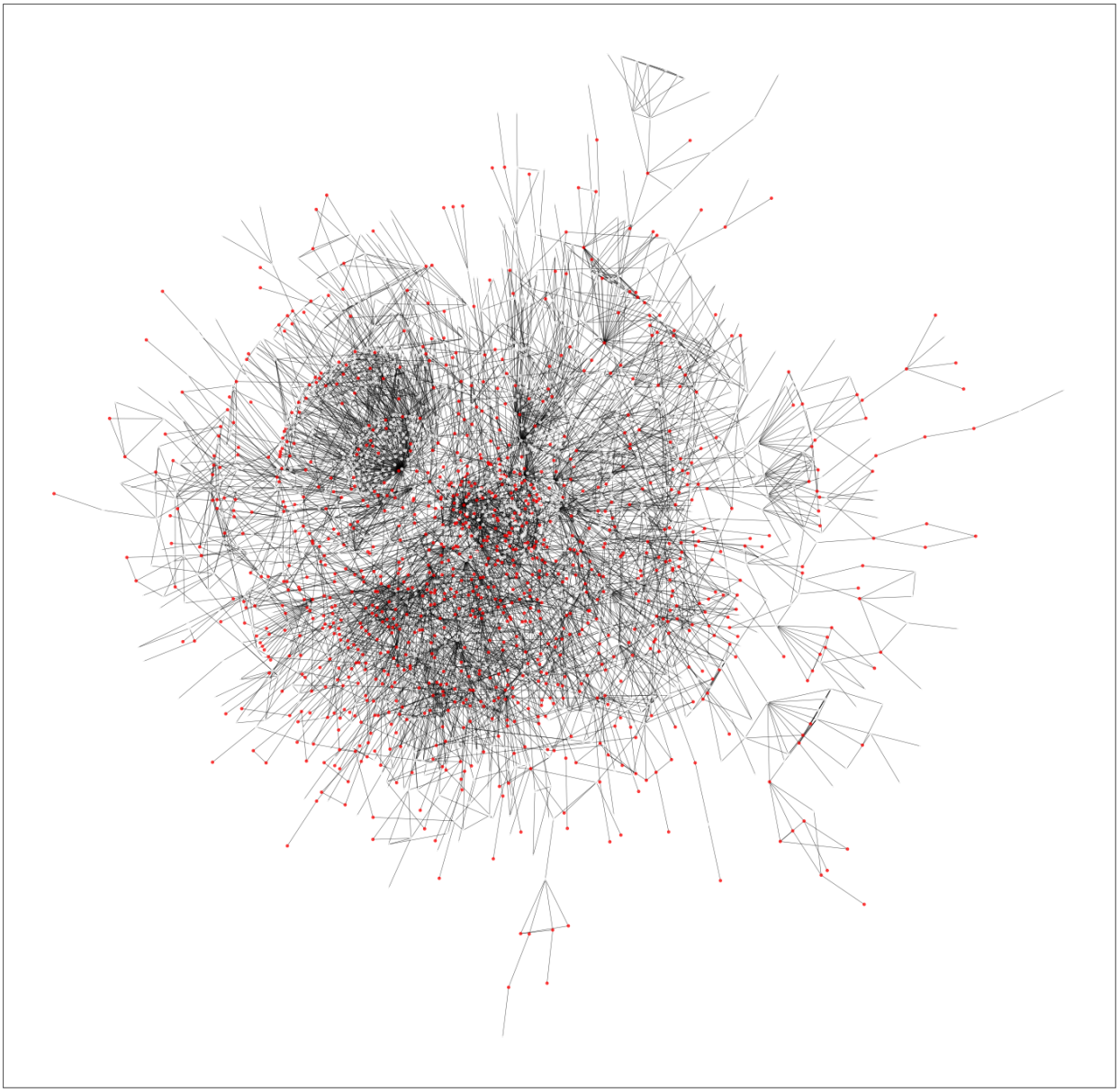


Logistic Regression

Logistic Regression Classifier Model Predictions Confusion Matrix Heat Map

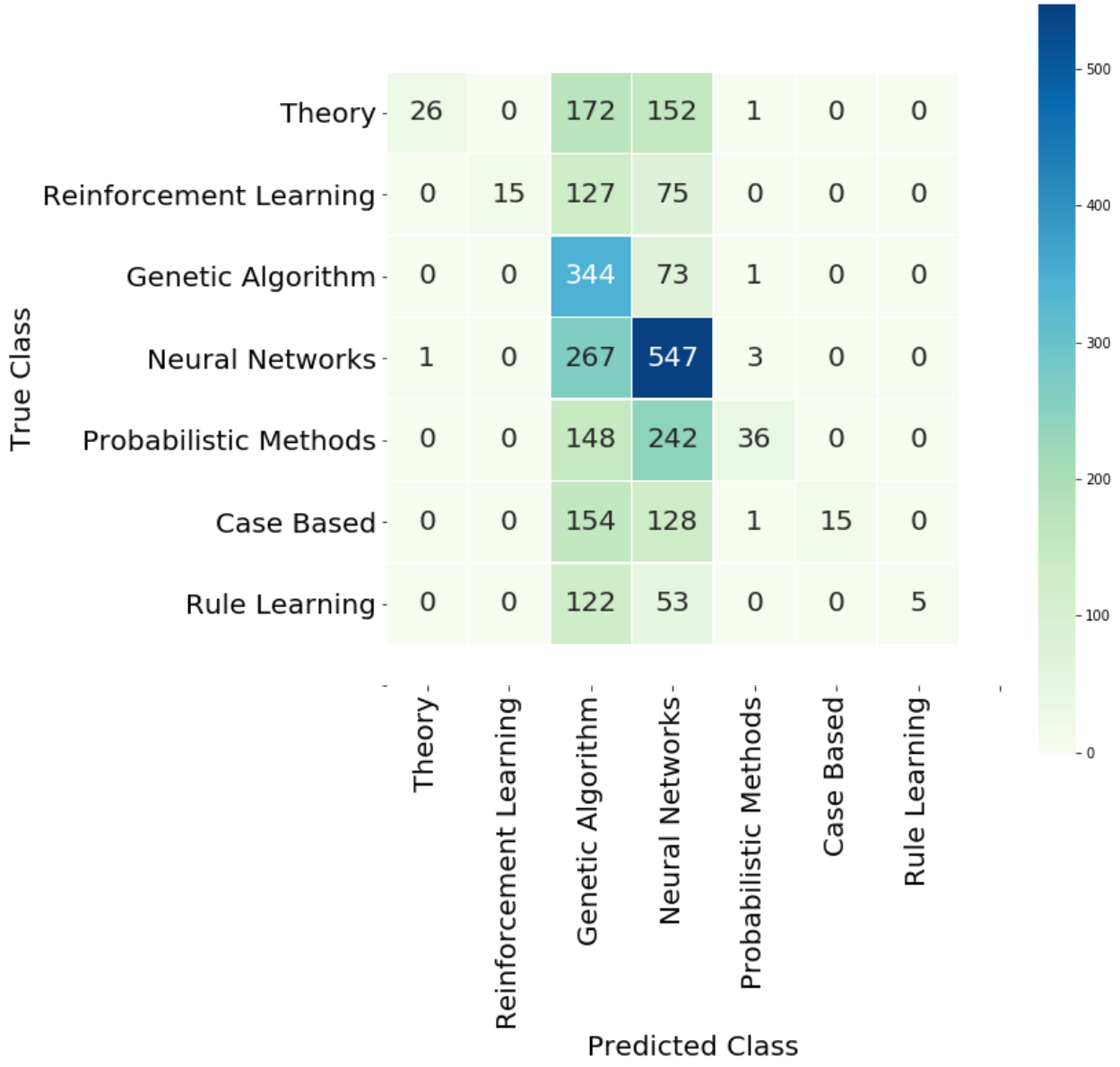


Visualize False Predictions in Logistic Regression Classifier Model



K-Nearest Neighbors

K-Nearest Neighbors Classifier Model Predictions Confusion Matrix Heat Map



Visualize False Predictions in K-Nearest Neighbors Classifier Model