Inferring Nodes from Graph Neural Networks

Mihir Khandekar, Reza Shokri
School of Computing
National University of Singapore (NUS)
{*mihir, reza*}*@comp.nus.edu.sg*

***Abstract*—The usage of graph neural network architectures to perform tasks such as fraud detection in transaction networks, social network recommendation systems, narcotic molecule detection, etc. on graph data has recently been gaining popularity due to their performance. The sensitive nature of some of the data stored in the form of graphs has brought attention to their privacy issues. In this study, we demonstrate a system where an adversary with black-box access to a node classification graph neural network model can infer the nodes used during the training of the model. We do this by proposing an attack model which can differentiate between the nodes which are members and non-members of the training set. Our attacks assume different levels of knowledge for the attacker to evaluate the attack model empirically. We further analyse the properties of graph data and models which may be more prone to such attacks.**

***Index Terms*—graph, privacy, membership inference attack, machine learning, graph neural network**

1. Introduction

The use of machine learning to solve real-life problems and make systems more efficient has exploded in the past decade. One can now see machine learning systems being used across industries to solve varied tasks such as online shopping recommendation systems, language translation services, Elliptic fraud detection or even to guide automated driverless vehicles.

Traditionally, machine learning has been making use of data such as images, tabular data or natural language data to perform tasks such as language translation, fraud detection or object detection in images. Besides these, an important type of data, graph data has been gaining popularity more recently. This rise could be attributed to the rise in NoSQL systems such as Neo4j, which can efficiently store and query graphs, as well as the availability of algorithms to perform various tasks on graphs. Graph data enables its users to store not only entities (such as features of the entity), but also the complex relationships between them. An example of this may be a social network graph, where the entities (users) have different properties, along with relationships (connections/friendships) between users. Social networking sites such as Facebook, MySpace, etc, Citation Networks such as Citeseer, CORA or Elliptic transaction networks have all started utilizing graph databases to store their vast amounts of data.

Since data was being stored and manipulated in the form of graphs, traditional machine learning algorithms were deemed to be inefficient since they did not utilize the relationships and structures of the graphs to perform the tasks. This led to the development of graph neural networks (GNNs), a type of artificial neural network architecture that used the structure of the graph and the relationships within a graph to aid in performing tasks on graphs. These tasks may include recommending friends in a social network, to detecting fraudulent transactions in a social network to identifying molecules with narcotic properties.

The sensitive nature of the data stored in the form of graphs and used by GNNs to perform tasks have brought into discussion the security and privacy of these algorithms. A lot of work has been done on the privacy risks of artificial neural networks in the image, text and tabular domain. More recently, there has been an interest to extend such analyses to data and algorithms in the graph domain [1]. In particular, there has been work on analysing attacks such as poisoning attacks, where an attacker is able to ”poison” a graph (Such as modifying a small number of edges or nodes) to drastically alter the model's output; evasion attacks, where an attacker is able to create adversarial examples on the model which appear normal, but give results that are unexpected for the type of data. Membership Inference Attacks are another class of attacks, where an attacker tries to infer if some data was part of the data used for training the machine learning model. This attack is important to study since the data used while training machine learning models are real data belonging to actual people. Leakage of this data may result in serious privacy-related issues in both an ethical and legal view.

Our contributions

In this work, we analyse the privacy implications of membership inference attacks on graph neural networks. In particular, we use a neural network solving the task of node classification using a Graph Convolution Network (GCN), one of the most popular architectures used for node classification tasks. Assuming an attacker has black box access to the graph neural network model, we propose an attack methodology that enables the attacker to infer the nodes used while training the model with an accuracy enough to pose a serious threat to the privacy of the participants of the model.

While analysing this privacy leakage, we assume different levels of knowledge of the attacker and discuss how different classes of knowledge contributes to the privacy leakage. We further try to understand how our

observations may be correlated with the architecture of the model. We then do a comprehensive analysis of the strongest attack of the attacker and discuss how various parameters of the model, properties of the graph data and model access of the attacker affect the classification accuracy of the attack inference model.

In Section 3, we introduce graph data, graph neural networks and membership inference attacks. In Section 2, we discuss the previous work done on membership inference attacks on various kinds of data, as well as the current state-of-the-art attacks on graph data. We define the problem statement and objective of this attack In in Section 4 and the attack setup and the types of attacks based on the attacker's knowledge in Section 5. We discuss the results of our attacks in Section 6, and do a comprehensive analysis of these results in Section 7.

To summarise, we make the following contributions.

- We propose a methodology for node membership inference attacks on graph neural networks used for node classification
- We propose different types of attacks based on different levels of attacker knowledge
- We analyse the types of data and models that are more prone to such attacks

2. Related Work

3. Background

Since a lot of data is represented more effectively in the form of graphs, graph datasets have been adopted in the storage of social networks, citation networks, financial transaction data and much more. Due to this, graph neural network architectures were developed to perform tasks on graph data.

A graph is a data structure which consists of a set of nodes connected by edges. The edges can be represented by an adjacency matrix $A_{n\times n}$, where n is the number of nodes in the graph, and $A_{i,j} = 1$ if an edge between node i and node j exists, and is 0 otherwise. Edges can be directed or undirected, with the sign of the element in the adjacency matrix denoting the direction of the edge. An undirected graph results in a symmetric adjacency matrix.

Graph Neural Networks

To address the problem of performing machine learning tasks on graph data utilizing the structure of the graph, several graph neural network architectures were proposed. These architectures proved to be much better at such tasks, compared to sequential models or other neural network architectures, on graph data.

Graph neural network tasks are done on three different levels - graph, node and edge. A task on the graph-level could be to classify a graph representing a molecule as a narcotic substance; a task on the node-level could be one to classify users of a social network as fraudulent; a task on the edge level could be one to classify a transaction in a financial transaction network as illicit. These tasks may be done in a supervised manner, such as when a dataset of molecule graphs are provided with the labels as narcotic/non-narcotic, or in a semi-supervised manner, when a large graph social network dataset is provided with some of the nodes labelled.

The main intuition behind most graph neural network architectures is the aggregation of node features. In brief, each node's neighbouring node's features are iteratively aggregated with the node features to get an intermediate representation of a node. This intermediate node representation is either used directly as an input for node classification tasks, or a combination with other intermediate representations is done for edge classification and graph classification tasks. It has also been proven that graph neural networks are atmost as powerful as the Weisfeiler-Lehman (WL) test, which is a test which is used to detect non-isomorphism of two graphs using a graph colouring approach. In our paper, we consider the case of node classification in a semi-supervised training setting.

A graph convolution network (GCN) is one of the most popular graph neural network architectures for node classification. Here, aggregation of nodes is done using a form of message passing between neighbouring nodes. This is done using approximated spectral convolutions of the graph. The approximation is done for better performance, to avoid the eigendecomposition of the Laplacian of the graph matrix. The convolutions are performed for a configured number of iterations to get the intermediate representation of the nodes. The main intuition behind this aggregation is the feature smoothing across a neighbourhood. This can be compared to convolutions in images, where a neighbourhood of a pixel is learned using a learned filter. Further, the intermediate representations are sent as an input to a dense neural network, to learn the predicted label of the nodes.

Each layer in GCNs follows the below aggregation equation.

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

Here, $\tilde{A} = A + I$, which represents the adjacency matrix with added self-connections. $\tilde{D}_{n\times n}$ is the degree matrix, which is a diagonal matrix with the diagonal elements representing the degree of a node N_i, i.e, $\tilde{D}_{i,i} = degree(N_i)$. $H^{(l)}$ is the hidden representation of the node at the l^{th} layer. Here, $H^{(0)}$ is equal to the node feature vector. Further, $W^{(l)}$ is a trainable weight matrix. The final hidden representation, $H^{(n)}$, which can also be called the node embedding, is passed through dense layers to get the final task output.

Membership Inference Attacks

Since a lot of critical data is being stored in the form of graphs, it is possible that the data may become the target of adversaries. For instance, adversaries may attempt to poison the dataset to manipulate the task results, or try to infer the training data from the dataset. These threats have brought concern for the privacy and security of graph data, especially when there is access available to query and get the task outputs from the models. Since training data of graph-based tasks such as the participants of the dataset, or the relations between them are not intended for public use, leakage of such information is a serious privacy concern.

Neural network models tend to memorize the training data in order to better fit to the data and get a better result. This is even more prevalent for data with a long tail in the class distribution, where some data has to be memorized to be able to get a better test accuracy for the task. However, this memorization of the data may cause some unintended membership leakage. Membership inference seeks to answer the question if some data point was used while training a model. These attacks often make use of the data provided by the model, such as the model output logits, inputs and intermediate layer outputs. The access to the model of the adversary is classified between black-box and white-box, depending on the level of access to the model outputs and parameters. A black-box access assumes access to just the model outputs and inputs, while a white-box access may assume access to additional parameters such as intermediate layer outputs of the model while querying. The adversary attempts to use the data to classify its input data point as a member or non-member of the original training set.

Differential privacy is a method to provide a formal privacy guarantee, to quantify the privacy leakage of a model. It helps protect user data by methods such as adding noise while training using SGD. If such noise is added while training the data, then the various attack signals such as prediction logits, layer outputs, etc. become indistinguishable between the training data, and non-member data, making a membership inference attack ineffective.

In the past, membership inference attacks have been limited to models performing tasks on more conventional types of data, including text, images and tabular data. More recently, various kinds of attacks on graph neural networks have also been gaining interest from the community.

4. Problem Statement

We propose an attack methodology for node membership inference attacks on graph neural networks trained for node classification tasks. In particular, the attack is used by an adversary to infer whether a particular node in a graph was used while training a graph neural network.

4.1. Terminology

Given a large graph dataset $G(N, E)$ containing a set of nodes $N = \{n_i\}; i \in |N|$ and edges $E = \{e_{jk}\}; j, k \in |N|$ (where node j and node k are connected in the graph), a graph neural network model f is trained to perform a node classification task on a subset of this graph dataset. The node feature vectors are represented by $X_{n \times m}$, where the feature vector has length m, and the label vector is represented by $C_{n \times p}$, where the task output has p classes. We assume graph G is an undirected graph.

Each node n_i is represented by its feature vector x_i and label c_i. The edges are represented by an adjacency matrix $A_{n \times n}$, where $A_{ij} = 1$ if edge e_{ij} is present in the graph, else $A_{ij} = 0$. Since graph G is undirected, we can say that A is a symmetric matrix. Further, $\tilde{A} = A + I$ represents the adjacency matrix A with self-connections added for each node.

Next, we assume that a subset of nodes N have been used to train a GNN model f. We call this subset of nodes N_{IN}, with the edges connected to the included nodes as E_{IN}. This subgraph is denoted as $G_{IN}(N_{IN}, E_{IN})$. The nodes that are not members of the training set are denoted as N_{OUT}. A GNN model $f(G_{IN}, N)$ is trained using the subgraph G_{IN}, to perform the task of node prediction i.e, $f(G_{IN}, x_i) = y_i$, where y_i is the predicted label by the model for input features x_i.

4.2. Attack Objective

The study assumes that an adversary has a black-box access of the trained target GNN model f. The attacker can query the model f using a graph along with the feature vectors of the nodes, to get the node prediction logits. Assuming varying levels of knowledge for the attacker (Discussed in Section 5.3), the attacker tries to infer if the nodes he queried were part of the graph dataset used to train the model f. That is, the attacker attempts to infer if some node $n_k \in N_{IN}$, where nodes n_k has a feature vector x_k. The attacker may or may not know the actual label of the node c_k, its connected edges $e_{kl}; l \in |N|$. The attacker's level of knowledge may affect the accuracy with which the attacker infers node membership.

5. Experimental Setup

5.1. Target Model

In our experiments, for the target model, we use a graph convolutional network (GCN), one of the most popular GNN architectures used in node classification tasks. We perform the attack using different number of hidden layers. The training uses a dropout of 0.2 for better generalization. The model is trained for 50 epochs for the Facebook, CORA and Citeseer datasets, 250 epochs for the PubMed dataset and 500 epochs for the Elliptic dataset, with early stopping. The proportion of nodes (with their edges) from the graph used for training the model is 30% for CORA, Facebook and Citeseer datasets and 10% for the Elliptic and PubMed datasets. Details about the datasets are presented in Section 6.1. Further, we evaluate other GNN models for node classification such as Graph Attention Network (GAT).

5.2. Reference Model

As a reference model, We train a sequential model f_r to compare the test accuracy of the GNN target model and results of our attack model with. This model is trained to perform the node classification task using a Sequential Keras model having 2 hidden layers of size 64 and 32 respectively. The number of epochs and other hyperparameters are similar to that of the graph target model used for each dataset respectively.

5.3. Attacker Knowledge

The attacker has a black-box access to the target model f, where the attacker is able to query the target model using his knowledge and gets the prediction logits in

return as output. These prediction logits, along with the other knowledge the attacker has, is used to train a attack model g, which is used to infer membership of a node. The attack methodology and attack models for various levels of attacker knowledge are explained in the following subsections.

5.3.1. Unsupervised Inference. The first attack assumes that the attacker has node knowledge of a subset of data from the graph G, N_{IN} and N_{OUT}. This is the most difficult attack setting for the adversary. We assume that nodes from the IN set and OUT set are randomly selected with its edges in an equal proportion, and are available to the attacker in a shuffled state. The attacker does not have knowledge of the membership of these nodes. We denote the node knowledge that the attacker may have as N_{know} with feature vectors X_{know}. We assume that $|N_{know}| \leq 0.4 \times |N|$ by default. However, we also vary this ratio of attacker knowledge to analyse its effect on attack accuracy, which is discussed in Section 7.4.

In this setup, the attacker may query the node knowledge that he may have with the target model f to get the prediction logits. Since the attacker does not know the membership information about the known nodes, he trains an unsupervised model on the entropies and binary crossentropy loss for the node predictions. The normalized entropy for prediction logits Y is calculated using Equation 1.

$$normalized\_entropy(Y) = \sum_{i \in |Y|} \left(y_i * \frac{\log(y_i)}{\log(|Y|)} \right) \quad (1)$$

The unsupervised model is trained using the K-means clustering algorithm to divide the datapoints into two clusters. These clusters are labelled as IN and OUT, and the attack accuracy is calculated. Algorithm 1 shows the method to generate data for the attack using the target model.

Algorithm 1 Unsupervised attack methodology

```
loss_entropy = []
function attack_data(n_know, e_know, c_know):
    generate g_know from n_know and e_know
    logits = f (g_know, n_know)
    loss = crossentropy_loss(logits, c_know)
    entropy = normalized_entropy(logits)
    loss_entropy.add([loss, entropy])
    return loss_entropy
clusters = kmeans(loss_entropy, num_clusters=2)
```

5.3.2. Supervised Inference (Shadow Model). In this attack, we assume that the attacker has complete knowledge of a shadow model. This includes its nodes, feature vectors, edges and labels. We denote the shadow graph as $G_{shadow}(N', E')$ having nodes $N'(X', c')$. To train an attack model, the attacker first separates the nodes which he uses to train a shadow model, and its adjacent edges, called as N'_{IN} and E'_{IN} respectively. The nodes which are not used while training are represented as N'_{OUT}.

The graph used for training shadow model f' is denoted by $G'_{IN}(N'_{IN}, E'_{IN})$. The shadow model uses a similar architecture to the target model explained in Section 5.1, including the hyperparameters such as number of layers, activation function, etc.

The attacker now queries this shadow model f' using the nodes from both N'_{IN} and N'_{OUT} to get the prediction logits, which are used to train an attack model. Since we are using a different dataset for the shadow model than for the target model, we cannot use the prediction logits directly for the attack model, since they have different sizes (due to different number of classes) and different class distributions. Hence, we use the normalized entropy (Equation 1) to train an attack model which tries to distinguish between entropies between the member and non-member sets. A support vector machine (SVM) model is used to generate a threshold to perform this inference. Algorithm 2 shows the attack methodology.

Algorithm 2 Attack methodology with subset knowledge

```
f_shadow = GCN().fit(G_shadow, N_in)
logit_in = f_shadow(G_shadow, N_in)
entropies_in = normalized_entropy(logit_in)
logit_out = f_shadow(G_shadow, F_out)
entropies_out = normalized_entropy(logit_out)
entropies = [entropies_in, entropies_out]
membership = [[1 for range(len(entropies_in))], [0 for
range(len(entropies_out))]]
g = SVM().fit(entropies, membership)
function get_membership(f, n):
    y = f(G, n)
    logits = f(g_know, n_know)
    entropy = normalized_entropy(y)
    return g(entropy)
```

5.3.3. Supervised Inference (Subgraph). This attack assumes that the attacker has knowledge of a subgraph of data from the graph G, including the knowledge of the membership of these nodes. We denote the node knowledge that the attacker may have as N_{know} with feature vectors X_{know}. We assume that $|N_{know}| \leq 0.4 \times |N|$ by default. Simliar to the unsupervised attack, we vary this ratio of attacker knowledge to analyse its effect on attack accuracy.

This attack is the strongest since it provides the maximum knowledge to the attacker. The attacker queries the model f to get the losses, prediction logits and normalized entropy of the prediction logits to train a Multi-layer perceptron model to distinguish between the members and non-members. This MLP model has 2 hidden layers with 32 and 16 layers respectively.

5.4. Attack

The attacks discussed in the previous sections are performed on the 5 datasets discussed in Section 6.1, and run 15 times with the results averaged to get the attack performance. The attack models calculate a threshold for the attack parameters such as loss, logits or entropy, which is used to classify for membership.

5.5. Metrics

We use the AUC (Area under the curve) metric of the ROC (Reciever Operating Characteristics) to quantify the

TABLE 1: Dataset Descriptions

Dataset	# Nodes	# Edges	Feature vector size	# Classes
CORA	2708	5429	1433	7
Citeseer	3312	4732	3703	6
PubMed	19717	44338	500	3
FB	22470	171002	1666	4
Elliptic	203769	234355	166	3

TABLE 2: Test accuracy of Target Model

Dataset	Test Accuracy
CORA	0.75
Citeseer	0.74
PubMed	0.7
FB	0.82
Elliptic	0.74

performance for the attack model in inferring the node membership. AUC is a commonly used metric for binary classification tasks such as ours to understand the trade-off between the true positives and the false positives, rather than using the overall accuracy as a metric. We use the test accuracy for the target model.

6. Empirical Results

6.1. Datasets

We use 5 datasets from different domains to evaluate our attacks. We use CORA, Citeseer and PubMed datasets, which are commonly used benchmark datasets in the domain of graph node classification. These datasets are citation networks, where research publications are represented as graph nodes, and citations are represented by the edges. The node labels are the different categories the papers are classified into. Further, we use the Facebook dataset[1], which is used to represent connections between pages on the social networking site, with the node labels representing the type of the page. These graphs use one hot word vectors for the graph nodes as the feature vector. The final dataset used is the Elliptic bitcoin dataset[2], where the transactions between users are represented in the graph. The user features are represented in the form of feature vectors and the dataset is used to identify the nodes which are illicit.

Table 1 provides details about these datasets.

In the shadow model attack, each of these datasets is used as a shadow dataset to train a shadow model for the other datasets.

6.2. Target Model

We first train the target Graph Convolution Network (GCN) model using the hyperparameters and setup described in Section 5.1. The test accuracy for the models trained using the training set nodes is given in Table 2.

1. https://github.com/benedekrozemberczki/datasets#facebook-page-page-networks
2. https://www.kaggle.com/ellipticco/elliptic-data-set

TABLE 3: Attack AUC of unsupervised attack for different thresholds

Dataset	Attack AUC (Thresholds)		
	Loss	Entropy	Loss + Entropy
CORA	0.57	0.58	**0.65**
Citeseer	**0.63**	**0.63**	**0.63**
PubMed	0.57	0.57	**0.63**
FB	0.51	0.51	**0.54**
Elliptic	0.57	**0.58**	0.57

TABLE 4: Attack AUC of shadow model attack

Dataset	Shadow Dataset				
	CORA	Citeseer	PubMed	FB	Elliptic
CORA	-	0.64	**0.71**	0.52	0.66
Citeseer	**0.76**	**-**	0.67	0.7	0.75
PubMed	**0.67**	0.66	-	0.53	0.58
FB	0.57	**0.68**	0.5	-	0.66
Elliptic	**0.6**	**0.6**	0.59	0.57	-

6.3. Unsupervised Inference

This attack assumes that the attacker has knowledge of a randomly selected, balanced and shuffled subset of nodes from both the member and non-member sets of the graph. The attacker also has edge and class knowledge and queries the model to get the prediction logits. An unsupervised attack model is trained on the prediction logits entropy and crossentropy loss, after querying the target model with the node knowledge the attacker has. In table 3, we note the attack AUC for this attack using loss value and normalized entropy thresholding, as well as both together.

Here, we observe that the attack is not very effective for the unsupervised attacks, compared to a random guess for membership. Further, for the Facebook and Elliptic datasets, the attack is ineffective and these models generalize better with GCNs. However, in Sections 7.2 and 7.3, we see that this attack is more effective for certain types of nodes than others. If the adversary has additional knowledge of this information, it may leave these nodes more vulnerable to membership inference attacks.

6.4. Supervised Inference (Shadow Model)

In this attack, the adversary has access to a shadow dataset. The attacker splits the shadow dataset into member and non-member sets and trains a shadow model that thresholds of normalized entropy between the member and non-member nodes.

As seen in Table 4, the CORA dataset when used as a shadow model gives the best attack AUC for most other target models. Interestingly, we see in Section 7.2 that the CORA dataset is also the dataset with the highest edge degree density among the five datasets. Further, the Citeseer model gets the highest attack AUC of 0.76 when the CORA dataset is used to train its shadow model, which is a big improvement over the unsupervised attack in Section 6.3. This type of attack is more effective at membership inference than the unsupervised attack.

TABLE 5: AUC for attack using subgraph knowledge

Dataset	Attack AUC	
	Loss Threshold	Logit Classification
CORA	**0.77**	**0.77**
Citeseer	0.82	**0.85**
PubMed	**0.71**	**0.71**
Facebook	0.67	**0.68**
Elliptic	**0.69**	0.64

6.5. Supervised Inference (Subgraph)

This attacks assumes the highest knowledge for the attacker, which is that of a subgraph of the original graph. This subgraph contains an equal number of member and non-member nodes, with knowledge of their edges and membership information. The attacker queries the target model using the knowledge he has and trains a supervised model for node membership based on loss thresholding and on prediction logits classification.

As seen in Table 5, we see that this attack gives us the highest attack AUC, compared to the previous attacks. We further observe that in most cases, using prediction logits leads to a higher attack accuracy than using a loss threshold.

The Citeseer dataset gives the highest attack AUC of 0.85 using by using the prediction logits. It is also not very effective on the Facebook and Elliptic datasets without additional information about the nodes that are to be attacked. In Sections 7.2 and 7.3, we see that this attack is more effective for certain types of nodes than others. Additional attacker knowledge of this information may leave these nodes more vulnerable to membership inference attacks.

7. Analysis

7.1. Reference Model

We train the reference sequential model on the node features and its labels. As expected intuitively, we get a lower test accuracy for most models, owing to the architectural benefits that GNNs offer over sequential models.

Table 6 also shows that the attack AUC is higher in case of the target model trained using GCNs. This shows that with the current training parameters, though graph neural networks perform better than sequential models in most cases, this also leads to a higher attack accuracy, making them more vulnerable to membership inference attacks. This may show that in these situations, graph neural networks leak more information about the training set, compared to sequential models.

An interesting exception to this is the Elliptic dataset, which generalizes better in the sequential model, and the sequential model is also prone to membership inference attacks than the GNN model.

7.2. Node Degree

The GCN architecture is based on aggregation of neighbouring node features to get an intermediate representation. Considering this, it can be thought that the number of neighbouring nodes for a node may also affect the membership leakage.

TABLE 6: Test and Attack accuracies in Reference and Target models

Dataset	Reference Model (Sequential)		Target Model (Graph)	
	Test Acc	Attack AUC	Test Acc	Attack AUC
CORA	0.62	0.73	0.75	0.77
Citeseer	0.63	0.73	0.74	0.85
PubMed	0.7	0.7	0.7	0.71
FB	0.71	0.61	0.82	0.68
Elliptic	0.83	0.78	0.74	0.64

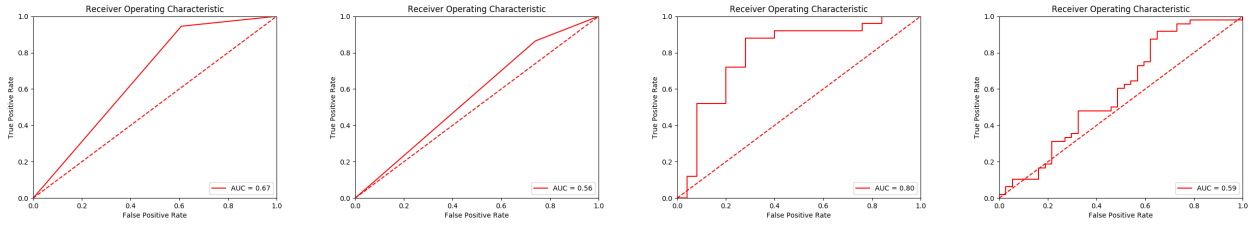
TABLE 7: Attack AUC for unsupervised and supervised attacks for low degree nodes (<4 neighbours) and high degree nodes (>= 4 neighbours)

Dataset	Unsupervised Attack		Supervised Attack	
	$Deg < 4$	$Deg >= 4$	$Deg < 4$	$Deg >= 4$
CORA	0.65	0.55	0.75	0.76
Citeseer	0.67	0.56	0.87	0.81
PubMed	0.57	0.53	0.69	0.68
FB	0.67	0.56	0.8	0.59
Elliptic	0.52	0.53	0.64	0.63

In this experiment, we analyse the effects of node degree on the membership leakage of the model. Intuitively, we see that if a node has a higher degree, its prediction logit is affected by more neighbouring nodes. Due to this, we observe that the attacks that use the prediction logits are more powerful for smaller degree nodes, since their logits are more predictable having fewer edges. This behaviour is also consistent with work done on graph poisoning attacks, which prove that such attacks are more effective on nodes that have a lower degree. On the other hand, the unsupervised attack is more powerful for higher degree nodes. This may be since it might be easier to classify such nodes using just the entropy.

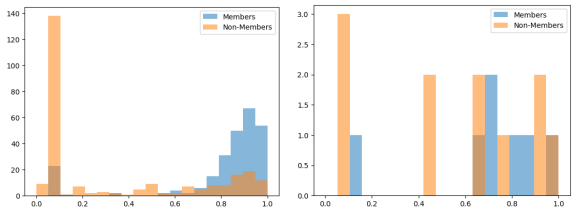
In Table 7, we observe that for both unsupervised and supervised settings, the attacks are more powerful for nodes that have a smaller degree. This is consistent with our intuition. The data shows us that an adversary with degree knowledge of a node is able to infer membership more effectively. For instance, without this knowledge, an attacker is able to attack a model on the Facebook dataset with an accuracy of 0.68. However, with the degree knowledge, a lower degree node can be inferred with a much higher accuracy of 0.8, making it an effective attack (Figure 1). We also plot the histogram for membership inference model prediction for various degrees, where we can observe a difference in distinguishability for a smaller degree and higher degree node (Figure **??**).

Edge Density. Given our observations that lower degree nodes are more prone to node membership inference attacks, we extend our experiment on a dataset level. We make observations for edge density, where datasets with higher edge density are most vulnerable to attacks. The



(a) $deg < 4$ (b) $deg >= 4$ (c) $deg < 4$ (d) $deg >= 4$

Figure 1: ROC curve for FB dataset. Figures 1a and 1b are for unsupervised attack, 1c and 1d are for supervised attack



(a) $deg < 4$ (b) $deg >= 4$

Figure 2: Sample histogram for predictions for low and high degree nodes in the FB dataset for members and non-members

TABLE 8: Edge density and Best Attack AUC

Dataset	Edge Density (E-3)	Best Attack
CORA	2.65	0.77
Citeseer	0.43	0.85
PubMed	0.11	0.71
FB	0.34	0.68
Elliptic	0.05	0.69

density is calculated using the below equation.

$$Edge\ Density = \frac{2|E|}{|N|(|N|-1)}$$

In Table 8, we observe an anomaly that the Elliptic dataset, in spite of having the lower edge density also have a low attack accuracy. This anomaly is also observed for this dataset for the reference model accuracy discussed in 7.1.

7.3. Node Label

Next, we perform an experiment to see if nodes with certain types of labels are more prone to membership leakage. When the attacker has knowledge of the node label and the distribution of node labels in the dataset, he may be able to perform the attack with better accuracy. We observe that across datasets, for labels which do not have a high presence in the dataset, the model memorizes the data more.

In Figure 3, we visualize the distribution of various classes in the dataset, compared with the attack accuracy for the class. We can conclude that with the knowledge of distribution of classes, an attacker can perform the attack more effectively, especially for the nodes having classes with a lesser presence in the dataset.

TABLE 9: Neighbour distributions for False Negative and False Positive nodes

Data	False Negative Node				False Positive Node			
	TP	FN	TN	FP	TP	FN	TN	FP
CORA	0.13	0.03	0.52	0.32	0.2	0.03	0.45	0.32
FB	0.22	0.11	0.5	0.17	0.23	0.02	0.57	0.18

7.4. Attacker Knowledge

In our experiments, we assume that the attacker knowledge in the supervised attack is 60% of the total nodes, including both member and non-member nodes. We analyse if this knowledge affects the performance of the attack. In Figure 4, we observe for the CORA and Citeseer datasets, the attacker knowledge does not affect the attack performance by a large amount. This shows us that even a small amount of dataset knowledge can be used to attack a model effectively.

7.5. Node Neighbours

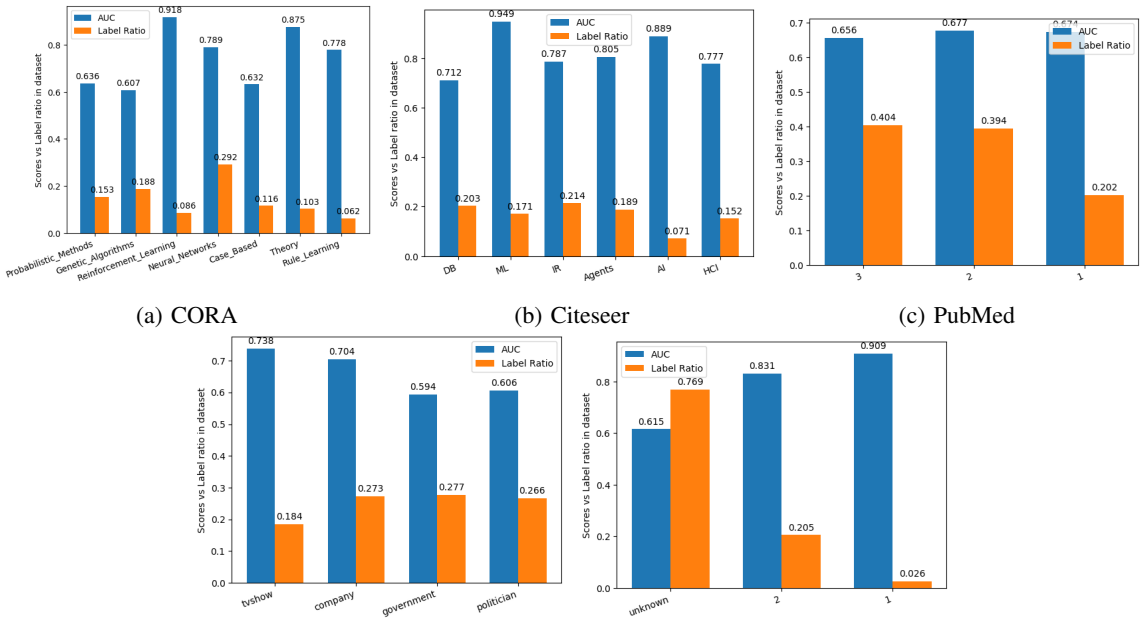
Next, we analyse the properties of the neighbours of nodes which are more easily attacked. That is, we want to analyse how the neighbours of nodes affect the membership leakage of a particular node. We run the test and try to analyse how the neighbours are classified for correctly and incorrectly classified nodes.

Table 9 shows the distributions of neighbouring node predictions for False Negative and False Positive nodes for the CORA and Facebook datasets. We observe that for most incorrectly classified nodes, the neighbours are classified as non-members. This behaviour is consistent for both false positive and true positive classified nodes. Figure 5 shows examples of node neighbours in the CORA and FB datasets for incorrectly classified nodes.

7.6. Classification Consistency

We next perform an experiment to analyse how consistently particular nodes are classified as correct or incorrect, in the 15 iterations the experiment is run. For instance, we see if a node that is classified correctly is always classified incorrectly, or not, and vice versa. This helps us understand if the nodes that are incorrectly classified are random across iterations, or some kinds of nodes are frequently classified incorrectly.

As observed in the histogram in Figure 6, nodes that are identified incorrectly or correct are more frequently identified with the same correctness. A value closer to 1



(a) CORA (b) Citeseer (c) PubMed

(d) FB (e) Elliptic

Figure 3: Class distribution and Attack accuracy for datasets

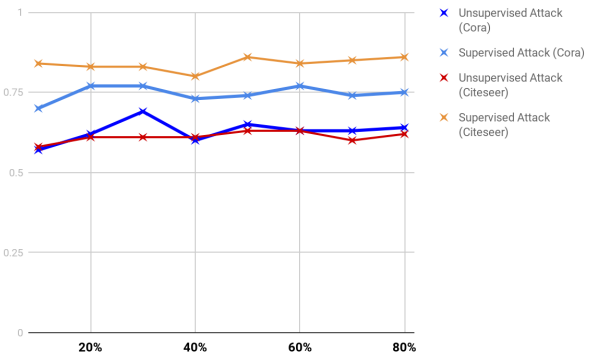
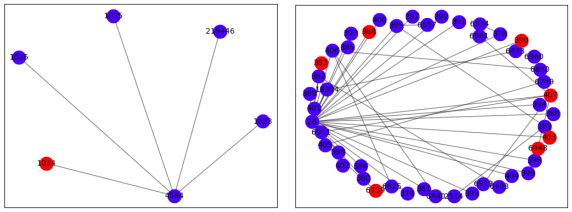


Figure 4: Attack performance vs Attacker knowledge



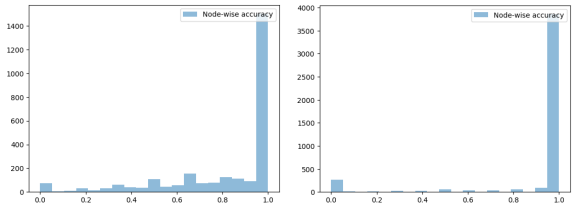
(a) CORA (b) FB

Figure 5: Neighbours of an incorrectly classified node

indicates that a node has been classified correctly for most of the 15 iterations, while a value closer to 0 indicates that the node was classified incorrectly for most of the iterations.

7.7. Edge Perturbations

Next, we try perturbing random edges in the training set to see how this affects the attack accuracy. Specifically, we delete random edges in the graph and check for the



(a) CORA (b) FB

Figure 6: Neighbours of an incorrectly classified node. Blue nodes are correctly classified, while the ones in red are incorrect.

accuracy of the attacker. We do this to analyse how much the presence of edges affects the membership leakage, and consequently the attack accuracy. In this experiment, we remove edges incrementally, in steps of 10% of the total number of edges. We observe the changes to the attack accuracy on this change. Our observations show that greater perturbation in the edges leads to a lower attack accuracy. Figure 7 shows how the membership leakage decreases when the edges are removed from the query data.

7.8. GNN Architecture

In our previous experiments, we used the popular Graph Convolution Network to perform our attacks. We try to see if architectures other than GCNs are equally vulnerable to membership leakage. In Graph Attention Networks (GAT), due to the attention mechanism of the architecture, we may expect better generalization leading to stronger privacy. However, Table 10 shows that this is not the case.

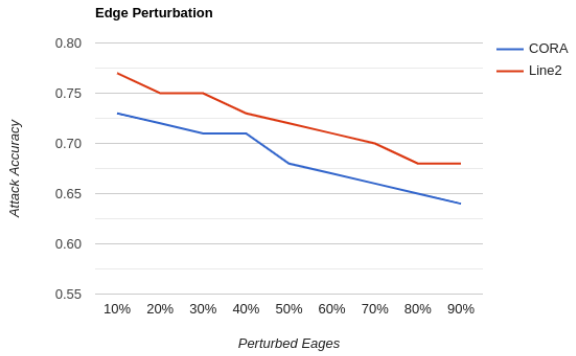


Figure 7: Attack accuracy for different levels of edge perturbation

TABLE 10: Test Accuracy and Attack AUC for GCN and GAT models

Data	GCN		GAT	
	Test Acc	**Attack AUC**	**Test Acc**	**Attack AUC**
CORA	0.75	0.77	0.74	0.78
Citeseer	0.74	0.85	0.73	0.74
PubMed	0.7	0.71	0.65	0.78
FB	0.82	0.68	0.8	0.64
Elliptic	0.74	0.69	0.74	0.75

8. Conclusion

References

[1] Sun, Lichao, et al. “Adversarial attack and defense on graph data: A survey.” arXiv preprint arXiv:1812.10528 (2018).