Friend Recommendation using Graph Neural Networks in Social Platforms

Vasileios Karaiskos
Department of Computer Science & Engineering
University of Ioannina,
Ioannina, Greece
vaskaraiskos@gmail.com

Vasileios Tsolis
Department of Computer Science & Engineering
University of Ioannina,
Ioannina, Greece
vasilistsolis96@gmail.com

***Abstract*— The proliferation of online social platforms has reshaped interpersonal interactions, emphasizing the significance of effective friend-recommendation systems. This study delves into the realm of friend recommendation within social networks, focusing on the utilization of Graph Convolutional Networks (GCNs) to address the challenges posed by large-scale social graphs. Leveraging extensive experiments on datasets from prominent social platforms, namely Facebook and Twitter, we investigated the performance of GCNs compared to a baseline Truncated Singular Value Decomposition (tSVD) model. Our findings underscore the superiority of GCNs over tSVD in early and intermediate friendship recommendation tasks, demonstrating higher accuracy (up to 18% gains) and better ranking relevancy in both the early (up to 109% hit rate, 100% NDCG, and 115% MRR gains) and intermediate (up to 98% hit rate, 83% NDCG, and 86% MRR gains) stages. We observed a correlation between the model accuracy and ranking relevancy across different training epochs, with an optimal threshold for training epochs of approximately 200 epochs. Challenges persist, particularly in addressing the multifaceted nature of social network graphs. Future research could explore more complex model architectures and innovative preprocessing techniques to enhance recommendation accuracy and alleviate data sparsity issues. Overall, our study highlights the potential of GCNs as effective tools for friend recommendations, suggesting avenues for further exploration and improvement in personalized recommendation systems within social platforms.**

Keywords—Graph Convolution Network, Social Recommendation, Social Networks, Python, PyTorch-Geometric

I. INTRODUCTION

In recent years, the advancement of social platforms has transformed the way individuals interact, share information, and form connections. With the proliferation of online social networks, recommending friends or connections has become increasingly important for enhancing the user experience and engagement. Friend recommendation systems powered by sophisticated algorithms play a pivotal role in facilitating meaningful interactions and expanding social circles within these platforms.

This paper delves into the realm of friend recommendation systems, focusing on the utilization of Graph Neural Networks (GNNs)[1] to address the challenges inherent in large-scale social networks. While existing methods, such as graph embedding techniques and traditional recommendation models, have shown promise, they often falter when confronted with the complexities and scale of modern social graphs.

Furthermore, we aimed to explore techniques in machine learning and network analysis to enhance the performance of our recommendation system. The primary objectives of our work are twofold: first, to develop a robust friend recommendation system capable of efficiently handling large-scale social networks. Second, we evaluated the effectiveness of our proposed approach against existing methods, particularly in terms of prediction accuracy and ranking relevancy.

In brief, our methodology involves preprocessing social network data, training a baseline Graph Convolutional Network model (GCN)[2], and implementing a friend ranking module for recommendation generation. We conducted experiments on real-world datasets obtained from two prominent social platforms, Facebook[3] and Twitter[4], to evaluate the performance of our approach.

Our results demonstrate the superiority of our GCN-based model over traditional approaches, showcasing higher accuracy in predicting (12% to 18% gains) new user connections and providing more relevant friend recommendations in both the early stage (top 5, 100% to 109% hit rate gains) and the intermediate stage (top 50, 73% to 98% hit rate gains). Additionally, our study sheds light on the intricate dynamics of social network graphs and highlights avenues for future research in this domain.

Therefore, the main contributions of our work lie in the development of an effective friend recommendation system powered by a GCN and a comprehensive evaluation of its performance against existing methods. Our findings underscore the potential of GCN in addressing the challenges of large-scale social networks, paving the way for more sophisticated and personalized recommendation systems in the future.

The structure of this paper is presented as follows: Section II provides an overview of existing research efforts in the field of friend recommendation systems, with a focus on the utilization of GCN. Section III outlines the methodology framework and provides details on data pre-processing techniques, background on the GNN model, specifics of the GCN model architecture, friend ranking module, and the friend recommendation algorithm. Section IV presents the experiments conducted to evaluate the performance against a baseline Truncated Singular Value Decomposition (tSVD)[5] model. Section V analyzes the effectiveness of GNNs and the limitations with potential areas for future research and improvement.

II. RELATED WORK

The development of friend-recommendation systems utilizing GNNs has garnered significant attention in recent research efforts aimed at enhancing the user experience within social platforms. Here, we review pertinent studies in this domain.

In recent years, graph-embedding methods have gained prominence for friend recommendation tasks. These methods learn unsupervised embedding techniques to generate embeddings that capture the structural features of nodes within a graph[6]. Widely used embedding models such as node2vec[7] achieve this by determining the likelihood of a node in a random walk. Consequently, these methods demonstrate effective performance in predicting the links between nodes in a graph.

However, these methods are less practical for large-scale networks[1], [7]. As the size of the network increased, the computational resources required for training and inference also increased significantly. Therefore, although graph embedding methods offer promising solutions for friend recommendation tasks, their constraints are challenges in large-scale social networks.

GNNs propagate information from the local neighborhoods of nodes throughout the graph. A notable architecture within GNNs is the Graph Convolutional Networks (GCNs)[8], which learn by node degrees, utilizing the graph Laplacian matrix. Many models have expanded upon GCNs by introducing various learnable aggregators, such as self-attention mechanisms, mean pooling, and max pooling functions. These advancements have consistently outperformed the embedding techniques based on random walks.

In another study, researchers treated friend suggestions as a friend-ranking problem and conducted experiments on large datasets to investigate its effectiveness [3]. They proposed a neural architecture capable of learning expressive user representations from multi-modal features and user-user interactions. The methods were compared against strong feature-based ranking models and ranking metrics for evaluation. In this study, we modeled a baseline GCN to learn friend rankings and compared it with an additional embedding algorithm.

III. METHODOLOGY

This section provides a comprehensive overview of the methodology used in this study. Initially, data procurement is discussed, along with a brief statistical analysis of the available datasets, followed by the Data Pre-Processing subsection, which encompasses the preparatory steps undertaken to ensure dataset integrity. Next, the Graph Neural Networks Background section is presented, offering a comprehensive exposition of foundational concepts. Following this, the Graph Convolutional Network Model is detailed, highlighting its architectural framework and implementation specifics. Subsequently, the Friend Ranking Module is discussed, elucidating its role within the methodology. Finally, the Friend Recommendation subsection is presented, underscoring its pivotal function within the proposed approach.

A. *Large-scale social network dataset procurement*

Large-scale social network datasets have a vital role in this study, to accurately sample real-life, large-scale social networks, and millions of users with billions of user-to-user interactions that occur every day. Two distinct datasets were selected to accurately represent two of the largest social media networks available worldwide, namely ego-Facebook and ego-Twitter.

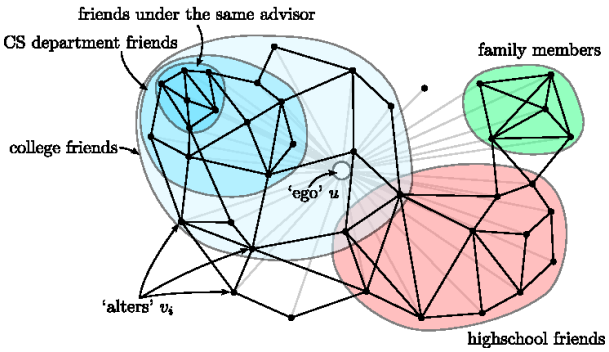

Figure 1 This ego-network contains users, features, and social circles from Facebook. The task aims to adjust the features and user embeddings and calculate the ranking of friends[9].

Attributes	Data Set	
	Facebook	***Twitter***
Nodes	4.039	81.306
Edges	88.234	1.768.149
Node Features	1.406	216.839
Average clustering coefficient	0,6055	0,5653
Diameter	8	7
90-percentile effective	4,7	4,5

TABLE I. DATASET

The datasets were obtained through McAuley and Leskovec's prior work on social circles in ego networks, currently hosted on the Stanford SNAP dataset library[9] as presented in Fig. 1.

It includes user-to-user connections/interactions, denoted as edges (nodeId.edges), communities (nodeId.circles), features for each of the users represented as nodes (nodeId.feat), features for the ego user (nodeId.egofeat), and the names of each feature dimension (nodeId.featnames) from Facebook and Twitter. The nodes, features (profiles), circles, and ego networks have been anonymized to ensure user privacy. Dataset anonymization was achieved by replacing the original IDs and obscuring feature interpretations. Table [I] provides a brief description of the datasets.

B. *Data Pre-Processing*

The data processing approach for the ego-Facebook network involved several key steps to enhance the dataset for subsequent analysis. The ego features were unified with the user features to create a cohesive representation of the individual user characteristics. The feat names were then mapped to their respective feature names for each column, establishing a clear association between feature identifiers and their names. As part of this process, all networks were consolidated to create a final integrated file. This file incorporates all features that are precisely aligned with their respective names.

The refined dataset serves as a valuable resource for model training and facilitates an in-depth analysis in our scientific report. The processed data file is ready to contribute significantly to the exploration and creation of the Data Graph within the network.

A similar procedure was initially applied to the large-scale ego-Twitter network. Hardware limitations lead to an alternative approach to extracting features from the Twitter dataset, and a rising opportunity for a novel categorical feature extraction process. Each ego feature and user feature file was processed in sequence, unifying in a dictionary list of variable size, consisting of the word representation of the feature labels available for each user. Feature words were encoded in succession in integer representation embeddings, while edge pairings were mapped so that each node ID belonged to the numeric space beginning from zero up to the number of the total nodes in the dataset. As a final point, the feature list was padded with zeros to create a uniform-sized matrix representation of the features, with sizes up to the maximum possible number of features a user had. The process then converged to the same point as the ego-Facebook process and continued as described in the following sections.

Subsequently, a custom dataset class[10], [11] derived from the PyTorch Geometric data processing library was defined to encapsulate the data and store them in the memory of the appropriate processing device (CPU or GPU if CUDA is available and enabled). This class object represents a graph compatible with PyTorch Geometric[12], suitable for training, validation, and experimentation with GNNs.

C. Graph Neural Networks Background

GNNs[13] are a category of neural networks specifically designed for processing and learning from graph-structured data. In contrast to conventional neural networks that primarily operate on grid-structured or sequential data, GNNs are tailored to handle data represented as graphs composed of nodes (vertices) interconnected by edges (links). They have garnered significant attention due to their efficacy in modeling intricate relationships and dependencies inherent in data with irregular structures such as social networks, biological networks, recommendation systems, and knowledge graphs. GNNs facilitate various tasks including node classification, link prediction, graph classification, and graph generation.

At their core, GNNs employ an iterative approach to update node representations by aggregating information h from neighboring nodes within the graph as shown in equation (1), (2). This iterative process enables the GNNs to capture the local graph structure and propagate information m across the entire graph. A multitude of architectural designs and techniques have been proposed to realize this concept, including Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), GraphSAGE, and Graph Isomorphism Networks (GINs), among others[13].

Our approach to the friendship recommendation model is split into a GCN model to learn node embeddings and a friend ranking model to utilize node embeddings knowledge for friendship suggestions based on higher relevancy ranking scores. Finally, the effectiveness of the GCN model was evaluated against a baseline TruncatedSVD model for learning node embeddings.

D. Graph Convolutional Network Model

To model the learning process for node embeddings, a graphical model of a neighborhood neural network was developed, focusing on the analysis of social networks through the Facebook platform as derived from the Facebook dataset. The model is based on a Graph Convolutional Network (GCN)[2], [14] and is designed to learn information about the users and calculate the features for each node based on the neighbor nodes as shown in equation (3). The fundamental concept is to perform convolution on the k-neighbor node features while maintaining the structural connections between nodes. First, the model receives data from the Facebook dataset, which includes relationship connections between users and user features. The input is represented as a graph, with users as nodes and connections as edges.

Incorporated within the model are two hidden layers employing graphical convolutions with ReLU activations and a dropout layer in between. This allows the extraction of important features from the graph and user features. Additionally, a Stochastic Gradient Descent optimizer is utilized as well as a StepLR scheduler to control the learning rate decay. At the final output level, the loss is minimized from the calculations of the positive and negative scores against the accuracy of the predictions of the relationships between users[13]. Let X be the feature matrix of the graph nodes, and A be the symmetric adjacency matrix of the graph. The pre-processed adjacency matrix A is computed as:

$$A = D^{\frac{1}{2}} A D^{-\frac{1}{2}} \tag{1}$$

Where D represents the diagonal matrix of node degrees, the forward model of the GCN is expressed as depicted in formula (2):

$$Z = f(X, A) = softmax(A\ ReLU(AXW^{(0)})W^{(1)} \tag{2}$$

Where ReLU denotes the Rectified Linear Unit activation function, $W^{(0)}\ and\ W^{(1)}$ are the weight matrices and the softmax activation function is applied row-wise to obtain class probabilities as:

$$softmax(x_i) = \frac{1}{z}\exp(x_i)\ with\ z = \sum_i \exp\ (x_i) \tag{3}$$

Our GCN model takes the feature matrix X and the adjacency matrix A as inputs. The pre-processing step computes the normalized adjacency matrix A. Then, the forward model applies two graph convolutional layers followed by a softmax activation to obtain class probabilities.

We used the dropout layer to prevent overfitting during training of the neural network model. Overfitting occurs when a model learns to memorize the training data rather than generalize well to unseen data. Dropout addresses this issue by randomly dropping a fraction of the input units during training, forcing the network to learn more features. During training, the weights are learned using gradient descent. The loss function used is the cross-entropy loss, computed over the labeled examples as present in equation (4):

$$L = -\sum_{l \in y_L} \sum_{f=1}^{F} Y_{l\,f}\ ln\ Z_{l\,f} \tag{4}$$

where Y_L denotes the set of node indices that are labeled.

The Stochastic Gradient Descent (SGD) optimizer was used because it iteratively updates the model parameters in the direction opposite to the gradient of the loss function, aiming to minimize the loss and improve the model performance. The momentum term, set to 0.9, accelerates the convergence and reduces oscillations by adding momentum to the parameter updates.

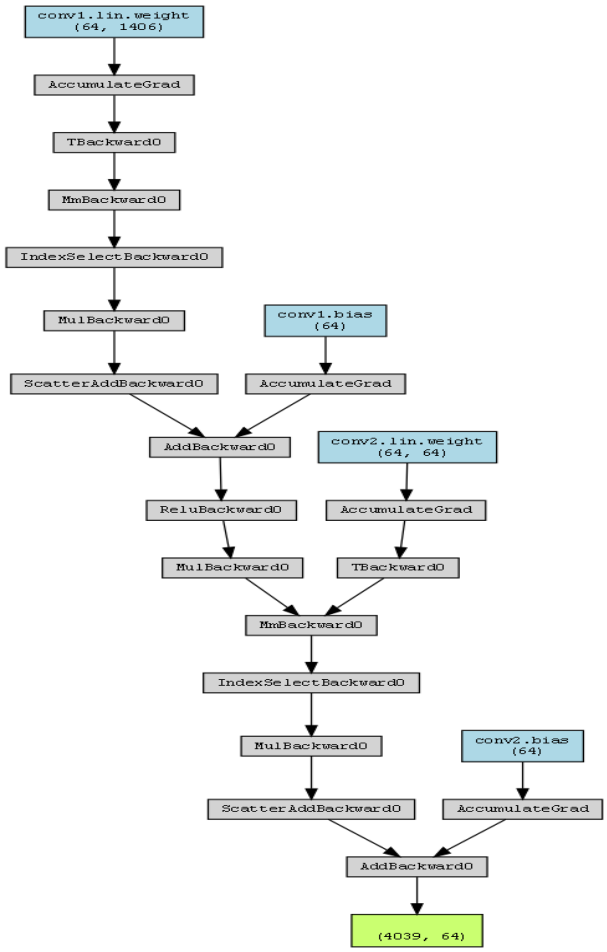

Figure 2. Model GCN

In addition, the StepLR scheduler was used to gradually reduce the learning rate over time during training. By reducing the learning rate by a factor of 0.5 every 25 epochs, the scheduler facilitates a more stable and efficient convergence of the optimizer. Consequently, the scheduler is used to manage the learning rate throughout the training, improve the optimization process, and enable more effective parameter updates.

Furthermore, the loss function quantifies the disagreement between the predicted logits and true labels by computing the binary cross-entropy, thereby facilitating the adjustment of the model parameters to minimize this discrepancy. By penalizing the deviations between the predictions and ground truth, the binary cross-entropy loss guides the model towards making more accurate classifications.

Finally, we used a training and evaluation pipeline for this learning model, which initialized the chosen model, along with a predictor, optimizer, and learning rate scheduler. It then conducted training over a specified number of epochs (30,50,100,200,1000 epochs), computing the loss, performing propagation, and updating the model parameters. After training, the function evaluates the performance of the model on a test set, calculates the Area Under the Curve (AUC) score, and prints the loss. Fig. 2 illustrates the architecture of the proposed GCN model.

An additional baseline model used as a reference was designed based on the TruncatedSVD[5] model. Truncated SVD is based on dimensionality reduction while preserving the most important information in the data. Truncated SVD is applied in collaborative filtering-based recommendation systems to efficiently handle sparse user-item interaction matrices. It helps in identifying latent factors or features that represent user preferences and item characteristics, enabling more accurate recommendations, thus making it a suitable comparison model for the GCN implementation. It features the same SGD optimizer and StepLR scheduler as those of the GCN model.

E. Friend Ranking Module

After training the model to obtain embedded representations of the nodes, the next objective is a prediction model for estimating the "likelihood" of two nodes connecting, by calculating the dot product between the two nodes embeddings.

The Friend Ranking Module[14], [15] utilizes edge index matrices as input and the node features h, as learned from the GCN/TruncatedSVD model. For each edge pairing, the module maps the features to the source and destination nodes.

Subsequently, the dot product between the two embeddings pairings is calculated. The dot product provides a score that indicates the "agreement" between these features. In the context of graph representation, this is interpreted as an indication of the probability of a connection between two nodes, denoted by an edge between the two nodes in the graph.

F. Friend Recommendation

In the ranking process[15], the generator and calculator functions determine the order of recommended friends for a given user within a social network. After computing the compatibility scores for potential friend candidates using a dot product predictor, both functions proceeded to the sorting phase.

The generator presented serves the purpose of facilitating personalized friend recommendations. The generator's functionality unfolds in several key steps. Initially, it identifies the friends associated with a specific user, denoted by the user ID. Subsequently, it crafts "negative edges" between the target user and individuals who are not within their social circle. Following this, the generator computes the scores for potential friend candidates using a dot product predictor, thereby assessing their compatibility with the target user. These scores were then sorted, and the top-ranked friends were presented in a list, with the number of recommendations limited by the specified k parameter.

The calculator function serves the purpose of computing recommendations for a given user. Operating in several distinct phases, the calculator initially identifies friends associated with the target user by utilizing the edge index data structure. Subsequently, it generates "negative edges" between the user and non-friendly individuals, ensuring comprehensive coverage of potential connections. Following this, the calculator employs a dot product predictor to assess the compatibility of each potential friend candidate with the user, generating scores that reflect their suitability. These scores are then sorted to prioritize the candidates with the highest compatibility scores. In addition, the calculator evaluates the relevance of each recommendation by checking against a set of positive test edges. Ultimately, the calculator

produced a ranked list of recommendations, with the number of recommendations limited by the specified parameter (500).

IV. EXPERIMENTS

In this section, we describe the experiments conducted in this study. We report on our experimental setup, describing in detail the datasets we used as input and how we preprocessed them, and the evaluation metrics we used to assess the effectiveness of the model in recommending new user-to-user connections along with the model's training process. Following this, we present the cross-evaluation GNN model we utilized to test against our GCN model implementation, Truncated Singular Value Decomposition. Finally, we showcase the results of our comprehensive experiments, state our observations, and offer insights.

A. Experimental Setup

1) Dataset

We evaluated the proposed GCN model utilizing two large-scale datasets, as authored by [9] and stored in Stanford's SNAP repository. The datasets contained information about ego networks on Facebook and Twitter, and each dataset was subjected to different preprocessing steps before use.

The ego-Facebook dataset contained information on 10 anonymized ego networks, with 4.039 nodes, 88.234 edges, and 1.406 features describing each node, as presented in Table [I]. The dataset was restructured into a file representing the node features matrix by combining the data contained in the ".egofeat", ".feat" and ".featnames" files into a space-separated file of 1.406 columns for each feature and 4.039 lines for each node, comprised of zeros and ones, along with the provided "facebook_combined.txt" containing information about graph connectivity.

In sequence, the dataset was loaded into the graph representing PyTorch Geometric[16] helper class "Data", which was used to procure the train and test True Negatives through the build-in negative_sampling function, along with a subgraph containing 70% of the original edges, to procure the training True Positives, while the remaining 30% was utilized to provide the test True Positives. This selective, highly curated separation process is necessary to contribute to a focused training environment that is vital for ensuring reliable performance on the ranking metrics utilized in this study.

An alternative approach was utilized for the ego-Twitter dataset preprocessing, due to the large-scale nature of the data, as well as existing hardware limitations. In contrast to the ego-Facebook dataset, the ego-Twitter dataset was encoded in categorical values, which were then converted to word embeddings to be loaded through the "Data" class. The remaining sampling procedure was performed as previously described.

2) Evaluation Metrics

This section describes the comprehensive evaluation process of our recommendation system's performance using prediction and ranking metrics.

These metrics serve a crucial role in measuring the accuracy of the model's capability to identify potential new user connections/friendships, along with the effectiveness of our model in predicting the top relevant neighbor users that are likely to attract stakeholder users' interests. Additionally, the metrics provide valuable insights into the system's optimization potential, facilitating ongoing improvements for enhanced user satisfaction.

Specifically, the metrics utilized for evaluating the recommendation model are distinguished into two types:

a) Accuracy-Based Metric: The evaluation of the model's performance was conducted on the test set by employing the Area Under the Curve (AUC) metric. AUC serves as a statistical measure symbolizing the area beneath the receiver operating characteristic (ROC) curve. Elevated AUC values signify improved discrimination between classes.

b) Ranking-Based Metrics: Friend recommendation tasks are sensitive to effective link predictions in both quality and ranking positions in the top K predictions. Specialized metrics are necessary for this quantitative evaluation, since standard accuracy metrics do not consider the position in the top K predictions. To rectify this shortfall, the following metrics were used:

- Hits@K: Measures the proportion of relevant items among the top k recommendations, providing insight into the system's ability to retrieve relevant items within a given recommendation list. It is calculated through the following formula (5), with Q representing a collection of test triples that have been positioned within the top q triples[17].

$$Hits@K = \frac{|\{q \in Q : q < k\}|}{|Q|} \in [0,1] \qquad (5)$$

- NDCG@K (Normalized Discounted Cumulative Gain): Evaluates the ranking quality of the recommendations at position K, by considering both relevance and position, offering a more nuanced assessment of the ranking quality. NDCG@K is computed as shown in equation (6), (7), where Q is the total number of queries $rel_i \in [0,1]$ is the relevancy score for the prediction, and j is the rank of the prediction[18].

$$NDCG = \frac{1}{Q}\sum_{j=1}^{Q} NDCG@k_j \qquad (6)$$

$$NDCG@k = \sum_{i=1}^{k} \frac{DCG@k}{IDCG@k} = \frac{\sum_{i=1}^{k\,(actual\ order)} \frac{rel_i}{\log_2(i+1)}}{\sum_{i=1}^{k\,(ideal\ order)} \frac{rel_i}{\log_2(i+1)}} \qquad (7)$$

- MRR (Mean Reciprocal Rank): Assesses the effectiveness of the recommendation by considering the rank of the first correct recommendation, as determined by the ensuing equation (8), where q refers to the rank position of the *first* relevant prediction, and Q denotes the total number of queries[19], [20].

$$MRR = \frac{1}{Q}\sum_{q=1}^{Q} 1/q \in [0,1] \qquad (8)$$

In all cases, larger values indicate better performance.

3) Training Process

Our model was trained using a configuration comprising layers of message-passing graph convolutional operations[21], each possessing a hidden dimension size of 64 and an output embedding dimension of 64. Within each layer, the aggregation of information from neighboring nodes occurs

through a weighted sum calculation of their respective features. Subsequently, dimensionality reduction is executed on the aggregated data via a linear transformation neural network layer augmented with the ReLU activation function. A dropout, with a default rate of 0.5, was applied between layers during model training.

The training process involved conducting a series of experiments across epochs, encompassing 30, 50, 100, 200, and 1000 maximum epochs. A Stochastic Gradient Descent optimizer, coupled with a learning rate of 0.01 and momentum of 0.9, was employed for this purpose. Additionally, a learning rate decay of 0.5 is applied every 25 epochs using the StepLR scheduler.

The experimental evaluation was conducted on two distinct machines: one equipped with an AMD Ryzen 7 2700 Eight-Core Processor, featuring two threads per core and 8 GB of shared CPU memory, operating on the Ubuntu Linux 20.04.6 LTS platform, and the other using an Intel Core i7 6700K 4 core processor featuring two threads per core and 16 GB of shared CPU memory, operating on the Windows 10 platform.

Notably, our PyTorch Geometric implementation is publicly available through the GitHub repository.

B. Cross-Evaluation Baseline

Aspiring to further evaluate the effectiveness of the GCN model, we performed a cross-evaluation comparison with the Truncated Singular Value Decomposition (tSVD) model for dimensionality reduction.

Truncated Singular Value Decomposition (SVD) is a mathematical technique used to reduce the dimensionality of a matrix while preserving its essential structure. It involves decomposing a matrix into three constituent matrices - the left singular vectors, the singular values, and the right singular vectors. The "truncated" aspect of this method involves retaining only the top k singular vectors and corresponding singular values, effectively reducing the dimensionality of the original matrix.

Truncated SVD finds applications in various fields, including data compression, noise reduction, feature extraction, and latent semantic analysis. It is widely utilized in machine learning tasks in recommendation systems where high-dimensional data matrices are common, and reducing their dimensionality can improve the computational efficiency and facilitate interpretation. By retaining the most significant information encoded in the original matrix, truncated SVD enables a more efficient and effective analysis of large datasets while mitigating the effects of noise and redundancy.

In our implementation, tSVD performed dimensionality reduction from the input dimension of 1406 to the output dimension of 64, accompanied by a linear transformation. The training process remained the same for the tSVD and GCN models to provide consistent and comparable results.

C. Experiment Results

Tables [II], [III], [IV], [V] and [VI] list the performance metrics for the two models, GCN and tSVD, across the various evaluation criteria, as stated in the training epochs and accuracy/ranking evaluation metrics.

30 training Epochs					
Model	***AUC***	***Hits***	***NDCG***	***MRR***	***Ranking***
GCN	0.8877	**0.0090**	0.0260	0.0209	Top 5
		0.0064	**0.0640**	**0.0305**	Top 50
tSVD	0.7545	0.0046	0.0143	0.0117	Top 5
		0.0037	0.0407	0.0180	Top 50

TABLE II. 30 TRAINING EPOCHS

50 training Epochs					
Model	***AUC***	***Hits***	***NDCG***	***MRR***	***Ranking***
GCN	0.9183	**0.0094**	0.0278	0.0221	Top 5
		0.0082	**0.0742**	**0.0336**	Top 50
tSVD	0.7596	0.0044	0.0132	0.0105	Top 5
		0.0038	0.0394	0.0168	Top 50

TABLE III. 50 TRAINING EPOCHS

100 training Epochs					
Model	***AUC***	***Hits***	***NDCG***	***MRR***	***Ranking***
GCN	**0.9316**	**0.009**	0.0252	0.0192	Top 5
		0.0086	**0.0761**	**0.0318**	Top 50
tSVD	0.7861	0.0047	0.0143	0.0114	Top 5
		0.0042	0.0434	0.0184	Top 50

TABLE IV. 100 TRAINING EPOCHS

200 training Epochs					
Model	***AUC***	***Hits***	***NDCG***	***MRR***	***Ranking***
GCN	0.9417	**0.011**	0.0319	0.0253	Top 5
		0.0094	**0.0861**	**0.0395**	Top 50
tSVD	0.8224	0.0055	0.0166	0.0134	Top 5
		0.0046	0.0475	0.0212	Top 50

TABLE V. 200 TRAING EPOCHS

1000 training Epochs					
Model	***AUC***	***Hits***	***NDCG***	***MRR***	***Ranking***
GCN	**0.9596**	**0.0096**	0.0273	0.0212	Top 5
		0.0095	**0.0868**	**0.036**	Top 50
tSVD	0.8556	0.0043	0.0126	0.0097	Top 5
		0.0048	0.0448	0.017	Top 50

TABLE VI. 1000 TRAINING EPOCHS

The observations and insights based on the provided results can be summarized as follows:

a) Model Accuracy: GCN achieved a notably higher AUC score (88.77% to 95.96%) compared to tSVD (75.45% to 85.56%). This suggests that GCN demonstrates superior discriminatory power in distinguishing between positive and negative instances.

b) Ranking Relevancy, top 5 results: GCN outperforms tSVD across all three metrics Hits (GCN 0.009-0.011 versus tSVD 0.0043-0.0055), NDCG (GCN 0.0252-0.0273 versus tSVD 0.0126-0.0166) and MRR (GCN 0.0209-0.0253 versus tSVD 0.0097-0.0134). GCN's higher values indicate better performance in accurately recommending relevant items within the top 5 recommendations.

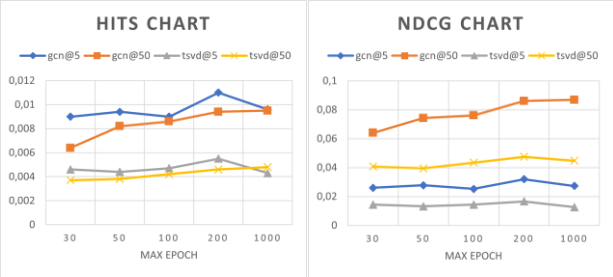

Figure 3. HITS and NDCG results among GCN and tSVD.

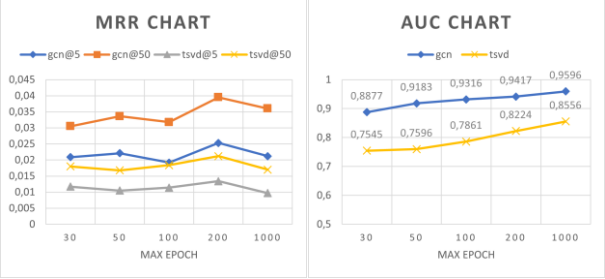

Figure 4. MRR and AUC results among GCN and tSVD.

c) Ranking Relevancy, top 50 results: Similar to the Top-5 metrics, GCN also outperforms tSVD across all three metrics Hits (GCN 0.0064-0.0095 versus tSVD 0.0037-0.0048), NDCG (GCN 0.064-0.0868 versus tSVD 0.0407-0.0475) and MRR (GCN 0.0305-0.0395 versus tSVD 0.017-0.0212). This implies that the GCN maintains its superiority in recommending relevant items across a larger set of recommendations (Top-50).

d) Max epoch training strategies: Across both models, longer training sessions improve the model prediction performance, as seen in AUC results, with ranking relevancy peaking at 200 epochs, with further training impeding the ranking relevancy performance, as viewed in the Fig.3 and Fig. 4.

e) Alternative approach, Twitter Dataset: Experimental results on the alternative preprocessed dataset ego-Twitter proved inconsistent, with AUC results on the GCN model of 50% accuracy, whereas the tSVD model failed to converge to a solution. Hence, the application of ranking metrics is unwarranted in this particular scenario because of the model's inability to predict new user–user connections.

In summary, based on the experimental results, the GCN proved to be a more effective model for friend recommendation in the ego-Facebook social network graph compared to tSVD. The higher AUC score and consistently better performance across various top-K recommendation metrics indicate GCN's superior predictive and ranking capabilities. Extensive training sessions increased the model accuracy, with the exception of ranking relevancy peaking at 200 epochs and further decreasing over time.

V. DISCUSSIONS

The experimental results demonstrated the superiority of the GCN model over tSVD in every category. The GCN achieved 12% to 18% gains in accuracy, hit rate gains of 100% to 109% in early recommendations, and 73% to 98% gains in intermediate recommendations. Early recommendations achieved gains in NDCG of 64% to 100% and MRR of 89% to 115%, while intermediate recommendations achieved gains in NDCG of 57% to 83% and MRR of 79% to 86%. Although both models are baseline implementations of the appropriate literature list, we ascertain that GCN models are more advantageous for friend recommendation tasks than tSVD implementations.

Notably, on the top 5 versus the top 50 recommendations, the model provided a better hit rate at the top 5 recommendations in all training epoch scenarios, while it produced better NDCG and MRR values on the top 50 recommendations. This attests to the model's high accuracy on early recommendations; in contrast with the high quality of ranking relevance at the full scale of the top 50 recommendations.

It is noteworthy how the AUC accuracy and the ranking relevance are correlated on the spectrum of training epochs. We observe that up to a certain threshold of epochs, ranking relevancy improves with improvements to the model's accuracy. However, further increases in epochs training improve the model's accuracy, while providing a decrease in hit rate and quality of friendship suggestions. We assume that model overfitting is the leading cause of this, and we observed that the threshold for the suggested training epochs in GCN models for friendship recommendations is closer to 200 or more epochs than 1000 or fewer epochs.

The GCN model in general achieved a run-of-the-mill performance from the initial assumptions. The model in comparison with previous literature, notably GraFRank, performed far below the standard within expectations. This is attributed to the multifaceted complex nature of the friendship recommendation ranking systems. These systems must grapple with the intricate structure of social networks, which evolve dynamically with the formation, evolution, and dissolution of connections between users. Accommodating the varied preferences and needs of users in forming friendships is crucial, requiring algorithms capable of efficiently scaling to manage substantial data volumes while delivering precise and pertinent recommendations. Furthermore, the cold start problem poses a significant hurdle, particularly for new users or those with limited activity history, requiring innovative approaches to mitigate data sparsity issues. The evaluation of recommendation algorithms must consider not only their accuracy but also their ability to foster user engagement and satisfaction.

Ultimately, we address the limitations of this study, which we aim to tackle in subsequent research efforts. The hit rate was found to be underperforming, even if it achieved double the hit rate of the cross-evaluation baseline model. Further analysis of the multifaceted nature of the social network graphs and additional convolution layers or model complexity could result in higher ranking accuracy and quality of the recommendations. Additional experimentation can provide a fine-grained approximation of the ideal training epoch threshold, leading to an elevated hit rate.

Large-scale datasets like the ego-Twitter preprocessing task and the training task emerged as problematic, with the alternate word embeddings preprocessing methodology failing to achieve predictive accuracy status above random guessing. While increased computational capacity would solve a part of the problem, the large-scale nature of the social networks demands lightweight and efficient algorithms to procure, prepare, and process data for Machine Learning and

Artificial Neural Networks. Word embedding encoded features should assist in minimizing large Laplacian matrices, albeit higher complexity on both the model architecture and the dataset preprocessing would be required to be introduced to. We anticipate investigating this.

VI. Conclusions

In this study, we explored the efficacy of Graph Convolutional Networks (GCNs) in the domain of friend recommendation within social networks. Leveraging extensive experiments on large-scale datasets from ego-Facebook and ego-Twitter networks, we investigated the performance of GCNs compared to a baseline Truncated Singular Value Decomposition (tSVD) model.

Our findings demonstrate the superiority of GCNs over tSVD in various aspects of friend recommendation tasks. GCNs consistently outperformed tSVD in accuracy-based metrics (12-18% AUC gains), while exhibited better-ranking relevancy in both early stage (100-109% Hits@K gains, 64-100% NDCG@K gains, 89-115% MRR gains) and intermediate stage (73-98% Hits@K gains, 57-83% NDCG@K gains, 79-86% MRR gains), highlighting their effectiveness in accurately recommending relevant items within the top-5 and top-50 recommendations.

We observed a correlation between the model's accuracy and ranking relevancy across different training epochs, with an optimal threshold for training epochs around 200. Beyond this threshold, further increases in training epochs led to diminishing returns and a decline in ranking relevancy.

Despite the promising performance of GCNs, challenges remain, particularly in addressing the multifaceted nature of social network graphs. Moreover, future research could explore more complex model architectures and innovative preprocessing techniques to enhance recommendation accuracy and alleviate data sparsity issues. This is particularly relevant as our study underscores the potential of GCNs as effective tools for friend recommendation in social networks, suggesting avenues for further exploration and improvement. By leveraging graph-based learning techniques, we can provide users with personalized and relevant friend recommendations, thereby enhancing user experience and engagement within social platforms.

References

[1] A. Sankar, Y. Liu, J. Yu, and N. Shah, "Graph neural networks for friend ranking in large-scale social platforms," in The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021, Association for Computing Machinery, Inc, Apr. 2021, pp. 2535–2546. doi: 10.1145/3442381.3450120.

[2] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," Sep. 2016, [Online]. Available: http://arxiv.org/abs/1609.02907

[3] "SNAP: Network datasets: Social circles." Accessed: Feb. 13, 2024. [Online]. Available: http://snap.stanford.edu/data/ego-Facebook.html

[4] "SNAP: Network datasets: Social circles." Accessed: Feb. 13, 2024. [Online]. Available: http://snap.stanford.edu/data/ego-Twitter.html

[5] "sklearn.decomposition.TruncatedSVD — scikit-learn 1.4.0 documentation." Accessed: Feb. 13, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

[6] P. Cui, X. Wang, J. Pei, and W. Zhu, "A Survey on Network Embedding," Nov. 2017, [Online]. Available: http://arxiv.org/abs/1711.08752

[7] J. Chen et al., "N2VSCDNNR: A Local Recommender System Based on Node2vec and Rich Information Network," Apr. 2019, [Online]. Available: http://arxiv.org/abs/1904.12605

[8] L. Chen, Y. Xie, Z. Zheng, H. Zheng, and J. Xie, "Friend Recommendation Based on Multi-Social Graph Convolutional Network," IEEE Access, vol. 8, pp. 43618–43629, 2020, doi: 10.1109/ACCESS.2020.2977407.

[9] J. M. Stanford, "Learning to Discover Social Circles in Ego Networks." [Online]. Available: http://snap.stanford.edu/data/

[10] "torch_geometric.data.Data — pytorch_geometric documentation." Accessed: Feb. 13, 2024. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.data.Data.html#torch_geometric.data.Data

[11] "Creating Graph Datasets — pytorch_geometric documentation." Accessed: Feb. 13, 2024. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_dataset.html

[12] "Introduction by Example — pytorch_geometric documentation." Accessed: Feb. 13, 2024. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/get_started/introduction.html#data-handling-of-graphs

[13] "Fraudulent Activity Recognition in Graph Networks | by Aman Kansal | Stanford CS224W GraphML Tutorials | Medium." Accessed: Feb. 13, 2024. [Online]. Available: https://medium.com/stanford-cs224w/fraudulent-activity-recognition-in-graph-networks-f60dc724feea

[14] "Graph Neural Networks with PyG on Node Classification, Link Prediction, and Anomaly Detection | by Tomonori Masui | Towards Data Science." Accessed: Feb. 13, 2024. [Online]. Available: https://towardsdatascience.com/graph-neural-networks-with-pyg-on-node-classification-link-prediction-and-anomaly-detection-14aa38fe1275

[15] "Friend Recommendation using GraphSAGE | by Yan Wang | Stanford CS224W GraphML Tutorials | Medium." Accessed: Feb. 13, 2024. [Online]. Available: https://medium.com/stanford-cs224w/friend-recommendation-using-graphsage-ffcda2aaf8d6

[16] "PyG Documentation — pytorch_geometric documentation." Accessed: Feb. 13, 2024. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/

[17] M. Ali et al., "Bringing Light Into the Dark: A Large-scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework," Jun. 2020, doi: 10.1109/TPAMI.2021.3124805.

[18] "Demystifying NDCG. How to best use this important metric… | by Aparna Dhinakaran | Towards Data Science." Accessed: Feb. 13, 2024. [Online]. Available: https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0

[19] "Mean Reciprocal Rank (MRR) explained." Accessed: Feb. 13, 2024. [Online]. Available: https://www.evidentlyai.com/ranking-metrics/mean-reciprocal-rank-mrr

[20] "Compute Mean Reciprocal Rank (MRR) using Pandas." Accessed: Feb. 13, 2024. [Online]. Available: https://softwaredoug.com/blog/2021/04/21/compute-mrr-using-pandas

[21] "Creating Message Passing Networks — pytorch_geometric documentation." Accessed: Feb. 13, 2024. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_gnn.html#implementing-the-edge-convolution