Friend Recommendation using Graph Neural Networks in Social Platforms

\*Note: Sub-titles are not captured in Xplore and should not be used

*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*—This electronic document is a “live” template and already defines the components of your paper [title, text, heads, etc.] in its style sheet. *\*CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract*. (*Abstract*)

Keywords—component, formatting, style, styling, insert (key words)

# Introduction

Social networks play a pivotal role in connecting individuals, and the task of friend recommendation within these networks is crucial for enhancing user experience. This project aims to address the challenge of friend recommendation using Graph Neural Networks (GNNs). The objective is to leverage GNNs to predict and rank potential new friends for users within social networks, particularly focusing on dataset from Facebook and Twitter.

The primary challenge is to develop a friend recommendation system capable of providing accurate and effective suggestions within the given social networks. The anonymized nature of the data introduces complexities in understanding individual characteristics, and the large-scale network necessitates scalable algorithms.

The following sections cover…

# Related work

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

**Friend Recommendation:** In recent years, graph embedding methods have gained prominence for friend recommendation tasks. These methods learn unsupervised embeddings techniques to generate embeddings that capture the structural features of nodes within the graph[1]. Widely used embedding models such as node2vec[2] achieve this by finding the likelihood of node in random walk. As a result, these methods demonstrate effective performance in predicting links between nodes in the graph.

However, these methods make them less practical for large-scale networks[2], [3]. As the size of the network grows, the computational resources required for training and inference also increase significantly. Therefore, while graph embedding methods offer promising solutions for friend recommendation tasks, their constraints challenges in large-scale social networks.

**Graph Neural Networks:** GNNs propagate information from local neighborhoods of nodes throughout the graph. A notable architecture within GNNs is Graph Convolutional Networks (GCNs)[4], which learn by node degrees, utilizing the graph Laplacian matrix. Many models have expanded upon GCNs, introducing various learnable aggregators such as self-attention mechanisms, mean pooling, and max pooling functions. These advancements have consistently outperformed embedding techniques based on random walks.

In research study[3] conducts experiments on extensive datasets and they formulated friend suggestion as a friend ranking problem. They propose 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 the ranking metrics for evaluation. In this study, we model a baseline GCN to learn friend ranking and compare with additional embedding algorithm.

# Methodology

Intro paragraph to lead to dataset

The dataset was obtained through McAuley and Leskovec prior work on social circles in ego networks, currently hosted on the Stanford SNAP dataset library[5].

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 names of each of the feature dimensions (nodeId.featnames) from Facebook, and Twitter. Nodes, features (profiles), circles, and ego networks have been anonymized to ensure user privacy. The dataset anonymization is achieved by replacing original IDs and obscuring feature interpretations. Table 1 provides a brief description of the aforementioned datasets:

## 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 individual user characteristics. Then, the featnames were mapped to their respective feature names for each column, establishing a clear association between feature identifiers and their names. As part of the process, all networks were consolidated to create a final, integrated file. This file incorporates all features precisely aligned with their respective names.

The refined dataset serves as a valuable resource for model training and facilitates further in-depth analysis within our science 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 applied to the large-scale network ego-Twitter initially. Hardware limitations lead to an alternative approach on extracting features from the twitter dataset, and a rising opportunity on a novel categorical feature extraction process. Each ego feature and user feature file were 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 in succession encoded in integer representation embeddings, while edge pairings were mapped so 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 size 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 following sections.

Subsequently, a custom dataset class derived from PyTorch Geometric data processing library was defined to encapsulate the data and store them in 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, suitable for training, validation, and experimentation with graph neural networks.

## Graph Neural Networks Background

Graph Neural Networks (GNNs) are a category of neural networks specifically designed for processing and learning from graph-structured data. In contrast to conventional neural networks, which 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 from neighboring nodes within the graph. This iterative process enables GNNs to capture the local graph structure and propagate information 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, Graph Isomorphism Networks (GINs), among others.

Our approach on 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 suggestion based on higher relevancy ranking score. Finally, the GCN model effectiveness was evaluated against a baseline TruncatedSVD model for the same task of learning node embeddings.

## Graph Convolutional Network Model

In the context of modeling 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)[6] and is designed to learn information about the users and calculate features for each node based on the neighbor nodes. 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 extracting 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 positive and negative scores against the accuracy of the predictions of relationships between users.

*a**b* 

The reason that we used dropout layer in this work is to prevent overfitting during the 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 input units during training, forcing the network to learn more features.

The Stochastic Gradient Descent (SGD) optimizer is 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's performance. The momentum term, set to 0.9, speeds up convergence and reduces oscillations by adding momentum to the parameter updates.

In addition, the StepLR scheduler is 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 more stable and efficient convergence of the optimizer. Consequently, the scheduler is used to manage the learning rate throughout training, improving the optimization process and enabling more effective parameter updates.

Furthermore, the loss function quantifies the disagreement between predicted logits and true labels by computing the binary cross-entropy, thereby facilitating the adjustment of model parameters to minimize this discrepancy. By penalizing deviations between 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 initializes the chosen model along with a predictor, optimizer, and learning rate scheduler. It then conducts training over a specified number of epochs (30,50,100,200,1000 epoch), computing the loss, performing propagation, and updating model parameters. After training, the function evaluates the model's performance on a test set, calculating the Area Under the Curve (AUC) score and printing the loss. Figure 2 illustrates our GCN model architecture.

The additional baseline model used as reference was designed based on the TruncatedSVD model. TruncatedSVD 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 handle sparse user-item interaction matrices efficiently. 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 to the GCN implementation. It features the same SGD optimizer and StepLR scheduler as the GCN model.

## 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 utilizes as input the edge index matrices, and the node features h, as learnt from the GCN/TruncatedSVD model. For each edge pairing, the module maps the features with the source node and the destination node.

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

## Friend Recommendation

Generator (targeted)

Calculator (all)

Ranking process

# experiments

In this section, we present the experiments of this study. We report on our experimental setup, describing in detail the datasets we used as input and how we preprocess them, the evaluation metrics we used to assess the model’s effectiveness 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. Lastly, we showcase the results of our comprehensive experiments, state our observations and offer insights.

## Experimental Setup

### Dataset

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

The ego-Facebook dataset contained information about 10 anonymized ego networks, with 4.039 nodes, 88.234 edges and 1.406 features describing each node, as presented on [TABLE X]. 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 the information about the graph connectivity.

In sequence, the dataset was loaded into the graph representing PyTorch Geometric 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 in contributing to a focused training environment 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 with the ego-Facebook dataset, the ego-Twitter dataset was encoded in categorical values, which in succession were converted to word embeddings to be loaded through the “Data” class. The remaining sampling procedure was performed unaltered, as described above.

### Evaluation Metrics

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

These metrics serve a crucial role in measuring the accuracy of the model’s capability in identifying potential new user connections / friendships, along with the effectiveness of our model in predicting the top relevant neighbor users that are likely to attract the stakeholder users' interest. 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:

#### Accuracy Based Metric: the model's performance is evaluated on the test set using the Area Under the Curve (AUC) metric. AUC is a statistical metric representing the area under the receiver operating characteristic (ROC) curve, where higher values indicate better discrimination between classes.

#### Ranking Based Metrics: Friend recommendation tasks are sensitive to effective link predictions in both quality and the ranking position in the top K predictions. Specialized metrics are nessesary for this quantative evaluation, since standard accuracy metrics do not consider the position in the top K predictions. To rectify this shortfall, the following metrics will be used:

* Hits@K: Measures the proportion of relevant items among the top k recommendations, providing insights into the system's ability to retrieve relevant items within a given recommendation list. It is calculated through the following formula[x], with Q representing a collection of test triples that have been positioned within the top q triples.
* 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 figure [X], where Q is the total number of queries reli ∈ [0,1] is the relevancy score for the prediction and j is the rank of the prediction.
* 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[x], where q refers to the rank position of the *first* relevant prediction, and Q denotes the total number of queries.

In all cases, larger values indicate better performance.

### Training Process

Our model is trained using a configuration comprising layers of message passing graph convolutional operations, 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, is applied between layers during model training.

The training process involves 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, is employed for this purpose. Additionally, a learning rate decay of 0.5 is applied every 25 epochs using the StepLR scheduler.

The experimental evaluation is conducted on two distinct machines: one equipped with an AMD Ryzen 7 2700 Eight-Core Processor, featuring 2 threads per core and 8 GB of shared CPU memory, operating on the Ubuntu Linux 20.04.6 LTS platform, and the other utilizing an Intel Core i7 6700K 4 core Processor featuring 2 threads per core and 16 GB of shared CPU memory, operating on the Windows 10 platform.

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

## Cross Evaluation Baseline

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

Truncated Singular Value Decomposition (SVD) is a mathematical technique used for reducing 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 computational efficiency and facilitate interpretation. By retaining the most significant information encoded in the original matrix, truncated SVD enables more efficient and effective analysis of large datasets while mitigating the effects of noise and redundancy.

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

## Experiment Results

The presented table [X] showcases the performance metrics for the two models, GCN and tSVD, across the various evaluation criteria, as stated in training epochs and accuracy / ranking evaluation metrics.

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

1. Table Type Styles

#### Model Accuracy: GCN achieves 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 positive and negative instances.

#### 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.

#### 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 GCN maintains its superiority in recommending relevant items across a larger set of recommendations (Top-50).

#### 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 impeading on the ranking relevancy performance, as viewed on the figure [X].

#### Alternative approach, Twitter Dataset: Experimental results on the alternative preprocessed dataset ego-Twitter proved inconsistent, with AUC results on GCN model of 50% accuracy, while tSVD model failed to converge to solution. Hence, the application of ranking metrics was unwarranted in this particular scenario due to the model’s inability to predict new user-to-user connections.

In summary, based on the experimental results, GCN proves 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 the superior predictive and ranking capabilities of GCN. Extensive training sessions increase the model accuracy, with exceptions on ranking relevancy peaking at 200 epochs and further decreasing over time.

# Discussions

Experimental results lead to the superiority of GCN model versus tSVD in every category. While both models are baseline implementations of the appropriate literature list, we ascertain that for friend recommendation tasks, GCN models are more advantageous than tSVD implementations.

Notably, on the top 5 versus the top 50 recommendations, the model provided 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 high quality of ranking relevance at the full scale of 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 less 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. Addressing the diverse preferences and requirements of users in friendship formation is paramount, necessitating algorithms that can scale efficiently to handle large volumes of data while providing accurate and relevant 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. Hit rate was found to be underperforming, even if it achieved double the hit rate of the cross-evaluation baseline model. Further analysis on the multifaceted nature of the social network graphs and additional convolution layers or model complexity could result on higher ranking accuracy and quality of the recommendations. Additional experimentation can provide a fine-grained approximation of the ideal training epoch threshold, leading to elevated hit rate.

Large-scale datasets like the ego-Twitter preprocessing task and the training task emerged 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.

# Conclusions

Conclusions paragraph.

##### References

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*.