Report for Course Project of Data Science Fundamentals: Graph Based Recommendation

Wenzheng Pan
520030910232
pwz1121@sjtu.edu.cn

Kuan Yang
518030910372
yang-kuan@sjtu.edu.cn

Abstract

Graph Neural Network (GNN) has performed well in tasks such as node classification and link prediction for heterogeneous and isomorphic graphs. However, in specific scenarios presented in the academic platform, challenges arise in collaborator recommendations, paper recommendations, and reviewer recommendations for journals and conferences. To address the issues, we propose a Graph Attention Network (GAT) based graph learning model that follows the Relational Graph Convolutional Network (RGCN) aggregation approach. Our designed model achieves an impressive F1-score of 0.9484 in the Kaggle competition, surpassing the majority of participating teams. This paper provides a comprehensive demonstration of the core architecture of our proposed method. Detailed implementation instructions are also disclosed for the purpose of reproducibility as a course project. For further information, see the Kaggle competition at `https://www.kaggle.com/competitions/cs3319-02-project-1-graph-based-recommendation` and the availability of code at `https://github.com/wzever/RGCN-GAT-Link-Prediction` after the due date of the project.

1 Task Formulation

In the paper recommendation scenario, where scholars rely on their collaborators and the paper community for scientific research, there remains a lack of comprehensive datasets and models for citation recommendation. The given dataset specifically tailored for a recommender system. This dataset leverages the network associations between users and products, enabling the extraction of group features for user groups and product categories. By considering the relationships between products and the connections among users, our approach provides users with more diverse and improved recommendations. In the context of an academic network, the recommendation task is formulated as a link prediction problem that focuses on a collection of 6,611 authors and 79,937 corresponding papers from top journals in the field of GeoScience, along with citation information for their publications.

To model the relationships within this network, we construct a heterogeneous network comprising two distinct types of nodes: author nodes and paper nodes. In this network, each edge between an author node and a paper node signifies that the authors have read the corresponding paper, connecting authors with the papers cited within their own works. Additionally, each edge between two author nodes denotes co-authorship, while each directed edge between two paper nodes represents a citation relationship.

2 Related Work

Many recent researches have studied GNN-based recommendation system, and a general framework is shown below, mainly comprised of constructing graph from interaction between users and items, encoding node feature representation by GNN blocks with tuned aggregation methods and finally making link prediction based on the features. Below summaries four specific methods for this problem.

Graph Convolutional Matrix Completion (GC-MC) GC-MC leverages the graph structure of user-item interactions to enhance the collaborative filtering process. It employs graph convolutional operations to capture the high-order relationships between users and items, enabling the propagation of information across the graph. By incorporating both user-item interactions and graph structure, GC-MC can learn more accurate representations and make better predictions for missing entries in the user-item interaction matrix. The model utilizes a combination of low-rank matrix factorization and graph convolutional layers to jointly optimize the latent embeddings of users and items. This approach allows GC-MC to effectively capture both the collaborative filtering signals and the graph-based relationships between users and items.

PinSage The authors presents a large-scale deep recommendation engine, developing a data-efficient Graph Convolutional Network (GCN) algorithm called PinSage at Pinterest, which combines efficient random walks and graph convolutions to generate embeddings of nodes. PinSage uses localized convolutional modules to generate embeddings for nodes, starting with input node features and then learn neural networks that transform and aggregate features over the graph to compute the node embeddings. The algorithm of PinSage is very similar to GraphSage, with slight differences, and the paper is divided into two explanations: the Convolve algorithm and minibatch. The process of minibatch is the same as the process of minibatch algorithm in GraphSage. Below is the Convolve algorithm.

DiffNet DiffNet proposes a differentiable neural architecture search method. It is a framework that automates the process of designing neural network architectures. DiffNet employs gradient-based optimization techniques to search for optimal architectures by jointly learning the architecture parameters and network weights. This approach allows for the discovery of neural network architectures tailored to specific tasks, such as image classification or natural language processing, without the need for manual design or expert knowledge.

Light GCN LightGCN is a graph convolutional network model that focuses on collaborative filtering-based recommender systems. It simplifies the traditional graph convolutional network architecture by removing user-specific and item-specific embedding layers. LightGCN only relies on the user-item interaction matrix to propagate information through a layer-wise propagation scheme, allowing for efficient computation and scalability. By using the user-item interaction data directly, LightGCN can effectively capture the collaborative filtering signals in a graph structure.

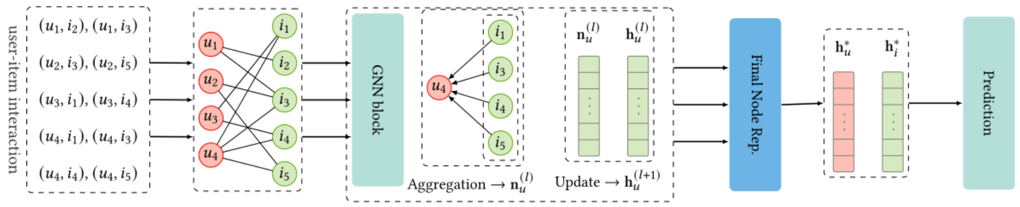

Figure 1: Commonly used modeling stucture for GNN-based recommendation tasks

In conclusion, recent related works have shown diverse ways to model the social recommendation problem based on GNN architecture. We studied some of them and plan to apply the classical design to solve link prediction in the academic graph. More methods like collaborative filtering and addition of graph attention network also perform well and will be studied during subsequent work.

3 Model Architecture

3.1 Overview

General Structure Our proposed model is structurally based on the Relational Graph Convolutional Network (RGCN) and utilizes a 4-layer Graph Attention Network (GAT) as its backbone. Fig 2 provides an intuitive visualization of the structure of the model. To ensure the rationale and reliability of our design, a series of experiments were conducted prior to its development. Experimented details are demonstrated in Table 1. Based on the experimental results, we determined that GAT outperformed GCN and SAGEConv, and we set the number of layers to 4, with 8, 4, 4, and 2 multiple attentive heads, respectively.

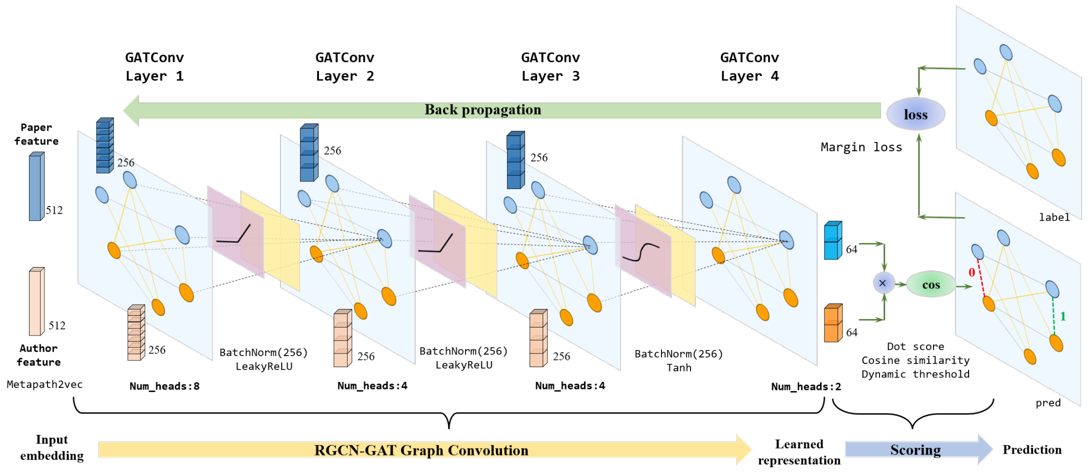

Figure 2: Proposed RGCN-GAT model architecture

Feature Embeddings For the initialization of node embeddings, we pretrained a Metapath2vec feature for author nodes and utilized the provided semantic features for paper nodes, both of which were represented by 512-dimensional vectors. For the hidden space, we utilized 256-dimensional embeddings as intermediate representations. For the output feature, we opted for a 64-dimensional representation to strike a balance between performance and computational efficiency.

Elaborate Settings Our experiments revealed that the inclusion of individual batch normalization layers for each node type after each layer significantly impacted the performance of the model. Furthermore, the selection of activation functions played an important role in fine-tuning our model. Experimental results also indicate that leaky_ReLU activation functions in three layers and a tanh function in the final layer produce great outcomes.

Final prediction After obtaining the final embeddings, a dot-predictor was employed to compute the score for link prediction. Subsequently, cosine similarity was calculated, and a dynamic threshold was applied to determine the existence of links between the currently focused node pairs, indicating whether or not a recommendation should be made.

Table 1: Performance of model with different types of graph convolutions, number of layers, normalization settings, aggregation types and featuer dimensions. * means submitted evaluation on Kaggle. Unshown settings/parameters remain untuned and fixed for comparison.

Num_layers	Hid_dim	Out_dim	Num_heads	Batch norm	RGCN aggr	F1-score
SAGE * 3	512	128	-	-	SUM	0.8501*
GAT * 3	512	256	4, 4, 4	-	SUM	0.8891*
GAT * 3	512	128	4, 4, 4	-	SUM	0.8934*
GAT * 2	1024	256	4, 4	2	MEAN	0.9249
GAT * 2	512	32	4, 4, 4	2	MEAN	0.9316
GAT * 4	256	32	4, 4, 4, 4	3	MEAN	0.9368
GAT * 4	256	64	4, 4, 4, 4	3	MEAN	0.9372*
GAT * 4	**256**	**64**	**8, 4, 4, 2**	**3**	**MEAN**	**0.9411***

3.2 Structural Basis: R-GCN

RGCN, short for Relational Graph Convolutional Network, handles graph tasks such as link prediction on heterogeneous graphs, where different types of nodes and relationships exist [9]. In the context of

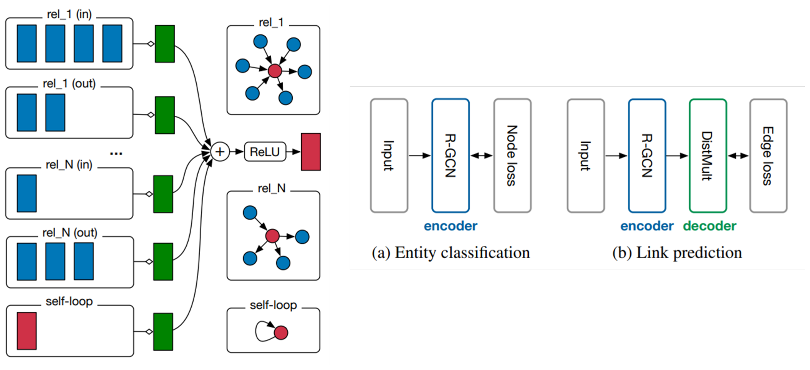

Figure 3: Diagram for Relational Graph Convolution Network

link prediction in our taSK, the goal is to predict missing or potential links between author nodes and paper nodes. RGCN operates by propagating information through the graph, with each layer capturing the relational dependencies between nodes. It assigns each relationship type a unique weight matrix, enabling the model to differentiate between different relationships during information propagation. By learning from the graph structure, RGCN can effectively predict links between specific types of nodes. Specifically, the model would consider the relationships between authors with co-authorship as well as citations among papers to predict potential new links or collaborations between authors. Guided by experimental results, the aggregator is selected to be `mean`, which averages the features computed by subordinate backbone network (GAT in our approach) of three types of edges to generate the final message.

3.3 Feature Initialization: Metapath2vec

Metapath2vec is a graph embedding technique designed to capture semantic similarities in heterogeneous networks. It is particularly useful for capturing structural and contextual information in networks with diverse node and relationship types. Metapath2vec leverages random walks on the network, combined with skip-gram or CBOW algorithms from natural language processing, to generate meaningful embeddings for nodes in the graph. These embeddings can be used for various downstream tasks such as node classification, recommendation systems, and similarity analysis.

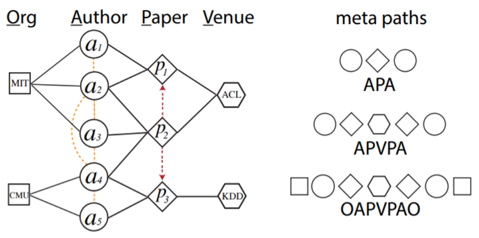

Figure 4: Diagram for Metapath2vec method

In our task, given the fact that only the paper nodes have initial features by semantic analysis, we suppose that author nodes should contain initial embedding with same dimensionality and naturally turn to Metapath2vec. Formally in the academic recommender system, a meta-path scheme $\mathcal{P}$ is defined as a path that is denoted in the form of $\texttt{author}\xrightarrow{\text{ref}}\texttt{paper}\xrightarrow{\text{beref}}\texttt{author}$. We trained the meta-path by `SparseAdam` for 10 epochs and output 512-dimensional embeddings to initialize the author nodes.

3.4 Backbone: Multi-layer GAT

GAT, which stands for Graph Attention Network, is a deep learning model designed for processing and analyzing graph-structured data [2]. GAT employs the concept of attention mechanisms, allowing the model to selectively focus on different parts of the input graph with different weights, or namely attentive coefficients, according to the similarity between nodes. These attention coefficients are learned during the training phase, allowing the model to adaptively assign different weights to different neighbors for each node. By considering the neighborhood information in a context-aware manner, GAT can capture complex dependencies and propagate information effectively through the graph.

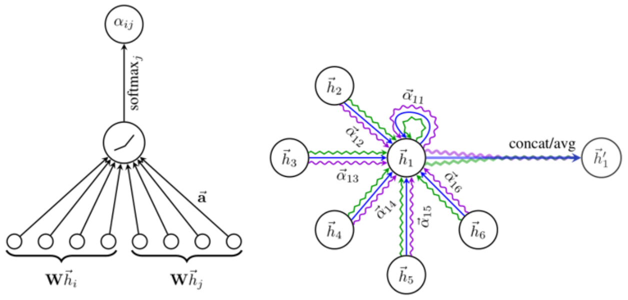

Figure 5: Computation of attention coefficient and message passing in GAT model

For each node i and all its neighbor j, GAT computs their similarity by

$$e_{ij} = \alpha([Wh_i||Wh_j]),\ j \in \mathcal{N}_i \tag{1}$$

where $[\cdot||\cdot]$ denotes concatenation of tensors and α calculates edge-wise normalization by softmax

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(e_{ij}))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(e_{ik}))} \tag{2}$$

GAT typically consists of multiple attention heads, each independently computing attention coefficients. This enables the model to capture different patterns and relationships simultaneously. The outputs of the attention heads are then combined to produce the final node representations.

$$h'_i(K) = \text{concat}_{k=1,\ldots,K}\ \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k W^k h_j\right) \tag{3}$$

For the final layer, averaging all heads instead makes better sense for output

$$h_i^{(l+1)}(K) = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k W_k^{(l)} h_j^{(l)}\right) \tag{4}$$

The selection of GAT as backbone in our approach has been due to its state-of-the-art performance on several graph-related tasks, surpassing previous graph convolutional network (GCN) models. Its ability to capture local and global dependencies through attention mechanisms makes it particularly effective in scenarios where information propagation and feature extraction across the graph are crucial. Equally importantly, the GAT module is suitably compatible for heterogeneous graph learning in our task setting and is conveniently embedded under the R-GCN architecture.

4 Implementation Details

4.1 Sampler

We based our project on the DGL framework, which is graph-learning friendly. DGL library provides two negative samplers for heterogeneous training. Table 2 demonstrates the performance of models with different samplers.

Uniform sampler Negative sampler that randomly chooses negative destination nodes for each source node according to a uniform distribution.

Global uniform sampler Negative sampler that randomly chooses negative source-destination pairs according to a uniform distribution.

As anticipated, the pair-wise global uniform sampler demonstrates superior performance compared to randomly selecting source and destination nodes independently. This discrepancy in performance can be attributed to the nature of the link prediction task, where the primary objective is to determine the presence or absence of each edge, rather than focusing solely on node sampling. In this context, the pair-wise global uniform sampler proves to be more effective, as it captures the inherent dependencies between source and destination nodes when generating training samples.

Table 2: Performance of model with different type of **negative samplers** and batch-sizes. * means submitted evaluation on Kaggle. Aforementioned settings/parameters are fixed optimal and others remain untuned for comparison.

Negative sampler	Batch-size	Loss function	F1-score
Uniform	Full graph	BCE loss	0.8934*
Uniform	16384	Margin loss	0.9371*
Global uniform	**8196**	**Margin loss**	**0.9395***

Initially, our attempts to train the model using binary cross-entropy (BCE) loss on the entire graph yielded suboptimal results compared to training on mini-batches within a single epoch. This observation suggests that applying gradient descent globally may cause the model to converge towards local minima.

Table 3: Performance of model with different **sampling size** from both positive and negative samplers. Aforementioned settings/parameters are fixed optimal and others remain untuned for comparison.

Positive samples	Negative samples	Batch-size	F1-score
4	(global pairs) 2	11000	0.9422
4	(global pairs) 3	11000	0.9445
4	**(global pairs) 4**	**10000**	**0.9447**
4	(global pairs) 5	8000	0.9433
4	(global pairs) 6	6000	0.9434
4	(global pairs) 7	5000	0.9432
5	(global pairs) 5	16384	0.9369
5	(global pairs) 5	8192	0.9319

In order to investigate the impact of the number of samples provided by the samplers on model performance, we conducted a series of experiments, the results of which are presented in Table 3. It is important to note that the batch size was adjusted to accommodate the limited memory of the server GPU, which is approximately 10.76GB.

Based on the aforementioned findings, we made the decision to utilize global uniform samplers, each providing 4 samples of edges per epoch. The substantial improvement in F1 performance, reaching a value of 0.94416, serves as strong evidence supporting the credibility and effectiveness of our methodology.

4.2 Loss Function

Given that we have framed the recommendation task as a link prediction problem, which essentially equals binary classification, we initially experimented with binary cross-entropy (BCE) loss. Subsequently, we explored the use of focal loss and margin loss as alternative approaches to compare their performance. Focal loss primarily aims to address the challenge of imbalanced data by assigning higher weights to misclassified samples, while margin loss learns from the original metrics (cosine similarity in our case) instead of providing binary predictions. Below lists their formulation, respectively.

$$\mathcal{L}_{BCE} = -\frac{1}{n}\sum_i y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i}) \tag{5}$$

$$\mathcal{L}_{margin} = \max\left(\frac{1}{n}\sum_i 1 - \text{pos}_i + \text{neg}_i, 0\right) \tag{6}$$

$$\mathcal{L}_{focal} = -\frac{1}{n}(1 - p_t)^{\gamma}\log(p_t) \tag{7}$$

where p_t equals $\hat{p}$ if $y_i = 1$ or $1 - \hat{p}$ otherwise.

Table 4: Performance of model utilizing different **loss functions**. * means submitted evaluation on Kaggle. Aforementioned settings/parameters are fixed optimal and others remain untuned for comparison.

Loss function	F1-score
BCE loss	0.9424
Focal loss	0.9367
Margin loss	**0.94333***

Table 4 displays the comparative performance achieved using various loss functions. Our experimental results indicate that margin loss is the preferred choice. However, it is worth noting that when opting for margin loss, the memory cost on the GPU is significantly higher compared to other loss functions due to its computational characteristics.

4.3 Dynamic Thresholding

After obtaining the scores for all edges to predict, a crucial task is to determine an effective threshold for generating the final binary predictions. Empirically, a fixed threshold of 0 is often used since cosine scores typically range from -1 to 1. In this approach, positive scores are mapped to 1 (indicating the presence of a link), while negative similarity scores are mapped to 0 (indicating the absence of a link) in the prediction. However, in our experiments, we observed that the scores computed by the model fluctuate throughout the training process, leading to variations in the score distribution. To address this, we initially explored using statistical measures such as the mean or median as the classification threshold. This approach resulted in a notable improvement in F1 performance. Building upon this idea, we further propose the **dynamic median policy** outlined below.

- Set median of scores (cosine similarity) as initial threshold θ
- Search for θ^* in range $[\theta - \delta\theta, \theta + \delta\theta)$ that maximizes F1 score, where $\delta\theta$ is a hyper-parameter
- Set $\theta_{i+1} = \theta_i^*$ and repeat searching during epoch iteration

Table 5: Performance of model utilizing different **threshold policies**. * means submitted evaluation on Kaggle. Aforementioned settings/parameters are fixed optimal and others remain untuned for comparison.

Threshold policy	F1-score
Fixed value	<0.92
Mean	0.936
Median	0.93716*
Dynamic median	**0.9447**

Table 5 displays the experimental results obtained using different threshold options. Notably, our dynamic median approach achieves an impressive F1 score of 0.9447, outperforming the other methods. This outcome substantiates the correctness and effectiveness of our proposed policy.

4.4 Tuned Hyper-parameters

The structural settings and training hyper-parameters in our model are both complex and numerous, with each parameter having a distinct impact on the overall performance. To provide transparency and reproducibility, we present Table 6, Table 7 and Table 8, which showcase different composition of activation functions, a range of values for the learning rate and weight decay, along with their corresponding performance outcomes. Furthermore, Table 9 offers a comprehensive list of all tunable settings and parameters that contributed to the performance achieved in our submitted results, enabling replication of our report.

Table 6: Performance of model with different **activation functions** after each layer. * means submitted evaluation on Kaggle. Aforementioned settings/parameters are fixed optimal and others remain untuned for comparison.

Learning rate	Weight decay	Activation function	F1-score
0.0005	0.0005	Tanh * 3	0.9366
0.0005	0.0005	ELU * 3	0.9319
0.0005	0.0005	ReLU * 3	0.9326
0.0005	0.0005	Leaky_ReLU * 3	0.9330
0.0005	**0.0005**	**Leaky_ReLU * 2 + Tanh**	**0.93716***

Table 7: Performance of model with different **weight decay**. * means submitted evaluation on Kaggle. Aforementioned settings/parameters are fixed optimal and others remain untuned for comparison.

Learning rate	Weight decay	Activation function	F1-score
0.0002	0.001	Leaky_ReLU * 2 + Tanh	0.9431
0.0002	0.0008	Leaky_ReLU * 2 + Tanh	0.9427
0.0002	0.0006	Leaky_ReLU * 2 + Tanh	0.94333*
0.0002	**0.0004**	Leaky_ReLU * 2 + Tanh	**0.94511***
0.0002	0.0002	Leaky_ReLU * 2 + Tanh	0.9430
0.0002	0.00008	Leaky_ReLU * 2 + Tanh	0.9408

Table 8: Performance of model with different **learning rate**. * means submitted evaluation on Kaggle and † means step scheduler is used for learning rate adjustment.

Learning rate	Weight decay	Activation function	F1-score
0.0005	0.0004	Leaky_ReLU * 2 + Tanh	0.9421
0.001	0.0004	Leaky_ReLU * 2 + Tanh	0.9406
0.0008	0.0004	Leaky_ReLU * 2 + Tanh	0.9393
0.0006	0.0004	Leaky_ReLU * 2 + Tanh	0.9410
0.0004	0.0004	Leaky_ReLU * 2 + Tanh	0.9429*
0.0002	0.0004	Leaky_ReLU * 2 + Tanh	**0.94405***
0.000198†	0.0004	Leaky_ReLU * 2 + Tanh	**0.94524***
0.0002†	**0.0004**	**Leaky_ReLU * 2 + Tanh**	**0.94632***

Based on our findings, it is evident that the choice of learning rate and weight decay parameters has a notable impact on the performance of the model. After conducting experiments, we determined that a learning rate of 2.00E-04 and a weight decay value of 4.00E-04 yielded outstanding results, achieving a significant score of 0.94405.

To further enhance the model performance, we observed the decreasing curve of the loss function and decided to incorporate the `StepLR` scheduler. This scheduler dynamically adjusts the learning rate during training, thereby optimizing the model's performance. Specifically, we tuned the scheduler to decay the learning rate by a factor of 0.78 every 10 epochs. As a result of this adjustment, we observed a promising F1 performance exceeding 0.946.

By strategically incorporating the `StepLR` scheduler and fine-tuning the learning rate, we were able to optimize the model performance and achieve impressive results. These findings highlight the importance of careful parameter selection and dynamic adjustments in training deep learning models.

Table 9: Reproducible settings and hyper-parameters for training

Name	Description	Value/Setting
loss_func	Loss function	Margin loss
num_layers	Number of GAT layers used	4
in_dim	Dimension of input node feature	512
h_dim	Dimension of hidden node embedding	256
out_dim	Dimension of output node embedding	64
optimizer	Optimizer for gradient descent	Adam
lr_scheduler	Scheduler to adjust learning rate dynamically	StepLR
num_epochs	Maximum training epochs	100
lr	(Initial) Learning rate	0.0002
wd	Weight decay	0.0004
lr_period	Number of epochs of learning rate decay	10
lr_decay	Multiplicative factor of learning rate decay	0.78
seed	Random seed for reproduction	1063
batch_size	Training and testing batch size	10000

4.5 Voting Mechanism

In our pursuit of the performance of our model, we devised a novel approach inspired by techniques commonly employed in distributed systems. Recognizing that a single prediction can be influenced by various subtle factors, we introduced a voting mechanism to mitigate the impact of individual predictions. This technique involved selecting up to 10 prediction CSV files that achieved the highest F1 scores and allowing them to collectively "vote" on each entry in the test set, determining whether it should be labeled as 1 or 0.

To determine the final label for an entry, we set a threshold of 6 or more positive predictions out of the selected files. If at least 6 predictions indicated a positive outcome, the entry would be written 1. This innovative method effectively expanded the information considered for the final decision and proved to be reliable and high-performing. By incorporating this voting mechanism, we were able to further enhance the F1 score from a fine-tuned value of 0.946 to an impressive 0.948.

The introduction of this voting mechanism exemplifies our commitment to exploring unconventional approaches and leveraging ideas from diverse domains to improve the overall performance of our model.

5 Conclusion and Discussions

5.1 Final Performance

Our RGCN-GAT model, after numerous refinements and experiments over the past few weeks, has demonstrated excellent performance in the task of link-prediction based academic recommendation. It achieved a remarkable F1 score of **0.94841** on the test dataset submitted to Kaggle, securing a **top 3** ranking among all valid teams in the course class.

prediction_final_0.0001.csv
Complete · Wenzheng Pan · 5d ago
Score: 0.94841

Figure 6: F1-score of our best submission on Kaggle.

5.2 Conclusion

Table 10 provides an overview of the key milestones achieved throughout the development of our project. These milestones encompass notable enhancements and innovative techniques that have significantly improved the performance of our model. Notably, the first milestone should be the correct selection of graph convolution layer type and number of layers. We have also diligently recorded the dates of these adjustments to acknowledge our continuous dedication to this project.

Table 10: Significant changes in model design or training procedure and their contribution to thre performance. All F1-scores are from submitted version on Kaggle.

Milestones	F1-score	Increasing rate	Date
2-layer GCN	0.77338	-	5/19
2-layer SAGEConv	0.79219	2.43%	5/19
3-layer SAGEConv	0.82675	4.36%	5/20
Adding bidirected edges	0.84376	2.06%	5/20
3-layer GAT	0.89338	5.88%	5/21-5/23
Applying mini-batch training	0.90803	1.64%	5/24
4-layer GAT with median threshold	0.92849	2.25%	5/24-5/25
Applying batch normalization	0.93716	0.93%	5/25-5/27
8-4-4-2 multi-head attention	0.93952	0.25%	5/28
Dynamic median threshold	0.94108	0.16%	5/29
Global-uniform sampler	0.94333	0.24%	5/30
Applying voting mechanism	0.94823	0.52%	6/1
Tuning hyper-parameters	**0.94847**	0.03%	All along

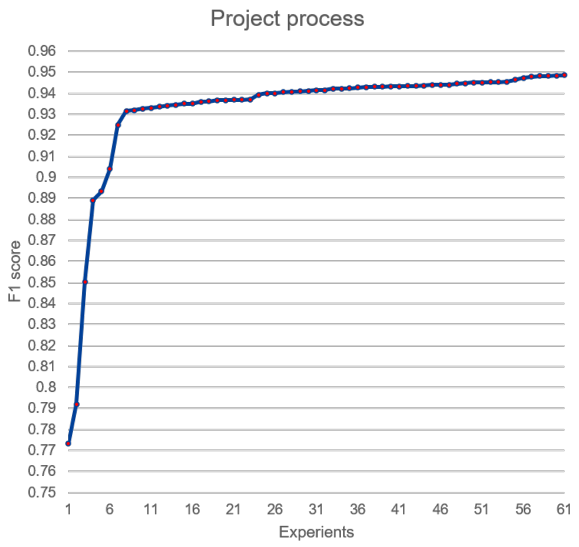

Figure 7: F1-score curve throughout the entire project development.

Demanding and grueling as the task might appear, we have derived great satisfaction from transforming our original ideas into reality and witnessing the evolution of our model through constructive competition as well as communication. Figure 7 visually illustrates the evolutionary process of our model's performance through a series of experiments and dedicated efforts.

5.3 Future Improvement

In terms of future work, there are several potential avenues for further improving the performance of our RGCN-GAT based model in the academic recommendation task.

Edge Feature One possible enhancement is to consider incorporating edge features into the model architecture. Edge features can provide valuable information about the relationships between nodes and potentially enhance the representative ability of the model to capture complex patterns and dependencies.

Better Backbone Exploring alternative layer backbones is another promising direction for future research. While the RGCN-GAT architecture has shown strong performance, investigating more state-of-the-art layer backbones, such as Heterogeneous Graph Transformer (HGT), LightGCN, DiffNet and GC-MC [4, 5, 6, 7], may reveal additional insights and lead to further improvements in accuracy and robustness.

Better Initial Embedding Enhancing the initial node embeddings is another aspect that could be explored. By incorporating domain-specific knowledge or utilizing more advanced techniques for node embedding generation, we can potentially provide the model with richer and more informative representations of the nodes, enabling it to make more accurate predictions.

Better Tricks Other techniques, such as residual connections, and dropout, can also be investigated to enhance the model's generalization capabilities and mitigate overfitting. Additionally, exploring hybrid approaches that combine different GNN architectures with other algorithms or incorporating topological features like deep walk and node2vec could lead to more comprehensive and powerful models for link prediction tasks.

Acknowledgments

Data science and graph learning hold immense significance in future research, particularly in the era of artificial intelligence. Sincere appreciation goes to Prof. Ding for lecturing in this insightful course and all TAs for their conducive guidance.

References

[1] Wu, S., Sun, F., Zhang, W., Xie, X., & Cui, B. (2022). Graph neural networks in recommender systems: a survey. ACM Computing Surveys, 55(5), 1-37.

[2] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

[3] Dong, Y., Chawla, N. V., & Swami, A. (2017, August). metapath2vec: Scalable representation learning for heterogeneous networks. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 135-144).

[4] Hu, Z., Dong, Y., Wang, K., & Sun, Y. (2020, April). Heterogeneous graph transformer. In Proceedings of the web conference 2020 (pp. 2704-2710).

[5] Berg, R. V. D., Kipf, T. N., & Welling, M. (2017). Graph convolutional matrix completion. arXiv preprint arXiv:1706.02263.

[6] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020, July). Lightgcn: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 639-648).

[7] Wu, L., Sun, P., Fu, Y., Hong, R., Wang, X., & Wang, M. (2019, July). A neural influence diffusion model for social recommendation. In Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval (pp. 235-244).

[8] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 974-983).

[9] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. In The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15 (pp. 593-607). Springer International Publishing.