Goodreads recommender system using GNN

BONVALOT Isione
Robotic Section
EPFL
Lausanne, Switzerland
isione.bonvalot@epfl.ch

TREIL Léonard
Robotic Section
EPFL
Lausanne, Switzerland
leonard.treil@epfl.ch

***Abstract*—This project presents a book recommendation system leveraging Graph Neural Networks (GNNs) made for Goodreads users. Traditional recommender systems like collaborative filtering (CF) often rely on either user or item characteristics, failing to capture the complex underlying structure. Our approach addresses this by utilizing GNNs, which can effectively model interactions between users and items by capturing the graph structure and propagating embeddings through neural message passing.**

The system constructs a bipartite graph of users and books, extending to a tripartite graph incorporating genres to enhance recommendations. The best model, incorporating user and book features, achieved a test accuracy of 0.835 and a test AUC of 0.9229. These results underline the efficacy of GNNs in capturing underlying relationships within data, providing a robust solution for personalized book recommendations.

***Index Terms*—GNN, recommender systems, Heterogeneous Graphs**

I. Introduction

Recommender systems help to find relevant information among almost infinite options. They are widely used by popular apps like Netflix, Amazon Prime, and online shopping to give users the impression of personalized choices based on what they like. Indeed, the quantity of online products or content is rapidly growing. It is, hence, difficult to manually choose which options are the most relevant to them. So, recommender systems suggest options that are considered interesting to the users.

Initially, recommender systems work as filters. For example, collaborative filtering (CF) recommends an item that other similar users like. So CF predicts user interest in items based on similarity with historical user interest patterns. They can also be based on user or item characteristics. While these models could work, they do not capture enough information because they only use user or item features to predict the suggestion.

In recent years, graph neural networks (GNN) have gained popularity. One popular application of GNN is building recommendation systems, especially because of its complex modeling capabilities. Indeed, compared to classical machine learning models, GNN can leverage the interactions between users and items and their features. Specifically, they can capture graph structure information, such as the K-hop paths between two nodes, using deep neural network approaches.

This project aims to build a book recommender system using GNNs. The code is available on GitHub.

II. Background

A. *Recommender systems for book readers*

Recommendation systems are commonly used by book readers. Given a set of users U and books B, they output x books scores S. In other words, given a user, the system recommends the x books with the highest score they have not yet read.

B. *GNNS*

GNNs generate embeddings through information propagation, also called neural message passing. They allow the propagation of embeddings along the edges of the node. Embedding-based models learn an embedding e_u for each user and e_i for each item. They pass messages among nodes along their edges and aggregate them for each node to update itself. Each node gathers information from multi-hop neighbors and infers structural features around them. The number of neighbors reached corresponds to the number of layers of the GNN.

1) GNN architecture: The inputs to the GNN at the first layer will be the input features of the node. Each layer of a GNN consists of three fundamental components: the message function, aggregation function, and update function.

The message function sends information from neighbors to the current node. A common way to do this is to use the hidden representation of the neighbor node and pass it into the computation to create the current node's embedding. Node embeddings encode information such as similarity between different nodes or graph structure information, which can be helpful for predictions. This allows users with similar interests and items with similar users' interests to have embeddings converge toward each other and optimize the scoring function for a recommendation.

The aggregation function combines the message information each neighboring node produces during the message function. Some common aggregation functions are mean and summation.

The update function updates the node's representation based on the output of the aggregation function. An update function usually has a multi-layer perceptron, a skip connection, and/or a non-linear activation.

2) Predictions: Predicting preferences is done through similarity metrics such as cosine similarity, euclidean distance etc.

III. Method

To build the recommender system, we followed the procedure:

- Preprocess and clean the dataset to extract only the relevant information
- Split the dataset into training, validation and testing sets
- Generate embeddings for all users and items using a GNN
- Train a model to predict the users' preferences
- Evaluate the predictions

A. Dataset

The dataset used for the experiments is Goodbooks-10k [4], which contains six million ratings for the ten thousand most popular (with the most ratings) books. There are also books marked to be read by the users and book metadata (author, year, etc.). The extended version (Goodbooks-10k extended [5]) adds features like a book or user favorite genres.

1) Dataset filtering: To ensure the dataset's quality, the users have at least 20 books rated. From the dataset, we extracted the favorite genres of the users. To filter the favorite genres for each user, we select the genres of all the books read and rated above 4 over 5 stars. Then, we select the 5 first genres with the highest score. Moreover, we filter out all the genres with no more than 10 books read by the users. If it filters out all genres for a user, we assume that those users have no favorite genres. To facilitate the load of the data we created new CSV files with the information we preprocessed.

B. Graph representation

Most real-world datasets can be represented as heterogeneous graphs, which is also true in this work. They are heterogeneous because they store information about different types of entities and their different types of relations. In this project, both types of graphs were analyzed: the bipartite and tripartite graphs.

1) Bipartite graph: A bipartite graph with users and books as nodes and edges between them indicating user-item interactions is used. The graph is bipartite because users can be interested in books, but books and users are independent.

2) Tripartite graph: Another way to create the graph is to set the nodes as users, books, and genres. The edges are ratings, belongs to, favorites, (named ”rates”, ”is”, ”likes”).

3) Nodes interactions: We present the possible edges for the bipartite and tripartite graphs.
User-book interactions: A user-book interaction is a book read and rated by the user. **Book-genre interactions**: A book-genre interaction is one or multiple genres to which a book belongs. **User-genre interactions**: A user-genre interaction represents a user's favorite genres.

4) Features: It is possible to set features to the nodes. Book features include information about the different genres a book belongs to, and user features include the user's favorite genres.

C. GNN

1) Architecture: In this project, we use the mean as an aggregation function, and the update function is done via convolution and relu layers. Two layers are set in the GNN architecture because we want the message to pass from a user to a book and back to another user.

2) Predictions: Predicting preferences is done through simple cosine similarity (inner product between users and items embeddings)

D. Model training

We split the dataset into an 80/10/10 train-validation-test split and trained the model using the mini-batching approach. The split is done at the edge level, not the user level. By doing the data split at the edge level, we focus the training on predicting where a link is in the graph. Moreover, it is also important to implement negative edge sampling to have a good training set because the dataset only has positive samples. This allows the model to learn where edges aren't supposed to be in the graph.

We use a binary cross-entropy function as a loss function associated with a sigmoid activation function at the end of the classification. The interest of using a cross-entropy loss function is that this function provides a probabilistic interpretation of the outputs, which can be useful in understanding the model's confidence in the results.

IV. Experiments

A. Experiment set-up

In the project, we studied three different graphs. The graphs are summarized as follows:

1) Graph 1 ”Book features”: Bipartite graph with books and users as nodes. Edges are represented by the user-book interactions. Book nodes have book features. The graph is represented in Figure 1.
2) Graph 2 ”Book and user features”: Bipartite graph with books and users as nodes. Edges are represented by the user-book interactions. Book nodes have book features, and user nodes have user features.
3) Graph 3 ”3 nodes”: Tripartite graph with books, users, and genres as nodes. Edges are represented by user-book interactions, user-genre interactions, and book-genre interactions. Nodes do not have any features. The graph is represented in Figure 2.

B. Hyperparameter tuning

To find the best hyperparameters to train our model, we first looked at all the hyperparameters that could be modified. We can classify them into two categories: the ones associated with the data generation and those associated with the model's training. The hyperparameters we used for the data generation are mainly related to how many links must be sampled and how to sample them :

- Number of neighbors that are sampled

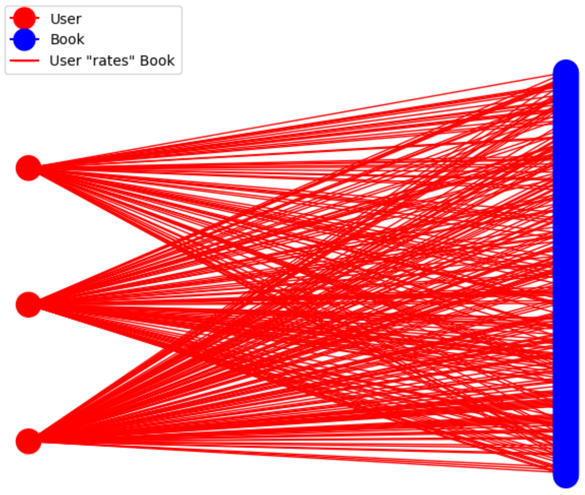


Fig. 1. Book-User Bipartie graph with only the first 3 users

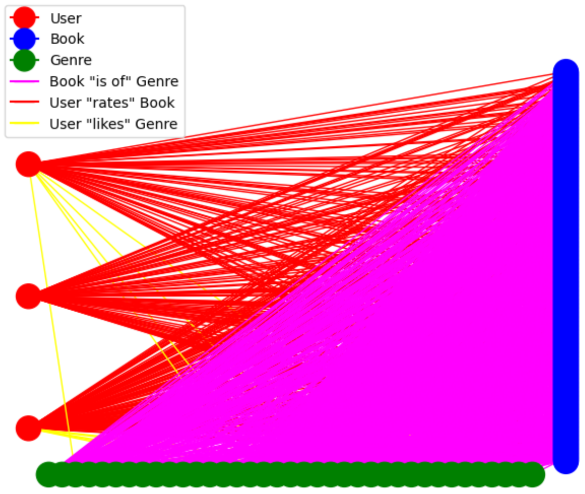


Fig. 2. Book-User-Genre Tripartite Graph with only the first 3 users

- Negative Sampling Ratio which is the ratio of the negative edge sampled compared to the positive edges that are sampled
- Batch size

Concerning the hyperparameters associated with the training of the model, we have more classical hyperparameters:

- Hidden Channels Dimensions
- Learning rate
- Optimizer

The hyperparameter tuning uses the Sweep function of Weights and Biases, which uses Bayesian optimization to search the hyperparameter space. The sweep results help us visualize which hyperparameters affect the metrics we want to optimize, as shown in the parallel coordinates plot displayed in Figures 4, 6 and 8. Parallel coordinate plots are useful in quickly seeing the trends in our hyperparameter values compared to our optimization metric.

Due to time constraints, we did not explore optimizing other hyperparameters. Still, it could be interesting to consider optimizing the number of layers of model GNN to create the embeddings or the loss function as future work.

Evaluation metrics: We used the ROC AUC (Receiver Operating Characteristic Area Under the Curve) as a validation metric. The ROC AUC metric is chosen because it provides a robust measure of the model's performance, particularly for binary classification tasks where it is crucial to consider both true positive and false positive rates. Unlike accuracy, which can be misleading in imbalanced datasets, ROC AUC evaluates the model's ability to discriminate between classes across all threshold levels, making it a more comprehensive metric for our validation purposes.

V. Results and Discussion

A. Graph 1 - "Book features"

As explained before, we used the ROC AUC as our validation metric to select the best hyperparameters for our graphs. In figures 3 and 4, we can see the results of hyperparameters' sweep experimented. We can gain interesting insights into some parameters by looking at these figures. For example, the Adam optimizer seems to work much better than SGD. The importance parameter in Figure 3 is calculated by training a random forest with the parameters and the criteria (ROC AUC metric in our case). According to this metric, a high number of hidden channels is also important for training an effective model.

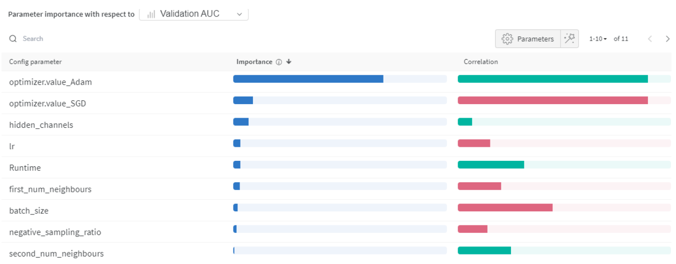


Fig. 3. Parameters Importance for Book feats Graph

Following those results, we were able to select the best model for the first graph, which uses the following hyperparameters: Adam optimizer, 64 hidden channels, a learning rate of 0.002, a batch size of 200, a sampling of neighbors of 10 for the first neighbors and 20 for the second neighbors and finally a negative sampling ratio of 2.

After selecting the best model, we computed its final performance on the test set. To have simple metrics to compare, we decided to compute the model's accuracy on the test in addition to the ROC AUC metric. The final results are presented in Table I.

B. Graph 2 - "Book and user features"

For the second graph, we proceed as done previously using node features from the book and user nodes. By looking at

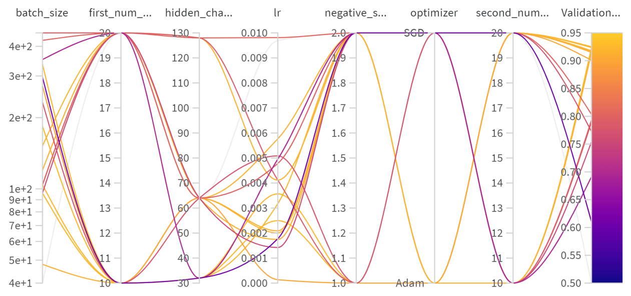


Fig. 4. Parallele plot for Book feats only sweep

the results of the sweep that are presented in the appendix, we extract the hyperparameters of the best model, which are: Adam optimizer, 128 hidden channels, a learning rate of 0.002, a batch size of 200, a sampling of neighbors of 20 for the first neighbors and 20 for the second neighbors and finally a negative sampling ratio of 2. These hyperparameters are quite similar to those of the first graph. We then compute the test metrics using this best model. Those metrics are presented in Table I.

C. Graph 3 - "3 nodes"

Finally, similarly to the aforementioned graphs, we look for the best model hyperparameters from the corresponding sweep, also presented in appendix. The hyperparameters of the best model are Adam optimizer, 128 hidden channels, a learning rate of 0.0025, a batch size of 200, a sampling of neighbors of 20 for the first neighbor and 20 for the second neighbor, and finally, a negative sampling ratio of 2. We compute the test metrics with this best model presented in Table I.

We also created the loss, training accuracy, validation accuracy, and validation ROC AUC curves for each graph, which are presented in the appendix.

TABLE I
TEST METRICS OF THE DIFFERENT GRAPHS

Test metrics	**Results for the test set on the different graphs**		
	Graph 1	***Graph 2***	***Graph 3***
Test Accuracy	0.8184	0.835	0.8171
Test AUC	0.9202	0.9229	0.9131

D. Discussion

The results demonstrate that incorporating book and user features (Graph 2) yields the highest performance, with a test accuracy of 0.835 and a test AUC of 0.9229. This indicates the model's ability to leverage additional node features significantly enhances its predictive accuracy and overall performance. However, it is interesting to note that the performance of the other models is quite similar. Hence, this demonstrates that adding more information to the graphs does not improve the results much while increasing the computational resources needed.

Indeed, graph 1 does not include user features but still reaches comparable results as the two other graphs. On the other hand, the information embedded in graphs 2 and 3 are the same but in a different form (features or edges, respectively), so GNNs learn on graph data instead of tabular data.

VI. CONCLUSION

The main results demonstrate that using GNNs for recommendation systems can achieve effective results even for the smallest model, which includes only the book features and a bipartite graph. This proves the effectiveness of using graphs that leverage structural information between users and books.

A. Further work

The preliminary results of the project achieved satisfying results. However, it could be interesting to analyze the influence of other features on the nodes. This work initializes nodes with basic features such as users' favorite genres or the genre of the books. Other interesting features to test could be adding the number of pages of the books and the year of publication (for literary currents, for instance). Moreover, it could also be interesting to test the impact of adding a user-book interaction representing the books marked as "to read" by the users.

REFERENCES

[1] " Recommender Systems with GNNs in PyG — by Derrick Li — Stanford CS224W GraphML Tutorials — Medium " https://medium.com/stanford-cs224w/recommender-systems-with-gnns-in-pyg-d8301178e377
[2] " Recommendation with Graph Neural Networks — Decathlon Digital ". https://medium.com/decathlondigital/building-a-recommender-system-using-graph-neural-networks-2ee5fc4e706d
[3] P. Geometric, " Link Prediction on Heterogeneous Graphs with PyG , Medium" https://medium.com/@pytorch$_$geometric/link-prediction-on-heterogeneous-graphs-with-pyg-6d5c29677c70
[4] " zygmuntz/goodbooks-10k: Ten thousand books, six million ratings ". https://github.com/zygmuntz/goodbooks$-$10k
[5] " malcolmosh/goodbooks-10k-extended: Extended version of the 2017 dataset, with additional fields ". https://github.com/malcolmosh/goodbooks$-$10k-extended

APPENDIX

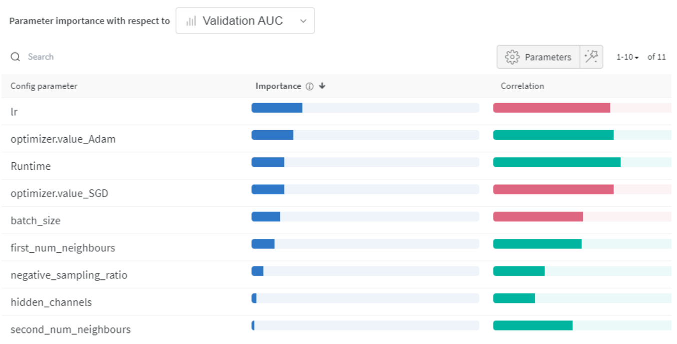


Fig. 5. Parameters Importance for Book user feats Graph

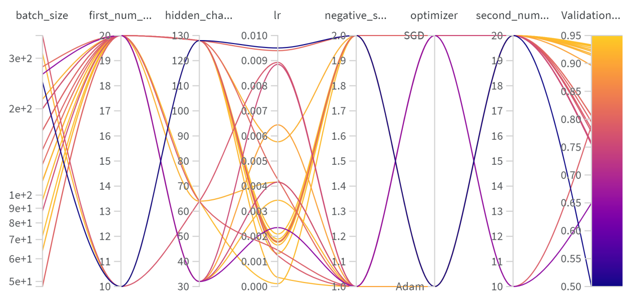
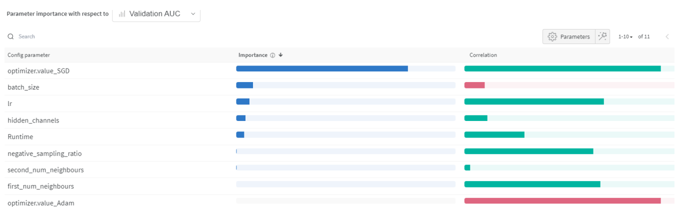


Fig. 6. Parallel plot for Book user feats sweep



Parameter importance with respect to Validation AUC

Search

Parameters 1-10 of 11

Config parameter	Importance	Correlation
optimizer.value_SGD		
batch_size		
lr		
hidden_channels		
Runtime		
negative_sampling_ratio		
second_num_neighbours		
first_num_neighbours		
optimizer.value_Adam		

Fig. 7. Parameters Importance for 3 nodes type graphs

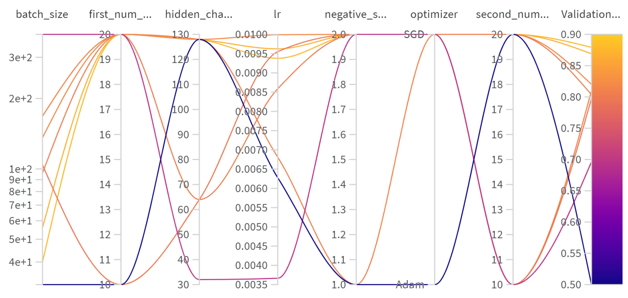


Fig. 8. Parallel plot for 3 nodes types sweep

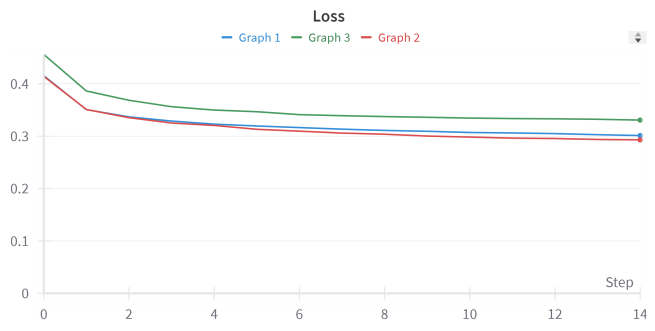


Fig. 9. Loss curve for the 3 graphs

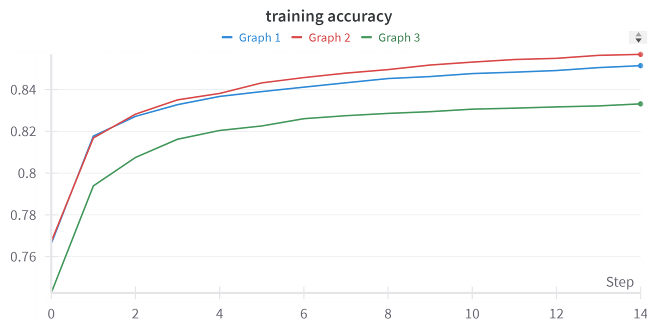


Fig. 10. Training accuracy curves for the 3 graphs

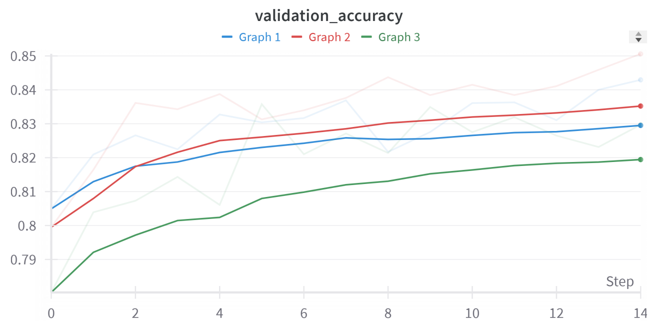


Fig. 11. Validation accuracy curves for the 3 graphs

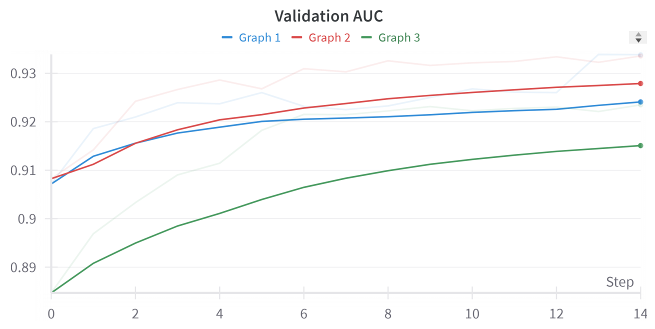


Fig. 12. Validation AUC curves for the 3 graphs