Building a Book Recommender System using Graph

Neural Network

## Arzu Isik Topbas

[arzuu.isik@gmail.com](mailto:arzuu.isik@gmail.com)

CSCI E-104 Advanced Deep Learning, 2024

May 8, 2024

# Abstract

This project explores the benefits of employing recommendation technologies, particularly focusing on books, within the context of large datasets such as Goodreads Book Reviews. By utilizing a Graph Convolutional Network (GCN) implemented through PyTorch Geometric (PyG), the project aims to personalize recommendations based on user-book interactions. The dataset comprises millions of ratings from users for thousands of books, without additional metadata. Python libraries like pandas, numpy, and scikit-learn are used for data handling and evaluation, while PyG facilitates graph manipulation. Two GCN models, LightGCN and NGCF, are implemented, each employing message passing between graph nodes to update embeddings. The Bayesian Personalized Ranking (BPR) loss function optimizes model parameters based on user interactions, with precision@K and recall@K metrics evaluating recommendation quality. Through training and evaluation, both models demonstrate effectiveness, with LightGCN generally exhibiting superior precision and recall metrics compared to NGCF.

# Summary page

Recommendation systems play a crucial role in promoting physical and digital content across various global platforms like Amazon, Apple, and Netflix. This project focuses on investigating the advantages of employing recommendation technologies specifically for books. By leveraging a Graph Convolutional Network (GCN) implemented using PyG, the project aims to analyze interactions between users and books to make personalized recommendations.

The project utilizes the Goodreads Book Reviews dataset, a subset of the Amazon Book rating dataset, containing interactions between users and books. It comprises approximately 2.9 million ratings from 52,643 users for 91,599 books. The dataset includes user-book interactions without additional metadata like user demographics or detailed book attributes.

The project is implemented in Python using various libraries and tools. PyTorch Geometric (PyG) is employed to build and optimize the recommender system using a Graph Convolutional Network (GCN). The project utilizes PyG’s functionalities for graph representation and manipulation. Additionally, other Python libraries such as pandas, numpy, matplotlib, and scikit-learn are used for data manipulation, visualization, and model evaluation.The project is implemented in Python using various libraries and tools. PyTorch Geometric (PyG) is employed to build and optimize the recommender system using a Graph Convolutional Network (GCN). The project utilizes PyG’s functionalities for graph representation and manipulation. Additionally, other Python libraries such as pandas, numpy, matplotlib, and scikit-learn are used for data manipulation, visualization, and model evaluation.

Two types of graph convolutional layers are implemented: LightGCN and NGCF (Neural Graph Collaborative Filtering). LightGCN simplifies and powers graph convolution networks for recommendation systems, while NGCF is an older architecture that applies GCNs to recommender systems. The architecture of both models involves message passing between nodes in the graph to aggregate information and update node embeddings.

The Bayesian Personalized Ranking (BPR) loss function is utilized to optimize the model parameters based on user interactions with positive and negative items. Additionally, precision@K and recall@K metrics are used to evaluate the performance of the recommender system in recommending relevant items to users.

The models are trained and evaluated over multiple epochs using training and testing datasets. The performance of the models is assessed based on total training loss, BPR training loss, regularization training loss, recall, and precision metrics. Both LightGCN and NGCF models achieve varying levels of recall and precision, indicating their effectiveness in recommending relevant books to users. However, LightGCN generally outperforms NGCF in terms of precision and recall metrics.

YouTube videos

1. Short Version: [https://www.youtube.com/watch?v=KoxsHPgpdzI](http://www.youtube.com/watch?v=KoxsHPgpdzI)
2. Long Version: [https://www.youtube.com/watch?v=HmEZaVW4tdY](http://www.youtube.com/watch?v=HmEZaVW4tdY)

# Methodology and Data

## Setup

The project setup involved integrating various essential packages, each contributing to system robustness and functionality. Crucially, meticulous version control was maintained to ensure compatibility and stability throughout development. Key packages included Torch (v2.2.0), Torch Geometric (v2.4.0), Matplotlib, NumPy, TensorFlow, Pandas, Scikit-Learn, and SciPy. This attention to versioning laid a solid foundation for experimentation and analysis.

Table 1: Package Versions

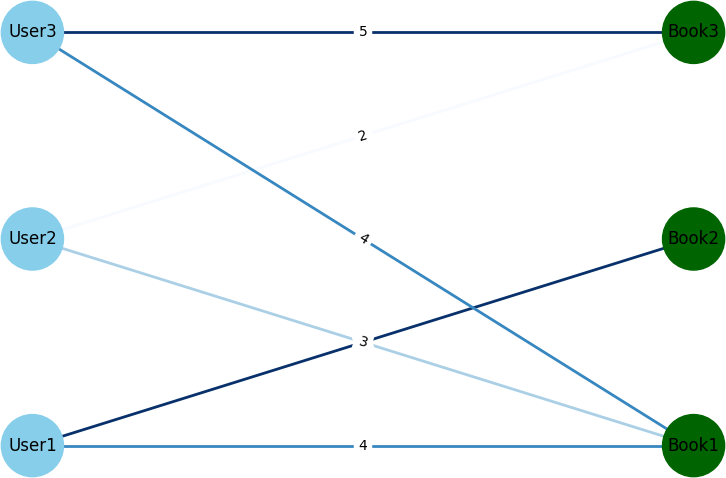
|  |  |
| --- | --- |
| **Package** | **Version** |
| torch | 2.2.0 |
| torch\_geometric | 2.4.0 |
| matplotlib | 3.8.3 |
| networkx | 3.2.1 |
| numpy | 1.26.4 |
| pandas | 1.5.3 |
| tqdm | 4.66.1 |
| scikit-learn | 1.3.0 |
| scipy | 1.11.2 |
| tensorflow | 1.11.2 |

## Dataset

For this project, we will utilize the [Goodreads Book Reviews,](https://www.kaggle.com/datasets/zygmunt/goodbooks-10k) which is a subset of the Amazon Book rating dataset referenced in the ["Item Recommendation on Monotonic Behavior Chains"](https://github.com/MengtingWan/goodreads/tree/master) paper. The complete dataset comprises over 2.3 million books, 876,000 users, and 228 million user-book interactions. Due to its size, I will be using the 10k dataset, which contains six million ratings for the ten thousand most popular books with the highest number of ratings. Additionally, this dataset includes information such as books marked as "to read" by users, book metadata (author, year, etc.), and tags/shelves/genres.

## Heterogeneous Graphs

We employ a bipartite heterogeneous graph to model the Goodreads dataset, comprising two distinct node types: users and books. Each edge in this graph signifies a user’s interaction with or reading of a particular book, accompanied by the user’s rating of the book on a scale from 0 to 5.



Constructing this graph is straightforward using the user-book interactions dataset. User nodes stem from unique user IDs, while book nodes originate from unique book IDs. The edges are derived directly from the dataset, as each entry represents a user’s interaction with a specific book.

With the graph constructed, the recommender system transforms into a link prediction task. Here, the presence of an edge between a user and a book suggests that the user would engage with (and rate) the book. This task aligns with network theory, focusing on predicting edge existence between pairs of nodes.

## Minibatch Sampling

Mini-batches are subsets of the entire dataset used for training. In this function, random users are selected, their positive interactions with items are retrieved, and negative interactions are sampled. This process helps in efficiently training the recommendation system by feeding it with diverse examples in each batch.

## Edge Index

In PyTorch Geometric (PyG), graphs are represented using sparse lists of node pairs. In our scenario of undirected graphs, each edge is duplicated to represent connections in both directions. To ensure node uniqueness, we augment the item tensor with user counts. The edge index is constructed by combining user and book node IDs for each interaction. This creates two edge types: ’user reads/rates book’ and ’book is read/rated by user,’ encapsulating user-book interactions within the graph.

# Model Architecture

## Neural Graph Collaborative Filtering

The Neural Graph Collaborative Filtering (NGCF) model is a type of Graph Convolutional Network (GCN) applied to recommender systems. NGCF aims to enhance the recommendation quality by capturing both the collaborative signal and the user-item interactions within the graph structure.

NGCF updates the embeddings of users and items through a layer-wise propagation mechanism. For each user u and item i, their embeddings are updated as follows:

For user u:

(*k*+1)

(*k*) Σ 1

(*k*)

(*k*)

(*k*) !

For item i:

*i*

*ei* = *σ*

*W*1*ei* +

√*|Ni|*√*|N*

(*W*1*eu* + *W*2(*eu ⊙ ei* ))

*|*

*eu* = *σ*

*W*1*eu* +

*i∈Nu*

√*|Nu*

*|*√*|N |* (*W*1*ei* + *W*2(*ei ⊙ eu* ))

(*k*+1)

(*k*) Σ 1

*u∈Ni*

(*k*)

*u*

(*k*)

(*k*) !

Here, **e**(*k*) and **e**(*k*) are the embeddings of user *u* and item *i* at the *k*-th iteration, *N* and *N*

*u i u* *i*

are the sets of neighbors

in the graph for user *u* and item *i* respectively, and *σ* denotes the activation function, specifically LeakyReLU in this context. **W**1 and **W**2 are trainable weight matrices that help in transforming the embeddings and capturing the interaction effects. The element-wise product *⊙* signifies the Hadamard product, aiding in integrating the mutual influence between a user and an item.

## Class Definition and Initialization

class NGCFConv(MessagePassing): def init (self, latent\_dim,

dropout, bias=True, \*\*kwargs):

super(NGCFConv, self). init (aggr=’add’, \*\*kwargs) self.dropout = dropout

self.lin\_1 = nn.Linear(latent\_dim, latent\_dim, bias=bias) self.lin\_2 = nn.Linear(latent\_dim, latent\_dim, bias=bias) self.init\_parameters()

NGCFConv extends MessagePassing, allowing for the implementation of the message passing scheme.

self.lin\_1 and self.lin\_2 correspond to the weight matrices **W**1 and **W**2 in the equations.

## Parameter Initialization

def init\_parameters(self): nn.init.xavier\_uniform\_(self.lin\_1.weight) nn.init.xavier\_uniform\_(self.lin\_2.weight)

Weights are initialized using the Xavier uniform method, which helps in the stabilization of gradients during training.

## Forward Propagation

def forward(self, x, edge\_index): from\_, to\_ = edge\_index

deg = degree(to\_, x.size(0), dtype=x.dtype) deg\_inv\_sqrt = deg.pow(-0.5) deg\_inv\_sqrt[deg\_inv\_sqrt == float(’inf’)] = 0 norm = deg\_inv\_sqrt[from\_] \* deg\_inv\_sqrt[to\_]

out = self.propagate(edge\_index, x=(x, x), norm=norm) out += self.lin\_1(x)

out = F.dropout(out, self.dropout, self.training) return F.leaky\_relu(out)

Normalization coefficients are calculated to weight the influence of each node based on its connectivity. propagate

method handles the passing of messages between nodes, integrating neighbor embeddings.

## Message Computation

def message(self, x\_j, x\_i, norm):

return norm.view(-1, 1) \* (self.lin\_1(x\_j) + self.lin\_2(x\_j \* x\_i))

Messages for each edge are computed using the normalization coefficients and transformations through **W**1 and **W**2. The Hadamard product **x***j ⊙* **x***i* represents interactions between the connected nodes.

## LightGCN

The Light Graph Convolutional Network (LightGCN) simplifies the Neural Graph Collaborative Filtering (NGCF) approach by eliminating weight matrices and non-linear activation functions. This makes it a streamlined model that focuses solely on neighborhood aggregation for generating user and item embeddings in recommendation systems.

(*k*+1)

√*|N*

*|*√*|N | ei ei* =

Σ 1

*i∈Nu*

*u*

*i*

(*k*)

(*k*+1)

Σ 1

*u∈Ni*

*u*

(*k*)

√*|Ni|*√*|N*

(1)

*Nu*: the set of all neighbors of user *u* (items liked by *u*) *Ni*: the set of all neighbors of item *i* (users who liked *i*) *e*(*k*) : k-th layer user embedding

*u*

*eu* =

*eu*

*|*

*e*(*k*) : k-th layer item embedding

*i*

## Class Definition and Initialization

class LightGCNConv(MessagePassing): def init (self, \*\*kwargs):

super(). init (aggr=’add’)

LightGCNConv uses the MessagePassing base class from PyTorch’s geometric library, specifying an aggregation type of add. This indicates that messages from different nodes will be summed up.

## Forward Propagation

This function calculates normalization coefficients for each edge, which are crucial for preventing scale issues due to varying node degrees. It uses these coefficients to normalize the messages during the propagation phase. The propagation leverages the normalized adjacency matrix as the message passing pathway.

## Message Computation

def message(self, x\_j, norm): return norm.view(-1, 1) \* x\_j

Messages are computed as the product of the neighbor’s embedding and the normalization coefficient. This aligns with the mathematical formulation, ensuring each node’s influence is appropriately scaled.

LightGCN’s design directly impacts the system’s performance by simplifying computations and potentially enhancing interpretability. By focusing solely on the structural information from the graph via neighbor aggregation, LightGCN can effectively capture the collaborative filtering effect, making it particularly suitable for large-scale recommendation systems where efficiency is crucial.

## Recommender System GNN

The ‘RecSysGNN‘ class is a comprehensive framework tailored for building recommender systems using Graph Neural Networks (GNNs). This Python class is designed to accommodate both the NGCF and LightGCN models by stacking their convolutional layers. Below, we delve into the architecture, functionality, and specific components of the RecSysGNN class, explaining the purpose and operation of each part.

## Class Structure and Initialization

class RecSysGNN(nn.Module):

def init (self, latent\_dim,

num\_layers,

num\_users,

num\_items,

model, dropout=0.1):

super(RecSysGNN, self). init ()

assert (model == ’NGCF’ or model == ’LightGCN’), ’Model must be NGCF or LightGCN’ self.model = model

self.embedding = nn.Embedding(num\_users + num\_items, latent\_dim) if self.model == ’NGCF’:

self.convs = nn.ModuleList(NGCFConv(latent\_dim, dropout=dropout) for \_ in range(num\_layers))

else:

self.convs = nn.ModuleList(LightGCNConv() for \_ in range(num\_layers)) self.init\_parameters()

The RecSysGNN class inherits from nn.Module and initializes embeddings for users and items combined. The model allows selection between ’NGCF’ and ’LightGCN’.

For NGCF, dropout is an additional parameter that provides regularization.

The convolutional layers (convs) are set up based on the chosen model. The use of nn.ModuleList allows stacking multiple layers, which enables the diffusion of information across multiple hops in the graph.

## Parameter Initialization

def init\_parameters(self): if self.model == ’NGCF’:

nn.init.xavier\_uniform\_(self.embedding.weight, gain=1) else:

nn.init.normal\_(self.embedding.weight, std=0.1)

Different initialization strategies are used for the embeddings based on the model. NGCF uses Xavier uniform initialization to maintain variance across layers, while LightGCN uses normal initialization, recommended for its simplicity and performance.

## Forward Propagation

def forward(self, edge\_index): emb0 = self.embedding.weight embs = [emb0]

emb = emb0

for conv in self.convs:

emb = conv(x=emb, edge\_index=edge\_index) embs.append(emb)

out = (torch.cat(embs, dim=-1) if self.model == ’NGCF’ else torch.mean(torch.stack(embs, dim=0), dim=0))

return emb0, out

The forward method propagates embeddings across all nodes in the graph simultaneously, considering the entire n-hop neighborhood.

The embeddings are updated layer by layer. The final output can be either a concatenation (NGCF) or an average (LightGCN) of all layer embeddings. This difference reflects the distinct propagation styles of the two models.

## Minibatch Embedding Encoding

def encode\_minibatch(self, users, pos\_items, neg\_items, edge\_index): emb0, out = self(edge\_index)

return (out[users], out[pos\_items], out[neg\_items], emb0[users], emb0[pos\_items], emb0[neg\_items])

This function retrieves specific embeddings for users, positive items, and negative items from a minibatch, essential for training sessions that involve personalized recommendations or triplet loss calculations.

The RecSysGNN class is engineered to handle the dynamics of graph-based recommendation systems effectively, leveraging the strengths of GNNs to model user-item interactions comprehensively. This approach allows the model to capture complex patterns and relationships within the data, potentially leading to more accurate and personalized recommendations.

## Loss function and metrics

The ‘RecSysGNN‘ class, designed for recommendation systems using GNNs like NGCF and LightGCN, employs a Bayesian Personalized Ranking (BPR) loss function to enhance recommendation accuracy by favoring positive item

interactions over negative ones for each user. Additionally, the system uses evaluation metrics such as precision@K and recall@K to measure its effectiveness. Here, I’ll explain the implementation of the BPR loss function and the evaluation metrics in more detail.

Bayesian Personalized Ranking (BPR) Loss The BPR loss is a pairwise ranking loss, which is widely used in recommendation systems to ensure that the predicted scores of items that a user interacts with (positive samples) are higher than those of items the user has not interacted with (negative samples). The mathematical formulation is given by:

*LBP R* = *−* Σ Σ Σ ln *σ*(*y*ˆ*ui − y*ˆ*uj*) + *λ||E*(0)*||*2 (2)

*M*

*u*=1 *i∈Nu j∈/Nu*

*y*ˆ*u*: predicted score of a positive sample

*y*ˆ*uj*: predicted score of a negative sample

*λ*: hyperparameter which controls the L2 regularization strength

def compute\_bpr\_loss(users, users\_emb, pos\_emb, neg\_emb, user\_emb0, pos\_emb0, neg\_emb0): # Regularization term

reg\_loss = (1 / 2) \* ( user\_emb0.norm().pow(2) + pos\_emb0.norm().pow(2) + neg\_emb0.norm().pow(2)

) / float(len(users))

# BPR loss calculation

pos\_scores = torch.mul(users\_emb, pos\_emb).sum(dim=1) neg\_scores = torch.mul(users\_emb, neg\_emb).sum(dim=1) bpr\_loss = torch.mean(F.softplus(neg\_scores - pos\_scores))

return bpr\_loss + reg\_loss

Regularization Loss: Computed as the half norm squared of the initial embeddings, scaled by the number of users to avoid overfitting.

BPR Loss: Uses the ‘softplus‘ function to smoothly penalize instances where negative samples score higher than positive ones.

**Evaluation Metrics (Precision@K and Recall@K)** To assess the model’s performance, precision and recall are calculated for the top *K* recommendations made by the system.

Recall = *TP TP* + *FP*

*TP*

; Precision =

*TP* + *FN*

(3)

Precision@K measures the proportion of recommended items in the top *K* set that are relevant. Recall@K measures the proportion of relevant items that are captured by the top *K* recommendations.

**Relevance Score Calculation:** A matrix multiplication between user and item embeddings to calculate scores for all possible interactions.

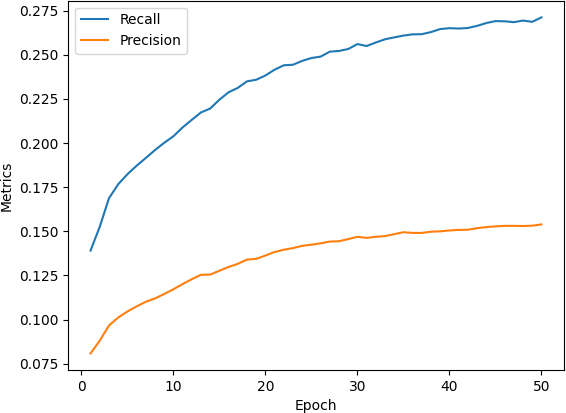
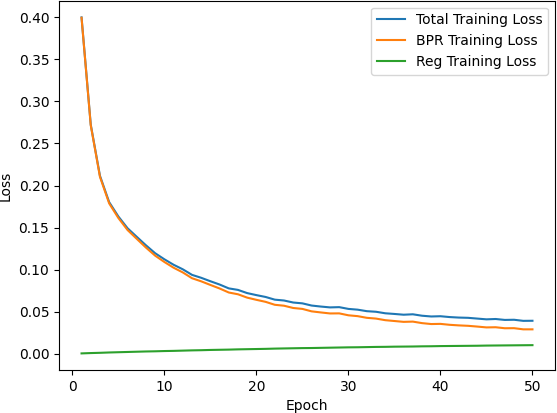
**Interaction Masking:** Masks the scores for interactions already present in the training set to focus only on potential new recommendations.

**Metric Calculation:** Determines how many of the top *K* recommendations are actual interactions in the test set.

Together, these methods provide a robust mechanism for training and evaluating recommendation models, ensuring that the recommendations are both accurate and relevant.

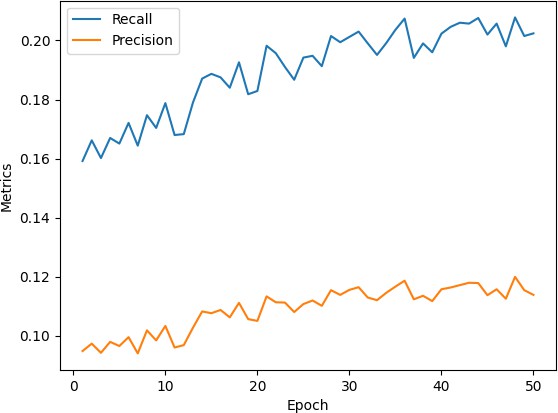
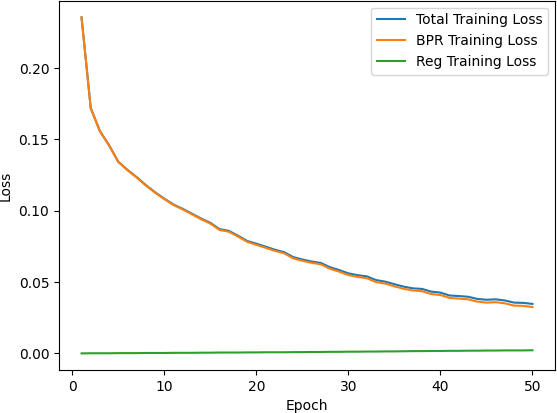
# Conclusion

Based on the results, we observe distinct differences in the performance of NGCF and LightGCN models. Despite both models being trained using the same minibatching procedure on an 80/20 train-test split, LightGCN exhibits smoother precision@20 and recall@20 curves compared to NGCF.



(a) Loss (b) Recall

Figure 1: LightGNN Loss and Recall



(a) NGCF Loss (b) NGCF Recall

Figure 2: NGCF Loss and Recall

Notably, it is evident that the LightGCN model consistently outperforms the NGCF model in terms of both precision and recall throughout the training epochs.

The LightGCN model achieves a maximum precision of 0.1539 and a maximum recall of 0.2711. The recall graph for LightGCN shows a significant upward trend, suggesting an improving ability to identify all relevant items as training progresses. Precision also increases over time, albeit at a slower rate, indicating an enhancement in the accuracy of the predictions. These trends highlight that LightGCN is effectively learning to rank relevant items higher while simultaneously increasing the coverage of relevant items identified.

In contrast, the NGCF model reaches lower maximum values, with a precision of 0.12 and a recall of 0.2078. Both precision and recall for NGCF increase as training continues, with recall exhibiting a relatively steeper ascent. However, these metrics remain notably lower compared to those of the LightGCN model. This suggests that while NGCF is improving in identifying and accurately predicting relevant items, it is less effective than LightGCN in this regard.

The analysis of the loss graphs shows a rapid decline in total training loss for both models, indicating that each model is learning effectively from the training data. The Bayesian Personalized Ranking (BPR) loss, which focuses on the order of recommendations, shows a significant decrease, reinforcing that both models are enhancing their ranking capabilities

as training progresses. Additionally, the regularization loss remains consistently low, which is indicative of effective control over model complexity and avoidance of overfitting.

These observations suggest that LightGCN may have a better mechanism for capturing and utilizing the graph structure of the data compared to NGCF. This leads to superior performance in terms of both identifying the correct items (recall) and in the accuracy of these identifications (precision). LightGCN’s more effective learning and prediction capabilities make it a preferable choice for tasks where higher precision and recall are crucial.

# Future Work

In the final project, we outline a comprehensive approach to building a book recommender system utilizing advanced techniques such as Graph Neural Networks (GNNs). Our system currently incorporates models like LightGCN and NGCF, which excel in capturing user-item interactions for recommendation purposes. Looking towards future enhancements, we aim to integrate the Knowledge Graph Attention Network (KGAT) architecture. KGAT offers a sophisticated method to leverage side information encoded as knowledge graphs, enabling more nuanced, diverse, and explainable recommendations. By incorporating entity features such as directors, cast, and genre, our system can better profile items, enhancing recommendation quality. Additionally, we recognize the significance of considering seasonality and session-based interactions for refined recommendations. Strategies such as training on recent transactions and weighting recent interactions more heavily based on the time elapsed can ensure our system recommends relevant items tailored to current trends. Furthermore, exploring session-based recommender systems like SR-GNN and TAGNN could empower our system to infer user interests from short interaction patterns, improving engagement with new customers. By integrating these advancements, we anticipate significant enhancements in the performance and user experience of our book recommender system.

# References

1. Harper, F. Maxwell, and Konstan, Joseph A. “The MovieLensDatasets: History and Context.” ACM Transac- tions on Interactive Intelligence Systems (TiiS) 5, 4. 2015. [https://colab.research.google.com/drive/](https://colab.research.google.com/drive/1VQTBxJuty7aLMepjEYE-d7E9kjo51CA1?usp=sharing) [1VQTBxJuty7aLMepjEYE-d7E9kjo51CA1?usp=sharing](https://colab.research.google.com/drive/1VQTBxJuty7aLMepjEYE-d7E9kjo51CA1?usp=sharing)
2. He, Xiangnan, et al. “LightGCN: Simplifying and powering graph convolution network for recommendation.” Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020. <https://arxiv.org/abs/2002.02126>
3. Wang, Xiang, et al. “Neural graph collaborative filtering.” Proceedings of the 42nd International ACM SIGIR Con- ference on Research and Development in Information Retrieval. 2019.<https://arxiv.org/abs/1905.08108>
4. Goodreads Datasets<https://github.com/MengtingWan/goodreads?tab=readme-ov-file>
5. Kaggle Code <https://www.kaggle.com/code/dipanjandas96/lightgcn-pytorch-from-scratch>
6. NGCF Code <https://github.com/SytzeAndr/NGCF_RP32/blob/hand-in/NGCF.ipynb>
7. Book Recommendations [https://medium.com/stanford-cs224w/using-pyg-to-get-book-recommendations-a7a65ddb](https://medium.com/stanford-cs224w/using-pyg-to-get-book-recommendations-a7a65ddb0e12)
8. BPR: Bayesian Personalized Ranking from Implicit Feedback <https://arxiv.org/pdf/1205.2618>