GRAPH NEURAL NETWORK APPLICATION: BOOK RECOMMENDATION SYSTEM

Abhinav Bajpai (abbajpai@iu.edu)

Department of Data Science
Indiana University - Bloomington

ABSTRACT

With the increased ability to process large datasets through distributed computing, Network science has emerged to provide solutions to some of the existing issues in different domains. By decoding the interactions between the components of the network, the collective behavior of a system can be studied and visualized in greater depth. Bioscience (genes and proteins), social networks (friends and family), communication networks, power grids, trade networks, and other fields have utilized Network science methodologies with great success and successfully discovered new patterns and knowledge. Our study investigates the interaction between users and products using Graph Neural Networks in order to learn users' usage behavior and suggest products to buy. Our experiments were conducted using the Amazon book open dataset between 2016 and 2017. Both human-centric and metric-based evaluation methods were applied to evaluate two different approaches based on a Unipartite and Bipartite graph.

Index Terms— *Recommendation system, Graph Neural Network, Collaborative Filtering*

1. INTRODUCTION

The recommendation systems have become ubiquitous and help businesses reduce customer information overload. Imagine a customer looking through hundreds of movie titles to find the one they want to watch on streaming platforms like Netflix, Amazon Prime, etc. The tedious task of searching could drive them away from the platform to some other form of entertainment that is less time-consuming and tiresome. The recommendation system in this scenario can suggest movies based on previous movie-watching history, likes, dislikes, movie ratings, and similar viewing patterns.

Similarly, customers on shopping platforms (e.g., Amazon, BestBuy, etc.) may choose from thousands of products and compare them with similar products to get a better price, specification, or quality. A customer may have to spend hours searching the portal for a similar product if it is a manual process. It is still not guaranteed that he/she will find the best possible match after hours of searching. Using product features, price, ratings, and customer behavior, a recommendation system can provide suggestions to customers. As a result, customers save time and may get a better product. By improving customer satisfaction, shopping portals can also boost sales and increase customer loyalty.

We have explored two graph-based approaches to build a recommendation system for this project. In our first approach, we convert our user-book bipartite graph to a unipartite user graph through projection and then use a graph convolutional network to generate embeddings for each user. We then use the Euclidean distance measure on these embeddings to find each user's nearest neighbor and their top-rated books. We use these books as a recommendation for the current user. In our second approach, we see the recommendation problem as a missing link prediction problem on a graph. We don't create a projection graph but instead, use the original bipartite graph of the user-book to train a Graph Neural Network and predict the missing links between books and users. The details of both approaches are discussed in Section 4.

2. RELATED WORK

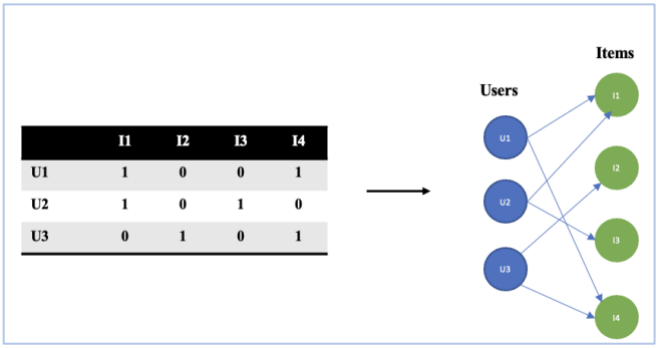
One of the most successful techniques in the past few decades for recommendation systems has been Collaborative filtering (CF). (Goldberg et al., 1992) coined the term "Collaborative Filtering" in 1992 while developing an email filtering system at the Xerox Palo Alto Research Center. They used the word "Collaborative" because they designed a system where users collaborated to record their reactions to documents they read as annotations. Later, when a user searches for a document in the database using keywords, documents that receive annotation endorsements (other users have found the keyword in the past) appear higher in the search results. Today's rating system on shopping platforms can be described as a variation of this system, in which shoppers decide what to buy based on the experiences of other users (collective ratings). Simply, people collaborate to help each other make better buying decisions.

The concept of collaborative filtering was further improved by (Konstan et al., 1997) who used it to filter news articles users would like to read. Rating news articles from 1 to 5 was introduced, followed by a matrix-filling exercise where people were represented as columns, rows as news, and cells as ratings. Their algorithm then predicted the missing ratings in the matrix, which meant how much the user will like the news articles that he/she has not read yet. This algorithm generated personalized recommendations by calculating user similarity using Pearson correlation. It was an evolution of the neighborhood-based recommendation method that had been popular for some time.

The next big development was by (Linden et al., 2003) who introduced a collaborative filtering algorithm based on an item similarity table that kept track of the most popular and correlated items based on the user's previous purchases and ratings. Results from this table were used to make product recommendations. This was the first major switch to item-item collaborative filtering.

Over the past decade, graphs have been used increasingly to characterize interactions between objects of interest, to model simple and complex networks, or to represent problems in general. Graphs are data structures containing vertices (or nodes/vertices) and edges (or relations/lines) representing relationships between objects. Graph-based recommendation systems use a special representation of the graph - a bipartite graph, where a relationship occurs only between nodes in one subset and nodes in the other subset; no relationships exist between nodes in the same set. This figure is an example of a bipartite graph where all users are on the left and all items are on the right.

Figure 1: A Bipartite Representation of User/Items



Among the first researchers to demonstrate the use of graph representation in recommendation systems were (Li & Chen, 2013). Using a graph kernel, the researchers developed an approach that examined customers and items related to the focal user-item pair as its context to determine if a link exists. The network kernel generated random walk paths based on user-item pairs' intersections and then used this information to define the similarity between the user-item pairs. Their work was broadened by (Berg et al., 2017) who also studied matrix completion for recommendation systems from the point of view of link prediction on graphs. Their recommendation algorithm introduced a variant of graph autoencoders. A graph encoder model Z = f(X, A), takes N ×D feature matrix X and a graph adjacency matrix A as input, and produces a N×E node embedding matrix Z = $[z_1^T,\ldots\ldots,z_N^T]^T$. It has a pairwise decoder model $\grave{A} = g(Z)$, which takes pairs of node embeddings (z_i, z_j) and predicts respective entries $\grave{A}_{ij}$ in the adjacency matrix. Taking the properties of the bipartite graph, they modified the decoder function to return a rating matrix in their approach.

GCN (Graph Convolutional Network) accelerated the innovation in using graphs to recommend products. With the use of multiple layers of graph convolution in embedding propagation, GCN effectively captures collaborative signals of high-hop neighbors, improving the problem of data sparsity in collaborative filtering in some way. The LightGCN approach introduced by (He et al., 2020), is the latest state-of-the-art GCN approach, where users and item embeddings are learned by linearly propagating them through the user-item interaction graph, and the final embedding is calculated as the weighted sum of all embedding. Compared with the Neural Graph Collaborative Filtering (NGCF) model (Wang et al., 2019) which has been an industry standard for quite some time, their approach showed a 16% improvement.

3. DATA

3.1 Data Description

The user/item dataset we will utilize for analysis is a subset of Amazon Review Data compiled by (Ni et al., 2019) during their research study. As our study aims to develop a recommendation system for books, we will only use a subset of this dataset. For analysis, two types of datasets are downloaded and merged.

1. **Books Review** – Fifty-one million reviews about books also provide star ratings, review time, reviewer IDs, and the review text. We will subset a specific time of these fifty-one million reviews as processing this amount of data will need additional computing infrastructure support.

2. **Books Metadata –** This dataset contains information about books in the review data, including book descriptions, categories, similar items, rank, prices, and brands.

Both datasets identify products using the Amazon Standard Identification Number ("ASIN") field, which is used to combine the dataset.

Source: Amazon review data (ucsd.edu)

4. METHOD

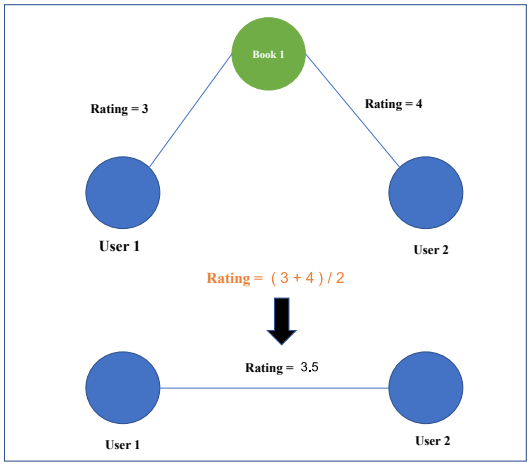
The project aimed to explore Graph Neural networks and available approaches to solving real business problems. To achieve this, we researched the latest Graph-based approach and decided to experiment with below mentioned two approaches:

- Graph Neural Network (Unipartite)
- Graph Neural Network (Bipartite)

For both approaches, we have relied heavily on the NetworkX python library for converting the tabular dataset to a network and the PyTorch Geometric (PyG) python library (an extension of PyTorch for Graph Neural Networks) for training the model.

Graph Neural Network (Unipartite) - We have already discussed and showcased in figure 1 how user-item interaction can be represented through a bipartite graph. This bipartite graph can be further converted into a unipartite graph through projection such as shown in figure 2 below

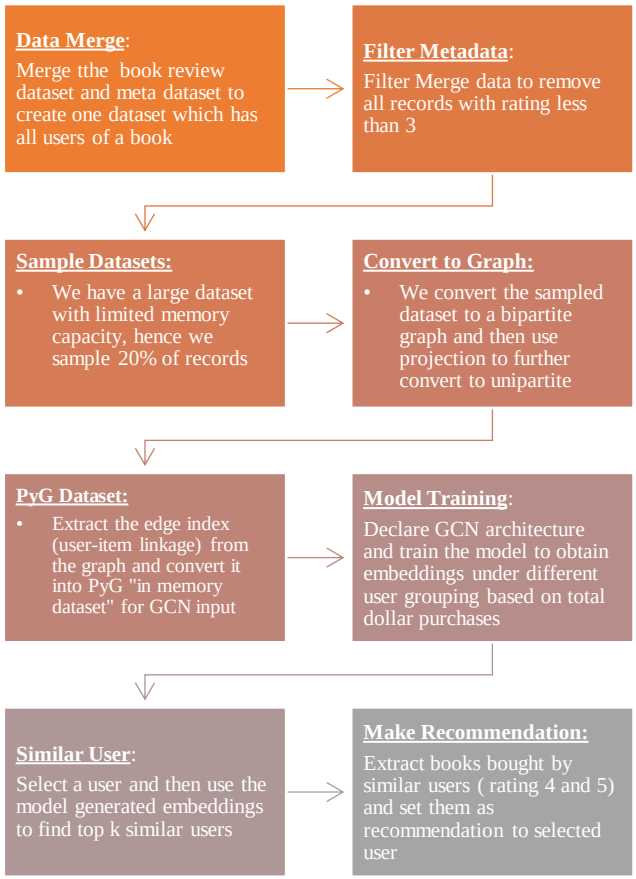
Figure 2: Conversion of Bipartite Network to Unipartite Network



To fit the data, embeddings must capture similarities between users or items. The conversion to a unipartite graph allows us to generate only one node-type embedding that can be easily used to find the nearest neighbors, an approach like generating the "most_similar" nodes we have studied in BERT and Word2Vec embedding models in the classroom.

In recent Graph Neural Network literature, conversion of a user-item graph to a unipartite network and generating embeddings are not discussed. Nevertheless, we decided to combine this approach with the current industry standard practice approach of missing link prediction in the spirit of academic experimentation.

Below we provide the steps we implemented in our first approach:



In our approach, we use graph convolutional networks (GCNs); convolutional since the parameters of the filters are shared across all graph locations. The model learns a function of signals/features on a graph G = (V, E) from the following inputs:

1. Feature description $\boldsymbol{x_i}$ for every node ***i***; summarized in an N×D feature matrix X (N: number of nodes, D: number of input features)

2. A description of graph structure as a matrix e.g.- an adjacency matrix A or edge index

and produces a node-level output Z (an N×F feature matrix, where F is the number of output features per node). Every neural network layer can then be written as a non-linear function

$$H^{(l+1)} = f\,(\,H^{(l)}, A),$$

with H(0) = X and H(L) = Z where L is the number of layers. Specific models differ only in how f(·,·) is chosen and parameterized.

For model training, we implemented the following Graph convolutional structure and used **user node degree** (number of connections) as the feature vector and **edge index** for graph structure input.

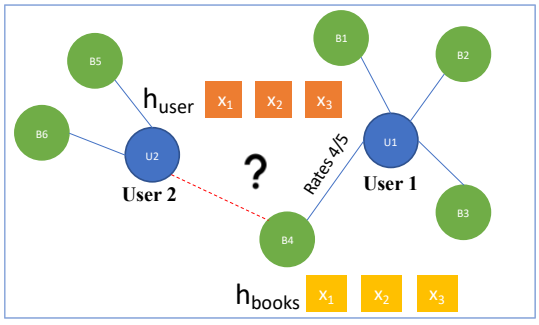
Figure 3: Graph Convolution Network Architecture – (Unipartite)

```
GCN(
  (conv1): GCNConv(1, 4)
  (conv2): GCNConv(4, 4)
  (conv3): GCNConv(4, 2)
  (classifier): Linear(in_features=2, out_features=4, bias=True)
)
```

Other features of nodes were also tested, such as the gender of users (extracted from their names), book category, and node degree. On evaluation, however, we found this model's performance unsatisfactory and discarded it. We also tried to build a book-to-book similarity model, but couldn't come up with a satisfactory evaluation metric, so its results are also not included in this work.

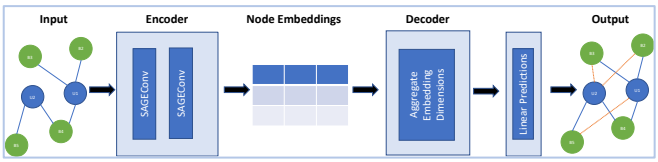
Graph Neural Network (Bipartite) - In the second approach, the product recommendation is structured as a link prediction problem using graphs. In the figure below, we have given a problem statement stating that User 1 likes books B1, B2, B3, and B4. Based on the graph embeddings generated by the GNN model, what is the rating User 2 will give to book B4 if User 1 and User 2 are similar?

Figure 4: Link Prediction - Bipartite Graph



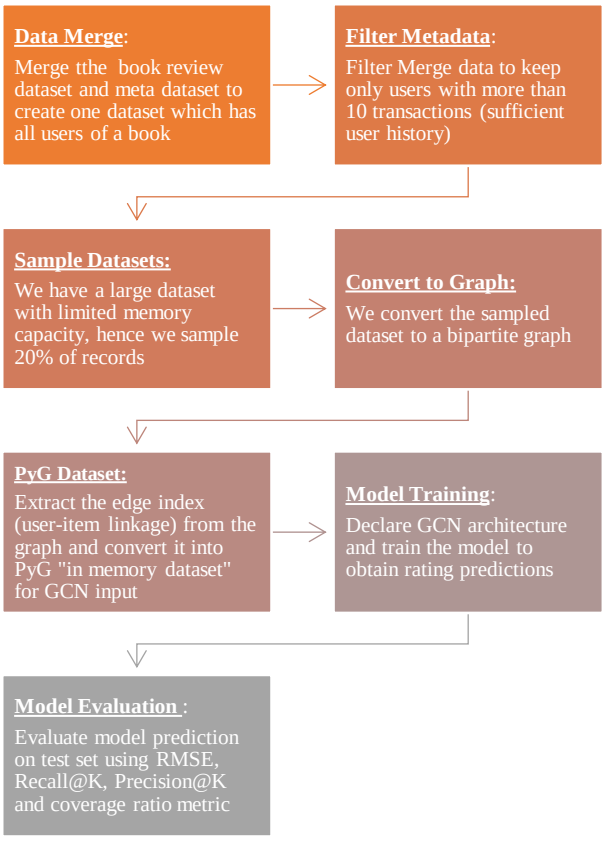
Our first approach uses projections to generate a graph only for users, while our bipartite approach generates embeddings for both users and books. After both types of embeddings have been generated, they can be concatenated and passed as input into a linear model that produces the final book ratings. The below image showcases the network architecture we deployed to implement the GNN Bipartite approach:

Figure 5: Graph SAGE Convolution Architecture – (Bipartite)



To overcome the challenges of training large graphs and managing unseen nodes, the concept of SAmpling and AggreGatE (SAGE) Convolution was introduced by (Hamilton et al., 2017). In SAGE convolution, the embeddings are generated by sampling and aggregating the features of the local neighborhood of a node instead of individual embeddings of the node.

Below we provide the steps we implemented in our second approach:



The next section discusses the evaluations results from both approaches.

5. RESULTS

In both models, the initial dataset had more than a million records, which were then sampled and trained on the IU HPC Carbonate Large memory cluster. Due to the cluster's memory size limitation of 503GB, extending beyond 20% of the sample size resulted in memory out failure as edges in the graph grow exponentially as we add more user and book nodes. Model training using a distributed memory approach was beyond the scope of this project. While mini-batching for unipartite (homogeneous) graphs is available in PyG, it is still not feasible to do min-batching on bipartite (heterogeneous) graphs.

To test a recommendation system and tune its hyperparameters for personalized recommendations, A/B testing is the industry standard. It requires extensive hypothesis testing and experimentation by adding variations to the hypothesis being tested and re-experimenting. Due to limited resources, we chose to explore alternative human and metric-based testing approaches for our project. Since we have two model implementations, we have evaluated our first approach by manually reviewing the recommendations and observing if they are relevant. In our second model implementation, we analyzed model performance using standard metrics.

Graph Neural Network (Unipartite)

The sampled user graphs created after projection have 321603 user nodes and 7310186 edges. This dataset is split in a 70/30 ratio as a train and test dataset for model training and evaluation.

Table 1: Unipartite GNN Model - Training Set

Dataset	User Nodes	Number of Edges
Train	225122	5117130
Test	96480	2193066

The recommendations in this implementation are based on books bought by similar users. Similar users are suggested by the distance between the embeddings we generated by training the GNN model on the projection graph. The projection graph was built by connecting users who bought the same book so it is our understanding that users who bought the same book or a similar category book should be closer to each other in the vector space. To test this assumption, we randomly reviewed some of the recommendations generated by the model. Below are a few examples from different book categories:

Table 2: Unipartite GNN Model - Manual Evaluation Example 1

Example: 1 **Reviewer ID:** A1RDP8H7WC7EDK **Book Bought:** Protecting Dakota (SEAL of Protection) **Book Category:** Contemporary		
Recommendations		
Category	**ASIN**	**Book Name**
Contemporary	990738884	**Protecting the Future (SEAL of Protection) (Volume 8)**
Contemporary	996572082	The Nanny Crisis
Contemporary	1477849734	No Ordinary Billionaire (The Sinclair's)
Genre Fiction	1623221099	Delicate Ink (Montgomery Ink)
Mystery	996683054	The Pretty Ones (A Kate Reid Novel) (Volume 6)
Contemporary	998959952	The Lake: A Berry Springs Novel

In the above example, the recommendations have a high overlap with the 'Contemporary' book category that the user bought. We decided to compare this with the recommendations Amazon would make for this book and found that our first recommendation "**Protecting the Future (SEAL of Protection) (Volume 8)" is** actually a frequently bought-together book on amazon.

Protecting Dakota (SEAL of Protection): Stoker, Susan: 9781943562305: Amazon.com: Books

Table 3: Unipartite GNN Model - Manual Evaluation Example 2

Example: 2 **Reviewer ID:** A1D176OJB2GKMW **Book Bought:** Forks Over Knives – The Cookbook **Book Category:** Cooking by Ingredient		
Recommendations		
Category	**ASIN**	**Book Name**
Baking	125002658X	Mug Cakes: 100 Speedy Microwave Treats to Satisfy Your Sweet Tooth
Kitchen Appliances	985124822	O M Gee Good! Instant Pot Meals, Plant-Based & Oil-free
Herbal Remedies	1603420789	Rosemary Gladstar's Herbal Recipes for Vibrant Health

Two useful recommendations were provided to the user in our second example: one on a kitchen appliance to help prepare plant-based recipes, and another one on different herbal recipes. More such examples are provided for review in the Excel file on our GitHub repository (src/2. Unipartite (User Similarity Model)/Recommendation Example.xlsx).

The above approach to reviewing individual recommendations, though is encouraging as we find relevant results, it is time-consuming and does not provide an overall sense of the model performance unless we follow a standard A/B testing approach. We, therefore, decided to look at additional recommendation system evaluation metrics in our second approach.

Graph Neural Network (Bipartite)

The sampled user-book bipartite graph has 21301 user nodes, 178224 book nodes, and 564521 edges. This dataset is split in an 80/10/10 ratio as train, validation, and test dataset for model training and evaluation. For the bipartite link prediction model, the split is done using edges and not user/book nodes.

Table 4: Bipartite GNN Model - Training, Validation, and Test Set

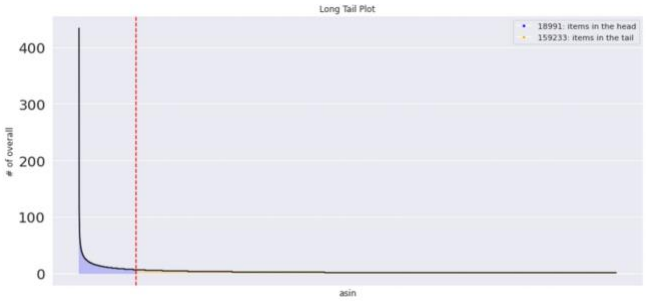
Dataset Type	Number of Edges
Train (80%)	451617
Validation (10%)	56452
Test (10%)	56452

The following graph shows the distribution of books in our training dataset. It is evident that users purchase a few books frequently, while others are less frequently purchased. As a result of this distribution, the graph adjacency matrix will be sparse, which means a lot of user nodes will be connected to few book nodes frequently purchased. It will reduce the coverage of books with a minimal transaction history as recommendation items. It is also likely that this "cold start" problem will affect model performance and the evaluation parameters.

Table 5: Top 10 Books by Purchase Count

Book ID (ASIN #)	Category	Purchase Count
1683247353	Genre Fiction	434
1503943372	Genre Fiction	422
038568231X	Mystery	298
312577222	Literature & Fiction	276
1503935310	Genre Fiction	270
1410472922	Humor & Satire	245
1503934713	Thrillers & Suspense	239
1477848665	Thrillers & Suspense	235
1910751774	Thrillers & Suspense	228
385541198	Thrillers & Suspense	202

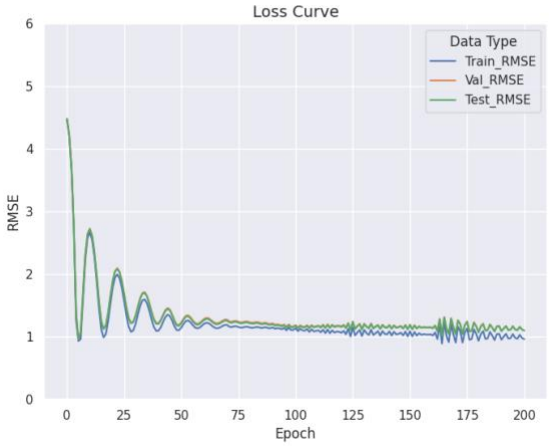
Figure 6: Book Distribution



To evaluate the results of our second implementation we used the following metrics:

Root Mean Squared Error (RMSE) – This measure determines whether the ratings given by recommender systems are close to those given by users. After training the model for 200 epochs, the train RMSE is .96, while the validation RMSE and Test RMSE are slightly above 1. Based on this, the model appears to be performing reasonably well at predicting user ratings.

Figure 7: RMSE Performance



Precision and Recall

Precision and recall are the measures used in measuring classification performance.

Precision – measures the proportion of positively predicted labels that are correct. It is calculated by dividing correctly

predicted positive examples by the number of positive examples that were predicted.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Recall – The recall metric measures how many correct positive predictions were made out of all possible positive predictions. As opposed to precision, which only identifies the correct positive predictions out of all positive predictions, recall identifies the missed positive predictions. As a result, recall provides some information about how well a positive class has been covered.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

As part of a recommendation system, we can categorize recommendations based on their actual and predicted ratings. For this purpose, we initially classify a book as relevant or irrelevant by using a threshold rating of 4. The book is relevant if the actual rating is greater than or equal to 4, otherwise, it is irrelevant. In the same way, we make recommendations based on predicted ratings. It is recommended if the predicted rating is greater than or equal to 4, not recommended if it is less than 4.

Table 6: Product Classification - Relevant and Recommended Example

Book	Actual Rating	Predicted Rating	Classification
A	5	4	Relevant and recommended
B	4	3	Relevant but not recommended
C	3	4	Not relevant but recommended
D	3	3	Not relevant and not recommended

Based on the above classification we can create the confusion matrix as shown below:

Table 7: Confusion Matrix Structure - Recommendation System

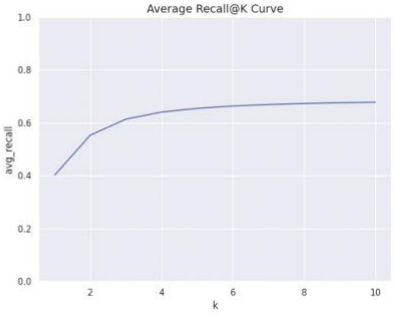
	Relevant	Not Relevant
Recommended Top K	True Positive (TP)	False Positive (FP)
Not Recommended	False Negative (FN)	True Negative (TN)

Recall @K – Given that we can now classify the True/False positives and negatives, the Recall @K recommendation can be calculated as

$$Recall = \frac{\#\text{ of top k recommendations that are relevant}}{\#\text{ of all relevant items}}$$

Figure 8 : Recall @K Curve

K	Recall @K
1	40.22%
2	55.36%
3	61.40%
4	64.08%
5	65.49%
6	66.36%
7	66.94%
8	67.32%
9	67.59%
10	67.81%



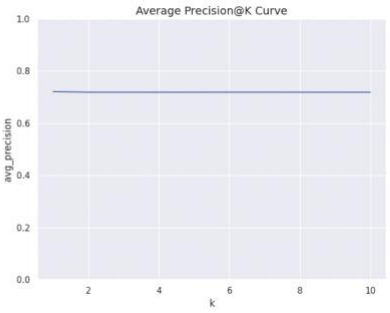
The top-k results contain nearly 68% of the total number of relevant items.

Precision @K – Similarly, Precision @k is a fraction of the top k recommended items that are relevant to the user

$$Precision = \frac{\#\text{ of top k recommendations that are relevant}}{\#\text{ of items that are recommended}}$$

Figure 9: Precision @K Curve

K	Precision @K
1	72.10%
2	71.90%
3	71.89%
4	71.89%
5	71.90%
6	71.91%
7	71.91%
8	71.90%
9	71.90%
10	71.90%

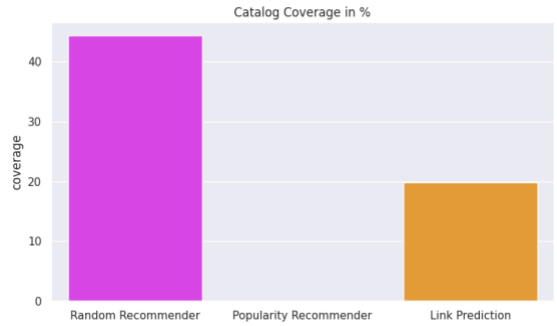


According to the model, 72% of recommendations are relevant to the user.

Both Precision and Recall numbers show that our model is performing moderately well and need additional hyperparameters tuning to improve these results.

Catalog Coverage – According to (Ge et al., 2010), the coverage metric assesses the ability of the recommendation system to suggest all the items in the training set to users. We discussed at the beginning of this section that only a small percentage of books are frequently purchased. Therefore, if the recommendation system recommends only these popular books, the recommendation system will have near zero coverage. When we recommend the top 10 books in the training set to every user, we observe that only popular item recommendation leads to near-zero coverage.

Figure 10: Catalog Coverage



In contrast, the random book suggestion covers **45**% of the books available, while our recommendation system covers only **20**%. This low coverage means users will encounter new products rarely which will be disastrous for business from the seller's perspective and lead to a poor buying experience for the consumer. The model performance based on the catalog coverage ratio is below par and will need additional effort from our end to improve these results.

Personalization – As defined by (Zhou et al., 2008), Personalization represents a measure of diversity that accounts for the uniqueness of recommendations based on different users' preferences, or inter-user diversity. Higher scores indicate that a recommendation system can generate highly personalized recommendations for each user.

Using Cosine similarity, we compared the recommendation lists for users in our testing dataset. Our recommendation system personalization score is **51.81%,** suggesting that each set of recommendations is not very different and there is overlap in recommended items. We believe the root cause of this issue is the bias in the dataset where few books are bought more frequently than others. More research is needed from our end to identify the mitigating techniques we can implement to improve the Personalization score.

For model re-implementation and review the code and underlying data is available on our GitHub and IU SharePoint.

GitHub Link - abbajpai/DLS_Project: Repository to submit Deep Learning System Project Work (iu.edu)

Data and Model Output – The data and model outputs are too large for GitHub, so they have been uploaded to IU SharePoint.

6. CONCLUSION

With this project, we don't claim to have built a state of art architecture model but have wholeheartedly explored a completely novel domain of Graph Neural Networks and applied it to a real-world problem. We had to overcome two challenges - Understanding Graph Neural Networks' current literature, its architecture, data processing, and mode training, as well as how recommendation systems work and how we can integrate these two knowledge sets into a GNN for recommendation system. For Future work, we want to refine the models we have developed to further improve their performance. The next step is to learn how to perform distributed computation on graphs for training large datasets and explore heterogeneous graphs beyond bipartite graphs.

7. ACKNOWLEDGEMENT

Our knowledge base for this project is derived from the work of previous researchers and authors in the open-source community. These researchers and authors have demonstrated methodologies and metrics at great length in order to assist students like us in exploring new topics. We would also like to acknowledge the PyG (PyTorch Geometric) community, and the examples they have shared on their GitHub and community pages that helped us overcome GNN coding challenges. Moreover, examples provided by recmetrics and Surprise, two Python libraries that evaluate recommendation systems, helped us to illustrate how our model is evaluated. This project would not have been possible without the prior work of these researchers and community members.

8. REFERENCES

Berg, R. v. d., Kipf, T. N., & Welling, M. (2017). Graph Convolutional Matrix Completion. In.

Ge, M., Delgado-Battenfeld, C., & Jannach, D. (2010). Beyond accuracy: Evaluating recommender systems by coverage and serendipity.

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, *35*(12), 61–70. https://doi.org/10.1145/138859.138867

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. In.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., & Riedl, J. (1997). GroupLens: applying collaborative filtering to Usenet news. *40*(3), 77. https://dl.acm.org/doi/10.1145/245108.245126

Li, X., & Chen, H. (2013). Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support*

Systems, *54*(2), 880. https://dl.acm.org/doi/10.1016/j.dss.2012.09.019

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, *7*(1), 76-80. https://doi.org/10.1109/MIC.2003.1167344

Ni, J., Li, J., & McAuley, J. (2019). Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. EMNLP,

Wang, X., He, X., Wang, M., Feng, F., & Chua, T.-S. (2019). Neural Graph Collaborative Filtering. In.

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., & Zhang, Y.-C. (2008). Solving the apparent diversity-accuracy dilemma of recommender systems. In.