Incorporating Review Analysis into GNN-Based Book Recommender Systems

Ekam Kaur

May 2024

Introduction

Recommendation systems have become an increasingly popular area due to their ability in assisting people navigate between and choose suitable products and content. The primary goal of recommendation systems is to provide personalized suggestions based on the users' previous interactions – this is of course very useful in the face of choice/information overload and also given the rapid development of e-commerce sites and related platforms. Recommendation systems aim to utilize various pieces of data, such as user profiles, item metadata, and user-item interaction information in order to find underlying patterns and make good quality predictions.

Recently, graph neural networks (GNNs) have emerged as a powerful tool in learning and making predictions about nodes in graph structures. User-product data can be naturally modelled as bipartite graphs (where edges represent the interactions), making them well advisable for GNNs. GNNs are very flexible and can model the highly complex relationships present in user-product graphs, enabling them to make high quality recommendations. Particularly, given the flexibility of neural networks in general, GNNs can model high order relationships between users and products, while also leveraging the structural properties of the graph like node importance and clustering coefficients.

Book Recommender systems in particular are traditionally such that the system uses ratings each user gives to different books, and also information about the books (genre, title, author, etc.) and the user (demographics etc.) in order to recommend other books. In this project, we aim to build a recommender system using GNNs with sentiment analysis of the user reviews as a key component – and see if the sentiment analysis makes a tangible difference in recommendation performance.

Usually, the given number of interactions between the users and products are very few compared to the number of users and products themselves (ie. it is a sparse graph), and so the recommendations are not particularly good ([4]). We aim to find out whether incorporating the user reviews actually makes a noticeable difference in performance. We compare recommender system models which incorporate user reviews to that which just use the numerical rating. In what follows, I will refer to the review models as treatment models and the no review model (just rating) as the base model.

Data Collection

For this project, I used the Books data from this Amazon Reviews Dataset ([3]). The data was collected in 2023 by the McAuley Lab, (building on from previous versions) and contains data for over 54 million users and 48 million different Amazon products. For this project, I extracted just the books subset which contained over 10 million user, 4 million books, and 29 million reviews in total.

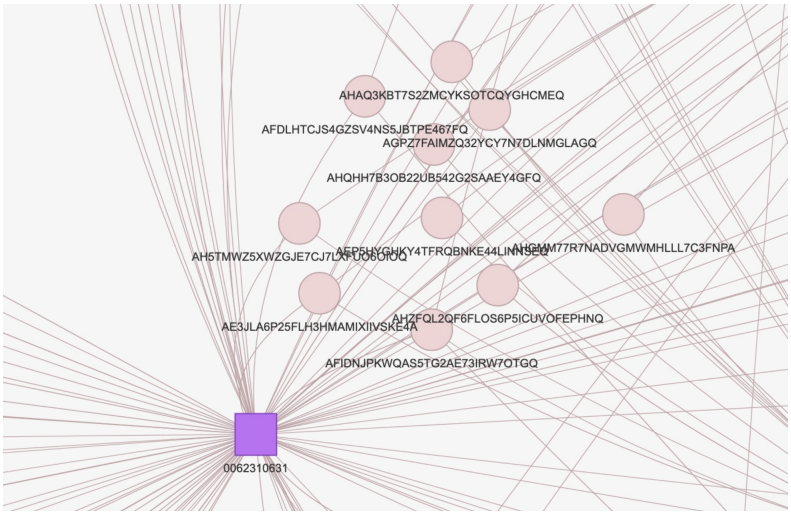

Figure 1: Preview of the user-book graph

I restrict to just the fiction subset in order to consider more specific recommendations and also to make the amount of data manageable. After scrubbing the data, there are about two million users and 50,000 books.

This is obviously way too large for a standard computer's computing power and memory to handle – and we will need to stick to a smaller subset of the data.

We first reduce dataset by limiting to just the books and users with at least 10 reviews. A significant amount of users and products in the previous dataset had very little reviews and this reduction leaves the dataset with about 25,000 users, 14,000 books, and 333, 000 edges. Removing these less prevalent books and users certainly creates bias in the book/user collection, but such a reduction is needed to make the data manageable – and we also want to ensure high prediction quality. Making predictions in a sparse graph is known to be a challenging task ([4]), and so we stick to a relatively dense subset of the original fiction data. It will be necessary to reduce this sample even further in order for me to run the models, and we discuss this in the following sections.

Overview of Graph Neural Networks

In this section, we give a brief overview of the use of graph neural networks and how they work. Graph Neural Networks (GNNs) are a powerful and increasingly popular architecture for learning about graph-structured data. The neural networks and highly flexible and adaptable for capturing complex relationships and can be used for a variety of different tasks.

The main power of GNNs is to pass information along the graph edges, while utilizing the powerful (standard) neural network architecture. Given a graph G, where the nodes represent information via node embeddings, we use the edges to pass along information with a neural network to successively update the node embeddings. The structured way to put this is that for each node, we use its neighbors and apply a matrix transformation and activation, combine it with the previous embedding, to get the updated embedding.This is known as message passing – where the nodes exchange and aggregate along information from their neighbors. Iteratively updating the node embeddings in this way, the GNNs learn representations that encode the underlying structure of the graph.

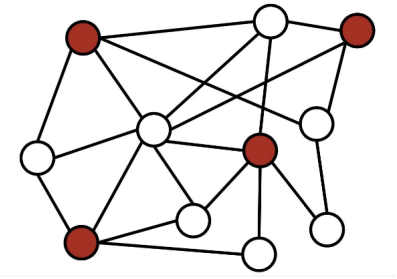

Figure 2: Example of a bipartite graph

In this case, we have a *bipartite* graph, as we only have edges between a given user and a given product (indicating a review). As part of the goal of the project, we wish to learn the "personality" of a user – as represented by updating of the user node embeddings. The product node embeddings, however, represent objective information about the listed books – and don't need updating. However, given the GNN structure and the bipartite nature of the graph, it is necessary update the product node embeddings in order to pass information at long distances and between users.

When we add sentiment analysis to the network, we add edge embeddings. This is an extension of the standard GNN structure which also incorporates the respective edge embedding when passing information along an edge. For this project, we use the 'NNConv' layer in the Pytorch geometric library – created specifically to incorporate edge embeddings ([2]). Mathematically, as given in the documentation, a node embedding x_i is updated as

$$x_i' = \Theta x_i + \sum_{j \in \mathbf{N}(i)} x_j h_\Theta e_{i,j},$$

where h_Θ denotes a learnable neural network.

The 'NNConv' class does not account for bipartite graphs – and hence both the user and products are treated in the same way.

Hence, as this should explain, GNNs are considered superior than traditional recommender systems (like collaborative filtering) in their ability to capture very non-linear relationships, handle sparse graphs, and utilize contextual information.

Data Pre-processing and Preliminary Analysis

As described earlier, the current data set size of 333, 000 reviews from 25,000 users and 14,000 users is very large and difficult to handle, and we will need to reduce this further. Given that we need to include user, book, and review embeddings into the model, $10,000$ reviews appears to be a reasonable sample size. Additionally, we see that the rating distribution of this large dataset is highly skewed to the left as shown below – with most users giving 5 star ratings, as seen in Figure 3.

To make the input data more robust, we choose the sample of 10, 000 edges in a stratified manner such that eac rating category getting an equal amount of reviews (2,000 in this case). The sample I got in this manner had 6214 users and 6159 books.

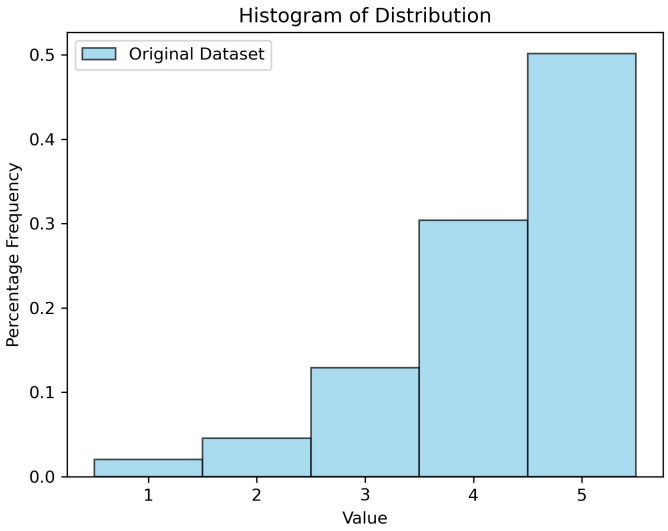

Figure 3: Rating Distribution of the Previous Dataset

An overview of the densest portion of the graph is given in Figure 4.

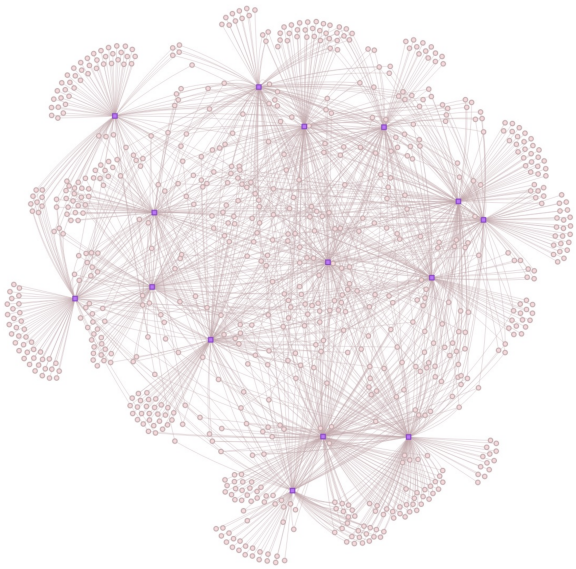

Figure 4: Densest portion of the sample

Next we move to the embedding information. For each review in the

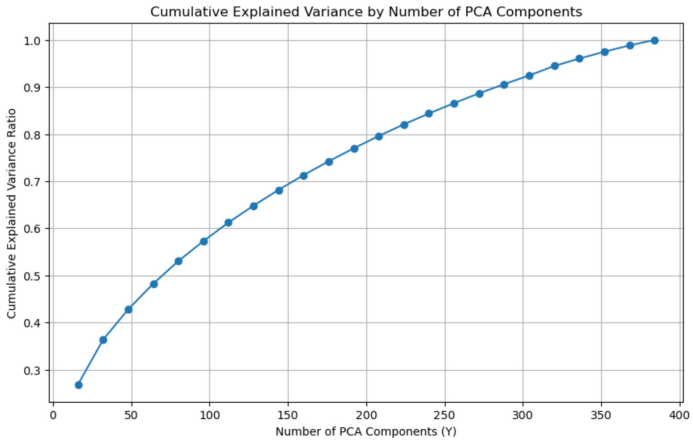

Figure 5: PCA Cumulative Variance of the Review Embeddings

dataset, we are given the user identification, timestamp, rating, review title, and the review itself. Similarly, the book metadata consists of the title, author, description, features, price, genre tags, and average rating. Occasionally, the description or features column may be empty, and we just skip this information if so. Now we want to to store this information as numerical feature vectors. We create these embeddings with the all-MiniLM-L6-v2 Sentence Transformer model available on Hugging Face (see [1]). The sentence transformer only outputs embeddings with 384 features each – but this is too large to handle. Hence, we apply Prinipal Component Analysis to reduce the size of all embeddings. The graph indicating the cumulative variance for each number of components is shown in Figure 5.

Choosing 16 components explains 26.84% of the total variance, 32 components explain 36.34% of the total variance, 64 components explain 48.20% of total variance, and 128 components explain 64.82% of total variance. Given this, we go ahead and test the model using 16, 64, and 128 sized edge embeddings. We don't go beyond 128 components for computational reasons.

Methodology

As mentioned earlier, we use the NNConv implementation layer as the key component of the GNN for our recommender system ([2]). We use this same implementation for both the treatment model (with reviews) and the base model (incorporating only rating).

Model Setup

First, we split the sampled dataset of $10,000$ reviews into a 70/30 train-test split. We consider the training book and edge embeddings calculated earlier. All user embeddings initially start as the zero vector (of size 16) – since we don't have any specific user information and any review information is already held in the edge embeddings. In this manner, we pass the user embeddings, book embeddings, review embeddings, and edge pairs (ie. which user-book pairs have edges between them) into the model. In the GNN model, we run this NNConv layer twice followed by a simple linear layer. This loop of three layers is run forward twice.

Model Training and Evaluation

For assessing the current model embeddings, we use the Bayesian Personalized Ranking (BPR) loss at each step. This loss function aims to find an optimal permutation of books based on the book and review embeddings. The BPR loss is a pairwise ranking loss function which uses *implicit* information provided by the user-book interactions, as opposed to explicit target labels. This is very useful for our case, as we don't really have target labels for each user-book interaction that would enable us to use a standard loss function. (For example, we can't use ratings as targets these are highly correlated to the reveiws themselves.)

The BPR loss aims to rank positive user interactions higher than negative ones. How this works is that for each user, the function samples an item that the user has reviewed and one that the user has not. The algorithm assigns a positive value to the former and a negative value to the latter. The algorithm then computes the predicted score the user would have for both items, trying to ensure that this value is higher for the positive item than the negative one. The loss penalizes instances where the negative item scores higher than the positive one. This inherently assumes that the user prefers their reviewed books over the ones the user did not review. This is not exactly what we want, since interacting with a book does not quite indicate preference, though they are related. However, this is acceptable in this case since this the algorithm does take the actual reviews/ratings to take into account – and as the embeddings are learned, the model shifts more toward the desired direction. The overall BPR loss is the sum of the losses of the sampled (user, positive item, negative item) triples.

For both models, we successively keep track of the BPR loss, precision, and

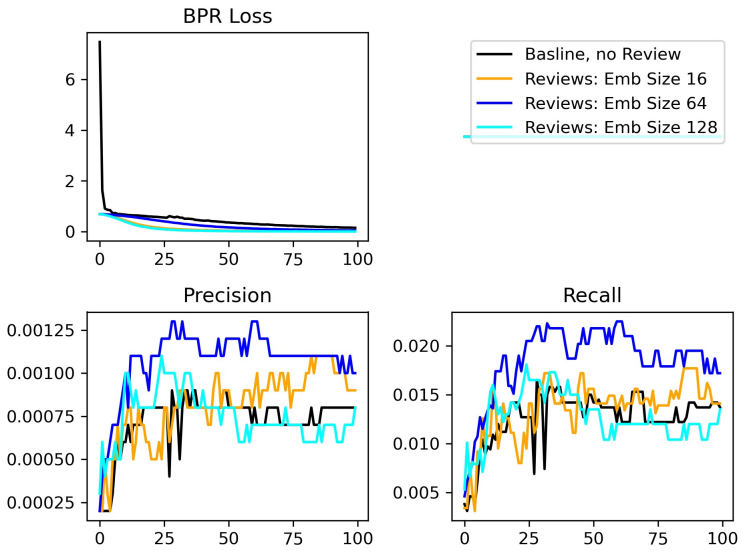

Figure 6: Initial Model Comparison

recall. The model updates parameters using the Adam optimizer.

Results

I ran the above specified model four times – with review embeddings of size 16, 64, 128, and also with no embeddings, just ratings. In the case of the base model, the model is run with the edge embedding vector set to have size 1 and value equal to the rating.

We run each model for 100 epochs, after which the loss appears to converge, though the precision and recall oscillate throughout the training, meaning that these metrics are very sensitive to small changes. The comparison between these models is shown in Figure 6.

As we can see, the model with size 64 review embeddings appears to perform best in the precision and recall metrics, though the size 64 embeddings capture just 48.20% cumulative variance of original embeddings. Among the treatment models, the one with size 128 embeddings is the worst performing even though the embeddings here hold the greatest amount of information. I suspect this is due to overfitting of the model; since the matrices in the models longer embeddings have many more parameters, the models become quite prone to overfitting. Nonetheless, the average treatment model still appears to perform

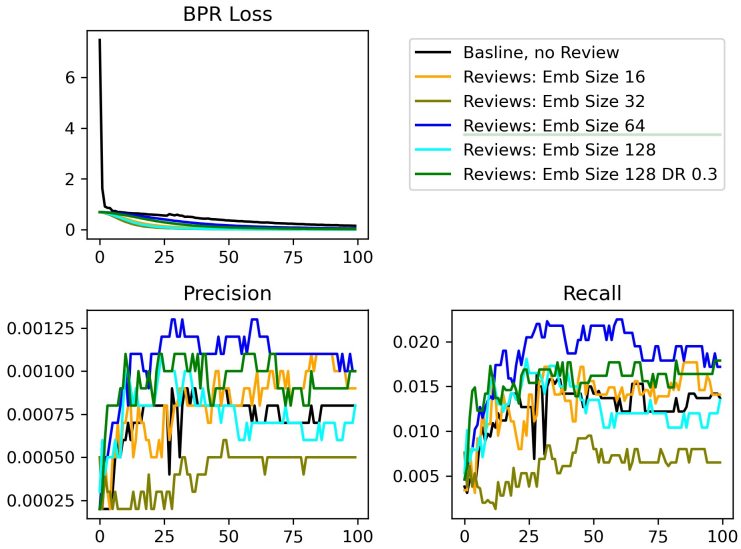

Figure 7: Included Dropout on 128-embedding Model

slightly better than the no review base model. And performing a paired t-test test shows that the average of the three treatment models perform better in both the precision and recall metrics than the control model, under a 0.01 significance level. The BPR loss is lower for the treatment models as well.

To somewhat account for the overfitting in the size 128 embedding model, we use dropout on all the NNConv layers in this model. We use a dropout rate of 0.3. A comparison with this model included is given in Figure 7.

The new dropout model perform superior than the original 128-embedding model, indicating the earlier model was indeed likely overfitting. Though the 128-embedding dropout model still performs worse than the original 64-embedding model. This is likely a question of overfitting, variance, and tuning the hyper-parameters in general – and optimizing these values can be a research question of its own. But still, our conclusion should be clear, that incorporating reviews into the recommender system improves performance (at the 0.01 siginifance level) – but since the actual values of the metrics still stay very small, making good recommendations remains a difficult task.

Conclusion

In this project, we looked at book recommender system models built with the GNN architecture which use sentiment analysis of user reviews, and compared it to the base model which just uses the one number user rating (and no reviews) – to see if incorporating reviews makes a siginificant difference in recommender system performance.

It may be obvious choice to say that it certainly should – however, user/product graphs tend to be sparse making it challenging to perform meaningful analysis and make good predictions (see [4]). So it is not necessary that including elaborate review information will significantly improve model performance.

As we saw in the Results section, including reviews indicates a faster decreasing BPR loss rate and slightly better precision and recall metrics (statistically significant results using the paired t-test test). Nonetheless, the actual values on these metrics remain quite low and making helpful recommendations is still a challenging task.

Additionally, we made many simplifying assumptions throughout the process, such as taking a very small sample of the dataset and reducing review embedding size, which likely impacted the results. As a future experiment, it would be helpful to cluster similar users in order to effectively reduce the dimension and this would collectively take better advantage of the embeddings.

References

[1] sentence-transformers/all-MiniLM-L6-v2 · Hugging Face — huggingface.co. `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`. [Accessed 28-05-2024].

[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[3] Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024.

[4] Zexi Huang, Joao Pedro Rodrigues Mattos, Mert Kosan, Arlei Lopes da Silva, and Ambuj Singh. Attribute-enhanced similarity ranking for sparse link prediction, 2024.