Book recommender system using user book review history

Zihao Mao
University of Massachusetts Amherst
Amherst, USA
zmao@umass.edu

Hongyu Tu
University of Massachusetts Amherst
Amherst, USA
hongyutu@umass.edu

ABSTRACT

In this project, we implemented different book recommendation system that incorporates various attributes as feature vectors, including the user's rating history, review history, and rating history in different genres. These attributes used independently to make recommendations. We also tried to use the deep learning network Ultra Graph Convolutional Network to learn the user's review history and make recommendations.

KEYWORDS

datasets, neural networks, book review, text tagging

ACM Reference Format:
Zihao Mao and Hongyu Tu. 2018. Book recommender system using user book review history. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In this project, we present a book recommendation system that uses the Goodreads dataset as the source of our data. The system uses various attributes as feature vectors for making recommendations, including the user's review history, and rating history in different genres, as we conclude that the rating of a book is only related to the preference of the reviewer as well as the book's genre combination. These data was processed and combined together to feed a DNN for rating prediction, then a wrapper was applied so the model can be used as a backend of a Book recommender system. Since we have two students in our group, we also proposed a separate approach to deal with this problem: apply a Ultra Graph Convolutional Network (GCN) to learn the review history and make recommendations based on the learned representation.

There are several reasons why GCNs can be useful for book recommendation systems. First, GCNs are able to effectively incorporate both the content and the structure of the data in the recommendation process. This means that they can take into account not only the text of the books, but also the relationships between the books, such as which books are often read together or which books have similar themes or styles. This allows GCNs to make more accurate and personalized recommendations.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Second, GCNs are able to handle large-scale and complex data. This is important for book recommendation systems because there are often many books to consider and the relationships between them can be complex. GCNs are able to effectively deal with this kind of data, allowing them to make accurate recommendations even in large and complex data sets.

Third, GCNs are able to learn from the data in an unsupervised manner. This means that they do not require explicit labels or annotations in the data, which can be expensive and time-consuming to obtain. Instead, GCNs can learn directly from the data itself, which can be more efficient and cost-effective.

For the consideration of time effiency and computation power limitaion, we chose UltraGCN, a lighter version of GCN for this project.

2 RELATED WORK

2.1 UltraGCN

Kelong's team from Huawei achieved high end result in recommendation systems with better efficiency and simpler version of GCN name UltraGCN. [1] Due to its performance and efficiency, it became our best option of advanced deep learning neural network structure to train with our own dataset.

3 APPROACH

3.1 Baseline

For the baseline we will only consider rating history of the users as the feature vector. We measured the similarities between the feature vectors by two approaches: Euclidean distance, and Pearson correlation.

3.1.1 *Euclidean distance.*

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

3.1.2 *Pearson correlation.*

$$\text{sim}(u,v) = \frac{\sum_{i \in I} (r_{ui} - \overline{r_u})(r_{vi} - \overline{r_v})}{\sqrt{\sum_{i \in I} (r_{ui} - \overline{r_u})^2} \sqrt{\sum_{i \in I} (r_{vi} - \overline{r_v})^2}}$$

The prediction of the rated score is calculated using the following formula:

$$\text{pred}(u,i) = \frac{\sum_{v \in N} \text{sim}(u,v) \cdot (r_{vi})}{\sum_{v \in N} \text{sim}(u,v)}$$

The books will be sorted by the predicted rate score and the top number of listing books will be recommended.

3.2 Feature Vector

Within the Goodreads Repo, there are two subsets that were useful for this approach: 'goodreads_book_genres_initial.json' and ' goodreads_reviews_dedup.json'. The first one contains book ids as

the key and the categories corresponding to the particular book, while the second one contains review information, where each row includes the username, the book id that gets reviewed, as well as the rating the user gave to the book.

Our first step was to group books together that were reviewed by the same user. Our assumption is that each user will have an interest in a certain and unique combination of categories, and this gives us the theoretical foundation to build our user descriptor. There were 10 categories within the 'goodreads_book_genres_initial.json' dataset, and the descriptor, therefore, was set to a vector with its length equal to 10. Each digit is the average rating the user gave to books in a certain category. In the case of books that were in two categories, its rating will be counted toward both of them. After going through the dataset, we managed to find 464,279 users, and all of them were represented with a vector that has a length of 10.

Next is to represent books, just like user descriptors, they have a length of 10 as well to match the total number of different categories. However, instead of taking the rating from the user into account, they were simply one-hot encoded, as different users will have different opinions towards a book, and it has nothing to do with the book itself. To make up the difference between the user rating and the 1 used to mark the categories, all the descriptors were multiplied by 5.

With both user and book descriptors ready, we set to represent the entire 'goodreads_reviews_dedup.json' dataset with the descriptors. On each row, there's a user and a book, as well as the rating that the user assigned to the book. We concatenate the user and book descriptors together on each row to form a vector with a length of 20. This essentially forms our data to be based to the model for training, with X being the 20 digit-long feature vector, and y being the rating, which is just a float point number. After all the processing (dealing with books with missing information like categories or ratings), there were 15,595,083 rows within our data.

To train the model, we used an 8:1:1 split for the training, validation, and test set. PyTorch with GPU acceleration was used to train a 4-layer fully connected neural network. With the use of the validation set as well as TensorBoard, the structure of the model was fine-tuned to produce the best result. Since we are dealing with float point comparison, we decided to use Mean Squared Error as the loss function. More will be covered in detail in section 4.2.

Lastly, to know exactly how well our model is performing, two custom scoring functions were coded up. With the model ready to give a score to any new user with any new book that the model either has or hasn't seen before, it's ready to make recommendations. It will go through all the books related to a user in the test set, where none of the entries have been seen by the model. Again, results will be covered in section 4.2.

3.3 UltraGCN

We will be using review history of the users as the feature vector in this approach.

For the data, Goodreads has a large volumes of data. 2080190 books are rated by different users in the dataset. To reduce data size for the limited computation power and time, and to keep the most valid information, we only considered the users who reviewed more than 1000 books.

And then we further filtered the data set by using only the top 3000 most reviewed books in the dataset. 3546374 review records were still left, which was still too large. We selected top 10000 most active users as the final filtering procedure.

The dataset is then split into training and testing set with the ratio of 8 and 2. Each row of the data file contains the user id at the first, and the indexed ids of the books the user reviewed.

The book ids are indexed from 0 to 2999.

4 EXPERIMENT

4.1 Baseline

- Euclidean distance: With 20 testing samples, the mean squared error of the predicted scores is 1.626, mean absolute error is 1.020.
- Pearson correlation: Also with 20 testing samples, the mean squared error of the predicted scores is 1.885, mean absolute error is 1.126.

For the comparison of the two similarity functions in the baseline approach, the Euclidean distance model is doing slightly better.

4.2 Feature Vector

To get the best results, there were a lot of attempts on fine-tuning the setup for the neural network. With the optimizer, we started out with SDG with a learning rate of 0.001 and momentum of 0.9, however, in the plot below, the yellow line suggests it's converging relatively slowly. It seemed better when we switched to using Adam as our optimizer. However, due to the layer size, there are signs of clear over-fitting going on with the validation loss starting to increase. More work was done to finally reach a good model where the validation loss is actually better than the training loss after 28 epochs (Pink line). The final structure we settled on was a 4-layer network, first going from 20 to 200, then 200 to 200, followed by 200 to 10, and then 10 to 1. All these are linked by a leaky Relu of slope -0.1, and a dropout layer of probability 0.25 right in the middle (between the first two and last two). Overall I'm happy with how the model was trained, it only took 10 minutes for 30 epochs on my RTX 2080 Super, which made it possible for me to try out more than 30 different configurations.

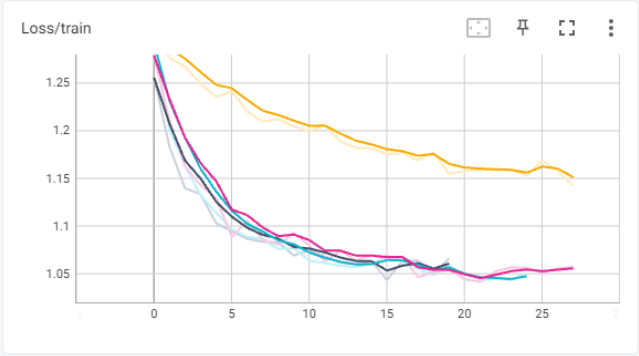

Figure 1: Training Set Loss

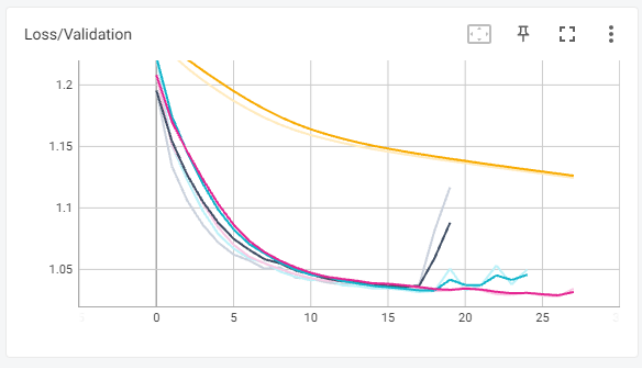

Figure 2: Validation Set Loss

Continue on the evaluation, the first was ranking accuracy. Given a user in the test set, the model will be used to predict a list of books' ratings given by that user. Then the book will be sorted based on the score they get. However, the true ratings we have in hand are all integers instead of float points, like the scores the model gave us. To deal with this problem, we grouped the true ratings into 5, 4, 3, 2, and 1 and counted the number of books in each rating level. Say there are N books that were given a rating of 5, as long as the books that have a 5 for true rating were ranked in the top N with respect to the predicted list, it will be considered correct, as there's no better between 5-star books. Then for the O number of books with a true rating of 4, if they were ranking within the first (O + N), they will be considered correct. And here are the results:

Table 1: Ranking Accuracy of NN Model

Subset Size	No. of Users used in Eval	Accuracy
2	151956	0.92040
5	72850	0.81627
10	37724	0.75945
20	17010	0.72963
All Available	100730	0.90918

In real-world applications though, the entire ordering always doesn't matter as much. Thus, a separate scoring function was written to check the top predictions. We take the top k books that the model believes the user will like the most and see if it is in the highest tier of all the books. In other words, if there are 5-star books in the pool of recommendations, will the model be able to pick it as the most suggested book? In case there are no 5-star books in the pool, the list of the lower tier will be used. And here are the results.

Table 2: Hit rate of NN Model

Top K	No. of Users used in Eval	Hit rate
1	151956	0.678847
3	88608	0.861953
5	61727	0.914121
10	34085	0.958926
15	22441	0.974154

From the table, we can see that given the chance of recommending 5 books to a user, the possibility of the user will be able to find a book that is 5 stars to him/her is 92%. I consider this a really good result, given that these are all test data that the model has never seen before.

4.3 UltraGCN

The trend of loss and nDCG value during training is shown in the figure 3 and figure 4. We can see that the model is learning correctly.

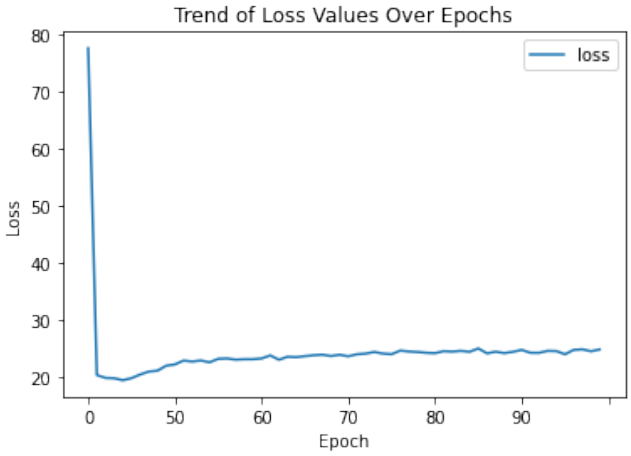

Figure 3: Loss Values Over Epoch of UltraGCN

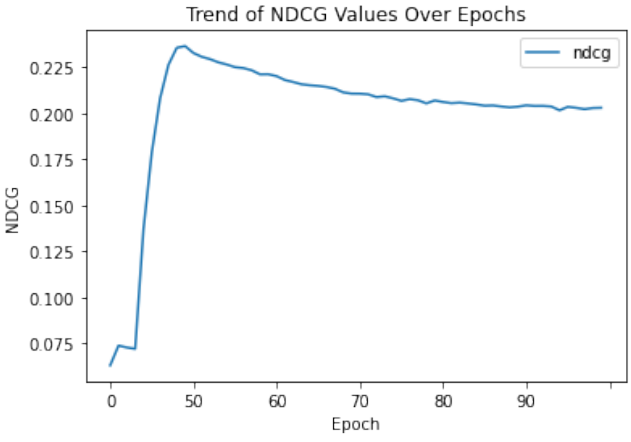

Figure 4: Trend of NDCG Values Over Epochs

The result in 2048 batch size is shown in the table3. Although the model is effectively trained, the result is still much lower than our feature vector approach. We believe this is majorly due to the relationship between users' preference and their review history soley without considering the genre and their ratings to the books.

Table 3: Result of UltraGCN

Recall@20	nDCG@20	Precision@20
0.18391	0.23639	0.18939

5 CONCLUSION

Through this project, we attempted three different approaches to try to create a book recommender system using the information retrieval techniques that we have learned this semester. Both the feature descriptor model and the GCN model perform well when compared to the baseline model. However, if given more time, we can potentially add more comparison experiments between these two models and have more scoring benchmarks, especially, use the same feature vector as the input for two models if we can, so we can have a better understanding of the advantages/disadvantages of each model. Overall, we are really glad about this experience and we have learned a lot through the process.

REFERENCES

[1] Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. 1993. Maintaining Discrete Probability Distributions Optimally. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 700)*. Springer-Verlag, Berlin, 253–264.