EE-452 Final Project Report: Goodreads Recommender System

Hans Kristian Bjorgo Kvaerum | 381875 | hans.kvaerum@epfl.ch
Kaede Johnson | 357472 | kaede.johnson@epfl.ch
Git Repository: https://github.com/kaedejohnson/EE452_2024_Team5_Project

1 Introduction

This report details our experiments with recommender system (RS) architectures on the Goodreads-10k dataset (Zajac, 2017). Our project compares traditional RS approaches such as Singular Value Decomposition (SVD) and User-Based Collaborative Filtering (UCBF) to modern graph based approaches. We solve two tasks with graph-based recommendation methods: binary edge prediction to estimate a user's desire to interact with a book and edge weight regression to estimate a user's rating (and therefore enjoyment) of a book. While each task yields a standalone RS, we close by proposing a way to combine our predictors into an ensemble RS model.

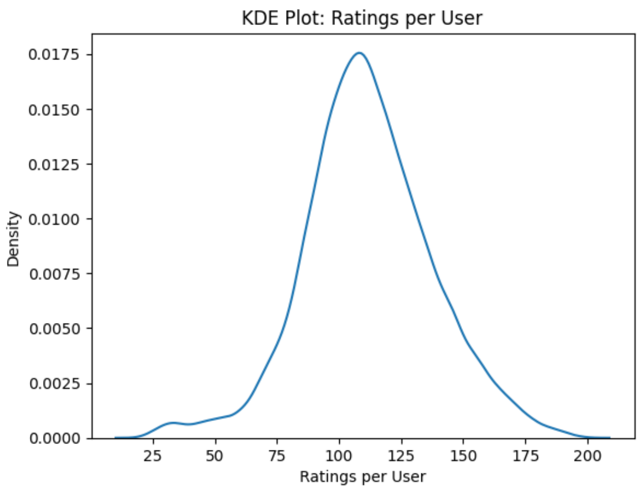


Figure 1: Rating count by user

2 Exploratory Analysis

We first explore the Goodreads dataset to identify pertinent features and appropriately filter out users and books. The original dataset has roughly 6 million ratings of 10,000 books made by 60,000 users. With limited resources available, we decide to focus on a representative sample from the dataset. We first filter the data to English-language books published after 1800 in an effort to reduce noise (see Appendix Figures 1 and 2). Additionally, after exploring rating count distributions on the user (see Figure 1) and book levels, we retain only those users with 100+ ratings and books with 200+ ratings, ensuring the remaining users and books will have enough data to exploit collaborative signal. Note that these filtering steps retain the majority of observations while removing cases that potentially suffer from lack of information. Finally, we sample 3,000 users and 1,500 books with probability proportional to number of ratings placed or received, yielding about 200,000 ratings in our final dataset (along with about 25,000 to-reads).

In an RS-setting, lack of interaction between a user and a book (or 'negative interaction') can occur for two reasons: either the user is unaware of the book (lack of exposure) or the user is aware of but uninterested in the book (active decision) (Lin et al., 2019). In our analysis, we find a strong, positive, and statistically significant correlation between the number of ratings on a book and the number of times users list a book as 'to-read'. Hence, we expect to-read data to be helpful for delineating between the two types of negative interactions in a collaborative message passing context, and experiment with including to-read data in user-book interaction matrices during model development. Supporting this decision is past research showing such data can improve RS performance in a similar context (Schnabel et al., 2016). Including to-reads data does introduce one complicating factor, however: to-reads are binary expression of sentiment whereas ratings include a quantitative expression of sentiment. This encourages us to experiment with two recommendation approaches, the first being binary user-item interaction prediction.

The second domain of our experimentation involves rating prediction. Per Figure 2, we find users generally rate books positively, with the modal score at five out of five stars. Even so, a sizable portion of ratings dip to three and below. Lower ratings provide a valuable signal to an RS by highlighting what users might not like (Cena et al., 2023). Consequently, we decide to include the full range of ratings when pursuing rating prediction.

3 Experimental Setup

For graph-based methods, we organize our 3,000 users and 1,500 books into a bipartite graph with edges from users to books based on rating or to-read information.

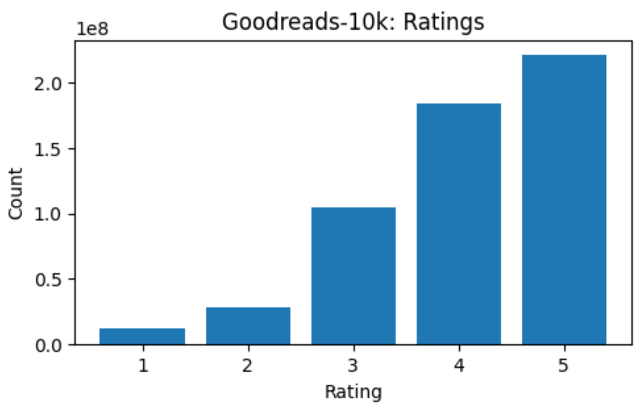


Figure 2: Rating count by value

To ensure all architecture implementations are trained and evaluated on the same data, we create stratified train and test splits prioritizing users. Our split retained 70% of each user's edges for train data and the remaining 30% for test data. Stratified splitting addresses the cold-start problem by ensuring information about all users is included in both the training and test sets. A random split, meanwhile, might exclude select users from either train or test data, potentially hindering learning, evaluation, or inference.

To evaluate RS performance, we consider the metrics precision@k, recall@k (Yang et al., 2018), Normalized Discounted Cumulative Gain (NDCG) (He et al., 2017) and Mean Reciprocal Rank (MRR). Precision@k is the mean fraction of relevant items (found in the validation set) among the k recommended candidates for a user. Recall@k is similar, except it divides the number of relevant items found by the total number of relevant items in the validation set (for a particular user). NDCG measures ranking quality by first assuming there is an ideal ranking of items sorted in descending order of relevance, and then measures how close a set of recommendations is to this ideal ordering. Finally, MRR measures how quickly the RS proposes the first relevant item among k candidates. Note that, as discussed earlier, we do not have full information about negative interactions and so these metrics do not paint a full picture despite providing a foundational overview of RS performance. Finally for all experiments we use $k = 20$ per Wang et al. unless explicitly stated otherwise.

4 Approach 1: Binary Interaction Prediction

We apply Neural Graph Collaborative Filtering (NGCF) to predict user-item interactions (Wang et al., 2019). NGCF is well-suited for a sparse, bipartite user-book matrix that lacks ample data for feature extraction, in particular because of its ability to represent higher order collaborative signals. The full set of hyperparameters can be seen in Appendix Table 1.

To arrive at our set of hyperparameters, we first perform a model development stage on an even smaller sample of 500 books and 1,000 users. We then train versions of the NGCF interaction predictor both with and without to-read edges as well as with BCE loss versus with BPR loss. The impetus for exchanging loss functions is as follows: we theorize that the abundance and ambiguity of non-interactions in Goodreads data may make the negative sampling under BPR problematic even with the signal of to-read data. By considering all user-book interactions independently, BCE avoids the need to select sufficiently informative negative samples (and may avoiding skewing interaction probabilities as a result). NDCG@20 and Recall@20 outcomes for our four model development variants can be seen in the Results section below.

5 Approach 2: Rating Prediction

This section outlines the three different architectures employed to predict user ratings. The general idea is to impute the negative interactions in the rating interaction matrix, recommending top k candidates based on estimated rating.

5.1 Singular Value Decomposition (SVD)

SVD is a popular matrix factorization technique that decomposes a matrix A into three matrices: U, Σ, V^T, where U is an $m \times r$ orthogonal matrix representing user features vectors, Σ is an $r \times r$ diagonal matrix containing singular values, and V^T is the transpose of an $n \times n$ orthogonal matrix representing, in our case, book feature vectors. r represents the rank of matrix A. SVD is useful in a RS-setting for two reasons: it is able to capture latent relationships between users and items, allowing us to compute predicted likeliness of a book by a user through the dot product of the corresponding row in U and column in V^T, and it can produce a low-dimensional representation of the original user-item space through the selection of r (Sarwar et al., 2000). For our implementation, we utilized the python library `surprise`.

5.2 User-Based Collaborative Filtering (UCBF)

The general idea behind user-based collaborative filtering is that people often get good recommendations from those with similar taste. In UCBF, we derive a user-to-user similarity matrix by computing the cosine similarity of each user's ratings in the user-item interaction matrix. From the resulting similarity matrix, one can select the n most similar users and base recommendations on their preferences, by computing a weighted average of the these users' ratings using their cosine similarity as weights. By normalizing the resulting score by the sum of the weights, we obtain scores in the range 1-5 which can be interpreted as ratings. For recommendations, we select the top k candidates among these scores.

5.3 Dual Graph Neural Network (GNN)

To address the bipartite nature of the graph, we also implement a custom dual GNN-architecture for rating prediction. It employs two separate GNNs that learn node embeddings for users and books respectively. Each GNN module, in our case either a GCN or GIN, computes node embeddings by aggregating and transforming local neighborhood information based on the input features x and the connectivity structure provided in the $edge\_index$. In essence, node embeddings are generated by passing the input sequentially through layers of the chosen GNN architecture, where each layer updates the node's feature by incorporating features from its immediate neighbors as defined by the $edge\_index$, followed by non-linear transformations and optional dropout for regularization.

By concatenating a user and a book embedding, we then create edge embeddings which are passed to an MLP to predict the final rating. The model can be trained through a regression objective using MSE or Huber loss, or for multi-class classification using CrossEntropy. During inference, the architecture can make rating predictions on edges not seen in the train-graph through either cosine similarity or the MLP. If trained for classification, we apply softmax on the raw output logits of the MLP to compute class probabilities used in a weighted sum for rating estimation. Unlike the regression objective, this score is bounded on the [1,5] domain by definition. Additionally, one strength of this approach lies in its ability to incorporate node-features, meaning qualitz handcrafted features for users and books can improve its performance. In our experiments, we only employ features easily derived from the ratings-dataset: distribution of ratings (count per star) and mean rating. These input embeddings are then passed through GNN-layers to compute final node embeddings.

6 Results

6.1 Approach 1: Binary Interaction Predictor

NDCG@20 results for the interaction predictor's development stage are visible in Figure 3. We find that for NDCG, the models with BCE loss outperform their BPR-loss counterparts both with and without to-reads information included. Furthermore, the inclusion of to-reads information boosts NDCG under both BCE and BPR loss. That our NGCF architecture is better able to position positive-interaction books above negative-interaction books (in terms of probability of user-book interaction) seemingly confirms our hypotheses that (1) to-reads data contains important signals about negative

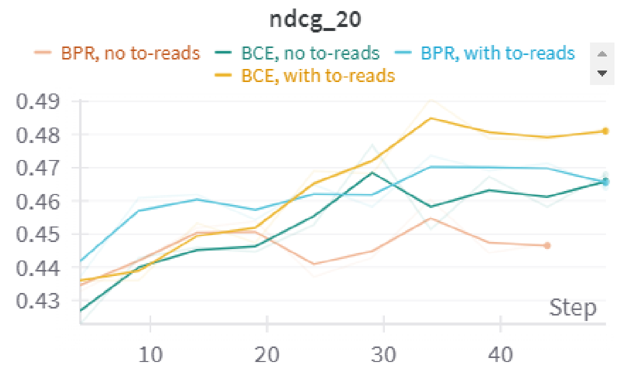


Figure 3: Interaction Predictor Variant NDCG@20 (Gaussian smoothing, $\sigma = 3$)

interactions in a ratings-only framework and (2) BCE loss avoids the pitfalls of sampling the 'wrong sort' of negative interactions.

The inclusion of to-reads data has negligible effects on Recall@20 (see Appendix Figure 3; note that while with-to-reads curves are slightly beneath their no-to-reads counterparts, the number of global positive interactions per user with to-reads data - and thus the denominator of Recall@20 - is higher). To test whether the to-reads information is simply underutilized under the prevailing architecture, we also train a model variant with more hidden layers. Early results support the notion that increased capacity could help squeeze more power from to-reads, but a leveling off of Recall@20 in later epochs means more tuning may be required for the model to exploit the increased capacitz correctly (see Appendix Figure 4).

We train the model on our larger subset of 1,500 books and 3,000 users with BCE loss and to-reads included given this variant's success in the development stage. NDCG@20 and Recall@20 for our model and a LightFM Matrix Factorization baseline (Kula, 2015) are visible in Table 1. We find that while NDCG sees a substantial boost under our NGCF implementation relative to LightFM, Recall@20 is slightly lower. To test once again whether increased capacity could improve NGCF performance, we train on the full subset using more layers and a higher learning rate; Appendix Figure 5 suggests once again that more tuning is required to exploit excess capacity.

Model	NDCG@20	Rec@20
LightFM	0.116	0.092
NDCG	0.380	0.083

Table 1: Interaction Prediction Architectures with their Respective Scores.

Model	Pre@20	Rec@20	NDCG	MRR
SVD	0.108	0.101	0.079	0.286
UBCF	0.207	0.208	0.233	0.590
dGCN R	0.112	0.104	0.003	0.210
dGCN C	0.111	0.104	0.003	0.214
dGCN C cw	0.101	0.091	0.002	0.208
dGIN R	0.111	0.103	0.003	0.224
dGIN C	0.111	0.103	0.002	0.240
dGIN C cw	0.087	0.080	0.002	0.240

Table 2: Rating Prediction Architectures with the Respective Scores.

6.2 Approach 2: Rating Predictor

In Table 2, we observe UCBF achieving the highest scores across all metrics. The double GNN architecture achieves slightly better precision and recall than SVD yet lower NDCG and MRR. This suggests that while our dual GNN might be slightly better at identifying relevant items, its ordering of these item is less ideal. All GCN variants are trained for 100 epochs with 4 GNN layers, 4 MLP layers and a hidden dimension of 64 for all modules. On the same set of hyperparameters, we observe that formulating the prediction task as a regression (dGCN R) or classification (dGCN C) problem does not substantially impact performance (what minor differences exist are not shown in the table due to rounding). However, it does impact predicted ratings, with the regression model predicting scores above 5. Introducing class weights to address rating imbalances negatively affected performance on the base hyperparameters across both GCN and GIN implementations. It also produced a greater spread in rating predictions, but that rarely produced better recommendations. All hyperparameters for the double GNN model are detailed in Figure 6. For SVD, we used $r = 50$, and for UCBF we computed based the weighted sum on the ratings of the 5 most similar users.

7 Discussion and Future Work

Regarding binary interaction prediction, our NGCF approach's relative lack of success in recall despite improvements in NDCG suggest future work should target the former metric. Because we had originally intended to combine our two approaches (see the end of this section), we were comfortable obtaining high precision alone from the interaction predictor and later relying on the rating predictor to surface ideal books from imperfect interaction probabilities. Complications with the rating predictor inhibited this plan.

Regarding rating prediction, unfortunately, our dual GNN approach did not yield the desired performance. The main challenge appears to stem from the overall design of the GNN's architecture and forward pass. In collaborative filtering models like UCBF and SVD, success largely depends on effectively capturing latent representations from the user-item interaction matrix. These methods work well because they manage to infer underlying patterns from the interaction data, which are then used for predictions or recommendations. However, our double GNN approach aims to approximate an edge attribute function that is largely dependent on the quality of our node embeddings. It is not unlikely that the latent representations found in the collaborative filtering methods contain useful information currently not captured by our structural GNN embeddings. Additionally, hyperparameter tuning and changing the GNN-module architectures to GIN or GraphSAGE, seemingly had limited effect on model performance, further suggesting the overall design is flawed.

Originally, our plan was to improve on our current GNN approach with LightGCN, which retains only the neighborhood aggregation of GCN while excluding feature transformation and nonlinear activation. LightGCN learns user and item embeddings by linearly propagating them on the user-item interaction graph, capturing the collaborative signal through neighborhood aggregation. The final embedding is the weighted sum of the embeddings learned at all layers, capturing multi-hop relationships between users and items (He et al., 2020). Unfortunately, we had difficulties implementing an effective implementation of this framework. Additionally, we could have tried to repurpose the NGCF model from interaction prediction for rating predictions.

As a final note on rating prediction, we discovered that computing user similarities for UCBF on the SVD latent user representation did not negatively impact UCBF performance. This interesting result means we can extract the same performance from SVD while reducing the user embedding dimension from 1500 in the original interaction matrix to 50 in the SVD representation - a computational boon which is outside the scope of this project but worth mentioning nonetheless.

We close by proposing a way to combine our two tools. Our approaches target the probability of user-book interaction and estimated user ratings. The heterogeneity of these approaches means they can be combined to form an ensemble recommendation engine. By obtaining two vectors with estimated ratings $\boldsymbol{R}$ and interaction probability $\boldsymbol{I}$ for each book for a particular user, we can compute an ensemble score by taking the Hadamard product of the two vectors $I \odot R$. Final recommendations would be the top k candidates based on ensemble score, and should benefit from incorporating two perspectives on recommendation.

References

Federica Cena, Luca Console, and Fabiana Vernero. 2023. How to deal with negative preferences in recommender systems: a theoretical framework. *Journal of Intelligent Information Systems*, 60(1):23–47.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182.

Maciej Kula. 2015. Metadata embeddings for user and item cold-start recommendations. *arXiv preprint arXiv:1507.08439*.

Sheng-Chieh Lin, Yu-Neng Chuang, Sheng-Fang Yang, Ming-Feng Tsai, and Chuan-Ju Wang. 2019. Negative-aware collaborative filtering. In *RecSys (Late-Breaking Results)*, pages 41–45.

B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. 2000. Application of dimensionality reduction in recommender systems: A case study. In *WebKDD Workshop at the ACM SIGKKD*.

Tobias Schnabel, Paul N Bennett, Susan T Dumais, and Thorsten Joachims. 2016. Using shortlists to support decision making and improve recommender system performance. In *Proceedings of the 25th International Conference on World Wide Web*, pages 987–997.

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19. ACM.

Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. Hop-rec: high-order proximity for implicit recommendation. In *Proceedings of the 12th ACM conference on recommender systems*, pages 140–144.

Zygmunt Zajac. 2017. Goodbooks-10k: a new dataset for book recommendations. *FastML*.

A Appendix

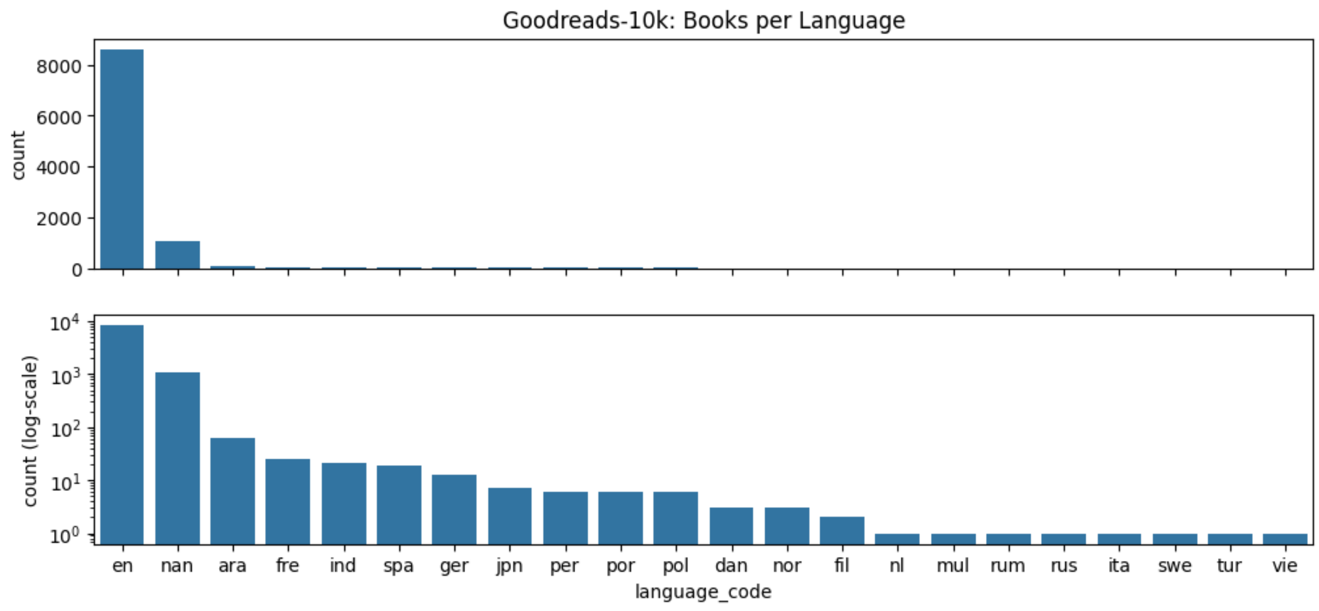


Figure 1: Book Languages

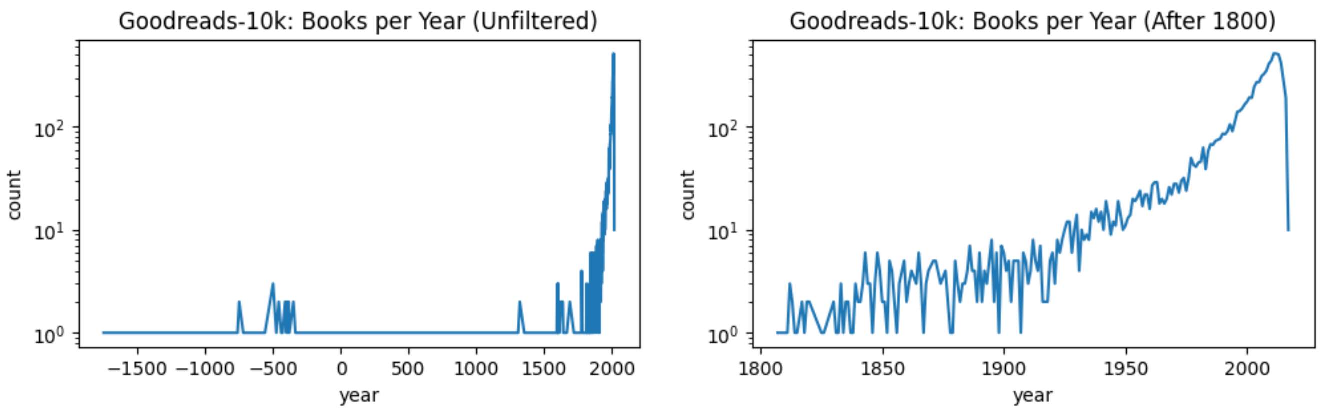


Figure 2: Book Release Years

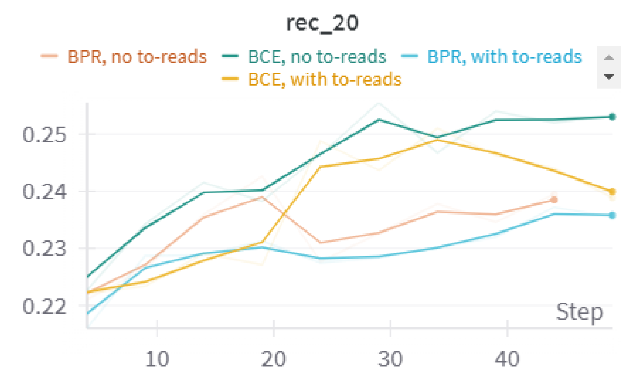


Figure 3: Interaction Predictor Variant Recall@20 (Gaussian smoothing, $\sigma = 3$)

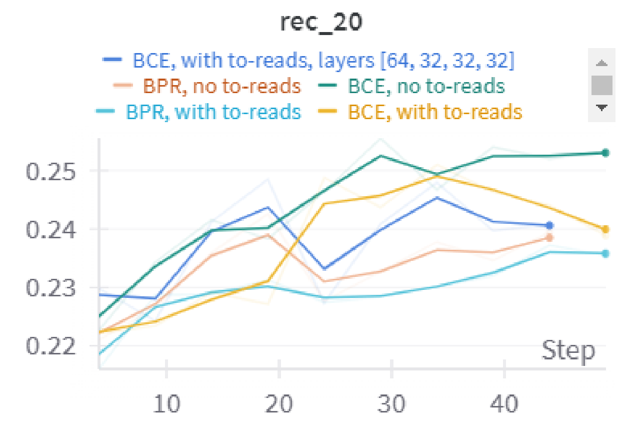


Figure 4: Interaction Predictor Variant Recall@20, with Variant that Includes More Layers (Gaussian smoothing, $\sigma = 3$, layers = [32, 16] where not specified)

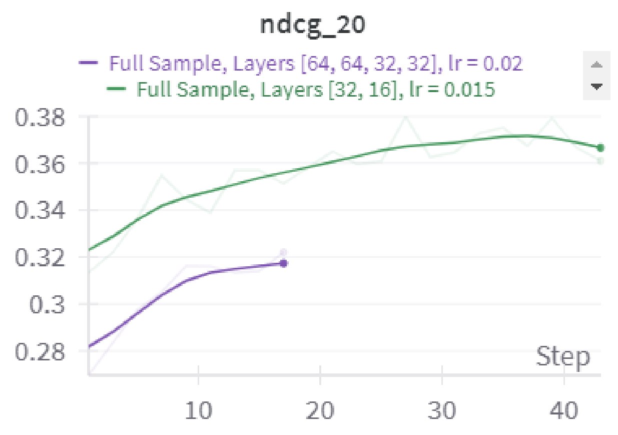
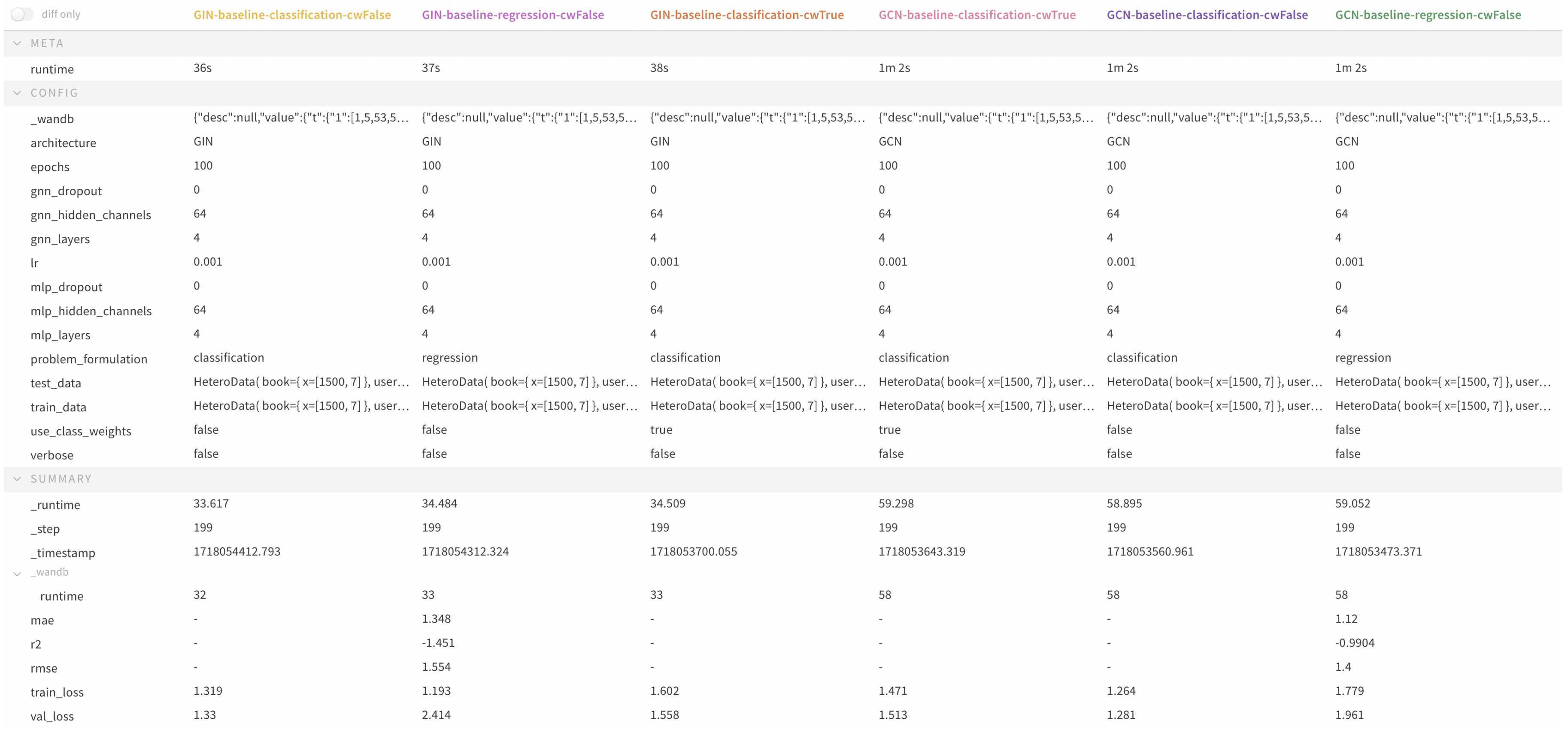


Figure 5: Interaction Predictor Full Sample NDCG@20, with Variant that Includes More Layers (Gaussian smoothing, σ = 3, layers = [32, 16] where not specified)

	Loss Function	To-Reads As Edge	Cutoff for Pos. Interaction	Batch Size	Hidden Layers	Emb. Dim.	Learning Rate	Regularization	Message Dropout	Node Dropout
Interaction Predictor	BCE	True	0.9	16	[32, 16]	52	0.015	0.005	0.01	0.01

Table 1: Training Parameters



diff only	GIN-baseline-classification-cwFalse	GIN-baseline-regression-cwFalse	GIN-baseline-classification-cwTrue	GCN-baseline-classification-cwTrue	GCN-baseline-classification-cwFalse	GCN-baseline-regression-cwFalse
META						
runtime	36s	37s	38s	1m 2s	1m 2s	1m 2s
CONFIG						
_wandb	{"desc":null,"value":{"t":{"1":[1,5,53,5…	{"desc":null,"value":{"t":{"1":[1,5,53,5…	{"desc":null,"value":{"t":{"1":[1,5,53,5…	{"desc":null,"value":{"t":{"1":[1,5,53,5…	{"desc":null,"value":{"t":{"1":[1,5,53,5…	{"desc":null,"value":{"t":{"1":[1,5,53,5…
architecture	GIN	GIN	GIN	GCN	GCN	GCN
epochs	100	100	100	100	100	100
gnn_dropout	0	0	0	0	0	0
gnn_hidden_channels	64	64	64	64	64	64
gnn_layers	4	4	4	4	4	4
lr	0.001	0.001	0.001	0.001	0.001	0.001
mlp_dropout	0	0	0	0	0	0
mlp_hidden_channels	64	64	64	64	64	64
mlp_layers	4	4	4	4	4	4
problem_formulation	classification	regression	classification	classification	classification	regression
test_data	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…
train_data	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…	HeteroData(book={ x=[1500, 7] }, user…
use_class_weights	false	false	true	true	false	false
verbose	false	false	false	false	false	false
SUMMARY						
_runtime	33.617	34.484	34.509	59.298	58.895	59.052
_step	199	199	199	199	199	199
_timestamp	1718054412.793	1718054312.324	1718053700.055	1718053643.319	1718053560.961	1718053473.371
_wandb						
runtime	32	33	33	58	58	58
mae	-	1.348	-	-	-	1.12
r2	-	-1.451	-	-	-	-0.9904
rmse	-	1.554	-	-	-	1.4
train_loss	1.319	1.193	1.602	1.471	1.264	1.779
val_loss	1.33	2.414	1.558	1.513	1.281	1.961

Figure 6: Hyperparameters used for our GNN rating prediction runs.