Report.

Dataset.

The data used consists of 100,000 movie reviews and requires a lot of processing. I encoded all temporal features in a cyclical approach. Text data was tokenized and embedded using Spacy. I used pgeocode.Nominatim to parse US zip codes and extract coordinates. The dataset had the wrong schema, which demanded some adjustments. There were several corrupted samples, so I removed them. As for users, I encoded the occupation using OHE.

As the goal is creating a recommendation system, I split the dataset in two ways: first - an ordinal 20/80 test/train split; second - I separated a group of users to get approximately the same 20/80 proportion but with disjoint user sets. I looked at the distribution of the number of reviews per user in the dataset and chose the range from 70 to 100 to be the most representative, as users within it haven't reviewed too many or too few movies. Then, I shifted the borders a little to fit the proportion requirement.

Model.

For modeling, I chose the XGBoost regressor. It provides a simple interface, supports large feature sets, and can use GPU for a faster learning process. I wrapped it into a grid search and a progress bar (for convenience). I ran it for the "2080" and "disjoint" train/test sets.

Then, I created a class that would load the models and provide a user-friendly interface. The "recommendation()" method takes a series or list of user IDs, then encodes everything and maps onto those IDs. It predicts users' reviews on the provided data and returns a convenient report of N best movies for each user according to the predictions.

Evaluation.

I chose ordinal MAE, MSE, and RMSE metrics as a regression task for this recommendation system. However, the nature of its utilization requires some adjustments. I grouped the metrics in 2 ways, resulting in a 2x2x3 tensor.

First, I separately counted the metrics for rounded (discrete) and raw (continuous) predictions. Second, I did it for two different test sets: 2080 and disjoint. The results are as follows:

model	prediction_type	MAE	MSE	RMSE
2080	continous	0.752017	0.901181	0.949305
2080	discrete	0.717289	0.984102	0.992019
disjoint	continous	0.856391	1.157878	1.076047
disjoint	discrete	0.820828	1.227491	1.107922

Results.

As we can see, the 2080 dataset shows better results, which basically shows the difference between the model familiarized with certain users and the one generalizing data from all the users and trying to unify the predictions. The second model has some space for improvement by being periodically fed more user-specific data. But in the end, it has decent results, proving the possibility of unifying at least some portion of data across different users.

For the discrete/continuous separation, although the difference isn't as prominent as with different models, it favors continuous representation.

Pros: The second model can be easily maintained and expanded by adding more users to the tables with empty reviews and then updating occasionally. The model itself is pretty simple and doesn't take too much memory.

Cons: The unification exploited in the "disjoint" model results in worse loss and is proven to be at least partially solvable by adding the reviews from the same user it shall be tested on. Solving this problem would require regular refitting of the whole model, which might be costly.