Steam Game Recommender System Design Proposal

zID: z5521676
Name: KAIYUAN LI

Games are a great way to relax and entertain during leisure time. As an experienced game enthusiast, I deeply understand the joy that comes from playing games. However, after playing hundreds of games, I started to feel lost, not knowing what to play next. Steam, as the largest PC gaming platform, has over 20 million daily active users. It offers a recommendation feature called "Explore Your Queue," which is intended to suggest new games for users to try. However, when I used this feature, I was quite dissatisfied—the recommendations were inconsistent and it was difficult to find games I truly wanted to play. Therefore, I want to design a recommendation system based on Steam games, to recommend new games that better match user preferences and help solve the problem of having "nothing to play."

Scope

The domain of our recommender system is games on the Steam platform. The target users are registered Steam users. The system will be implemented on the web, displaying 10 games recommended by the recommender system to each user. These 10 games will be shown one by one. Before the next game is shown, the user will indicate whether they are interested or not, which allows us to initially assess the quality of the recommendations. After a user purchases a game, we will check whether the purchase results in a refund to further evaluate the success of the recommendation (Steam has a refund policy that allows users to get a refund only if both of the following conditions are met: the request is made within 14 days of purchase and the playtime is less than two hours). After the user has played the game for two hours, we will conduct a questionnaire to ask whether they are satisfied with the game and with our recommender system.

We adopt a collaborative filtering-based recommendation approach. When using the system in a dynamic scenario, we consider that new games are continuously being added. To address this, our recommendation model is retrained on a weekly basis to adapt to such updates.
At the same time, in order to simulate real user recommendation scenarios, we also need to make predictions for new users who do not exist in the original dataset, which introduces the cold-start problem. To solve this, we allow new users to select a few favorite games to construct a small rating vector. Given a fixed item (game) vector matrix, we then solve for the corresponding user vector, enabling us to generate personalized recommendations without retraining the entire model.

From a business value perspective, successful recommendations can promote game sales. Personalized recommendations based on user preferences can stimulate purchase desire. Even if the user does not buy the game immediately, there is a high chance they will add it to their wishlist and purchase it during discounts. If the user has a good gameplay and purchase experience, it will increase user retention.

Dataset

Our dataset comes from Game Recommendations on Steam on Kaggle (URL:https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam/data?select=games.csv).
The dataset contains over 41 million cleaned and preprocessed user recommendations (reviews) from the Steam Store. It consists of three files, among which we only use recommendations.csv and games.csv.

For recommendations.csv, it reflects user-game interaction records. We only focus on the following features:
appID, unique identifier for each app
is_recommended, Is the user recommending the product?
hours, How may hours played by user
user_id, User's anonymized ID

For games.csv, it reflects various information about the games. We only focus on the following features:
appID, unique identifier for each app
Title, Game name
date_release, Product release date
windows/mac/linux, device platform
rating, Product rating category
positive_ratio, Ratio of positive feedbacks
user_reviews, Amount of user reviews availble on the Steam page

Among these, we use the following fields to train the SVD model:
user_id: As the unique identifier of users, used to construct the row index of the rating matrix.

app_id: As the unique identifier of games, used to construct the column index of the rating matrix.

is_recommended: Indicates whether the user recommends the game; we binarize the rating (True = 1, False = 0).

As for the remaining features, they are used to assist the recommendation system.
For example, to optimize the presentation of recommended games, we can consider

hours, windows/mac/linux, rating, positive_ratio, and user_reviews to prioritize games that match the user's operating system and have long playtime and high positive feedback.
We also use rating in the computation of the Novelty metric during model evaluation.

For limitations, many features of games are dynamically updated over time. This may result in the model learning outdated information. At the same time, the dataset is cleaned, and some other key information has been removed during preprocessing, which leads to fewer features in games.csv, making it harder to gain deeper insights into the games. These potential issues may affect the accuracy of the recommendation results.

Method

We adopted a collaborative filtering-based matrix factorization method (SVD) to build the recommendation system. The goal is to match games that are similar to those the user has already played. Since it is difficult to find high-quality games based solely on selecting tags, we need such a recommendation system.
First, we construct a user-game rating matrix. From recommendations.csv, we extract the user_id, app_id, and is_recommended fields. We map is_recommended to binary ratings, assigning True = 1 and False = 0 in the original dataset, thereby constructing a sparse user-game matrix R. Next, we perform low-rank matrix factorization on the rating matrix R to obtain the user vector matrix and the game vector matrix. We then split the constructed rating matrix into training and validation sets, and train using TruncatedSVD from scikit-learn.

We use the model.predict function to evaluate training performance. Note that since we defined True = 1 and False = 0, the predicted scores are values between 0 and 1. (To match the actual situation of our dataset, we set biased=False to disable adding bias terms.)

Next, for each user, we recommend the top 10 unplayed games with the highest predicted scores as recommendation results.

To apply the recommendation system to real users, we can let them select a few favorite games to build a small rating vector. Under the condition that the item (game) vector matrix is fixed, we use the model.recalculate_user function in the implicit library to solve the user vector matrix for the real user. Then, based on this user vector matrix, we can recommend the top 10 unplayed games with the highest predicted scores. The advantage of this method for users not present in the dataset is that we do not need to retrain the model (i.e. adding user_id, app_id, and is_recommended into the training set and retraining), which effectively solves our cold-start problem.

In the above process, we link recommendations.csv with games.csv via app_id,

allowing us to use features from games.csv for display purposes. Additionally, before presenting the final top 10 recommended games, we can optimize the ranking by incorporating the **Win/Mac/Linux** and **rating** columns.

In this way, we obtain 10 candidate games in sorted order, which are presented to the user one by one. The user can mark each recommended game as either interested or not interested.

If the user purchases the game, does not request a refund, and plays for more than two hours, we will collect final feedback from the user through a questionnaire.

Evaluation

For the recommendation model itself, we use the following metrics:
RMSE: During the training process of the recommendation model, we use RMSE to measure how closely the predicted ratings match the actual binary feedback. A lower RMSE indicates better predictive performance on historical data and helps us fine-tune the model before making top-N recommendations.

Top-N Precision: Among the top 10 recommended games, how many are actually marked as "interested" by the user.

Novelty: We define a game's popularity as its **rating**. This metric is used to determine whether the system tends to recommend only popular games, or whether it can also discover less-known but well-matching titles.
There can still be high-quality games among niche ones, and we do not want the recommender to be biased only toward popular games.

Among the three metrics above, Top-N Precision is the most important. We care more about whether users are interested in the recommended games. From a business perspective, user satisfaction and willingness to purchase are what matter most.

For the recommendation system itself, we use the following metrics:

Click-through Rate: The proportion of recommended games marked as "interested" by the user.

Purchase Rate: Among all recommended games, the proportion that are ultimately purchased and not refunded.

Questionnaire Satisfaction Score: After playing for 2 hours, users will complete a questionnaire to provide subjective evaluations of the relevance and satisfaction of the recommendations.

We believe Purchase Rate is the most important indicator. For a recommender system with commercial value, the ability to drive purchases is the top thing. Even if users regret the purchase later due to personal reasons, the recommender system itself should not be blamed.

Since the system adopts a collaborative filtering-based matrix factorization method, the recommendation process only requires computing the dot product between user and game latent vectors. Therefore, once the model is trained, generating recommendations for each user incurs relatively low computational cost. However, as our dataset is large, the training process itself is computationally intensive. To balance performance and efficiency, we schedule the model to retrain weekly to accommodate newly added games and updated user-game interaction data.

Regarding the design of a user study:

We will build a simple web-based interface to simulate the real scenario of users using the recommender system.
We will invite more than 100 Steam users and obtain all the games from their Steam libraries to build a small rating vector. Each user will be shown 10 recommended games, ranked by model prediction.
For each game, users can indicate whether they are interested or not.

Two weeks later, we will ask users to fill out a short questionnaire to assess their satisfaction with the recommendation model and the overall system.

We will prioritize feedback from users who purchased the recommended games and played for more than 2 hours, as their feedback is more reliable. However, we will also take into account the responses from other users, though with lower weight.
(Note: Purchasing a game is not mandatory, and since Steam has a refund policy, users can get a refund within the stipulated time if they change their mind.)