# Steam Game Recommendation Prediction


## ABSTRACT

I select the Review Data and User and Item Data from the Steam Video Game and Bundle Data and use 6 models to predict the recommendation.


## 1. DATASET ANALYSIS

The dataset I chose is divided into two parts: Review Data and User and Item Data. The Review Data includes the user ID, item ID, user's review of the game, whether they recommend the game, and the time of the review. As shown in Table 1, the total number of reviews in the reviews data is 5,905, the number of recommended reviews is 52,473 (accounting for 88.5%), and the number of not recommended reviews is 6,832 (accounting for 11.5%). The Review Distribution is shown in Figure 1. It is evident that there is an imbalance in the data, with a disproportionately large number of positive samples and too few negative samples. Therefore, I sample additional negative samples to balance the data distribution. The specific method is detailed later in the text.

**Table 1. Reviews Distribution**

| Total Reviews | Recommend | Not Recommend |
|---|---|---|
| 59,305 | 52,473 | 6,832 |

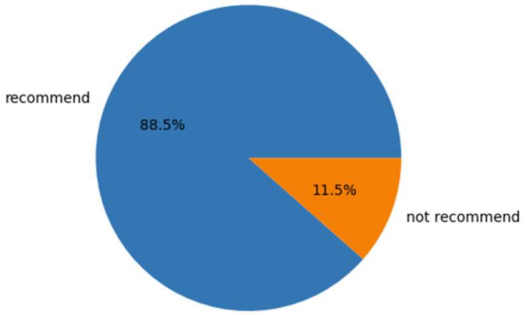


**Figure 1. Review Distribution.**

As a game player, I want to know which games are recommended the most and which games are not recommended the most. Therefore, I conduct a further exploration of the Review Data. The results are shown in Figure 2 and Figure 3. The top 3 most recommended games are item ID 440 (3,611), Counter-Strike: Global Offensive (3,478), and Garry's Mod (1,685). And the top 3 not recommended games are Counter-Strike: Global Offensive (281), PAYDAY 2 (227), and DayZ (162).

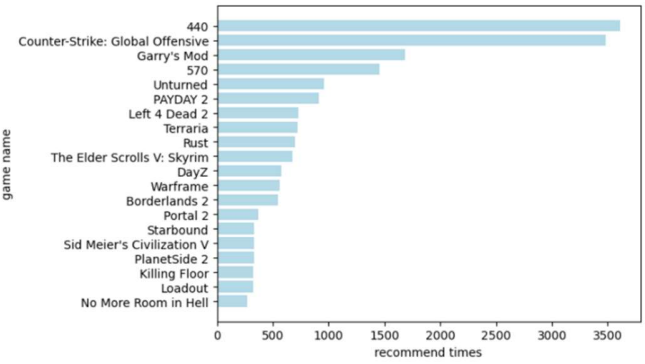


**Figure 2. Most Recommended Games.**

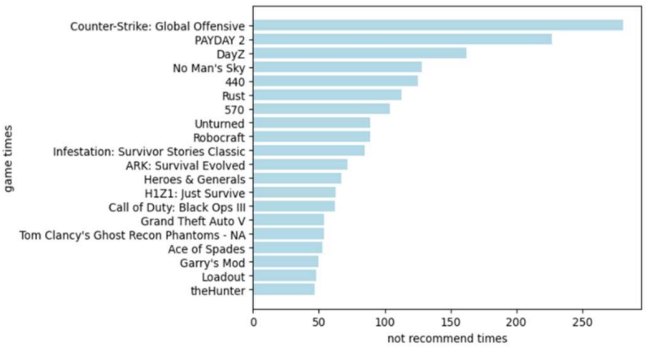


**Figure 3. Most Not Recommended Games.**

In the process, I find an issue. The game names are extracted from the User and Item Data; however, some item IDs in the review data can not be matched with corresponding item names in the User and Item Data. As a result, the Review Data and the User and Item Data are not fully aligned. I address this issue in the subsequent experiments.

The User and Item Data includes the user ID, the number of games owned by the user, the games owned by the user, the item ID, playtime forever, and playtime in the last two weeks of each game. In this dataset, the user with the most games owns 7,762 games (1 player), while the user with the fewest games owns 0 games (16,806 players). The total number of users is 88,310, as shown in Table 2.

**Table 2. User Distribution**

| Total Users | Own 7,762 games | Own 0 game |
|---|---|---|
| 88,310 | 1 | 16,806 |

The distribution between users own 0 games (accounting for 19%) and other users (accounting for 81%) is shown in Figure 4.

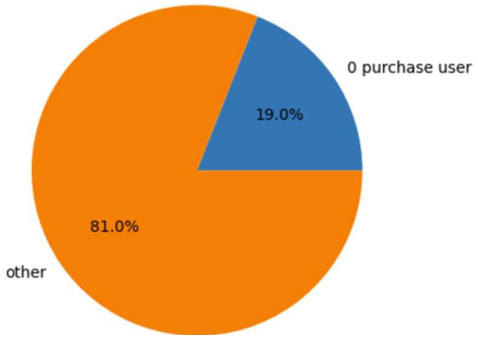


**Figure 4. User Distribution.**

I also find that the total number of games owned by all users is 5,153,209. Among these, the number of games with recorded playtime is 3,285,246, while the number of games purchased but never played is 1,867,963.

In all these games, the top 3 most purchased games are Dota 2 Test (49,571), Counter-Strike: Global Offensive (43,776) and Garry's Mod (43,301). As shown in Figure 4. The total number of games purchased by all users is 5,153,209.

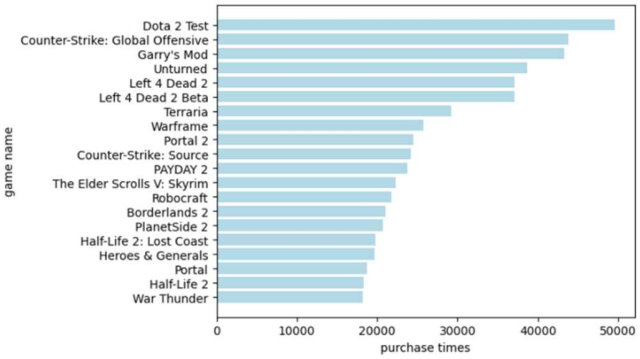


**Figure 4. Most Purchased Games.**

The top 3 games with the longest playtime forever are Counter-Strike: Global Offensive (785,184,267), Garry's Mod (448,366,616), and Terraria (154,974,541). As shown in Figure 5 and the total playtime forever among all games is 5,109,381,683.

The top 3 games with the longest playtime in the last two weeks are Counter-Strike: Global Offensive (8,163,312), Garry's Mod (1,886,608), and Rocket League (1,760,902). As shown in Figure 6 and the total playtime last 2 weeks among all games is 46,918,458.

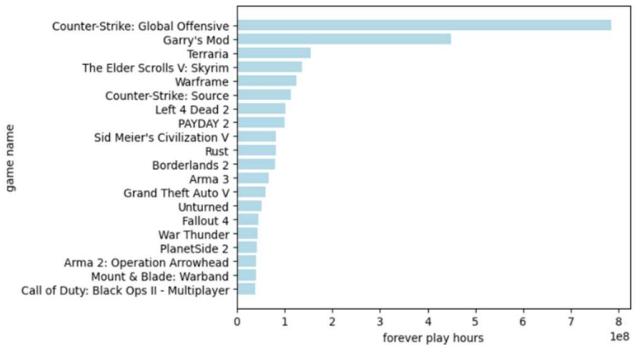


**Figure 5. Playtime Forever Distribution.**

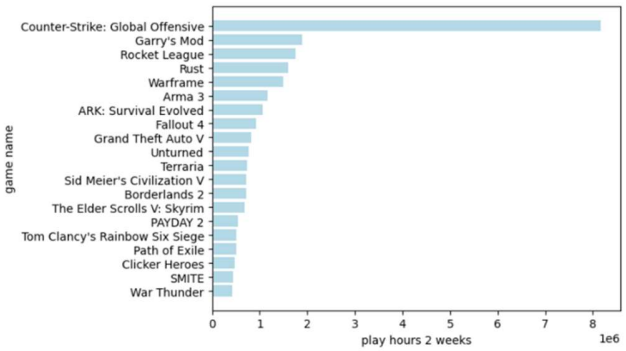


**Figure 6. Playtime Last 2 weeks Distribution.**

## 2. PREDICTIVE TASK

The task I chose to work on is to predict whether a user will recommend a game or not. From the Review Data and User and Item Data, I selected the following as features: user ID, item ID, user's playtime forever, and playtime in the last week. For the label, I used whether the user recommends the game or not.

Initially, I encounter issues while reading the data. Since the Review Data and User and Item Data are not in a standard JSONL format, I keep getting errors when using json.load due to formatting issues related to single quotes (') and commas (,). Then I try using ast.literal_eval, and it solves the problem.

When constructing the dataset, because that the content of the Review Data and User and Item Data is not entirely consistent. Therefore, I only include data where both the user ID and item ID appear in both the Review Data and the User and Item Data in my dataset.

Next, I address the issue of data imbalance mentioned earlier. I adopt the sampling method I have previously used in Homework 3 and Assignment 1. Specifically, I first randomly select users from the dataset and then, from the User and Item Data, randomly select games owned by these users where the playtime forever is 0

to serve as negative samples. I set the playtime forever and playtime last 2 weeks of these samples to 0 and mark their recommend value as False. The total amount of these negative samples are 29,869. And, after I adding these samples, the total samples in the dataset is 74,759.

## 3. MODEL SELECTION

When selecting and building a model, I first attempt to identify which games are popular. If a user's reviewed game belongs to the set of popular games, I predict that the user will recommend it; otherwise, I predict it as not recommend. This approach is similar to what I use in Assignment 1 and Homework 3. However, the performance of this method is poor, with accuracy just slightly above 50%, so I abandon it and try other models.

In total, I try 6 models, which are the ones I have learned in class: Collaborative Filtering Model (using Jaccard Similarity), Latent Factor Models (SVD++), Gaussian Naïve Bayes, Logistic Regression, Random Forest and Deep Neural Network. And I set Collaborative Filtering Model the baseline model.

### 3.1 Collaborative Filtering Model

In the Collaborative Filtering Model, I use the data consisting of userID, itemID, and whether recommend, without including playtime. For a given pair of user and item, I divide the users who have reviewed the same item into two categories: recommend and not recommend.

Within each category, I find the user with the highest Jaccard similarity to the target user. I then compare the scores of these two users. If the user from the recommend category has a higher score, I predict that the target user will recommend the item. Otherwise, if the user from the not recommend category has a higher score, I predict that the target user will not recommend the item.

$$s_{\text{rec}} = \max_{u' \in U_{\text{rec}}(i)} J(u, u')$$

$$s_{\text{not\_rec}} = \max_{u' \in U_{\text{not\_rec}}(i)} J(u, u')$$

$$If\, s_{\text{rec}} > s_{\text{not\_rec}}, predictR(u,i) = 1$$

$$If\, s_{\text{rec}} \leq s_{\text{not\_rec}}, predictR(u,i) = 0$$

### 3.2 Latent Factor Models

In the SVD++ model, I utilize a dataset that includes three key components: user ID, item ID, and a binary indicator of whether the user recommends the item or not. Notably, I do not incorporate additional information such as playtime in this model. For the recommendation data, I set the rating scores to a binary scale of 0 and 1, where a score of 0 indicates that the user does not recommend the item, and a score of 1 signifies that the user does recommend the item. This binary setup simplifies the modeling process and directly reflects the recommender's decision.

The prediction function used in this SVD++ model is detailed below, showcasing how the user-item interactions are processed to predict the recommendation score based on the available data.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j\right)$$

$\mu$ (Global Average): The average of all recommend in the dataset.

$b_u$ (User Bias): Captures how much the user u tends to recommend higher or lower than average.

$b_i$ (Item Bias): Captures how much the item i tends to receive higher or lower ratings than average.

$q_i^T$ (Item Latent Factor Vector): A latent feature vector representing item i.

$p_u$ (User Latent Factor Vector): A latent feature vector representing user u.

N(u) (Set of Items Interacted by User u): The set of items that user u has interacted with (e.g., rated, viewed, or purchased).

$y_j$ (Implicit Feedback Vector): A latent vector representing the implicit influence of item j on user u.

The loss function is:

$$\mathcal{L} = \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\| \boldsymbol{q}_i \|^2 + \| \boldsymbol{p}_u \|^2 + b_u^2 + b_i^2 + \sum_{j \in N(u)} \| \boldsymbol{y}_j \|^2)$$

$\lambda$ is the regularization term to prevent overfitting.

### 3.3 Gaussian Naïve Bayes

In the Gaussian Naïve Bayes model, I utilize a dataset that consists of four main components: user ID, item ID, playtime forever, and an indicator of whether the user recommends the item. Among these features, playtime forever is particularly important as it represents the total amount of time a user has spent playing a particular game, which can serve as a valuable predictor for whether the user is likely to recommend it. Additionally, the binary recommendation indicator is used as the target variable, where a value of 0 represents that the user does not recommend the game, and a value of 1 indicates that the user does recommend it.

One notable detail in this model is that I deliberately exclude the feature playtime last 2 weeks from the dataset. This decision is based on experimental results, which showed that including playtime last 2 weeks actually reduced the model's accuracy. This decrease in accuracy could be due to noise or inconsistencies in short-term playtime data, making it less effective as a predictor compared to the total playtime (playtime forever). As a result, for all models that involve using playtime as a feature, I consistently avoid using playtime last 2 weeks to ensure better performance and more reliable predictions.

### 3.4 Logistic Regression

In the Logistic Regression model, I utilize a dataset that includes the following key components: user ID, item ID, playtime forever, and an indicator of whether the user recommends the item. This dataset allows the model to predict the likelihood of a recommendation based on the total amount of time a user has spent playing a specific game (playtime forever) and the user-item interaction information. The recommendation indicator serves as the target variable, where a value of 0 indicates that the user does not recommend the game, and a value of 1 signifies that the user does recommend it.

Logistic Regression is a relatively simple and interpretable model, making it a good baseline for classification tasks such as this one. To control the complexity of the model and prevent overfitting, I set the regularization parameter C to 1. This parameter determines the strength of the regularization, with smaller values of C leading to stronger regularization and larger values allowing the model to fit the training data more closely. By setting C to 1, I aim to strike a balance between underfitting and overfitting, ensuring that the model generalizes well to unseen data.

Overall, this Logistic Regression model provides a straightforward approach to understanding how features like playtime forever influence the likelihood of a user recommending a game, while the inclusion of regularization ensures that the model remains robust and avoids overfitting to the training data.

### 3.5 Random Forest

In the Random Forest model, I use a dataset that consists of the following key features: user ID, item ID, playtime forever, and an indicator of whether the user recommends the item. The feature playtime forever plays an important role in capturing the total time a user has spent playing a particular game, which can provide meaningful insights into their likelihood of recommending it. The binary recommendation indicator is used as the target variable in this classification task, where a value of 0 represents that the user does not recommend the game, and a value of 1 indicates that the user does recommend it.

The Random Forest Classifier is a powerful ensemble learning method that builds multiple decision trees during training and combines their outputs to make robust predictions. For this model, I have carefully set the hyperparameters to achieve a balance between accuracy and computational efficiency. Specifically, the Random Forest is composed of 100 decision trees, which provides a strong ensemble effect while maintaining reasonable computational performance. Additionally, the maximum depth of each tree is limited to 5 levels, which prevents the trees from growing too deep and overfitting to the training data. By restricting the depth, the model focuses on learning general patterns in the data rather than memorizing it. To ensure reproducibility of the results, I set the random seed to 42, which controls the randomness involved in the bootstrap sampling and feature selection processes during tree construction.

### 3.6 Deep Neural Network

In the Deep Neural Network model, I use a dataset that consists of the following key features: user ID, item ID, playtime forever, and an indicator of whether the user recommends the item.

I have constructed a simple deep neural network with a total depth of 3 layers. The network includes two hidden layers, each containing 16 neurons, which ensures that the model has enough capacity to learn meaningful patterns without being overly complex. The final layer outputs a single value (logit) that represents the model's prediction of whether the user recommends the game. To process the input data, each layer applies Linear transformations. After each layer, I apply the ReLU activation function, which introduces non-linearity into the network. This non-linearity is crucial for enabling the model to learn complex patterns and relationships in the data that cannot be captured by purely linear transformations.

For training the model, I use the Binary Cross-Entropy with Logits Loss function as the loss function. This loss function is well-suited for binary classification tasks, as it combines the Sigmoid activation function with Binary Cross-Entropy loss in a numerically stable way. It calculates the difference between the predicted probabilities and the actual labels, guiding the model to make more accurate predictions. To optimize the model's parameters, I use

the Adam optimizer, which is an adaptive optimization algorithm that adjusts the learning rate for each parameter dynamically. Adam is widely used in deep learning due to its efficiency and ability to converge quickly.

## 4. RELATED LITERATURE

The dataset of Steam Video Game and Bundle Data was used in the paper Generating and Personalizing Bundle Recommendations on Steam[1]. However, the task I chose to work on is different from the task described in the paper. The tasks chosen in this paper are divided into two main steps. First, the authors use Bayesian Personalized Ranking (BPR) to generate an item ranking for each user based on the items they have previously purchased. Then, in the second step, they utilize a Greedy Algorithm to generate personalized bundles (collections of items) for users by leveraging the item ranking derived in the first step.

I feel that this task is a bit too complicated, so I chose to focus on the prediction and recommendation task instead. However, this paper was still helpful to me. For example, in the Steam Video Game and Bundle Data, there is an excessive amount of negative data, so it is necessary to sample some negative samples. The paper uses a graph sampling algorithm specifically designed for Bayesian Personalized Ranking (BPR), which I did not adopt. Instead, I used a method similar to the one applied in Assignment 1 and Homework 3 to sample negative samples.

Moreover, in the task I am working on, I find the Steam Video Game and Bundle Data to be similar to the Goodreads Book Reviews. The 'recommend' in the Steam Video Game and Bundle Data corresponds to the 'rating' in the Goodreads Book Reviews. Both datasets include user IDs and item IDs, and the Steam Video Game and Bundle Data additionally provides playtime, which allows me to explore more methods for prediction.

## 5. RESULTS AND CONCLUSION

The evaluation metrics I have chosen for assessing the model's performance include Precision, Accuracy, Recall, F1 Score, and the Area Under the Curve (AUC) value.

Precision evaluates the proportion of correctly predicted positive instances among all predicted positives, focusing on the accuracy of the model's positive predictions.

Accuracy measures the overall correctness of the model by calculating the proportion of all correctly classified instances among the total instances.

Recall, also known as sensitivity, assesses the model's ability to correctly identify all actual positive instances.

F1 Score combines Precision and Recall, providing a balanced evaluation when there is a trade-off between these two metrics.

Lastly, the AUC value evaluates the model's ability to distinguish between classes across various thresholds, giving a global metric for assessing its discriminatory power.

The results are shown in Table 3:

**Table 3. Model Evaluation**

| | Precision | Accuracy | Recall | F1 Score | AUC |
|---|---|---|---|---|---|
| **CF** | 65.33% | 68.39% | 85.74% | 74.15% | 67.32% |
| **SVD++** | 78.51% | 77.63% | 84.05% | 80.71% | 78.10% |
| **Gaussian** | 91.61% | 56.45% | 19.56% | 32.10% | 58.73% |
| **LR** | 92.26% | 73.10% | 53.65% | 67.84% | 74.30% |
| **RF** | 88.23% | 92.86% | 99.81% | 93.66% | 92.43% |
| **DNN** | 90.31% | 91.23% | 93.45% | 91.85% | 91.10% |

Among these models, what surprised me the most are GaussianNB, Random Forest Model, and Deep Neural Network. The Collaborative Filtering Model performed well enough as the baseline model. SVD++, being more complex, naturally delivered better performance, which is expected. The poor performance of GaussianNB may be due to the randomly generated negative samples, which causes the dataset to deviate from the normal distribution assumption. Additionally, there might still be an issue of class imbalance in the dataset, which could also negatively impact the model's performance. The reason why the Random Forest Model and Deep Neural Network performs exceptionally well might be due to the negative samples I generated, where the 'playtime forever' feature is always 0. These two models were better at capturing this characteristic, especially since only a single feature was used as input. However, this may not reflect real-world scenarios, where a user might play the game for some time before deciding to give a 'not recommend' review.

## 6. REFERENCES

[1] Apurva Pathak, Kshitiz Gupta, and Julian McAuley. 2017. Generating and Personalizing Bundle Recommendations on Steam. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17). Association for Computing Machinery, New York, NY, USA, 1073–1076. https://doi.org/10.1145/3077136.30807