Name:Akshay Kumar Gyara
SID: 013836277

Game Recommendation System for Steam

Computer Engineering Department, San Jose State University, CA
akshaykumar.gyara@sjsu.edu

Abstract:
Steam is the largest digital distribution platform for PC gaming which houses games from huge publishers as well as from indie developers. A game recommendation system can increase the commercial revenue of the platform as well as connect gamers to niche games. In this project, we build and evaluate multiple recommendation models based on a rating implicitly derived from the playtime of a game. We developed two types of recommendation models that improve the same business objective in different ways. We can improve these models further by collecting more data as well as incorporating additional information like genre, publisher, price. We also present our analysis on data understanding and preparation to build these models.

Introduction:

Steam is a digital distribution platform and marketplace where users can purchase and play games. Our goal is to develop a recommendation system which suggests top-k unplayed games to a user based on the user's personal gaming history and the preferences of similar users. This can improve sales by increasing conversion rates and decreasing the spends on marketing and promotional activities. This also attracts more users because good recommendation systems tend to increase users' engagement on the platform. From a business perspective, we can set two different goals that a game recommendation system should achieve. One goal is to recommend top-k games to a user in decreasing order of liking. This implies that user is likely to rate ith game higher than jth game, if ith game appears first in the recommendation of top-k games. Such a recommendation system places emphasis on the likings of the user and encourages a user to buy the top-k games thus increasing the business revenue. Recommendations from this model belong to section 'Games you might like'.

In this case, we build continuous rating models with the target variable being a continuous rating rui (rating of ith game by the uth user). In data science terminology, the goal of the continuous case is to predict ˆ rui for any user u and $\forall$ i $\in$ items and select top-k ratings. The goal of the binary case is to predict $\forall$ i $\in$ items, the probability Pr(Buyui = 1) of the user u buying the game i and selecting the top-k probabilities for recommendation.

Data Collection:

Steam provides a Web API interface [1] to collect game level and user-level data. Every user on the platform is assigned a unique SteamID and similarly every application is assigned a unique AppID. These API calls return rich user-level and game-level features like game playtime of users, friends linked to a user, genre of the game, publisher of the game etc.

To build a recommendation system, we require welldefined ratings rui. However, Steam does not maintain explicit ratings for its content. Given that game playtime is usually a good indicator of a user's

engagement in a game, we decide to use game playtime feature as an implicit rating. For the binary case, we require data indicating whether the user has bought the game or not. A playtime value of non-zero is equivalent to the user buying the game and thus we can derive the binary target1variable from the implicit ratings.

With the API call GetOwnedGames which takes SteamID as input, we collect the number of games owned by the user along with the playtime of every game owned by the user. However, there is no API call to collect SteamIDs and thus we resorted to scraping SteamIDs from a large discussion group on the Steam platform called 'Steam Universe' [2]. Naturally the users on this group don't represent the full population and we discuss introduction of biases in the next section. We scraped 50000 unique SteamIDs from the group webpage and then extracted the playtime data from the API.

Data Analysis and Feature Understanding:

An important characteristic for recommendation system data is the density / sparsity of the user x item rating matrix Rui. A dense matrix carries more information that can used for recommendation. Given that a user is very unlikely to interact with a lot of games, we expect our matrix to be sparse. Figure 1 shows a distribution of unique AppIDs per user.

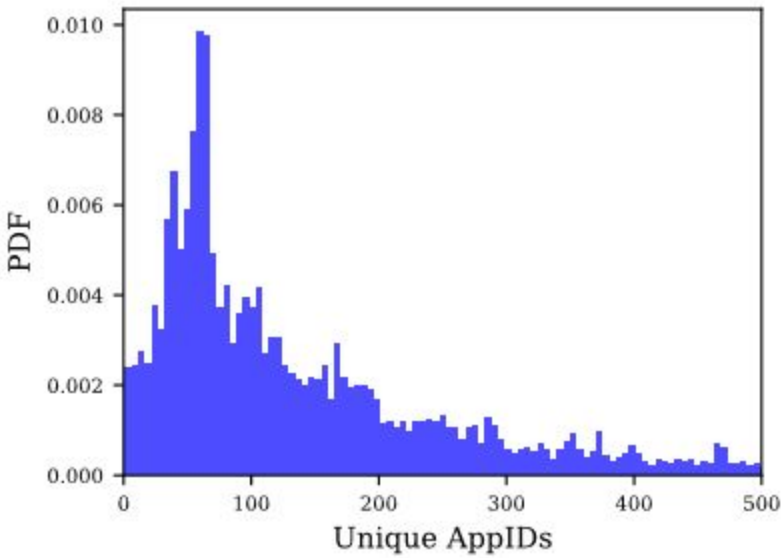

Figure 1: Distribution of AppIDs per user

We observe that most of the users have few games in the library compared to the total unique AppIDs indicating that the matrix is sparse. We also observed the playtime distribution of most AppIDs is skewed to the left (with more users having a relatively smaller playtime). Figure 2 shows one such distribution.

Name:Akshay Kumar Gyara
SID: 013836277

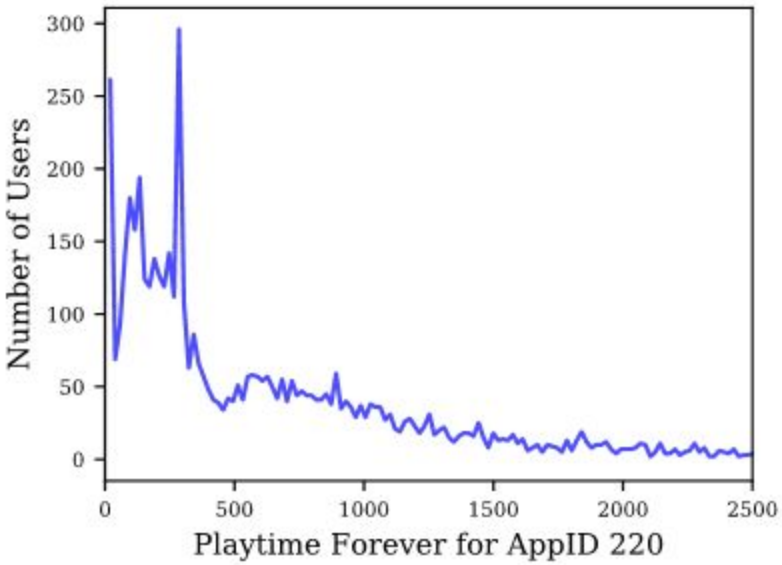

Figure 2: Playtime distribution for AppID 220

Figure 3 show the corresponding cumulative distribution function of the playtime for AppID 220.

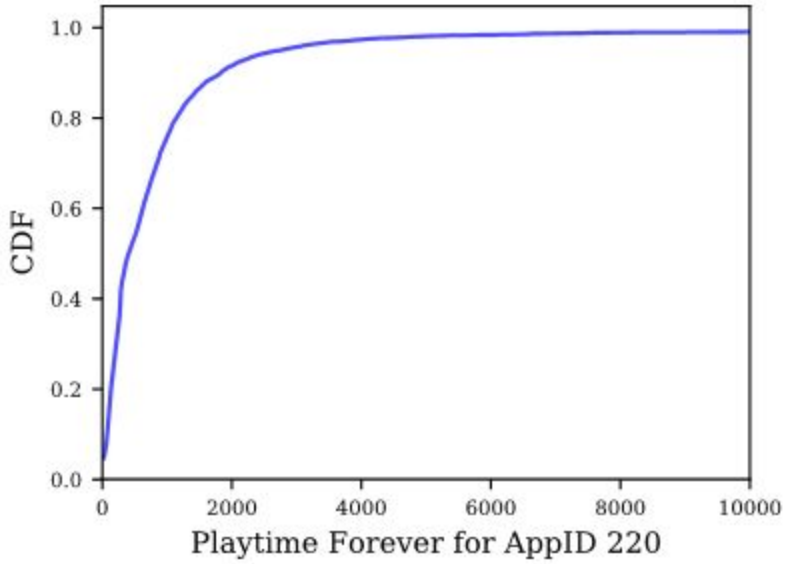

Figure 3: Playtime empirical CDF for AppID 220

The empirical cumulative distribution function also allows us to map any arbitrary playtime distribution to a 0-1 scale. We make use of this to construct our continuous rating scale. Additionally, this allows to bring games with diverse playtime distributions on the same footing.

3.1 Data Cleaning:

The dataset fetched by scraping the API had two issues :

1. Empty data - API call returned empty dictionary

Name:Akshay Kumar Gyara
SID: 013836277

2. Zero game count data - API call returned zero games owned

We remove such entries since they are non-informative from a modeling perspective (cold start problem - since model can't learn about a user who hasn't played any games). Removing such entries naturally omitted a lot of SteamIDs and left us with data in the form of tuples (SteamID, AppID, Playtime) where playtime is non-negative. We also decided to treat zero-playtime SteamID-AppID pair as equivalent to the user not having that game in his/her library and thus remove them from the data. This also helps us maintain smaller sized data without losing any information. This cleaning resulted in a user x item matrix with density 0.99%.

Sampling: We assume that games with a small playerbase are not useful for recommendation and thus discard them. Similarly, we assume that players that only small number of games in the library are also non-informative and we discard them. Also, this helps us in increasing the density of the matrix without losing much of the original data. Broadly, we define two criterias based on which we discard any data

1. Discard an AppID if it has less than c1 unique players
2. Discard a SteamID if it has less than c2 unique games played

We determine cutoffs c1 and c2 by the distribution of SteamID counts and AppID counts respectively. To determine c1, we plot the number of AppIDs against unique SteamIDs shown in Figure 4.

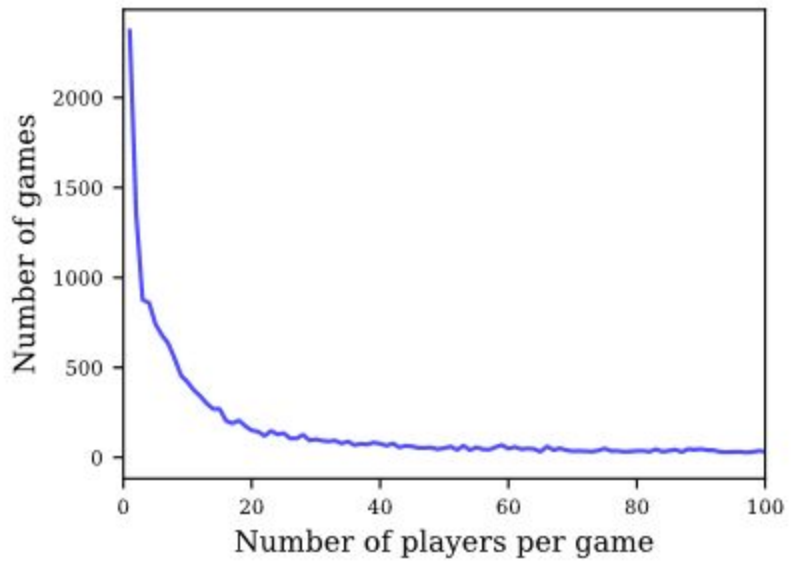

Figure 4: Data point (m, n) corresponds to there being n games that have exactly m players

We target increasing density without losing too much information. Based on the Figure 4, we choose the cutoff c1 = 10 that best achieves this purpose. By removing these sparse games, the density of the new data improved to 1.65%.

To determine the cutoff c2, we plot the distribution of AppID counts in Figure 5.

Name:Akshay Kumar Gyara
SID: 013836277

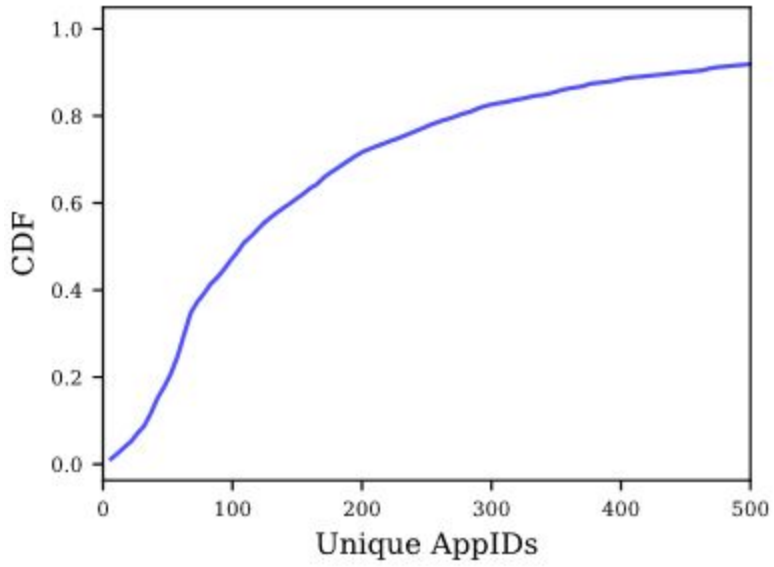

Figure 5: Empirical CDF of AppID counts (x-axis is cutoff at 500)

We determine c2 to be 65, the reason being that the distribution of AppID counts starts to increase rapidly af

3

ter 65 (users with number of unique games less than 65 are rare). We want to discard such users because they are quite sparse compared to the rest of the data and result in negligible data loss but noticeably increase the density. After this data cleaning process, we are left with 13,845 unique SteamIDs and 12,982 unique AppIDs with a density of 1.84 %. We use this data to build our recommender models.

Feature Engineering:

We observed that there is a lot of variation of playtime among different games. Given the diversity of games in terms of genres and game content, we expect such variation. For example, a story-based game which is typically completed in 20 hours will have its playtime distribution on a different scale to match-up based games where there is technically no completion time. To be able to treat such diverse games on the same footing, we transform the playtime to the same scale by using the empirical cumulative distribution function of playtime. For each game, we calculate empirical cumulative distribution of playtime and and transform it to 0-1 scale by evaluating it at that point. Empirical cdf is defined as

$$F_j(x) = \frac{1}{N} \sum_{x_i \in R_j} I\{x_i \leq x\} \quad (1)$$

where Rj is the jth column of R (corresponding to jth app) and N is the number of non negative entries in Rj

For example, if the playtime of appid 220 ranges from 1 minute to 10000 minutes, then we will convert 1 minute to value 0 and 10,000 minute to value 1 (See the figure - CDF of PlaytimeForever for Appid 220)

Name:Akshay Kumar Gyara
SID: 013836277

Recommendation Models:

For this project we have chosen **Matrix factorization** as the recommendation model which is a class of Collaborative filtering. The algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. The goal is to decompose user x item rating matrix R into two matrices P (users x latent factors) and Q (latent factors x items).We also assume that users and games have inherent bias (bu and bi respectively) and we treat this as parameter in our model. The equation for predicted rating becomes

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u$$

where μ is global mean, bu is bias of uth user, bi is bias of ith item, qi and pu are ith column and uth row of Q and P matrix respectively. These parameters are obtained by minimizing the following loss function

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

λ being the regularization parameter.
We use SGD as optimization technique to find optimal parameters. We use surprise library [3] for the implementation of the model.

Performance and Evaluation

For testing the recommendation engine the authors used random uniform sample of 2000 unique users who have purchased and played minimum of “d” games in other to reduce intrinsic randomness of recommending a single game based on a single feature. We made sure that all 2000 users selected for testing are unique. Then we selected “n” random game for every user in the testing group and altered their behavior by removing those “n” game purchase from the main matrix R. Then the authors checked, if the removed games appeared in the recommended set of “f” games. Then the mean of all of the recalls has been computed to arrived at global recall. Finally through hyper-parameters (k, n, d, f) tuning the authors arrived at global recall of 28% by setting k=101, n=2, d=10, f = 100 . (k value of 101 allows for rank of 23).

Name:Akshay Kumar Gyara
SID: 013836277

Results:

And the below is Evaluation result of the model.

```
def getRecall(games_altered, test_users, data, num_recommendations, predictions, user_id_groups):
    recalls = []
    for i in tqdm(range(0, len(test_users),1)):
        matches = 0
        test_user_id = test_users[i]
        recommendations = recommend_games(user_id, num_recommendations, predictions, user_id_groups)
#         print(games_altered[i],recommendations)
#         break
        for game in games_altered[i]:
            if game in recommendations:
                matches+=1
        recall = matches/len(games_altered[i])*100
        recalls.append(recall)
    return np.mean(recalls)
```

```
print(getRecall(games_altered, test_users, beautiful_df, 1000, predictions,user_id_groups))
```

```
100%|██████████| 1000/1000 [00:44<00:00, 25.14it/s]
78.8
```