Generating User-Similarity-Based Bundle Recommendations on Steam

Dzhangir Bayandarov dbayanda@ucsd.edu
Rohith Pillai rpillai@ucsd.edu
Theodore Alo tjalo@ucsd.edu

Abstract

Throughout the CSE 158 course we have learned ways to apply recommender systems to various datasets. For assignment 2, we have decided to use the steam dataset from the provided datasets. We will address the issues concerning recommendation models that would provide steam games to users based on their tastes and similarities to other users. To create effective recommendations we find the user's 10 most popular games and find similar users that play those games. We then recommend a game that our user has not played, that is also played by a user with the highest similarity. We assess different issues such as splitting up the data into training and testing and creating valid ways to evaluate the recommender model to avoid bundles that the user purchased before.

2. Related Work / Literature

Valve has recently (in 2020) created the Interactive Recommender which is their machine learning model currently used to recommend new games to Steam users. Valve describes the Interactive Recommender to take into account 2 main criteria: the amount of hours a user has played a particular game and the games of users that have "similar play habits".[1] The former, from Steam's description, is most likely some regressor/classifier that uses the amount of hours as a feature to predict if a user would buy a particular game or not. As for the latter, this approach sounds like a recommender system that uses collaborative filtering based on the games of other similar Steam users. This approach is similar to our model in that we use the Jaccard similarity between a user's set of games and all other users' set of games and based on the most popular/most played game amongst the most similar users, this game is recommended to the user.

Another task on this dataset was using a model that incorporates the Bayesian Personalized Ranking to recommend and generate bundles of games to a user. In a paper written by Julian McAuley and others about the same dataset used here, BPR was used in a model that not only recommended existing bundles but also generated bundles of games that a given user would most likely buy.[2] This BPR model was first trained by a user's preferences for individual games and then these parameters were adapted to learn a user's preferences for groups of items/ bundles of games. Based on the compatibility of a user and a group of games, the model would recommend/ not recommend a bundle to a user.

Instead of adapting the aforementioned BPR model to groups of items, we could take the already trained model on individual games and use this model to recommend single games to users. Each latent factor in the BPR model could be built with the dataset using the set of users who have bought a particular game and the set of games a particular user has bought.

Another approach to recommending games can make use of deep neural networks as was done in a paper by researchers at a Japanese video game company called Silicon Studio.[3] These researchers would feed the deep neural network feature vectors that not only included a set of games a user played and the hours they played those games but also temporal data such as the game they played most recently. This resulted in a video game recommendation that was both similar to a user's set of games and also relevant to the games they were currently playing.

3. Datasets

The first dataset we are using includes data about the Steam market in Australia called australian_user_items. We studied a set of Steam users and the games they have purchased. For each user, there was information about their username, the amount of games they played, and the specific games they played along with their play times. We also created a subset of this dataset that only stored the top 10 most played games per user by hours so we could get a better representation of the actual games users' enjoyed.

User Data Set (australian_user_items)

Users	25,799
Games	10,947
Average Amount of Games Per User	53.35
Average Amount of Games Per User Actually Played*	37.20

*The user player at least played 1 minute of the game

Though the average amount of games per user seems high, a handful of users did not actually play the games they have purchased. Out of the ~53 games most user's have purchased, only ~37 games were actually played by the user. This would suggest that some games were purchased from a bundle and only a subset of those games in that bundle were actually played.

Top 5 Most Purchased Games

Game	Amount of Purchases
1. Dota 2 Test	49571
2. Counter Strike: Global Offensive	43776
3. Unturned	43301
4. Left 4 Dead 2 Beta	38682
5. Left 4 Dead 2	37044

It should be of no surprise that 4 of the top 5 games are Valve titles.

The second dataset we studied was the steam_reviews, a massive 1.3 gigabyte dataset that included over 7 million reviews. Each review included a user, the amount of hours played on the game they reviewed, the id of the game they reviewed, and the actual text of the review. The length of the text of the review ranged from comedic, sentence comments to essay length, detailed descriptions of their time playing the game. Below are some basic statistics on the steam_reviews dataset.

User Reviews (steam_reviews)

Games	15,474
Users	7,793,069
Avg. hours played	111.46
Avg. review length	337.16 characters

The previous table suggests that users, on average, will play over 100 hours before reviewing a game. And the reviews themselves are often very small at around ~300 characters total which seems to indicate reviews are often 2-3 sentences long and often tibbits of the actual game.

4. Problem Formulation

Before describing our approach to game recommendation, we introduce some basic notation. Let U = {u1, . . . ,u|U | } be a set of users, I = {i1, . . . ,i |I | } a set of items(games) and B = {b1, . . . ,b|B | } a set of bundles, such that each bi $\subseteq$ I.

5. Game Ranking

We utilize the 'australian_users_items' data to create a list of users that play a certain game. We then only consider users who have played more than one game since those are the ones relevant to the recommendation model. We split the data into 70% training, 30% testing. We proceed to find the top 10 (capped, in instances users have played less we take all) games users have played by hours played.

5.1 Jaccard User Similarity

The Jaccard similarity takes a value between 0 (when Ui and Uj do not overlap at all, and thus have no intersection) and 1 (when the intersection is equal to the union, i.e., the items were consumed by exactly the same set of users). Essentially, Jaccard similarity considers users with a lot of games in common to be similar. Based on the top-10 games of the current user, we find the Jaccard similarity for each user in the train dataset. We then sort the user similarity scores and select the 5 most similar users. From those users we pick the game that the current user has not played and recommend that game.

5.2 Pearson User Similarity

There are big limitations that arise in Jaccard similarity. Since users that play similar games may not necessarily have the same sentiment towards them. We have used Pearson similarity using the hours spent playing each game as a metric of how much users enjoyed it. Thus, the similarity evaluation was based on how many games users had in common and how many of those games they have played the most. Pearson similarity allowed for a more weighted approach to finding similar users.

6. Evaluation

We will be evaluating our model using precision @ k and also recall @ k. With precision, we will measure how many k games were "relevant" to the games in a user's set of games. In our case, relevance refers to the recommended k amount of games actually purchased by a user. So in turn, we are dividing the number of games that were both in the recommended games set by the number of games in the recommended games set. We can do the same type of evaluation using recall @ k except dividing the intersection of games purchased by a user and recommended by the number of games purchased by a user.

Precision @ k:

$$\frac{\text{Games purchased by user} \cap \text{k amount of games recommended}}{\text{k amount of games recommended}}$$

Recall @ k:

$$\frac{\text{Games purchased by user} \cap \text{k amount of games recommended}}{\text{k amount of games recommended}}$$

We can compare these evaluation metrics with the baseline model which simply recommends games in the top percentile of the most popular games.

7. Models

To make recommendations, we developed 2 different models: one using the austrailian_user_items dataset that has information about users and the items they bought, and one using the steam_reviews dataset that has 1.3 gigabytes of user reviews for games. The reason we went ahead with this model is that we wanted to perform an amazon-style recommendation technique for a user where we recommend

items the most similar users to a particular user bought, since amazon has had great success with this recommendation style.

Model 1

We started by creating train and test data by iterating through every user-items object and adding 70% of these items for that user to training as a user-items object and 30% of items for that user as a user-items object to test. Then, we iterated through the training data and ran a nested for-loop to find the top 5 most similar users to the current user by finding the Jaccard similarity between top 10 games (by hours played) for the current user, and top 10 games of all other users. The top 5 most similar users (by Jaccard) were assigned to each user.

To create recommendations for a user, we found all of the top 7 games played by the 5 similar users and created a pool of 35 games. Then, we removed all games in that 35 game pool that the user owns (based on training data). If the remaining games left exceeded 20, we randomly selected 20 games and recommended these ones. Otherwise (which was the large majority of cases), we recommended the top k games (where k<20).

To evaluate the recommendations, we computed the precision @ k (proportion of the recommendations that were contained in the user's bought items in the test set) and recall @ k (proportion of user's bought items in the test set that were contained in the recommendations) for each user's recommendations. We then averaged all of these precision @ k and recall @ k to evaluate the model.

Model 2

We started by creating train and test data by first extracting the relevant user-item objects only (users with at least 2 games reviewed) and split all the data into 70% training and 30% test, ensuring that all the users in test also belonged in training. Then, we created a "ratingDict" by making the assumption that the more hours a user played a game, the more they liked it. Over 50 hours was given a rating of 5, over 20 a rating of 4, over 8 a rating of 3, over 2 a rating of 2, and between 0-2 a rating of 1. Once this was done, we iterated through the training data and ran a nested for loop to find the top 5 similar users to each user in training by evaluating the Pearson similarity between every user and the current user by using these assigned ratings, and by utilizing dictionaries such as userAverages, itemAverages which had average rating for users/items. After this, we proceeded with the same recommendation technique for a user as in model 1, we created a pool of 30 games from 6 of each of those 5 similar users' favorites and removed all of the games owned by that user in train. Then, to evaluate our recommender, we computed our precision @ k and recall @ k just as in model 1 on the withheld test set.

Baseline

For the baseline, we found the top 20 most popular games in the entire dataset by hours played cumulatively between all users and recommended these for every user.

Model Validation

While we did not run into issues related to overfitting, since as demonstrated in the results, our model's performance was relatively weak. It was very difficult to scale the model to the entire data since it took us long to run it on our systems. The model used in comparison was the baseline. The values we tweaked along the way included the number of similar users we considered (cemented on 5 for both of the models) and the number of similar games from these users we took to create the game pool (35 for model1 and 50 for user2). After much experimentation, we decided that these values worked well for us since other choices (described in the results section) were not as successful.

Model Strengths: Highly personalized, lot of similar models have been successful (amazon, netflix), useful in the-real time since it can utilize new data to improve recommendations

Model Weaknesses: Takes LONG to train, has to be retrained each time when new data is added, doesn't consider user-item latent features (latent factor models).

Results

Unfortunately since our systems were modest, we could not run this algorithm on the entire dataset, but running model 1 on half of the dataset, and evaluating on test the average precision @ k is 0.08 and the average recall @ k is 0.04. Running model 2 on about 30000 users in the steam reviews data, our average precision @ k is 0.014 and recall @ k is 0.08. This was the best result we could get after adjusting the game pool to values in the range 20-50 (35 was optimal) and trying out the number of similar users compared from 3-10 (5 worked best). Other values for this didn't work as well for model 1. For model 2, the 50 game pool worked best, and the number of similar users that worked best was 5 (same as model 1). These results do not demonstrate that these are well performing models yet, but it could be an issue with the fact that we could not experiment with more data as our modest systems inhibited this. Additionally, while the precision @ k and recall @ k are not that high, our recommender system could still recommend games that users would enjoy since similar users enjoyed them too, even if they did not technically buy them yet- so this is something to consider as well. Our baseline, which recommended the top 20 games always, and used the same data as model 1, had average precision @ k of 0.05 and recall @ k of 0.07. Thus, precision-wise this isn't better performing than model 1 but has a better recall. For the reason as to why model 1 didn't perform incredibly well, we would cite that we couldn't use all the data so could not consider all potential similar users. Additionally, our recommender more likely made *more useful recommendations* than the baseline since they were personalized. Additionally, if we could make a timestep into the future, we might be able to make evaluations that might point to our 2 models being successful, as we collect data on if the users actually bought the recommended games or not- which would be a better test set than the one we hand-created.

References

[1] Introducing the Steam Interactive Recommender (Steam blog)

Steam Labs
Steam, 2020
[2] Generating and personalizing bundle recommendations on Steam
Apurva Pathak, Kshitiz Gupta, Julian McAuley
SIGIR, 2017
[3] A Machine-Learning Item Recommendation System for Video Games
Paul Bertens, Anna Guitart, Pei Pei Chen, and Africa Perianez
Yokozuna Data, Silicon Studio, 2018