Steam Game Recommender

Dataset

The data used for this project was provided by Professor Julian McAuley on his website (https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data). The data consists of item_user data, review data, item metadata, and bundle data. For the purposes of this project, only the item_user and item meta data were used.

steam_games

- app_name (string): app name for game
- title (string): full title of the game
- item_id (int64): id of game
- url (string): link to main page for game
- review_url (string): link for the game review
- genres (list): categories of the game
- tags (list): categories of the game. similar to genre but functions as hashtags
- specs (list): specifications, description of game
- release_date (string): date when game was released
- discount_price (float64): discounted price of game
- price (float64): listed price of game
- early_access (boolean): whether the user had early access to the game (can play before the official release date)
- developer (string): creator of the game
- publisher (string): publisher of game
- metascore (int64): a score of game

user_items

- user_id (string): primary key of the user
- steam_id (int64): steam id number. sometimes the same as user_id but not always. steam_id appears to always be numeric.
- user_url (string): link to steam user profile
- items (list): games owned by user
    - item_id (int64): id of game
    - item_name (string): title of game
    - playtime_forever (int64): number of minutes played since purchase
    - playtime_2weeks (int64): number of minutes played in the last two weeks

Data Normalization

The data provided came as JSON files. Both steam_games and user_items were indexed by user_id, containing list attributes such as genres, tags, and specs for steam_games and items for user_items. Therefore this data needed to be normalized before being converted to pandas dataframes. The list attributes for steam_games were one-hot-encoded, with each possible genre, tag, or spec having their own attribute. For cases where there were overlapping names for a genre, tag, or spec (e.g., a genre called Action and a tag called Action), the set intersection of their one-hot-encoded attributes were used instead. The list of games for each user in user_items were expanded such that each row contains a single (user, game) pair, storing its corresponding game's metadata and identifiers for the user.

Since there were over 10 million (user, game) pairs in user_items, its normalization could not be processed on memory alone. Therefore, the normalization process was split into 10 batches, each processing 1 million rows, storing them into csv files which were concatenated after normalization.

Dataset Imputation and Cleaning

After normalization, we converted our data from csv to pandas dataframes, and realized a lot of cleaning work needed to be done. While the user_items data was mostly clean, the steam_games and user_reviews data contained a fair amount of missing or unclean fields that needed to be fixed or dropped. Missing values for fields that were meant to be textual, such as developer and app_name were simply filled with empty strings. Missing values for game id, which numbered two cases in total, were dropped, since almost all other fields were also empty and no data could be scraped to retrieve them.

Scraping Price / Release Dates

Missing values within steam_games' price and release_date fields were retrieved from the game's store page available in the url field through web scrapers built with the beautifulsoup4 library.

Scraping missing prices revealed the messy structure of steam's store page's html source code. Price values were stored in different places and under different div tags and locations depending on whether or not there was a running discount, if there was a demo version available, if the game was in open beta, or if it was coming soon. Even after accommodating these different situations, some game prices we retrieved were simply gibberish. It was revealed to us that any arbitrary value could be put into the price field for a game, since a few games had prices values along the lines of "730640"(an arbitrary number) or "Try to survive the 5 first days on the island". We confirmed that these were the actual reported prices on steam by manually checking the store pages. These cases, which numbered in the tens, were dropped from the dataset, while the rest were stripped of their dollar signs and converted to float type. Alongside these, cases where no price value could be scrapped were also dropped, as seemed to correspond to games that were no longer purchasable, such as the original Grand Theft Auto which was taken off

steam and made free on their own website. It makes sense to remove these cases from our training data, since we would have no reason to recommend games you can't even purchase.

Imputating release dates on the other hand proved to be a lot less messy, at least in terms of scraping. Release dates for all games were stored within a single class attribute called date. Release dates that were available within a game's store page were retrieved as need be. Games which had no reported release date on their store pages revealed to mostly be games which were in development but never released, and therefore were dropped from the dataset.

Cleaning and Parsing

This brings us onto the next task, where we cleaned and parsed our unclean text fields. For fields which had fairly consistent structures such as user_review's helpful attribute ("73 of 96 people (76%) found this review helpful"), their information was simply parsed and converted with regex. For example, converting the given helpful example to a floating point percentage representing how many users found the review helpful. For fields which lacked a consistent format that could be parsed with regex, particularly the date fields in user_reviews and steam_games, we accommodated and parsed dates for the most common formats, while dropping the rest. Dropped values were either listed in foreign languages or listed as gibberish such as "When it is finished" or "When it's done!'. Within the common formats, we encountered the fact that certain reported dates contained only a year, or a year and a month. In order to convert these dates into proper datetime objects, we needed them to fulfill the format of "%Y-%m-%d", so we imputed the missing months and/or days. This was done by randomly sampling from the distribution of months for a given year, and the distribution of days for a given month. After that, we checked that there were no missing values within any of our fields and successfully casted our fields into their respective data types for use in model training.

user_reviews

- user_id (string): primary key of user
- posted (datetime64): when review was posted
- item_id (int64): id of game
- recommend (boolean): if the user recommends the game
- review (string): text of review

steam_games

- developer (string): creator of the game
- app_name (string): app name for game
- url (string): link to main page for game
- release_date (datetime64): date when game was released
- price (float64): listed price of game
- item_id (int64): id of game
- genres (int64 one-hots): categories of the game
- tags (int64 one-hots): categories of the game. similar to genre but like hashtags
- specs (int64 one-hots): specifications, description of game

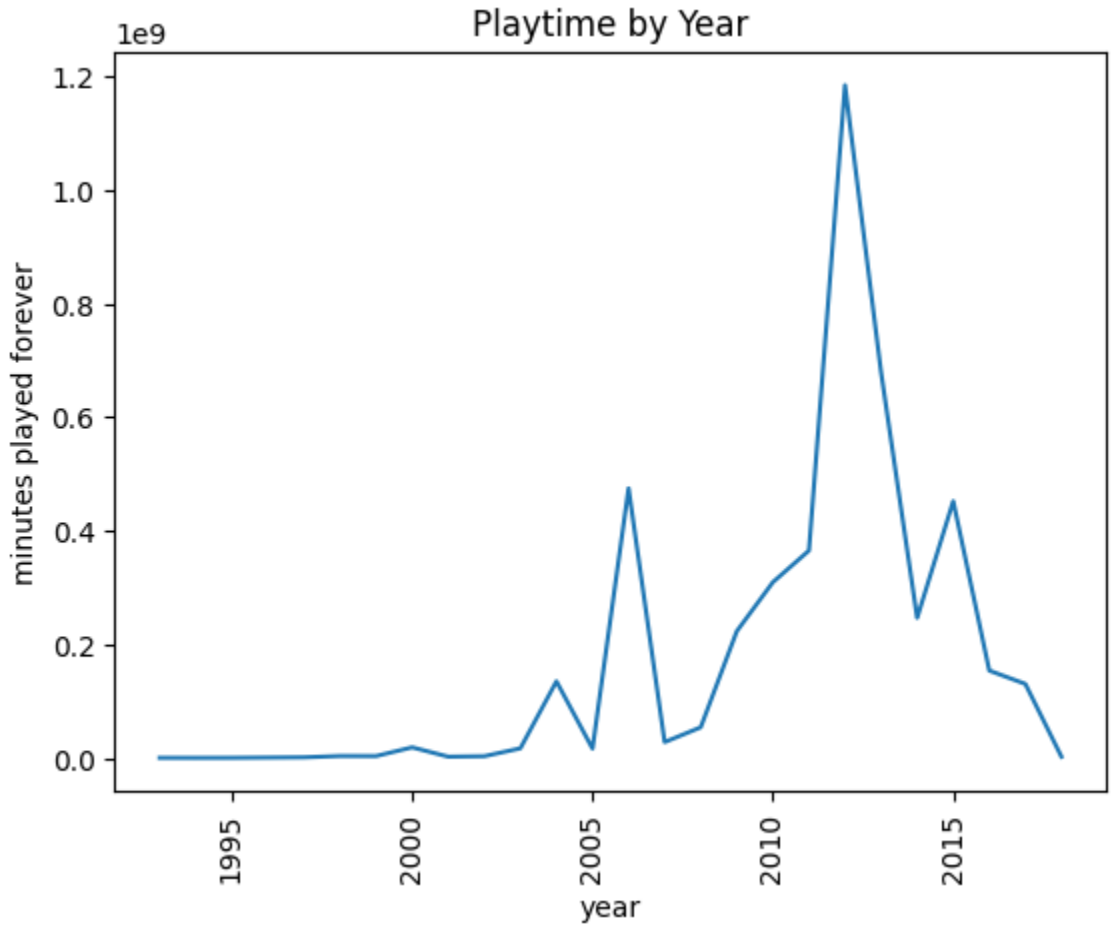
user_games

- user_id (string): primary key of the user
- steam_id (int64): steam id number. sometimes the same as user_id but not always. steam_id appears to always be numeric.
- user_url (string): link to steam user profile
- item_counts (int64): number of games user owns
- item_id (int64): id of game
- Item_name (string): title of game
- Playtime_forever (int64): number of minutes played since purchase
- Playtime_2weeks (int64): number of minutes played in the last two weeks

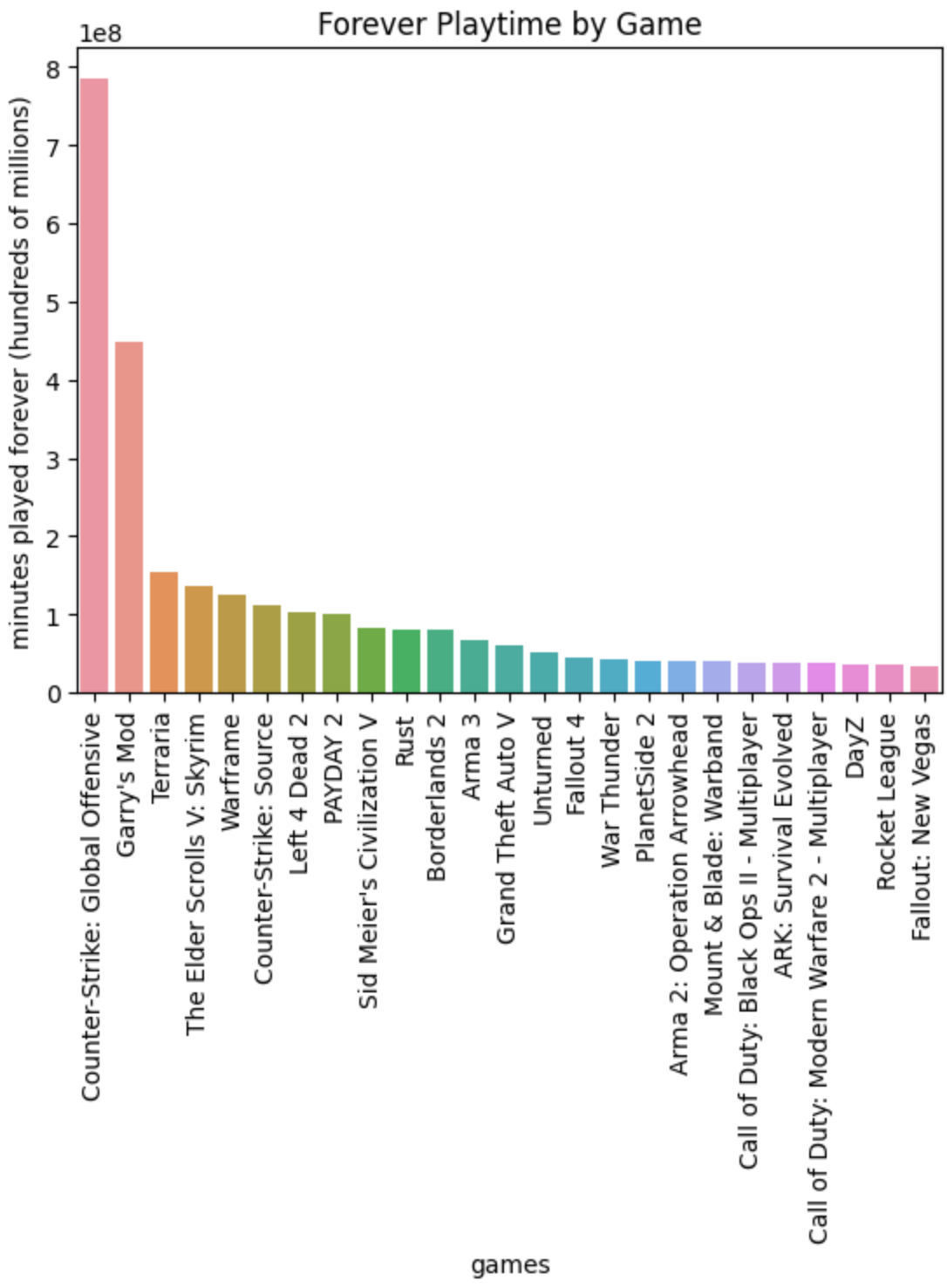
Exploratory Data Analysis

Dataset Statistics:

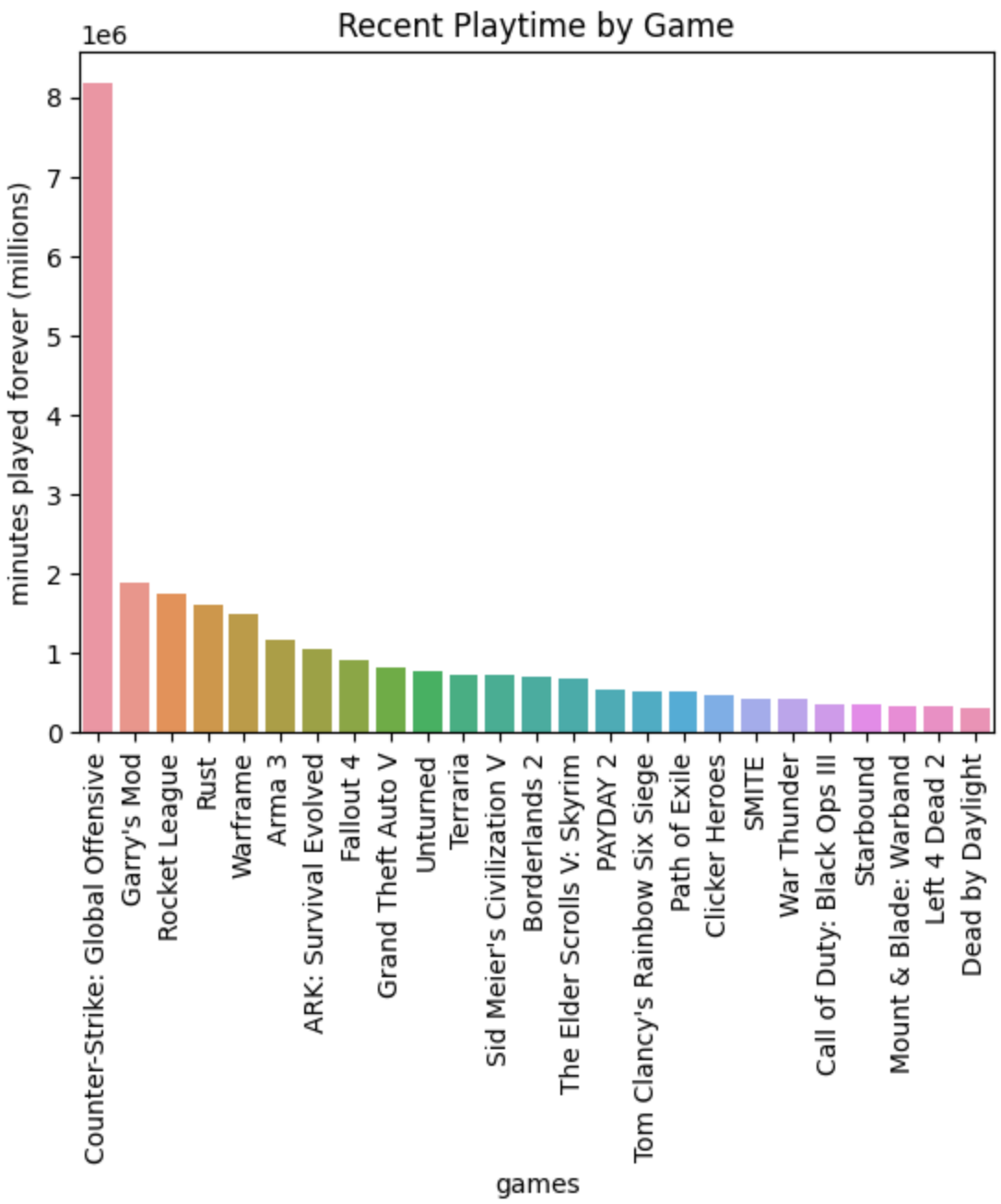
- Average Number of Games per User: 71.84
- Average Total Playtime per User: 72052 minutes
- User “phrostb” Owned the Most Games at 7762 Games Owned
- Top 10 Most Popular Tags:
    - Single-player, Indie, Action, Downloadable Content, Steam Trading Cards, Casual, Adventure, Multi-player, Simulation, Strategy



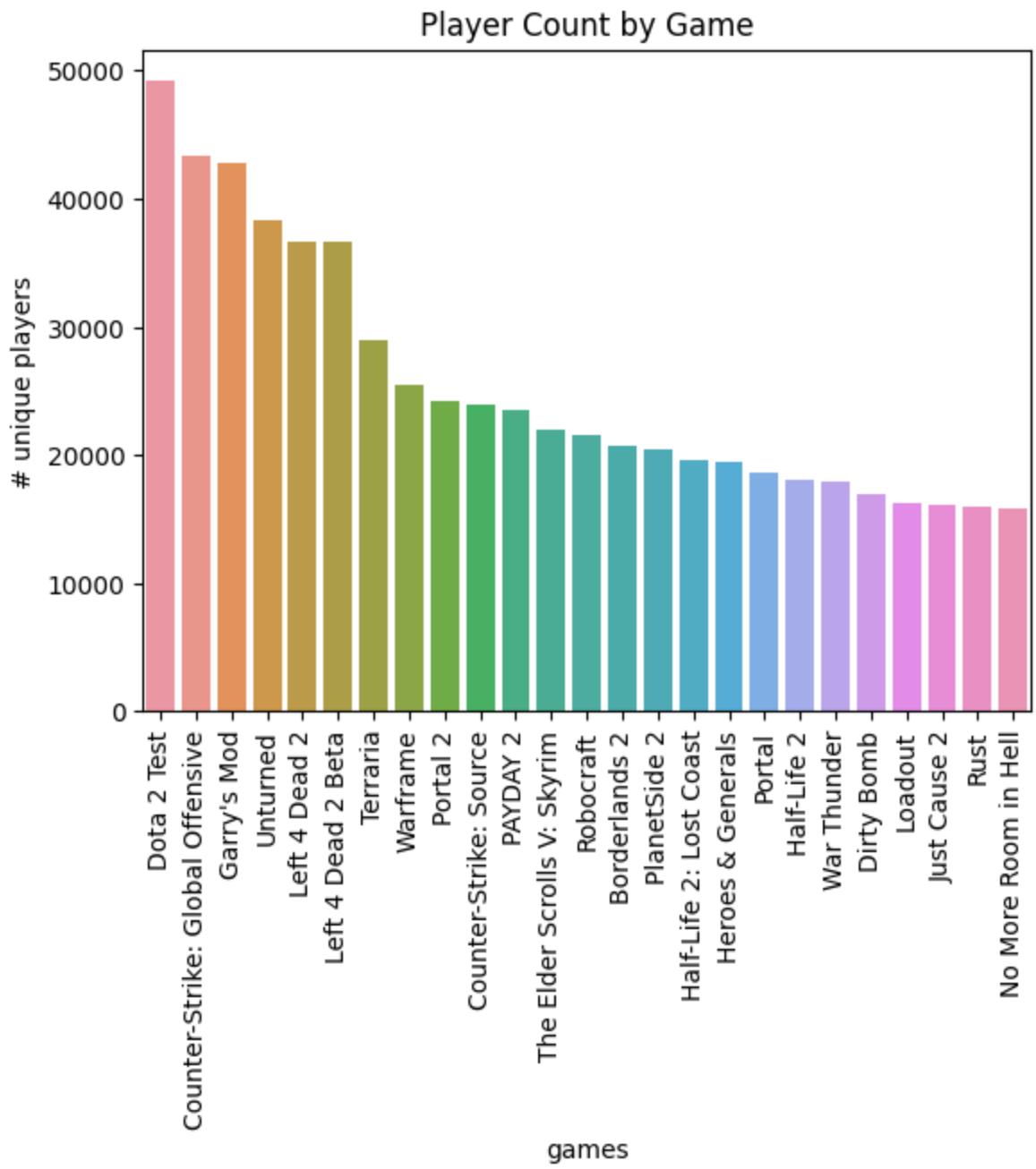
This plot of total playtimes by the release date of video games, shows that the majority of players in our data prefer games released from 2010 to 2015. Maybe these games were more popular? Or more games were released in these years?



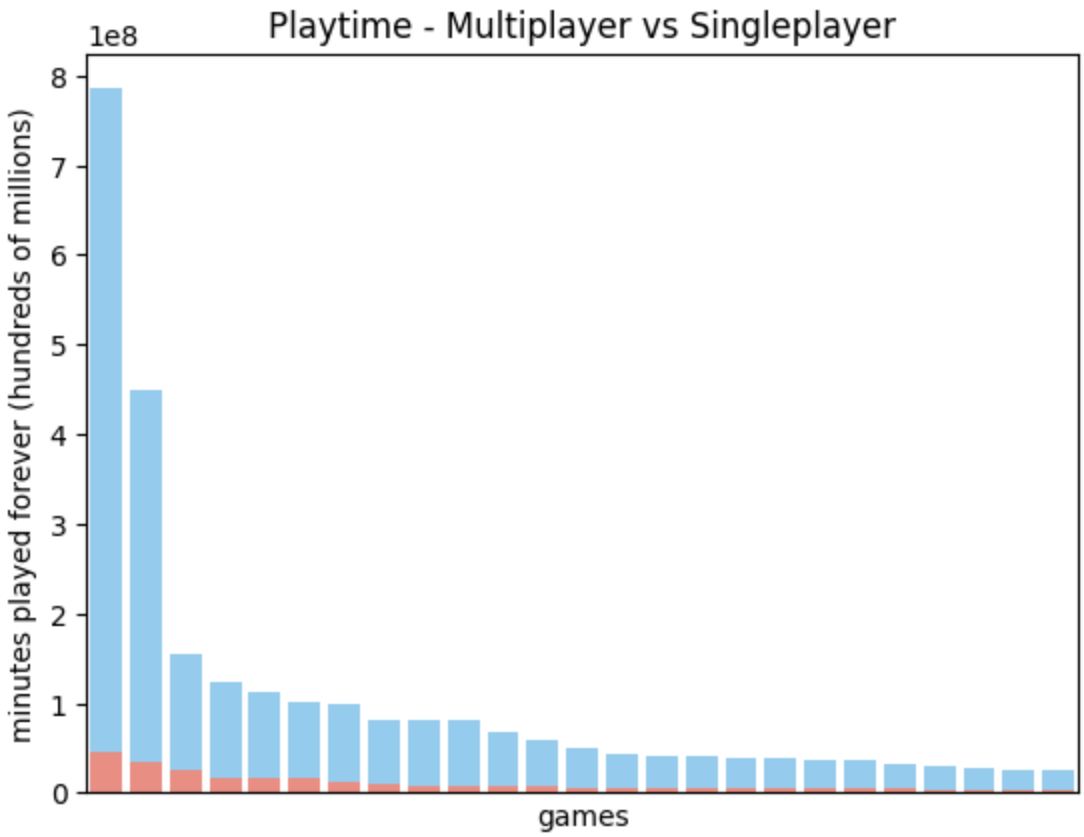
This shows that certain games such as CSGO and Garry's Mod dominate in total playtime. These popular games may be overrepresented in the dataset and therefore be recommended more often.



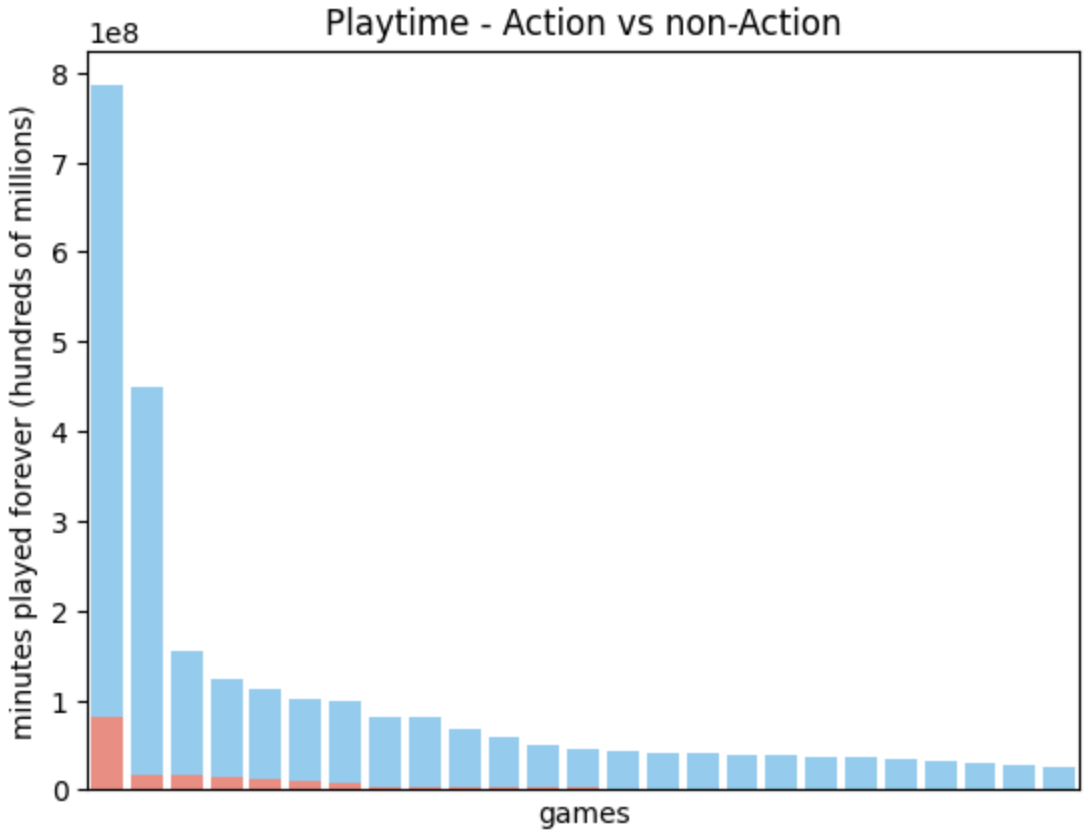
Comparing this to the previous plot shows that certain games, such as CSGO, remain very popular, while other games such as Garry's Mod have fallen in popularity, either falling in ranking or dropping out of the top 25 most played games.



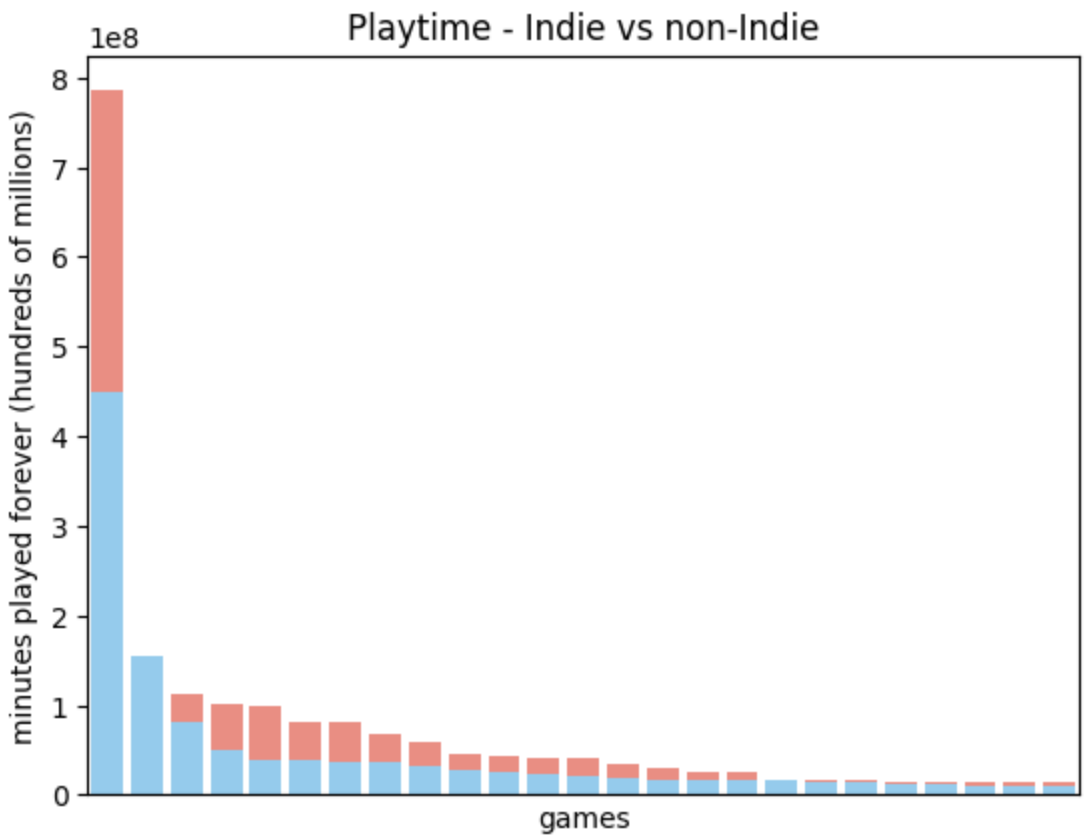
Ranking the most popular games by ownership rate rather than playtime shows another way that popular games may skew the recommendation results, or simply, which games may be best to recommend.



This plot shows the difference in total playtimes between the top 25 most played multiplayer and single player games, where Multiplayer is blue and Singleplayer is red. This shows that multiplayer games on average have higher playtimes than single player games and may be recommended more often when playtime is taken into account in the recommendation model. A likely explanation for this fact is that multiplayer games more often than not center around player interaction fueling infinite replayability and player retention, compared to single player story games that have at most tens of hours of content that users usually only play through once.



Plotting the difference between total playtimes for the top Action(blue) and non-action(red) games shows a similar pattern. While this may be influenced by innate features of Action games, it is more probable that the pattern rides off the fact that most Multiplayer games are action games.



Finally, the difference between total playtimes for Indie(blue) and non-Indie(red) games shows slightly higher playtimes for non-indie games. This may be fueled by the fact that bigger triple A studios often have more resources to pour into their games than Indie companies, leading to more gameplay content. However, it also means that our recommender system may slightly favor non-Indie games when making recommendations, possibly to the detriment of Indie developers.

Predictive Task

For the purposes of this assignment, our recommender system will be evaluated on its accuracy in predicting whether or not a random (user, game) pair is or is not owned by the given user. This will be the same as our Assignment 1 mandatory task. Our model will mostly be trained on the (user,games) interaction data in user_games, with steam_games being used to filter

recommended games by game metadata at the end. The dataset will be split into 80% training and 20% testing data. Our training data will contain 3,370,914 user interactions, with our testing data containing 842,729. Since our interaction data only includes positive samples, we also need negative samples for testing. Our test set will then be augmented to also include negative samples, by randomly sampling and including a game that the user does not own, for each user in the original test set. Considering that our test set contains almost a million rows even with a 80/20 split, we will be testing our intermediary models and tuning hyperparameters on a subset of 20,000 rows, similar to what we did for Assignment1, and only testing the final model on the entire test set.

Baseline Model

Our baseline model in this case will implement something similar to the popularity/similarity model used to recommend books for Assignment 1. This simple model will compute for each (user, game) pair, the Jaccard similarity between the given user, and 20 other random users that own the game. Only 20 other users are compared rather than all other users, or else extremely popular games, such as CSGO with 43776 players would take forever to query. The model then predicts that the user owns the game if either the game is in the top 1000 games by playtime, or if the maximum of the computed similarities is greater than 0.45.

This model exhibited an 84.96% accuracy rate on a test set of 20,000 samples.

Improved Model

To improve upon our baseline model, we chose to implement matrix factorization techniques, specifically Bayesian Personalized Ranking and Alternating Least Squares models, to more effectively utilize the large amount of interaction data we have and generate multiple recommendations. Rather than predicting ownership for a given (user, game) pair, these models

can recommend a list of games to a given user. These matrix factorization techniques aim to decompose a given user-item interaction matrix into separate user and item matrices, representing users' preferences and item features. Recommendations are then made by multiplying a given user's preference vector, taken from the decomposed user matrix, with the item feature matrix to get a list of rankings for each item. Recommended items are then taken as a subset of the top ranked items.

Bayesian Personalized Ranking vs. Alternating Least Squares

Bayesian Personalized Ranking and Alternating Least Squares mainly differ in their loss functions and optimization techniques. Both are iterative algorithms that attempt to optimize their decomposed user and item matrices while minimizing loss between the reconstructed interaction matrix (decomposed user matrix multiplied with item matrix) and the original interaction matrix. BPR optimizes by maximizing the probability that, for random positive(owned) and negative(un-owned) sample, the positive sample is ranked higher than the negative sample. On the other hand, ALS optimizes by minimizing mean squared error between the reconstructed and original interaction matrix. The impact of this is that rating estimates made by ALS will be closer to true ratings, but may not maintain relative orderings of ratings as well as BPR.

Interaction Matrices

In order to implement these models, we first needed to create interaction matrices to factorize. For this, we took two approaches, one interaction matrix encoding purely binary (owned/not-owned) data for (user,game) pairs, and another interaction matrix that would encode a sort of implicit rating for each (user, game) pair. For the binary interaction matrix, interactions were encoded as 1 if the user owned the game, or 0 if the user did not own the game. For the implicit rating interaction matrix, we calculated implicit ratings from 1-5 for each (user,game) pair

where the user owns the game. Rating scores were then calculated with respect to each game's median playtime and the user's playtime on that game:

$$implicit\_rating = \begin{cases} 5,\ \text{playtime}_{\text{user}} > \text{medianplaytime}_{\text{game}} \\ 4,\ \text{playtime}_{\text{user}} > 0.75 * \text{medianplaytime}_{\text{game}} \\ 3,\ \text{playtime}_{\text{user}} > 0.5 * \text{medianplaytime}_{\text{game}} \\ 2,\ \text{playtime}_{\text{user}} > 0.25 * \text{medianplaytime}_{\text{game}} \\ 1,\ \text{playtime}_{\text{user}} < 0.25 * \text{medianplaytime}_{\text{game}} \end{cases}$$

(user, game) pairs where the user does not own the game are represented as 0. Interaction matrices were produced by pivoting the user_games dataframe and then converted to sparse matrices to adress memory constraints.

Model Performance

In order to produce recommendations, we will be implementing the BPR and ALS models from Ben Frederickson's implicit python library. Each of the above matrix factorization models were then trained with default parameters on both the binary and implicit rating interaction matrices. The models were then evaluated on 20,000 test samples based on whether or not, for each (user, game) pair, that game was within the top N(1000) recommended games.

	accuracy
baseline	0.84825
binary_bpr	0.79355
binary_als	0.87225
implicit_rating_bpr	0.79375
implicit_rating_als	0.87520

Evaluation results show that our ALS model performs significantly better than the BPR model, in terms of prediction accuracy. However, it does not show a drastic performance difference between using a binary interaction matrix versus our implicit ratings matrix. However, since the

implicit ratings interaction matrix likely contains more nuanced player interaction data than the binary one, it may be more likely to produce more unique recommendations. As a result, we will move forwards with our implicit rating ALS model for parameter tuning and final deployment.

Hyperparameter Optimization

For our ALS model, we decided to tune the number of latent factors that represented user preferences and item features, the l2 regularization value, the number of optimization steps (iterations), and the number of top recommended games to evaluate the (user, game) pair on.

```
{'factors': [5, 10, 25, 50, 75, 100],
 'n_recommended': [100, 1000],
 'regularization': [0, 0.001, 0.01, 0.1, 1, 10, 20, 35, 50, 60, 75, 90, 100],
 'iterations': [5, 10, 15, 25, 35]}
```

Each model within the gridsearch was evaluated on 20,000 test samples, with the total hyperparameter tuning time taking 10 hours and 27 minutes, with more than 3 days of total cpu runtime (multithreading).

The best performing model came out with the following parameters:

- factors: 100
- n_recommended: 1000
- regularization: 100
- iterations: 15

Validation Accuracy: 88.93%

This model was then evaluated on the complete test set of 1,685,458 samples, giving us a testing accuracy of 89.31%.

Conclusion

We have produced a model that is quite accurate at predicting whether or not a user owns a certain game. This prediction accuracy, may however, not perfectly reflect the reality of recommending users a given set of games. To evaluate the performance of our model through the eyes of actual users, we have written a function using our final model that recommends a certain number of games for a given user, filtered by arbitrary tags and release date constraints. Taking a quick look at the game recommendations for a few random users shows quite good diversity in the games recommended. Recommendations also do not seem to be dominated by the most popular games, for users who do not yet own them.

	Top Games
0	Counter-Strike: Global Offensive
1	Garry's Mod
2	Unturned
3	Left 4 Dead 2
4	Terraria
5	Warframe
6	Portal 2
7	Counter-Strike: Source
8	PAYDAY 2
9	Robocraft
10	Borderlands 2
11	PlanetSide 2
12	Heroes & Generals
13	Portal
14	Half-Life 2
15	War Thunder
16	Dirty Bomb
17	Loadout
18	Just Cause 2
19	Rust

Recommendations for User “--000--”:

	game	rank	release_date	url
0	Brawlhalla	0.638917	2017-10-17	http://store.steampowered.com/app/291550/Brawl...
1	Saints Row IV	0.620747	2013-08-19	http://store.steampowered.com/app/206420/Saint...
2	Brick-Force	0.497074	2012-07-12	http://store.steampowered.com/app/335330/Brick...
3	Spiral Knights	0.488031	2011-06-14	http://store.steampowered.com/app/99900/Spiral...
4	PAYDAY 2	0.430909	2013-08-13	http://store.steampowered.com/app/218620/PAYDA...
5	No More Room in Hell	0.420246	2011-10-31	http://store.steampowered.com/app/224260/No_Mo...
6	Dungeon Defenders II	0.416389	2017-06-20	http://store.steampowered.com/app/236110/Dunge...
7	Loadout	0.395099	2014-01-31	http://store.steampowered.com/app/208090/Loadout/
8	Block N Load	0.383582	2015-04-30	http://store.steampowered.com/app/299360/Block...
9	Gear Up	0.379616	2015-01-28	http://store.steampowered.com/app/214420/Gear_Up/
10	Defiance	0.377757	2014-06-04	http://store.steampowered.com/app/224600/Defia...
11	Survarium	0.374527	2015-04-02	http://store.steampowered.com/app/355840/Surva...
12	Transformice	0.374057	2015-01-30	http://store.steampowered.com/app/335240/Trans...
13	Warface	0.365061	2014-07-01	http://store.steampowered.com/app/291480/Warface/
14	Don't Starve Together	0.363539	2016-04-21	http://store.steampowered.com/app/322330/Dont_...
15	The Expendabros	0.361837	2014-08-05	http://store.steampowered.com/app/312990/The_E...
16	Tribes: Ascend	0.360996	2012-06-27	http://store.steampowered.com/app/17080/Tribes...
18	Cry of Fear	0.341087	2013-04-25	http://store.steampowered.com/app/223710/Cry_o...
19	Poker Night at the Inventory	0.338689	2010-11-22	http://store.steampowered.com/app/31280/Poker_...
20	WARMODE	0.336743	2015-08-25	http://store.steampowered.com/app/391460/WARMODE/

Recommendations for User “76561198066143243”:

	game	release_date	url
0	Tribes: Ascend	2012-06-27	http://store.steampowered.com/app/17080/Tribes...
1	APB Reloaded	2011-12-06	http://store.steampowered.com/app/113400/APB_R...
2	Warframe	2013-03-25	http://store.steampowered.com/app/230410/Warfr...
3	Defiance	2014-06-04	http://store.steampowered.com/app/224600/Defia...
4	War Thunder	2013-08-15	http://store.steampowered.com/app/236390/War_T...
5	Blacklight: Retribution	2012-07-02	http://store.steampowered.com/app/209870/Black...
6	RIFT	2013-10-24	http://store.steampowered.com/app/39120/RIFT/
7	Robocraft	2017-08-24	http://store.steampowered.com/app/301520/Roboc...
8	Loadout	2014-01-31	http://store.steampowered.com/app/208090/Loadout/
9	The Lord of the Rings Online™	2012-06-06	http://store.steampowered.com/app/212500/The_L...
10	Warface	2014-07-01	http://store.steampowered.com/app/291480/Warface/
11	Emily is Away	2015-11-20	http://store.steampowered.com/app/417860/Emily...
12	Arma 3	2013-09-12	http://store.steampowered.com/app/107410/Arma_3/
13	Counter-Strike: Global Offensive	2012-08-21	http://store.steampowered.com/app/730/CounterS...
14	DayZ	2013-12-16	http://store.steampowered.com/app/221100/DayZ/
15	Block N Load	2015-04-30	http://store.steampowered.com/app/299360/Block...
16	Evolve Stage 2	2015-02-10	http://store.steampowered.com/app/273350/Evolv...
17	Star Conflict	2013-02-27	http://store.steampowered.com/app/212070/Star_...
18	Mount & Blade: Warband	2010-03-31	http://store.steampowered.com/app/48700/Mount_...
19	Natural Selection 2	2012-10-30	http://store.steampowered.com/app/4920/Natural...

This user only owns 1 game, but the recommendations differ quite a bit from simply the top most popular games.

Recommendations for User “cretanarcher”:

- Filtered for only “Anime” games (tag)

	game	release_date	url
90	Valkyria Chronicles™	2014-11-11	http://store.steampowered.com/app/294860/Valky...
93	Jet Set Radio	2012-09-19	http://store.steampowered.com/app/205950/Jet_S...
154	Sonic Adventure DX	2011-03-04	http://store.steampowered.com/app/71250/Sonic_...
161	Brawlhalla	2017-10-17	http://store.steampowered.com/app/291550/Brawl...
188	APB Reloaded	2011-12-06	http://store.steampowered.com/app/113400/APB_R...
272	Spiral Knights	2011-06-14	http://store.steampowered.com/app/99900/Spiral...
279	Lethal League	2014-08-27	http://store.steampowered.com/app/261180/Letha...
321	Revolution Ace	2014-03-19	http://store.steampowered.com/app/274560/Revol...
324	Labyronia RPG	2015-08-14	http://store.steampowered.com/app/391260/Labyr...
336	Labyronia RPG 2	2015-08-28	http://store.steampowered.com/app/397500/Labyr...
340	Skullgirls	2013-08-22	http://store.steampowered.com/app/245170/Skull...
341	Without Within	2015-01-22	http://store.steampowered.com/app/345650/Witho...
350	TERA	2015-05-05	http://store.steampowered.com/app/323370/TERA/
384	Marble Mayhem: Fragile Ball	2015-07-07	http://store.steampowered.com/app/370510/Marbl...
415	DETOUR	2011-05-16	http://store.steampowered.com/app/92100/DETOUR/
422	Sins Of The Demon RPG	2016-05-13	http://store.steampowered.com/app/461640/Sins_...
426	Skyborn	2014-02-21	http://store.steampowered.com/app/278460/Skyborn/
430	Secret Of Magia	2015-08-21	http://store.steampowered.com/app/396160/Secre...
450	Data Hacker: Reboot	2015-04-17	http://store.steampowered.com/app/331790/Data_...
452	HuniePop	2015-01-19	http://store.steampowered.com/app/339800/Hunie...

Most of these are definitely “Anime” games. However, some of these such as “Spiral Knights” and “Brawlhalla” are well acclaimed games that I myself wouldn’t exactly call “Anime” games. Maybe forgoing the use of tags, which can be modified by any user, and sticking with only genre and specs may produce better filtering results.

Note on Recommendation Quality: I personally have 500+ hours on TERA D:

While I have yet to include my own user data into the interaction matrix, and therefore cannot argue too much for the quality of recommendations. I'd reckon that these recommendations are least passable, due to the diversity in recommended games, and the fact that most recommended games seem quite well received (steam store page reviews). This indicates that the user preferences and item feature matrices that our model arrived at encode good latent factors for explaining why users enjoy/purchase certain games over others.

Related Literature

In the following section we will cover literature and research related to the task we addressed and the dataset we used.

Self-Attentive Sequential Recommendation

Wang-Cheng Kang, Julian McAuley

https://cseweb.ucsd.edu/~jmcauley/pdfs/icdm18.pdf

The Steam games dataset we used was originally scraped by Professor Julian McAuley, Wang-Cheng Kang, and other graduate students for this paper. The Steam games dataset was used alongside two Amazon datasets, containing information on Beauty and Games user interactions, and a MovieLens dataset to evaluate their self-attentive sequential recommender model. From their notes, both the MovieLens and Amazon datasets are widely used for evaluation of collaborative filtering algorithms. Their self-attentive sequential recommender technique takes inspiration from the use of Transformer models in NLP translation tasks, particularly using self-attention matrices to influence how much each previous action influences the next action prediction, allowing their model to take account of a user's entire interaction history, while avoiding the computational costs of neural networks. Each dataset was used to evaluate 9 recommender models on Top-N evaluation metrics Hit Rate@10 and NDCG@10,

with SASRec outperforming all baselines, with a 6.9% Hit Rate and 9.6% NDCG improvement compared to the strongest baseline.

Generating and Personalizing Bundle Recommendations on Steam

Apurva Pathak, Kshitiz Gupta, Julian McAuley

https://cseweb.ucsd.edu/~jmcauley/pdfs/sigir17.pdf

The Steam games dataset was scraped with assistance from the authors of this paper, Apurva Pathak, and Kshitiz Gupta, graduate students working with Professor Julian McAuley. The user games interaction data was used to train a BPR model that would learn user preferences for items and item to item correlation. Those learned parameters were then used in the estimator function for a bundle recommendation BPR model. Including bundle size and correlation of bundle items, allowed their model to outperform baselines (regular BPR, BPR with item features, …) in bundle ranking in terms of AUC. This bundle ranking model was then used in a greedy algorithm to generate and iteratively optimize/improve personalized bundles.

A Simple Convolutional Generative Network for Next Item Recommendation

Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, Xiangnan He

https://liqiangnie.github.io/paper/p173-he.pdf

The authors of this paper present an improvement to the current matrix factorization approaches for collaborative filtering. They hypothesize that the inner product function used to linearly combine latent user and item features may not sufficiently capture the complexity of user interaction data. And propose the use of deep neural networks to replace that interaction function. They introduce a neural matrix factorization model, taking in user and item embeddings from both a matrix factorization(GMF) and multi-layer perceptron (MLP), and feeding that through the deep neural network, outputting a score representing compatibility

between the user and item. An evaluation on Hit Rate @ 10 shows their DNN model outperforming baselines, including bayesian personalized ranking and alternating least squares.

Building Recommender Systems for Video Games on Steam

Brian Choi, Misun Ryu, Dharmesh Panchal, Andrew Le

https://library.ucsd.edu/dc/object/bb5021836n/_3_1.pdf

The authors of this paper were graduate students advised by Professor Julian McAuley. They addressed the same task, using the same dataset as me: to model a recommender system that accurately predicts and recommends the video games that users are most likely to buy. They also included item to item recommendations. Unlike our approach, they chose to include only the top 1000 most popular games within their interaction matrix, and therefore their recommendations, with the argument that data on games past the top 1000 are extremely sparse. Following suit with the de facto methods for this type of task, they implemented matrix factorization techniques. They approached the problem with a simple binary 1/0 interaction matrix for user-game ownership. Their work focused on the bayesian personalized ranking and weight approximate-rank pairwise techniques which they implemented with Tensorflow and LightFM. They evaluated their models with respect to AUC, tuning their model with various hyperparameters we had restricted access to, since we used Implicit python library rather than building the models from scratch. With that they reached a final AUC of 0.9151. They also then filtered by varying degrees of playtime to explore the dynamics between bought and played games. For that, their experiment results showed that a model trained on cases where the game was played for more than the 10th percentile of minutes played for that game and tested on games played for more than 30 minutes performed the best with an AUC of 0.9185. However they chose to stick with the unfiltered model for its generability. They then combined that model with game genres, price, release year, and median playtime with LightFM to create a hybrid model utilizing both collaborative filtering and item features. With this they were able to achieve

similar AUC scores as the collaborative filtering only model, but were unable to finalize hyperparameters within the project's time period.