Blake Hernandez

Final Report:
Steam Game Recommender System

Problem Statement

There is fierce competition in the world of digital entertainment nowadays, with so many easily accessible sources, it is important to keep users engaged with your platform lest they look for entertainment elsewhere. One way to do this is by suggesting relevant content to users rather than asking them to seek it out for themselves on your platform. This is where recommender systems come in.

Steam is the largest digital video game retailer. On steam users can purchase and download games. Then they can access their purchased games through their steam library. For a long time Steam was the only large digital game store, but recently there has been competition popping up. The Epic Games store has been making headway in the digital sales market, and new game subscription services like Xbox Game Pass have also been on the rise. Now more than ever it is important for Steam to be able to keep customers engaged with their platform.

The goal of this project is to create a video game recommender system for Steam. By providing relevant recommendations to users the system will have the potential to increase user retention on the platform and increase sales. The model's being considered are memory based collaborative filtering, matrix factorization, the models available in the Surprise recommenders library, and the LightFM model.

Data Wrangling

The data for this project came from three sources. The initial data source was kaggle, where there were two datasets for Steam games. One dataset contained user playtime data in minutes for different games in their steam libraries. The second dataset had feature data for a number of games in Steam. The feature data included game genres, tags, developer, and price among other things. I found out shortly after starting to work with the data that the sources for both datasets were public APIs. The first dataset came from Steam's own API. After finding this out I made a few calls to the API and pulled additional user play hour data and a list of all apps in the Steam Store. The second dataset came from the SteamSpy API, a source independent from Steam that

provides metadata for games that is not readily available through Steam's own API. I used this API to get additional feature data for each game.

The primary data used for modeling was user play hour data, which after consolidating the portion from kaggle and that gathered from Steam's API had a total of 13,065 unique users and 5,155 unique games. The data came in the following format:

	userID	gameName	behavior	playHours
120316	62990992	resident evil 4 / biohazard 4	purchase	1.0
120317	62990992	resident evil 4 / biohazard 4	play	4.2
121158	62990992	iBomber Defense Pacific	purchase	1.0
121488	62990992	Zoo Park	purchase	1.0
120622	62990992	Zombie Zoeds	purchase	1.0
120623	62990992	Zombie Zoeds	play	1.6
121487	62990992	Zombie Driver HD Apocalypse Pack	purchase	1.0

- userID
    - Unique user identifier for each steam user

- gameName
    - The name of the video game in question

- behavior
    - This column is populated by either "play" or "purchase", indicating the type of user interaction.

- playHours
    - Populated by 1.0 for all records with the behavior column equal to "purchase". For all records with the behavior column equal to "play" this column has the user's total play hours for the game.

The purchase vs play behaviors needed to be sorted out first. The solution was to split the data into two separate dataframes, one with all the play behaviors and the other with all the purchase behaviors. Then merge them back together on userID and gameName, this left me with a single entry for each game a user either purchased or

played. For all entries where the user purchased a game but did not play it, the playHours column was set to zero.

The next step was to merge the play hour / interaction data from the Steam API with the feature data from kaggle. This presented two challenges, the first being that the data gathered from kaggle did not include unique appids for each game, meaning my only way of linking the two datasets was by the game name. There were slight differences in naming conventions between the interaction data and the feature data, thankfully these differences were almost exclusively in the form of special character and capitalization, so by removing all special characters and setting all names to lower case the two datasets could be merged without issue.

The second challenge was that the feature data from kaggle was not complete, there were some 17,000 records in the interactions data that were lacking features. This had two causes.The interaction data contained not only full games but game add-ons in the form of dlc (downloadable content) while the feature data only contained full games. On top of that the feature data as a whole was incomplete, missing a large number of games. It was at this point that I sought out the source of the kaggle feature data, the SteamSpy API.

To reduce the amount of data I needed to pull from SteamSpy's API, I got a complete list of all Steam's games from the Steam API, and connected each game in the interaction data to its appid again using the game name as the key. I then got a list of all appid's in the interactions data, and made a series of calls to SteamSpy's API to get the feature data for each one of them

EDA

EDA began with looking at the distribution of play hours across all users and games. This revealed that the play hour data was heavily skewed to the left, with a median of 4.5 hours, a mean of 48.9 hours, and a maximum value of over 10,000 hours. The primary reason for this is that not all games are meant to be played for the same amount of time. Some online games are designed to provide hundreds if not thousands of hours of entertainment, while others are meant to only deliver a short couple hours. With this in mind I decided to look at the play hour distributions of games on an individual level, which revealed that there was still skew present on an individual item level, though not to the same extent as the population.

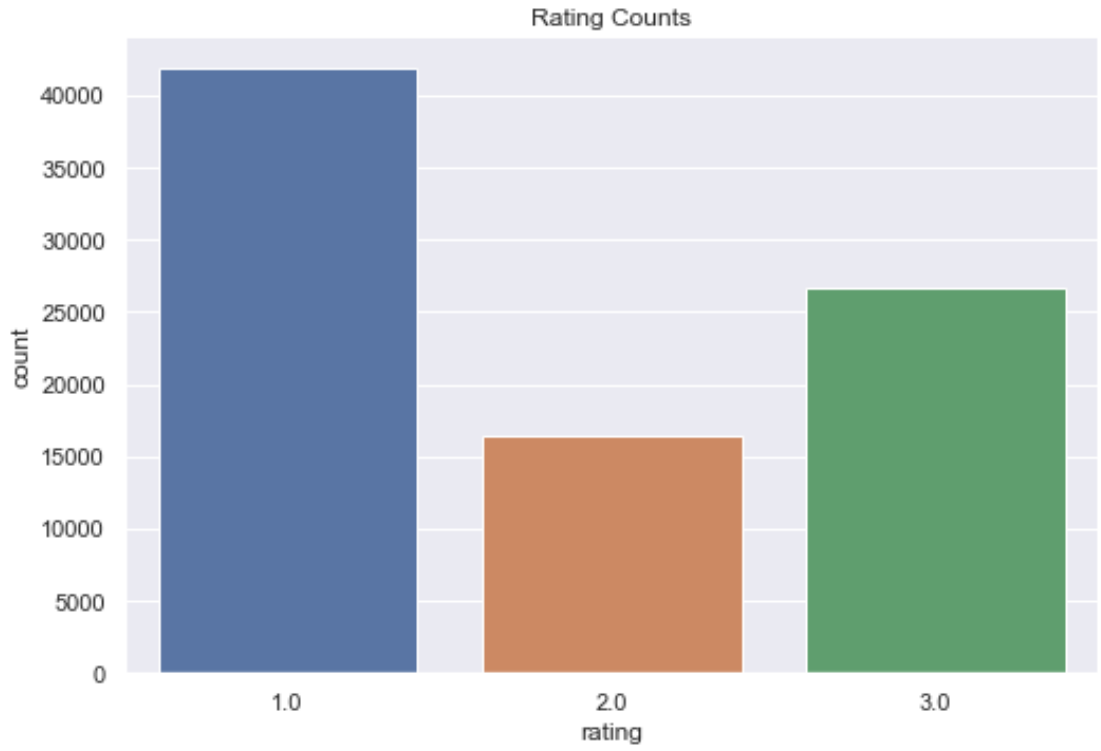
The skew present on an individual game level is largely caused by a significant number of purchased games having zero play hours (roughly ⅓ of the data). This is due to a combination of free games and the notorious Steam sales, during which it is not uncommon for games to have their price reduced by 70-90%. When a game is free or costs only a fraction of its full price, many users will download it on a whim, even if they are only moderately interested in it.

I briefly considered removing the zero play hour games from the data before modeling, but ultimately decided against it. Dropping this data would have significantly increased the sparsity of the matrix that was used for collaborative filtering. Additionally, even if a user never actually played one of the games in their library, they still showed some interest in it by purchasing it.

To deal with outliers in play hours, I capped the data more than 1.5 IQR away from the 1st and 3rd quantiles of the data. This was done on a game by game basis since a value that may be an outlier for one game may not be for another. Next I scaled the hours for each game so that they all fell between 0-1 using sklearns MinMaxScaler class.

The next step was to convert the play hours for each game into ratings. This was a necessary step since some of the models being used came from the Suprise library, which only accepts explicit rating data as input for its models. The scaled play hours for each game was assigned to one of three bins, ratings from 1-3, on a user by user basis based on their position relative to the mean scaled play hours of all games in the users library. This was done to account for differences in user preferences. Some users may simply not be as interested in playing any games for extended periods of time, and thus may put significantly fewer hours than other users into a game that they still enjoy.

Final Distribution of game ratings:



One additional step was to reduce the size of the data. There were two motivations for this, the first being that a smaller sample of the data would make model training faster, the second being that it could help with the cold start issue that collaborative filtering models suffer from. By limiting the data to only users who had at least 35 games in their library, and games who had been purchased by at least 50 users, the size of the data was reduced from 180,000 records to around 87,000 records. This also brought the sparsity of the data from 0.003 to 0.11.

Modeling

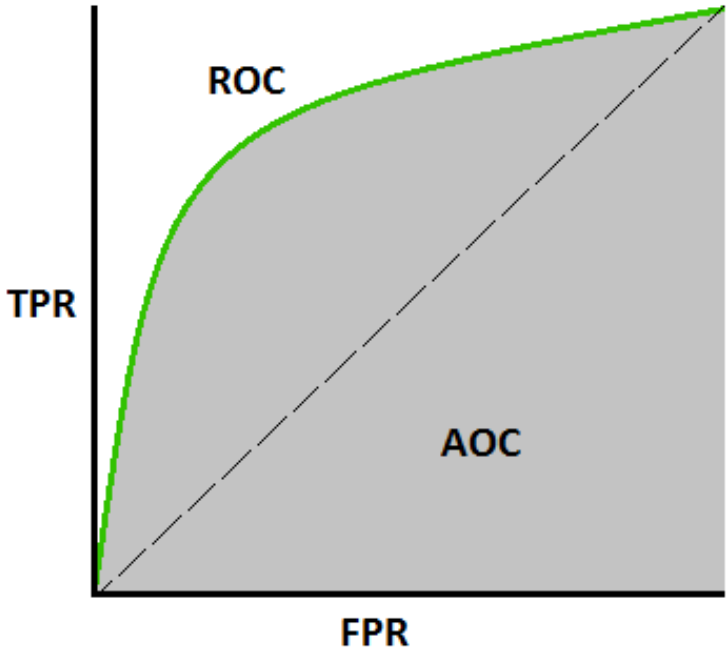
Metrics

For modeling I focused on three metrics, root mean squared error (RMSE), area under the receiver operating characteristic curve (AUC), and precision at k (percison@k). There were some constraints in terms of the available evaluation metrics with the models I used, but by spreading the evaluation across these three metrics I was able to compare all models.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE is the root of the sum of squared errors for each prediction divided by the total number of predictions. It is a common metric for evaluating models.

AUC is the area under the ROC curve, which is a plot of the true positive and false positive rates at different classification thresholds. A high AUC value is an indication of good model performance for classification models.



Precision@k was the last metric used for model evaluation. This metric is one that is unique to recommender systems and focuses on the top k items being recommended to a user. Precision@k calculates the precision of these top k items based on a predetermined threshold for what is considered to be a relevant recommendation.

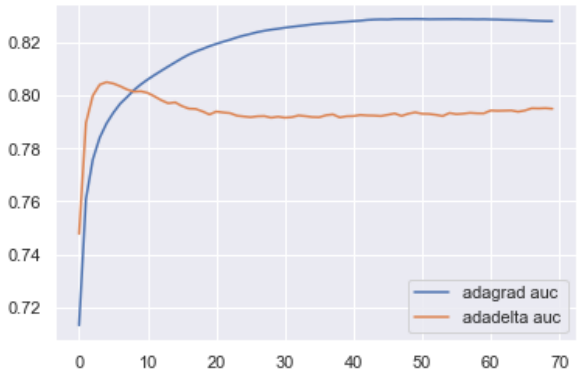
Models

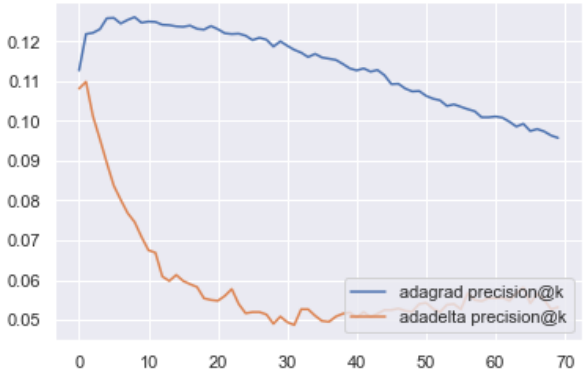
For this project I decided on a collaborative filtering model. I experimented with both user and item based filtering and discovered that for the data being used user based filtering tended to be the better performer.

The first model I experimented with was a memory based collaborative filtering. Two of these models were created, one using a similarity matrix, and the other relying on a KNN model for classification. The model using KNN was the better performer of the two by a slim margin. These two models were the worst performing overall, in terms of both speed and metrics. They were both very simplistic, as I built them from the ground up with little reliance on any imported classes, but I think they were valuable since they helped me get a good grasp on how collaborative filtering works.

The next set of models used were those from the Surprise recommenders library. The Surprise library offers a wide range of models, and after experimenting on the best performing ones I was left with the Singular Value Decomposition++ (SVD++) model. This is a matrix factorization model that builds upon the standard SVD algorithm by including the effect of a user choosing to rate (or purchase) an item to begin with. This model performed better than the memory based models in terms of RMSE, but had a low value for precision@k (only 4% at k=10).

The final and best performing model was the LightFM model. LightFM uses matrix factorization, and allows for metadata/features for both users and items to be incorporated into the model, making it a hybrid system. Interestingly enough, I found that including item metadata actually decreased the performance of the model, especially when working with a dense interaction matrix. This indicates that the hybrid model may be best suited for dealing with new users who haven't purchased many games, while the pure collaborative filtering model is better for users who already have a sizable purchase history.





The LightFM model allows for the use of four different loss functions, and two different learning schedules. The WARP loss function was selected as the best performing based on grid search results. The Adagrad learning schedule was determined to be the better learning schedule based on plots of the AUC and precision@k over different numbers of epochs.

The LightFM model performed the best overall, beating out the memory based model in AUC and the SVD++ model in precision@k by a wide margin. It was also the most efficient model in terms of training time.

Model | Metric	Percision@k	AUC	RMSE
Memory Base Collaborative Filtering		0.62	0.81
Item Based Surprise SVD++	0.04		0.76
User Based LightFM	0.12	0.82	

With this model completed I was able to generate a list of recommendations for both myself and a few of my friends using data gathered about each of us from Steam's API. The top ten recommendations for myself are shown below. Of these recommendations I can say with confidence that three of them are relevant, since they are games I have played before (outside of Steam) and greatly enjoyed. Of the remaining recommendations there are two that I would consider myself interested in. Overall I was very satisfied with this model's results.

	gameName	appid	rank
24	portal 2	620.0	1
18	portal	400.0	2
154	the witcher enhanced edition	20900.0	3
358	batman arkham city goty	200260.0	4
93	bioshock 2	8850.0	5
703	the witcher 3 wild hunt	292030.0	6
21	left 4 dead	500.0	7
337	terraria	105600.0	8
406	the walking dead	207610.0	9
168	fallout 3 game of the year edition	22370.0	10

Takeaways/Future Work

In the future it would be nice to figure out the best trade off between matrix density and data loss. For this project I experimented with a few different density thresholds and selected the best one, but I'm sure an algorithm could be used to figure out what the best tradeoff is. Another improvement would be to properly address the cold start problem. This could be done by either creating a model that is effective at dealing with new users, or by creating two separate models, one for new users and one for users who have passed a certain threshold of interactions.

Something else that could improve the model would be to get more explicit user feedback from users on the recommendations the model generates. As previously stated I was able to get a few friends to review the recommendations my model generated for them, but if I was able to get feedback from a larger group of people I may be able to pick up on trends in how the model recommends games, and make improvements based on that data.

Data Sources:

Kaggle Steam interaction data:
https://www.kaggle.com/datasets/tamber/steam-video-games

Kaggle Steam game feature data:
https://www.kaggle.com/datasets/nikdavis/steam-store-games

SteamSpy API
https://steamspy.com/api.php

SteamWorks API:
https://partner.steamgames.com/doc/webapi_overview

Image Sources

RMSE formula:
https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

AUC - ROC image:
https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5