Recommender System for Steam Games

Andreas Edte[1920339], Lennart Fertig[1961809], Kirill Gratchev[1638059], Shiyu Li[1869814], and Marc Popescu-Pfeiffer[1638093]

University of Mannheim

1 Introduction

Steam is a popular digital distribution platform for video games, developed and operated by Valve Corporation. It was launched in 2003 and has since become one of the largest and most widely used platforms in the gaming industry. Steam offers a vast library of games from various genres. Users can purchase games, download them digitally, and leave a review of games they've played. With over 10.000 newly released video games in just 2022, for a user to find games they will like can be difficult without an automated recommendation system.[1]

This paper focuses on a dataset which contains information on Steam users purchased games and their hours of playtime in specific games. From how much time a user spends playing a video game, an implicit rating can be derived. We will build recommender systems based mostly on that implicit rating derived from a players hours of playtime.

Recommendation systems are useful for surfacing products that match a user's interests and preferences, ensuring both customer satisfaction and platform revenue. By analysing past user activities, including the selection of products a user spent time on, these systems can make personalised product recommendations that cater to the users individual unique taste. Our recommender system should recommend to Steam-users potential new video games to purchase and play, based on their previous purchases and, more importantly, playtime in certain video games on Steam, and those of other users on the platform.

We will be implementing and evaluating multiple different approaches using Python in a Jupyter notebook. Utilizing the Python Surprise library, which offers convenient evaluation metrics, we can compare the performance of various pre-processing approaches as well as different rating prediction algorithms.

We hope to uncover interesting behaviours of the various algorithms and to achieve a useful recommendation system that can perform well on our Steam video game dataset.

2 Methodology

In order to build a recommender system for Steam games, different approaches and a variety of different algorithms were selected and implemented enabling us to explore diverse solutions.

2.1 KNN Baseline (item based)

A first approach to recommend steam games to users of the platform is the implementation of a *KNN Baseline (item based)* algorithm with the baseline estimate concept from Python Surprise's KNNBaseline.
KNN Baseline (item based) is a variant of the k-nearest neighbors (kNN) approach specifically designed for item-based collaborative filtering in recommender systems. Unlike traditional model-based algorithms, this algorithm is memory based. Instead of explicitly learning a model from the training data, it stores the training instances in memory.[2]
One key advantage of memory-based algorithms is their ability to make recommendations based on item similarity. This similarity is determined using distance metrics, such as Euclidean distance, Pearson correlation coefficient or cosine similarity. Note that in all following charts and statistics the pearson correlation is used. This metrics qualify the similarity between steam games based on their features. Due to this the algorithm provides a simple and intuitive approach to recommend items, which can handle new items without retraining the model. Similar to the k-Nearest Neighbors algorithm the concept of neighborhood is applied for the *KNN Baseline (item based)* algorithm. For each item in the dataset, a set of similar items is defined. The number of neighbors, denoted by "k" in kNN, determines how many items are considered when making the recommendations. By leveraging the ratings of the nearest neighbors, the algorithm can predict rating for unrated items.[2]
On the other hand KNNBaseline is an algorithm that combines the k-nearest neighbors (KNN) approach with a baseline estimate in recommender systems. It aims to improve the accuracy of recommendations by incorporating a baseline estimate that captures the inherent trends and biases in the data.
This can help capture the overall trends and biases in the data, leading to more accurate recommendations. In this algorithm, the prediction for an item is typically calculated by taking a weighted average of the ratings from its neighbors. The weights assigned to each neighbor are determined by their similarity to the target item. With the KNNBaseline, instead of solely relying on the weighted average of ratings from the nearest neighbors, you could include the baseline estimate to adjust the predicted ratings.

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v_i^k(u)} sim(u,v) \cdot (r_{vi} - b_{vi})}{\sum_{v_i^k(u)} sim(u,v)} \tag{1}$$

2.2 Singular Value Decomposition (SVD)

In addition to the *KNN Baseline (item based)* algorithm, also the "SVD" algorithm is implemented as an approach to recommend steam games to users of the platform. "SVD" stands for Singular Value Decomposition and is a model-based collaborative filtering algorithm. It is a matrix factorization technique especially used in recommender systems. A advantage of this algorithm is the possibility to factor the user-item rating matrix into low-dimensional matrices to capture underlying patterns and preferences[3]. Therefore the "SVD" the evaluation matrix M is broken down into three matrices:

$$U, \sigma and V$$

$$M = U * \sigma * V$$

U - represents the user feature matrix, where each row corresponds to a user and captures the user's preferences for various latent features.
σ - is a diagonal matrix containing the singular values representing the importance of each latent feature.
V - represents the item-feature matrix, where each row corresponds to an item and captures the item's association with various latent features.
By multiplying U, σ and V, the algorithm reconstructs the original score matrix M and predicts missing or unscored values. By doing so "SVD" is effective in reducing the dimension of the scoring matrix and capturing factors affecting user-item interactions. It is often used in recommendation systems such as film or music recommendations, where it can provide accurate predictions based on learned latent factors[4].

2.3 Singular Value Decomposition++ (SVD++)

"SVD++" is an extension of the "SVD" (Singular Value Decomposition) algorithm for collaborative filtering in recommendation systems. It integrates both explicit and implicit feedback to generate personalized and accurate recommendations in general and also in recommending steam games to users.
In contrast to the traditional "SVD" method, "SVD++" takes into account user and item-specific biases in addition to the user-item ratings. It introduces a user bias vector and an implicit feedback component. The User Bias Vector captures individual user preferences and propensities for reviews to provide personalized recommendations. The implicit feedback component accounts for user interactions with items that are not explicitly rated, such as clicks or purchases. By combining explicit ratings, user bias and implicit feedback, "SVD++" improves the accuracy and relevance of recommendations[5]. It is iteratively trained to optimize the model parameters and minimize the prediction errors between the

observed and predicted scores. “SVD++” is suitable for both explicit and implicit feedback data and is used in various recommendation scenarios such as e-commerce or news recommendations. It uses both explicit and implicit signals to generate personalized and accurate recommendations based on the learned models[4].
By leveraging all available interactions - both explicit and implicit - modern recommender systems can provide more precise and customized recommendations.

2.4 Random algorithm

The ”random” algorithm is a simple approach that does not take any user-item interactions or patterns into account. It only assigns random ratings to the items. This algorithm serves as a basic benchmark used to compare it to more advanced recommendation algorithms. It allows evaluating the performance of other recommender systems and assessing their effectiveness.
Unlike more sophisticated algorithms, the ”random” algorithm does not require extensive training or tuning. It independently generates random ratings for each user-item pair. For this reason, it is often used as a starting point to set a minimum performance in recommendation systems, also in the recommending of steam games to users of the platform.
However, it is important to note that due to the random nature of the ratings, the ”random” algorithm does not take any user preferences or item characteristics into account. The recommendations it generates are therefore not personalized and may not reflect the interests of users.
Although the ”random” algorithm is not intended for practical use in real scenarios, it can serve as a starting point to develop more advanced recommendation models. Its main advantages are simplicity and easy implementation.
In terms of accuracy and relevance, the ”random” algorithm often performs worse than more advanced recommendation approaches. Therefore, it should be noted that it is primarily used for educational or baseline comparison purposes and is unable to generate meaningful recommendations in practical scenarios.

2.5 GridSearchCV

GridSearchCV (also known as Grid Search Cross Validation) is an algorithm to tune the hyperparameters of other algorithms. It doesn’t recommend items to users directly but instead helps with the process of optimizing an existing algorithm.
GridSearchCV works by testing every possible combination of provided hyperparameters. For each hyperparameter combination the model is fitted to the training data. Then cross validation is used to evaluate the model on a given testset. Lastly a metric such as RMSE is calculated and the hyperparameter combination that produces the the best results is chosen.
Within this work GridSearchCV was used to optimize the hyperparameters of the previously mentioned SVD++ algorithm.

3 Experimental setting

3.1 Dataset User Playtime

The first dataset contains the users playtimes and the product names. The dataset consists of a total of 70489 rows, with 11350 unique users and 3600 unique games.

If a matrix between users and products is created and the corresponding playtimes are filled out 99.8274% of the fields are empty. It's to be expected for this matrix to be sparse, but this case is very sparse compared to e.g. the often used movielens dataset (ML-100K) which is only 93.696% sparse [17].

An additional challenge is that 6559 users of the dataset only played one recorded game. The mean number of games per user 6.21 and the mean number of hours played 303.56, are skewed due to power users having large playtimes in a lot of different games. The median values for games per user and number of hours played respectively are 1 game per player and 19.5h. This might cause difficulties during the development of the recommendation system since game and user correlations will be difficult to estimate.

A similar powerlaw effect can be observed within the game data. Roughly a third of the games in the dataset have only one user (1018 games to be exact), but the top 1% of games is played by a large portion of the userbase.

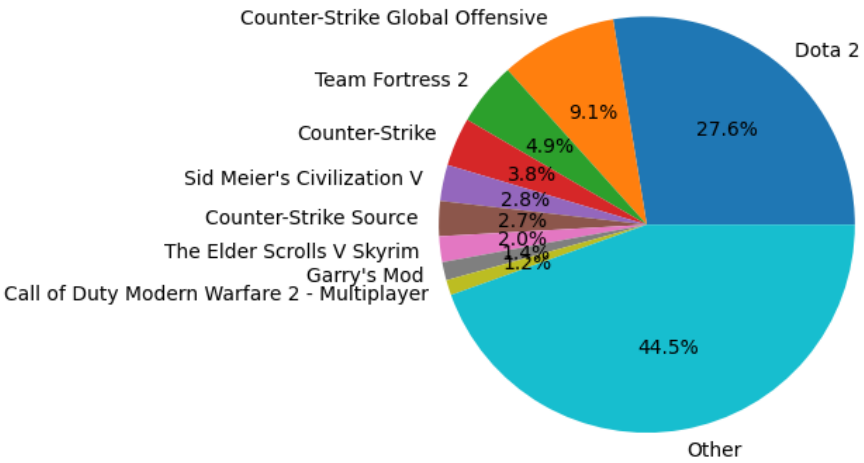

Fig. 1. Total hours spent by all players in games.

3.2 Python surprise library

Python Surprise is a popular open-source recommendation system library that simplifies the implementation and evaluation of collaborative filtering algorithms. It provides a convenient interface for building and analyzing recommendation systems based on user-item interactions. [15]

It offers a range of collaborative filtering algorithms, including matrix factorization-based methods like Singular Value Decomposition (SVD). While providing a clean and intuitive API for loading and handling datasets, it supports various file formats such as CSV, JSON, and custom formats, allowing users to easily import their own data. Surprise also provides utility functions for splitting datasets into training and testing sets, making evaluation of models very convenient.

For evaluation, Surprise also includes built-in evaluation metrics such as Root Mean Squared Error (RMSE) and others, which help assess the performance of recommendation models. Cross-validation techniques are available as well to estimate the generalization ability of models and avoid overfitting.

See the utilization of surprise in detail in the attached notebook.

3.3 Experiment implementation

Firstly, the main CSV file is read and written into a pandas dataframe. That data is then processed and prepared to be converted into a so-called dataset, offered by the surprise library. We create multiple different datasets, each utilizing a different method of deriving an implicit rating of users for specific games.

One method to do this is to define a set number of bins, each spanning a range of hours of playtime, and to then alter the dataset to contain a users implicit rating which is equivalent to the bin that the users playtime has been assigned to.

Table 1. Analysed datasets and their implicit rating derivation method.

Dataset	Implicit rating method	Rating scale
Dataset 1	3 equal-sized playtime bins	0 - 1
Dataset 2	5 equal-sized playtime bins	0 - 1
Dataset 3	10 equal-sized playtime bins	0 - 1
Dataset 4	20 equal-sized playtime bins	0 - 1
Dataset 5	portion of users total playtime	0 - 1
Dataset 6	portion of the games total playtime	0 - 1

Just using the hours of playtime as an implicit rating as given in the dataset without altering it leads to multiple issues with the result. For example, the resulting RMSE can be massive, due to an error of over 100 hours of playtime still being considered pretty small in the context of maximum playtimes reaching far over 10.000 hours. Similarly, if we just naively normalise the playtimes by dividing by the maximum playtime, the resulting RMSE can be extremely tiny.

Either way the recommender system won't be working well because most ratings in our range of interest, which could arbitrarily be defined as around 1 - 50 hours of playtime, will be compressed to a very small fraction of the total rating scale.

To mitigate this problem, we can bin the hours of playtime of each player to create an implicit rating on a range of 0 to 1. We will try this with 3 discrete rating values (0, 1/2, 1) and up to 20 discrete rating values. The bins are created in a way that each bin has the same amount of rating fall into it. To achieve this, the borders/cut-offs are generated automatically through the source dataset.

Table 2. Bin definitions.

Amount of bins	Boundaries	Discrete rating scheme
3 Bins	[0, 1.7, 11.6, 11754.0]	[0, 0.5, 1]
5 Bins	[0, 0.7, 2.6, 7.9, 28.0, 11754.0]	[0, 0.25, 0.5, 0.75, 1]
10 Bins	[0, 0.3, 0.7, 1.4, 2.6, 4.5, 7.9, 14.0, 28.0, 76.0, 11754.0]	[0, 0.11, 0.22, .., 0.88, 1]
20 Bins	omitted due to length	

Additionally, we can derive an implicit rating from 0 to 1 from the portion of a users total playtime that the user has dedicated to this game. This is how dataset 4 is built. A similar apporach we will also try is to take the portion of total hours spent on a game that a user has played it.

For each of these datasets with different implicit rating methods, 4 different algorithms from the surprise library will be fitted and evaluated. Then it will be possible to compare the performance of the different algorithms and how they behave differently with different rating methods.

The hyperparameters used for SVD, SVD++ and SVD++(tuned) are [100 factors, 20 epochs, 0.005 learning rate, 0.02 regularization], [20 factors, 20 epochs, 0.007 learning rate, 0.02 regularization] and [20 factors, 30 epochs, 0.005 learning rate, 0.2 regularization] respectively.

3.4 Experiment results

We added a random selection algorithm to compare the other algorithms against. RMSE performance between different Implicit rating derivation schemes cannot really be compared because they stem from different datasets (The source dataset has been altered in a different way for each of these).

The first thing to notice is the steadily declining RMSE as the number of bins increases. This has a mathematical reason and is not indicative of more bins improving the recommender system. In fact, it can be observed that independently of the number of bins, all the algorithms seem to behave the same and to yield the same improvement over the 'random' which was added for comparison.

Deriving an implicit rating from a users portion of their total playtime seems to be yielding the most pleasing results. The RMSE after applying KNN Baseline on this dataset has the best improvement over the 'random' algorithm added for comparison, almost halving it.

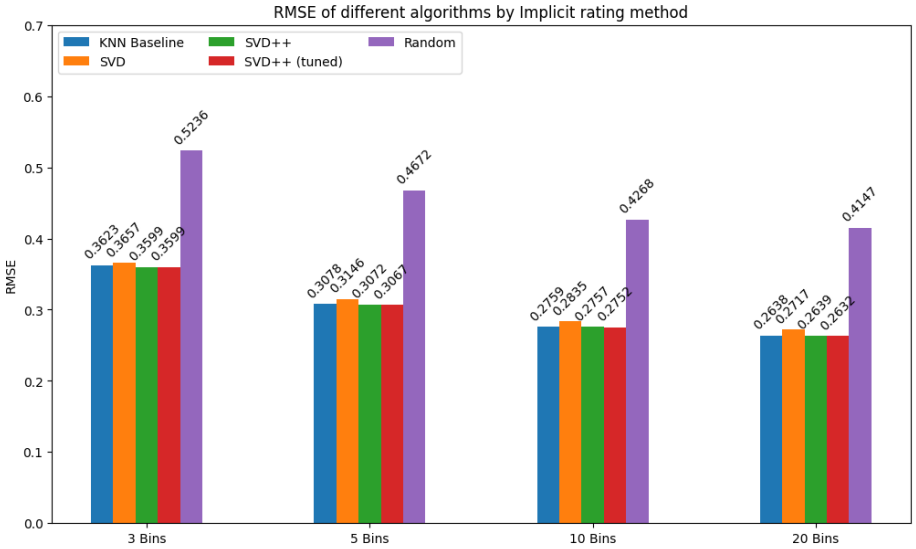

Fig. 2. RMSE values of algorithms by binning method

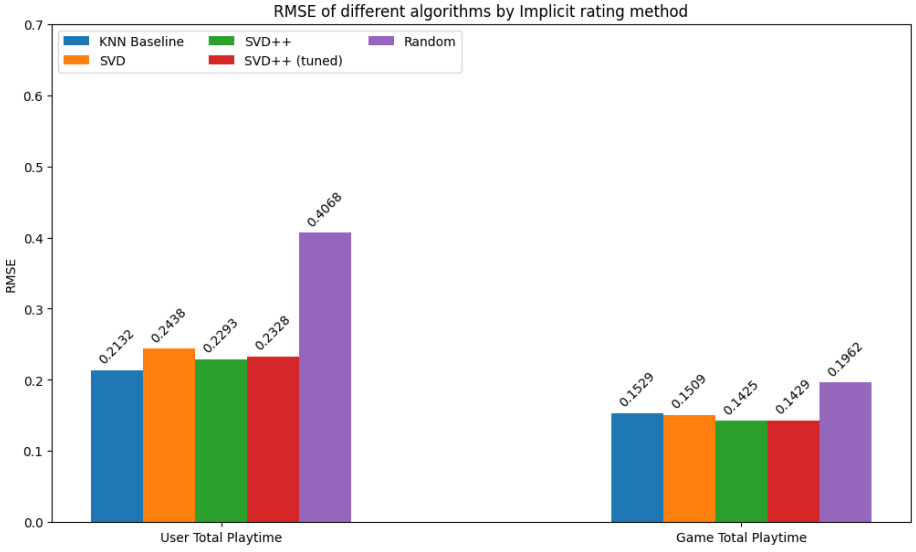

Fig. 3. RMSE values of algorithms by other implicit rating methods

Fig. 4. Total result table

Preprocessing	Metric	KNNBaseline	SVD	SVD++	SVD++ (tuned)	Random
3 Bins	RMSE	0,3623	0,3669	0,3606	0,3599	0,5250
	MSE	0,1313	0,1346	0,1300	0,1295	0,2756
	MAE	0,3000	0,3079	0,3021	0,3051	0,4286
	FCP	0,6349	0,6252	0,6305	0,6328	0,4772
5 Bins	RMSE	0,3078	0,3146	0,3071	0,3067	0,4647
	MSE	0,0948	0,0990	0,0943	0,0941	0,2159
	MAE	0,2530	0,2617	0,2553	0,2577	0,3801
	FCP	0,6291	0,6028	0,6248	0,6284	0,4972
10 Bins	RMSE	0,2759	0,2841	0,2758	0,2751	0,4303
	MSE	0,0761	0,0807	0,0760	0,0757	0,1852
	MAE	0,2260	0,2350	0,2280	0,2304	0,3517
	FCP	0,6240	0,5997	0,6204	0,6224	0,4943
20 Bins	RMSE	0,2638	0,2714	0,2634	0,2630	0,4095
	MSE	0,0696	0,0737	0,0694	0,0692	0,1677
	MAE	0,2157	0,2240	0,2174	0,2198	0,3343
	FCP	0,6156	0,5968	0,6077	0,6142	0,5019
User Playtime	RMSE	0,2132	0,2429	0,2296	0,2328	0,4045
	MSE	0,0455	0,0590	0,0527	0,0542	0,1636
	MAE	0,1100	0,1400	0,1382	0,1354	0,2868
	FCP	0,3804	0,4387	0,4587	0,5491	0,4584
Game Playtime	RMSE	0,1529	0,1503	0,1424	0,1429	0,1980
	MSE	0,0234	0,0226	0,0203	0,0204	0,0392
	MAE	0,0551	0,0682	0,0587	0,0557	0,1173
	FCP	0,6540	0,5700	0,6310	0,6436	0,4424

3.5 Trying the recommender system on our own Steam profiles

To judge the recommender system on a more personal basis we calculated the recommendations for the steam profile of one of our project members can be found in Table 3:

Table 3. The Steam account.

Game name	playtime (hours)
Space Engineers	860.0
Tom Clancy's Rainbow Six Siege	856.7
Grand Theft Auto V	608.8
Arma 3	522.3
Counter-Strike: Global Offensive	426.1
Robocraft	350.5
The Elder Scrolls V: Skyrim Special Edition	325.4
ARK: Survival Evolved	216.0
Factorio	215.8
Unturned	147.5
Just Cause 3	143.2
Satisfactory	131.9
ELDEN RING	111.5
Shakes and Fidget	110.7
The Witcher 3: Wild Hunt	104.0
Hollow Knight	96.4
SCP: Secret Laboratory	76.6
Terraria	73.9
Executive Assault	68.4
Cyberpunk 2077	60.8

The resulting predicted ratings (normalized between 0 and 1) can be found in Table 4.

It can be observed that the games being rated the highest are mostly just the games with the highest average playtime across all players from our dataset. This is indicative of the fact that the recommendation model likely needs to be trained with more data to then actually be able to give more personalized video game recommendations. As it stands, at least for this one experiment, the recommendations by our system appear pretty generalized and with room for improvement. Additionally recommendations for games that have multiple release versions per year still need to be reduced to prevent flooding the users feed. We don't want to completely remove them since in some cases older games are preferred by users over newer releases.

Table 4. The resulting recommendations for our Steam account.

Game name	predicted rating (0-1)
Dota 2	0.573191580088516
Professional Farmer 2014	0.44870583499856337
Football Manager 2012	0.4127281078709767
Football Manager 2009	0.39481872519164296
Football Manager 2013	0.37636519848734185
Star Trek D-A-C	0.36615819265684013
NBA 2K9	0.3630547645419008
Pro Evolution Soccer 2015	0.3576379489248566
Football Manager 2010	0.3490678433503833
F1 2012	0.3477340109740507

4 Evaluation and discussion of the results

As we can see in Fig. 2 and Fig. 3, all algorithms outperform a random recommendation. The discretization of the playtime in 20 bins resulted in a lower RMSE than dividing it to 3, 5 or 10 bins, however this result may be due to the effect of the predictions simply being closer to the bins and therefore lowering the residuals. An indication for this is that the average improvement throughout the models is about 0.1, but also the random recommendations score a about 0.1 lower RMSE with 20 bins.

Since Steam is, as mentioned, a widely used gaming Platform, there has already been some research done to create or improve recommender systems based on the publicly available data. Unfortunately Steams own algorithm is not publicly available.

One approach we found on kaggle used a hybrid aproach of combinded collaborative and content-based filtering models, but failed to provide evaluation metrics.[12]

Another hybrid approach consisting of a Popularity-based recommendation, Quality-based Recommendation, Content-based Recommendation and Collaborative-filtering Recommendation achieved a RMSE of 3.60 [8] [9]

A third approach also used binning to create implicit ratings (in this case stars from 1-5) and evaluated SVD++ and KNN.[7] It reached a NRMSE of 0.1913 (rounded) for SVD++ and 0.1906 (rounded) for KNN. Since the NRMSE is calculated by

$$NRMSE = \frac{RMSE}{r_{max} - r_{min}}$$

and after our normalisation the min and max values are 0 and 1, our RMSE equals the NRMSE and we can compare our results directly to the results of this work. We achieve slightly worse results with a best NRMSE of 0.2638 for KNN and 0.02634 for SVD++, but it must be noted that the underlying dataset in

that work was webscraped in 2016, thus not being completely congruent with the one used in this project.

Other published recommender systems for steam games use more data about users and games, like game description and in-game trophys and -while performing better- allow therefore only a very limited comparison. [13][10]

5 Conclusions

In this project, we have explored the creation of a recommender system for suggesting games to users based on their interests. The aim was to provide a more efficient and personalized user experience by reducing the time and effort required to find relevant games on the Steam platform.

To evaluate the performance of our recommender systems, we compared them against a random selection algorithm. The random selection algorithm served as a baseline for comparison, demonstrating the effectiveness of our approach in generating more relevant game recommendations. However, it's important to note that comparing the performance between different implicit rating derivation schemes, such as RMSE, may not be suitable due to the variations in the datasets and alterations made.

While our recommender systems provide an effective solution for suggesting games, there is always room for improvement. Future enhancements could include incorporating more advanced machine learning techniques, such as neural collaborative filtering [16] or content-based filtering, to further refine the recommendations. Additionally, gathering more explicit user feedback and ratings could enhance the accuracy of the recommendations. Additional rule based filters for special cases such as add-on products being only recommended if the user owns the base product, only recommending products that are compatible with the users device or preventing too similar products from flooding the users recommendations could also improve the user experience. The greatest benefit and future improvement, however, could be drawn from using a better, bigger dataset with lower sparsity. We think the high sparsity of our dataset has hindered our recommendation system the most.

From a business perspective, another enhancement could be to incorporate the different prices of games or purchase history of the users and e.g. recommend the most expensive of the recommended games to users that seem to be willing to spend more money based on their purchase history. These ideas could be the topic of future research.

References

1. Statista, Number of games released on Steam worldwide from 2004 to 2022 https://www.statista.com/statistics/552623/number-games-released-steam/
2. Medium, K-Nearest Neighbor https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4
3. Koren, Y.: Factorization meets the neighborhood: A multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM (2008)
4. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web. ACM (2001)
5. Paterek, A.: Improving regularized singular value decomposition for collaborative filtering. In Proceedings of the 2007 ACM conference on Recommender systems. ACM. (2007)
6. Salakhutdinov, R., Mnih, A: Probabilistic matrix factorization. In Advances in neural information processing systems (2008)
7. Gabmeier, J.: Recommender Systems in the Domain of Video Games. Graz University of Technology (2016)
8. Analyzing Steam Reviews and Users Data, https://towardsdatascience.com/steam-recommendation-systems-4358917288eb. Last accessed 25 May 2023
9. Steam Recommendation System, https://github.com/AlbertNightwind/Steam-recommendation-system. Last accessed 25th May 2023
10. Sifa, R, Bauckhage, C, Drachen: Archetypal Game Recommender Systems. In: LWA 2014 (pp. 45–56).
11. Analyzing Steam Reviews and Users Data https://medium.com/@jonduke90/analyzing-steam-reviews-and-users-data-7a4ff3c5ce1a, Last accessed 25 May 2023
12. Game Recommendations on Steam — Kaggle https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam Last accessed 25 May 2023
13. Notten, Bram: Improving Performance for the Steam Recommender System using Achievement Data. Tilburg University (2017)
14. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
15. Nicolas Hug, Surprise: A Python library for recommender systems, Journal of Open Source Software 2020, https://doi.org/10.21105/joss.02174
16. Neural Collaborative Filtering, He, Xiangnan and Liao, Lizi and Zhang, Hanwang and Nie, Liqiang and Hu, Xia and Chua, Tat-Seng https://dl.acm.org/doi/10.1145/3038912.3052569
17. GroupLens, Movielens Dataset https://grouplens.org/datasets/movielens/