Repositório ISCTE-IUL

From Implicit Preferences to Ratings: Video Games Recommendation based on Collaborative Filtering

Rosária Bunga[1], Fernando Batista[1,2]🆔[a] and Ricardo Ribeiro[1,2]🆔[b]

[1]*Iscte - Instituto Universitário de Lisboa, Av. das Forças Armadas, Portugal*
[2]*INESC-ID Lisboa, Portugal*
{*Rosaria_Patricia_Bunga, fernando.batista, ricardo.ribeiro*}*@iscte-iul.pt*

Abstract: This work studies and compares the performance of collaborative filtering algorithms, with the intent of proposing a videogame-oriented recommendation system. This system uses information from the video game platform *Steam*, which contains information about the game usage, corresponding to the implicit feedback that was later transformed into explicit feedback. These algorithms were implemented using the Surprise library, that allows to create and evaluate recommender systems that deal with explicit data. The algorithms are evaluated and compared with each other using metrics such as RSME, MAE, Precision@k, Recall@k and F1@k. We have concluded that computationally low demanding approaches can still obtain suitable results.

1 Introduction

There has been a rapid growth of content, products, and services provided by sites like Google, Youtube, Amazon, Netflix, Steam, among others. The great diversity of information available in the Internet easily began to overwhelm users, leaving them indecisive and thus hindering the decision-making process. This large amount of information, instead of generating a benefit, became a problem for users. While choice is good, more choice is not always better (Ricci et al., 2011). Then, as this phenomenon intensified, the more important became to help users filtering the relevant items from a whole range of available alternatives, in order to facilitate the choice of products or services that best suited them. To minimize or solve this information filtering problem, recommendation systems emerged and started to assist the decision making process by providing personalized recommendations to users (Jannach et al., 2011).

Recommendation is something we are all familiar with, whether a friend recommends a new book to read or a movie to watch, it is all about giving good options and helping you make a choice. These recommendations are usually given based on knowledge about what we like. Recommendation systems work in exactly the same way: the system tries to use the users history or profile to predict which products or services to recommend. All of the platforms mentioned above have started implementing some sort of recommendation system that caters to the needs of their users, providing personalized content with the goal of providing the user with a better experience, thereby increasing user loyalty, selling more and diverse products, and ultimately, improving the revenue of these companies (Ricci et al., 2011).

In this work, we focus on the video games domain, mainly due to the exponential growth of this market and its particularities. As case study, we use the video game platform *Steam*[1]. According to *Steam*'s website, they currently provide 30,000 different games for Windows, Mac, and Linux operating systems. In such a fast-moving market, which has a large amount of classic games, new releases, and indie games, dozens of games are coming out every month. Given such multitude of choice, users/players find it very difficult to find new games they might be interested in.

With a market as diverse as that, there is clearly a need to implement recommendation systems with high accuracy that provide users with relevant unknown games and/or new releases that cater to their tastes. This becomes even more evident, when we look at *Steam*'s 2014 records, where about 37% of games purchased were never played by the users who bought them. A user plays his favorite games often,

---

[a]🆔 https://orcid.org/0000-0002-1075-0177
[b]🆔 https://orcid.org/0000-0002-2058-693X

---

[1]https://store.steampowered.com

but also wants to discover new games that are relevant to him. This presents itself as a form of challenge for this market: the need for video games that encourage the user to come back and help users find new games that will be consumed as much as the ones they already liked. Recommendation systems can greatly benefit the video game market by their ability to suggest new games to users in a personalized way. Nevertheless, there is still much to be explored about recommendation systems in the video game domain.

The goal of this work is to develop a recommendation system that provides video game suggestions to a user based on his gaming history and the tastes of users similar to him. We will explore several approaches, based on different collaborative filtering algorithms, using data from the *Steam* platform. The major challenge is on understanding how to use implicit data to infer explicit ratings of the users.

We aim to answer the following research questions with reference to the video game domain:

RQ1 What is the performance of different collaborative filtering recommendation approaches, when we applied on implicit data?

RQ2 Can playing time of the users serve as an adequate implicit representation of the users preferences?

This paper is organized as follows: Section 2 overviews the existing literature concerning collaborative filtering approaches and video game recommendation. Section 3 presents the dataset. Section 4 describes the process of obtaining explicit ratings from the existing information. Section 5 overviews the the recommendation approaches used in this work. Section 6 analyses and discusses the obtained results. And, finally, Section 7 presents the overall conclusions and pinpoints future work directions.

2 Related Work

Collaborative filtering based recommendation approaches are still an active topic in this research area. Pérez-Marco et al. (Pérez-Marcos et al., 2020) present a hybrid video game recommendation system, through the use of collaborative filtering and content-based filtering. The system takes as input a list of ratings of a user, the communities of the games the system contains, and the communities of the users the system contains. Content-based filtering is applied first: for each item of the active user, the system searches for the games that are in the closest community to that item. As a result, a list of the games

that are most similar to those of the active user is obtained. Then, collaborative filtering is applied, restricted to the items obtained in the previous step, using ratings. In the end, a matrix with the recommended games and their predicted ratings is returned. In the case of content-based filtering, they use graph-based techniques to find games similar to those of the active player, reducing the computational load of the system because recommendations are made on a subset of items. Anwar et al. (Anwar et al., 2017) propose a recommendation system that uses a collaborative filtering technique to suggest video games to users. The recommendation system was implemented using item-based collaborative filtering and Pearson correlation to identify unrated games, finding similarities between unrated and highly rated games by the user. Next, the system employs user-based collaborative filtering. However, the system was not able to provide a better accuracy in the case of the cold boot scenario.

Many traditional collaborative filtering algorithms fail to consider that Users' preferences often vary over time. (Joorabloo. et al., 2019) proposes a recommendation method that predicts the similarity between users in the future, and forecasts their similarity trends over time. Experimental results show that the proposed method significantly outperforms classical and state-of-the-art recommendation methods.

Given its current popularity, artificial neural networks-based approaches are also being explored as recommendation methods. STEAMer (Wang et al., 2020), a video game recommendation system for the *Steam* platform, uses *Steam* user data in conjunction with a Deep Autoencoders learning model (a specific type of neural network architecture that attempts to force the network to learn a compressed representation of the original input data). Cheuque et al. (Cheuque et al., 2019) propose recommendation models based on Factorization Machines (FM), Deep Neural Networks (DeepNN) and a combination of both (DeepFM), chosen for their potential to receive multiple inputs as well as different types of input variables. All algorithms achieve better results than an ALS (Alternating Least Squares Model) baseline. Despite being a simpler model than DeepFM, DeepNN was found to be the best performing algorithm: it was able to better exploit user-item relationships, achieving consistent results on different datasets. They also analyzed the effect of sentiment extracted directly from game reviews and concluded that it is not as relevant for recommendation as one might expect.

Closer to our work, also exploring implicit feedback, Bertens et al. (Bertens et al., 2018) propose a

system that recommends video games to users based on their experience and behavior, i.e., playing time and frequency of activity. They explore two models: Extremely Randomized Trees (ERTs) and DeepNNs. The results show that the prediction performance of DeepNNs and ERTs is similar, with the ERT model achieving a marginally better performane and scaling more easily in a production environment. Pathak et al. (Pathak et al., 2017) proposed a recommendation system based on latent factor models, in particular Bayesian Personalized Ranking (BPR), that is trained using implicit ranking (i.e., purchases vs. no purchases), and that uses the trained features of an item recommendation model to learn personalized rankings over bundles. The authors showed that the model is robust to cold bundles and that new bundles can be generated effectively using a greedy algorithm. Their main focus is on item-to-group compatibility, generating and evaluating custom package recommendation on the *Steam* video game platform. Finally, Sifa et al. (Sifa et al., 2015) present two approaches for Top-*N* recommendation systems: a matrix factoring-based model and a user-based neighborhood model operating in low dimensions. Both models are based on archetypal analysis, a method similar to cluster analysis, thus grouping users into archetypes. The data used for this analysis is composed of implicit feedback, specifically information about game ownership and play times, with the ultimate goal of recommending games that have the longest predicted play time. The authors compare their algorithm to several baselines and an item-based neighborhood model, which they were able to outperform.

3 Dataset

In this work we use the "Steam Video Game and Bundle Data"[2] shared by Pathak et al. (Pathak et al., 2017). The dataset consists of the purchase history of Australian users of the *Steam* platform, indicating for each one the list of items purchased with a small collection of metadata related to game playing time. We expanded each users item list or item bundle from the original dataset, so that each record in the set could be viewed as a user/item interaction.

Table 1 shows the attributes of the original dataset. As it is possible to observe, there is no rating that can be used as an explicit measure of users preference, i.e., our dataset has no explicit ratings. Therefore, for experimenting the recommendation algorithms we resorted to implicit information.

For our experiments, we filtered the dataset: we have removed all players with less than 30 games and all games that were played by less than 10 players. Only games that had a playing time of more than zero minutes were considered. This data filtering step led to a new dataset containing 33,307 unique users and 6,387 unique games, with about 2.8 million records.

4 From Implicit Preferences to Ratings

As previously described, the dataset only provides information related to game playing time. However, for collaborative filtering algorithms we need user ratings. Most approaches to understanding user preferences are based on having explicit user ratings. However, in many real-life situations, we need to rely on implicit ratings, such as how many times a user has listened to a song or played a game. Considering the importance mentioned in the literature on the relationship between explicit and implicit ratings in recommendation systems (Parra and Amatriain, 2011; Yi et al., 2014), we chose the total playing time, "*playtime_forever*", to infer the users explicit ratings, and to understand the users preferences. Although implicit ratings can be collected constantly and do not require additional efforts from the user when inferring the users preferences from their behavior, one cannot be sure, if that behavior is interpreted correctly. Still, Schafer et al. (Schafer et al., 2006) report that in some domains, such as personalized online radio stations, collecting implicit ratings can even result in more accurate user profiles than what is achieved with explicit ratings. Furthermore, it has also been discussed that implicit preference data may actually be more objective, since there is no bias arising from users responding in a socially desirable way (Buder and Schwind, 2012). This also emphasizes that the proper interpretation of implicit ratings can be highly dependent on the domain.

We focus on the total playing time, in minutes, *playtime_forever*, to quantify the relevance of an item to a specific user. Our assumption is that if a user plays a game for a long time, he likes that game. For the task of converting implicit ratings into explicit ratings, we applied the Python `cut()` function to group the game times into five intervals. Then we applied the `rank()` function to assign a rank – equal values are assigned a rank that is the average of the ranks of those values. Our ranking system uses a scale ranging from 1 to 5. Table 2 contains an excerpt of the final result, where *pt_2weeks* and *pt_forever* correspond to *playtime_2weeks*, and *playtime_forever*, respectively.

Table 1: Dataset attributes.

Attribute	Type	Description
user_id	string	User
steam_id	integer	Steam user identification
user_url	string	User URL
item_id	integer	Game identification
item_name	string	Game title
playtime_2weeks	integer	Number of minutes played in the last two weeks
playtime_forever	integer	Total number of minutes played

Table 2: Excerpt of the final dataset, containing only the most relevant fields.

	steam id	item_id	item name	pt_2weeks	pt_forever	rating
2794716	...5400	45770	Dead Rising 2: Off the Record	0	72	3
2794717	...5400	46510	Syberia 2	0	40	2
2794718	...5400	466500	35MM	0	19	1
2794719	...5400	55230	Saints Row: The Third	0	1029	5
2794720	...5400	6300	Dreamfall: The Longest Journey	57	68	2
2794721	...5400	6310	The Longest Journey	42	42	2
2794722	...5400	72850	The Elder Scrolls V: Skyrim	0	2842	5
2794723	...5400	7670	BioShock	0	219	4
2794724	...5400	8850	BioShock 2	0	131	3
2794725	...5400	8870	BioShock Infinite	0	438	4

5 Approaches to Recommendation

There are three common methods for recommendation systems in the literature: content-based filtering, collaborative filtering, and hybrid filtering. In this paper, we focus on collaborative filtering. In collaborative filtering, recommendations for each user are generated by making comparisons with their liking for an alternative against other users who have rated the product similarly (Shah et al., 2017). A common approach is to split the dataset into two sets: the training set and the test set. A recommendation model is built with respect to the training set. And the test set, in turn, is subdivided into two: the query set and the target set. Based on the query set, the model suggest items or predict ratings for the items in the target set. For the implementation of the various recommendation algorithms, we use Surprise, Simple Python Recommendation System library (Hug, 2020). This library provides several recommendation algorithms and tools to evaluate, analyze, and compare algorithms.

5.1 Nearest Neighbors-based Algorithms

Neighborhood-based algorithms use user-user similarity or item-item similarity to make recommendations from a ratings matrix (Aggarwal, 2016). KNNBasic is a neighborhood-based collaborative filtering algorithm. The concept of neighborhood implies that we need to determine similar users or similar items to make predictions. The prediction $\hat{r}_{ui}$ is defined in Equation 1, where $\text{sim}(u,v)$ can be calculated based on the cosine similarity, based on Pearson's correlation coefficient, or based on the similarity of the mean squared difference. $\hat{r}_{ui}$ indicates a predicted ranking, as opposed to one that has already been observed in the original rank matrix. $N_i^k(u)$ represents the set of k users closest to the target user u, who specified ratings for item i.

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v)} \qquad (1)$$

KNNWithMeans is identical to KNNBasic, but takes into account the average ratings of each user. The weighted average of the mean-centric rating of an item in the top-k peer group of the target user u is used to provide a mean-centric prediction. The average rating of the target user is then added back to this prediction to provide a raw rating prediction $\hat{r}_{ui}$ of the target user u, for item i, as defined in Equation 2. There is also an item-based version of this algorithm, in which we check the k closest items rated by user u and use their ratings and similarities to item i to predict the rating of item i (Mittal and Subraveti, 2017).

$$\hat{r}_{ui} = \mu_i + \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j) \cdot (r_{uj} - \mu_j)}{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j)} \quad (2)$$

5.2 Algorithms based on Matrix Factorization

In its basic form, matrix factorization characterizes items and users by vectors of factors inferred from item rating patterns. A high correspondence between item and user factors leads to a recommendation. These algorithms have become popular in recent years, combining good scalability with predictive accuracy. In addition, they offer a lot of flexibility to model various real-life situations. Recommendation systems rely on different types of input data, which are usually placed in a matrix with one dimension representing the users and the other dimension representing the items of interest. The most convenient data is high quality explicit ratings, which includes explicit input from users about their interest in products (Koren et al., 2009). The Singular Value Decomposition (SVD) maps users and items to a common latent factor space of dimensionality f, so that user-item interactions are modeled as inner products in that space. The latent space attempts to explain ratings by characterizing products and users into factors automatically inferred from user feedback. Thus, each item i is associated with a vector $q_i \in R^f$, and each user u is associated with a vector $p_u \in R^f$. For a given item i, the elements of q_i measure the extent to which the item has these factors, positive or negative. For a given user u, the elements of p_u measure the extent of interest the user has in items that are high in the corresponding factors, again, they can be positive or negative. The resulting scalar product, $q_i^T p_u$, captures the interaction between user u and item i, i.e., the users overall interest in the features of the item. The final rating is created by also adding baseline predictors that depend only on the user or item. Equation 3 shows how the prediction is calculated, where μ is the global average, b_u is the user tendency, b_i is the item tendency, and $q_i^T p_u$ is the interaction between the user and the item (Koren et al., 2009).

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \quad (3)$$

SVD++ is an extension of SVD that takes implicit ratings into account. In cases where independent implicit feedback is absent, a significant signal can be captured by considering which items users rate, regardless of their rating value. This has led to several methods that model a users factor by the identity of the items he rated. One of these is SVD++, which has been shown to perform better than SVD. For this purpose, a second set of item factors is added, relating each item i, to a vector of factors $y_i \in R^f$. These new item factors are used to characterize users based on the set of items they have rated. The prediction is given by Equation 4, where R_u is the items evaluated by user u, and u is modeled as $p_u + |R_u|^{-\frac{1}{2}} \sum_{j \in R_u} y_j$. p_u is learned based on the provided explicit classification.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |R_u|^{-\frac{1}{2}} \sum_{j \in R_u} y_j \right) \quad (4)$$

Non-Negative Matrix Factorization (NMF) is a matrix factorization algorithm, in which the user-item matrix is decomposed into user and item factors, that have non-negative values. The user and item factors are initialized with random values and the optimization is done by stochastic gradient descent (SGD). This algorithm is highly dependent on the initialized values for the factors. User and item baselines can also be incorporated, but the model becomes susceptible to oversizing, which, however, can be controlled by a good choice of the regularization parameter (Luo et al., 2014). The prediction is given by Equation 5.

$$\hat{r}_{ui} = q_i^T p_u \quad (5)$$

5.3 SlopeOne

It works on the intuitive principle of a popularity differential between items for users. In pairs, it determines how much better one item is liked than another. One way to measure this differential is to simply subtract the average rating of the two items (Jannach et al., 2011). In turn, this difference can be used to predict another users rating of one of these items, given the rating of the other. Many such differentials exist in a training set for each unknown classification and an average of these differentials can be used. Thus, given a training set and any two items i and j with ratings r_{ui} and r_{uj}, respectively, in some user evaluation u the average deviation of item i from item j is given by Equation 6.

$$\text{dev}(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj} \quad (6)$$

The algorithm considers that each evaluation of user u that does not contain r_{ui} and r_{uj} will not be included in the sum. U_{ij} is the set of all users that classified items i and j. The prediction is given by Equation 7 (Lemire and Maclachlan, 2005).

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i,j) \quad (7)$$

5.4 Co-Clustering

Clustering is an unsupervised learning technique that seeks to group similar objects together. The main idea of this algorithm is to simultaneously obtain user and item neighborhoods by co-clustering and generate predictions based on the average ratings of the co-clusters (user-item neighborhoods), taking into account individual user and item trends. Users and items are assigned to clusters C_u and C_i, and co-clusters C_{ui}, and the prediction is given by Equation 8, where $\overline{C_{ui}}$ is the average classification of co-cluster C_{ui}, and $\overline{C_u}$ and $\overline{C_u}$ are the average classification of co-clusters C_u and C_i, respectively (Lemire and Maclachlan, 2005).

$$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i}) \qquad (8)$$

6 Analysis and Discussion of Results

The proposed algorithms were evaluated in terms of MAE, RMSE, Precision@k, Recall@k and F1@k. The dataset has a total of 2,794,726 records, each with unique information about the user-item interactions. This set was divided, in the proportion of 80/20. Being 80% for the training set (2,235,780) and 20% for the test set (558,946). The test set was subdivided into a query set of 10% and a target set of 10%.

6.1 Evaluation Metrics

Recommendation system research has used various types of metrics to evaluate the quality of a recommendation system. We use metrics that calculate the error between the actual classification and the predicted user classification, such as MAE and RMSE. From these metrics it is possible to infer how close the predicted values are to the actual ratings given by users. However, accuracy metrics are not suitable for measuring the classification performance of recommendation systems. Therefore, the study also conducted an evaluation based on Precision@k, Recall@k and F1@k ranking metrics. These metrics take into account the number of true positives (TP), false negatives (FN), false positives (FP) and true negatives (TN) present in the recommendation list.

MAE The Mean Absolute Error is the average deviation of the recommendations from their actual user-specified values. It is the average over the test sample of the absolute differences between the prediction and the actual observation, where all individual differences have equal weight. The

Table 3: Results achieved for each one of the methods.

	RMSE	MAE
KNNBasic	1.2270	1.0159
KNNWithMeans	1.2483	1.0356
SVD	1.2386	1.0049
SVD++	1.2179	0.9727
NMF	1.2229	1.0011
SlopeOne	1.1977	0.9831
CoClustering	1.2354	1.0212

smaller the MAE, the more accurately the recommendation engine predicts the users ratings (Sarwar et al., 1998);

RMSE The Root Mean Square Error is the square root of the mean of the difference between predicted and actual ratings (Afoudi et al., 2019).

Precision@k is the proportion of recommended items in the top-k set that are relevant. A high precision@k score indicates that most of the recommendations are relevant;

Recall@k is the fraction of relevant items recommended;

F1@k is a way to combine precision@k and recall@k of the model, and is defined as the harmonic mean of the precision and recall of the model.

6.2 Results

We calculated the RMSE and MAE to evaluate the performance, for all the collaborative filtering algorithms we set out to implement, using k-fold cross-validation with $k = 5$.

Table 3 shows the results for each algorithm. We find that for this dataset the algorithms have similar performances. However, the best results were achieved by SlopeOne and SVD++. Considering the RMSE, SlopeOne exhibited the best result when compared to the other algorithms, followed by SVD++. With the predicted ranks deviating from the actual ranks ≈ 1.1977 for SlopeOne and ≈ 1.2179 for SVD++. Concerning the MAE, SVD++ showed the lowest MAE value (≈ 0.9727) followed by SlopeOne (≈ 0.9831). For these metrics, KKNWithMeans had the worst results in both metrics. The distribution of the predicted ratings on the test set is noticeably different for all algorithms, not reflecting the actual distribution of implicit ratings.

Table 4 compares precision@k, recall@k, and f-measure@k for different k on the target test set. Concerning precision, SlopeOne achieved the best result – when considering precision@5, 78% of the recommendations made by SlopeOne are relevant to the

Table 4: Values of precision@*k*, recall@*k*, and F1@*k*, with *k* = 5, 10, 20.

	precision			recall			F1		
	@5	@10	@20	@5	@10	@20	@5	@10	@20
KNNBasic	0.7374	0.7144	0.7064	0.2866	0.3718	0.3954	0.4128	0.4891	0.5070
KNNWithMeans	0.7293	0.7043	0.6955	0.2767	0.3595	0.3841	0.4012	0.4760	0.4951
SVD	0.7619	0.7162	0.7003	0.3367	0.4609	0.5028	0.4670	0.5608	0.5854
SVD++	0.7757	0.7309	0.7145	0.3397	0.4640	0.5095	0.4725	0.5677	0.5948
NMF	0.7525	0.7073	0.6920	0.3275	0.4470	0.4882	0.4564	0.5478	0.5725
SlopeOne	0.7840	0.7336	0.7183	0.3391	0.4535	0.4920	0.4735	0.5605	0.5840
Co-Clustering	0.7713	0.7288	0.7150	0.3174	0.4229	0.4558	0.4498	0.5352	0.5567

user (71% for precision@20). Concerning recall, we found that SVD++ performed better ($\approx$ 0.5 for recall@20), although there is not much difference with SVD.

We observed that SlopeOne and SVD++ achieved the best results for all metrics. Concerning the average running time for training and testing of the evaluated recommendation algorithms, SlopeOne takes about 24 second to train, and takes about 77 seconds for testing, in contrast with SVD++, which takes about 4903.77 seconds for training, and 110 seconds for testing.

7 Conclusions and Recommendations

In this work, we addressed the topic of recommendation systems in the video game domain, based on implicit user information. We compared recommendation algorithms based on collaborative filtering, using user data from the *Steam* platform. We performed data pre-processing tasks before training the recommendation algorithms and the conversion of implicit information into explicit ratings. This conversion was based on the use of total playing time as an implicit rating, transforming it into an explicit rating, ranging from 1 to 5.

We explored seven recommendation algorithms, using the Surprise library, with *k*-fold, *k* = 5, cross-validation. To assess the performance of the different recommendation approaches, when using implicit information, we used RMSE, MAE, precision@*k*, recall@*k* and F1@*k*, common metrics used in the evaluation of recommendation systems. The results show that SlopeOne and SVD++ were the best performing recommendation algorithms. Taking into account the computational cost of each recommendation algorithm, SlopeOne seems to be the top contender.

RQ1 What is the performance of different collaborative filtering recommendation approaches, when

we applied on implicit data? The results show that for the used dataset there is not a significant difference between the explored algorithms.

RQ2 Can users playing time serve as an adequate implicit representation of the users preferences? The results show that it is possible to use playtime to infer explicit ratings for making recommendations in the video game domain and specifically on the *Steam* platform.

The presented algorithms, besides being of easy applicability and lower computational cost when compared to more complex algorithms, can produce good recommendations. For future work, it is important to explore other methods of transforming playing time into explicit ratings. Another possibility is to explore other types of implicit expression of preferences, assuming that they are available.

ACKNOWLEDGEMENTS

REFERENCES

Afoudi, Y., Lazaar, M., and Al Achhab, M. (2019). Collaborative filtering recommender system. In *Advances in Intelligent Systems and Computing*, volume 915, pages 332–345. Springer.

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1 edition.

Anwar, S. M., Shahzad, T., Sattar, Z., Khan, R., and Majid, M. (2017). A game recommender system using collaborative filtering (GAMBIT). *Proceedings of 2017 14th International Bhurban*

Conference on Applied Sciences and Technology, IBCAST 2017, pages 328–332.

Bertens, P., Guitart, A., Chen, P. P., and Perianez, A. (2018). A Machine-Learning Item Recommendation System for Video Games. In *IEEE Conference on Computatonal Intelligence and Games, CIG*, volume 2018-Augus.

Buder, J. and Schwind, C. (2012). Learning with personalized recommender systems: A psychological view. *Computers in Human Behavior*, 28(1):207–216.

Cheuque, G., Guzmán, J., and Parra, D. (2019). Recommender systems for online video game platforms: The case of steam. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, 2:763–771.

Hug, N. (2020). Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174.

Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2011). *Recommendation system: An Introduction*, volume 91. Cambridge University Press, New York.

Joorabloo., N., Jalili., M., and Ren., Y. (2019). A new temporal recommendation system based on users' similarity prediction. In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KDIR,*, pages 555–560. INSTICC, SciTePress.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, pages 471–475.

Luo, X., Zhou, M., Xia, Y., and Zhu, Q. (2014). An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284.

Mittal, A. and Subraveti, S. (2017). Comparison of Recommendation Models On the Amazon Automotive Dataset. https://github.com/abhaymittal/Recommendations-on-Amazon-Automotive-Dataset.

Parra, D. and Amatriain, X. (2011). Walk the Talk: Analyzing the relation between implicit and explicit feedback for preference elicitation. *Proc. of the 19th international conference on User Modeling, Adaption, and Personalization*, pages 255–268.

Pathak, A., Gupta, K., and McAuley, J. (2017). Generating and personalizing bundle recommendations on steam. *SIGIR 2017 - Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1073–1076.

Pérez-Marcos, J., Martín-Gómez, L., Jiménez-Bravo, D. M., López, V. F., and Moreno-García, M. N. (2020). Hybrid system for video game recommendation based on implicit ratings and social networks. *Journal of Ambient Intelligence and Humanized Computing*.

Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2011). *Recommender Systems Handbook*. Springer.

Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. (1998). Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proc. of the 1998 ACM Conf. on Computer Supported Cooperative Work*, page 345–354. ACM.

Schafer, B. J., Frankowski, D., Herlocker, J., and Sen, S. (2006). Collaborative filtering recommender systems. *Research Journal of Applied Sciences, Engineering and Technology*, 5(16):4168–4182.

Shah, K., Salunke, A., Dongare, S., and Antala, K. (2017). Recommender systems: An overview of different approaches to recommendations. In *Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017*, pages 1–4.

Sifa, R., Drachen, A., and Bauckhage, C. (2015). Large-scale cross-game player behavior analysis on steam. *Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015*, 2015-Novem:198–204.

Wang, D., Moh, M., and Moh, T. S. (2020). Using deep learning and steam user data for better video game recommendations. *ACMSE 2020 - Proceedings of the 2020 ACM Southeast Conference*, pages 154–159.

Yi, X., Hong, L., Zhong, E., Liu, N. N., and Rajan, S. (2014). Beyond clicks: Dwell time for personalization. *RecSys 2014 - Proceedings of the 8th ACM Conference on Recommender Systems*, pages 113–120.