UNIVERSITY OF SKÖVDE

RECOMMENDING STEAM GAMES TO USERS

Matching appropriate games based on time requirements and content

Master Degree Project in Informatics with a Specialization in Data Science
One year level, 15 ECTS
Spring term 2019

Niclas Elmäng

Supervisor: Jiong Sun
Examiner: Jiong Sun

Abstract

This project has studied if a content-based recommender system that matches games with an appropriate time requirement to users can be created. A large dataset was retrieved, cleaned and aggregated on using a few applications and libraries. The resulting dataset was used to calculate a weighted cosine similarity on a target user's preferences of certain attributes compared to those attributes of all other games to finally get a list of recommended games. Evaluation was done using an offline method and the subjective analysis of the recommendations show promising results. With further work, more attributes and an actual user study or online evaluation this kind of recommender system could be very beneficial to users with very strict time schedules dedicated to gaming.

Keywords: Content-based, collaborative filtering, recommender system, steam games

Table of content

1 Introduction

Steam today has a discovery queue that recommends game titles to users. This queue is made up of new releases, popular titles and games recommended to a user based on a lot of factors. Some users can spend a lot of hours every week to play certain games while other users maybe only have time to spend a couple hours, or even less, per week. One area that the current recommender system might fail at is finding the most relevant game based on how much time needs to be spent on it. Not a lot of research has been done on this type of recommender system. Users with a limited time for playing games might appreciate being able to complete games in a reasonable amount of time, instead of spreading one game session over weeks if not months. The same can be done for those with much more time available. With the huge number of indie games available there might to be some game/hidden gem that satisfy the genre and range of time necessary to suit the unique experience per each user.

2 Background

2.1 Recommender systems

Recommender systems have gained popularity both in online applications such as e-commerce, or video-on-demand (streaming) services, and research applications. They are used to, like the name suggests, recommend products to users based on some set of metrics, (Shani & Gunawardana, 2008). An e-commerce store might use such a system to show products that are frequently bought together, while a video streaming service might use it to pair videos to users based on how similar the user's interests are.

The recommendations are often based on user feedback by letting them rate products based on how much they like or disliked them. Other more implicit forms of feedback consist of collecting the behavior of a user. This is common in online e-commerce shops where the act of buying or even just viewing an item shows the user's propensity in taste. (Aggarwal, 2016, p. 14-15)

2.1.1 Collaborative filtering

Collaborative filtering is the process of predicting ratings, e.g. for movies, based on many other user's ratings on that same movie. The idea behind it being collaborative is because missing ratings can be inferred since many of the observed ratings often are highly correlated between users. For example, two users, A and B, who have rated the same item similarly are very likely to have similar taste. If user A have rated an item that user B has not, then user B's missing rating can be inferred to be very similar to user A's. (Aggarwal, 2016, pp. 8-9)

Collaborative filtering works by looking for high correlation between items or between users, also called item-based or user-based collaborative filtering. Sometimes both variants are used together. Aggarwal also mention that a problem with implementing a collaborative filtering method is that many times, the ratings matrix is sparse. Looking back at the movie example mentioned before, many users will only have seen a very small part of all movies in existence.

Aggarwal (2016, p. 9) mentions the advantage with collaborative filtering being that it is easy to implement, and the recommendations are often explainable. He also talks about the disadvantage with this method being caused by the ratings matrix often being very sparse. The sparsity might lead to less robust recommendations because the number of correlated ratings can be insufficient.

2.1.2 Content-based filtering

Content based filtering works by using the attributes of items to make recommendations (Aggarwal, 2016, pp. 14-15). For example, keywords in a movie's actors, director, title and genre are such attributes. This kind of filtering can be used when the ratings from different users are unavailable, but it is known that the user A has liked movie B. By using the attributes of movie B other movies with similar attributes could be found and recommended to user A, without the need of additional users.

Aggarwal (2016, p. 15) talks about the advantages and disadvantages with using a content-based filtering method. An advantage is that it is better at making recommendations for new items where a lot of rating data is missing, because some of those items might have similar attributes to items the target user already has rated. Some disadvantages are that the recommendations can become obvious since the attributes between items are compared, i.e. items with similar content have a higher chance of being recommended. This also means that items with a unique set of attributes that the user never has interacted with before will never be recommended, which can reduce the diversity of the recommendations.

2.1.3 One hot encoding

In natural language processing, one hot encoding is the process of turning categorical words into vectors of binarized data. The reason why one hot encoding is sometimes required is because if some categories were defined using a set of numbers in increasing order, then the categories with a higher value could be treated as a better category. With binarized data, the categories become are signified by either a 1 or a 0. This is needed to be able to do calculations on the categories, such as similarity calculations or for use in machine learning.

2.2 Evaluating recommender systems

2.2.1 User studies

User studies, (Aggarwal, 2016, p. 227) is an evaluation method that has the advantage of being able to answer many different questions. This relies on gathering users and letting them test the recommender system. This allows the behavior when using the system to be tested. The main disadvantage with user studies is the fact that it is expensive to arrange, both in terms of time and cost, if the testers are paid, and it also has the risk of introducing bias. The test subjects know that they are being specifically tested which might give biased answers.

2.2.2 Online evaluation

The online method is much like conducting a user study but instead of actively recruiting test subjects, random real users using some established service are tested. Compared to normal user studies, this method is not as sensitive to bias because the users are already established into the service. The problem that arises with this method is the fact that a sufficiently large userbase is needed to even consider testing. Thus, it is hard to do testing on a newly released service with only a small userbase. Often, the testing of using such data is limited to the people behind the commercial entity of the service, which make it very specific to the service in question (Aggarwal, 2016, pp. 227-228).

2.2.3 Offline evaluation

Offline experiments, as mentioned by Shani & Gunawardana (2008) is the process of using pre-collected data and simulate the behavior of the users. This kind of experiment is by far the most common type because it has the advantage that no user-interaction is needed. This method relies on having access to historical user data. Benchmarking using this method is much more robust because the data and benchmarking methodology can be standardized for further comparisons between other algorithms. Aggarwal (2016, p. 229) mention that a big disadvantage with the offline method is that it is only historical data. The tendencies of real users are thus harder to evaluate, among other aspects like *serendipity*.

2.2.4 Metrics

The metric *precision* measures how many out of all predicted instances were correct, while *recall* measures how many of the relevant instances were predicted out of all available instances. In the context of a recommender system the *precision* would be the fraction of how many out of the recommended items were good, while the *recall* would be how many of the good items got recommended out of all possible good items.

$$precision = \frac{\{good\ items\} \cap \{recommended\ items\}}{\{recommended\ items\}}$$

$$recall = \frac{\{good\ items\} \cap \{recommended\ items\}}{\{good\ items\}}$$

There is one big problem with using the metrics *precision* and *recall* when evaluating a recommender system. “In truth, you can’t really determine precision and recall in the normal sense — virtually no real recommender application has complete ground truth against which to measure.” Joseph Konstan, 2016. User interest is prone to change over time and such a change is much more difficult to measure using an offline method. Even an online method with a real userbase can have a hard time giving accurate *precision* and *recall*.

Serendipity, literally translated to “lucky discovery”, is a metric to measure if any recommendations are surprisingly good. It basically means that an unexpected outcome turned out to be positive, which, in the context of recommender systems, translates to how many of the least obvious items ended up being a good recommendation to the user. Measuring *serendipity* using an online method is done by letting users give feedback on how good a recommendation was but also how obvious. Measuring the same thing using an offline method requires more work by using another recommender system that generates the most obvious recommendations, which in turn are compared against the user’s implicit top items.

2.3 Related Work

Sifa, Bauckhage & Drachen (2014) implemented a game recommender system using archetypal analysis with the intent to make it easier for users to find relevant games fit to their specific interests. Archetypal analysis is an algorithm much like cluster analysis but with the difference being that the extreme points, the ‘archetypes’, are observed instead of the cluster centers. These archetypes are later used as convex combinations to approximate all other underlying types. (Cutler & Breiman, 1994).

> Archetypal Analysis [5] is a matrix decomposition technique based on decomposing the given matrix into a collection of extreme entities, called archetypes, and stochastic coefficient vectors to represent each data point as a convex combination of the found archetypes.
>
> Sifa, Bauckhage & Drachen, 2014

Sifa et.al. base their implementation on the fact that there already exists a huge number of games and it is increasing quickly. Their recommender model uses implicit feedback by hiding some of the games with the purpose of recommending them back to the user. Their model looks at how much time the user spent on a game and the preferences the user has towards that kind of game based on the genre. Using precision and recall for evaluating their performance they managed to get successful results when using the archetypal analysis model.

O’Neill et.al. (2016) performed an extensive data gathering operation on the Steam platform, gathering data on all info on the games, what games all users have played and what the users friend network looked like. This study was more of as exploratory analysis on the Steam network; thus, it does not implement any kind of recommender system. They presented some interesting data like how diverse the games are in user friend groups or how many games are owned or played by users. They mention that the top 20% Steam users are responsible for the top 73% total market value of owned games, which is an interesting fact about the userbase. One of their contributions was to release their dataset, consisting of everything there is on the Steam network, for the public to use.

3 Problem definition

Today users have an increasing difficulty of finding games that may be of interest to them. With the sheer amount of games that gets published every month on distribution platforms such as iOS App Store (PocketGamer.biz, 2019) and Steam ("Number of games", 2018), the process of searching for, and finding, those games that are most fit to the gamer is bound to be difficult, if done by the user. Some users have different demands or constraints than others, e.g. they only have a set number of hours available to play games during a short span of time, or they only like a certain kind of genre.

Games vary a lot in how much time usually is spent on them, as can be seen when using SteamSpy's service ("Top By Playtime"). Being able to complete a game in a reasonable amount of time, with respect to how much time the user usually spends in two weeks, can be something many users would appreciate. A recommender system that helps users find games with similar genres and a more fitting required playtime would be of interest. Thus, the research question is "Can a content-based recommender system be made that makes relevant recommendations to users based on the user's preferences in terms of game genre and time required to complete the game?"

A problem that the Sifa et.al. (2014) does not consider is the users playtime behavior. They only consider how much time a user has spent on a specific game/games and then find recommendations based on that. They have no consideration of how much time a user is willing to spend on games or how long a game usually is spent playing.

4 Method

4.1 Algorithm

A recommender system using content-based filtering with extra weight added to the user's personal time profile and preferred genres and, to an extent, the release year of games has been implemented. The reason content-based filtering is used is because the recommendations are supposed to find interesting and fitting games for the specific user based on their own playtime behavior and genre preferences. This information is retrieved implicitly from the dataset as the ground truth.

4.2 Evaluating performance

The performance of the recommender system should be evaluated by using a user study, but the time frame of this project would not allow that. A *precision*/*recall* measure is unable to be evaluated in this type of recommender system because there is no good ground truth to compare the recommendations against, which has been discussed in section 2.2.4. This project has been evaluated by using pre-existing knowledge about the target game and the game recommendations and subjectively say if the recommendations make sense or not.

5 Implementation

5.1 Dataset

The 170 GB large dataset from O'Neill et.al. (2016) consists of 11 different tables in SQL-format: *Achievement_Percentages, App_ID_Info, Friends, Games_1, Games_2, Games_Daily, Games_Developers, Games_Genres, Games_Publishers, Groups, Player_Summaries*. The tables that were used are: *App_ID_info, Games_1, Games_Genres* and the contents are described below.

App_ID_Info

- *Appid* - ID of the app.
- *Title* - Title of the app, as users see it.
- *Type* - The type of "app", consists of values such as "demo", "dlc", "game".
- *Price* - Current price on Steam shown in US dollars. Free items have a price of 0.
- *Release_Date* - The date the app was first available on Steam.
- *Rating* - Rating on Metacritic.
- *Required_Age* - Assigned minimum age requirement for viewing and buying the app.
- *Is_Multiplayer* - A 0 or 1 depending on if multiplayer content is included.

Games_1

- *Steamid* - ID of the user.
- *Appid* - ID of the app.
- *Playtime_2weeks* - Total time the user has run the app in the two-week period from the time of retrieval. Values in minutes.
- *Playtime_forever* - Total time the user has run the app since installing it. Values in minutes.
- *Dateretrieved* - Timestamp of data retrieval.

Games_Genres

- *Appid* - ID of the app.
- *Genre* - Genre of the app. Apps can have multiple genres in which case there are multiple distinct rows with the same appid.

The two tables containing info about all apps and the genres of games are relatively small, i.e. there are far fewer number of apps than users (18 000 apps vs 100 million users at time of retrieval), and because there is only one instance of information per app, except for when there are several genres. An example of what a data entry in *App_ID_Info* and *Games_Genres* look like are shown in Table 1 and Table 2 below.

Table 1 Example of a data entry in *App_ID_info* showing info about a single app.

appid	Title	Type	Price	Release_Date	Rating	Required_Age	Is_Multiplayer
570	Dota 2	game	0	2013-07-09 00:00:00	90	0	1

Table 2 Example of some data entries in *Games_Genres* showing an app with multiple genres.

appid	Genre
570	Action
570	Free to Play
570	Strategy

The third table, *Games_1,* is substantially larger in size, about 20 GB in fact. This is because it contains information about every user's playtime for every app they have ever played, example shown in Table 3 below. O'Neill (2016) state that there are upwards of 109 million users and 716 million games, which is evident by the huge amount of data they have collected.

Table 3 Example of a data entry in *Games_1* showing one user and one of their apps.

steamid	appid	playtime_2weeks	playtime_forever	Dateretrieved
76561198015114140	570	111	6827	2013-06-25 08:19:53

Another dataset that was retrieved was from Steamspy's service. Steamspy does some data aggregation and analysis on its own, but it also provides an API to retrieve certain types of data. In this case information about all games available on Steam was retrieved and the contents are described below.

Steamspy

- *Appid* - ID of the app.
- *average_2weeks* - Average time all users spent during the last two weeks.
- *average_forever* - Average time all users spent in total since forever.
- *Developer* - Developer of the game.
- *median_2weeks* - Median time all users spent during the last two weeks.
- *median_forever* - Average time all users spent in total since forever.
- *Name* - Name of the app.
- *Negative* - Number of negative reviews users have given the app.
- *Positive* - Number of positive reviews users have given the app.
- *Publisher* - The publisher of the game.

5.2 Preprocessing

The preprocessing step consisted of extracting the tables mentioned in section 5.1. This part of the process started with using command line tools in Linux to extract the relevant lines out from the large .SQL-file. The now separate tables were then imported to a database using MariaDB to immediately be exported into CSV-files. The next step for these CSV-files were the feature selection part.

5.3 Feature selection

With the relevant tables extracted the first step in the feature selection part was to aggregate the features needed for the recommender system. Using Python and the parallel computing library Dask (2016) the next step was to remove all instances where the weekly playtime was NULL in the table *Games_1*. This is because the recommender system is meant to base its recommendations on the user's weekly playtime, which can only be done if the data exist. After that, the user's playtime behavior was calculated as the playtime in total over two weeks.

After retrieving the total weekly playtime per user, all other entries of that user were dropped while keeping the instance with the highest weekly playtime. This gives the most played game during that period. An example of what the user data looked like after aggregating the weekly playtime and dropping all other entries is shown below in Table 4. At this point this table was still very large, which was mitigated by sampling a small fraction of all rows. A fraction of 4.75% of the original data was sampled, which still gave about 1.2 million rows of users.

Table 4 Example of what a data entry from *Games_1* looks like after aggregating the weekly playtime.

steamid	appid	playtime_2weeks	playtime_forever	user_2weeks_playtime
76561198086266678	570	1931.0	15431.0	6092.0

The *Games_Genres* table, as shown in Table 2 earlier, needed to be binarized. This was done by creating dummy variables, also called one hot encoding, of all genres. Instead of letting an app have multiple entries with different genres attached, the final table contains all unique genres as their own column, containing a 1 or a 0, with a 1 indicating the genre is relevant and a 0 that it is not. This implementation can represent all unique combinations of genres numerically instead of with text. An example of what the binarized genres look like can be seen in Table 5 below.

Table 5 Example of binarized genres. Only a few out of all 22 genres are shown here for easier presentation.

appid	Genre_Action	Genre_Adventure	Genre_Casual	…	Genre_Free to Play	Genre_Sports	Genre_Strategy
570	1	0	0	…	1	0	1

The binarized *Games_Genres* were joined with the *App_ID_info* table, as well as the dataset retrieved from Steamspy, which created one big table consisting of all app information. All the tables were joined together on the *appid* column, creating a table with 34 unique columns in total. The contents of each table are described in section 5.1.

The last thing to be dropped from the final table was entries with non-existent Rating, which originates from the *App_ID_Info*-table. Only those apps that have a rating given are kept with the purpose of sorting the recommendations in top rated order, if so desired. This whole process has achieved two datasets consisting of information about all apps and information about a fraction of users and their playtime.

5.4 Similarity function

The process of finding similar games to the user's preferences with the constraints of the user's playtime dedication was done by calculating a weighted cosine similarity. The weights are generated as such that the genres that are present in the user's preferences are weighted higher than all other genres, 1.5 vs 1.0 respectively. The user's playtime was weighted neutral at 1.0, while the release year was weighted the lowest with a weight of 0.5

The similarity function was implemented using the cosine distance/similarity between two vectors

$$similarity = \frac{u \cdot v}{\|u\| \|v\|}$$

as well as the weighted cosine distance/similarity.

$$weighted\ similarity = \frac{\sum_{i=1}^{n} w_i\, u_i\, v_i}{\sqrt{\left(\sum_{i=1}^{n} w_i\ {u_i}^2\right)}\ \sqrt{\left(\sum_{i=1}^{n} w_i\ {v_i}^2\right)}}$$

where *u* and *v* are vectors containing the user's preferences of the attributes and the games attributes respectively, and *w* is the vector containing weights for each component in those vectors.

5.5 Recommender system

Everything done in Python was done using Jupyter Notebook (Kluyver, 2016) for a more exploratory approach to code execution. Jupyter Notebook allows code to be executed independently by dividing code into cells. As mentioned in section 5.3, the library Dask was initially used because of their ability to do parallel computing on dataframes. The dataset was initially very large which made this library a necessity to progress in the project.

When the datasets had been reduced in size and initially cleaned, the python library Pandas (McKinney, 2010) was used to retrieve the features of interest, i.e. time, genres and release year, from the datasets. Pandas was also used in conjunction with SciPy (Jones et.al., 2001-) to compute the cosine similarity between the target user and all other games, which finally led to producing recommendations. A figure of the execution flow of this project, from start to finish is shown in Figure 1 below. Every new recommendation requires acquiring a new random user, generate similarities and retrieve the top most similar games as recommendations.

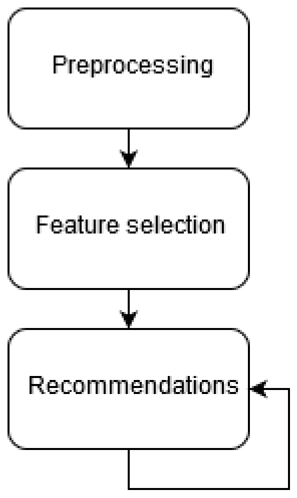
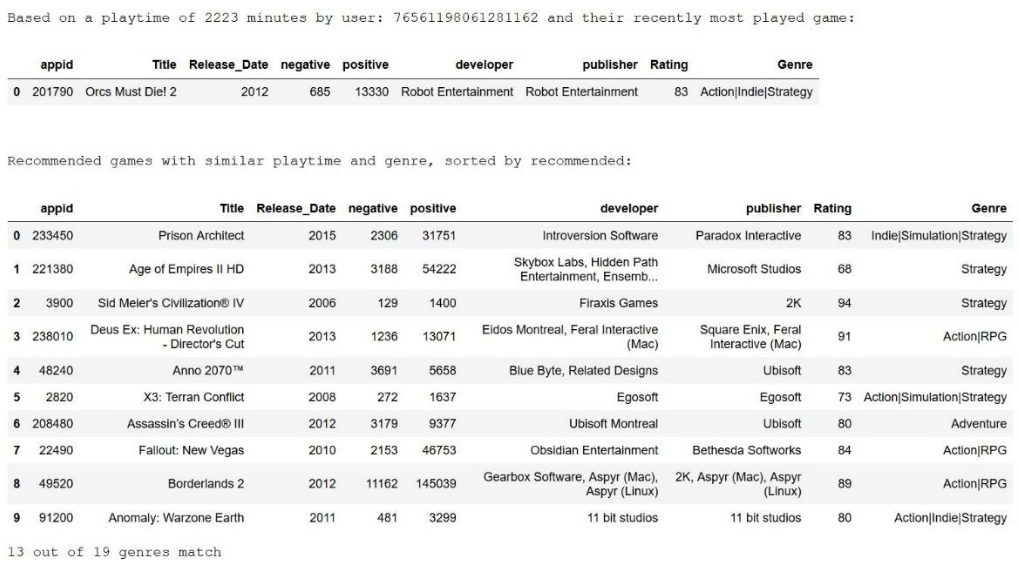


Figure 1 Project flow as discussed in section 5.2-5.4 to end up with recommendations based on the initial dataset.

6 Results

As mentioned in section 4.2, the evaluation could not be conducted by calculating *precision* and *recall*. This was because of the nature of the types of games the recommender system is supposed to recommend. Since it is supposed to give games with a similar total playtime, genre and, to an extent, the same release year, it will not give back exactly the games the user already has played. This is because the user must have spent the average amount of time anyone spends on the game in forever, just during the two weeks that the data consists of. The user needs to be an average person since forever in just a mere two weeks. This fact rules out the testing method that removes a game from the user's list of preferred games in an attempt to recommend it back to the user, like Sifa et.al (2014) did in their study.

Subjective analysis of the recommendations and what they are based on show relatively satisfying results, see Figure 2 below. These recommendations are much more diverse, which could increase *serendipity* based on how the user would choose to react. Since the performance can only be measured by having pre-existing knowledge about the recommended games, this type of recommender system should be evaluated by conducting a user study to get the user's opinions. While conducting the user study the actual preference of the users can be established, such as how much time is spent each week or what genres are the most interesting.



```
Based on a playtime of 2223 minutes by user: 76561198061281162 and their recently most played game:
```

	appid	Title	Release_Date	negative	positive	developer	publisher	Rating	Genre
0	201790	Orcs Must Die! 2	2012	685	13330	Robot Entertainment	Robot Entertainment	83	Action|Indie|Strategy

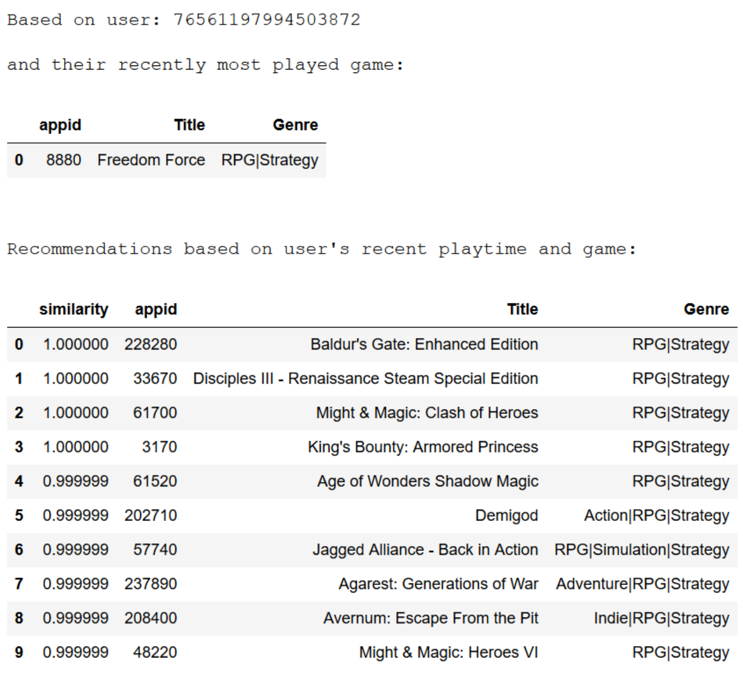
```
Recommended games with similar playtime and genre, sorted by recommended:
```

	appid	Title	Release_Date	negative	positive	developer	publisher	Rating	Genre
0	233450	Prison Architect	2015	2306	31751	Introversion Software	Paradox Interactive	83	Indie|Simulation|Strategy
1	221380	Age of Empires II HD	2013	3188	54222	Skybox Labs, Hidden Path Entertainment, Ensemb...	Microsoft Studios	68	Strategy
2	3900	Sid Meier's Civilization® IV	2006	129	1400	Firaxis Games	2K	94	Strategy
3	238010	Deus Ex: Human Revolution - Director's Cut	2013	1236	13071	Eidos Montreal, Feral Interactive (Mac)	Square Enix, Feral Interactive (Mac)	91	Action|RPG
4	48240	Anno 2070™	2011	3691	5658	Blue Byte, Related Designs	Ubisoft	83	Strategy
5	2820	X3: Terran Conflict	2008	272	1637	Egosoft	Egosoft	73	Action|Simulation|Strategy
6	208480	Assassin's Creed® III	2012	3179	9377	Ubisoft Montreal	Ubisoft	80	Adventure
7	22490	Fallout: New Vegas	2010	2153	46753	Obsidian Entertainment	Bethesda Softworks	84	Action|RPG
8	49520	Borderlands 2	2012	11162	145039	Gearbox Software, Aspyr (Mac), Aspyr (Linux)	2K, Aspyr (Mac), Aspyr (Linux)	89	Action|RPG
9	91200	Anomaly: Warzone Earth	2011	481	3299	11 bit studios	11 bit studios	80	Action|Indie|Strategy

```
13 out of 19 genres match
```

Figure 2 Top 10 recommended games using the implemented recommender system with weighted playtime, release date and genre.

The first attempt of similarity calculation was done using only the playtime and the genre as features. This very limited 2-dimensional feature set led to a lot of similarities of 1.0, as seen in Figure 3 below. This is because there is a sufficient amount of games with varying average playtime such that any given user's weekly playtime is highly likely to be exactly like at least some game's playtime, while also matching on the genres.



```
Based on user: 76561197994503872

and their recently most played game:
```

	appid	Title	Genre
0	8880	Freedom Force	RPG|Strategy

```
Recommendations based on user's recent playtime and game:
```

	similarity	appid	Title	Genre
0	1.000000	228280	Baldur's Gate: Enhanced Edition	RPG|Strategy
1	1.000000	33670	Disciples III - Renaissance Steam Special Edition	RPG|Strategy
2	1.000000	61700	Might & Magic: Clash of Heroes	RPG|Strategy
3	1.000000	3170	King's Bounty: Armored Princess	RPG|Strategy
4	0.999999	61520	Age of Wonders Shadow Magic	RPG|Strategy
5	0.999999	202710	Demigod	Action|RPG|Strategy
6	0.999999	57740	Jagged Alliance - Back in Action	RPG|Simulation|Strategy
7	0.999999	237890	Agarest: Generations of War	Adventure|RPG|Strategy
8	0.999999	208400	Avernum: Escape From the Pit	Indie|RPG|Strategy
9	0.999999	48220	Might & Magic: Heroes VI	RPG|Strategy

Figure 3 Top 10 recommended games using an early version of the implemented recommender system where only unweighted playtime and genre is considered. Many games can be seen to have complete similarity.

7 Conclusion and discussion

The problem of recommending games, other than just taking the most popular, highest rated one right now, is an interesting topic. Variations of it already exists, Sifa et.al. (2014). Accommodating a certain type of constraint in the users is something relatively unheard of, at least in the case of giving them an appropriate gaming experience in terms of time to complete.

The results answer the research question, albeit subjectively, as it is indeed possible to create a content-based recommender system that can give sensible recommendations of games while accommodating the time dedicated by the user to gaming. Future work is definitely needed to more robustly, and objectively, evaluate if such a recommender system can be created on a large scale.

The problem with the current implementation is that some games, even though they have the same genre and similar playtime, have completely different target audiences. Someone playing a game with a recommended age of 18+ could suddenly be recommended a game for ages 7+. This phenomenon has the benefit that it could lead to increased *serendipity* and it should be adjustable to fit the user's current preferences.

8 Future Work

A clear improvement needed is an actual user study, which was not possible in time frame of this project. This would give quantifiable results that could be used to improve this model, as well as being compared to the results of other studies. Depending on how the user study is conducted even more questions about user behavior could be answered and used for improvement.

The model could be improved by bringing the developer and publisher into the calculation. This would be good if the user likes the types of games a certain developer makes, but the weights in the similarity function might need adjusting. As mentioned in section 7, the age restriction for a game might be beneficial to implement. Finding games that fit the time aspect, genre and the age restriction might lead to a big increase in recommendation quality. Furthermore, including user-generated tags about the actual content of a game could also be very beneficial to find games that have similar gameplay, not just genres, The genres might be deceiving as to what the gameplay is like, as two games can have completely different target audiences, even though they have the same genres. Using the content tags that Steam lets users vote on could make the recommendations even better.

Instead of using a discrete playtime value, the time could made into an interval instead. This would increase the diversity of the recommendations while still staying true to the time behavior and genre preference of the user. Using this interval, it would be beneficial to allow the users to choose their own preferred playtime and genre, maybe even select a game to base some other features on like developer, director, artists or title.

The recommender system could also be allowed to recommend games with genres the user rarely play to give them the opportunity to discover something completely new that they never would have considered before, as is the purpose of the recommender system. This might lead to another increase in *serendipity*.

References

Aggarwal, Charu C. (2016), *Recommender Systems: The Textbook*. Available online: http://pzs.dstu.dp.ua/DataMining/recom/bibl/1aggarwal_c_c_recommender_systems_the_textbook.pdf [2019-05-24]

Cutler, A., Breiman, L., (1994). *Archetypal Analysis*. Technometrics, 36(4) pp. 338-347. Available online: http://digitalassets.lib.berkeley.edu/sdtr/ucb/text/379.pdf [2019-04-09]

Dask Development Team (2016). Dask: Library for dynamic task scheduling. (Version 1.2.1) [software]. Available: https://dask.org

Jones, E., Oliphant, E., Peterson, P., et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [2019-05-30]

Kluyver, T., et.al. (2016). *Jupyter Notebooks—a publishing format for reproducible computational workflows*, pp. 87-90. doi: 10.3233/978-1-61499-649-1-87. [2019-05-30]

Konstan, J., (2016), Head of GroupLens Research Lab. https://www.quora.com/How-do-I-compute-Precision-and-Recall-in-Recommender-Systems [2019-05-24]

McKinney, W. (2010) *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference. pp. 51-56. Available online: http://conference.scipy.org/proceedings/scipy2010/mckinney.html [2019-05-30]

Number of games released on Steam 2018 | Statistic. (2019). Available online: https://www.statista.com/statistics/552623/number-games-released-steam/ [2019-04-09]

Number of Steam users 2018 | Statistic. (2018). Available online: https://www.statista.com/statistics/308330/number-stream-users/ [2019-04-09]

O'Neill, M., Vaziripour, E., Wu, J., Zappala, D. (2016). *Condensing Steam: Distilling the Diversity of Gamer Behavior*. Available online: https://steam.internet.byu.edu/oneill-condensing-steam.pdf [2019-04-20]

PocketGamer.biz: App Store Metrics (2019). https://www.pocketgamer.biz/metrics/app-store/ [2019-05-14]

Shani & Gunawardana (2008) *Evaluating Recommender Systems*. Fourth International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution. Available online: http://www.bgu.ac.il/~shanigu/Publications/EvaluationMetrics.17.pdf [2019-05-14]

Sifa, Bauckhage & Drachen (2014). *Archetypal Game Recommender System*. Available online: https://pdfs.semanticscholar.org/f057/a249c97f908d8e262cffab709c93be9d2314.pdf [2019-04-09]

Top By Playtime. (n.d.). SteamSpy - All the data about Steam games. Available online: https://steamspy.com/ [2019-04-09]