DESIGNING A HYBRID RECOMMENDER SYSTEM FOR STEAM GAMES

JUSTIN V. BOON
ANR: 611423
SNR: U1258630

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN COMMUNICATION AND INFORMATION SCIENCES,
MASTER TRACK DATA SCIENCE BUSINESS & GOVERNANCE,
AT THE SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
OF TILBURG UNIVERSITY

Thesis committee:

prof. dr. ir P.M.H. Spronck
dr. M.M. van Zaanen

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
April 2019

Abstract

As the number of available games on online distribution platforms increases every day, the complexity of matching the right game with the right user increases alongside with it. Several studies have been conducted into building a recommender system for the Steam platform, all of them incorporating collaborative filtering methods that predict playtime based on a user's similarity to other users. Given that research suggests that combining multiple different algorithmic approaches may lead to more accurate recommendations, this study aims to provide insight into what extent the recommender system for Steam can benefit from combining a content-based filtering method with the current collaborative filtering method. Data on playtime of users was harvested using Steam's API, whereas metadata on games was collected by web scraping tags from Steam's website. Playtime predictions were subsequently made and hybridized, making use of two different switching hybrids. Results show that a hybrid system is, in correspondency to theory, able to outperform both a stand-alone collaborative filtering system and a stand-alone content-based system.

Keywords hybrid recommender system · Steam · game analytics

Table of contents

M) Overview cosine matrices

N) Overview summary data

O) R code for predicting playtime

P) R code for evaluating playtime predictions

Q) R code for hybridizing playtime predictions

R) Overview playtime predictions

S) Overview prediction evaluations

T) Overview results T-test

1. Introduction

This chapter describes the context, the scientific relevance and the practical relevance of this research. Additionally, it posits the problem statement of this study and its corresponding research question. The chapter is concluded with a brief outline of the coming chapters.

1.1 Context

From its commercial debut in the early 1950s to today, gaming has evolved into one of the most lucrative entertainment industries on the planet. Currently, there are over 2 billion gamers in the world, most of which are PC gamers (Newzoo, 2018). Steam, the largest digital distribution platform for PC gaming, presently offers more than 30,000 unique titles to its users (PC Gamer, 2019a). Although having access to such a varied supply of products seems desirable, having to manually choose between a selection of tens on thousands of games can be somewhat of a discouraging process. This cognitive phenomenon, called choice overload, can lead to dissatisfaction with or avoidance of choices (Chernev, Böckenholt & Goodman, 2015). Relying on peers or suggestions from experts for decision support is not always the optimal solution to deal with an abundance of choice, as another closely related cognitive phenomenon, called information overload, limits our capacity to make a decision when having too much information on a particular issue (Melinat, Kreuzkam & Stamer, 2014).

Over the years, distribution platform owners have tried to tackle these issues by developing recommender systems, which are defined as: “[systems that] generate meaningful recommendations to a collection of users for items or products that might interest them” (Melville & Sindhwani, 2017). These recommender systems typically operate through either collaborative filtering methods or content-based filtering methods, which recommend products to users based on user behavior and the characteristics of their currently owned products, respectively (Çano & Morisio, 2017). Both methods, however, have disadvantages that can have negative impacts on the accuracy of their recommendations. Systems that make use of collaborative filtering can, for instance, experience trouble with recommending products to so-called “grey sheep”, as their unique user behavior impedes the recommendation of products based on their similarity to other users (Khusro, Ali & Ullah,

2016; Zheng, Agnani & Singh, 2017). Content-based systems, on the other hand, have trouble with the lack of distinctiveness that metadata on its own can offer (Khusro et al., 2016).

Research has demonstrated that hybrid recommender systems can provide more accurate recommendations than systems that only make use of a singular algorithmic approach (Çano & Morisio, 2017). Hybrid recommender systems combine the advantages of two (or more) algorithms by hybridizing the recommendation components. A hybrid recommender system for Steam can therefore consist of a collaborative filtering method that uses playtime data to calculate similarity between users, as well as a content-based method that assesses similarity between games by processing metadata. Using a hybrid method, the content-based component of the recommender system may thereby limit gray sheep from negatively influencing the accuracy of the model, accounting for their unique player profile by making the prediction based more on the characteristics of their game collection, rather than their similarity to other players. Meanwhile, the collaborative filtering component may make up for the lack of detail that metadata alone accounts for.

A game recommender system for Steam that makes use of collaborative filtering has already been developed by Sifa, Bauckhage & Drachen (2014a). In their study, playtime data from over 6 million users was harvested using Steam's API, which ultimately resulted in a system with a mean recall value of 90 to 94%. Notten (2017) later tried to enhance the performance of this recommender system, by adding data on achievement progress to a replication of the algorithm. This did not result in higher recall values. Research, however, suggests that compared to fine-tuning and polishing one particular approach (in this case collaborative filtering), combining multiple different approaches may lead to more accurate recommendations to Steam users. The aim of this study therefore is to examine whether this holds true for a recommender system for Steam games.

1.2 Research question

Given that 1) research has demonstrated the potential of hybrid recommender systems in outperforming singular algorithmic approaches, and 2) Steam Recommender Systems as developed in previous studies solely incorporate collaborative filtering methods (Notten, 2017; Sifa et al., 2014a), the following research question is formulated.

RQ: Can a non-hybrid recommender system for Steam be improved by combining a content-based filtering method with a collaborative filtering method?

1.3 Outline

In Chapter 2, earlier relevant work will be discussed that is related to the Steam platform, choice and information overload, recommender systems and the incorporated statistical methods. Chapter 3 covers the general approach of this study, the computational algorithms that were used, and contains a detailed description of the experimental procedure. In Chapter 4, the results are reported, which are evaluated and discussed in Chapter 5. Chapter 6 serves as an overview of the answers that are formulated to the research question, as well as a summary of how the results that are obtained from this study can be placed in the context of existing research.

2. Related work

In this chapter, earlier relevant work that is related to the Steam platform and its current recommender system will be discussed in Section 1. The cognitive phenomenon called choice overload, which is one of the main incentives for developing recommender systems, is covered in Section 2. Additionally, an explanation of how recommender systems work is given in Section 3, alongside an overview of the different methods that they generally incorporate. Earlier academic work related to recommender systems for Steam is presented in Section 4. In Section 5, archetypal analysis and cosine similarity are discussed, which serves as an overview of the statistical methods that were incorporated in this study. Section 6, the final section, summarizes this chapter and touches upon the new methods that will be used in this study.

2.1 Steam

The Steam platform is a digital distribution platform for purchasing and playing video games. It was first released in September 2003, by the American video game developer Valve Corporation. By the end of 2018, the platform had around 90 million monthly active users, having gained approximately 23 million monthly active users in little over a year (PC Gamer, 2019b). With its collection of over 30,000 unique game titles, Steam currently is the largest digital distribution platform for PC gaming, both in terms of game collection and market share (VentureBeat, 2017). Steam allows users to tag games, which provides the platform with a crowdsourced set of metadata for each game that can in turn be used to categorize all of the different titles. Additionally, the existence of these tags enables people to scrape this metadata from Steam's website, as has been done by several researchers (Windleharth, Jett, Schmalz & Lee, 2016).

Valve Corporation maintains an API for Steam, that can be used by researchers and web developers to query some of its databases for various purposes. Multiple studies have been conducted in which this API was used, for instance to gather data from gamers on playtime and reviews (Ahn, Kang & Park, 2017; Sifa, Bauckhage & Drachen, 2014b). However, as of April 11, 2018, Valve has made some changes in handling the privacy of Steam users, that are

most likely aimed at complying with the EU General Data Protection Regulation (Steam, 2018). Therefore, the visibility of a users' games collection is no longer set to public by default, which reduces the amount of data that can be generated from Steam's API. Currently, data on playtime can only be gathered from players who have explicitly chosen to share their data with the public.

In 2014, Steam released a recommender system as part of a revision of their storefront. The company's motivation behind the Discovery Queue, as their system was termed, was "[making] it easier to find exactly what you want when shopping for a new game" (Steam, 2014). The 3,700 different titles that were available then were already considered to potentially result in an abundance of choice; in the meantime, the number of available games has increased more than eightfold. More information on Steam's recommender system is detailed in Section 2.4.

2.2 Choice and information overload

Asking a player to choose between more than 30,000 games is a good example of choice overload, which is defined as: "a scenario in which the complexity of the decision problem faced by an individual exceeds the individual's cognitive resources" (Chernev, Böckenholt & Goodman, 2015). Research into this cognitive phenomenon was fueled by a study published by Iyengar & Lepper (2000), which reported that an assortment size of 6 different products had resulted in higher sales than an assortment size of 24 at a local food market. Later research has shown that there is no unambiguous threshold at which a certain assortment size leads to overchoice; four key factors that moderate the impact of assortment size on choice overload have been identified as: 1) choice set complexity, 2) decision task difficulty, 3) preference uncertainty and 4) decision goal (Chernev et al., 2015). While as of yet no research has specifically addressed choice overload in purchasing games, the phenomenon has been observed to occur while purchasing a variety of other products, such as electronic devices and chocolates (Berger, Draganska & Simonson, 2007; Chernev, 2003).

In the event of choice overload, people have historically relied on recommendations from peers or experts to guide them in coming to a decision. However, as assortment sizes

increase, so do the number of available reviews and recommendations on the internet and elsewhere. This relates to a phenomenon that is associated with choice overload, known as information overload. Information overload is defined as: “the feeling of too much information to be processed for the cognitive capacity of a person” (Melinat et al., 2014). In order to overcome this abundance of information, distribution platforms have taken interest in developing recommender systems that automatically present users with a set of items that are expected to be of interest to them.

2.3 Recommender systems

Recommender systems are defined as: “[systems that] generate meaningful recommendations to a collection of users for items or products that might interest them” (Melville & Sindhwani, 2017). A recommender system for books was first described by Karlgren (1990), and was later followed by several computerized prototypes during the mid-nineties (Hill, Stead, Rosenstein & Furnas, 1995; Resnick et al., 1994). In recent years, online platforms have demonstrated their acknowledgement of a well-functioning recommender system as a tool for business success. In 2009, media-services provider Netflix set up a competition in which the prizewinners were offered $1,000,000 for bettering the company’s recommender system by 10 percent (Wired, 2009). The final solution of the winning team blended more than a hundred different algorithmic approaches into a single prediction (Bell, Koren & Volinsky, 2009). The majority of recommender systems, however, make use of either a collaborative filtering, content-based filtering or a more toned-down hybrid approach. These different methods will be further described in the current section, along with their advantages and disadvantages.

Collaborative filtering methods

A recommender system that makes use of a collaborative filtering method recommends items to users based on their similarity to other users, which is assessed by a comparison between their recorded user feedback. This feedback can, for instance, consist of explicit feedback, like star ratings on movies, or implicit feedback, like playtime on games. The reasoning behind collaborative filtering methods is that users who demonstrated similar user behavior in the past will most likely bear similar interests (Schafer, Konstan & Riedl, 1999).

Recommender systems that incorporate a collaborative filtering method usually are model-based or memory-based. Whereas memory-based systems directly compare users by calculating the similarity between them, model-based systems use learning techniques such as neural networks in order to make predictions (Burke, 2002). An advantage of collaborative filtering is its indepence of any machine-readable representation of the items that it recommends (Burke, 2002). A collaborative filtering method therefore works well for various types of items, such as movies, music and games. Figure 1 serves as a schematic overview of how the method works.

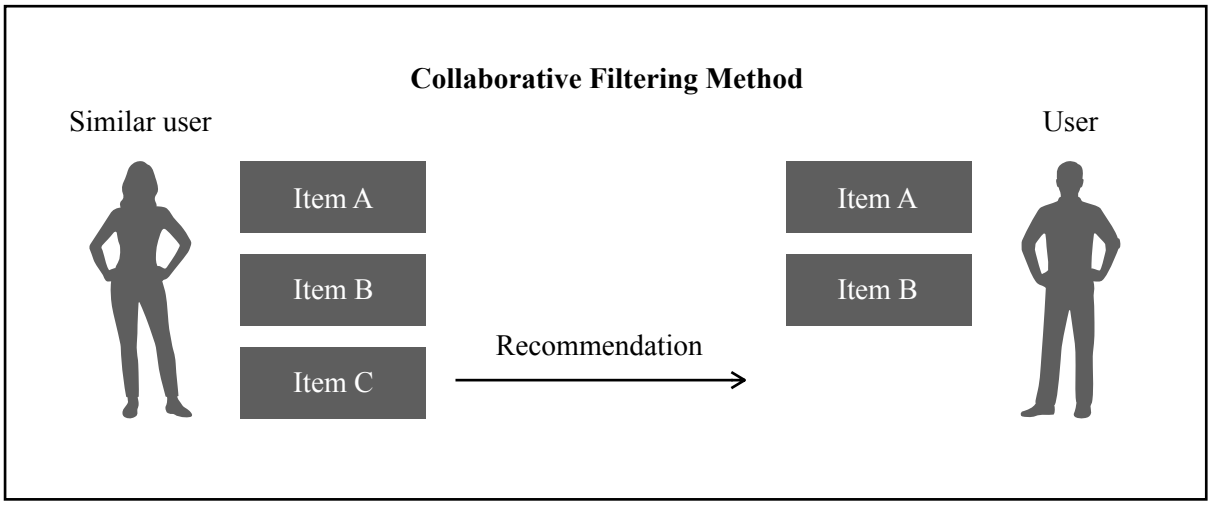


Figure 1. Schematic overview of a collaborative filtering method.

One of three main problems that exist with collaborative filtering methods is its performance in the event of so-called “grey sheep”, which are users who demonstrate rather unique user feedback (Claypool, et al., 1999). Data sparsity, whereby few users like the same items, serves as an additional challenge for collaborative filtering methods (Su & Khoshgoftaar, 2009). Another issue is the trouble that this method has with recommending in the event of a cold start, in which recommendations have to be made for new items or new users (Lika, Kolomvatsos & Hadjiefthymiades, 2014).

Content-based filtering methods

A content-based recommender system makes recommendations to a user based on its knowledge about items, and how these compare to other items that were previously liked or bought by that user (Pazzani & Billsus, 2007). The reasoning behind content-based filtering

methods is that users who have enjoyed an item with certain characteristics in the past will most likely enjoy similar items in the future (Lops, de Gemmis & Semeraro, 2010). Whereas a collaborative filtering method has the ability to 'think outside the box', a content-based method only processes data 'inside the box', as it bases its predictions solely on previously liked items (Burke, 2002). Various learning techniques, such as neural nets, decision trees and vector-based representations have been used with this method to derive user profiles in order to come up with recommendations (Burke, 2002). Figure 2 serves as a schematic overview of a content-based filtering method.

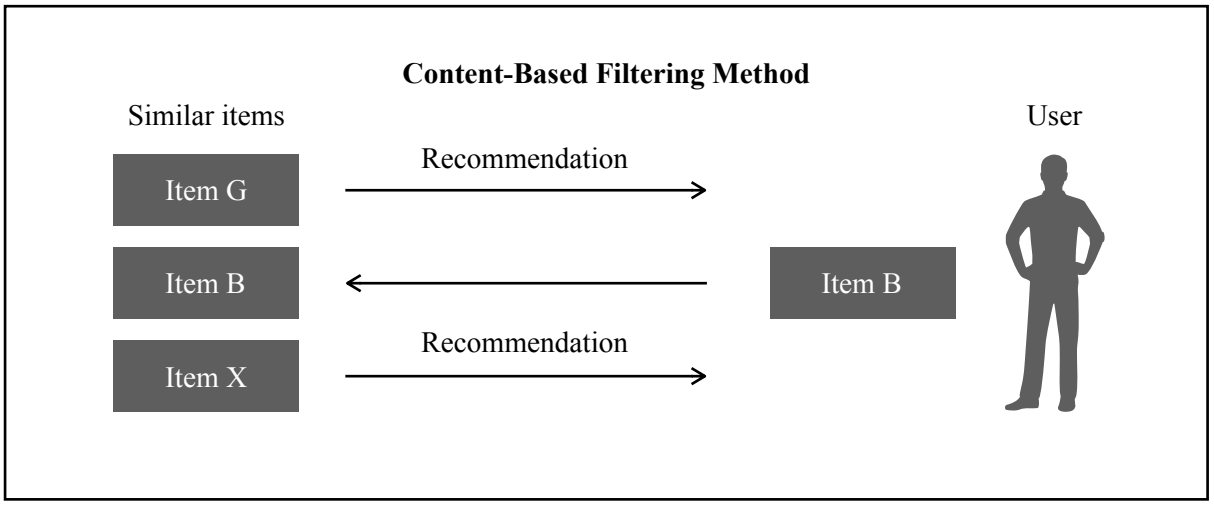


Figure 2. Schematic overview of a content-based filtering method.

As content-based filtering methods only use knowledge about items, they are dependent on the quality and quantity of metadata associated with the items it is trying to recommend. Just like collaborative filtering methods, content-based filtering experiences trouble with a cold start, as little to no data on item preference is available for new users (Lika et al., 2014).

Hybrid recommender systems

Hybrid recommender systems are defined as: "systems [that] combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one" (Burke, 2002). By combining multiple algorithmic approaches, these recommender systems are aimed to overcome commonly encountered problems in recommendation, such as performance issues with "grey sheep" and data sparsity. The most prevalent hybrid recommender systems are combinations of a collaborative filtering method

and a content-based filtering method, in which the different recommendation components are hybridized into a definitive recommendation (Çano & Morisio, 2017).

One of the generally encountered methods for hybridizing the recommendation components is a so-called switching hybrid (Çano & Morisio, 2017). In a switching hybrid, the recommender system switches between different recommendation strategies based on some criteria. These criteria are dependent on the specific recommendation task at hand. Other commonly used hybridization techniques include weighted hybrids, mixed hybrids and feature combination. A basic example of a weighted hybrid is a gradually established linear combination of recommendation scores (Claypool et. al, 1999). In mixed hybrids, the recommendations from the different techniques are presented together. Feature combinations often use information derived from a collaborative filtering method as additional feature data. The appended dataframe is then subjected to a content-based filtering method in order to obtain the final recommendations.

2.4 Steam recommender systems

According to Steam, its current recommender system combines three different approaches in order to come up with recommendations to users (Steam, 2014). The Discovery Queue, which is updated on a daily basis, bases its recommendations on 1) the similarity between a given game and the games owned by a user, 2) the general popularity of a given game and 3) the novelty of a given game. This system can therefore be classified as a hybrid recommender system, that incorporates a content-based method in order to assess similarity between games. The accuracy of Steam's system, or how the different recommendations components are calculated and hybridized, is currently unknown to the public.

In 2014, Sifa, et al. (2014a) developed a Steam recommender system that uses a collaborative filtering method in order to predict playtime for users on certain games. Data on 100,000 players was used, that was harvested by querying the platform's API client. Their algorithm first executed archetypal analysis on a reshaped version of the data as a way of extracting features. Thereafter, the cosine similarity between each player was calculated, which was later used to return a top 5 of most similar players for each user. In order to predict the

playtime for a user on a certain game, the sum of the cosine similarities to all 5 most similar players was multiplied by the total amount of time that these users had played that particular game. The sum of these outcomes was thereafter divided by the sum of the same 5 cosine similarities, to return the predicted playtime. If a blinded (random) game from a user's top 3 games was returned in a top-L ranking of predicted playtimes for that user, a recall value of 1 was assigned to that iteration of the algorithm. Ultimately, the mean of all recall values generated by the algorithm was used to evaluate the overall performance of the recommender system. A schematic overview of the procedure of this algorithm can be found in Appendix A.

Notten (2017) later tried to enhance the performance of this algorithm, by adding data on achievement progress to a replication of the recommender system. This did not result in higher recall values: between 82.1 and 85.3 percent compared to 82.2 to 85.4 percent achieved by the replication that did not incorporate achievement data. The recommender systems that were developed by Sifa, et al. (2014a) and Notten (2017) both used playtime data that was extracted from Steam's API before Valve changed its privacy policy in April 2018. Both recommender systems incorporated a collaborative filtering method, that recommends products to users based on user behavior, which is hereby operationalized by playtime and playtime of similar users (and achievement progress, in the case of Notten). A schematic overview of the study that was executed by Notten can be found in Appendix B.

2.5 Statistical methods

Cosine similarity

Cosine similarity (or cosine distance) is a metric used to measure the cosine of the angle between two vectors projected in a multi-dimensional space. It is generally used when the magnitude of the vectors is of no importance (Emmery, 2017), as cosine similarity corrects for differences in total playtime when assessing similarity between users. Cosine similarity is often employed in high-dimensional positive spaces, such as is the case with text mining and information retrieval. As euclidian distance, another widely used metric, does not correct for the magnitude of the vectors, cosine similarity is more suitable to use in this study. After all, resemblance between players is more about their pattern of playing, rather than the absolute amount of time invested in all of the different games. Moreover, the data that is used in this

study can indeed be classified as high-dimensional, given the thousands of different columns (games) and rows (users) in the data.

Archetypal analysis

Archetypal analysis is an unsupervised clustering method that "represents each individual in a dataset as a mixture of individuals of pure type, or archetypes" (Cutler & Breiman, 1994). The method seeks extremal points, rather than cluster centers, such as is the case with typical cluster analysis. Archetypal analysis has been widely applied in game analytics research, for instance to cluster players and games based on gameplay-interest models (Drachen, Sifa, Bauckhage & Thurau, 2012; Sifa, Drachen, Bauckhage, Thurau & Canossa, 2013; Sifa et al., 2014a). An R package for archetypal analysis was first developed by Eugster and Leisch (2009). As calculating the cosine similarity between vectors containing playtime on thousands of different games would take too long given the time frame available for this study, these vectors are first subjected to archetypal analysis as a form of feature extraction.

2.6 Summary

Steam, the largest digital distributor for PC gaming, currently offers more than 30,000 unique games on its platform. This serves as a good example of choice overload: an abundance of choice that impedes ones abilities to come to a satisfactory decision. A well-functioning recommender system serves as a way to combat choice overload, as it presents users with recommendations that are predicted to be relevant. These systems generally incorporate a collaborative filtering method (CF), a content-based filtering method (CBF) or multiple different methods to come up with predictions. Systems that use a collaborative filtering method make recommendations based on the items that are liked by similar users. System that incorporate a content-based filtering method base its recommendations on previously liked items, and their similarity to the item it is trying to predict interest for. Hybrid systems combine two or more methods, in order to benefit from the advantages that hybridizing multiple different methods can have. The recommender system that Steam currently employs makes use of a content-based method, that attributes a larger weight to popular and novel games while making recommendations. Within the field of game analytics, studies have already been conducted into building a recommender system for Steam games, all of them

incorporating collaborative filtering methods in order to come up with predictions. As previous research has demonstrated that hybrid recommender systems are generally able to outperform systems that only incorporate one approach, the current study aims to hybridize multiple different algorithmic approaches into a single playtime prediction for Steam users. The method and experimental approach that was incorporated in this study will be discussed in Chapter 3.

3. Method

In this chapter, the experimental approach that was taken in this study will be discussed. In Section 3.1 of this chapter, it is detailed how data on games and playtime of users was harvested from Steam's API and website. Section 3.2 covers the way in which feature extraction was applied to the data. In Section 3.3, it is explained how the cosine similarity between both users and games was calculated to aid in predicting playtime for users on random games. The way in which these predictions were made, is covered in Section 3.4. In Section 3.5, the evaluation method of the predictions is discussed, after which the hybridization process of the predictions is described in Section 3.6. Section 3.7 elaborates on the way in which the hybridized recommendations were evaluated. Section 3.8, which serves as the final section of this chapter, lists the hardware and software that was used in this research. A schematic overview of the method that was incorporated in this study can be found in Appendix C.

3.1 Collecting and pre-processing data

Playtime data

Data on playtime of Steam users was collected by querying Steam's API in R (Appendix E), whereby a user is limited to a maximum of 100,000 requests per day. A list of 39,203 Steam user IDs was provided by the thesis supervisor, but the 'GetOwnedGames'-method of the API only returned playtime data on 190 unique players due to Steam's updated privacy policy. In order to gather data on more users, the friend lists of these 39,203 users were first requested from the API using the 'GetFriendList'-method (Appendix D). Thereafter, the user IDs that were collected by using this method were again used to request their friend lists from the API. This process resulted in a list of 1,827,048 unique Steam users. Their IDs were then used to request playtime data from the Steam API using the 'GetOwnedGames'-method, resulting in a data frame containing 2,790,464 rows, representing 19,730 different users and 23,630 unique game IDs (Appendix G). Querying playtime data on these 1,827,048 IDs took approximately 180 hours to complete.

After the playtime data was collected from the Steam API, only the four columns containing the different user IDs, the IDs of the games, the names of the games and the amounts of playtime were selected for further analysis. Incorrectly coded game IDs and game names were detected and corrected accordingly, as a few dozen games in the data appeared with different IDs and names for different users. Users with a game collection smaller than the size of 5 or a cumulative playtime shorter than 1500 minutes were omitted from the dataset, as their user feedback was considered too scarce to draw a reasonable comparison between users. Games with a cumulative playtime of less than 1500 minutes were also omitted from the dataset, as were games owned by fewer than 5 people, given their unpopularity in the sample. These thresholds were arbitrarily chosen. As games were included in the dataset that were no longer available on Steam, these titles were removed from the data. This was accomplished by filtering out the games that metadata could not be retrieved on later in the study. Games on which metadata could not be collected for other reasons were omitted from the data as well, as were games that had less than 5 tags attributed to them. These removals resulted in a data frame containing 1,079,875 rows, representing 12,976 different users and 7,105 unique game titles (Appendix G).

Metadata

Metadata was collected by web scraping the tags of each unique game ID that occurred in the partially pre-processed playtime dataset from Steam's website. This process took approximately 4 hours to complete, and resulted in a data frame containing 87,318 rows, representing 368 unique tags and 7,105 unique game titles (Appendix H). An overview of the R code that was used to web scrape the metadata can be found in Appendix F. Appendix I serves as an overview of all unique tags and their (relative) occurrence in the data.

3.2 Feature extraction

To speed up the process of calculating similarity between users and games, archetypal analysis was conducted on both the playtime data and the metadata. In order to make the playtime data suitable for archetypal analysis, the data frame was reshaped by turning each unique game title into a column and transforming each individual user into a row. Consequently, each cell represented the playtime of a user on a particular game (Appendix J).

In order to reshape the metadata for archetypal analysis, each unique tag was turned into a row and each individual game was transformed into a column, whereby each (binary) cell indicates whether a certain tag is applicable to a particular game (Appendix J).

The R package 'archetypes' was used to carry out this analysis, which provides an implementation of the algorithm in R (Eugster and Leisch, 2009). Archetypal analysis was executed on both the playtime data and the metadata for $k = 9$. Ideally, one would execute archetypal analysis for a range of k, and visually inspect a scree plot of the RSS values for each k in order to pick its optimal value, based on the elbow method (Eugster and Leisch, 2009). However, as this is a computationally expensive process and the time frame for this study is limited, an identical value for k was picked as was chosen (among others) by Sifa et. al (2014a) and Notten (2017), which allows a fair comparison. Both executions of the algorithm were capped at 25 iterations. The minimal value of improvement between two iterations was set at 0.001. Executing archetypal analysis on both datasets took approximately 15 hours in total to complete.

3.3 Calculating similarity

Playtime data

In order to assess similarity between players, the cosine similarity between the archetype vectors was calculated for all different users. This resulted in a matrix of 12,976 by 12,976, with each cell representing the cosine similarity between two users' vectors (Appendix M). An overview of the R code that was used for the calculations can be found in Appendix L. Using these cosine similarities, a data frame was constructed that, for each user, contained the five most similar players (in terms of cosine similarity) and the cosine similarities for these five players. This data frame (Appendix N) was later used in the process of predicting playtime for a random selection of games for a user (Section 4.5). Calculating similarity between players for the whole matrix took approximately 57 hours to complete.

Metadata

To determine similarity between games, the cosine similarity between the archetype vectors was calculated for all unique game titles as well, resulting in a matrix of 7,105 by 7,105. This

took approximately 9 hours to complete. In this data frame, each cell represents the cosine similarity between the vectors of two games (Appendix M). An overview of the R code that was used for this task can be found in Appendix L. Using these cosine similarities, a data frame was constructed that, for each game, contained the five most similar games (in terms of cosine similarity) and the cosine similarities for these five games. This data frame (Appendix N) was later used in order to predict playtime for a random selection of games for a user (Section 4.5). Figure 3 serves as a schematic overview of the process of calculating similarity between both users and games.

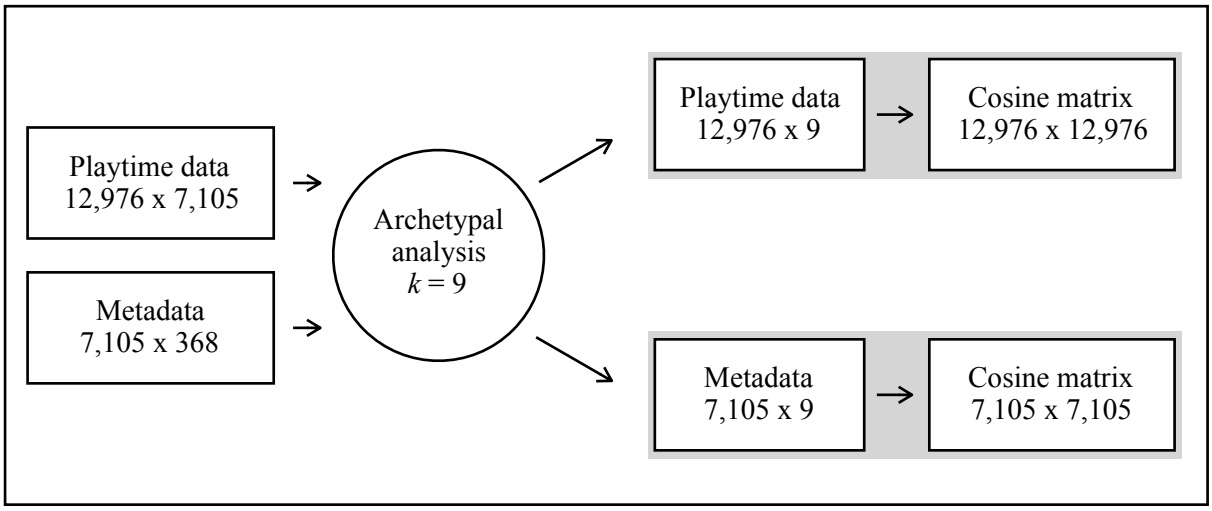


Figure 3. Schematic overview of calculating the similarity between both users and games.

3.4 Predicting playtime

Playtime was initially predicted using a twofold approach, incorporating both a collaborative filtering method and a content-based filtering method. The following section elaborates on how these predictions were made for each different approach.

Collaborative filtering method

For user *i*, the playtime on game *j* was predicted employing the same function that Sifa et al. (2014a) and Notten (2017) used in their studies. In this formula, *U* stands for the grouped 5 players that are most similar to user *i*. Playtime is predicted by multiplying the sum of the cosine similarities between user *i* and users *U* by the time that users *U* have played game *j*. In order to obtain the predicted playtime, the outcome of this multiplication has to be divided by the sum of the cosine similarities between user *i* and users *U*.

$$\hat{t}_{ji} = \frac{\sum_{u \in U} Sim(P_i, P_u) t_{ju}}{\sum_{u \in U} Sim(P_i, P_u)}$$

Content-based method

In the content-based method, the playtime for user *i* on game *j* was predicted using a modified version of the function that Sifa et al. (2014a) and Notten (2017) employed in their studies. In this variant of the formula, *U* represents the grouped 5 games that are most similar to game *j* among the 250 most popular games, which is an arbitrarily chosen threshold. Playtime is then predicted by multiplying the sum of the cosine similarities between game *j* and games *U* by the time that user *i* has spent playing games *U*. To retrieve the predicted playtime, the outcome of this multiplication has to be divided by the sum of the cosine similarities between game *j* and games *U*.

$$\hat{t}_{ji} = \frac{\sum_{u \in U} Sim(P_j, P_u) t_{ui}}{\sum_{u \in U} Sim(P_j, P_u)}$$

Each prediction task was carried out 100 times, by sampling 100 random games from the 500 most popular titles, and 100 random users from the total data for each iteration. Additionally, for each user in a given iteration, playtime was predicted on a (blinded) randomly selected game from the user's top three favorite games (in terms of playtime). These predictions were ultimately stored in a data frame (Appendix R), containing a total of 2,020,000 playtime predictions (100 users * 101 predictions * 100 iterations * 2 methods). An overview of the R code that was used for this prediction task can be found in Appendix O. Figure 4 serves as an overview of the general way in which predictions were made for both methods.

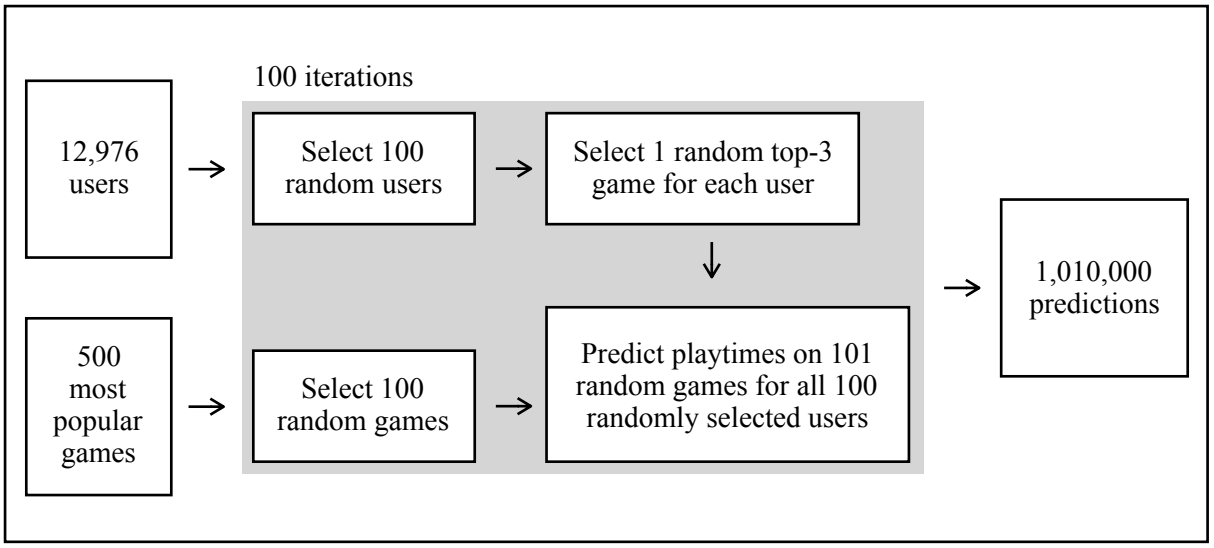


Figure 4. Schematic overview of the prediction task that was performed for all methods.

3.5 Evaluating (baseline) predictions

In order to evaluate whether the predictions that are derived from the two different algorithms are meaningful, 160,000 (100 * 100 * 16) playtime rankings were constructed for both algorithms (Appendix S). For each algorithm and all iterations, the 101 different predicted playtimes on 101 randomly selected games were ranked. If a users' randomly selected (blinded) top-3 game was included in a top-L of that ranking (L varying between 5 and 20), that user was assigned a recall value of 1 given that specific iteration and algorithm. This ultimately resulted in two vectors, both containing 320,000 recall values for all 16 different values of L. The means of these two vectors served as the total recall values of the two (baseline) recommender systems. An overview of the R code that was used for this evaluation task can be found in Appendix P.

3.6 Hybridizing predictions

The predictions of both models were then merged by hybridizing the recommendation components. This was accomplished by making use of a switching hybrid, in which the system switches between the models according to some criteria. In order to determine these criteria, a data frame was created that, for each model, contained the mean recall values for all players and all games (over all iterations).

A prediction task was subsequently executed in which the algorithm switched between formulas to predict playtime, based on the user it was trying to make predictions for. The formula that was used hereby depended on the mean recall values that were obtained for that user by the two different models; if on average the collaborative filtering component outperformed the content-based component, its matching function was employed to predict playtime for that user (and vice versa). Additionally, a prediction task was executed that based its function for prediction on the game it was trying to predict playtime on. The formula that was used hereby depended on the mean recall values that were obtained for that game by the two different models. An overview of the R code that was used for this hybridization task can be found in Appendix Q.

The predictions that were obtained (Appendix R) were then evaluated by running them through the exact same evaluation process as was described in Section 3.5. The hybridization method that resulted in the highest mean recall was subsequently picked to serve as the hybridization method of the definitive hybrid recommender system.

3.7 Evaluating recommender system

Ultimately, the performance of the hybrid recommender system was compared to the performances of the two individual systems that serve as the components of the hybrid system. These include 1) the recommender system using a collaborative filtering method, as was constructed by querying data from Steam's API, and 2) the system using a content-based method that was constructed by scraping metadata off Steam's website.

3.8 Software & hardware

All code and each analysis was executed using the programming language R (Version 3.4.4) in the RStudio IDE (Version 1.1.463). In order to query Steam's API, the packages 'httr' and 'jsonlite' were used to work with web addresses and parse JSON data. The 'rvest' package was used to web scrape Steam's website. Archetypal analysis was executed employing the 'archetypes' package, whereas the 'lsa' package was used to calculate cosine similarities. Other packages, like 'car', 'dplyr', 'ggplot2', 'magrittr', 'reshape2' and 'tidyr' were used for general efficiency purposes (Table 1).

An HP Omen 870-130nd, containing a 2.70 gigahertz quad core processor and 16 gigabytes of RAM memory, was used to execute all code and each analysis in this study.

Table 1

R packages used in this study

package	version	description
archetypes	2.2-0	Archetypal Analysis
car	3.0-2	Companion to Applied Regression
dplyr	0.7.5	A Grammar of Data Manipulation
ggplot2	2.2.1	Create Elegant Data Visualisations Using the Grammar of Graphics
httr	1.4.0	Tools for Working with URLs and HTTP
jsonlite	1.6	A Robust, High Performance JSON Parser and Generator for R
lsa	0.73.1	Latent Semantic Analysis
magrittr	1.5	A Forward-Pipe Operator for R
reshape2	1.4.3	Flexibly Reshape Data: A Reboot of the Reshape Package
rvest	0.3.2	Easily Harvest (Scrape) Web Pages
tidyr	0.8.3	Easily Tidy Data with ‘spread()’ and ‘gather()’ Functions

Note.

3.9 Summary

In this study, a hybrid recommender system was developed for Steam games, that makes use of both a collaborative filtering method (CF) and a content-based filtering method (CBF). Data on playtime of Steam users was harvested by querying Steam’s API, whereas data on the characteristics of the games that appeared in the data was collected by web scraping tags from the platform’s online store. The number of features in the data was then reduced by applying both feature selection (removing unpopular games) and feature extraction (applying archetypal analysis to the remaining features). This allowed for a faster calculation of the cosine similarities between all different users and games. Playtime was then predicted for 100 random users on 100 random games, 100 different times for each method. Additionally, for each user, playtime was predicted on a random (blinded) game from that user’s personal top

three. If this game subsequently appeared in the top-L of a ranking of these predictions (L varying between 5 and 20), the given user was assigned a recall value of 1 for that iteration. After evaluating both the collaborative filtering method and the content-based filtering method, these recommendations were hybridized using a switching hybrid, that switched between models based on the user it was trying to make predictions for. Additionally, a switching hybrid was constructed that switched between models based on the game it is trying to predict playtime on. These two switching hybrids were then evaluated once more, in order to assess which switching hybrid performed best overall. All code and each analysis was executed using the programming language R in the RStudio IDE.

4. Results

In this chapter, results are presented regarding the extraction of the features, the performances of the two non-hybrid recommender systems and the performances of the two switching hybrids. In order to be able to assess how well the extracted features maintain the structure of the original data, the RSS of the archetypal analyses are reported and discussed in Section 4.1. Additionally, in order to demonstrate the performances of the different non-hybrid recommender systems, their mean recall values over all iterations are reported in Section 4.2. These results are then further broken down to inspect the performances of the systems for different clusters of users and games. Section 4.3 serves as an overview of the performances of the two hybrid recommender systems. Ultimately, in Section 4.4, the results that were obtained in this study are compared to the results that were obtained in earlier work on recommender systems for Steam.

4.1 Feature extraction

Archetypal analysis was executed on the playtime data, as well as the metadata, in order to reduce the number of features (games and tags respectively). The residual sums of squares (RSS) of these analyses were obtained and reported, in order to assess the discrepancy between these extracted features and the original ones. An overview of the archetypes that were calculated can be found in Appendix K.

For the playtime data, 13 iterations were executed in order to converge, which took approximately 15 hours. This resulted in a RSS of 0.17. 12 iterations were executed in order to converge for the metadata, which resulted in a RSS of 0.71. Approximately 1 minute elapsed before converging. An overview of the results that were obtained from the feature extraction process can be found in Table 2.

Table 2

Comparison of archetypal analyses on playtime data and metadata

	Playtime data	**Metadata**
RSS	0.17	0.71
Number of iterations	13	12

Note. Original playtime data = 12,976 x 7,105. Original metadata = 7,105 x 368. $k = 9$.

Figure 5 serves as a visual representation of the archetypal analyses converging over iterations for both datasets.

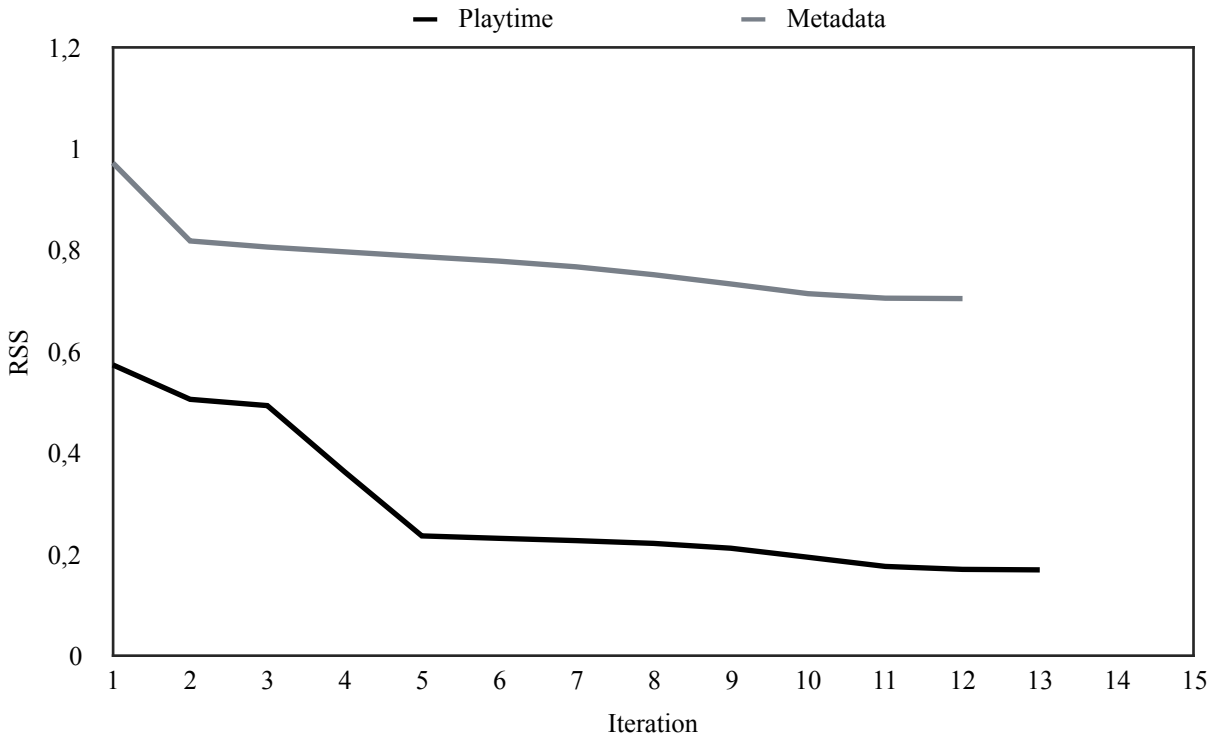


Figure 5. Visual representation of archetypal analysis converging over iterations for both datasets.

4.2 Predictions

The recommender system that only incorporated a collaborative filtering method (CF) resulted in a mean recall value of 80.9% (*SD* = 4.25). The system that solely based its recommendations on a content-based method (CBF) resulted in a mean recall value of 28.9% (*SD* = 4.68). The collaborative filtering method thereby outperformed the content-based method.

In order to gain insight into how the two systems performed for different groups of users, two instances of *k*-means clustering were executed, that partitioned the users in the evaluation sample based on the size of their game collection ($k = 3$) and their average cosine similarity to the five users that were most similar to them ($k = 2$). Additionally, two instances of *k*-means clustering were executed that partitioned the evaluated games based on the size of their user base ($k = 3$) and their average cosine similarity to their five most similar counterparts ($k = 2$). The different values that were picked for *k* were arbitrarily chosen. A visual representation of these clusters, in the form of density plots, is presented in Figure 6.

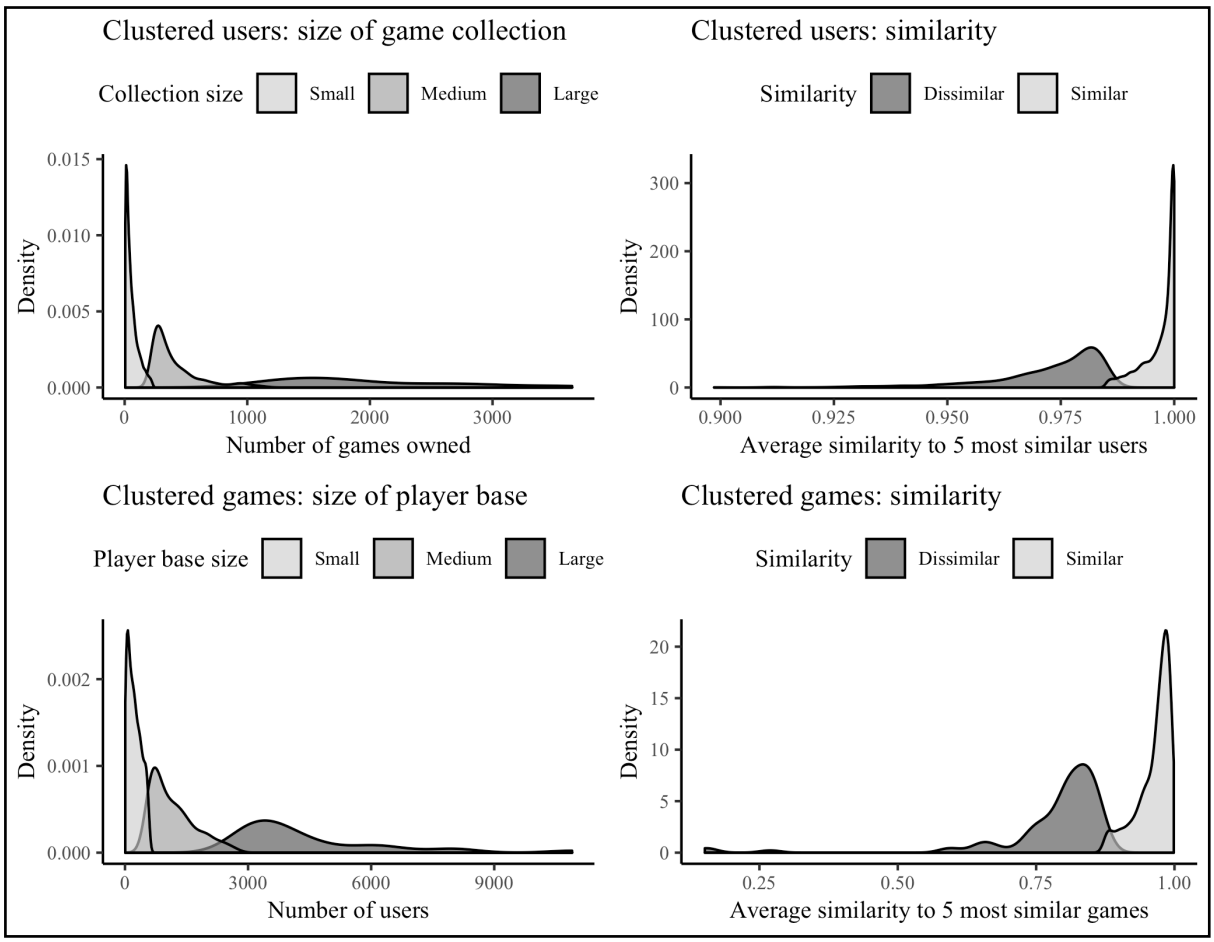


Figure 6. Density plots of the different clusters that were obtained through *k*-means clustering.

Regarding the clustering of users based on the size of their game collection ($k = 3$), users with a "small" collection own approximately 55 games ($n = 6568$), on average. Users with a "medium" game collection, on average, own approximately 406 games ($n = 402$), whereas a "large" collection averagely consists of approximately 1,966 games ($n = 24$). Concerning the

clustering that was based on similarity ($k = 2$), "dissimilar" users are users who have an average cosine similarity of approximately 0.97 to the five players that are most similar to them. "Similar" users, on the other hand, are users with an average cosine similarity of approximately 1.00 to their five most similar counterparts.

Concerning the partitioning of games based on the size of their player base ($k = 3$), games with a "small" player base ($n = 441$) have been played by an average of 228 users. A "medium" player base averagely consists of 1,162 users ($n = 212$), whereas games with a "large" player base ($n = 31$) have been played by an average of 4,459 users. Regarding the clustering that was based on similarity ($k = 2$), "dissimilar" games are games that have an average cosine similarity of approximately 0.78 to their five most similar counterparts. "Similar games" are games with an average cosine similarity of approximately 0.96 to the five games that are most similar to them. The average recall values that were obtained by both methods are listed in Table 3 for all different clusters of users and games. This table also includes the relative number of occurrences that the collaborative filtering method outperformed the content-based method for each cluster and vice versa.

The results that are listed in Table 3 show that the collaborative filtering method outperformed the content-based method for almost all different clusters of users and games in the evaluation sample, except for the cluster of games with a relatively small player base. For these games ($n = 441$), the collaborative filtering method resulted in an average recall value of 17.2% ($SD = 33.08$), whereas the content-based filtering method achieved a mean recall value of 23.3% ($SD = 35.75$). A possible explanation for this difference is that in the collaborative filtering method, games with a relatively small player base have a lower chance of appearing in the collections of the 5 most similar players to a given user. After all, a high similarity between users can be achieved without them having identical game collections.

Table 3

Mean recall values of the predictions, grouped by multiple clusters

			CE		CBE			
cluster	***n***	**mean***	**recall**	***SD***	**recall**	***SD***	**CF > CBF**	**CBF > CF**
Overall (100 iterations)	100	—	**80.9%**	4.25	28.9%	4.68	**56.6%**	4.6%
All users in the evaluation sample	6994	82	**81.1%**	36.62	28.9%	39.49	**69.7%**	6.3%
Users with a small collection	6568	55	**81.6%**	36.24	28.9%	39.48	**70.0%**	6.0%
Users with a medium collection	402	406	**75.3%**	41.10	29.0%	39.94	**65.4%**	11.5%
Users with a large collection	24	1966	**55.9%**	43.89	26.2%	37.38	**59.1%**	22.7%
Similar users	6240	1.00	**81.0%**	36.65	28.4%	39.34	**69.6%**	6.0%
Dissimilar users	754	0.97	**82.1%**	36.44	32.6%	40.58	**70.2%**	8.8%
All games in the evaluation sample	684	468	**29.0%**	36.99	23.6%	33.15	**36.4%**	26.8%
Games with a small player base	441	228	17.2%	33.08	**23.3%**	35.75	18.8%	**31.3%**
Games with a medium player base	212	1162	**45.5%**	32.83	23.2%	27.90	**65.1%**	19.8%
Games with a large player base	31	4459	**84.0%**	20.51	31.4%	27.14	**90.3%**	9.7%
Similar games	589	0.96	**30.2%**	36.71	24.6%	33.03	**38.7%**	27.5%
Dissimilar games	95	0.78	**21.2%**	37.97	17.7%	33.46	22.1%	22.1%

Note. CF = Collaborative Filtering. CBF = Content-Based Filtering.

* The means reflect the average number of games in the collection of a user, the cosine similarity to the 5 most similar users for the average user, the average number of users that own a given game, and the cosine similarity to the 5 most similar games for the average game respectively.

These results furthermore indicate that the bigger the game collection of a user got, the harder it was for the collaborative filtering method to outperform the content-based filtering method in making the most accurate recommendations. An explanation for this effect could be that it is harder to find similar users as more games appear in a users' collection. Additionally, the popularity of a given game in the evaluation sample seems to have an effect on the relative number of occurrences in which the models outperformed each other. As mentioned before, games that are played by a relatively small player base have a lower chance of appearing in the game collections of the 5 most similar players to a given user; this serves as a possible explanation as to why this effect occurs. These two observations are visualized in Figure 7.

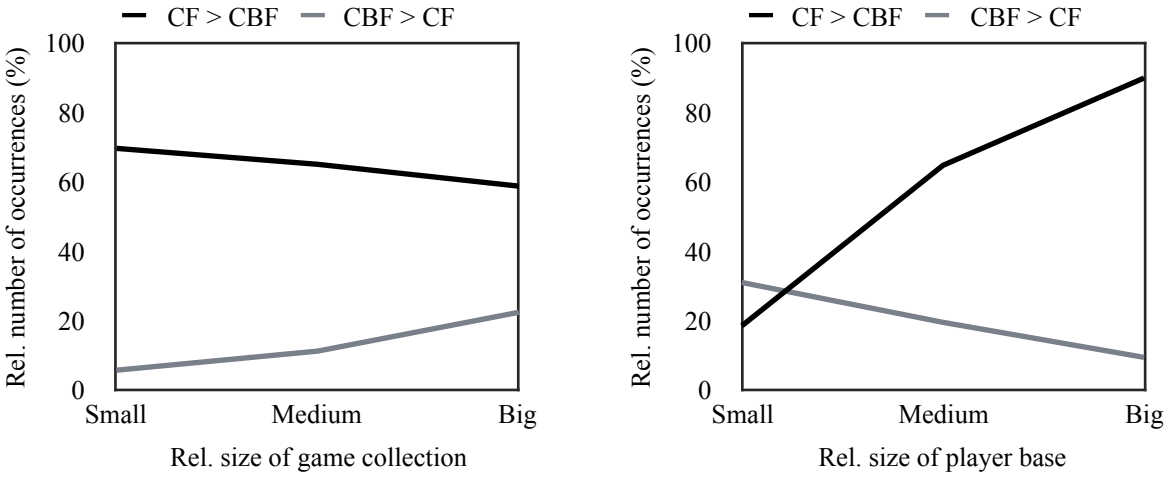


Figure 7. Relative number of occurrences in which the two different methods were able to outperform each other, grouped by the size of a users' game collection (left) and the size of a games' player base (right).

For the cluster of users that are relatively dissimilar to others, the collaborative filtering method (M = 82.1%, SD = 36.44) still outperformed the content-based method (M = 32.6%, SD = 40.58) in terms of mean recall. Interestingly, the collaborative filtering method furthermore resulted in a higher mean recall value for users who are relatively dissimilar to others, compared to users who are relatively similar (M = 81.0%, SD = 36.65). However, the users who are deemed to be relatively dissimilar to others still have an average cosine similarity of 0.97 to their 5 most similar counterparts, which makes this otherwise small difference less notable.

The collaborative filtering method resulted in the highest mean recall value for the cluster of games with a relatively big player base (M = 84.0%, SD = 20.51), compared to its mean recall values for other clusters. From a theoretical standpoint, this can be explained by the fact that games with a relatively big player base have a higher chance of appearing in the collections of the 5 most similar players to a given user. The content-based filtering method, on the other hand, resulted in the highest mean recall value for the cluster of users that are relatively dissimilar to others (M = 32.6%, SD = 40.58) compared to its mean recall for other clusters.

4.3 Hybridizing predictions

A mean recall value of 84.6% (SD = 3.88) was obtained for hybrid recommender system A, that incorporated a switching hybrid based on the average performance per user. Hybrid

system B, that employed a switching hybrid based on the average performance per game, resulted in a mean recall value of 83.1% (*SD* = 4.32). Hybrid system A therefore outperformed system B, and additionally, the system surpassed the best performing non-hybrid system that was constructed in this study. The results of an independent samples T-test show that there is a significant difference in the mean recall values that were obtained for the best performing non-hybrid system (*M* = 80.9, *SD* = 4.25) and the best performing hybrid system (*M* = 84.6, *SD* = 3.88); $t(99) = -20.12$, $p < .0001$. An overview of the results of this T-test, including the results of the assumption checks, can be found in Appendix T. Table 4 serves as an overview of the different overall recall values that were obtained by the two hybrid systems, as well as those that were obtained by the two non-hybrid systems.

Table 4

Comparison of evaluation metrics between systems

method	**recall**	**iterations *(n)***	***SD***
CF	80.9%	100	4.25
CBF	28.9%	100	4.68
Switching hybrid A (users)	**84.6%**	100	**3.88**
Switching hybrid B (games)	83.1%	100	4.32

Note. CF = Collaborative Filtering. CBF = Content-Based Filtering.

On average, the two hybrid systems disagreed on which non-hybrid system was to be used in 9.0% of the iterations that were executed (*SD* = 2.83). For 15.4% of the 684 games that appeared in the evaluation sample, the two hybrids disagreed (*SD* = 29.48). When these 684 games are divided over the three clusters that separate the games based on the size of their player base, the games with a relatively medium player base were most often disagreed upon (*M* = 18.2%, *SD* = 27.85). Divided over the two clusters that separate the games based on their similarity to other games, disagreement most often occurred when games are relatively similar to the 5 games that are most similar to them (*M* = 16.2%, *SD* = 29.55). These results are listed in Table 5.

Table 5

Mean recall values of the predictions

cluster	***n***	**mean***	**disagreement**	***SD***
Overall (100 iterations)	100	—	9.0%	2.83
All users in the evaluation sample	6994	82	8.6%	26.33
Users with a small collection	6568	55	8.5%	26.23
Users with a medium collection	402	406	**9.9%**	28.06
Users with a large collection	24	1966	8.3%	24.08
Similar users	6240	1.00	8.6%	26.35
Dissimilar users	754	0.97	8.4%	26.23
All games in the evaluation sample	684	468	15.4%	29.48
Games with a small player base	441	228	14.6%	30.71
Games with a medium player base	212	1162	**18.2%**	27.85
Games with a large player base	31	4459	8.0%	19.40
Similar games	589	0.96	**16.2%**	29.55
Dissimilar games	95	0.78	10.9%	28.82

Note. CF = Collaborative Filtering. CBF = Content-Based Filtering.

* The means reflect the average number of games in the collection of a user, the cosine similarity to the 5 most similar users for the average user, the average number of users that own a given game, and the cosine similarity to the 5 most similar games for the average game respectively.

When the evaluations are grouped by the 6,994 different users in the sample, the two hybrids disagreed for 8.6% of these users (*SD* = 26.33). When these users are clustered based on the size of their game collection, users with a relatively medium game collection were most often met with disagreement (*M* = 9.9%, *SD* = 28.06). These results seem to indicate that the hybrid systems agree more often in the event of extremes, such as players with a relatively small or large game collection, or games with a relatively small or large player base. This makes sense from a theoretical point of view, as for a user with a relatively small game collection, the collaborative filtering method is clearly preferable above the content-based method, whereas this is not so evidently the case for users with a relatively medium-sized collection. After all,

it is less likely that users with a relatively small collection have played any of the five games that are most similar to the game that a content-based method is trying to make predictions for.

4.4 Evaluating recommender system

Compared to the recommender system that was developed by Notten (2017), both hybrid systems that were constructed in this study resulted in higher overall recall values (Table 6). Whereas Notten (2017) achieved a mean recall value of 82.2% with a stand-alone collaborative filtering method, the two switching hybrids that were developed this study (A and B) resulted in mean recall values of 84.6% and 83.1% respectively. Compared to the system that was constructed by Sifa et al. (2014a), all systems that were developed resulted in lower overall recall values than the approximately 90% overall recall value that was achieved by their system. This can largely be explained by the fact that data on more users and fewer games was harvested in this study, which most likely resulted in better fits of the archetypal analyses and enabled the matching of counterparts that were (slightly) more similar. Additionally, it could be argued that the sample that was used in this study differs from the samples that were used in the other studies, due to the restrictions that Steam's API was subjected to. More discussion on the effects of these restrictions can be found in Chapter 5.

Table 6

Comparison of evaluation metrics between studies

study	**method**	**users (*n*)**	**games (*n*)**	**recall**	***SD***
Sifa et al. (2014a)	CF	100000	3007	~ 90%	—
Notten (2017)	CF	13033	4673	82.2%	4.45
Notten (2017)	CF (achievements)	13033	4673	82.1%	4.09
Boon (2019)	CF	12976	7105	80.9%	4.25
Boon (2019)	CBF	12976	7105	28.9%	4.68
Boon (2019)	CF + CBF	12976	7105	84.6%	3.88

Note. CF = Collaborative Filtering. CBF = Content-Based Filtering.

Archetypal analyses: $k = 9$.

5. Discussion

In this chapter, the findings of this study and its contribution to the existing framework will be discussed. Section 5.1 serves as a general discussion about the results that were obtained. Section 5.2 presents an overview of the way in which the system that was developed in this research could be incorporated by Steam. The limitations that accompany this study are highlighted in Section 5.3, whereas suggestions for future research are given in Section 5.4. The chapter is concluded with a short summary in Section 5.5.

5.1 General discussion

This study aimed to improve a non-hybrid recommender system for Steam, by combining multiple different algorithmic approaches into a single prediction. This was achieved by combining a collaborative filtering method with a content-based filtering method, after which the predictions of both methods were hybridized using two different switching hybrids. The switching hybrid that ultimately resulted in the highest mean recall value switched between the two different filtering methods based on the user it was trying to predict playtime for. Compared to the stand-alone collaborative filtering method, which serves as the best performing non-hybrid system, this is a significant improvement.

Although the mean recall values that were obtained are inferior to those achieved by Sifa et al. (2014a), the most likely hypothesis for this difference in recall is that in their studies, data on a larger number of users could be retrieved from Steam's API. However, the results of this study do show that by combining multiple different algorithmic approaches, a hybrid recommender system for Steam games is able to outperform a non-hybrid recommender system for Steam games. These findings support the theoretical framework underlying this study, from which could be derived that hybrid recommender systems are generally able to outperform non-hybrid systems (Burke, 2002; Çano & Morisio, 2017).

5.2. Practical implementation

Currently, Steam's recommender system bases its recommendations on 1) the similarity between a given game and the games owned by a user, 2) the general popularity of a given

game and 3) the novelty of a given game (Steam, 2014). Although not much more is known about the workings of the Discovery Queue, these criteria display a motivation to prioritize popular and novel games in making recommendations to users. This is understandable for a number of reasons: it is more likely that a user will show interest in a popular, novel game, and it saves both time and computational power compared to predicting playtime for all unique games on Steam. Therefore, in order to implement the hybrid system that was developed in this study, it is advised to do so as is visualized in Figure 8.

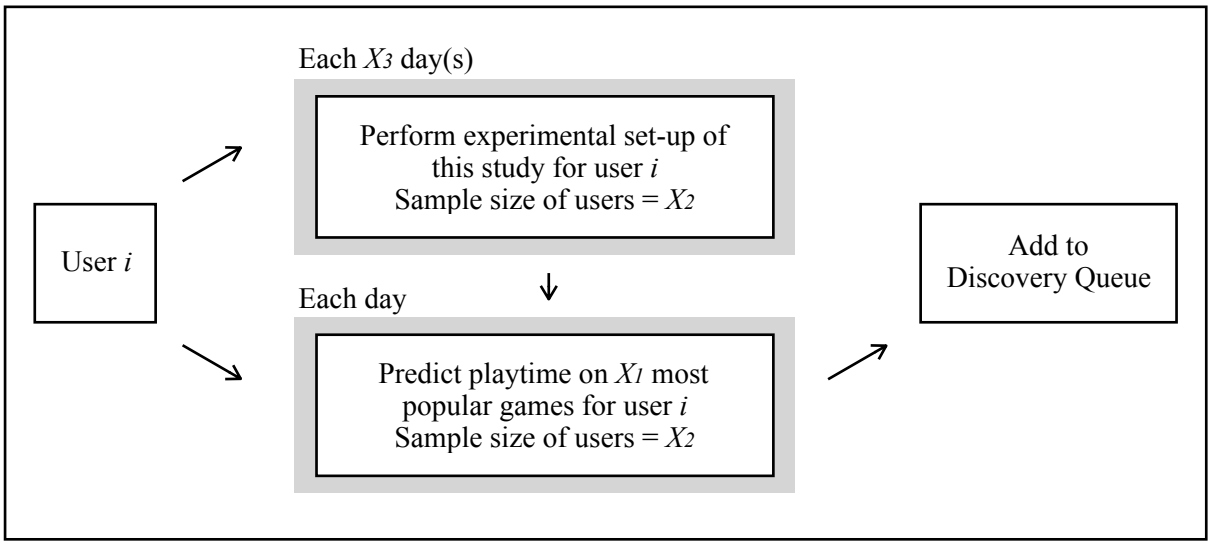


Figure 8. Schematic overview of the way in which Steam could incorporate the hybrid system.

It is suggested that, for a given user, playtime is predicted on the X_1 most popular games (at that point in time) by both the collaborative filtering method and the content-based method. Predictions should only be made for games that are not already owned by the given user and have not already been rejected by the given user. This prediction task could be executed daily, which is the same time interval at which the Discovery Queue currently refreshes. The data that is needed in order to execute this task, such as the cosine similarities to similar users, could be calculated periodically using a randomly drawn sample of playtime data (number of users = X_2; days between calculating similarities = X_3). In order to evaluate which switching hybrid performs best for the given user, the experimental setup of this study should be executed periodically as well, using the same randomly drawn sample of playtime data (number of users = X_2). Of course, predictions and evaluations would only have to be made for the given user, drastically reducing the amount of time needed for the algorithm to

converge. In turn, the X_1 most popular games could be presented to the user, ranked in descending order based on the predicted amount of playtime by the best performing switching hybrid for that particular user. The four mentioned parameters (X_1, X_2, X_3 and X_4) can be tuned according to the available time and computational power.

5.3 Limitations

The restriction of Steam's API by Valve Corporation served as the most influential limitation of this study, as a lot more data on playtime from Steam users could have been harvested without it. Compared to the approximately 1.8 million Steam ID's that were ran through the API, only 1.1% of these requests resulted in actual playtime data of users. Being able to gather more data would have been beneficial to the accuracy of the recommendations, as the more data is available, the higher the chances that both users and games can be matched with highly similar counterparts. Results show that for games, the accuracy of recommendations benefits from a higher similarity to their complements. Currently, data on playtime can only be gathered from users who have explicitly chosen to share their data with the public. Next to the accuracy issues that are caused by the limited data harvest, the fact that data could only be gathered on users who made a conscious choice to open up their profile to the public could have led to a sample of users that is not representative for the entire population of Steam users.

The limited time frame in which this research was conducted serves as another limitation to this study. With more time (or computational power) at hand, data on more users could have been collected from Steam's API, which in turn presumably would have led to higher accuracies for the different recommender systems that were developed. More time or computing power could have also enabled archetypal analysis to be carried out with a lower minimal value of improvement between two iterations, as well as a higher maximum number of iterations. This would logically have resulted in higher accuracy values as well, due to a decreased discrepancy between the extracted features and the original ones. For similar reasons, archetypal analysis could have been executed for multiple different (higher) values of k. The evaluation of the different recommender systems would have been aided by a less limited time frame as well, as more iterations of the evaluation process could have been

executed as a consequence. In turn, this would have led to a more accurate assessment of the different systems' accuracies.

5.4 Future research

In future research, more algorithmic approaches could be combined on top of a collaborative and a content-based filtering method. An example of a possible method to incorporate is demographic filtering, in which demographic data is used in order to identify categories of users. With Steam's API, it is possible to query demographic data on its users, such as city, state and country of residence. However, as generally few people disclose this information to the public, and as this data is entirely self-reported, questions concerning reliability could arise (Çano & Morisio, 2017).

The way in which the collaborative filtering method and content-based method are designed in this study could be altered in future research as well. As the methods in this study are both memory-based, studies could be conducted into building or adding model-based methods that predict playtime on Steam games. These methods could, for instance, include the usage of latent factor and matrix factorization models.

Additionally, more different hybridization techniques could be incorporated into the system, such as a weighted hybrid or a form of feature combination. In case of feature combination, the information that is derived from the collaborative filtering method could, for instance, be used as additional feature data in the content-based filtering method.

5.5 Summary

By combining a collaborative filtering method with a content-based filtering method, this study aimed to improve the performance of a non-hybrid recommender system for Steam games. The results that were obtained show that a hybridization of these two methods is indeed able to outperform systems that only make use a singular approach. The limited time frame and computational power at hand, as well as the restrictions that were applied to Steam's API, serve as limitations to this study. Suggestions for future research include

combining more algorithmic approaches, using a model-based method on top of a memory-based one and incorporating more different hybridization techniques into the system.

6. Conclusion

In this study, the following research question was formulated:

RQ: Can a non-hybrid recommender system for Steam be improved by combining a content-based filtering method with a collaborative filtering method?

Both hybrid recommender systems that were constructed did result in higher mean recall values compared to those obtained by the two non-hybrid systems. Therefore, it can be concluded that a hybridization of a content-based method and a collaborative filtering method can indeed result in a better performing recommender system for Steam games. These findings support the theoretical framework underlying this study, from which could be derived that hybrid recommender systems are generally able to outperform non-hybrid systems. The results that were obtained in this study thereby underline the advantages of hybridizing multiple different approached, compared to fine-tuning and polishing one particular approach, in the event of designing a recommender system for Steam games. Future research could therefore include more algorithmic approaches and hybridization techniques, in order to achieve better results.

Acknowledgements

This Master's Thesis marks the end of my higher education at Tilburg University. I look back with content at a period in which I learned a lot about the inner workings of the human mind during my bachelor studies, and the opportunities of big data during my master studies.

First, I would like to thank my parents for unconditionally supporting me in so many ways, throughout the course of my life. I would also like to thank my supervisor, prof. dr. ir. Pieter Spronck, for the pleasant collaboration and the helpful insights that he gave me. Thanks go out to my sister, my friends (and co-founders) and my lovely grandmothers as well, and of course to Jessica; my moon and my rock in life.

Justin Boon
Tilburg, April 2019

References

Ahn, S., Kang, J., & Park, S. (2017). What makes the difference between popular games and unpopular games? Analysis of online game reviews from Steam platform using word2vec and bass model. *ICIC Express Letters*, *11*(12), 1729-1737.

Bell, R. M., Koren, Y., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, *42*(8), 30-37.

Berger, J., Draganska, M., & Simonson, I. (2007). The influence of product variety on brand perception and choice. *Marketing Science*, *26*(4), 460-472. doi:10.1287/mksc.1060.0253

Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, *12*(4), 331-370. doi:10.1023/A:1021240730564

Çano, E., Morisio, M. (2017). Hybrid Recommender Systems: A Systematic Literature Review. *Intelligent Data Analysis*, *21*(6), 1487-1524. doi:10.3233/IDA-163209

Chernev, A. (2003). Product assortment and individual decision processes. J*ournal of Personality and Social Psychology*, *85*(1), 151-162. doi:10.1037/0022-3514.85.1.151

Chernev, A., Böckenholt, U., & Goodman, J. (2015). Choice overload: A conceptual review and meta-analysis. *Journal of Consumer Psychology*, *25*(2), 333-358. doi:10.1016/j.jcps.2014.08.002

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., & Sartin, M. (1999). Combining content-based and collaborative filters in an online newspaper. *Proceedings of the ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*. Retrieved from https://web.cs.wpi.edu/~claypool/papers/content-collab/content-collab.pdf

Cutler, A., & Breiman, L. (1994). Archetypal Analysis. *Technometrics*, *36*(4), 338-347.

Drachen, A., Sifa, R., Bauckhage, C., & Thurau, C. (2012). Guns, swords and data: Clustering of player behavior in computer games in the wild. *Proceedings of IEEE Computational Intelligence in Games*, 163-170. doi:10.1109/CIG.2012.6374152

Emmery, C. (2017, March 25). Euclidean vs. Cosine Distance [Blog post]. Retrieved from https://cmry.github.io/notes/euclidean-v-cosine

Eugster, M. J. A., & Leisch, F. (2009). From Spider-Man to Hero — Archetypal Analysis in R. *Journal of Statistical Software*, *30*(8), 1-23.

Hill, W., Stead, L., Rosenstein, M., & Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 194-201. doi:10.1145/223904.223929

Iyengar, S. S., & Lepper, M. R. (2000). When choice is demotivating: Can one desire too much of a good thing? *Journal of Personality and Social Psychology*, *79*(6), 995-1006. doi:10.1037/0022-3514.79.6.995

Karlgren, J. (1990). *An Algebra for Recommendations: Using Reader Data as a Basis for Measuring Document Proximity* (The Systems Development and Artificial Intelligence Laboratory, Stockholm University Working Paper No. 179). Retrieved from https://pdfs.semanticscholar.org/d663/d25cbc8212adf560b2b1f19a8800bd610ec2.pdf

Khusro, S., Ali, Z., & Ullah, I. (2016). Recommender Systems: Issues, Challenges, and Research Opportunities. In: K. J. Kim, & N. Joukov (Ed.), *Information Science and Applications*. Singapore, Singapore: Springer.

Lika, B., Kolomvatsos, K., & Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert Systems with Applications*, *41*(2), 2065-2073. doi:10.1016/j.eswa.2013.09.005

Lops, P., de Gemmis, M., & Semeraro, G. (2010). Content-based Recommender Systems: State of the Art and Trends. In: F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Ed.), *Recommender Systems Handbook*. Heidelberg, Germany: Springer.

Melinat, P., Kreuzkam, T., & Stamer, D. (2014, September). Information Overload: A Systematic Literature Review. Paper presented at *Perspectives in Business Informatics Research*. Retrieved from https://www.researchgate.net/publication/265906917_Information_Overload_A_Systematic_Literature_Review

Melville, P., & Sindhwani, V. (2017). Recommender Systems. In: C. Sammut, & G. I. Webb (Ed.), *Encyclopedia of Machine Learning and Data Mining*. Boston, United States: Springer.

Newzoo. (2018). Global Games Market Revenues in 2018. Retrieved from https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2018-light-version/

Notten, B. (2017). *Improving Performance for the Steam Recommender System using Achievement Data* (Master's thesis, Tilburg University). Retrieved from http://arno.uvt.nl/show.cgi?fid=144995

Pazzani, M. J., & Billsus, D. (2007). Content-Based Recommendation Systems. In P. Brusilovsky, A. Kobsa, & W. Nejdl (Ed.), *The Adaptive Web: Methods and Strategies of Web Personalization* (pp. 325-341). Berlin, Germany: Springer.

PC Gamer. (2019a). Steam now has 30,000 games. Retrieved from https://www.pcgamer.com/steam-now-has-30000-games/

PC Gamer. (2019b). Steam now has 90 million monthly users. Retrieved from https://www.pcgamer.com/steam-now-has-90-million-monthly-users/

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175-186. doi: 10.1145/192844.192905

Schafer, J. B., Konstan, J., & Riedl, J. (1999). Recommender Systems in E-Commerce. *Proceedings of the First ACM Conference on Electronic Commerce*, 158-166. doi:10.1145/336992.337035

Sifa, R., Bauckhage, C., & Drachen, A. (2014a). Archetypal Game Recommender Systems. *Proceedings of the 16th LWA Workshops: KDML, IR and FGWM (CEUR Workshop Proceedings)*, 45-56.

Sifa, R., Bauckhage, C., & Drachen, A. (2014b). The Playtime Principle: Large-scale Cross-games Interest Modeling. *Proceedings of IEEE Computational Intelligence in Games*, 365-372. doi:10.1109/CIG.2014.6932906

Sifa, R., Drachen, A., Bauckhage, C., Thurau, C., & Canossa, A. (2013). Behavior Evolution in Tomb Raider Underworld. *Proceedings of IEEE Computational Intelligence in Games*, 1-8. doi:10.1109/CIG.2013.6633637

Steam. (2014). The Steam Discovery Update: A Smarter Storefront, Personalized Just For You. Retrieved from https://store.steampowered.com/about/newstore

Steam. (2018). New Profile Privacy Settings. Retrieved from https://steamcommunity.com/games/ 593110/announcements/detail/1667896941884942467

Su, X., & Khoshgoftaar, T. M. (2009). A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, *2009*, 1-19. doi:10.1155/2009/421425

VentureBeat. (2017). Valve won't manually curate Steam because it dominates PC gaming. Retrieved from https://venturebeat.com/2017/02/13/valve-wont-manually-curate-steam-because-it-dominates-pc-gaming /

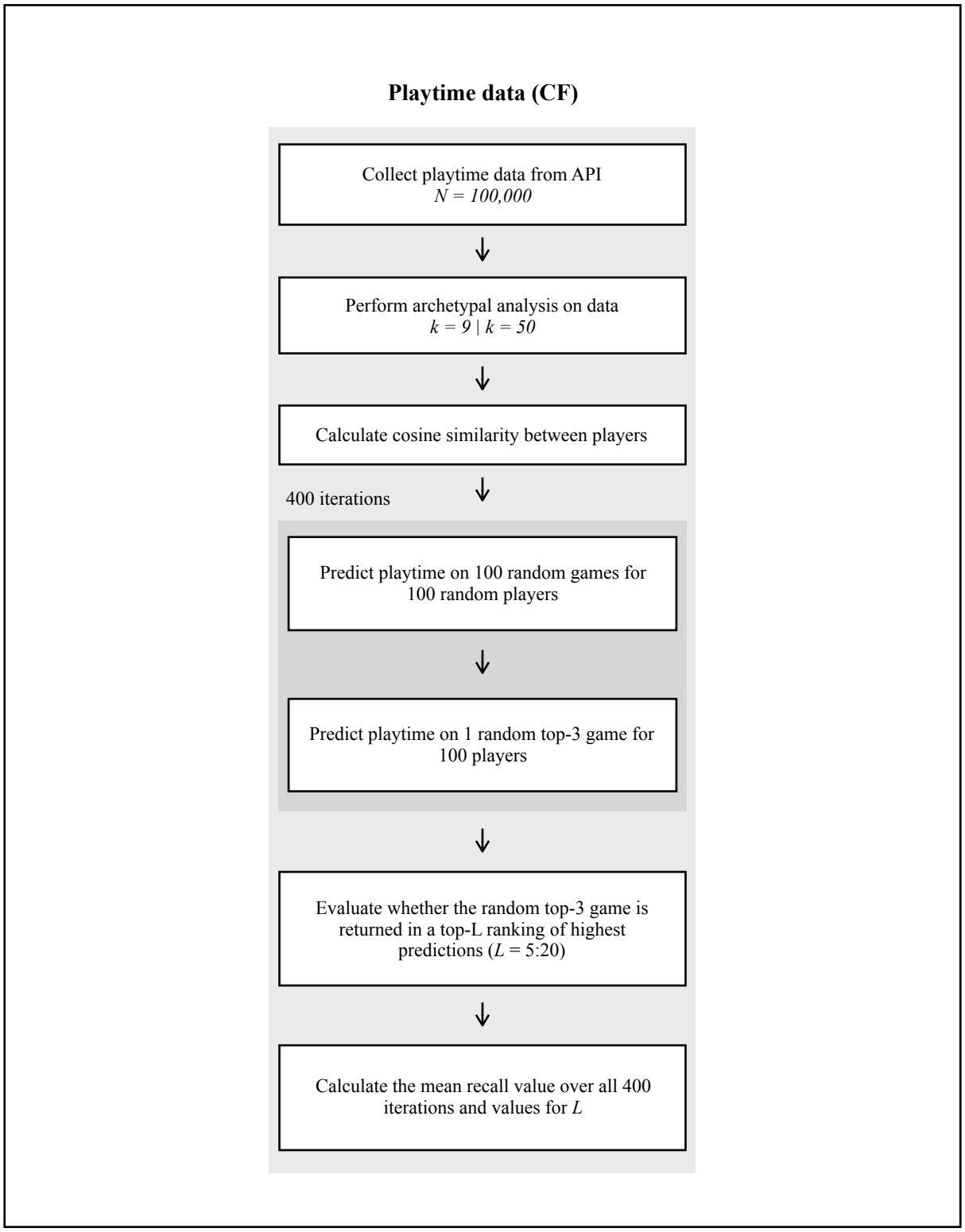
Windlehart, T. W., Jett, J., Schmalz, M., & Lee, J. H. (2016). Full Steam Ahead: A Conceptual Analysis of User-Supplied Tags on Steam. *Cataloging & Classification Quarterly*, *54*(7), 418-441. doi:10.1080/01639374.2016.1190951

Wired. (2009, September 22). How The Netflix Prize Was Won. Retrieved from: https://www.wired.com/2009/09/how-the-netflix-prize-was-won/

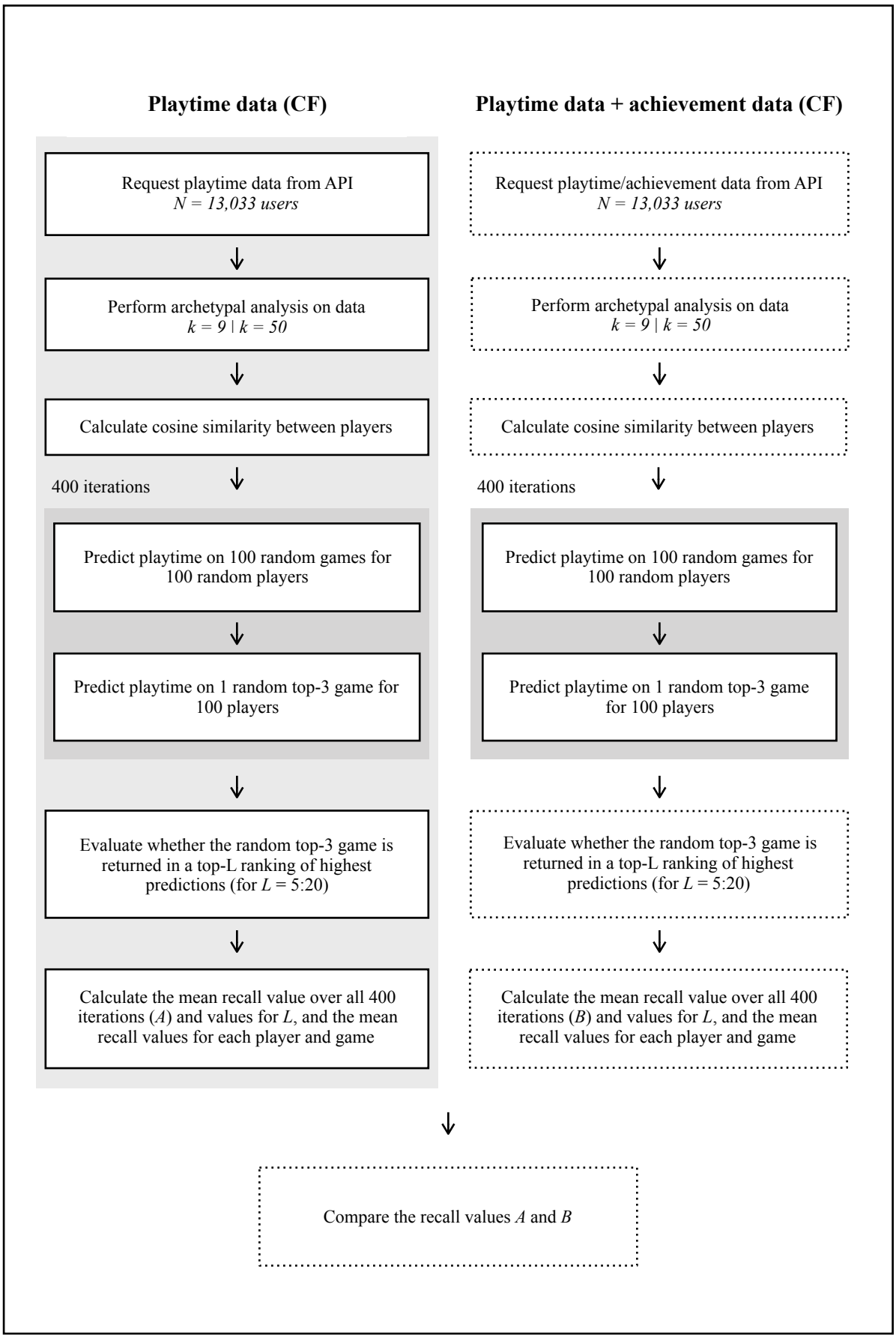
Zheng, Y., Agnani, M., & Singh, M. (2017, November). Identification of Grey Sheep Users By Histogram Intersection In Recommender Systems. Paper presented at *The 13th International Conference on Advanced Data Mining and Applications*. Retrieved from https://www.researchgate.net/publication/319302088_Identifying_Grey_Sheep_Users_By_The_Distribution_of_User_Similarities_In_Collaborative_Filtering

Appendix

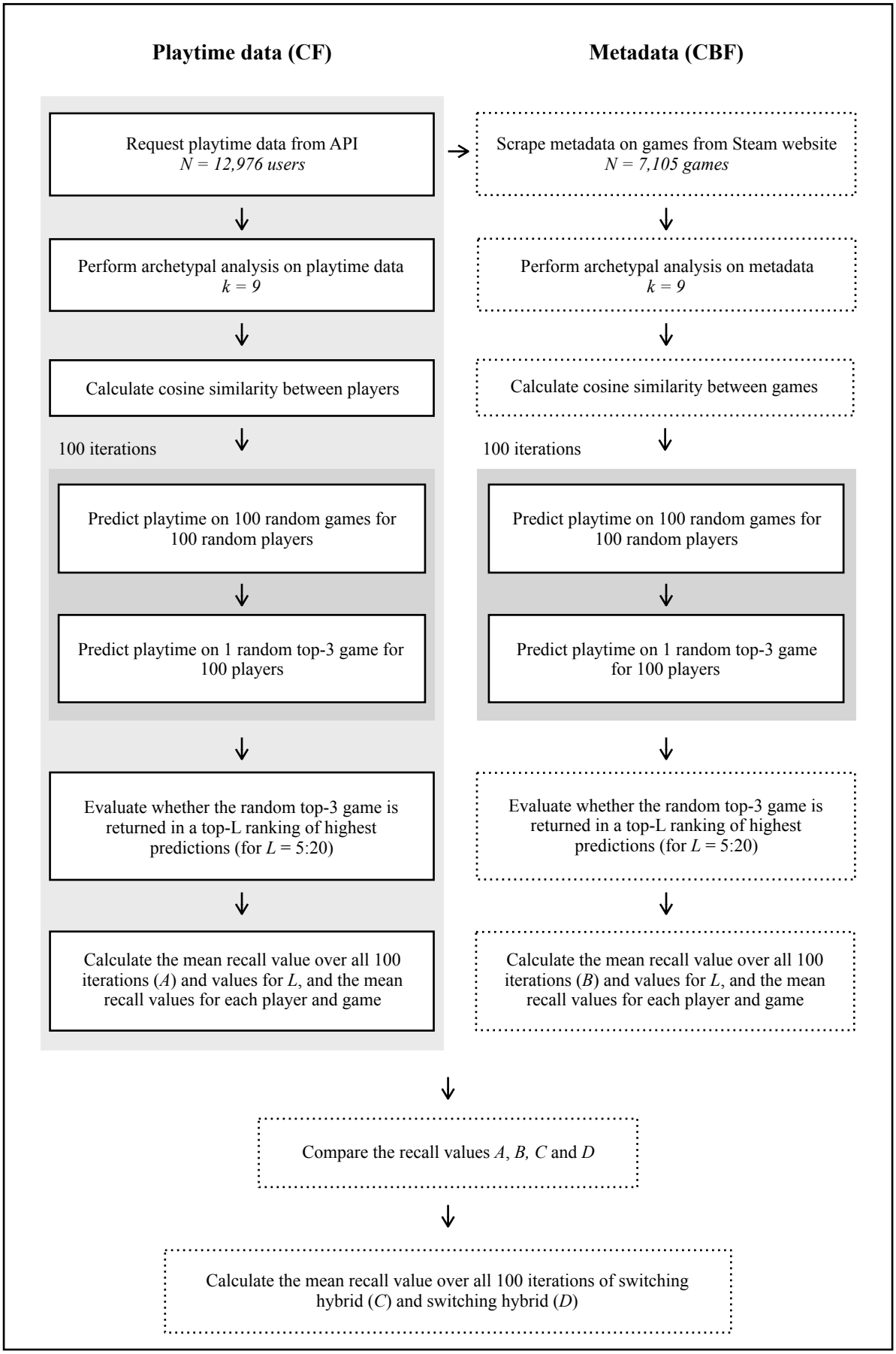
A) Schematic overview Sifa et al. (2014a)



B) Schematic overview Notten (2017)



C) Schematic overview current study



D) R code for harvesting friend lists from Steam's API

R code for harvesting friend lists from Steam's API

```
# Loading required packages
library("dplyr")
library("httr")
library("jsonlite")
library("magrittr")

# Setting up an empty data frame to store data on friends
friends_raw <- data.frame(friendslist.friends.steamid = character(),
                          friendslist.friends.relationship = character(),
                          friendslist.friends.since = numeric(),
                          stringsAsFactors = FALSE)

# Making an API request for each Steam ID
for (x in steam_ids$player_id){
      temp <- paste("http://api.steampowered.com/ISteamUser/
                    GetFriendList/v0001/
                    ?key=################################
                    &steamid=", x, "&relationship=friend", sep = "")

      friend <- GET(temp)

      text_content <- content(friend, "text", encoding = "UTF-8")

      # Preventing errors from terminating the loop
      if (text_content == "{\"response\":{}}") next
      if (grepl("Error", text_content) == TRUE) next
      if (grepl("error", text_content) == TRUE) next

      json_content <- text_content %>% fromJSON
      owned_games <- as.data.frame(json_content)

      friends_raw <- bind_rows(friends_raw, friend)}

# Pre-processing data on friends
friends <- friends_raw %>%
      select(1) %>%
      rename(player_id == friendslist.friends.steamid) %>%
      unique()
```

E) R code for harvesting playtime data from Steam's API

R code for requesting playtime data from the Steam API

```
# Loading required packages
library("dplyr")
library("httr")
library("jsonlite")
library("magrittr")

# Setting up an empty data frame to store playtime data
playtime_raw <- data.frame(player_id = character(),
                           response.games.name = character(),
                           response.games.playtime_forever = numeric(),
                           stringsAsFactors = FALSE)

# Making an API request for each Steam ID
for (x in steam_ids$player_id){
      temp <- paste("http://api.steampowered.com/IPlayerService/
                     GetOwnedGames/v0001/?
                     key=################################&steamid=", x,
                    "&format=json&include_appinfo=1&include_played_
                     free_games=1", sep = "")

      owned_games <- GET(temp)

      text_content <- content(owned_games, "text", encoding =
                           "UTF-8")

      # Preventing errors from terminating the loop
      if (text_content == "{\"response\":{}}") next
      if (grepl("Error", text_content) == TRUE) next
      if (grepl("error", text_content) == TRUE) next

      json_content <- text_content %>% fromJSON
      owned_games <- as.data.frame(json_content)

      owned_games$player_id <- x

      playtime_raw <- bind_rows(playtime_raw, owned_games)}

# Pre-process the playtime data
playtime <- playtime_raw %>%
      select(1, 6, 5, 7) %>%
      rename(game_id = response.games.appid,
             game_name = response.games.name,
             playtime = response.games.playtime_forever) %>%
      filter(playtime != 0)

playtime$player_id <- as.character(playtime$player_id)
```

F) R code for web scraping metadata from Steam's website

R code for web scraping the Steam tags

```
# Loading required packages
library("rvest")

# Setting up an empty data frame to store the metadata
tags <- data.frame(game_id = character(),
                   tags = character(),
                   stringsAsFactors = FALSE)

# Scraping the web for each unique game title
for (x in games$game_id){

      url <- paste("https://store.steampowered.com/app/", x, "/",
                   sep = "")

      webpage <- read_html(url)
      rank_data_html <- html_nodes(webpage, ".app_tag")
      rank_data <- html_text(rank_data_html)

      rank_data <- gsub("\t", "", rank_data)
      rank_data <- gsub("\r\n", "", rank_data)

      if (identical(rank_data, character(0)) == TRUE) next

      temp <- data.frame("game_id" = x, "tags" = rank_data,
                         stringsAsFactors = FALSE)

      tags <- bind_rows(tags, temp)}
```

G) Overview playtime data

Table G.1

Playtime data after querying the Steam API

	[1]	[2]	[3]	[4]
	user_id	**game_id**	**game_name**	**playtime**
	<chr>	<int>	<chr>	<int>
[1]	user_1	440	Team Fortress 2	72
[2]	user_1	550	Left 4 Dead 2	451
…	…	…	…	…
[2790463]	user_19730	570	Dota 2	5256
[2790464]	user_19730	205790	Dota 2 Test	0

Note. Number of unique users = 19,730. Number of unique game IDs = 23,630. Average cum. playtime per user ≈ 157,738. Average cum. playtime per game ID = 131,704.

Playtime in minutes.

Table G.2

Playtime data after removing unpopular/discontinued games and inactive players

	[1]	[2]	[3]	[4]
	user_id	**game_id**	**game_name**	**playtime**
	<chr>	<int>	<chr>	<int>
[1]	user_1	10	Counter-Strike	311
[2]	user_1	240	Counter-Strike: Source	168
…	…	…	…	…
[1079874]	user_12976	784950	Conflict of Nations: World War 3	1
[1079875]	user_12976	813820	Realm Royale	144

Note. Number of unique players = 12,976. Number of unique games = 7,105. Average cum. playtime per user ≈ 199,054. Average cum. playtime per game ≈ 363,536.

Playtime in minutes.

H) Overview metadata

Table H.1

Metadata after web scraping the Steam website

	[1]	[2]	[3]
	game_id	**game_name**	**tags**
	<int>	<chr>	<chr>
[1]	10	Counter-Strike	1980's
[2]	10	Counter-Strike	1990's
…	…	…	…
[87317]	1020470	Evoland Legendary Edition	Indie
[87318]	1020470	Evoland Legendary Edition	RPG

Note. Number of unique games = 7,105. Number of unique tags = 368.

Average number of tags per game ≈ 12.

I) Overview individual tags

Table I.1

Tags (1)

	tag	n()	%		tag	n()	%
	<chr>	<int>	<chr>		<chr>	<int>	<chr>
[1]	1980s	59	0,8%	[34]	Base Building	145	2,0%
[2]	1990's	97	1,4%	[35]	Baseball	6	0,1%
[3]	2.5D	77	1,1%	[36]	Based On A Novel	20	0,3%
[4]	2D	1636	23,0%	[37]	Basketball	7	0,1%
[5]	2D Fighter	71	1,0%	[38]	Batman	19	0,3%
[6]	360 Video	1	0,0%	[39]	Battle Royale	38	0,5%
[7]	3D	38	0,5%	[40]	Beat 'em up	143	2,0%
[8]	3D Platformer	115	1,6%	[41]	Beautiful	65	0,9%
[9]	3D Vision	43	0,6%	[42]	Benchmark	7	0,1%
[10]	4 Player Local	180	2,5%	[43]	Bikes	4	0,1%
[11]	4X	88	1,2%	[44]	Blood	66	0,9%
[12]	6DOF	16	0,2%	[45]	Board Game	108	1,5%
[13]	ATV	3	0,0%	[46]	Bowling	4	0,1%
[14]	Abstract	65	0,9%	[47]	Building	312	4,4%
[15]	Action	4119	58,0%	[48]	Bullet Hell	199	2,8%
[16]	Action RPG	271	3,8%	[49]	Bullet Time	32	0,5%
[17]	Action-Adventure	141	2,0%	[50]	CRPG	52	0,7%
[18]	Addictive	97	1,4%	[51]	Capitalism	24	0,3%
[19]	Adventure	3718	52,3%	[52]	Card Game	137	1,9%
[20]	Agriculture	37	0,5%	[53]	Cartoon	61	0,9%
[21]	Aliens	140	2,0%	[54]	Cartoony	102	1,4%
[22]	Alternate History	49	0,7%	[55]	Casual	2401	33,8%
[23]	America	32	0,5%	[56]	Cats	28	0,4%
[24]	Animation & Modeling	22	0,3%	[57]	Character Action Game	30	0,4%
[25]	Anime	870	12,2%	[58]	Character Customization	202	2,8%
[26]	Arcade	639	9,0%	[59]	Chess	13	0,2%
[27]	Arena Shooter	68	1,0%	[60]	Choices Matter	198	2,8%
[28]	Artificial Intelligence	6	0,1%	[61]	Choose Your Own Adventure	82	1,2%
[29]	Assassin	30	0,4%	[62]	Cinematic	21	0,3%
[30]	Asynchronous Multiplayer	20	0,3%	[63]	City Builder	125	1,8%
[31]	Atmospheric	1538	21,6%	[64]	Class-Based	27	0,4%
[32]	Audio Production	10	0,1%	[65]	Classic	570	8,0%
[33]	BMX	1	0,0%	[66]	Clicker	82	1,2%

Note. Number of unique tags = 368. Average number of tags per game ≈ 12.

Table I.2

Tags (2)

	tag	n()	%		tag	n()	%
	<chr>	<int>	<chr>		<chr>	<int>	<chr>
[67]	Co-op	1076	15,1%	[100]	Driving	118	1,7%
[68]	Co-op Campaign	18	0,3%	[101]	Dungeon Crawler	184	2,6%
[69]	Cold War	40	0,6%	[102]	Dungeons & Dragons	12	0,2%
[70]	Colorful	229	3,2%	[103]	Dynamic Narration	14	0,2%
[71]	Comedy	689	9,7%	[104]	Dystopian	64	0,9%
[72]	Comic Book	43	0,6%	[105]	Early Access	1440	20,3%
[73]	Competitive	192	2,7%	[106]	Economy	136	1,9%
[74]	Conspiracy	21	0,3%	[107]	Education	84	1,2%
[75]	Controller	398	5,6%	[108]	Emotional	23	0,3%
[76]	Conversation	5	0,1%	[109]	Epic	40	0,6%
[77]	Crafting	272	3,8%	[110]	Episodic	72	1,0%
[78]	Crime	58	0,8%	[111]	Experience	8	0,1%
[79]	Crowdfunded	30	0,4%	[112]	Experimental	74	1,0%
[80]	Cult Classic	92	1,3%	[113]	Exploration	481	6,8%
[81]	Cute	496	7,0%	[114]	FMV	36	0,5%
[82]	Cyberpunk	148	2,1%	[115]	FPS	727	10,2%
[83]	Cycling	4	0,1%	[116]	Faith	6	0,1%
[84]	Dark	264	3,7%	[117]	Family Friendly	498	7,0%
[85]	Dark Comedy	29	0,4%	[118]	Fantasy	945	13,3%
[86]	Dark Fantasy	135	1,9%	[119]	Fast-Paced	265	3,7%
[87]	Dark Humor	109	1,5%	[120]	Female Protagonist	878	12,4%
[88]	Dating Sim	148	2,1%	[121]	Fighting	177	2,5%
[89]	Demons	55	0,8%	[122]	First-Person	950	13,4%
[90]	Design & Illustration	25	0,4%	[123]	Fishing	24	0,3%
[91]	Destruction	95	1,3%	[124]	Flight	86	1,2%
[92]	Detective	109	1,5%	[125]	Football	30	0,4%
[93]	Difficult	1036	14,6%	[126]	Foreign	1	0,0%
[94]	Dinosaurs	36	0,5%	[127]	Free to Play	758	10,7%
[95]	Diplomacy	17	0,2%	[128]	Funny	948	13,3%
[96]	Documentary	8	0,1%	[129]	Futuristic	133	1,9%
[97]	Dog	14	0,2%	[130]	Gambling	10	0,1%
[98]	Dragons	48	0,7%	[131]	Game Development	28	0,4%
[99]	Drama	53	0,7%	[132]	GameMaker	55	0,8%

Note. Number of unique tags = 368. Average number of tags per game ≈ 12.

Table I.3

Tags (3)

	tag	n()	%		tag	n()	%
	<chr>	<int>	<chr>		<chr>	<int>	<chr>
[133]	Games Workshop	36	0,5%	[166]	Lemmings	6	0,1%
[134]	Gaming	7	0,1%	[167]	Level Editor	119	1,7%
[135]	God Game	47	0,7%	[168]	Linear	38	0,5%
[136]	Golf	9	0,1%	[169]	Local Co-Op	421	5,9%
[137]	Gore	557	7,8%	[170]	Local Multiplayer	358	5,0%
[138]	Gothic	42	0,6%	[171]	Logic	15	0,2%
[139]	Grand Strategy	97	1,4%	[172]	Loot	94	1,3%
[140]	Great Soundtrack	1930	27,2%	[173]	Lore-Rich	30	0,4%
[141]	Grid-Based Movement	28	0,4%	[174]	Lovecraftian	71	1,0%
[142]	Gun Customization	17	0,2%	[175]	MMORPG	158	2,2%
[143]	Hack and Slash	256	3,6%	[176]	MOBA	63	0,9%
[144]	Hacking	34	0,5%	[177]	Magic	137	1,9%
[145]	Hand-drawn	105	1,5%	[178]	Management	270	3,8%
[146]	Heist	24	0,3%	[179]	Mars	17	0,2%
[147]	Hex Grid	64	0,9%	[180]	Martial Arts	21	0,3%
[148]	Hidden Object	136	1,9%	[181]	Massively Multiplayer	475	6,7%
[149]	Historical	265	3,7%	[182]	Masterpiece	166	2,3%
[150]	Hockey	3	0,0%	[183]	Match 3	77	1,1%
[151]	Horror	879	12,4%	[184]	Mature	202	2,8%
[152]	Horses	8	0,1%	[185]	Mechs	75	1,1%
[153]	Hunting	29	0,4%	[186]	Medieval	163	2,3%
[154]	Illuminati	86	1,2%	[187]	Memes	310	4,4%
[155]	Indie	5235	73,7%	[188]	Metroidvania	162	2,3%
[156]	Intentionally Awkward Control	7	0,1%	[189]	Military	158	2,2%
[157]	Interactive Fiction	56	0,8%	[190]	Mini Golf	5	0,1%
[158]	Inventory Management	24	0,3%	[191]	Minigames	10	0,1%
[159]	Investigation	5	0,1%	[192]	Minimalist	146	2,1%
[160]	Isometric	221	3,1%	[193]	Mining	15	0,2%
[161]	JRPG	212	3,0%	[194]	Mod	66	0,9%
[162]	Jet	1	0,0%	[195]	Moddable	208	2,9%
[163]	Kickstarter	113	1,6%	[196]	Modern	13	0,2%
[164]	LEGO	39	0,5%	[197]	Motocross	6	0,1%
[165]	Lara Croft	3	0,0%	[198]	Motorbike	7	0,1%

Note. Number of unique tags = 368. Average number of tags per game ≈ 12.

Table I.4

Tags (4)

	tag	n()	%		tag	n()	%
	<chr>	<int>	<chr>		<chr>	<int>	<chr>
[199]	Mouse only	49	0,7%	[232]	Point & Click	483	6,8%
[200]	Movie	14	0,2%	[233]	Political	39	0,5%
[201]	Multiplayer	2058	29,0%	[234]	Politics	38	0,5%
[202]	Multiple Endings	128	1,8%	[235]	Pool	5	0,1%
[203]	Music	126	1,8%	[236]	Post-apocalyptic	179	2,5%
[204]	Music-Based Procedural Generatic	21	0,3%	[237]	Procedural Generation	186	2,6%
[205]	Mystery	318	4,5%	[238]	Programming	25	0,4%
[206]	Mystery Dungeon	7	0,1%	[239]	Psychedelic	37	0,5%
[207]	Mythology	43	0,6%	[240]	Psychological	53	0,7%
[208]	NSFW	19	0,3%	[241]	Psychological Horror	389	5,5%
[209]	Narration	44	0,6%	[242]	Puzzle	1208	17,0%
[210]	Naval	48	0,7%	[243]	Puzzle-Platformer	231	3,3%
[211]	Ninja	52	0,7%	[244]	PvE	55	0,8%
[212]	Noir	53	0,7%	[245]	PvP	220	3,1%
[213]	Nonlinear	18	0,3%	[246]	Quick-Time Events	20	0,3%
[214]	Nudity	469	6,6%	[247]	RPG	1831	25,8%
[215]	Offroad	26	0,4%	[248]	RPGMaker	209	2,9%
[216]	Old School	87	1,2%	[249]	RTS	280	3,9%
[217]	On-Rails Shooter	17	0,2%	[250]	Racing	294	4,1%
[218]	Online Co-Op	376	5,3%	[251]	Real Time Tactics	56	0,8%
[219]	Open World	1088	15,3%	[252]	Real-Time	68	1,0%
[220]	Otome	49	0,7%	[253]	Real-Time with Pause	97	1,4%
[221]	Parkour	115	1,6%	[254]	Realistic	227	3,2%
[222]	Parody	44	0,6%	[255]	Relaxing	230	3,2%
[223]	Party-Based RPG	58	0,8%	[256]	Remake	99	1,4%
[224]	Perma Death	118	1,7%	[257]	Replay Value	328	4,6%
[225]	Philisophical	20	0,3%	[258]	Resource Management	118	1,7%
[226]	Photo Editing	2	0,0%	[259]	Retro	654	9,2%
[227]	Physics	269	3,8%	[260]	Rhythm	72	1,0%
[228]	Pinball	8	0,1%	[261]	Robots	160	2,3%
[229]	Pirates	66	0,9%	[262]	Rogue-like	358	5,0%
[230]	Pixel Graphics	949	13,4%	[263]	Rogue-lite	226	3,2%
[231]	Platformer	887	12,5%	[264]	Romance	132	1,9%

Note. Number of unique tags = 368. Average number of tags per game ≈ 12.

Table I.5

Tags (5)

	tag	n()	%		tag	n()	%
	<chr>	<int>	<chr>		<chr>	<int>	<chr>
[265]	Rome	15	0,2%	[298]	Split Screen	80	1,1%
[266]	Runner	40	0,6%	[299]	Sports	292	4,1%
[267]	Sailing	15	0,2%	[300]	Star Wars	24	0,3%
[268]	Sandbox	647	9,1%	[301]	Stealth	228	3,2%
[269]	Satire	31	0,4%	[302]	Steam Machine	5	0,1%
[270]	Sci-fi	958	13,5%	[303]	Steampunk	120	1,7%
[271]	Science	35	0,5%	[304]	Story Rich	1151	16,2%
[272]	Score Attack	46	0,6%	[305]	Strategy	2025	28,5%
[273]	Sequel	9	0,1%	[306]	Strategy RPG	47	0,7%
[274]	Sexual Content	312	4,4%	[307]	Stylized	93	1,3%
[275]	Shoot 'Em Up	290	4,1%	[308]	Submarine	7	0,1%
[276]	Shooter	960	13,5%	[309]	Superhero	56	0,8%
[277]	Short	292	4,1%	[310]	Supernatural	33	0,5%
[278]	Side Scroller	322	4,5%	[311]	Surreal	143	2,0%
[279]	Silent Protagonist	31	0,4%	[312]	Survival	691	9,7%
[280]	Simulation	1563	22,0%	[313]	Survival Horror	228	3,2%
[281]	Singleplayer	3883	54,7%	[314]	Swordplay	38	0,5%
[282]	Skateboarding	3	0,0%	[315]	Tactical	393	5,5%
[283]	Skating	4	0,1%	[316]	Tactical RPG	70	1,0%
[284]	Skiing	2	0,0%	[317]	Tanks	48	0,7%
[285]	Sniper	16	0,2%	[318]	Team-Based	115	1,6%
[286]	Snow	2	0,0%	[319]	Tennis	2	0,0%
[287]	Snowboarding	2	0,0%	[320]	Text-Based	22	0,3%
[288]	Soccer	30	0,4%	[321]	Third Person	525	7,4%
[289]	Software	19	0,3%	[322]	Third-Person Shooter	175	2,5%
[290]	Software Training	9	0,1%	[323]	Thriller	32	0,5%
[291]	Sokoban	10	0,1%	[324]	Time Attack	33	0,5%
[292]	Souls-like	25	0,4%	[325]	Time Management	24	0,3%
[293]	Soundtrack	31	0,4%	[326]	Time Manipulation	26	0,4%
[294]	Space	449	6,3%	[327]	Time Travel	45	0,6%
[295]	Space Sim	63	0,9%	[328]	Top-Down	229	3,2%
[296]	Spectacle fighter	21	0,3%	[329]	Top-Down Shooter	121	1,7%
[297]	Spelling	5	0,1%	[330]	Touch-Friendly	91	1,3%

Note. Number of unique tags = 368. Average number of tags per game ≈ 12.

Table I.6

Tags (6)

	tag	n()	%		tag	n()	%
	<chr>	<int>	<chr>		<chr>	<int>	<chr>
[331]	Tower Defense	184	2,6%	[364]	World War I	28	0,4%
[332]	TrackIR	21	0,3%	[365]	World War II	142	2,0%
[333]	Trading	46	0,6%	[366]	Wrestling	4	0,1%
[334]	Trading Card Game	59	0,8%	[367]	Zombies	337	4,7%
[335]	Trains	28	0,4%	[368]	e-sports	54	0,8%
[336]	Transhumanism	4	0,1%				
[337]	Turn-Based	544	7,7%				
[338]	Turn-Based Combat	161	2,3%				
[339]	Turn-Based Strategy	314	4,4%				
[340]	Turn-Based Tactics	144	2,0%				
[341]	Tutorial	6	0,1%				
[342]	Twin Stick Shooter	117	1,6%				
[343]	Typing	13	0,2%				
[344]	Underground	6	0,1%				
[345]	Underwater	41	0,6%				
[346]	Unforgiving	3	0,0%				
[347]	Utilities	29	0,4%				
[348]	VR	248	3,5%				
[349]	Vampire	27	0,4%				
[350]	Video Production	13	0,2%				
[351]	Villain Protagonist	43	0,6%				
[352]	Violent	429	6,0%				
[353]	Visual Novel	413	5,8%				
[354]	Voice Control	4	0,1%				
[355]	Voxel	67	0,9%				
[356]	Walking Simulator	217	3,1%				
[357]	War	247	3,5%				
[358]	Wargame	43	0,6%				
[359]	Warhammer 40K	21	0,3%				
[360]	Web Publishing	12	0,2%				
[361]	Werewolves	14	0,2%				
[362]	Western	37	0,5%				
[363]	Word Game	12	0,2%				

Note. Number of unique tags = 368. Average number of tags per game ≈ 12.

J) Overview reshaped data

Table J.1

Playtime data after reshaping the data for archetypal analysis

	[1]	[2]	[3]	[4-12977]	[12978]
	game_id	**game_name**	**user_1**	**…**	**user_12976**
	<int>	<chr>	<int>	<int>	<int>
[1]	10	Counter-Strike	0	…	0
[2]	20	Team Fortress Classic	0	…	0
…	…	…	…	…	…
[7104]	1007350	Mirror Maker	0	…	0
[7105]	1020470	Evoland Legendary Edition	0	…	0

Note. Number of unique players = 12,976. Number of unique games = 7,105.

Playtime in minutes.

Table J.2

Metadata after reshaping the data for archetypal analysis

	[1]	[3]	[4-7105]	[7106]
	tag	**Counter-Strike**	**…**	**Evoland Legendary Edition**
	<chr>	<int>	<int>	<int>
[1]	1980’s	1	…	0
[2]	1990’s	1	…	0
…	…	…	…	…
[367]	Wrestling	0	…	0
[368]	Zombies	0	…	0

Note. Number of unique games = 7,105. Number of unique tags = 368.

Average number of tags per game ≈ 12.

K) Overview archetypes

Table K.1

Playtime data containing the k different archetypes (k = 9)

	[1]	[2]	[3-9]	[10]
	user_id	**archetype_1**	**…**	**archetype_9**
	<chr>	<dbl>	<dbl>	<dbl>
[1]	user_1	34.3537339	…	2.724776520
[2]	user_2	86.9476577	…	159.769842515
…	…	…	…	…
[12975]	user_12975	1.1895905	…	0.008965055
[12976]	user_12976	0.3011402	…	1.461288950

Note. RSS = 0.17.

Table K.2

Metadata containing the k different archetypes (k = 9)

	[1]	[2]	[3-9]	[10]
	game_name	**archetype_1**	**…**	**archetype_9**
	<chr>	<dbl>	<dbl>	<dbl>
[1]	Counter-Strike	0.028584448142	…	-0.015887284
[2]	Team Fortress Classic	0.028606176457	…	-0.015887284
…	…	…	…	…
[7104]	Mirror Maker	-0.00086647361	…	0.738738061
[7105]	Evoland Legendary Edition	-0.00072206134	…	-0.003971821

Note. RSS = 0.71.

L) R code for calculating cosine similarities

R code for calculating cosine similarity between archetype vectors

```
# Loading required packages
library("lsa")

# Setting up an empty data frame to store cosine distances
cosine_matrix <- data.frame(matrix(nrow = nrow(archetypes),
                                   ncol = nrow(archetypes)))

# Calculating the cosine similarity between each player
for (x in 1:nrow(archetypes)){
      for (y in 1:nrow(archetypes)){
            if (is.na(cosine_matrix[y, x]) != TRUE){
                  cosine_matrix[x, y] <- cosine_matrix[y, x]
                  next}

            vec1 <- c(archetypes[x, 2], archetypes[x, 3],
                      archetypes[x, 4], archetypes[x, 5],
                      archetypes[x, 6], archetypes[x, 7],
                      archetypes[x, 8], archetypes[x, 9],
                      archetypes[x, 10])

            vec2 <- c(archetypes[y, 2], archetypes[y, 3],
                      archetypes[y, 4], archetypes[y, 5],
                      archetypes[y, 6], archetypes[y, 7],
                      archetypes[y, 8], archetypes[y, 9],
                      archetypes[y, 10])

            cosine_matrix[x, y] <- cosine(vec1, vec2)}}

# Renaming rows and columns
colnames(cosine_matrix) <- archetypes[ , 1]
rownames(cosine_matrix) <- archetypes[ , 1]
```

M) Overview cosine matrices

Table M.1

Cosine matrix of similarity between players

		[1]	[2-12975]	[12976]
		user_1	**…**	**user_12976**
		<dbl>	<dbl>	<dbl>
[1]	**user_1**	1	…	0.002406274
[2]	**user_2**	0.999618157	…	0.008753756
…	**…**	…	…	…
[12975]	**user_12975**	0.055884537	…	0.012219261
[12976]	**user_12976**	0.002406274	…	1

Note.

Table M.2

Cosine matrix of similarity between games

		[1]	[2-7104]	[7105]
		Counter-Strike	**…**	**Evoland Legendary Edition**
		<dbl>	<dbl>	<dbl>
[1]	**Counter-Strike**	1	…	0.4178244612
[2]	**Team Fortress Classic**	0.774914817	…	0.76514726
…	**…**	…	…	…
[7104]	**Mirror Maker**	0.0164128512	…	0.381070158
[7105]	**Evoland Legendary Edition**	0.4178244612	…	1

Note.

N) Overview summary data

Table N.1

Overview of 5 most similar players and top 3 games per player

	[1]	[2]	[3-5]	[6]	[7]	[8-10]	[11]
	user_id	**sim_1**	**…**	**sim_5**	**cos_1**	**…**	**cos_5**
	<chr>	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>
[1]	user_1	user_12621	…	user_11931	1.0000	…	0.9999
[2]	user_2	user_3509	…	user_9499	0.9999	…	0.9999
…	…	…	…	…	…	…	…
[12975]	user_12975	user_12605	…	user_12849	0.9999	…	0.9992
[12976]	user_12976	user_10851	…	user_6827	1.0000	…	0.9999

Note.

Table N.2

Overview of 5 most similar games per game (top 250 only)

	[1]	[2]	[3-5]	[6]	[7]	[8-10]	[11]
	game_id	**sim_1**	**…**	**sim_5**	**cos_1**	**…**	**cos_5**
	<int>	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>
[1]	10	299360	…	34330	0.9989	…	0.9693
[2]	20	17520	…	320	0.9975	…	0.9570
…	…	…	…	…	…	…	…
[7104]	1007350	438100	…	698780	0.9046	…	0.6979
[7105]	1020470	252150	…	555570	0.9728	…	0.9420

Note.

O) R code for predicting playtime

R code for predicting playtime with the collaborative filtering method

```
# For all 1,010,000 predictions that have to be made…
for (x in 1:nrow(predictions_cf)){

      # Store the user ID and game ID that the prediction concerns.
      temp_user_id <- predictions_cf[i, 2]
      temp_game_id <- predictions_cf[i, 3]

      temp <- final_data_cf %>%
            filter(user_id == temp_user_id)

      # Store the 5 users that are most similar to the current user.
      user_1 <- temp[1, 2]
      user_2 <- temp[1, 3]
      user_3 <- temp[1, 4]
      user_4 <- temp[1, 5]
      user_5 <- temp[1, 6]

      # Store the similarities between these 5 and the current user.
      cos_1 <- temp[1, 7]
      cos_2 <- temp[1, 8]
      cos_3 <- temp[1, 9]
      cos_4 <- temp[1, 10]
      cos_5 <- temp[1, 11]

      # Store the 5 most similar users' playtime on the current game.
      playtime_1 <- filter(playtime, player_id == user_1,
                                     game_id == temp_game_id)[1, 4]
      playtime_2 <- filter(playtime, player_id == user_2,
                                     game_id == temp_game_id)[1, 4]
      playtime_3 <- filter(playtime, player_id == user_3,
                                     game_id == temp_game_id)[1, 4]
      playtime_4 <- filter(playtime, player_id == user_4,
                                     game_id == temp_game_id)[1, 4]
      playtime_5 <- filter(playtime, player_id == user_5,
                                     game_id == temp_game_id)[1, 4]

      # If a user hasn't played the game, turn the NA into a zero.
      if (is.na(playtime_1) == TRUE){playtime_1 <- 0}
      if (is.na(playtime_2) == TRUE){playtime_2 <- 0}
      if (is.na(playtime_3) == TRUE){playtime_3 <- 0}
      if (is.na(playtime_4) == TRUE){playtime_4 <- 0}
      if (is.na(playtime_5) == TRUE){playtime_5 <- 0}

      # Calculate the predicted playtime.
      total <- round(((playtime_1 * cos_1) + (playtime_2 * cos_2) +
                      (playtime_3 * cos_3) + (playtime_4 * cos_4) +
                      (playtime_5 * cos_5)) /
                      (cos_1 + cos_2 + cos_3 + cos_4 + cos_5))

      # Add the prediction to the dataframe.
      predictions_cf[i, 6] <- total}
```

R code for predicting playtime with the content-based method

```
# For all x predictions that have to be made…
for (x in 1:nrow(predictions_cbf)){

      # Store the user ID and game ID that the prediction concerns.
      temp_user_id <- predictions_cbf[i, 2]
      temp_game_id <- predictions_cbf[i, 3]

      temp <- final_data_cbf %>%
            filter(game_id == temp_game_id)

      # Store the 5 games that are most similar to the current game.
      game_1 <- temp[1, 2]
      game_2 <- temp[1, 3]
      game_3 <- temp[1, 4]
      game_4 <- temp[1, 5]
      game_5 <- temp[1, 6]

      # Store the similarities between these 5 and the current game.
      cos_1 <- temp[1, 7]
      cos_2 <- temp[1, 8]
      cos_3 <- temp[1, 9]
      cos_4 <- temp[1, 10]
      cos_5 <- temp[1, 11]

      # Store the playtimes on the 5 most similar games.
      playtime_1 <- filter(playtime, player_id == temp_player_id,
                                     game_id == game_1)[1, 4]
      playtime_2 <- filter(playtime, player_id == temp_player_id,
                                     game_id == game_2)[1, 4]
      playtime_3 <- filter(playtime, player_id == temp_player_id,
                                     game_id == game_3)[1, 4]
      playtime_4 <- filter(playtime, player_id == temp_player_id,
                                     game_id == game_4)[1, 4]
      playtime_5 <- filter(playtime, player_id == temp_player_id,
                                     game_id == game_5)[1, 4]

      # If a user hasn't played the game, turn the NA into a zero.
      if (is.na(playtime_1) == TRUE){playtime_1 <- 0}
      if (is.na(playtime_2) == TRUE){playtime_2 <- 0}
      if (is.na(playtime_3) == TRUE){playtime_3 <- 0}
      if (is.na(playtime_4) == TRUE){playtime_4 <- 0}
      if (is.na(playtime_5) == TRUE){playtime_5 <- 0}

      # Calculate the predicted playtime.
      total <- round(((playtime_1 * cos_1) + (playtime_2 * cos_2) +
                      (playtime_3 * cos_3) + (playtime_4 * cos_4) +
                      (playtime_5 * cos_5)) /
                      (cos_1 + cos_2 + cos_3 + cos_4 + cos_5))

      # Add the prediction to the dataframe.
      predictions_cbf[i, 6] <- total}
```

P) R code for evaluating playtime predictions

R code for evaluating playtime predictions

```
# Loading required packages
library("dplyr")
library("magrittr")

# Filter out target games and put them in a separate data frame.
evaluations <- filter(predictions, target == 1)

# Add a column to this data frame, indicating whether the target was
predicted correctly (set 0 as default).
evaluations$correct <- 0

# Add another column, indicating the value for L that will be set to
evaluate these predictions (set 5 as default).
evaluations$l <- 5

# Make a temporary copy of the target predictions.
evaluations_copy <- evaluations

# Repeat the data 16 times for all different values of L.
for (x in 6:20){
      temp <- evaluations_copy
      temp$l <- x
      evaluations <- rbind(evaluations, temp)}

# For each target prediction…
for (p in 1:nrow(evaluations)){
      # Make a temporary copy of the data containing all predictions.
      temp <- predictions %>%
            # Only include user & iteration of current prediction.
            filter(player_id == evaluations[p, 3],
                   iteration == evaluations[p, 2) %>%
            # Rank the games based on their predicted playtime.
            arrange(desc(pred_playtime))

      # Remove all predictions except for the top-L.
      temp <- temp[1:evaluations[p, 7], ]

      # For each prediction in the top-L ranking…
      for (o in 1:nrow(temp)){
            # If the target game occurs in the top-L ranking…
            if ((temp[o, 4] == 1) == TRUE){
                        # Classify the prediction as correct.
                        evaluations[p, 7] <- 1}}}
```

Q) R code for hybridizing playtime predictions

R code for hybridizing playtime predictions (1)

```
# Loading required packages
library("dplyr")
library("magrittr")
library("tidyr")

# Group the recall values by the users and the two different methods.
evaluations_grouped_a <- evaluations %>%
      group_by(method, user_id) %>%
      summarise(recall = mean(correct))

# Group the recall values by the games and the two different methods.
evaluations_grouped_b <- evaluations %>%
      group_by(method, game_id) %>%
      summarise(recall = mean(correct))

evaluations_grouped_a <- spread(evaluations_grouped_a, method, recall)
evaluations_grouped_a$cbf_win <- 0
evaluations_grouped_a$cf_win <- 0

evaluations_grouped_b <- spread(evaluations_grouped_b, method, recall)
evaluations_grouped_b$cbf_win <- 0
evaluations_grouped_b$cf_win <- 0

for (i in 1:nrow(evaluations_grouped_a){
      if (evaluations_grouped_a[i, 2] > evaluations_grouped_a[i, 3]){
            evaluations_grouped_a[i, 4] <- 1}}

for (i in 1:nrow(evaluations_grouped_a){
      if (evaluations_grouped_a[i, 3] > evaluations_grouped_a[i, 2]){
            evaluations_grouped_a[i, 5] <- 1}}

for (i in 1:nrow(evaluations_grouped_b){
      if (evaluations_grouped_b[i, 2] > evaluations_grouped_b[i, 3]){
            evaluations_grouped_b[i, 4] <- 1}}

for (i in 1:nrow(evaluations_grouped_b){
      if (evaluations_grouped_b[i, 3] > evaluations_grouped_b[i, 2]){
            evaluations_grouped_b[i, 5] <- 1}}
```

R code for hybridizing playtime predictions (2)

```
# Hybrid model A (users).
hybrid_a <- evaluations
hybrid_a$keep <- "drop"

for (i in 1:nrow(hybrid_a)){
      user <- hybrid_a[i, 3]

      win <- evaluations_grouped_a %>%
            filter(user_id == user)

      if (win[1, 4] == 1){hybrid_a$keep[hybrid_a$user_id == user &
                           hybrid_a$method == "cbf"] <- "keep"}

      if (win[1, 5] == 1){hybrid_a$keep[hybrid_a$user_id == user &
                           hybrid_a$method == "cf"] <- "keep"}

      if (win[1, 2] == win[1, 3]){hybrid_a$keep[hybrid_a$user_id == user
                                  & hybrid_a$method == "cf"] <- "keep"}}

hybrid_a <- filter(hybrid_a, keep != "drop")

# Hybrid model B (games).
hybrid_b <- evaluations
hybrid_b$keep <- "drop"

for (i in 1:nrow(hybrid_b)){
      game <- as.numeric(hybrid_b[i, 4])

      win <- evaluations_grouped_b %>%
            filter(game_id == game)

      if (win[1, 4] == 1){hybrid_b$keep[hybrid_b$game_id == game &
                           hybrid_b$method == "cbf"] <- "keep"}

      if (win[1, 5] == 1){hybrid_b$keep[hybrid_b$game_id == game &
                           hybrid_b$method == "cf"] <- "keep"}

      if (win[1, 2] == win[1, 3]){hybrid_b$keep[hybrid_b$game_id == game
                                  & hybrid_b$method == "cf"] <- "keep"}}

hybrid_b <- filter(hybrid_b, keep != "drop")
```

R) Overview playtime predictions

Table R.1

Dataframe with predicted playtimes for the CF and the CBF

	[1]	[2]	[3]	[4]	[5]	[6]
	method	**iteration**	**user_id**	**game**	**target**	**pred_playtime**
	<chr>	<int>	<chr>	<chr>	<int>	<int>
[1]	cf	1	user_1	1	0	0
[2]	cf	1	user_1	2	0	0
…	…	…	…	…	…	…
[2019999]	cbf	100	user_100	100	0	0
[2020000]	cbf	100	user_100	101	1	0

Note. 2 methods * 100 iterations * 100 users * 101 games.

Table R.2

Dataframe with predicted playtimes for hybrid model A and hybrid model B

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
	model	**iteration**	**user_id**	**game**	**target**	**method**	**pred_playtime**
	<chr>	<int>	<chr>	<chr>	<int>	<chr>	<int>
[1]	a	1	user_1	1	0	cf	0
[2]	a	1	user_1	2	0	cf	0
…	…	…	…	…	…	…	…
[2019999]	b	100	user_100	100	0	cbf	0
[2020000]	b	100	user_100	101	1	cf	25022

Note. 2 models * 100 iterations * 100 users * 101 games.

S) Overview prediction evaluations

Table S.1

Dataframe with prediction evaluations for the CF and the CBF

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
	method	**iteration**	**user_id**	**game**	**L**	**pred_playtime**	**correct**
	<chr>	<int>	<chr>	<chr>	<int>	<int>	<int>
[1]	cf	1	user_1	578080	5	44126	1
[2]	cf	1	user_1	578080	6	44126	1
…	…	…	…	…	…	…	…
[319999]	cbf	100	user_100	730	19	0	0
[320000]	cbf	100	user_100	730	20	0	0

Note. 2 methods * 100 iterations * 100 users * 16 values for *L*.

Table S.2

Dataframe with prediction evaluations for hybrid model A and hybrid model B

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
	model	**iteration**	**user_id**	**game**	**L**	**pred_playtime**	**correct**
	<chr>	<int>	<chr>	<chr>	<int>	<int>	<int>
[1]	a	1	user_1	578080	5	44126	1
[2]	a	1	user_1	578080	6	44126	1
…	…	…	…	…	…	…	…
[319999]	b	100	user_100	730	19	25022	1
[320000]	b	100	user_100	730	20	25022	1

Note. 2 models * 100 iterations * 100 users * 16 values for *L*.

T) Overview results T-test

Overview results T-test

```
Paired t-test

t = -20.124, df = 99, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
99.9999 percent confidence interval:
 -0.04630918 -0.02724082
sample estimates:
mean of the differences
              -0.036775
```

Overview Brown-Forsythe test for homogeneity of variance

```
Levene's Test for Homogeneity of Variance (center = median)

       Df F value Pr(>F)
group   1  1.0059 0.3171
      198
```