Dr. med. univ. Dietmar C. Maurer, BSc

Evaluating the Suitability of Recommender Approaches for Narrative-Driven Video Game Recommendations

Master's Thesis

to achieve the university degree of
Master of Science

submitted to
Graz University of Technology

Supervisor
Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Co-Supervisor
Dipl.-Ing. Lukas Eberhard, BSc

Institute of Interactive Systems and Data Science

Faculty of Computer Science and Biomedical Engineering

Graz, February 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Diplomarbeit identisch.

Datum	Unterschrift

Abstract

Since the dawn of the internet, the available information has been growing steadily. Hence, algorithms which help humans to filter the information are needed. Recommender Systems became widely utilized for this task over the last decade. Today, they are used almost ubiquitous. The general idea of RS is to identify items which could be of potential interest to a user and present them as a list. RS rely heavily on the provided information of users, items and their ratings. Generally, the more the better. However, there are cases in which only sparse information is available. This thesis is about the implementation of a recommender engine which tackles that problem. To illustrate the information sparsity, requests from the subreddit (a section within the online news aggregation service reddit) *r/gamingsuggestions* were collected. In this subreddit, users can ask for video game recommendations based on information they are willing to share. A typical request only includes a few game titles and maybe some further constraints, such as the preferred support of controllers. To recommend video games, three state-of-the-art recommender algorithms (i.e., collaborative filtering, matrix factorization and term frequency-inverse document frequency) were implemented. In order to improve the results of the algorithms, post-filtering techniques were applied. By comparing the recommendations of the algorithms with the suggestions of the reddit community, typical scores, such as precision, recall and F1-score were computed. To further investigate the recommendations, a qualitative evaluation was performed. Humans were asked to rate the recommendations on the 3-way-scale, whether they were good, ok, or bad.

Kurzfassung

Seit Anbeginn des Internets stieg die Menge an verfügbarer Information stetig an, sodass Algorithmen gefunden werden mussten, welche den Menschen halfen, diese Informationen zu filtern. Für diese Aufgabe wurden in den letzten zehn Jahren immer häufiger Recommender Systeme eingesetzt, weswegen sie heutzutage beinahe überall anzutreffen sind. Das Hauptaufgabe von RS ist es, Entitäten zu identifizieren, welche momentan von pontentiellem Interesse für die Benutzer sind und diese in einer geordneten Liste darzustellen. Dabei sind RS von den Informationen der Benutzer, den Entitäten sowie deren Bewertung abhängig. Normalerweise gilt, je mehr Informationen verfügbar sind, desto besser sind die Vorhersagen der RS. In manchen Fällen sind jedoch nur begrenzt Daten verfügbar. In dieser Diplomarbeit wird eine Recommender engine vorgestellt, welche sich genau diesem Problem widmet. Um die Recommender engine mit Daten aus der realen Welt zu testen, wurden Reddit-Anfragen vom Subreddit (eine Kategorie innerhalb des online news Aggregationsservices reddit) *r/gamingsuggestions* gesammelt. In besagtem Subreddit ist es Usern möglich, Empfehlungen für Videospiele zu erfragen, und zwar ausschließlich basierend auf Informationen, welche vom Benutzer bereitgestellt wurden. Typische Anfragen beinhalten meist nur wenige bereits gespielte Titel und ein paar weitere Einschränkungen, z.B. dass der Support von Gamepads bevorzugt wird. Als Basis für die Empfehlungen kamen drei state-of-the-art Recommender Algorithmen zum Einsatz (collaborative filtering, matrix factorization und term frequency-inverse document frequency). Um die daraus resultierende Liste mit passenden Spielen zu verbessern wurden sogenannte post-Filter integriert. Durch den Vergleich der Empfehlungen der Algorithmen mit denen der Reddit-Community konnten Kennzahlen wie precision, recall und f1-score ermittelt werden. Die Algorithmen wurden außerdem einer qualitativen Evaluierung unterzogen, in welcher Menschen die Spielempfehlungen als gut, ok oder schlecht bewerten konnten.

Contents

List of Figures

List of Tables

1 Introduction

Since the dawn of the internet, the available information has been growing steadily. According to IBM (2017), today humans create 2.5 quintillion bytes of data each and every day. Therefore, about 90 percent of the world's data has been made in the past two years. In their *data never sleeps* report, Domo (2016) states, that about 400 hours of new video material is uploaded to YouTube[1] every minute, which equals 576,000 hours of new video material per day. However, the content is not limited to videos, it could be books, games, music, newspaper articles, scientific articles, and pretty much everything. One problem with this huge amount of data is, that the bulk of it likely is not of much interest for a specific user. Furthermore, it would be too time consuming or even impossible to search through the whole data for items of interest.

In, for example, a physical book store, there is only a limited shelf space available to display books. Therefore, only a certain number of books can be obtained, thus the choice of which book is going to be bought is influenced by known or unknown factors. Most likely those factors are economic ones. In order to maximize profit, it would be wise to present products which are popular and hence are often bought. Vice versa, a customer is only able to buy a small subset of all the available books in the world. In an online book store, there is no limitation regarding the number of books which are displayed. Nevertheless, the huge amount of items would be overwhelming. It would be difficult to find the specific books in which the customer is interested the most. There has to be some sort of filtering. However, in contrast to the physical book store, each customer should get her or his own shelf, containing books with the highest interest. This is were recommendation systems (RS) can help and typically yield good results (Ricci, Rokach, and

[1] http://www.youtube.com

Shapira, 2015).

By making a preselection of all available items based on the user's interests, former behavior, geographical location, given ratings or other measurable interactions, RS try to present only appropriate content. Hence, this makes the user experience more enjoyable increases the chance of a successful purchase. RS are widely used, large companies, such as Amazon, YouTube, Netflix, Google, Tripadvisor and numerous others implement them in their services (Ricci, Rokach, and Shapira, 2015; Felfernig et al., 2014). Additionally, recommendations could be of a general, non-personalized character. Those recommendations are by far easier to obtain, however, they are bound to be less useful for a single user. Examples include top-10 lists of best selling books, top-10 movies of most watched movies or most popular songs of December. Ricci, Rokach, and Shapira (2015) define RS as

> "software tools and techniques providing suggestions for items to be of use to a user. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read."

One of the most common and simplest forms of an output of a RS is a ranked list, similar to those top-10 lists mentioned above. What a RS tries to do is to order the huge number of available options based on the user's preferences and interests, thus trying to predict the most suitable products or services (Ricci, Rokach, and Shapira, 2015).

In order to calculate such predictions, knowledge about the user preferences is necessary. Those preferences could either be explicitly or implicitly stated. Explicitly stated knowledge includes, but is not limited to, for example, ratings a user has given for a specific item. Implicit preferences are based on the interpretation of user interactions. They could be comprised of, for instance, the time a user has spent on a web page describing a product, the duration the user played games from different genres, or the click-path the user used to get to a specific site (Ricci, Rokach, and Shapira, 2015). The underlying rationale for the development of RS is the observation, that individuals often tend to rely on recommendations given by their friends or acquaintances. For example, it is typical to rely on the suggestions of a person's colleagues when selecting a book, a movie, or even a restaurant

(Mahmood and Ricci, 2009; Ricci, Rokach, and Shapira, 2015).

As stated above, knowledge about a user is necessary to calculate good recommendations. Generally speaking, the more, the better. But what if there is only few information available? Suppose that only a small set of items a user likes is exposed. Suppose further, there are other constraints, such as a maximum price for a meal, a genre of a movie or a minimum play time for a game. Another person could give rather good recommendations, depending on various aspects. Is the recommending user capable of identifying latent factors for the given items and knows other items which could match? Does the user has experience in that field, and to some extent, does the recommending user has some empathy for the requesting user.

On reddit[2] (a social news aggregation service), a subreddit *r/gamingsuggestions*[3] exists, which enables the above scenario to happen online in the area of video games . Although it is very likely to get fairly good recommendations after posting a request there, there also are a few disadvantages:

- Requests may stay unanswered, due to very exotic games or restrictive constraints.
- One could wait some time for an appropriate answer.
- Maybe the post is missed due to a high number of new posts in a certain time.
- Recommendations given by the community often only consider a part of the given constraints.
- Community results may be biased due to, for example, geographical, age-related or cultural conditions.

To tackle the issues mentioned above, this thesis aims to describe and implement a recommender engine which enables people to ask for recommendations based on a list of games they liked in the past and some

[2] http://www.reddit.com
[3] https://www.reddit.com/r/gamingsuggestions

contextual information. Using only this very sparse information, it was tried to mimic the community taste, that is, the recommendation engine tries to produce similar recommendations as redditors—user on reddit—would have suggested. More precisely, following research questions were addressed:

RQ_1 How accurately can the community taste from requests posted on the subreddit *r/gamingsuggestions* be fitted with RS?

RQ_2 Are the recommendations suggested by the recommender engine viable, that is, does it yield a reasonable and satisfying output?

RQ_3 Considering post-filtering, how can recommendation results be influenced with re-ranking the recommendation list?

In order to answer those questions, the *steam platform* was used to obtain sufficient data about video games. Some of the data was available through an application programming interface (API), while the rest of the relevant data was obtained by scraping the steam website.

After the data acquisition and preprocessing, three standard recommendation algorithms (i.e., collaborative filtering, matrix factorization, term-frequency-inverse-document-frequency) were implemented. To gather train- and testcases, the subreddit *r/gamingsuggestions* was manually searched for suitable requests. Next, the post-filters were implemented and trained and evaluated in different combinations with the standard algorithms.

Afterwards, a qualitative evaluation of the recommendation results was performed, by asking people how well the recommended games are fitting the request.

The remainder of this thesis has the following structure: Chapter 2 gives an overview of RS, the knowledge discovery process, different recommendation algorithm approaches, the steam and the reddit platform. Chapter 3 summarizes already published work concerning the research questions stated above, Chapter 4 describes the steps and methods used for this thesis in

detail, Chapter 5 presents the specific results which are later on discussed and summarized in Chapter 6.

2 Background

This chapter gives a theoretical overview of important concepts and terms used in this thesis. First, an overview of RS including underlying principles and definitions of terms is given. Second, the knowledge discovery process is described in detail. Third, some common recommendation algorithms which had been used in for this thesis are introduced. A description of the two platforms *Steam*[1] and *reddit* concludes this chapter.

2.1 Recommender Systems

RS are tools and techniques that provide useful recommendations on, for example, "which item to buy", "which movie to watch" or "what online news to read" to a user (Ricci, Rokach, and Shapira, 2015). Melville and Sindhwani (2010) state, that

> "the goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them."

There are two main concepts of RS, *collaborative filtering* systems , which analyzes past interactions and behavior of the user (for example previously purchased items, given ratings to specific products), and *content based filtering* systems, which are based on item attributes and give recommendations based on the similarity of those properties (Melville and Sindhwani, 2010). Often, these two approaches are combined to hybrid recommender systems. Nevertheless, much smaller concepts include, but are not limited to, *knowledge-based*, *demographic* and *social* recommender systems (Ricci, Rokach,

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

and Shapira, 2015).

Most of the readers of this thesis already have come across RS in some way. Suppose, a user ("Frank") got a book recommendation by a friend, and searches the book online on amazon.com. The book is displayed as one of the search results, and there likely is another section on the page which could be called "Customers who bought this item also bought" or "Frequently bought together". In this section, other (similar) books are listed that Frank might be interested in. If he visits the site frequently, this section becomes personalized more and more, depending on Frank's behavior (purchase history, given ratings, ...). The underlying software that creates this list is a RS. One particular word has to be stressed at this point, *personalized*. Every user sees a different version of that section, fitted to the user's taste. In contrast, it would also be possible to simply display the top-10 bestseller of 2017, but the resulting list would likely be less useful (Jannach, Zanker, Felfernig, et al., 2011).

2.1.1 History

Humans often relied on recommendations they got from their acquaintances or friends in the past, so listening to the advice of another person has therefore always been a critical component of the decision making process. Since the emergence of huge internet marketplaces such as amazon, buyers are confronted with an increasing number of available products, while sellers are being faced with the demand of a personalized shopping experience (Melville and Sindhwani, 2010).

The idea of a computer recommending something to people is not very new, in fact, it can be traced back to 1979, where Rich (1979) introduced the computer program *Grundy*. Grundy tried to mimic the job of a librarian, that is giving good recommendations on books based on some stereotypes[2] of persons. In order to achieve that, the user was asked some questions regarding the name and some self-describing keywords. Grundy then matched the keywords to predefined stereotypes. Linked to those stereotypes, Grundy

[2] a stereotype is a compilation of frequently occurring characteristics of users

had some lists of books which might be of interest and proposed that books to the user (Rich, 1979).

The next big step was the introduction of the term "collaborative filtering" by (Goldberg et al., 1992). It was used for the description of the first commercial RS, *Tapestry*. The purpose of Tapestry was to recommend documents from newsgroups to users. Collaborative filtering means that users help other users to distinguish relevant from irrelevant documents by recording their reaction after they read them (Goldberg et al., 1992; Melville and Sindhwani, 2010). A good comparison to this early filtering approach is the usage of tags today. Users annotate certain documents or items with keywords. Other users are able to filter documents by using this keywords, thus receiving only relevant content.

In 1994, Resnick et al. announced *GroupLens*, an open architecure for collaborative filtering of netnews. They became the first to describe collaborative filtering as it is known today. They sorted the news according to a predicted score which was based on the ratings other, similar users gave the article. This implies that users who agreed in the past are likely to agree again.

Moreover, it was also in the mid-1990s when RS developed as an individual field of research. However, the need for good recommendations is growing, since there are more and more application areas emerging, such as recommending books, music, videos, vacations, financial services and much more (Adomavicius and Tuzhilin, 2005).

In 2009, RS gained some publicity outside the academic world, when Netflix announced the winner of their Netflix Prize. The challenge started in 2006 and the goal was to improve their algorithm Cinematch by at least 10% to win $1.000.000 (Netflix, 2009).

The interest in the research of RS has grown rapidly over the past few years. Reasons include the increase of electronic commerce, the ubiquity of information access, the rise of the Social Web, risen industrial interest and new application opportunities in the Mobile Web. The ACM Recommender Systems conference as well as the increasing number of publications serve as a good indicator for that growth (Jannach, Zanker, Ge, et al., 2012).

In the mid-2000s the research was primarily focused on increasing the accuracy of predictions of unknown items. Today, the spotlight is on improving other quality parameters like diversity, novelty or serendipity (Jugovac, Jannach, and Lerche, 2017).

2.1.2 Basic Terms

In this subsection, basic terms and concepts of RS are explained, based on (Ricci, Rokach, and Shapira, 2015). A typical RS consists of three entities which are related to each other, namely *users*, *items* and *transactions*. Those entities, as well as other important definitions are described below.

User

Looking at personalized recommendations, it is crucial to have information about users and the goals and characteristics about them. Which information, that depends on the recommendation technique. For example, in a demographic RS, that information could be age, gender or profession. For a collaborative filtering approach, a list of items the user liked/disliked or rated somehow would suffice. The user data constitutes the user model (Fischer, 2001; Billsus and M. Pazzani, 1996), meaning that the collected data of users encodes their preferences and needs. Without a user model, personalized recommendations would hardly be possible. Reconsidering a collaborative filtering approach, a user model could consist of direct interpretation of ratings. However, taking those ratings into account and deriving others factors which distinguishes users from each other would also be feasible. Other possibilities to obtain user data could be the analysis of browsing patterns (Taghipour, Kardan, and Ghidary, 2007) on a website or travel search patterns (Mahmood and Ricci, 2009).

Item

Items are the objects that are recommended and are characterized by their complexity and their value for the user. If an item is useful to a user, its value may be considered to be positive. If the selected item is not useful, that is, the user made the wrong decision of selecting it, the value may be negative.

Simple items in the context of RS can be movies, games, songs, books whereas more complex items also include insurance policies and financial investments (Montaner, López, and Rosa, 2003). Each item has specific properties or features, for example, a movie can be linked to various genres or actors.

Transaction

A transaction is defined by Ricci, Rokach, and Shapira as a

> "recorded interaction between a user and the RS."

Ratings are the most common form of transaction data. Ratings can be explicit or implicit. Explicit ratings are obtained by asking the opinion of the user about an item on a rating scale. Ratings can be of various types (Schafer et al., 2007):

- Numerical ratings, such as one to five stars for a product on amazon.com, or one to ten stars on imdb.com.
- Ordinal type, which is common in opinion polls, where users have to select the answer that suit them most, such as "Strongly agree, agree, neutral, disagree, strongly disagree" or the reactions on facebook.com, namely"like, love, haha, wow, sad, angry".
- Binary ratings, such as the thumbs up or thumbs down on youtube.com or on steam.

- Unary ratings, that indicate that a user observed, purchased, or otherwise positively rated an item.

With implicit ratings, the RS tries to guess the users' opinions based on their actions. For example, if a user types the word "cat" in the search field on youtube.com, the website first registers that the user might be interested in cats. As a result of the search query a list of video clips appears on the screen. If the user clicks on a video which is about some strange behavior of cats, the RS may conclude that the user is interested in funny videos with cats. Without about the user's opinion on cats, the RS has generated implicit information about the user's taste.

Feature

"A feature is an individual measurable property of a phenomenon being observed"(Chowdhury and Bhattacharyya, 2017). According to this definition, features of, for example, images could include colors, textures and contours. In other words, features describe the properties of an item in a suitable way for an algorithm. They can be of various types, for instance, numerical or categorical.

Prediction

To fulfill its purpose, that is, finding helpful items for a user, a RS must be able to rank the huge number of available items somehow. It must predict the value of an item to a user, or at least compare different items and decide which one would satisfy the user's needs more than the other. Hence, the core of a RS can be viewed as "the prediction of the utility of an item for a user" (Adomavicius and Tuzhilin, 2005; Ricci, 2014). The usefulness of an item to a user may also depend on contextual factors. It is, for example much more likely that users are interested in restaurants near them, even though other restaurants would have a higher utility.

Recommendation

A recommendation is the final result of the above mentioned prediction process. It usually consists of a small subset of the available items, ordered by their utility for the users. It should be noted that the final recommendation list may not only contain the highest ranked items, it could also incorporate the most popular items or items which are totally different or contrasting compared to the ones with the highest predictions. This technique should prevent recommendations from staying in the so called *filter bubble*, which is a phenomenon that describes the isolation of people from a diversity of viewpoints or content (Nguyen et al., 2014; Pariser, 2011).

Utility Matrix

The utility matrix serves as the foundation for various recommendation algorithms. As stated above, there are the users, the items and the transaction (i.e., the ratings between them). The utility matrix pictures those relations in a compact format.
The utility function maps each item-user pair to a rating and is defined as

$$f_u : U \times I \rightarrow R, \tag{2.1}$$

where U is the set of user, I is the set of items, and R is a set of ratings, for example $R = \{1, 2, 3, 4, 5\}$. The numbers obtained by the utility function can be represented by a utility matrix $\boldsymbol{M} \in \mathbb{R}^{n \times m}$, where n is the number of users and m the number of items. Hence, each user is represented by a row-vector with the given ratings r for an item as a value in the corresponding column: $\vec{u} = (r_1, r_2, r_3, \cdots, r_n)$.

It is assumed, that the matrix is sparse, which means, most of the entries are unknown, thus the users' preferences for most of the items is unknown. An example of a typical utility matrix is shown in equation 2.2, with values taken from table 2.1 (Helic and Kern, 2018; Ullman, 2018).

	HP1	HP2	HP3	The Hobbit	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Table 2.1: Example ratings, adapted from Helic and Kern (2018) and Ullman (2018). Users are represented by capital letters A through D. HP1, HP2 and HP3 stand for Harry Potter 1-3, SW1, SW2, SW3 for Star Wars Episode 1-3

$$\boldsymbol{M} = \begin{pmatrix} 4 & & & 5 & 1 & \\ 5 & 5 & 4 & & & \\ & & & 2 & 4 & 5 \\ & 3 & & & & 3 \end{pmatrix} \tag{2.2}$$

Note, that empty cells in the utility matrix represent unknown ratings for the respective user-item pair. The goal of a recommender system is to predict ratings $\hat{R}_{u,1}, \cdots, \hat{R}_{u,N}$ for each user u.

2.1.3 Classes of Recommender Systems

There are several classed of RS. Burke (2007) distinguishes between six different recommender approaches.

- Content-Based
- Collaborative Filtering
- Demographic
- Knowledge-Based
- Community-Based
- Hybrid

2.1.4 Evaluation

In order to measure the accuracy of RS, several metrics exist. Since the goal of this thesis was to fit the community taste, that is, how well do the results of the algorithms overlap with the suggestions from the community, typically used evaluation metrics are described below.

Precision

As defined by Perry, Kent, and Berry (1955), precision, in the information retrieval context, is based on sets of *retrieved documents* and *relevant documents*. The set of retrieved documents contains all documents a, for example, search engine lists as the result of a query. The relevant documents contain all documents that are deemed relevant for that inquiry. Precision could be formulated as the question of "how many of the retrieved documents are relevant?". Therefore, precision is defined as

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|} \tag{2.3}$$

Recall

Recall is the percentage of relevant documents that are successfully retrieved. Recall could be formulated as the question of "how many of the relevant documents were found?", and is defined as

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|} \tag{2.4}$$

F1-Score

The F1-Score is the harmonic mean of precision and recall, thus combines the two measures mentioned above.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.5}$$

2.2 Knowledge Discovery Process

Historically, the collecting and analysis of data had many names. Data mining, information harvesting, data pattern processing, and information discovery, just to name a few (Fayyad, Piatetsky-Shapiro, and Smyth, 1996). One of the most established definition is by Fayyad, Piatetsky-Shapiro, and Smyth (1996) who define the Knowledge Discovery in Databases Process as

> "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."

Furthermore, they describe the process as

> "interactive and iterative, involving numerous steps with many decisions being made by the user."

Figure 2.1 pictures the Knowledge Discovery Process with its five major steps: Selection, preprocessing, transformation, data mining and interpretation/evaluation. The dashed lines indicate the iterative nature of the process, meaning that it is always possible to go back to a previous step and start over from there. It should be noted, that there are several other models of the process available (Cios et al., 2007). However, the Knowledge Discovery Process defined by Fayyad, Piatetsky-Shapiro, and Smyth (1996) as well as Brachman and Anand (1996) can be understood as the base of this thesis. Hence details about the individual steps are described below.

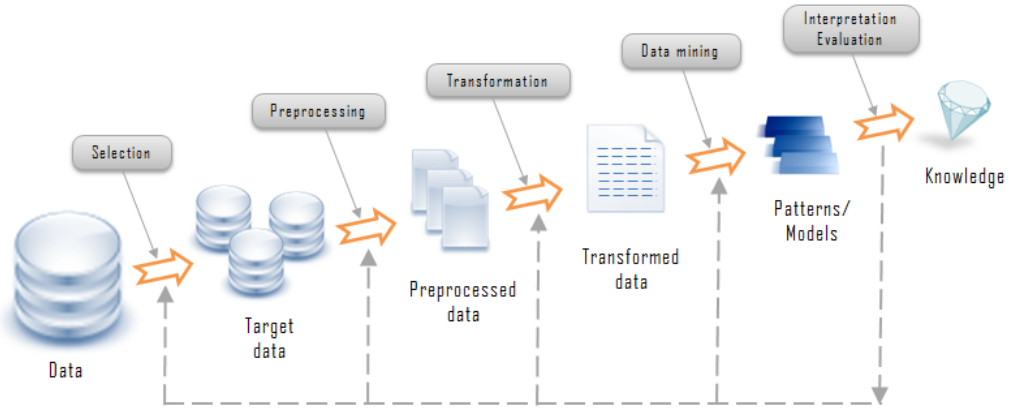

Figure 2.1: Knowledge Discovery Process with its major steps: Selection, Preprocessing, Transformation, Data Mining and Interpretation/Evaluation. The process may be non-linear, thus it is always possible to go back to a previous step as indicated by the dashed lines. Illustration taken from (Wigzo.com, 2018)

Selection

The goal of the selection step is to create a target data set. The target data set is a subset of variables, on which the knowledge discovery is to be performed. It is crucial to understand the application domain as well as relevant prior knowledge to obtain such a dataset. The selection step in this thesis consisted of finding and obtaining sufficient data for the later steps.

Preprocessing

Unwanted data, such as noise, should be removed in this step. Strategies on how to cope with missing data should be developed. In the realm of RS, data preprocessing could include the removal of HTML-Tags or adding average ratings instead of missing values.

Transformation

The third step is characterized by reducing the dimensionality of selected data, transforming the data in a useable format as well as finding useful features (feature extraction). For example, in this thesis the hours a user spent playing a specific game had been transformed to ratings between 1 and 5.

Data Mining

The data mining step is all about finding patterns in the data obtained from steps one to three. Such patterns may consist of classification rules or trees, regression and clustering. For example, suppose there is a set of numbers. During the data mining step, it might become clear that the numbers are distributed like a Gaussian distribution. Therefore, one could calculate the mean and standard deviation of the numbers, which would give a better understanding of the data. Hence, the Gaussian distribution with the calculated parameters would become the model or pattern of the data. Regarding RS, data mining could consist of fitting the parameters of the underlying model (of each approach).

Interpretation/Evaluation

The last step is all about visualization of the found patterns and models. The discovered knowledge should be incorporated in the system as well as documented and reported to interested parties. It is possible that the found knowledge interferes with the existing model. Such conflicts are subject to solving.

2.3 Recommender Algorithms

Ricci, Rokach, and Shapira (2015) define the recommendation problem as

> "estimating the response of a user for new items, based on historical information stored in the system, and suggesting to this user novel and original items for which the predicted response is high."

As stated in the introduction (Chapter 1) of this thesis, recommendations can be personalized or non-personalized (e.g. best selling books in a book store). The personalized approaches can be further divided in *content-based*, *collaborative filtering* and *hybrid* methods.

Content-based approaches try to identify common attributes or characteristics of items (i.e., "looking at the content") that the user rated positively and propose items, that share characteristics with the originally rated items (Balabanović and Shoham, 1997; Billsus and M. J. Pazzani, 2000). As pointed out by Shardanand and Maes (1995), methods that solely build upon content-based approaches generally suffer from limited content analysis and overspecialization. Limited content analysis occurs, when the system has difficulties in obtaining contentual information from the items. Reasons include privacy issues (i.e., the user might not give away personal information), the difficulty or cost intensity of content procurement or the fact that the content of an item often is insufficient to define its quality.

Overspecialization is inherent to content-based approaches. For example, a user might like video games with a lot of strategic thinking settled in the medieval age. Hence, a content-based RS recommends only strategy games or games within a medieval setting, therefore skipping out other potentially interesting games.

In contrast, collaborative filtering approaches rely on the rating information of other users and items in the system. The main idea is, that users who agreed on items in the past are likely to agree on other items as well. In other words, the rating a user would give to an unrated item would be similar to the rating another user gave the item, if both users rated the same items similar in the past. Collaborative filtering approaches deliver better results for items for which the content is difficult to assess, they take into account the quality of the item, and they might recommend different items outside of some filter bubble (Ricci, Rokach, and Shapira, 2015).

Collaborative filtering can be separated in *neighborhood-* and *model*-based approaches. With neighborhood-based methods, the ratings of users for items are directly used for the recommendation, either user-based or item-based (Breese, Heckerman, and Kadie, 1998; Deshpande and Karypis, 2004; Linden, B. Smith, and York, 2003).

Model-based approaches try to fit a model to the underlying data and perform the recommendations or predictions with that model. There are various approaches known, some of them are Bayesian Clustering (Breese, Heckerman, and Kadie, 1998), Latent Semantic Analysis (Hofmann, 2003), Support Vector Machines (Grčar et al., 2006) and Matrix Factorization (Singular Value Decomposition) (Bell, Koren, and Volinsky, 2007; Koren, 2008; G. Takács et al., 2008). On the following pages, two collaborative filtering approaches (item-based k-nearest-neighbor and matrix factorization) and one content-based (i.e., term frequency – inverse document frequency) are described. The actual description of the implementation of the algorithms is discussed in Section 4.5

2.3.1 Collaborative Filtering – kNN

The following section is based on the *Recommender Systems Handbook* by Ricci, Rokach, and Shapira (2015), other sources are marked individually.

Neighborhood-based recommendation build upon the underlying principle, that similar users prefer the same items and similar items are preferred by the same users.

Consider the following example in Table 2.2: Eric wants to know whether to watch the movie Titanic or if he would be better of by watching another one. Lucy has a similar taste for movies because she rated the same movies analogous to Eric. Eric therefore might ask Lucy if she liked Titanic and consider her opinion in his decision process. Eric also observes, that Diane has rated movies high which he did not like, hence he discards her opinion on movies or even considers picking the opposite of Diane's movies.
For collaborative filtering, two approaches are common, namely *user-based*

and *item-based* predictions.

Table 2.2: Example to illustrate collaborative filtering. Each line represents a user, each column a movie. The number indicates the rating that person gave that movie. Empty cells indicate that no rating was given.

	The Matrix	**Titanic**	**Die Hard**	**Forrest Gump**	**Wall-E**
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2	**?**	3	5	4
Diane	4	3	5	3	

User-Based

The sequence from the example above would be a user-based recommendation. Eric asks others users who are similar to him which movies they would recommend. More formally, the rating r_{ui} of a user u for an item i is calculated by looking at the ratings for i by users most similar to u. Suppose there is a weight w_{uv} for each user $v \neq u$ which denotes the similarity between u and v. The *k-nearest-neighbors* (k-NN), denoted by $\mathcal{N}_i(u)$, are the k users with the highest similarity w_{uv} to u that also rated item i.
The estimated rating $\hat{r}_{ui}$ for item i from user u is therefore

$$\hat{r}_{ui} = \frac{1}{|\mathcal{N}_i(u)|} \sum_{v \in \mathcal{N}_i(u)} r_{vi}. \tag{2.6}$$

Based on the ratings in Table 2.2, it is quite obvious that Lucy would have a higher similarity score to Eric than Diane. However, in the above formula, Diane's and Lucy's rating would be considered equally (assuming those two are the two nearest neighbors). Since the taste of Lucy and Diane is quite different, an equal consideration would not yield good recommendation results. To account for that problem, the ratings of the nearest neighbors should be weighted using their similarity to u. In order to stay in the allowed

rating range, the weights are normalized as:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \tag{2.7}$$

However, there is still a flaw in the equation. Diane only gave one movie a rating of 5, whereas Lucy gave three times 5. There appears to be a rating bias between the different users. Diane seems to give a rating of 5 only to her absolute favorite movie, Lucy seems to give 5 for every movie she likes. To get rid of that user bias, the ratings r_{vi} for each person are normalized to $h(r_{vi})$ as:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} h(r_{vi})}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}\right). \tag{2.8}$$

The predicted rating $\hat{r}_{ui}$ has to be converted back to the "real" rating the user would give, thus the user bias has to be added to the prediction (denoted by h^{-1}). There are several approaches on how to normalize the ratings and how to calculate the similarity. The ones which were used in this thesis are discussed in Section 4.5.3

Item-Based

Instead of relying on the opinion of like-minded users, the item-based approach looks at the ratings of similar (or "like-minded") items (Sarwar et al., 2001). Looking again at the example from Table 2.2, Eric would look at the ratings Titanic received from other users. He would notice, that the ratings from Titanic are similar to the movies Forrest Gump and Wall-E. Since he liked those two movies, he concludes that he would also like Titanic.

More formally, the prediction can be written as

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij} r_{uj}}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}, \tag{2.9}$$

$\mathcal{N}_u(i)$denoting the items rated by user u most similar to item i. To respect the different rating behaviors of the users, the normalization term from the user-based approach can be reintroduced:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij} h(r_{uj})}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}\right). \tag{2.10}$$

2.3.2 Matrix Factorization – SVD

The following section is based on the following papers: Gábor Takács et al. (2008), Ott (2008), D. D. Lee and Seung (2001) and D. Lee and Sebastian Seung (1999).

The idea of matrix factorization models is to discover *latent factors*. Latent factors describe attributes of items, which may not be obvious or explicitly stated. They are therefore derived solely from the user feedback (i.e., the ratings users gave items).
However, matrix factorization is not the only possible model for discovering latent factors. Others include, for example, probabilistic latent semantic analysis (pLSA) (Delgado and Ishii, 1999), latent Dirichlet Allocation (Billsus and M. J. Pazzani, 1998), or neural networks (Gori and Pucci, 2007).
Although the classic singular value decomposition (SVD) is well suited for identifying latent factors (M. Blei, Y. Ng, and Jordan, 2001), there is one major issue. SVD is not defined when the matrix is incomplete. Since utility matrices are generally very sparse, Funk (2006) proposed an iterative algorithm to circumvent that issue.
Mathematically, matrix factorization tries to find two matrices P and Q such

that their product approximates R:

$$R \approx P \times Q^T = \hat{R} \tag{2.11}$$

with $P \in \mathbb{R}^{n \times d}$ and $Q \in \mathbb{R}^{m \times d}$, n denoting the number of users, m denoting the number of items and d denoting the number of latent factors to discover. Each row in P thus describes the strength of association of a user to a factor whereas each row in Q characterizes the association of each item to the discovered factors. In order to calculate the prediction of an arbitrary item j for a user i, it is sufficient to multiply (dot-product) the corresponding vectors p_i and q_j:

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{d} p_{ik} q_{jk} \tag{2.12}$$

As mentioned above, Funk (2006) proposed a way of finding the two matrices P and Q in an iterative way. Both matrices are first initialized with arbitrary values. After that, the difference from the product of P and Q to the original ratings matrix R is calculated. The goal is to iteratively minimize that difference. As an error measurement, the regularized squared error is used:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - p_i^T q_j)^2 \tag{2.13}$$

In order to decrease the error, one has to know the direction, that is the sign in which the values in P and Q should be adapted. Such a method is called *gradient descent*. To get the direction of adaption, the error function has to be differentiated with respect to both variables:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij}) q_{jk} = -2 e_{ij} q_{jk} \tag{2.14}$$

$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij}) p_{ik} = -2 e_{ij} p_{ik} \tag{2.15}$$

After obtaining the gradient, the update rules for p_{ik} and q_{jk} can be formulated:

$$p'_{ik} = p_{ik} + \gamma \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\gamma e_{ij} q_{jk} \tag{2.16}$$

$$q'_{jk} = q_{jk} + \gamma \frac{\partial}{\partial q_{jk}} e_{ij}^2 = q_{jk} + 2\gamma e_{ij} p_{ik} \tag{2.17}$$

Additionally, a regularization term λ can be introduced to avoid overfitting. That changes the update rules to

$$p'_{ik} = p_{ik} + \gamma(2e_{ij}q_{jk} - \lambda p_{ik}) \tag{2.18}$$

$$q'_{jk} = q_{jk} + \gamma(2e_{ij}p_{ik} - \lambda q_{ij}). \tag{2.19}$$

The iterative process of finding P and Q is repeated until e reaches a certain threshold, in most cases 0.001, or until a maximum number of iterations is executed.

2.3.3 Term Frequency – Inverse Document Frequency

One well established way of defining the content of text is via keywords (Adomavicius and Tuzhilin, 2005). A keyword, also sometimes called a catchword, is "a significant or memorable word or term in the title, abstract, or text of a document"[3]. Keywords should therefore be important words, that describe the content of a text corpus. Such keywords can be created manually, by humans, or they can be selected automatically. Automatically generated keywords are usually determined with some weighting measure w_{ij}. One well established way of measuring the importance of a word inside a collection of text is the Term Frequency – Inverse Document Frequency (TF-IDF) (Salton, 1989).
The following explanation is based on Adomavicius and Tuzhilin (2005). Term frequency denotes the number of appearances of a word k_i in a given document $d_j \in D$:

$$TF_{ij} = \frac{f_{ij}}{max_z f_{zj}}, \tag{2.20}$$

with f_{ij} as the number of times a keyword k_j appears in a d_j, normalized by the maximum frequency $max_z f_{zj}$ of all keywords. While the introduction of the term frequency would yield the most used words in a document, for example, words such as "the", they would not help to distinguish the

[3] http://www.dictionary.com/browse/keyword?s=t

documents from each other. Hence, so called stopwords can be taken into account. Stopwords are words which are commonly used but do not give any information of the text. Typical examples are *the*, *or*, *and*. By filtering out those words, the information content can be increased. However, assume there are a lot of texts about computer science which should be described with keywords. One word that would be in nearly all texts would be *bit*. So even if it describes the content of the documents, the information gain by selecting that word would be fairly low since every document contains the word. Hence, the Inverse Document Frequency is introduced:

$$IDF_i = log\frac{N}{n_i}, \tag{2.21}$$

with N as the total number of documents and n_i as the number of documents containing keyword k_i. IDF is a measure of how much information a word provides for a given text, that is whether the word is common or rather rare in the whole set of documents.

Thus, the weight of a term k_i in a document d_j is

$$w_{ij} = TF_{ij} \times IDF_i \tag{2.22}$$

and the content of a document d_j is

$$content(d_j) = (w_{1j}, w_{2j}, ..., w_{kj}). \tag{2.23}$$

2.4 The Steam Platform

Steam is a digital distribution platform developed by Valve Corporation. It offers instant access to games and an active community with over 150 million people. It was created in 2002, mainly because Valve had updating issues with its online games (e.g., Counter-Strike). Since 2013, Steam is the worlds largest digital distribution platform for video games (Edwards, 2013). The steam client was first publicly provided in January 2003. In order to play Counter-Strike 1.6, it was mandatory to use. However, at that time it was only used to deliver patches for Counter-Strike. In 2007, large companies such as id-Software (Bramwell, 2007b), Eidos Interactive (Purchese, 2007) and Capcom (Bramwell, 2007a) began to deliver their games via Steam. Since then, the user base as well as the amount of sales on steam grew larger every year, reaching 150 million registered users in 2018, 18.5 million of them are online concurrently at peak hours (Steam, 2018a). According to Forbes, 50 to 70 percent of the 4 billion US$ market for downloadable video games belongs to Steam (Chiang, 2011).

The Steam Store is available via web[4], however, the downloadable client shown in Figure 2.2 provides the most functionality. The core functionality is the provision of purchased games, which are displayed in the *Library* tab. Within the *Store* tab, users can purchase games filtered by various options. For example, users could look for games their friends are playing. It is also possible to follow a so called *Curator*, who are "individuals or organizations that make recommendations to help others discover interesting games in the Steam catalog" (Steam, 2018a).

Steam itself provides the user with three different kinds of recommendations. To the best of the author's knowledge, the exact recommendation process is nowhere to be found, nevertheless, the observations are described below as accurately as possible:

The *Featured & Recommended* section contains a huge list of personalized recommendations. The recommendations seem to be based on the tags of the games currently played by the users and their friends. Steam provides

[4] http://store.steampowered.com/

the information on the tags that have been used for the recommendation. It seems that there are also games included which only share one or two tags with the games played, which indicates that they are featured over the whole Steam client.

The *Discovery Queue* contains twelve different games, again, based on the games recently played. However, the list is heavily weighted towards new top-sellers, meaning that the queue contains the best-selling games, in accordance to the users taste. The goal of the discovery queue is to make sure, the user does not miss anything (Schreier, 2014).

The *Curator Recommendation* suggests games, based on the recommendations given by curators mentioned above.

One key feature, which is a crucial part of this thesis, is the possibility to rate games. A typical rating page is depicted in Figure 2.3. Valve uses binary ratings (thumbs up, thumbs down) for the games on their platform, however, they also store the playing time a user spent playing a game and an optional review text. Especially the playing time in combination with the thumbs up, thumbs down, could provide precious additional information on how much a user liked or disliked a game.

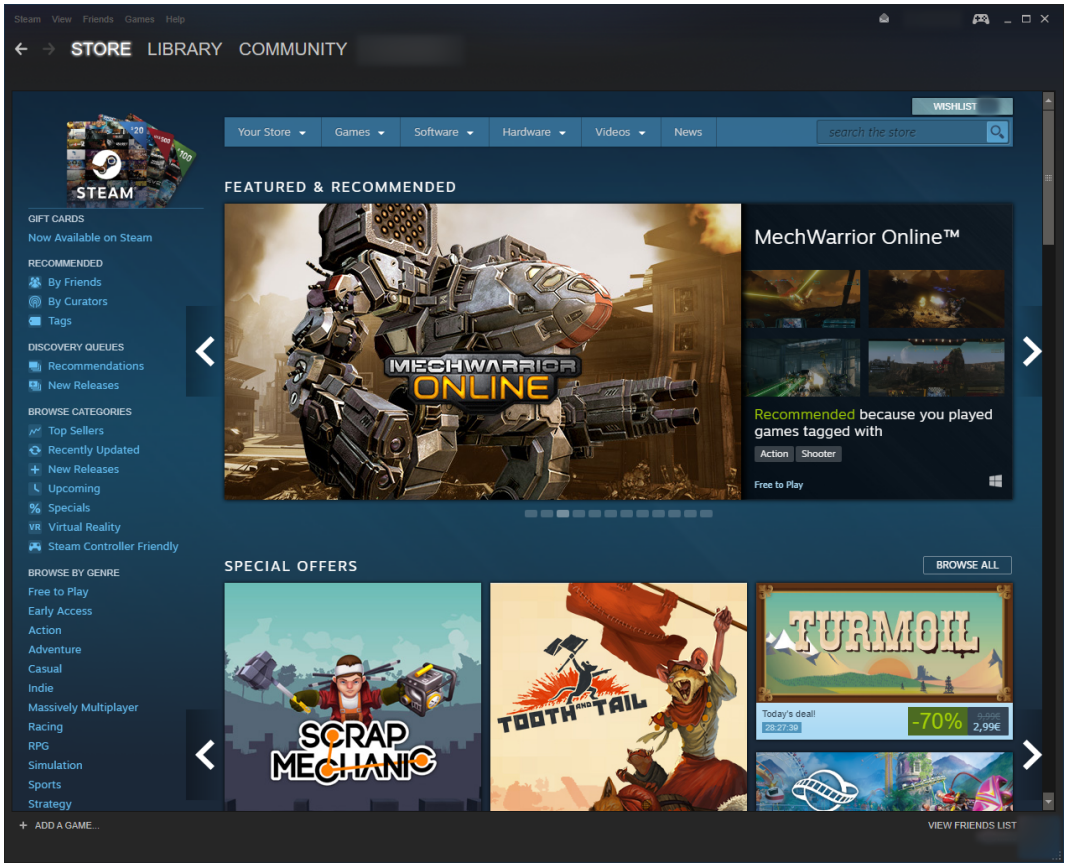

Figure 2.2: Screenshot of the steam client with personal information omitted.

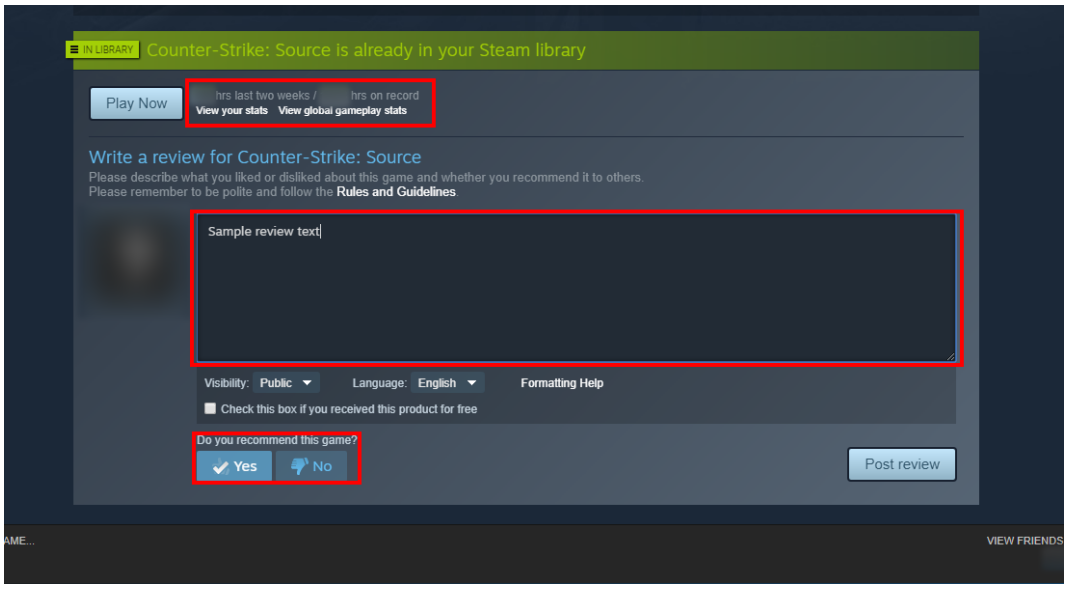

Figure 2.3: Rating page of the game Counter-Strike: Source. Note the highlighted areas for game time, review text and recommendation checkbox.

2.5 The Reddit Platform

Reddit is an American social news collection. According to (Reddit, 2018), it is "a source for what's new and popular on the web". Users provide the whole content and decide, via up- and downvoting, what is good and what is not worth reading. Essentially, Reddit is like a bulletin board. There are several subcategories (subreddits) which are dedicated to various topics. They can be understood as user-created areas of interest. Users can create threads (submissions) in a topic, users can comment the submission as well as vote for it. The subreddits are usually managed by the creators, however, other admins can be designated.

Reddit was launched in 2005 by Steve Huffman and Alexis Ohanian. Table 2.3 shows the dimensions of the whole Reddit community.
Figure 2.4 gives an overview of the subreddit */r/gamingsuggestions*. The tabs at the top order the various submissions according to different criteria. Each submission has a title and an optional content. The title is shown in the

Table 2.3: Overview Reddit dimensions (C. Smith, 2018)

	Amount
Registered users	250 million
Subreddits	853 824
Communities	50 000
Countries with Reddit Users	217
Average daily votes	25 million
Reported value of Reddit	1.8 billion US-$

overview of a subreddit. The highlighted arrows with the number between them indicate the sum of total up- and downvotes (i.e., the score). On the far right side, a short summary of the subreddit is given. On top of the summary, usage statistics are depicted. Figure 2.5 shows a submission inside */r/gamingsuggestions*. The title of the submission is displayed again on the top, followed by the description of the submission. In this case, the redditor (=user on Reddit) asks for games similar to Cuphead, with some other constraints. Altogether, seven users commented on this submission. Note the caption saying "10 points" on the right side of the username of the first comment which is the score for that post. 10 points indicate that a lot of users agree with "redmandolin", making that user's suggestions credible.

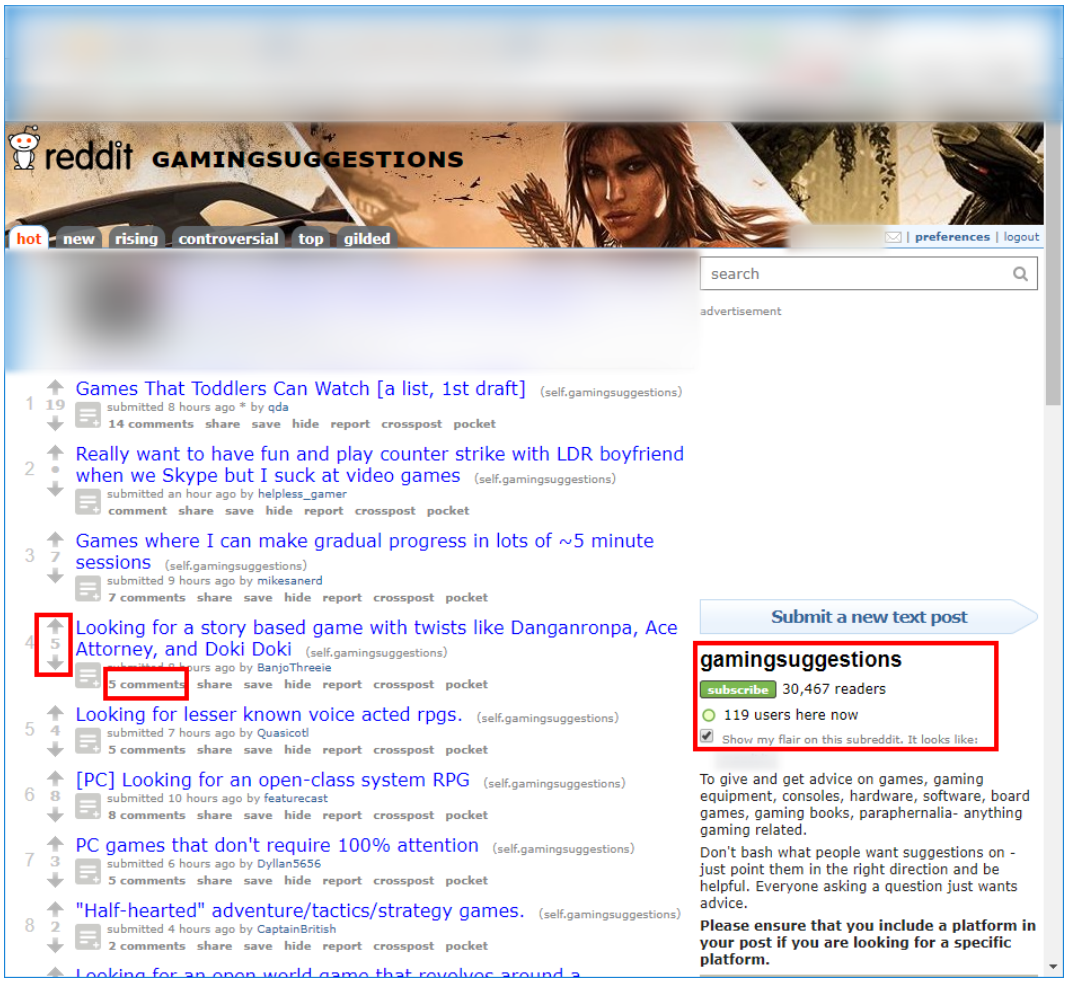

Figure 2.4: Overview of the subreddit /r/gamingsuggestions

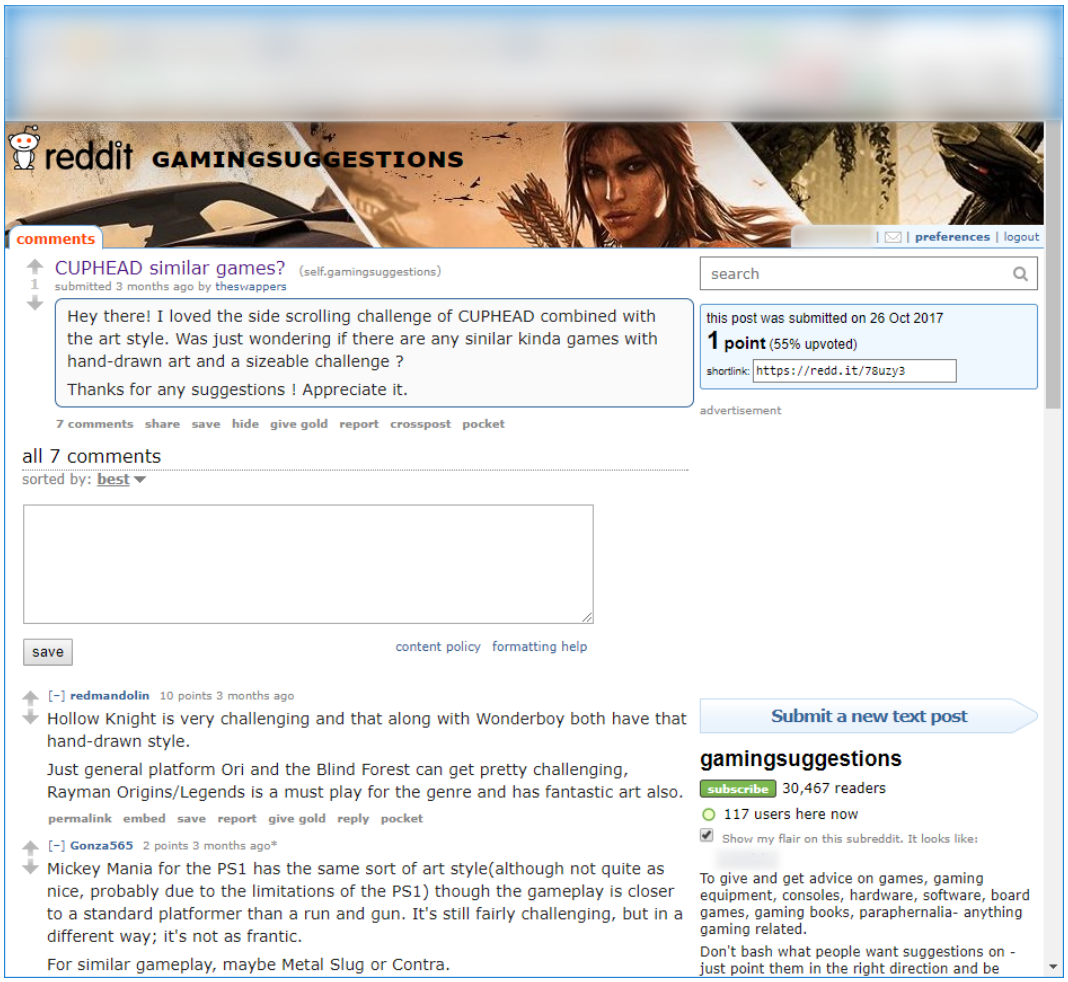

Figure 2.5: Detailed view of a submission in /r/gamingsuggestions

3 Related Work

Many studies about RS and recommendation algorithms had been conducted in the past few years. Recommending video games also gained attention in the past decade. Nevertheless, in the context of online communities and video games, not much work has been done yet.
Most of the already performed studies focus on algorithmic advantages in the domain of video games. Skocir et al. (2012) describe a method for recommending mobile games by looking at the interactions of users with them. The gathered information is used to model the users' success and progress. Based on that, other mobile games are recommended, matching the difficulty taste of the user.

Sifa, Bauckhage, and Drachen (2014) used a factor oriented model and a neighborhood oriented model to predict a list of games to users which they had not played yet and might be interested in. For both methods, archetypal analysis—which essentially is a dimensionality reduction like SVD—was performed.

Anwar et al. (2017) proposed a classical collaborative filtering approach for recommending video games. They used the ratings given for games by users inside a community, matched similar users with the requesting users and predicted new games for them. To ensure the quality of their algorithm, they tested it on a large standard dataset, namely the MovieLens dataset. In order to improve the recommendations, a genre-based filter was introduced.

Heinz, Lau, and Epstein (2017) built an online recommendation service[1] for movies, video games and TV shows. It is possible to enter up to five different entities and the system delivers recommendations based on the

[1] https://metarecommendr.herokuapp.com/

entries. Techniques, such as content filtering and collaborative filtering, were implemented. The used data foundation was scraped from metacritic.com, an online reviews aggregation service.

The RS used by Microsoft on its Xbox was described by (Koenigstein et al., 2012). They used an implicit binary rating scale based on the ownership of games. Positive ratings were comprised of a game ownership with a minimal playing time. Negative ratings for a player consisted of random games not owned, based on their popularity. The prediction model which has been established is a probabilistic matrix factorization technique with item biases.

Regarding narrative-driven or context-driven recommendations, Adomavicius and Tuzhilin (2011) introduced REQUEST, a recommendation query language that enables users to formulate and customize their recommendation needs.

Another approach for incorporating the current situation and the current needs of users was proposed by Hariri, Mobasher, and Burke (2013). They used an extended version of a Latent Dirichlet Allocation (LDA) model that combines users, items and context associated meta-data.

The term *narrative-driven recommendation* was first coined by Bogers and Koolen (2017), meaning that the "recommendation process is driven by both a log of the user's past transactions as well as a narrative description of their current need and the context of use". They further state, that narrative-driven recommendations are a combination of traditional recommendations based on user preferences, and the focused description of the current needs in a given context, making them a complex recommendation scenario.

In their study, Adomavicius and Tuzhilin (2011) state, that contextual information is very important when providing recommendations. In order to achieve that, they introduced contextual pre- and post-filtering techniques. The pre-filters adjust the raw data with the given context, before any state-of-the-art recommendation algorithms are applied. Post-filtering on the other hand, is added to the ranking which was obtained by recommendation approaches beforehand. These methods filter out irrelevant data or adjust

the ranking of the recommended items.

To the best of the author's knowledge, a video game RS covering several algorithmic approaches with applied post-filtering techniques to meet the narrative-driven recommendation needs of users in online communities has not been addressed in-depth before.

4 Materials and Methods

In this chapter, all undertaken actions are described to reproduce the results of this thesis. First, the utilized development environment including programming languages and libraries, the used hardware, and the operating systems are enumerated. Next, the necessary scripts for the data acquisition (i.e., the knowledge discovery process) are explained. Afterwards, the test case aggregation is demonstrated. Finally, the recommendation engine with its recommender pipeline is described, including the used recommendation algorithms, the post-filter techniques and the evaluation methods.

4.1 Development Environment

Two different machines were used, one for the development of the different software pieces as well as the testing, and another one for the actual execution of the crawling/scraping procedure and the recommendation calculations (see Table 4.1).

The following additional libraries and programs had been utilized:

- PeeWee 2.10.1
- Scrapy 1.4.0
- SciKit-learn 0.19.1
- gRequests 0.3.0
- MySQL 14.14

Table 4.1: Overview of the development environment

	Development machine	**Execution machine**
Processor	Intel Core i5-6200U CPU @ 2.3GHz, 2.4GHz	Intel Xeon CPU E5-2620 v3 @2.4 GHz, x 24
Memory	8 GB	252 GB
System	Windows 10 Home, 64 Bit	Ubuntu 14.04.5 LTS
IDE	Eclipse Oxygen 1a Release (4.7.1a) with python extension	
Python	3.6.2	3.4.3

4.2 Data Acquisition

In order to get a well suited data foundation, the steam platform was used. It provides a huge number of different games, and an even higher amount of users who reviewed the bulk of the game catalog with binary ratings and the time they spent playing particular games.

The data acquisition was split in five different scripts. Due to the architecture of the steam API and its output, each script relied on the output of the previous ones. Steam grants access to the data with an HTTP based API, which can be used to access a variety of Steamworks[1] features. Generally, the API is accessed by sending a request (either HTTP or HTTPS) to `api.steampowered.com`. The Web API is divided in multiple interfaces which are again separated in different methods. A typical call to the API therefore looks like this:

```
https://api.steampowered.com/<interface>/<method>/v<version>/
```

The result format can be specified with a format parameter, which yields

[1] Free suite of tools available to any developer to use in their game or software on Steam (Steamworks, 2018)

results in JSON[2], XML[3], CSV[4] or VDF[5] format. For the experiments in this thesis, a slightly modified JSON format was used, namely JSON-Lines. The difference to regular JSON format is, that each line corresponds to one JSON object, which makes it easier to split files and process the data in chunks.

At the time of writing this thesis, the usage of the API was limited. It was not possible to get a list of all reviews for a specific game, neither to achieve a list of all reviews for all games via the API. Although detailed information about a user was available, a list of all registered users could not be obtained through the API. The reviews and the user data were acquired through regular scraping. Furthermore, there was a time limit for calls to the API which is maximum 200 requests per 5 minutes. As it turned out, steam also had some defensive mechanisms against excessive scraping that protect their website from high loads. In order to speed up the scraping process, proxy servers from FineProxy[6] were utilized.

Finally, three scraping, respectively crawling scripts were necessary to obtain the raw data, and two scripts for the transformation and partially post-processing.

4.2.1 Overview

This section gives a short overview of the scraping process, details are discussed later on in the corresponding section of each script.

The first script, "Games-Crawler" fetched all available games from steam as well as the details. As an output, a list of links to the review page of each game is produced.

That file acts as the input for the next script, "Review-Crawler", which—in

[2]Javascript Object Notation
[3]Extensible Markup Language
[4]Comma Separated Values
[5]Valve Data Format
[6]http://fineproxy.org/eng/

the initializing run—scrapes all reviews for each link in the provided file. The reviews are stored temporary in one file per game. The reason for that is, that steam uses two different kinds of profile IDs, one numerical and one based on the username. A review is assigned to a user by the user's profile URL, which is either `http://steamcommunity.com/profiles/USER_ID` or `http://steamcommunity.com/id/USERNAME`. Since the profile ID is also used for some other calls to the API, it is considered as the more important feature and therefore used for identification. That decision lead to one problem, that is, that there is no efficient way to get a profile ID from a username.

Hence, another scraper, "SteamID-Crawler" was needed. It takes all profile links with a username (which can be easily distinguished because of the base URL, (`steamcommunity.com/profiles/...` vs. `steamcommunity.com/id/...`) and fetches the steam ID from the user's profile site and stores it in one file per game.

Since the only way to get users is to scrape reviews and to extract the profile URL, another limitation of the resulting dataset is, that only users who reviewed at least one game are included.

The stored information of each user (i.e., the profile URL and the profile ID) is then converted to the database with "Users-to-DB-converter". As a last step, the reviews can be added to the database ("Reviews-to-DB-converter"), since now every profile-URL has a profile ID which can be used to assign a user unambiguously to a review.

4.2.2 Games-Crawler

The Games-Crawler was utilized to first obtain a list of all applications on steam and second, to fetch all details for each application. Steam provides a variety of applications, Table 4.2 gives an overview of the different types. According to the official documentation of the steam API, there is no such function to expose all available applications on steam. After some research, it turned out, that there is (or was) an interface for that purpose, however, it

Table 4.2: Overview of application types on steam (Steam, 2018b)

Type	Description
Game	All games
Software	Different kinds of software, for example design and illustration or photo editing
DLC	Downloadable Content, additional content for games
Video, Series and Episode	Movies, TV-Shows, game videos, tutorials
Demo	Free demo versions of games

is not documented.

```
http://api.steampowered.com/ISteamApps/GetAppList/v0002/
```

delivers a JSON response with the desired information (i.e., application name and application ID). With the app IDs in that response, detailed information about each game can be obtained via the URL

```
http://store.steampowered.com/api/appdetails?appids=APP_ID
```

which is also not part of the official documentation. Table 4.3 gives an overview of the crawled data.

The collected game data is stored to a mySQL database after some processing steps discussed in Section 4.3. As a preparation for the next crawler, the Review-Crawler, a list with all review URLs is generated and saved to a simple text file. A URL to the review page of a game has the following format: `http://steamcommunity.com/app/GAME_ID/reviews/?browsefilter=mostrecent&p=1&filterLanguage=default`. The resulting file served as input file for the Review-Crawler as well as an input and control file for the unittest test cases described in Section 4.8. The tags however were not available via the API, hence a third party scraper[7] was utilized to collect that data.

[7]https://github.com/prncc/steam-scraper

Table 4.3: Application details

Attribute	Data type	Description
AppID	Integer	ID of the application
type	String	See table 4.2
name	String	Name of the application
required_age	Integer	The minimum age
is_free	Boolean	Indicates whether game is free to play
description	String	Description of the game
developers	List of Strings	List of the developers
publishers	List of Strings	List of the publishers
price	Integer	Price in smallest unit of currency
currency	String	International acronym for currency
metacritic	Integer	Metacritic score www.metacritic.com
categories	Integer and String	Category ID and description (for example singleplayer, multiplayer)
genres	Integer and String	Genre ID and description (for example action, rpg)
recommendations	Integer	Number of reviews
release_date	String	Release date, localized
coming_soon	Boolean	Indicates whether game is already released

4.2.3 Review-Crawler

At the time of writing this thesis, an interface for the acquisition of review data was not available. Hence, the reviews were traditionally scraped utilizing Scrapy, "an open source and collaborative framework for extracting the data you need from websites" (Scrapy, 2018). Perunicic (2017) describes an elegant way of fetching all reviews. The scraper suggested by him was slightly adapted to support proxy usage, file-saving, and incremental scraping. For each game, all reviews were stored in a corresponding file due to the above mentioned reasons regarding the profile ID of a steam user. Since the scraping of all reviews can be very time consuming - the first run took over 21 days—an incremental scraping feature was implemented. For each game, the latest review in the database was fetched. The reviews on steam are ordered descending (meaning newer reviews are first) by their creation date, hence if the date of the scraped review was less than the latest stored date, the scraping process for that game was aborted. That feature reduces the scraping time significantly, depending on the execution cycle of the scraper. Table 4.4 gives an overview of the available data of one review.

Furthermore, each profile URL which was customized by a user, that is, has a username instead of a profile ID is stored in a temporary file which serves as an input for the next step.

4.2.4 Steam-ID-Crawler

Each review is uniquely associated to a user via the profile-URL. Every user has a URL to the profile page in the following format: `http://steamcommunity.com/profiles/USER_ID`. However, it is hard to remember it. Therefore, steam provides the possibility to add a custom profile URL which enables the user to make use of a self-chosen name (`http://steamcommunity.com/id/USERNAME`). At the time of writing this thesis, it was not possible to utilize the custom URL or the chosen username for other calls to the API which require some user identification (e.g., getting friends

Table 4.4: Review Data

Attribute	Data type	Description
product_id	Integer	ID of the application
recommended	Boolean	Is the game recommended?
date	String	Submission date
text	String	Review text
hours	Float	Time spent playing before writing review
profile_url	String	URL to the user's profile
products	Integer	Amount of products the user has
publishers	List of Strings	List of the publishers
found_funny	Integer	People who found the review funny
found_helpful	Integer	People who found the review helpful
found_unhelpful	Integer	People who found the review unhelpful

of a user, getting owned games). Thus it is necessary to fetch the profile ID for each custom URL. Since there is no special API interface available for that purpose, each custom profile URL has to be manually scraped for the user ID. The results are saved temporarily on a per-game-basis to files.

4.2.5 Database Conversion

The games obtained by the Games-Crawler are stored directly in a mySQL database with all corresponding information. Since the type of the steam application is not visible until an API call, the application type of non-games (see Table 4.2) is also stored. Therefore, they can be skipped for future scraping executions. Next, the obtained profile IDs are converted to the database. Hence, a direct association between a profile URL and the underlying profile ID is available. That association is crucial for the following step, the review conversion.

4.3 Data-Preprocessing

In order to get usable information, some preprocessing steps were necessary. Since this thesis focuses on solely video games, steam applications with another type were eliminated, thus leaving only applications with type "Game". Furthermore, every application which has an ID with a non-zero number at the last place, is also not of type game, although maybe otherwise stated. Since the data from the Game-Crawler is already in JSON-Line format, no HTML has to be removed. Most of the data can be directly transformed to Integers or Strings. Others, for example the publishers or developers, require some special attention since they are delivered as lists. Some data was considered as crucial (game ID, name, developers, publishers, description, categories and genres). If one of them was missing, the game was ignored. However, if, for example, the price was missing, it was replaced with appropriate values (0 in this case).

The raw data scraped by the Review-Crawler and Steam-ID-Crawler was in HTML format, therefore, as a first step, all HTML tags were removed and the relevant data was extracted using the scrapy framework. Further preprocessing steps included the conversion from strings to numbers, standardization of the review date, removing non-alpha-numerical characters and the simplification of the recommended field from "thumbs up" and "thumbs down" to boolean values.

Reviews do not get deleted if a user decides to remove the account from steam. Therefore, it is possible to have reviews which are not linked to an active profile anymore. Those reviews have been kept, as long as no custom URL was created by the user before.

4.4 Overview of the Dataset

In order to get a feeling for the crawled data, a thorough analysis was performed before any further action was carried out. The first scraping execution took about 26 days, mainly due to the restrictions described above.

Table 4.5 gives an overview of the whole dataset.

Table 4.5: Overview of the crawled dataset

	Amount
Applications	32,119
of type game	21,098
with at least 1 review	18,340
with at least 40 reviews	8,056
Users	3,918,378
Reviews	9,710,554
recommended	7,962,971
not recommended	1,747,583
Genres	42
Categories	29
Developers	14,007
Publisher	10,733
Total hours played	1,149,811,342

As depicted in Table 4.6, some outliers and impossibly considered values are present in the dataset. For example, the maximum play time a user spent playing a game before a review was given is 39,048.8 hours, that is roughly 4.5 years. The game in question is "Star Trek Online", which was released in early February 2010. 4.5 years therefore seems fairly unrealistic, especially regarding the review date, which was July 2016. Furthermore, 153,525.9 hours playtime for one player (that is 17.5 years) also seems very unrealistic, particularly looking at the fact that steam began collecting playtime data as recently as in 2009 (Valve Developer Community, 2018).

Other data do look quite promising, for example, 50 percent of the users played a game more than 16.5 hours before they reviewed a it. One could argue, that a playing time of two thirds of a day testifies for a profound opinion.

Regarding the quality of the review text, it seems as if reviews tend to be nearly twice as much useless than useful (5.91 vs. 3.54). One possible explanation for that phenomenon could be, that there are quite a lot of reviews with uninformative content like Martin Luther King's "I have a dream" speech, cooking recipes, or single words, multiple times repeated. This behavior might indicate the presence of so called "internet trolls" (Sobkowicz and Stokowiec, 2016) and are therefore subject to data cleaning.

Table 4.6: Detailed descriptive statistics of the dataset

	Average	Min	Q1	Median	Q3	Max
Hours played per player	316.1	0	11.7	54.5	248	153,525.9
Hours played before review was given	128.1	0	4.2	16.5	67.8	39,048.8
Hours played per game	68,759.3	0.1	29.8	188.7	1723.5	178,871,733.6
Positive ratings per player	2	0	1	1	2	979
Negative ratings per player	0.4	0	0	0	1	1,356
Positive ratings per game	441.6	0	6	22	109	230,271
Negative ratings per game	95.2	0	2	9	7	41,478
Ratings ratio per game	4.48	0	1	2.4	5	112
Usefulness of reviews	3.54	0	0	0	1	29,549
Uselessness of reviews	5.91	0	0	0	3	30,989
Length of text in reviews	320	0	40	115	329	39,396
Publishers per game	0.68	0	0	1	1	4
Developers per game	0.53	0	0	1	1	9
Genres per game	3.1	1	2	3	4	12

4.5 Recommendation Algorithms

In order to answer the research question stated in the introduction, three different recommender algorithms had been implemented. This section first gives an overview of the recommendation engine as a whole, followed by a detailed description of the algorithms and their modifications.

4.5.1 Overview

As a foundation for the whole recommendation engine three different (see Section 2.3) described, state-of-the-art recommender algorithms had been developed. To optimize the resulting list of games and to get a better live performance, the algorithms were slightly adapted and post-filtering techniques, similar to those stated by Adomavicius and Tuzhilin (2011), were deployed. Each one of the three algorithms produces a list of 10 games, ordered by their similarity to the input games (i.e., games, that the users noted they already played and enjoyed. Games that the users did not enjoy were neglected in this thesis). The introduced post-filters try to eliminate games which are deemed irrelevant in the given context indirectly, or they re-rank the games by adding their very own weighted similarity scores. Although a game could be ranked rather low with a standard algorithm, if it shares enough common attributes (see Section 4.7) with the input games or explicitly stated information, it may be moved even to the very top of the final list of recommendations. For example, if a user is looking for a game "similar to Dishonored or Bioshock?", both games as well as their predecessor and successor serve as input games for the recommender algorithms. Furthermore, the user also writes, that "I enjoy the stealth of Dishonored as well as the magic/weapon combat". For instance, the keywords "stealth", "magic combat" and "weapon combat" are explicitly stated and hence are provided to the post-filter responsible for explicitly stated tags. That filter creates a score for each game in the base list of recommendations between 1 and 0 and is added to the recommendation score (i.e., the similarity of a game to the input games).

Since the overall goal of the recommender engine is to fit the community taste of the reddit community, a closer look to some submissions on *r/gamingsuggestions* seemed appropriate to identify promising post-filtering approaches (see Section 4.7 for details).

Figure 4.1 depicts the whole recommendation process.

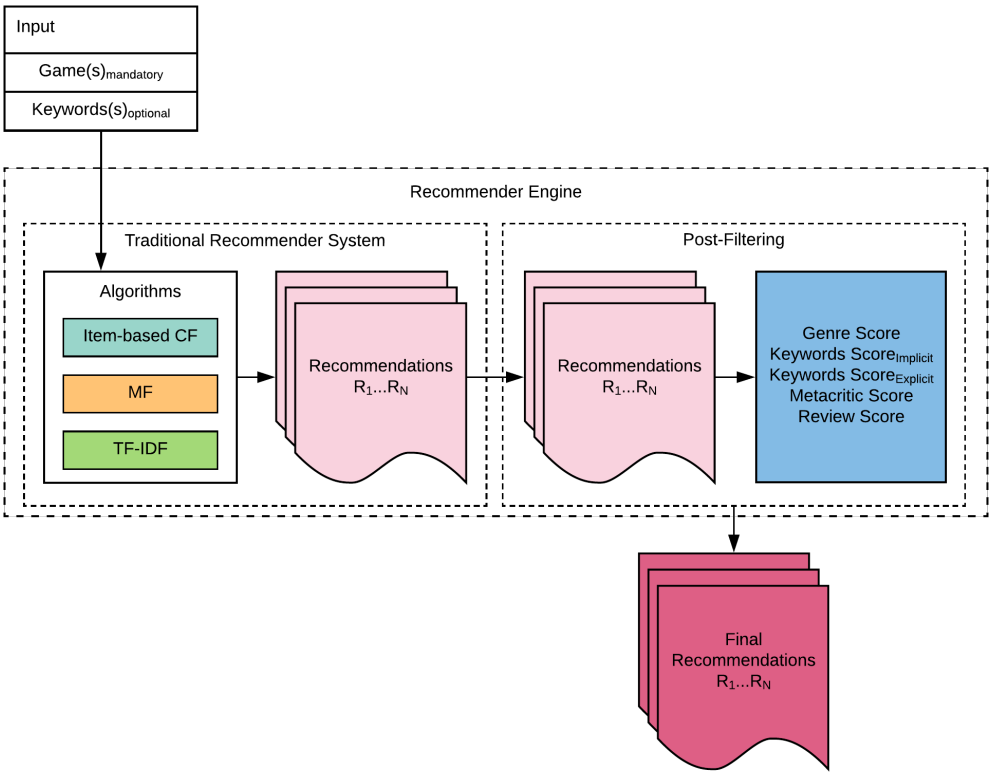

Figure 4.1: Depiction of the recommender engine.

4.5.2 Utility Matrix Generation

The utility matrix is composed of user-item pairs. Since the following experiments were executed with either a binary or a decimal rating scale, two utility matrices were created during this thesis. The binary version contained user-item pairs obtained from the reviews from steam, either with 1 (indicating a positive recommendation) or -1 (indicating a negative recommendation) in its cells. Blank cells indicate no rating for that user-item pair.
The decimal rating scale was derived from the playing time of a user for a game, hence it contains implicit ratings ranging from -5 to +5. The game times had been binned into 5 equally sized containers, differentiated for recommended or not recommended (as denoted by the review). Playing times below 0.1 hours for recommended reviews had been omitted. Table 4.7 gives an overview of the conversion limits.

Table 4.7: Implicit ratings conversion limits

Yes	≥115.1h	≥40.6h	≥18.4h	≥8.6h	≥0.1h
	5	4	3	2	1
No	≥36.5h	≥10.9h	≥3.9h	≥1.5h	≥0.1h
	-1	-2	-3	-4	-5

If a users played a game many hours and recommend it to others, it is assumed that they liked the game a lot, hence, the longer the playing time, the better the rating. Looking at the reviews which do not recommend the game, the relationship is inverse. If a user does not recommend a game and only needed a few minutes to come to that conclusion, a highly negative rating was deduced. However, if the game was not recommended, but it was still played over 36.5 hours, it was assumed that it is not all that bad.

Users with only one review provide very few information for the whole dataset, in fact, the only way they might influence the recommendations is by changing the average rating μ or the game bias b_i. Therefore, two thresholds were implemented. The best results were obtained by omitting users with less than 10 reviews and by excluding games with less than 40 reviews. Those thresholds were assessed by performing a grid search from

1 to 100 with steps of 10.

As pointed out by Desrosiers and Karypis (2011), users have their very own rating behavior, even if a rating scale is explicitly defined. For example, one user may be a hardcore gamer, who spends the main part of the day playing games. Naturally, the hours per game would be higher compared to a casual player. That phenomenon would still be present in the implicit utility matrix, hence, a normalization of the ratings was performed:

$$b_u = \bar{r}_u - \mu, \tag{4.1}$$

with b_u denoting the individual user bias, $\bar{r}_u$ the average rating of that user and μ the global average rating for all games.

Furthermore, some games require a higher playing time by their design, hence a game bias b_i for a game i with an average rating of $\bar{r}_i$ is introduced and defined as:

$$b_i = \bar{r}_i - \mu. \tag{4.2}$$

The centered utility matrix is therefore calculated by subtracting the global average rating μ, the corresponding user bias b_u and the corresponding game bias b_i from each rating:

$$z_{u,i} = r_{u,i} - \mu - b_u - b_i \tag{4.3}$$

The centering process was only applied to the implicit utility matrix.

4.5.3 Modified Collaborative Filtering

As illustrated in Section 2.3.1, the idea behind collaborative filtering is that similar users tend to like similar things. For this recommendation engine, an item-based approach was implemented. It is assumed, that the user who requests a recommendation is a new user. Hence, in order to recommend games, one has to know the k-most similar items to the positive games of the newly introduced user. As a prerequisite, the item-to-item similarities had been calculated from the utility-matrix for all game in the steam dataset. The similarity was calculated as proposed by Sarwar et al. (2001) with the

cosine-similarity. The similarity of two games i and j is therefore calculated as the cosine angle of the rating vectors of the two games g_i and g_j:

$$sim(i,j) = cos(g_i, g_j) = \frac{g_i \cdot g_j}{\|g_i\|_2 \|g_j\|_2}. \tag{4.4}$$

Applied to all different game-pairs, the similarity matrix $\boldsymbol{S}$ is defined as follows: $\boldsymbol{S} \in \mathbb{R}^{m \times m}$, where m is the number of games.

All input games the new user liked are treated as the best rating on the used rating scale. For each input game g_i, the most similar games $\vec{s_i} = (s_{i1}, s_{i2}, \ldots, s_{ik})$ are extracted from the similarity matrix $\boldsymbol{S}$. Summing up the individual similarities yields a accumulated similarity for each game. Consider the following example in Table 4.8:

Table 4.8: Similarities for input games 1 to 4

Input game	Game 1	Game 2	Game 3	Game 4
Game A	0.2	0	0	0.9
Game B	0.5	0.5	0.7	0
Game C	-0.1	0	0	0.4
Game D	0	0.3	0	0

From the above example, the following similarities can be obtained: 0.6 for Game 1, 0.8 for Game 2, 0.7 for Game 3 and 1.3 for Game 4. In that case, game 4 would be the highest ranked recommendation, game 1 the lowest. Those similarities are normalized between 0 and 1 with linear interpolation and build the baseline for the post-filtering.

4.5.4 Modified Matrix Factorization

Matrix factorization rests upon discovering latent factors based on the rating behavior of the users. Latent factors are attributes or characteristics of items which are not explicitly stated somewhere, that is, are hidden inside the data. Matrix factorization models became very popular in 2006 because of the Netflix Challenge (Ricci, Rokach, and Shapira, 2015). The theoretical

background was already covered in Section 2.3.2.
For this thesis, the singular value decomposition for sparse matrices (SVDS[8]) from the scipy package[9] was used. The matrices returned by this algorithm represent each user and each game in a latent factor space with f dimensions. A user u is therefore a vector $p_u \in \mathbb{R}^f$, and a game i is a vector $q_i \in \mathbb{R}^f$. The best results were obtained by setting $f = 110$. The amount of dimensions was found by trying values between 40 and 200, in steps of 10.

In order to recommend items, as described in section 4.5.3, the cosine similarity had been utilized. The similarity between two games hence is defined as:

$$sim(i,j) = cos(q_i, q_j) = \frac{q_i \cdot q_j}{\|q_i\|_2 \|q_j\|_2}, \tag{4.5}$$

with q_i and q_j as vectors, representing games in the f-dimensional feature space. Thus, only the matrix Q from the SVD is used for the similarity computation.

Again, each positive input game was considered to be rated with thumbs-up in the binary version of the utility matrix, a rating of +5 was assumed for the implicit one. Similar to the collaborative filtering approach, the individual similarities were summed up, resulting in an overall similarity score (see Table 4.8) for an example. This score was normalized as described before and serves as the baseline for the following post filters.

4.5.5 TF-IDF

Term Frequency - Inverse Document Frequency is a content-based approach which calculates similarities based on a textual corpus. In this thesis, the corpus of a game is composed of the name of the game, the description of the game and all reviews for that game. A closer look at the review text revealed some useless reviews with just letters or so called stopwords (such as “a”, “an”, “the”, “I”, “and”) with no information gain. The stopwords were excluded from the algorithm by providing a stopwords file for English

[8] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svds.html
[9] scipy.sparse.linalg.svds

and German words, arranged by Peter Graham[10]. Furthermore, review text was only considered in the TF-IDF-algorithm if it was between minimum 200 and maximum 7000 characters long. Those boundaries were obtained by the author of this thesis via the analysis of random samples of texts. The best results were obtained by limiting the maximum number of textual features to 900 (other tried values: 400, 800, 1000, 1200, 1500).
For calculating TF-IDF, the TF-IDF vectorizer from the scikit-learn package[11] was utilized. For a theoretical background, please refer to Section 2.3.3.
The TF-IDF for a term t in a document d is defined as

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t). \tag{4.6}$$

For the IDF calculation, a smoothed version is utilized with $df(d,t)$ denoting the document frequency of term t and n_d as the total number of documents:

$$\text{idf}(t) = \log \frac{1+n_d}{1+\text{df}(d,t)} + 1 \tag{4.7}$$

The resulting TF-IDF vectors are normalized with the Euclidean norm, to prevent a possible bias towards longer texts:

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} \tag{4.8}$$

The resulting matrix with TF-IDF vectors and the extracted features are calculated and stored locally, to speed up the recommendation process. In order to calculate the similarity between games, the input games are transformed to a single TF-IDF vector with the previously stored features as a vocabulary. As a text corpus, all names, descriptions and reviews from all input games were accumulated.
In order to recommend items, as described in Section 4.5.3, the cosine similarity had been utilized. The similarity between two games hence is defined as:

$$sim(i,j) = cos(t_i, t_j) = \frac{t_i \cdot t_j}{\|t_i\|_2 \left\|t_j\right\|_2}, \tag{4.9}$$

[10] https://github.com/6/stopwords-json
[11] http://scikit-learn.org/stable/index.html

t_i denoting the tf-idf-vector of game i and t_j as the tf-idf-vector of game j. This similarity score was normalized as described before and serves as the baseline for the following post filters.

4.6 Evaluation

The algorithms and post-filters were evaluated in two ways: First, with testcases obtained from the subreddit *r/gamingsuggestions* and second, a qualitative evaluation of the recommendations was performed. The detailed steps for both of this evaluation approaches are discussed in this section.

4.6.1 Testcases

Since the overall goal of the recommendation engine is to fit the community taste, a closer look to a gaming community seemed appropriate. For that purpose, the subreddit *r/gamingsuggestions* was inspected. According to the description, the subreddit is about

> "To give and get advice on games, gaming equipment, consoles, hardware, software, board games, gaming books, paraphernalia- anything gaming related."

In total, 100 testcases were obtained. However, in order to provide a fair environment and reproducibility, some restrictions apply:

- Submissions must contain the words "games like" or "game like" in the title
- There had to be a least one game provided as an example which is also available on steam
- Altogether, a minimum of 10 different games had to be suggested by the community, each of them available on steam

- A suggestion for a game was regarded as good, if at least three upvotes were given for the whole comment. This limit seemed appropriate in the author's view.
- A suggestion for a game was regarded as good, if requester said they would "definitely check this out", or they are "going to buy that" or stated otherwise that the suggestion was good.

To better illustrate the very nature of the requests, in Figure 4.2 some examples of the requests with to some extent very special requirements are depicted.

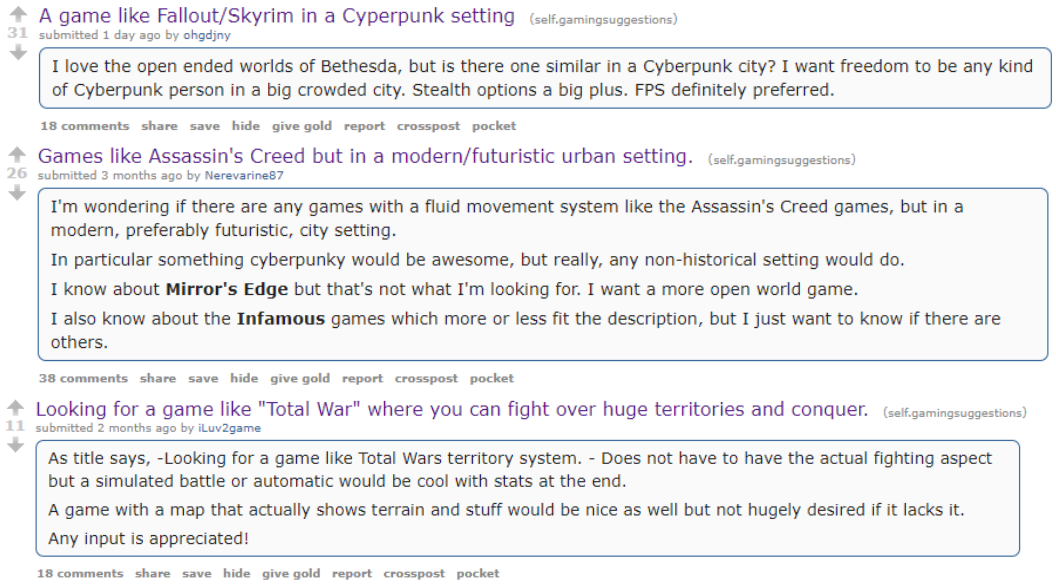

Figure 4.2: Example of some typical requests from users of */r/gamingsuggestions*. Note the very detailed and specialized constraints. Some of the asked attributes can only be complied with if the recommender actually played the game.

For each submission, the following steps were executed:

- For each game mentioned in the title or description of the submission, it was manually checked whether the game is available on steam, and if yes, the appropriate steamID was obtained
- Each comment was manually checked if it was either approved as a good suggestion by the requester, or if it had enough upvotes from the community
- For each viable comment, the games were manually checked for their availability on steam and the corresponding steamID was fetched
- Steam provides twelve similar games (presumably based on tags) on every game detail page (see Figure 4.3). Hence, for every input game, all twelve steam recommendations were gathered
- Each submission was manually searched for informative tags

Table 4.9 gives an overview of the testcases.

Table 4.9: Overview of the obtained testcases

	Amount
Testcases	100
Different Input Games	408
Different Community Recommendations	809
Different Steam Recommendations	1640
Average amount of community recommendations per submission	14.66
Average amount of input games per submission	7.8
Average text length of submission	473
Average tags per submission	2.96

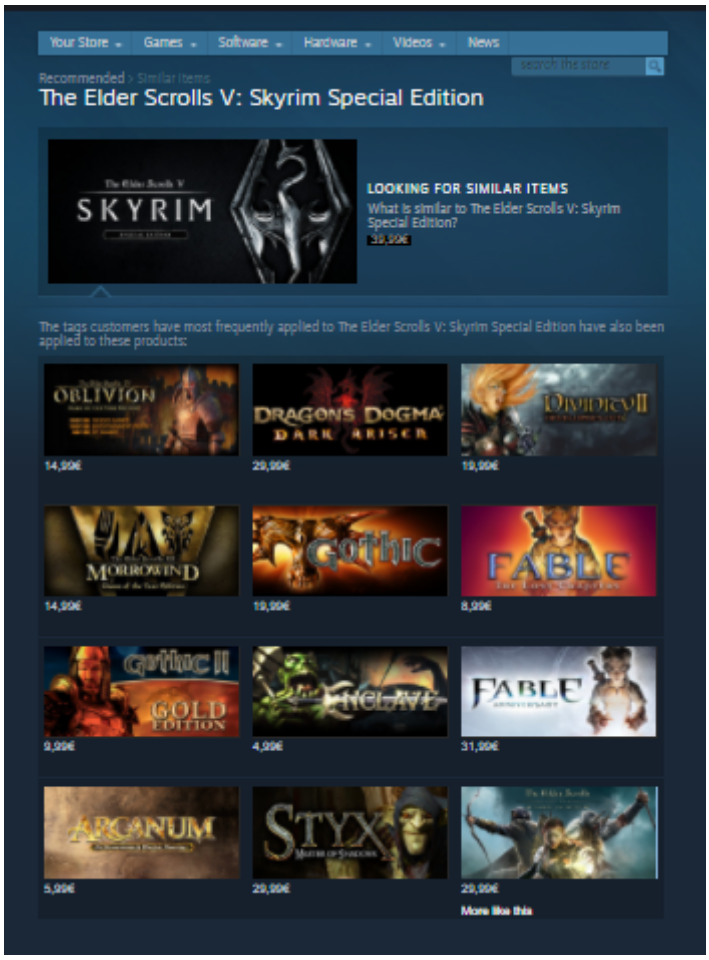

Figure 4.3: Steam recommends twelve similar games for each game on their website

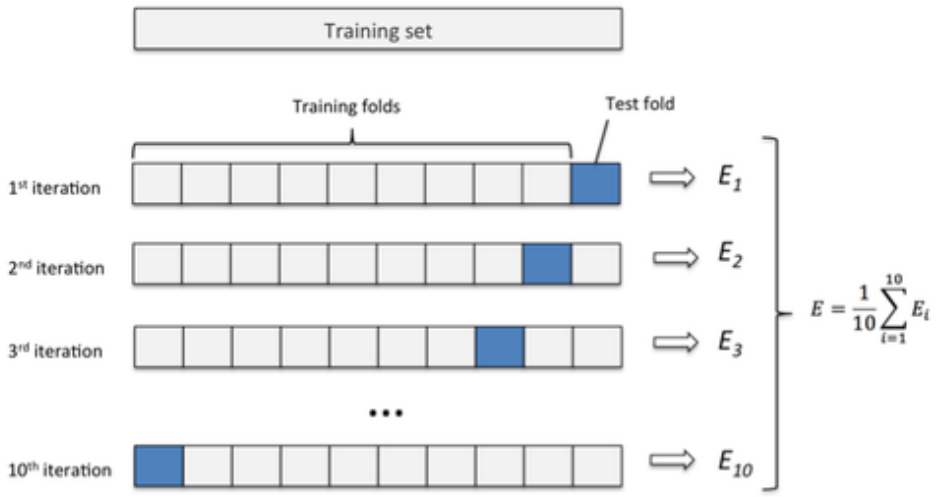

Figure 4.4: Illustration of the 10-fold-cross-validation (Raschka, 2018). Each fold creates a training set consisting of 90 testcases, and a testset containing ten testcases. For each fold, the filter weights were trained separately with a grid search experiment. The performance metrics such as precision, recall and f1-score were obtained from the previously unseen testset for each fold.

To comply with Salzberg (1997), who stated that "everything done to modify or train the algorithm has to be done in advance of seeing the test set", a 10-fold-cross-validation similar to those proposed by (Amatriain et al., 2011; Cremonesi and Turrin, 2009) was performed. In that way, the algorithms were trained on a training set consisting of 90 testcases each, and were tested on a test set of 10 testcases for each fold. By creating ten different training-testset-pairs, the limitation of only having 100 testcases is also mitigated to some extent.

In order to compare the recommendations of the algorithms to suggestions given by the community, typical performance metrics such as precision, recall and f1-score @N (with N=10) (see section 2.1.4) were utilized.

As a baseline, the steam recommendations were used. Since there are twelve games per input game, there could be more than ten different games in the resulting list. Steam does not seem to order the recommendations in any

particular way, hence, all the steam recommendations had been randomized and ten were chosen. In this thesis, the different folds were separated by the submission time stamp.

4.6.2 Qualitative Evaluation

The goal of the recommendation engine was to fit the community taste. However, the community might be biased in various ways. For example, people from the US might recommend other games compared to people from Asian countries. Older people could be recommending games from their childhood a bit more than recent games. A gender bias might also be present. Even if the community taste was not matched in some of the testcases mentioned above, the recommendations might still be viable.

To test that hypothesis, a qualitative evaluation was performed.

For each submission, the recommendations from the community, from each of the three algorithms and the games recommended by steam were merged together into a single list of maximum 50 games (ten games per approach). Duplicate games were only contained once in the list, hence the lists were usually shorter.

Each game in such a list could be rated with four different values, depending on whether the game matches the submission or not. In order to obfuscate the origin of the recommendation, the list was sorted alphabetically. Furthermore, since it was assumed that most of the games are unknown to the user, each game was linked with the corresponding steam website for detailed gameplay videos, descriptions, and screenshots. Moreover, the associated genres and tags were displayed beneath the game title directly.

The task of each evaluator was to measure the fit of the recommended game with the submission. Three ratings had been established, namely *good match*, *ok match* and *bad match / no match*. The fourth option was *?*, which could be checked if no sound opinion could be given.

The questionnaire was personalized, meaning every person got an individual link which enabled a pausing of the survey. A detailed task description was also sent with the link. Figure 4.5 shows the rating page and Figure 4.6 shows the possible options in detail.

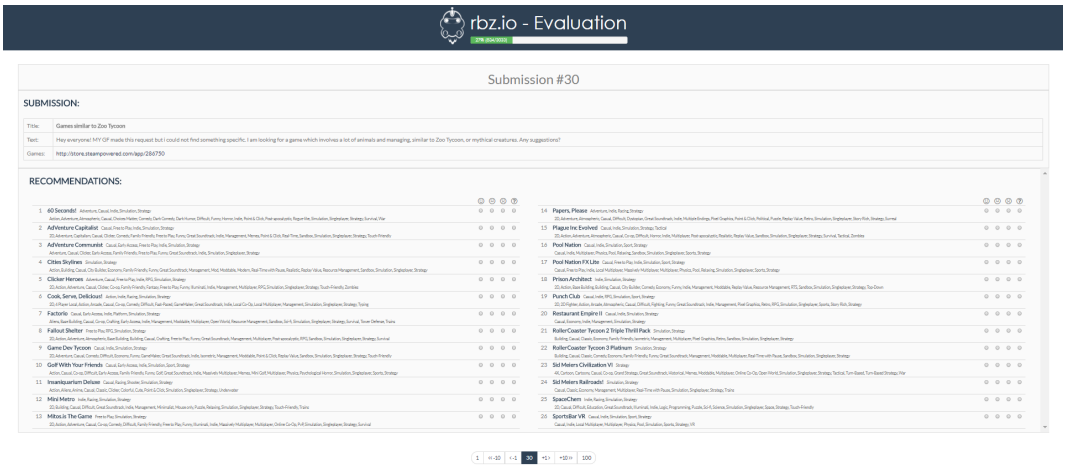

Figure 4.5: Overview of the rating page. At the top, the submission from reddit is displayed. Below, links to the games mentioned in the description are displayed in case the games are unknown to the evaluator. Further below, the alphabetically sorted list of recommendations is shown.

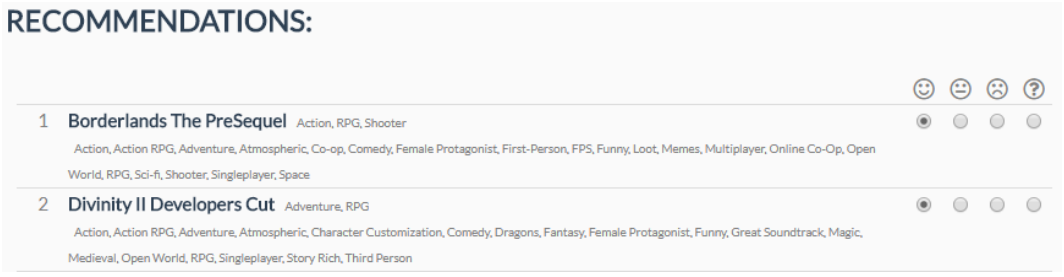

Figure 4.6: Detailed view of recommendations.

The survey was done by 20 different persons, varying in age (between 24 and 56), gender, gaming experience and education level.

4.7 Post-Filtering

In order to improve the recommendations from the three state-of-the-art algorithms, post filters similar to those advocated by Adomavicius and Tuzhilin (2011), were implemented.

Such post filters alter the list of recommendations given by the algorithms by removing irrelevant games or reordering the list by taking contextual information into account. For this thesis, only *soft* filters (filters without strict boundaries) were considered.

Since it is not viable to compute all post-filters for all games, the list of recommended games was truncated. However, to preserve the natural concept of each algorithm, the list length was set at 300 due to performance issues, however, other list lengths may be possible. All further improvements were made on that shortened list.

Since only soft post filters were implemented, a removing from the final list is therefore done implicitly by reordering the games.
Each game in the list of recommendations has a similarity score to the positive input games given by the user between 1 and 0. 1 indicating a rather good recommendation and 0 specifying a bad one (by the definition of the algorithms).

For each game in the list of 300 and for each post filter, a post filter score was computed. The score was then added to the recommendation score with a weighting influence. As a last step, the recommendatin list was reordered. By manually investigating the subreddit *r/gamingsuggestions*, the post-filters in the following subsections emerged.

4.7.1 Genre Score

By looking at the games which were stated as positive input games and the resulting recommendations of the community, it was noticed, that the genres of the games overlap in most cases. For example, the requesters specify *The Witcher* and *The Elder Scrolls V: Skyrim* as games they liked and the community recommends *Dragon Age: Origins*, all three games belong to the genre *RPG - Role Playing Game*. Hence, the implicit genre score S_{genre} was implemented. It is formally defined as

$$S_{genre}(i) = \sum_{j \in I_{games}} \frac{\left|G_i \cap G_j\right|^2}{|G_i| \, |G_j|}, \tag{4.10}$$

with I_{games} denoting the input games and G_i and G_j the associated genres of game i and j. The idea behind the implicit Genre Score is, that games which have the same set of genres should be ranked higher in the final list of recommendations.

4.7.2 Tag Score

A tag is a keyword which describes an item. By looking at the different submissions on *r/gamingsuggestions*, it became clear, that for most submissions explicit tags could be found. Hence, two tag scores are introduced:

The explicit tag score and the implicit tag score.

The explicit tag score S_{eTags} measures the overlap between the explicitly stated tags in the submission and the associated tags of each game. It is formally defined as

$$S_{eTags}(i) = \frac{\left|T_i \cap I_{tags}\right|^2}{|T_i| \, |I_{tags}|}, \tag{4.11}$$

T_i denoting the tags linked to each game and I_{tags} as the set of explicitly stated tags.

The implicit tag score S_{iTags} measures the overlap between the associated tags of the input games with the tags linked to the recommended games. It is formally defined as

$$S_{iTags}(i) = \sum_{j \in I_{games}} \frac{\left|T_i \cap T_j\right|^2}{\left|T_i\right| \left|T_j\right|}, \tag{4.12}$$

with I_{games} denoting the input games and T_i and T_j the affiliated tags of game i and j.

4.7.3 Metacritics Score

Metacritic is a rating aggregation service for different kinds of media products. It combines the opinion of most respected critics writing online and in print to a single number (Metacritics, 2018).

The metacritic score weighs games which were rated good a bit more, than games which got bad ratings. This way, qualitative games are ranked a bit higher than so called "trash games". Overall, games tend to be more recommended if their quality is higher than average.

Since the metacritic score was not available for some games, the average of all games was calculated and assigned to the missing values.

More formally, the score $S_{metacritic}$ for each game is defined as

$$S_{metacritic}(i) = 1 + \frac{M_i - \max(M)}{\max(M) - \min(M)}, \tag{4.13}$$

with M denoting the set of metacritic scores for all 300 recommendations and M_i the metacritic score for game i. The resulting list of metacritic scores for all recommendations was normalized between 1 and 0, matching the best metacritic score to 1 and the worst to 0 and interpolating the values between.

4.7.4 Review Score

A similar approach was followed regarding the review score. It was presumed, that games which have more total reviews tend to be more popular, regardless whether the reviews were good or bad in total, following the slogan "there is no such thing as bad publicity". Formally, the review score S_{review} is defined as

$$S_{review}(i) = 1 + \frac{|R_i| - |\max(R)|}{|\max(R)| - |\min(R)|}, \tag{4.14}$$

with R denoting amount of given reviews for each of the 300 recommendations and R_i the amount of reviews for game i. Again, the score was normalized between 1 and 0.

4.7.5 Category, Developer, Publisher Score

Other scores including the categories, publishers and developers of the games similar to those mentioned above had also been considered. However, no significant improvements could be achieved, hence they were abandoned in the final recommendation engine.

4.7.6 Weighting the Scores

Each of the above scores was added to original recommendation list by simply summing up the similarity values for each game with the score values for each game. Some of the scores are more important than others, hence they should be taken into account more. This is achieved by assigning each score a weighting factor between 1 and 0, depending on its importance. The different weights were obtained by a grid search experiment. Each score was evaluated with values between 0 and 1, in 0.2 steps. The obtained values where then again tested with -0.1 and +0.1. For example, if grid search revealed a value of 0.4, 0.3 and 0.5 also had been evaluated.

4.8 Testing

Python provides a unittest framework[12], which was utilized for most of the tests.
As proposed by Reitz (2018), the general rules of testing had been complied with. Hence, testing units were kept small to focus on tiny bits of functionality at a time. Furthermore, all testcases were executed before committing changes to the repository. After writing some code, the modules in question were also tested in regular intervals.

During the implementation work of this thesis, a test-driven development approach as advocated by Beck (2002) was followed. Therefore, short iterations were possible with clean and sleek code.

Testcases for functions like, for example, a helper function that normalizes values between 1 and 0 had been very straight forward to implement. Naturally, valid and invalid input had been tested.

Testing the scraper and the recommendation algorithms proved to be more complex.

Since the scraper relies on the existence of HTML sites, a local web server instance was set up containing a fully downloaded game page from steam. The reference output was manually defined by analyzing the downloaded web page. Furthermore, an URL parameter was added to simulate various HTTP errors. That way, for example, a 404-error or a 429-error[13] could be simulated.

In order to test the recommendation algorithms, a test database environment with some sample data was created as a prerequisite before each test run. The results of the algorithms had been calculated manually and served as a comparison to the implemented ones.

[12] https://docs.python.org/3/library/unittest.html
[13] Too Many Requests

5 Results

This chapter presents the results obtained in order to answer the research questions stated in Section 1. First, the outcome of the goal *fitting the community taste* is presented. Second, the results of the qualitative evaluation is described.

5.1 Fitting the Community Taste

In order to measure the overlap between the games suggested by the algorithms and the games recommended by the community, typical metrics (see Section 2.1.4) had been utilized.
Details about the single steps can be found in section 4.6.

5.1.1 Post-Filtering Weights

Following, the different weights for each applied post-filtering approach are depicted. The weights were calculated for each fold individually with a training set. The grid search experiment revealed the same weights for each fold, hence only one weight per approach is displayed.

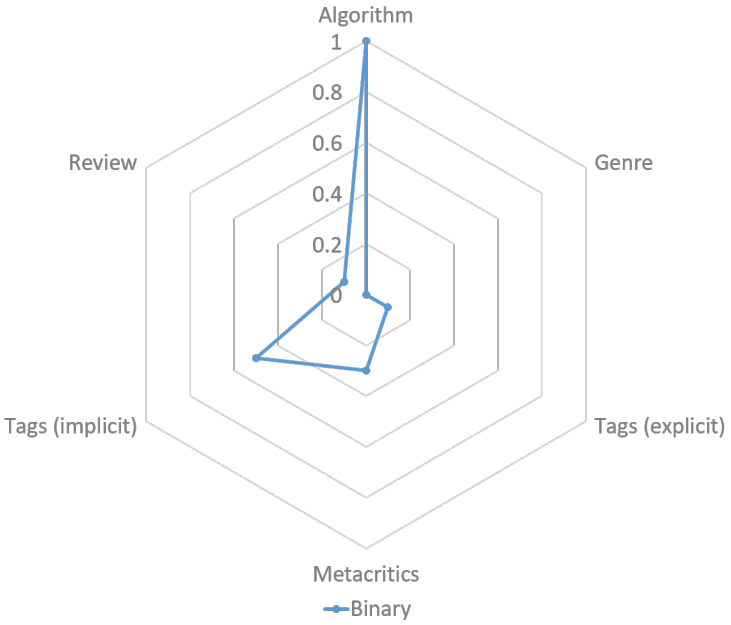

Figure 5.1: Weights for the various post-filtering approaches, obtained from the binary training sets.

Table 5.1: Weights for the different post-filtering approaches obtained from the binary training sets.

Filter	Weight
Genre	0
Tags (Explicit)	0.1
Tags (Implicit)	0.5
Metacritics	0.3
Review	0.1

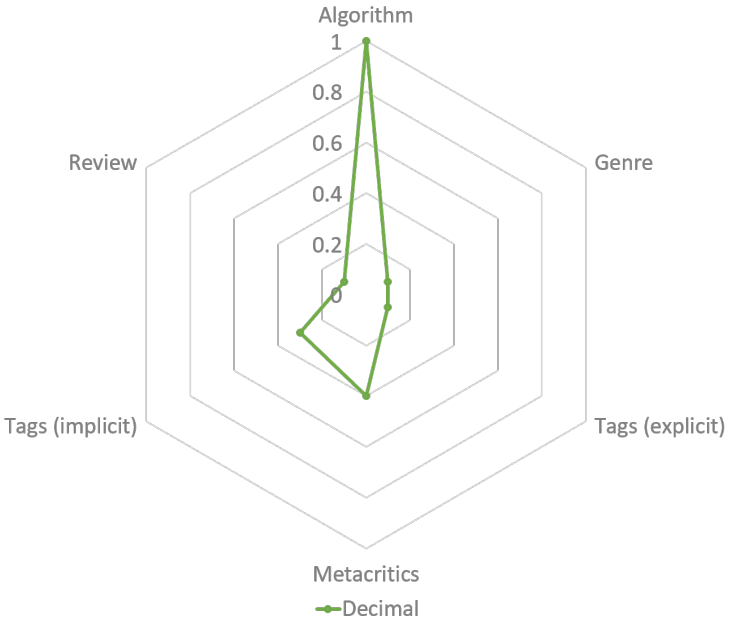

Figure 5.2: Weights for the various post-filtering approaches, obtained from the decimal training sets.

Table 5.2: Weights for the different post-filtering approaches obtained from the decimal training sets.

Filter	Weight
Genre	0.1
Tags (Explicit)	0.1
Tags (Implicit)	0.3
Metacritics	0.4
Review	0.1

5.1.2 Test Results

Following, the results of the test sets are depicted for each algorithmic approach and both rating variants (i.e., the binary, explicit rating and the decimal, implicit rating obtained from the playing time). The last section contains a summary of the test folds for each algorithm.

Binary Item-Based CF

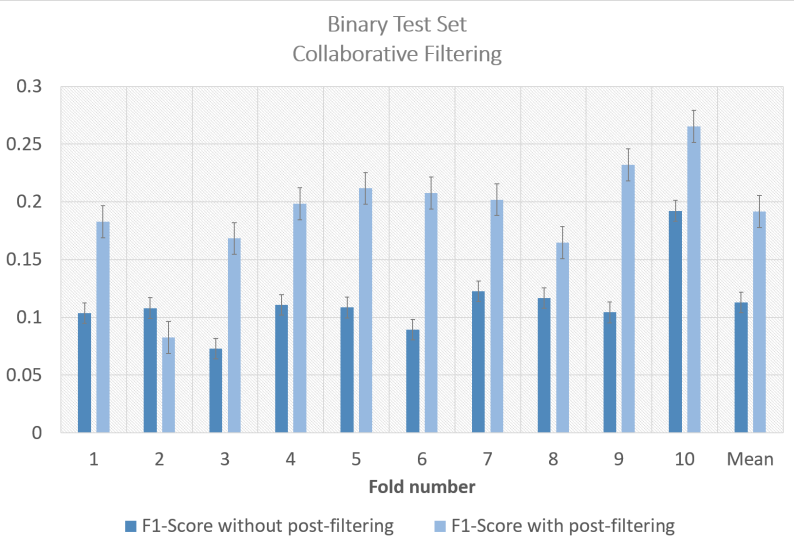

Figure 5.3: F1-Scores with item-based CF on the binary test sets obtained by 10-fold cross validation. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.3: Precision, recall and F1-Score of the binary test sets (item-based CF) without any post-filtering techniques.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.11	0.12	0.08	0.12	0.14	0.11	0.18	0.14	0.13	0.23	0.136
Recall	0.099	0.099	0.068	0.104	0.09	0.079	0.094	0.103	0.091	0.168	0.1
F1-Score	0.104	0.108	0.073	0.111	0.109	0.089	0.123	0.117	0.104	0.192	0.113

Table 5.4: Precision, recall and F1-Score of the binary test sets (item-based CF) with post-filtering techniques applied.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.19	0.09	0.18	0.22	0.26	0.27	0.31	0.21	0.28	0.33	0.234
Recall	0.177	0.077	0.159	0.184	0.183	0.177	0.151	0.139	0.204	0.228	0.168
F1-Score	0.183	0.083	0.168	0.198	0.212	0.208	0.202	0.165	0.232	0.265	0.192

Decimal Item-Based CF

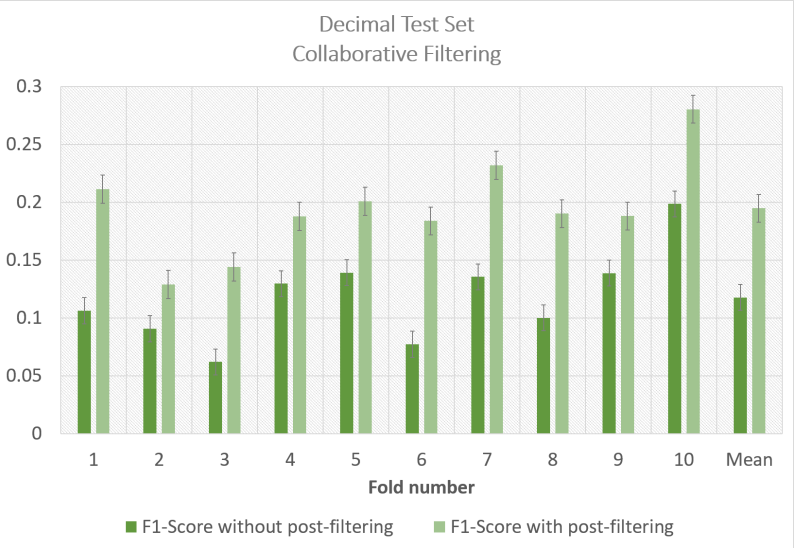

Figure 5.4: F1-Scores with item-based CF on the decimal test sets obtained by 10-fold cross validation. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.5: Precision, recall and F1-Score of the decimal test sets (item-based CF) without any post-filtering techniques.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.11	0.1	0.07	0.14	0.17	0.09	0.21	0.12	0.17	0.24	0.142
Recall	0.103	0.084	0.056	0.122	0.12	0.07	0.101	0.088	0.122	0.174	0.104
F1-Score	0.106	0.091	0.062	0.13	0.139	0.077	0.136	0.1	0.139	0.199	0.118

Table 5.6: Precision, recall and F1-Score of the decimal test sets (item-based CF) with post-filtering techniques applied.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.22	0.14	0.15	0.21	0.25	0.24	0.35	0.24	0.23	0.35	0.238
Recall	0.204	0.12	0.139	0.172	0.172	0.156	0.175	0.162	0.164	0.24	0.17
F1-Score	0.211	0.129	0.144	0.188	0.201	0.184	0.232	0.19	0.188	0.28	0.195

Binary MF

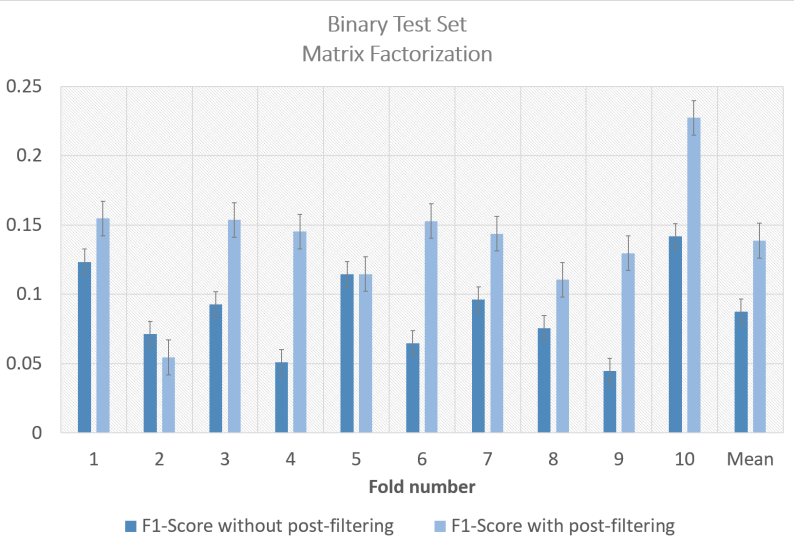

Figure 5.5: F1-Scores with matrix factorization on the binary test sets obtained by 10-fold cross validation. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.7: Precision, recall and F1-Score of the binary test sets (MF) without any post-filtering techniques.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.13	0.08	0.1	0.06	0.14	0.08	0.14	0.1	0.06	0.17	0.106
Recall	0.118	0.065	0.087	0.046	0.098	0.055	0.075	0.062	0.037	0.124	0.077
F1-Score	0.123	0.071	0.093	0.051	0.115	0.065	0.096	0.076	0.044	0.142	0.088

Table 5.8: Precision, recall and F1-Score of the binary test sets (MF) with post-filtering techniques applied.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.16	0.06	0.16	0.16	0.15	0.2	0.22	0.14	0.16	0.28	0.169
Recall	0.15	0.05	0.148	0.135	0.095	0.128	0.108	0.093	0.112	0.197	0.122
F1-Score	0.155	0.054	0.154	0.145	0.115	0.153	0.144	0.111	0.13	0.227	0.139

Decimal MF

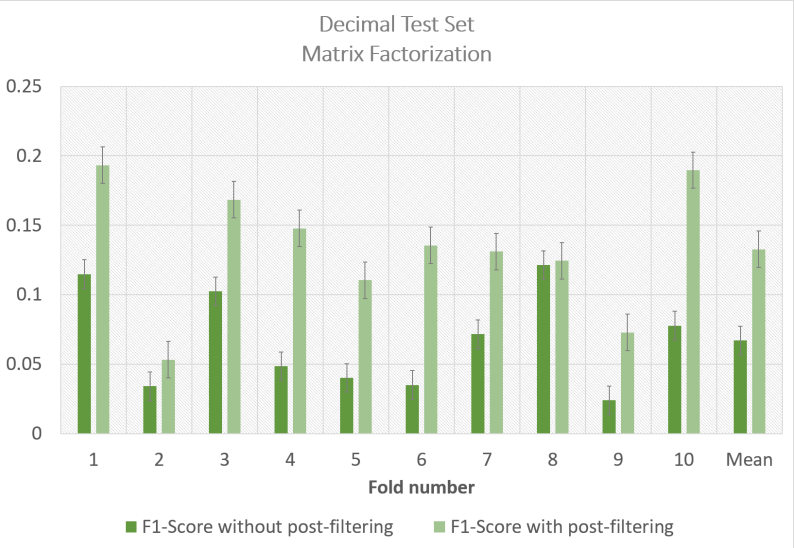

Figure 5.6: F1-Scores with matrix factorization on the decimal test sets obtained by 10-fold cross validation. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.9: Precision, recall and F1-Score of the decimal test sets (MF) without any post-filtering techniques.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.12	0.04	0.11	0.05	0.05	0.05	0.1	0.17	0.03	0.09	0.081
Recall	0.111	0.03	0.097	0.047	0.034	0.028	0.057	0.097	0.02	0.069	0.059
F1-Score	0.115	0.034	0.103	0.049	0.04	0.035	0.072	0.121	0.024	0.078	0.067

Table 5.10: Precision, recall and F1-Score of the decimal test sets (MF) with post-filtering techniques applied.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.2	0.06	0.18	0.16	0.15	0.17	0.2	0.17	0.09	0.22	0.16
Recall	0.188	0.048	0.159	0.139	0.089	0.118	0.098	0.101	0.062	0.169	0.117
F1-Score	0.193	0.053	0.168	0.148	0.11	0.135	0.131	0.124	0.073	0.19	0.133

Binary TF-IDF

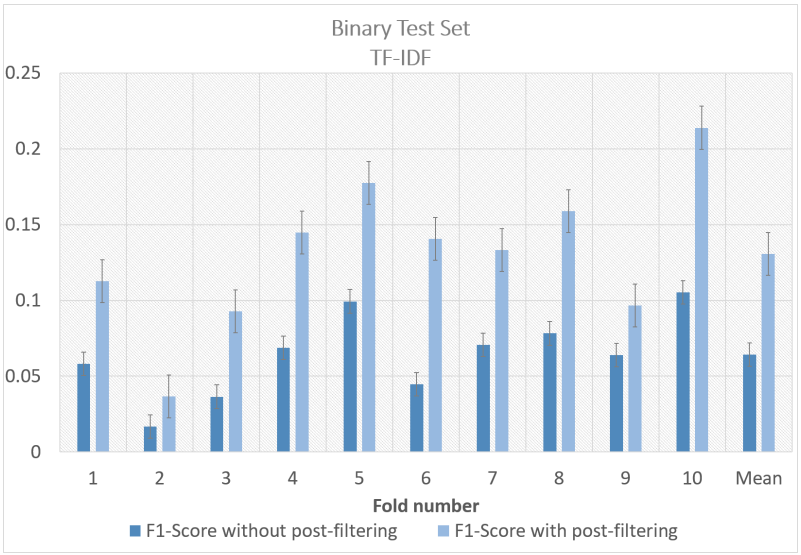

Figure 5.7: F1-Scores with TF-IDF on the binary test sets obtained by 10-fold cross validation. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.11: Precision, recall and F1-Score of the binary test sets (TF-IDF) without any post-filtering techniques.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.06	0.02	0.04	0.07	0.12	0.05	0.11	0.1	0.08	0.13	0.078
Recall	0.057	0.014	0.034	0.068	0.087	0.041	0.052	0.067	0.054	0.091	0.056
F1-Score	0.058	0.017	0.036	0.069	0.099	0.045	0.071	0.078	0.064	0.105	0.064

Table 5.12: Precision, recall and F1-Score of the binary test sets (TF-IDF) with post-filtering techniques applied.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.12	0.04	0.1	0.15	0.22	0.17	0.2	0.21	0.12	0.27	0.16
Recall	0.107	0.034	0.088	0.14	0.153	0.123	0.101	0.132	0.081	0.182	0.114
F1-Score	0.113	0.037	0.093	0.145	0.178	0.141	0.133	0.159	0.097	0.214	0.131

Decimal TF-IDF

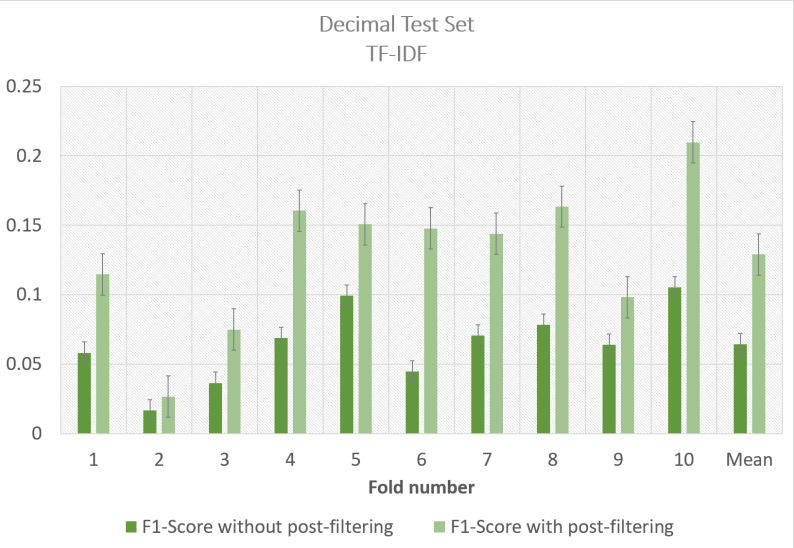

Figure 5.8: F1-Scores with TF-IDF on the decimal test sets obtained by 10-fold cross validation. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.13: Precision, recall and F1-Score of the decimal test sets (TF-IDF) without any post-filtering techniques.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.06	0.02	0.04	0.07	0.12	0.05	0.11	0.1	0.08	0.13	0.078
Recall	0.057	0.014	0.034	0.068	0.087	0.041	0.052	0.067	0.054	0.091	0.056
F1-Score	0.058	0.017	0.036	0.069	0.099	0.045	0.071	0.078	0.064	0.105	0.064

Table 5.14: Precision, recall and F1-Score of the decimal test sets (TF-IDF) with post-filtering techniques applied.

	Fold Number										
	1	2	3	4	5	6	7	8	9	10	Mean
Precision	0.12	0.03	0.08	0.17	0.19	0.18	0.21	0.21	0.12	0.26	0.157
Recall	0.11	0.024	0.071	0.154	0.128	0.129	0.111	0.138	0.084	0.18	0.113
F1-Score	0.115	0.027	0.075	0.161	0.151	0.148	0.144	0.163	0.098	0.21	0.129

Summary

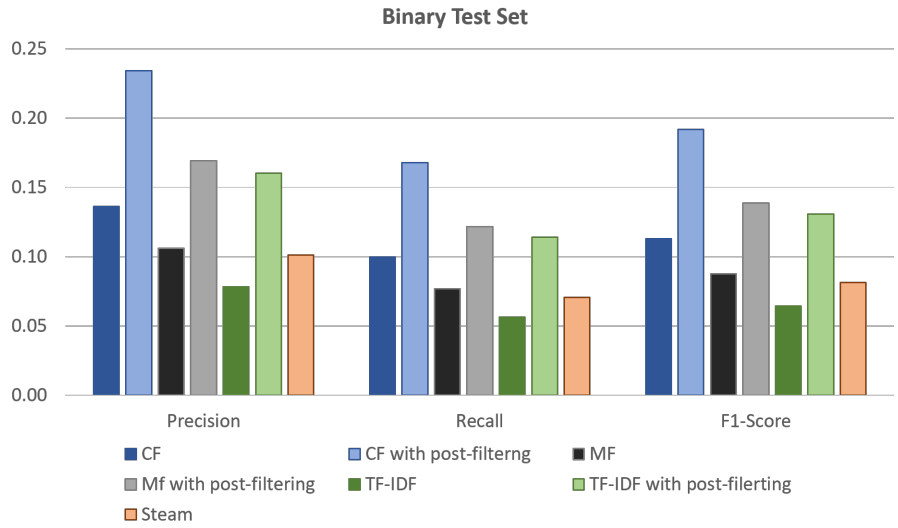

Figure 5.9: Average precision, recall and F1-Score of each algorithm on the binary dataset. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.15: Average precision, recall and F1-Score of each algorithm on the binary dataset without post-filtering.

	CF	MF	TF-IDF	Steam
Precision	0.136	0.106	0.78	0.101
Recall	0.1	0.077	0.056	0.071
F1-Score	0.113	0.088	0.064	0.081

Table 5.16: Average precision, recall and F1-Score of each algorithm on the binary test set with post-filtering techniques applied.

	CF	MF	TF-IDF	Steam
Precision	0.234	0.169	0.16	0.101
Recall	0.168	0.122	0.114	0.071
F1-Score	0.192	0.139	0.131	0.081

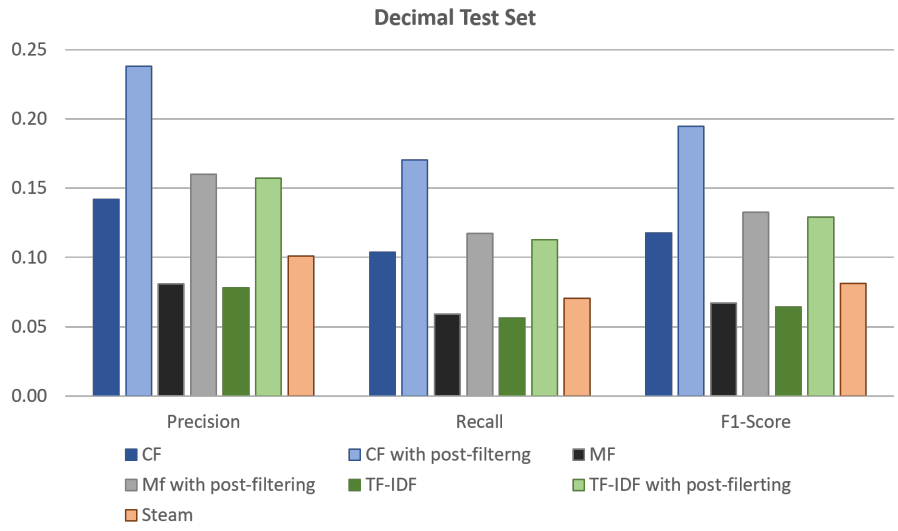

Figure 5.10: Average precision, recall and F1-Score of each algorithm on the decimal dataset. The darker columns represent the score before any post-filtering, the lighter ones after post-filtering techniques applied.

Table 5.17: Average precision, recall and F1-Score of each algorithm on the decimal dataset without post-filtering.

	CF	MF	TF-IDF	Steam
Precision	0.142	0.081	0.78	0.101
Recall	0.104	0.059	0.056	0.071
F1-Score	0.118	0.067	0.064	0.081

Table 5.18: Average precision, recall and F1-Score of each algorithm on the decimal dataset with post-filtering techniques applied.

	CF	MF	TF-IDF	Steam
Precision	0.238	0.16	0.157	0.101
Recall	0.170	0.117	0.113	0.071
F1-Score	0.195	0.133	0.129	0.081

5.2 Qualitative Evaluation

Following, the results of the qualitative Evaluation described in Section 4.6.2 are depicted.

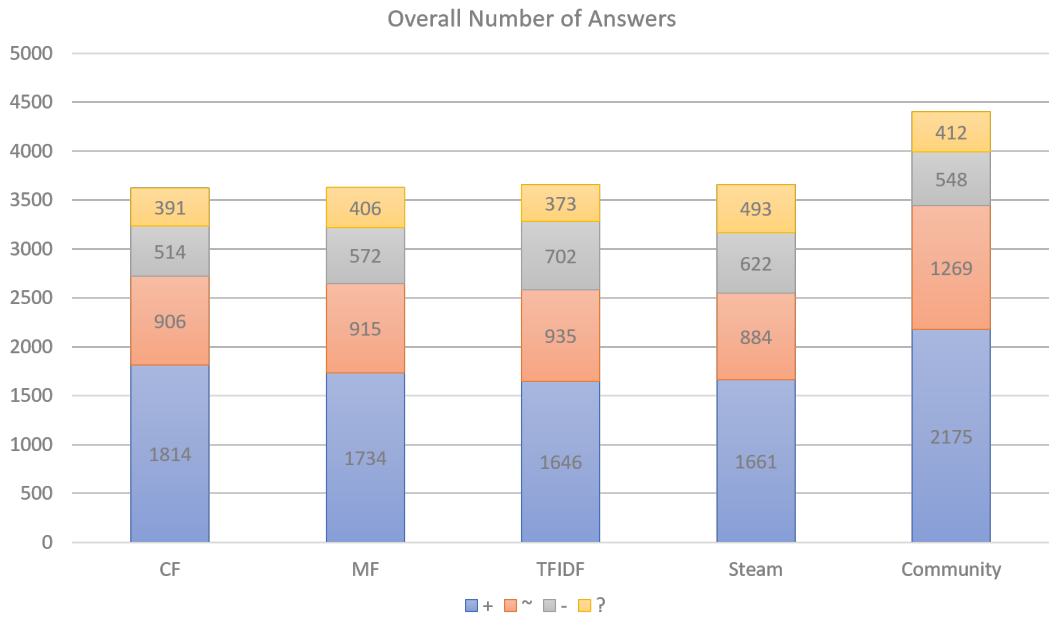

Figure 5.11: Absolute numbers of given answers for each approach and option (good, neutral, bad, unvalued).

Since there have been at least ten community suggestions per submission, the number of possible answers was a bit higher than the numbers of recommendations generated by the algorithms (which were limited to maximum ten). On average, 14.6 games were suggested by the community.

Table 5.19: Number of total answers per approach

	CF	MF	TF-IDF	Steam	Community
Overall answers, including unvalued	3625	3627	3656	3660	4404
Overall answers, without unvalued	3234	3221	3283	3167	3992

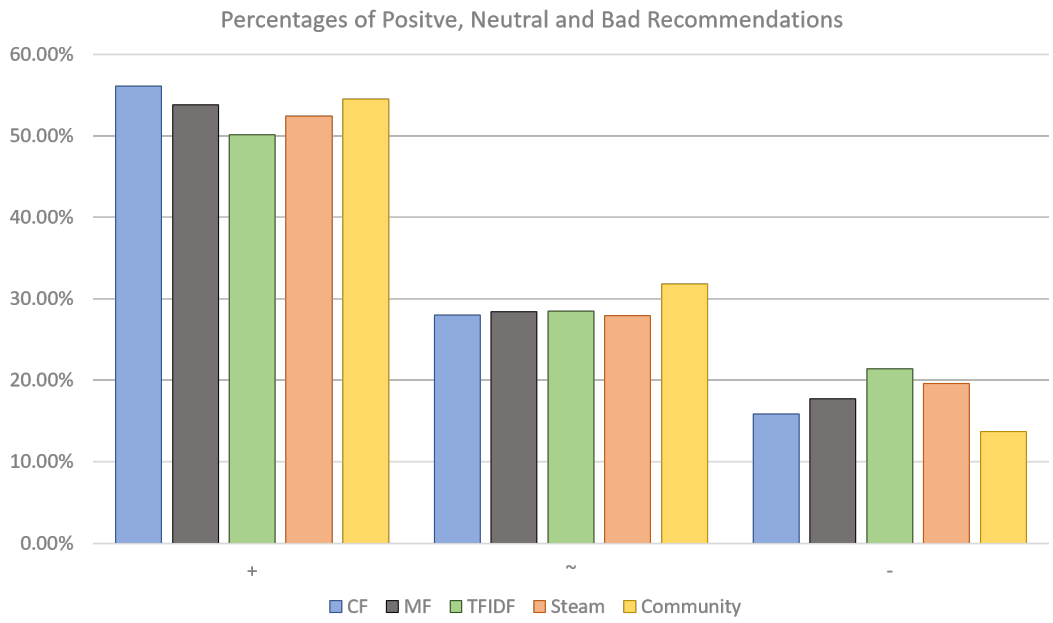

Figure 5.12: Percentages of given answers.

In figure 5.12, the number of answers normalized by the total number of answers is depicted.

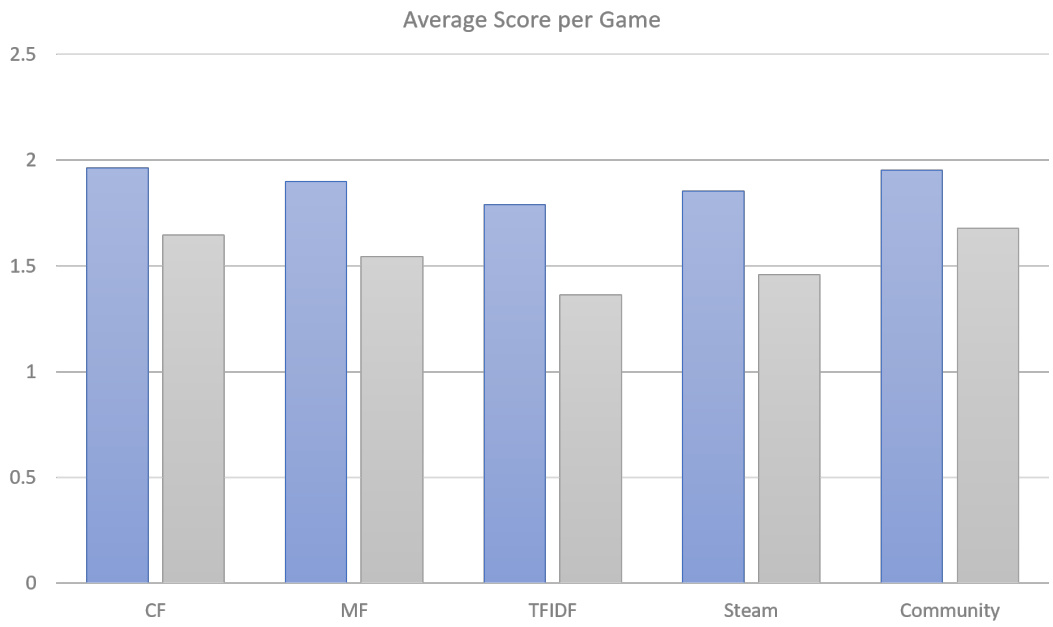

Figure 5.13: Average qualitative scores per game. The blue columns represent only good and neutral ratings, the grey columns are with a malus for negative ratings.

The average score is calculated as follows: Good recommendations receive 3 points, neutral recommendations 1 point. For each bad recommendation, -2 points are assigned. The numbers are summed up and divided by the total amount of answers.

6 Discussion

In this thesis, a recommendation engine for video games, with the goal of fitting the community taste of reddit users was implemented. More specifically, video game data from the steam platform was utilized to build three state-of-the-art recommendation algorithms. The output of the algorithms was compared to the suggestions provided by the community and the overlap was analyzed. However, with submissions obtained from reddit, the algorithms revealed great potential for improvement (see Section 5.1.2, especially Table 5.17 and Table 5.15). Regarding the matching of the community taste, the algorithmic approaches delivered results which were subject to fine tuning with post-filtering techniques.
Research question 1,

> "How accurately can the community taste from requests posted on the subreddit *r/gamingsuggestions* be fitted with RS?",

is hence answered.

By introducing several post-filtering approaches, the recommendations could be improved significantly. Depending on the algorithm, the F1-score almost could be doubled and the baseline recommendation could be topped by each approach.

Post-filters work by adding contextual information to the computed list of recommendation. They re-rank the video games in those lists by changing the overall recommendation score. By analyzing 100 different submissions and extracting common additional requests, five post-filtering approaches could be obtained: Genre, explicit Tags, implicit Tags, Metacritic Score and Reviews. Each post-filter was added to the original recommendation list with a specific weighting factor. The weighting factors were generated by

splitting the 100 submissions in training- and test-data via 10-fold-cross-validation and grid searching with different values. The post-filter weights all ranged between 0 and 0.5, depending on the dataset (see Section **??**, Table 5.1 and Table 5.2). The Metacritic-Score and the implicit tag score had the highest values. Thus, highly rated games seem to be superior to other games and the associated tags for each game play an important role in improving recommendations. Research question 3,

> "Considering post-filtering, how can recommendation results be influenced with adjusting the recommendation list?",

is considered as answered.
There are significant improvements with the application of post-filter-techniques, the community taste could be fitted to about 20 percent, which equals an improvement of nearly 70 percent in the best case. To verify whether the recommendations are reasonable or not, and in order to answer Research Question 2,

> "Are the query driven recommendations viable, that is, do they yield a reasonable and satisfying output?",

a qualitative evaluation of the results was performed. For each of the 100 submission from the subreddit *r/gamingsuggestions*, the list of recommendations @10 from each algorithm, ten recommended games from steam and the suggestions from the community had been merged to one list. A website containing that list, the submission text and the submission title was created. An individual link was sent to 20 different participants. Each game could be rated as either a good, an average or as a bad recommendation. As depicted in Section 5.2, the results are in the same range for each algorithm and the community respectively. Considering only *good* recommendations, the collaborative filtering approach seems to be the best with 56% good recommendations, compared to 54.5% from the reddit community. However, the community seems to give better *neutral* recommendations, 31.8% compared to 28% from the CF-algorithm. Furthermore, the community gives the least *bad* recommendations, 13.7% compared to 15.9% (CF). This could be to the lack of the implementation of hard filters and is subject to further investigation. Considering the average score with only good and neutral ratings, CF yields better results with 1.96 points per game, compared to 1.95 points from the community. To conclude, Research Question 2 can be

answered with a clear *yes*, the query driven recommendation algorithms yield reasonable output.

However, even if the above results look promising, there are a few limitations of this work. First, the amount of testcases is very low with just 100. The test case acquisition is a rather time intensive work, since every submission has to be searched by hand. Furthermore, each game in the requirements as well as in the suggestions has to be matched manually with a corresponding game on the steam platform. The same goes for the keywords. In order to get statistically significant results, more test cases are required.

Second, all requests are from a single community, hence all retrieved test cases might be biased towards for example a specific genre.

Third, although steam provides a huge variety of video games, not all games could be found. For example, the *Diablo*-Series or *World of Warcraft* are Blizzard-specific games which are not available on steam, however, they are often requested an suggested.

Fourth, in this thesis, the tags or keywords of a submission where created only by a single author. With multiple users, the quality of the explicit tags filter could be improved.

Fifth, some of the testcases are a few years old. Thus, the algorithms may produce viable recommendations with newer games which where not released when the submission was created. Therefore, the games could not have been suggested, hence the overlap might be different by taking that into account.

Sixth, at the time of writing this thesis, only soft filters had been implemented. By adding hard filters, for example, if a user specifically states that she or he wants to play with a female protagonist, games that does not support that should be removed from the resulting list. Currently, games are only removed implicitly by ranking games which support that feature higher.

Looking at the qualitative evaluation, more participants are required. Only one person finished the complete survey, the majority only completed less than 15%. However, as stated above, the results generated from the algorithms seems to be (however not statistically significant) in line with the community, even though the overlap is not very high.

7 Conclusion and Future Work

This thesis was about the realization of a video game recommender engine. The goal of the recommender engine was to fit the community taste, such as in the subreddit *r/gamingsuggestions*. Chapter 1 dealt with the history and the evolution of RS. The research questions which were answered during this thesis were also defined:

RQ_1 How accurately can the community taste from requests posted on the subreddit *r/gamingsuggestions* be fitted with RS?

RQ_2 Are the query driven recommendations viable, i.e. do they yield a reasonable and satisfying output?

RQ_3 Considering post-filtering, how can recommendation results be influenced with adjusting the recommendation list?

The chapter was concluded with the motivation and the need of specialized, query driven RS.

Chapter 2 described the underlying principles of RS. Necessary terms, for example *user*, *item* and *translation* were introduced. Furthermore, the theoretical background of the algorithms (collaborative filtering, matrix factorization and term frequency—inverse document frequency) utilized in this thesis was treated. Following, a short overview of the used platforms (Steam, Reddit) was given.

Chapter 3 gave a summary of the already conducted work in the field of video game recommendation. Moreover, the current state of research in the domain of narrative driven recommendations was depicted.

Chapter 4 included every single step to reproduce the work performed for this thesis. First, the utilized hardware is described. Second, the data acquisition as well as the data preprocessing process is broken down to details, covering all necessary development procedures. Since some adaptions to the standard recommender algorithms were performed, the actually implemented ones were described in this chapter. The chapter was concluded by depicting the evaluation- and testing process (including 10-fold-cross validation, qualitative evaluation and unit testing).

In Chapter 5, the results of the conducted experiments were displayed. For each algorithm, precision, recall and the F1-score compared to the community as a gold standard was depicted. Furthermore, the post-filtering weights were described, followed by the results of the qualitative evaluation.

The results were afterwards discussed and interpreted in Chapter 6. The algorithm which had the highest overlap of recommendations with the community was the item-based collaborative filtering approach. A coverage of roughly 20% was reached. The qualitative evaluation showed, that the recommendations of the algorithms and the recommendations of the community nearly were perceived equal by the human testers with good recommendations in over 50%. Also, the limitations of this work were discussed.

Naturally, the results could be improved in various ways. One could be the implementation of a community feedback loop. By releasing the recommender engine on the subreddit, the recommendations could be voted up or down by the community. Thus, the algorithms could also be trained on that feedback, likely resulting in better recommendations.

Moreover, hard filters could be implemented to further improve the results by eliminating *no-go* options beforehand. Another option would be the implementation of other algorithmic recommender approaches. Especially the field of neural network approaches does look quite promising.

Furthermore, natural language processing could be included. By implementing the ability to understand language, testcases could be obtained

easier, thus resulting in more training possibilities for the algorithms.

Bibliography

Adomavicius, Gediminas and Alexander Tuzhilin (2005). "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions." In: *IEEE Trans. on Knowl. and Data Eng.* 17.6, pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99. URL: https://doi.org/10.1109/TKDE.2005.99 (cit. on pp. 8, 11, 24).

Adomavicius, Gediminas and Alexander Tuzhilin (2011). "Context-Aware Recommender Systems." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 217–253. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_7. URL: https://doi.org/10.1007/978-0-387-85820-3_7 (cit. on pp. 34, 48, 63).

Amatriain, Xavier et al. (2011). *Data Mining Methods for Recommender Systems* (cit. on p. 59).

Anwar, S. M. et al. (2017). "A game recommender system using collaborative filtering (GAMBIT)." In: *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 328–332. DOI: 10.1109/IBCAST.2017.7868073 (cit. on p. 33).

Balabanović, Marko and Yoav Shoham (1997). "Fab: Content-based, Collaborative Recommendation." In: *Commun. ACM* 40.3, pp. 66–72. ISSN: 0001-0782. DOI: 10.1145/245108.245124. URL: http://doi.acm.org/10.1145/245108.245124 (cit. on p. 18).

Beck (2002). *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0321146530 (cit. on p. 67).

Bell, Robert, Yehuda Koren, and Chris Volinsky (2007). "Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems." In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '07. San Jose, California, USA: ACM, pp. 95–104. ISBN: 978-1-59593-609-7. DOI: 10.1145/1281192.1281206. URL: http://doi.acm.org/10.1145/1281192.1281206 (cit. on p. 19).

Billsus, Daniel and Michael Pazzani (1996). "Learning Probabilistic User Models." In: *In Proceedings of the Workshop on Machine Learning for User Models, Sixth International Conference on User Modeling, Chia*. Springer (cit. on p. 9).

Billsus, Daniel and Michael J. Pazzani (1998). "Learning Collaborative Information Filters." In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 46–54. ISBN: 1-55860-556-8. URL: http://dl.acm.org/citation.cfm?id=645527.657311 (cit. on p. 22).

Billsus, Daniel and Michael J. Pazzani (2000). "User Modeling for Adaptive News Access." In: *User Modeling and User-Adapted Interaction* 10.2-3, pp. 147–180. ISSN: 0924-1868. DOI: 10.1023/A:1026501525781. URL: http://dx.doi.org/10.1023/A:1026501525781 (cit. on p. 18).

Bogers, Toine and Marijn Koolen (2017). "Defining and Supporting Narrative-driven Recommendation." In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys '17. Como, Italy: ACM, pp. 238–242. ISBN: 978-1-4503-4652-8. DOI: 10.1145/3109859.3109893. URL: http://doi.acm.org/10.1145/3109859.3109893 (cit. on p. 34).

Brachman, Ronald J. and Tej Anand (1996). "Advances in Knowledge Discovery and Data Mining." In: ed. by Usama M. Fayyad et al. Menlo Park, CA, USA: American Association for Artificial Intelligence. Chap. The Process of Knowledge Discovery in Databases, pp. 37–57. ISBN: 0-262-56097-6. URL: http://dl.acm.org/citation.cfm?id=257938.257944 (cit. on p. 15).

Bramwell, Tom (2007a). *Capcom signs up to Steam*. URL: http://www.eurogamer.net/articles/capcom-signs-up-to-steam (visited on 02/01/2018) (cit. on p. 26).

Bramwell, Tom (2007b). *id games added to Steam*. URL: http://www.eurogamer.net/articles/id-games-added-to-steam (visited on 02/01/2018) (cit. on p. 26).

Breese, John S., David Heckerman, and Carl Kadie (1998). "Empirical Analysis of Predictive Algorithms for Collaborative Filtering." In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. UAI'98. Madison, Wisconsin: Morgan Kaufmann Publishers Inc., pp. 43–52. ISBN: 1-55860-555-X. URL: http://dl.acm.org/citation.cfm?id=2074094.2074100 (cit. on p. 19).

Burke, Robin (2007). "Hybrid Web Recommender Systems." In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 377–408. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_12. URL: https://doi.org/10.1007/978-3-540-72079-9_12 (cit. on p. 13).

Chiang, Oliver (2011). *The Master of Online Mayhem*. URL: https://www.forbes.com/forbes/2011/0228/technology-gabe-newell-videogames-valve-online-mayhem.html#7889f8773ac0 (visited on 02/01/2018) (cit. on p. 26).

Chowdhury, Hussain A. and Dhruba K. Bhattacharyya (2017). "mRMR+: An Effective Feature Selection Algorithm for Classification." In: *Pattern Recognition and Machine Intelligence*. Ed. by B. Uma Shankar et al. Cham: Springer International Publishing, pp. 424–430. ISBN: 978-3-319-69900-4 (cit. on p. 11).

Cios, Krzysztof J. et al. (2007). *Data Mining: A Knowledge Discovery Approach*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387333339 (cit. on p. 15).

Cremonesi, Paolo and Roberto Turrin (2009). "Analysis of Cold-start Recommendations in IPTV Systems." In: *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. New York, New York, USA: ACM, pp. 233–236. ISBN: 978-1-60558-435-5. DOI: 10.1145/1639714.1639756. URL: http://doi.acm.org/10.1145/1639714.1639756 (cit. on p. 59).

Delgado, Joaquin and Naohiro Ishii (1999). "Memory-Based Weighted-Majority Prediction for Recommender Systems." In: (cit. on p. 22).

Deshpande, Mukund and George Karypis (2004). "Item-based top-N Recommendation Algorithms." In: *ACM Trans. Inf. Syst.* 22.1, pp. 143–177. ISSN: 1046-8188. DOI: 10.1145/963770.963776. URL: http://doi.acm.org/10.1145/963770.963776 (cit. on p. 19).

Desrosiers, Christian and George Karypis (2011). "A Comprehensive Survey of Neighborhood-based Recommendation Methods." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 107–144. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_4. URL: https://doi.org/10.1007/978-0-387-85820-3_4 (cit. on p. 51).

Domo (2016). *Data Never Sleeps 4.0*. URL: https://www.domo.com/learn/data-never-sleeps-4-0 (visited on 02/01/2018) (cit. on p. 1).

Edwards, Cliff (2013). *Valve Lines Up Console Partners in Challenge to Microsoft, Sony*. URL: https://www.bloomberg.com/news/articles/2013-11-04/valve-lines-up-console-partners-in-challenge-to-microsoft-sony (visited on 02/01/2018) (cit. on p. 26).

Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996). "Knowledge Discovery and Data Mining: Towards a Unifying Framework." In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, pp. 82–88. URL: http://dl.acm.org/citation.cfm?id=3001460.3001477 (cit. on p. 15).

Felfernig, Alexander et al. (2014). *Basic Approaches in Recommendation Systems*. In: M., Robillard et al. *Recommendation Systems in Software Engineering*. ISBN: 978-3-642-45134-8 (cit. on p. 2).

Fischer, Gerhard (2001). "User Modeling in Human–Computer Interaction." In: *User Modeling and User-Adapted Interaction* 11.1, pp. 65–86. ISSN: 1573-1391. DOI: 10.1023/A:1011145532042. URL: https://doi.org/10.1023/A:1011145532042 (cit. on p. 9).

Funk, Simon (2006). *Netflix Update: Try This at Home*. URL: http://sifter.org/simon/journal/20061211.html (visited on 02/01/2018) (cit. on pp. 22, 23).

Goldberg, David et al. (1992). "Using Collaborative Filtering to Weave an Information Tapestry." In: *Commun. ACM* 35.12, pp. 61–70. ISSN: 0001-0782. DOI: 10.1145/138859.138867. URL: http://doi.acm.org/10.1145/138859.138867 (cit. on p. 8).

Gori, Marco and Augusto Pucci (2007). "ItemRank: A Random-walk Based Scoring Algorithm for Recommender Engines." In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., pp. 2766–2771. URL: http://dl.acm.org/citation.cfm?id=1625275.1625720 (cit. on p. 22).

Grčar, Miha et al. (2006). "kNN Versus SVM in the Collaborative Filtering Framework." In: *Data Science and Classification*. Ed. by Vladimir Batagelj et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 251–260. ISBN: 978-3-540-34416-2 (cit. on p. 19).

Hariri, Negar, Bamshad Mobasher, and Robin Burke (2013). "Query-driven Context Aware Recommendation." In: *Proceedings of the 7th ACM Conference on Recommender Systems*. RecSys '13. Hong Kong, China: ACM,

pp. 9–16. ISBN: 978-1-4503-2409-0. DOI: 10.1145/2507157.2507187. URL: http://doi.acm.org/10.1145/2507157.2507187 (cit. on p. 34).

Heinz, Stefan, Yvonne Lau, and Daniel Epstein (2017). *Metarecommendr: A recommendation system for video games, movies and TV shows*. URL: https://nycdatascience.com/blog/student-works/capstone/metarecommendr-recommendation-system-video-games-movies-tv-shows/ (visited on 02/01/2018) (cit. on p. 33).

Helic, Denis and Roman Kern (2018). *Knowledge Discovery and Data Mining 1 Slides, University of Technology Graz* (cit. on pp. 12, 13).

Hofmann, Thomas (2003). "Collaborative Filtering via Gaussian Probabilistic Latent Semantic Analysis." In: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*. SIGIR '03. Toronto, Canada: ACM, pp. 259–266. ISBN: 1-58113-646-3. DOI: 10.1145/860435.860483. URL: http://doi.acm.org/10.1145/860435.860483 (cit. on p. 19).

IBM (2017). *10 Key Marketing Trends for 2017*. Ed. by IBM Marketing Cloud. URL: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN (visited on 02/01/2018) (cit. on p. 1).

Jannach, Dietmar, Markus Zanker, Alexander Felfernig, et al. (2011). *Recommender Systems: An Introduction*. ISBN: 978-0-521-49336-9 (cit. on p. 7).

Jannach, Dietmar, Markus Zanker, Mouzhi Ge, et al. (2012). "Recommender Systems in Computer Science and Information Systems – A Landscape of Research." In: *E-Commerce and Web Technologies: 13th International Conference, EC-Web 2012, Vienna, Austria, September 4-5, 2012. Proceedings*. Ed. by Christian Huemer and Pasquale Lops. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 76–87. ISBN: 978-3-642-32273-0. DOI: 10.1007/978-3-642-32273-0_7. URL: https://doi.org/10.1007/978-3-642-32273-0_7 (cit. on p. 8).

Jugovac, Michael, Dietmar Jannach, and Lukas Lerche (2017). "Efficient optimization of multiple recommendation quality factors according to individual user tendencies." In: *Expert Systems with Applications* 81.Supplement C, pp. 321–331. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.03.055. URL: http://www.sciencedirect.com/science/article/pii/S0957417417302075 (cit. on p. 9).

Koenigstein, Noam et al. (2012). "The Xbox Recommender System." In: *Proceedings of the Sixth ACM Conference on Recommender Systems*. RecSys '12. Dublin, Ireland: ACM, pp. 281–284. ISBN: 978-1-4503-1270-7. DOI:

10.1145/2365952.2366015. URL: http://doi.acm.org/10.1145/2365952.2366015 (cit. on p. 34).

Koren, Yehuda (2008). "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model." In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '08. Las Vegas, Nevada, USA: ACM, pp. 426–434. ISBN: 978-1-60558-193-4. DOI: 10.1145/1401890.1401944. URL: http://doi.acm.org/10.1145/1401890.1401944 (cit. on p. 19).

Lee, Daniel D. and H. Sebastian Seung (2001). "Algorithms for Non-negative Matrix Factorization." In: *In NIPS*. MIT Press, pp. 556–562 (cit. on p. 22).

Lee, Daniel and H Sebastian Seung (1999). "Learning the Parts of Objects by Non-Negative Matrix Factorization." In: 401, pp. 788–91 (cit. on p. 22).

Linden, G., B. Smith, and J. York (2003). "Amazon.com recommendations: item-to-item collaborative filtering." In: *IEEE Internet Computing* 7.1, pp. 76–80. ISSN: 1089-7801. DOI: 10.1109/MIC.2003.1167344 (cit. on p. 19).

M. Blei, David, Andrew Y. Ng, and Michael Jordan (2001). *Latent Dirichlet Allocation* (cit. on p. 22).

Mahmood, Tariq and Francesco Ricci (2009). "Improving Recommender Systems with Adaptive Conversational Strategies." In: *Proceedings of the 20th ACM Conference on Hypertext and Hypermedia*. HT '09. Torino, Italy: ACM, pp. 73–82. ISBN: 978-1-60558-486-7. DOI: 10.1145/1557914.1557930. URL: http://doi.acm.org/10.1145/1557914.1557930 (cit. on pp. 3, 9).

Melville, Prem and Vikas Sindhwani (2010). "Recommender Systems." In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, pp. 829–838. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_705. URL: https://doi.org/10.1007/978-0-387-30164-8_705 (cit. on pp. 6–8).

Metacritics (2018). *Metacritics*. URL: http://www.metacritic.com/about-metacritic (visited on 02/06/2018) (cit. on p. 65).

Montaner, Miquel, Beatriz López, and Josep Lluís de la Rosa (2003). "A Taxonomy of Recommender Agents on the Internet." In: *Artificial Intelligence Review* 19.4, pp. 285–330. ISSN: 1573-7462. DOI: 10.1023/A:1022850703159. URL: https://doi.org/10.1023/A:1022850703159 (cit. on p. 10).

Netflix (2009). *Netflix Prize*. URL: https://www.netflixprize.com/index.html (visited on 01/08/2018) (cit. on p. 8).

Nguyen, Tien T. et al. (2014). "Exploring the Filter Bubble: The Effect of Using Recommender Systems on Content Diversity." In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: ACM, pp. 677–686. ISBN: 978-1-4503-2744-2. DOI: 10.1145/2566486.2568012. URL: http://doi.acm.org/10.1145/2566486.2568012 (cit. on p. 12).

Ott, Patrick (2008). "Incremental Matrix Factorization for Collaborative Filtering." In: (cit. on p. 22).

Pariser, Eli (2011). *The Filter Bubble: What the Internet Is Hiding from You*. Penguin Group , The. ISBN: 1594203008, 9781594203008 (cit. on p. 12).

Perry, James W., Allen Kent, and Madeline M. Berry (1955). "Machine literature searching X. Machine language; factors underlying its design and development." In: *American Documentation* 6.4, pp. 242–254. ISSN: 1936-6108. DOI: 10.1002/asi.5090060411. URL: http://dx.doi.org/10.1002/asi.5090060411 (cit. on p. 14).

Perunicic, Andre (2017). *Scraping user-submitted reviews from the steam store*. URL: https://intoli.com/blog/steam-scraper/ (visited on 02/01/2018) (cit. on p. 42).

Purchese, Robert (2007). *Eidos embraces Steam power*. URL: http://www.eurogamer.net/articles/eidos-embraces-steam-power (visited on 02/01/2018) (cit. on p. 26).

Raschka, Sebastian (2018). *10-fold-cross-validation*. URL: https://sebastianraschka.com/ (visited on 02/04/2018) (cit. on p. 59).

Reddit (2018). *Reddit - About*. URL: https://www.reddit.com/wiki/index (visited on 02/01/2018) (cit. on p. 29).

Reitz, Kenneth (2018). *Testing Your Code*. URL: http://docs.python-guide.org/en/latest/writing/tests/ (visited on 02/07/2018) (cit. on p. 67).

Resnick, Paul et al. (1994). "GroupLens: An Open Architecture for Collaborative Filtering of Netnews." In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. CSCW '94. Chapel Hill, North Carolina, USA: ACM, pp. 175–186. ISBN: 0-89791-689-1. DOI: 10.1145/192844.192905. URL: http://doi.acm.org/10.1145/192844.192905 (cit. on p. 8).

Ricci, Francesco (2014). "Recommender Systems: Models and Techniques." In: *Encyclopedia of Social Network Analysis and Mining*. Ed. by Reda Alhajj

and Jon Rokne. New York, NY: Springer New York, pp. 1511–1522. ISBN: 978-1-4614-6170-8. DOI: 10.1007/978-1-4614-6170-8_88. URL: https://doi.org/10.1007/978-1-4614-6170-8_88 (cit. on p. 11).

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2015). *Recommender Systems Handbook*. Vol. 2. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6 (cit. on pp. 1–3, 6, 9, 10, 17–19, 52).

Rich, Elaine (1979). "User Modeling via Stereotypes*." In: *Cognitive Science* 3.4, pp. 329–354. ISSN: 1551-6709. DOI: 10.1207/s15516709cog0304_3. URL: http://dx.doi.org/10.1207/s15516709cog0304_3 (cit. on pp. 7, 8).

Salton, Gerard (1989). *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-12227-8 (cit. on p. 24).

Salzberg, Steven L. (1997). "On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach." In: *Data Mining and Knowledge Discovery* 1.3, pp. 317–328. ISSN: 1573-756X. DOI: 10.1023/A:1009752403260. URL: https://doi.org/10.1023/A:1009752403260 (cit. on p. 59).

Sarwar, Badrul et al. (2001). "Item-based Collaborative Filtering Recommendation Algorithms." In: *Proceedings of the 10th International Conference on World Wide Web*. WWW '01. Hong Kong, Hong Kong: ACM, pp. 285–295. ISBN: 1-58113-348-0. DOI: 10.1145/371920.372071. URL: http://doi.acm.org/10.1145/371920.372071 (cit. on pp. 21, 51).

Schafer, J. Ben et al. (2007). "Collaborative Filtering Recommender Systems." In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 291–324. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_9. URL: https://doi.org/10.1007/978-3-540-72079-9_9 (cit. on p. 10).

Schreier, Jason (2014). *Steam Is Getting A Massive Overhaul*. URL: https://kotaku.com/steam-is-getting-a-massive-overhaul-1637818112 (visited on 02/01/2018) (cit. on p. 27).

Scrapy (2018). *Scrapy - An open source and collaborative framework for extracting the data you need from websites.* URL: https://scrapy.org/ (visited on 02/01/2018) (cit. on p. 42).

Shardanand, Upendra and Pattie Maes (1995). "Social Information Filtering: Algorithms for Automating &Ldquo;Word of Mouth&Rdquo;" in: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

CHI '95. Denver, Colorado, USA: ACM Press/Addison-Wesley Publishing Co., pp. 210–217. ISBN: 0-201-84705-1. DOI: 10.1145/223904.223931. URL: http://dx.doi.org/10.1145/223904.223931 (cit. on p. 18).

Sifa, Rafet, Christian Bauckhage, and Anders Drachen (2014). "Archetypal game recommender systems." In: 1226, pp. 45–56 (cit. on p. 33).

Skocir, Pavle et al. (2012). "The MARS – A Multi-Agent Recommendation System for Games on Mobile Phones." In: *Agent and Multi-Agent Systems. Technologies and Applications*. Ed. by Gordan Jezic et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 104–113. ISBN: 978-3-642-30947-2 (cit. on p. 33).

Smith, Craig (2018). *71 Amazing Reddit Statistics and Facts (January 2018)*. URL: https://expandedramblings.com/index.php/reddit-stats/ (visited on 02/01/2018) (cit. on p. 30).

Sobkowicz, Antoni and Wojciech Stokowiec (2016). *Steam Review Dataset - new, large scale sentiment dataset* (cit. on p. 46).

Steam (2018a). *Steam*. URL: http://store.steampowered.com/ (visited on 02/01/2018) (cit. on p. 26).

Steam (2018b). *Steam Website*. URL: https://partner.steamgames.com/doc/store/application (visited on 02/01/2018) (cit. on p. 40).

Steamworks (2018). *Steamworks*. URL: https://partner.steamgames.com/ (visited on 02/01/2018) (cit. on p. 37).

Taghipour, Nima, Ahmad Kardan, and Saeed Shiry Ghidary (2007). "Usage-based Web Recommendations: A Reinforcement Learning Approach." In: *Proceedings of the 2007 ACM Conference on Recommender Systems*. RecSys '07. Minneapolis, MN, USA: ACM, pp. 113–120. ISBN: 978-1-59593-730–8. DOI: 10.1145/1297231.1297250. URL: http://doi.acm.org/10.1145/1297231.1297250 (cit. on p. 9).

Takács, Gábor et al. (2008). "Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem." In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. RecSys '08. Lausanne, Switzerland: ACM, pp. 267–274. ISBN: 978-1-60558-093-7. DOI: 10.1145/1454008.1454049. URL: http://doi.acm.org/10.1145/1454008.1454049 (cit. on p. 22).

Takács, G. et al. (2008). "Investigation of Various Matrix Factorization Methods for Large Recommender Systems." In: *2008 IEEE International Conference on Data Mining Workshops*, pp. 553–562. DOI: 10.1109/ICDMW.2008.86 (cit. on p. 19).

Ullman, Jeff (2018). *Mining Massive Datasets: Recommendation Systems chapter 9, Stanford university* (cit. on pp. 12, 13).

Valve Developer Community (2018). *Steam API*. URL: https://developer.valvesoftware.com/wiki/Steam_Web_API (visited on 02/01/2018) (cit. on p. 46).

Wigzo.com (2018). *Knowledge Discovery Process Picture*. URL: https://www.wigzo.com/blog/how-multi-dimensional-analysis-of-information-database-sets-you-apart-from-the-competition/ (visited on 02/01/2018) (cit. on p. 16).