Jörg Gabmeier Bakk.rer.soc.oec.

Recommender Systems in the Domain of Video Games: A Comparison of Various Algorithms, Rating Scales and Implicit/Explicit Feedback Utilizing the Steam Platform

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Software Development and Business Management

submitted to

Graz University of Technology

Supervisor

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

Knowledge Technologies Institute

Faculty of Computer Science and Biomedical Engineering

Graz, September 2016

Abstract

Recommender systems are ubiquitous. This thesis deals with recommender systems in the domain of video games and specifically the Steam platform by comparing the performance of several selected algorithms provided by the *MyMediaLite* library. Furthermore, implicit feedback is converted into explicit ratings and compared to actual explicit ratings provided by the many users of the Steam network. Last but not least, binary to quinary rating scales are considered in order to find the one that performs best in terms of accuracy.

The thesis further consists of an introduction to recommender systems, an overview of various recommender system libraries, implemented algorithms, the Knowledge Discovery Process (KDP) and the Steam platform itself. Related work, also dealing with the domain of video games as well as diverse rating scales, is presented. The approach used for conducting the experiments, in order to answer the research questions, is based on the KDP and thoroughly described. The selected recommendation algorithms are tested on five datasets specifically created for this purpose, whereby the data is crawled directly from the Steam website. 10-fold cross-validation, with separate validation and test sets for hyperparameter optimization and evaluation, is applied. The NRMSE is calculated and presented for each algorithm and dataset in the form of tables as well bar plots. Finally, the Friedman test and the subsequent Nemenyi post-hoc test is applied in order to draw reliable conclusions.

Contents

List of Figures

List of Tables

1 Introduction

The information available on the web is growing steadily and so is the provided content of big websites such as Amazon, Netflix, YouTube and Steam. Content can be pretty much anything, such as books, movies, games and scientific papers, and sometimes the sheer amount of this content can be overwhelming. It can be difficult to retrieve good results not only due to this mass of information, but also due to the users' lack of knowledge or simply the content's uncommonness. Filtering seems to be a logical necessity for the users of these platforms in order to retrieve personalized content and this is where recommender systems are used in an attempt to assist.

Recommender systems are used to automatically fulfill the desire for receiving only appropriate content and are ubiquitous nowadays. In fact, all the platforms mentioned above implement some sort of recommendation system that caters to the needs of their users by providing personalized content with the goal of an improved user experience, thus enhancing customer loyalty, selling more and diverse items, and ultimately improving the companies' revenues. (Ricci, Rokach, and Shapira, 2015a)

Many of the companies even provide valuable information about their systems. For instance Goldberg et al. (1992), the developers of Tapestry, the first recommender system, coined the term collaborative filtering. (Resnick and Varian, 1997) Insight into Amazon's Item-to-Item Collaborative Filtering was provided by Linden, B. Smith, and York (2003), Davidson et al. (2010) presented YouTube's video recommendation system, and Gomez-Uribe and Hunt (2016) recently provided material on Netflix's range of varying recommender algorithms. On the other hand, to the best of the author's knowledge, Valve, developer of the Steam platform, made no scientific work on Steam's recommendation system available for researchers, yet they are certainly one of the major players in the gaming industry and probably worth billions. (Chiang, 2011)

One interesting difference between these platforms is their varying rating scale, which is the explicit basis for most recommender systems. While Amazon and Netflix use the classical rating scale of one to five stars, YouTube changed this scale into a binary one consisting solely of a like and a dislike rating after considering the fact that hardly anyone ever used ratings below five stars. (Rajaraman, 2009) Steam also provides this binary rating scale in the form of a thumbs-up and a thumbs-down. Despite these differences, there is little research that deals with the performance effects of varying rating scales, especially within the domain of video games.

A further significant distinction lies not within the rating scales, in other words explicit data, but in the implicit information of user preferences available to each system. Of course all of these websites could easily track simple user interaction, such as following links or purchasing items, however some services have quite unique domain specific information available. While YouTube and Netflix could track view counts of their videos or movies respectively, video game platforms like Steam have the advantage of easily recordable information on the time a user has spent playing a certain game.

Based on these observations, this thesis strives to answer the following research questions with reference to the domain of video games and specifically the Steam platform:

RQ 1 How well are various recommender algorithms performing in comparison to each other?

RQ 2 Can implicit data, specifically game times for each user, be utilized as an adequate replacement for explicit ratings?

RQ 3 Considering rating scales from binary to quinary, which rating scale is to be preferred in terms of accuracy?

Answering these research questions requires the application of the knowledge discovery process to the topic of recommender systems. The first major step in this process is the data acquisition from the Steam platform using web crawling, followed by preprocessing and cleaning of the retrieved data. Next, the data is transformed into a usable input for different recommender algorithms, which are chosen from existing recommender system libraries. Algorithms are selected with diversity in mind, trained, and finally evaluated, thus concluding the process.

The remainder of this thesis is structured as follows. Chapter 2 gives an overview of recommender systems, existing recommender system libraries along with assorted algorithms, the knowledge discovery process and the Steam platform. Chapter 3 provides a survey of literature relevant to the research questions raised. Chapter 4 makes up the major part of the thesis and contains the detailed materials and methods used to answer the research questions. Results are presented in Chapter 5 and discussed in Chapter 6. Last but not least, Chapter 7 concludes with a summary of the presented work.

2 Background

This chapter is dedicated to the theoretical background of the thesis. First, an introduction to recommender systems is given, which includes a historical view on such systems, the definitions of basic terms and concepts, a taxonomy on the varying recommendation techniques, associated challenges and issues, and the corresponding evaluation methods. The introduction is followed by an overview of various recommender system libraries and selected recommendation algorithms. Further, the knowledge discovery process used to answer the research questions is depicted and interpreted from the viewpoint of a recommender system. The chapter is concluded with a brief introduction of the Steam platform, which represents the data base for the experiments.

2.1 Recommender Systems

Be it the question of what book to read next, what movie to rent or what stock to buy, every day life is full of choices and options. Especially since the information available on the world wide web is steadily growing, decision-making can be tough for internet users. Recommender systems assist in this decision-making process by providing personalized recommendations to their users and have become a broad field in computer science. (Jannach et al., 2011)

Most of the internet users have already made contact with one or another recommendation system within the web. For instance, while shopping at Amazon, it is quite difficult to miss the user reviews along with the star ratings for the offered products or Amazon's recommendations on what other users bought together with the currently viewed item.

Actually providing a useful recommendation requires a user profile which reflects the preferences of the user. This preferences can be acquired by explicitly asking the user or by converting implicitly observed behavior of the user. Implicit information can be the user's browsing habits, purchases, and other interactions with the platform in question, while explicit data is mostly represented by ratings of the content. (Jannach et al., 2011)

Utilizing these user preferences and driven by the goal of raising the user's experience on the service, a recommendation system attempts to predict the user's preference or rating for items on which he has not yet expressed his opinion. It is most certainly a mutual benefit for the client and the customer, as a satisfied customer will more likely return and consume more, while the company profits in obvious and also not so obvious ways. The latter, for example, includes the opportunity to sell more diverse and not so popular items, also called the long-tail of an item catalog, eventually giving the company an advantage over others.

2.1.1 History

People have always been influenced by the recommendations of other people, regardless of whether they are domain experts or just friends with similar tastes. Putting trust in ones peers has limits though. For instance, a person interested in finding new movies to watch might never have heard of a specific movie that he might possibly like. This could be due to the uncommonness of the movie itself or uninformed peers, and recommender systems are used to help with such issues.

Research on recommender systems has been going on for more than 30 years now. The roots can be traced back to Grundy, which was a rather primitive system outlined by Rich (1979), but nevertheless an important step in the history of recommender systems. It modeled the user profiles by asking the user a few questions and then grouped them into stereotypes based on their answers. Recommendations were then made by hard-coded preferences attached to these stereotypes.

Another major step was taken by Goldberg et al. (1992), who coined the term *collaborative filtering*, and the Tapestry system they developed. (Resnick and

Varian, 1997) The system was intended to replace email systems and was not entirely automated. Users would have to annotate documents in order for others to be able to receive relevant content. They would then either directly query the system for certain annotations or other users, or set their preferences by installing filters, which would then deliver relevant content as soon as it became available.

Recommender systems have since become an important field in research due to the emergence of several scientific papers during the mid-1990s. (Adomavicius and Tuzhilin, 2005) Examples include the work of Resnick, Iacovou, et al. (1994) on GroupLens for recommending netnews articles, Shardanand and Maes (1995) on Ringo for music, and Hill et al. (1995) on the Bellcore Video Recommender for movies, which are all automated collaborative filtering approaches.

In the late 1990s, commercial usage of recommender technologies began, with one prominent example being Amazon and their usage of browsing and purchasing history for proposing new items to the customer. (M. D. Ekstrand, Riedl, and Konstan, 2011) Amazon's *item-to-item collaborative filtering* was described by Linden, B. Smith, and York (2003) and along with the work of Sarwar et al. (2001) and Karypis (2001), brought insight into a new way of recommending things. Later in the mid-2000s, Netflix gained massive attention from both scientists and hobbyists when they offered a million dollar prize to anyone able to improve their Cinematch recommendation system in terms of accuracy by 10%. (Netflix, 2009)

Much has changed since then and companies started to overthink their ways of recommending content. For example, YouTube changed their 5-star rating scale into a binary like/dislike system after considering the fact that most people hardly ever used a rating other than 5 stars. (Rajaraman, 2009) Netflix still utilizes the algorithms resulting from their competition, but has also extended the system by a variety of other approaches. (Gomez-Uribe and Hunt, 2016) Further, while most of the early research focused on improving the accuracy of recommender systems, more recent research also deals with other aspects such as navigability, reachability and diversity. (Nguyen et al., 2014; Lamprecht et al., 2015)

2.1.2 Basic Terms and Concepts

The three most important terms in the field of recommender systems are probably users, items and ratings. A user states his preferences for various items in the form of a rating, which is together often represented as a triple (user, item, rating). All triples together then form a rating matrix, which usually is very sparse. In the end, the recommender system calculates what is worth recommending by predicting the utility of the item for the user in question, thus filling up empty entries in the ratings matrix. Last but not least, the recommender system typically presents a list of n recommended items to the user. (M. D. Ekstrand, Riedl, and Konstan, 2011)

Table 2.1 shows an example of a small ratings matrix with ratings between 1 and 5 stars, whereby empty cells represent items a user has not yet rated. An example of a rating triple could therefore be (Alice, Django Unchained, 5). Based on the existing ratings, the recommender might infer a rating of 4 stars for Alice and Aliens, thus recommending Aliens to Alice.

	Django Unchained	The Matrix	Aliens	Schindler's List
Alice	5	4		1
Bob	4	5	4	
Charlie		3		4

Table 2.1: A small example of a ratings matrix with ratings between 1 and 5 stars. Empty cells represent items a user has not yet rated.

The following is a formal definition of the recommendation problem based on Adomavicius and Tuzhilin (2005): let U be the set of all users, I the set of all items that can be recommended and f the utility function that measures the usefulness of item i for user u, that is $f : U \times I \rightarrow R$, where R is a totally ordered set of ratings. Then, for each user $u \in U$, the recommender system is supposed to choose such item $i' \in I$ that maximizes the user's utility, or more formally:

$$\forall u \in U, \quad i'_u = \arg\max_{i \in I} f(u, i) \tag{2.1}$$

Definitions of the subsequent basic terms are all based on the elucidations of Ricci, Rokach, and Shapira (2015a).

2.1.2.1 User

Users of recommender systems are always accompanied by a user model that represents the personal preferences and needs, which can of course be very diverging. Without a user model, personalized recommendations would not be possible at all and therefore, they will always play a central role in the field. The constitution of a given user profile is heavily dependent on the system in question and can be a simple list of ratings, sociodemographic attributes such as age, gender, and the country they live in, behavior data like browsing histories, or relations to other users in the form of a trust level.

2.1.2.2 Item

Items in this context can be any number of things such as movies, video games, websites, research papers, and jokes. They represent the things to be recommended and can be characterized by their complexity, value and/or utility. It certainly does make a huge difference for the recommender system to recommend simple things like books and CDs, or much more complex things like insurance policies and financial investments. In order to model the items, properties and features can be attached to make them more distinguishable.

2.1.2.3 Transaction

Transactions represent a recorded interaction between a user and the recommendation system. A transaction can be rating data, which is the most common form, but also evaluation in the form of user-applied tags. For example, the user could tag a video game with certain expressions such as “boring”, “beautiful graphics” and so forth. Ratings on the other hand can either be expressed explicitly or implicitly, depending again on the domain and the platform involved, and can take on many forms. Explicit ratings can be unary (simply indicating that a user has made a positive interaction with the item), binary (positive or negative), numerical (for example 1 to 5 stars) and ordinal (for instance "agree", "neutral", "disagree"), while implicit ratings can be derived from many user interactions such as clicks on links, the watch count of videos, or playtime for video games.

2.1.2.4 Prediction

A prediction is usually made by the recommender in the sense of calculating an item's utility for a specific user, or at least by comparing the utility of various items. It is not necessary, though, that the utility is calculated explicitly as it may also be sufficient to apply some heuristics in order to make the decision of whether or not an item is useful to a user.

2.1.2.5 Recommendation

Finally, a recommendation is the output of a recommender system in the form of a list of n items, usually having the largest predicted utility. Typically, n is much smaller than the cardinality of the item set, thus filtering relevant items. It is further noteworthy that the list of n recommended items can also include other items than those with highest utility. For example, the list could also contain the most popular item and an item very different to what the user is familiar with in order to avoid being stuck in a filter bubble. The term *filter bubble* can be described as isolating people from a diversity of viewpoints or content and was recently explored by Nguyen et al. (2014).

2.1.3 Techniques

Burke (2007) distinguishes recommender systems based on their source of knowledge: collaborative, content-based, demographic and knowledge-based. Further, hybrid recommender systems are defined as any combination of the aforementioned techniques and Ricci, Rokach, and Shapira (2015b) adds another technique to the taxonomy: the community-based recommendation system. The following subsections describe the basic principles behind recommenders of these techniques.

2.1.3.1 Content-Based

Content-based recommendation systems rely on the ratings of users as well as attributes and/or features of the items to be recommended, which can for example be keywords in a document recommender application or genres, directors, actors, etcetera in one for movies. The system usually generates a content based user profile as a reflection of the user's preferences on certain items and their attributes based on the ratings. In order to find good recommendations, a similarity measurement is done on the user profile and the attributes of items to be considered, which is commonly done by some scoring heuristic such as the cosine similarity. However, heuristic formulas are not the only way of predicting the utility. Other methods include Bayesian classifiers, machine learning techniques, clustering, decision trees and artificial neural networks. (Adomavicius and Tuzhilin, 2005)

These recommenders are frequently used for recommending text-based items, such as websites, documents, and e-mail messages. The reason for this is not only the importance of text-based applications, but also early advances from researchers of the information retrieval and information filtering fields. Keywords are usually given weights in order to reflect their importance, which can be carried out in several ways, whereas the best-known measure is the *term frequency/inverse document frequency (TF-IDF)*. (Adomavicius and Tuzhilin, 2005)

2.1.3.2 Collaborative Filtering

The most widely used recommender technique is collaborative filtering with its two approaches user-user and item-item. User-user thereby refers to the principle of finding users similar to the recommendation recipient based on their previous item ratings, while item-item turns this around and finds similar items based on their ratings. (M. D. Ekstrand, Riedl, and Konstan, 2011)

The Core issue of this recommender is the search for the neighborhood of either users or items, thus making the item-item approach preferable if the number of users exceeds the number of items to be considered. However, the major advantage of the item-item approach is different, namely the possibility of pre-computing the similarity matrix. In the user-user approach, a change in the user's ratings can easily also change the neighborhood to be considered, as it is dependent on the ratings of other users as well. On the other hand, if the user-item ratio is high enough and an item has many ratings, the neighborhood of an item-item approach is likely to remain stable after one user changes a rating. The re-computation of the neighborhood can then be delayed until a substantial number of users have added ratings, while the user-user approach usually computes the neighborhood every time a recommendation is needed. (M. D. Ekstrand, Riedl, and Konstan, 2011)

As the rating space in these systems can become very high-dimensional, dimensionality reduction has become an important and obvious extension of collaborative filtering. Latent semantic analysis (LSA), also called latent semantic indexing (LSI), in the form of matrix factorization techniques such as Singular Value Decomposition (SVD) and Principle Component Analysis (PCA) alleviates the scalability issues. The idea behind this is to find common information amongst users or items, as it is done explicitly with content-based systems and item features, which are latent in the rating data. (M. D. Ekstrand, Riedl, and Konstan, 2011)

2.1.3.3 Demographic

Demographic recommenders take advantage of the user's demographics such as age, gender, language, and country in order to identify user types who like certain items. Usually the ratings of users in a certain demographic niche are combined to generate recommendations for each niche. (Burke, 2007; Ricci, Rokach, and Shapira, 2015b) According to Krulwich (1997), the demographic generalization approach comes at the expense of reduced accuracy of the recommendation system.

2.1.3.4 Knowledge-Based

Knowledge-based recommendation systems utilize specific domain knowledge about how the user's preferences can be satisfied by certain item features. Although all recommendation techniques use inference in some way, knowledge-based systems can be distinguished by functional knowledge. User profiles and the knowledge used by these systems can be manifold. An example of a user profile and its knowledge structure could be a simple query on a search engine. While these systems tend to be strong in the early stages after deployment, they may easily be outperformed by simpler methods, such as collaborative filtering, if they lack some sort of learning component. (Burke, 2002; Ricci, Rokach, and Shapira, 2015b)

2.1.3.5 Community-Based

Community-based systems, also called social systems, make recommendations based on the saying "Tell me who your friends are, and I will tell you who you are". In order to make a recommendation for a user, the ratings of persons connected to this user are considered, making this a very simple yet comprehensive approach. The popularity of social networks and indications for people's tendency to prefer recommendations from friends rather than strangers lead to this recommending technique becoming more and more interesting. (Ricci, Rokach, and Shapira, 2015b)

2.1.3.6 Hybrids

A combination of any of the former techniques constitutes a hybrid recommender system. The idea behind this is to utilize the advantages of one involved technique in order to improve upon the shortcomings of the other one and vice versa. Burke (2007) found that cascading systems is one of the best strategies for building a hybrid recommender. The results from the first recommender are thereby used as an approximation, which is then fine-tuned by a second recommender. Also, using a knowledge-based recommender as a contributing component leads to good results.

2.1.4 Challenges and Issues

Every technique has its advantages and disadvantages, while some challenges are common to the field of recommender systems in general. Ricci, Rokach, and Shapira (2015a) lists three essential challenges for the field:

1. Preference Acquisition and Profiling

2. Interaction
3. New Recommendation Tasks

Acquiring user preferences is certainly one of the core challenges of recommender systems. Collecting implicit user feedback, such as clicking on links or purchasing an item, can be much easier than collecting explicit feedback due to ease of availability. While utilizing implicit feedback can be profitable, it is worth noting that this feedback can only be positive by nature and negative user preferences would still have to be extracted from explicit feedback after all. Designers of recommender systems also have to take care when collecting explicit feedback to ensure that the system is not too intrusive for the user. Further, user preferences might change over time, whereby some user preferences can be of long-term use and others of short-term use.

Also of great importance and subject to further research is the form of the systems input from the user and the output to the user. Usage of decision psychology and cognitive psychology are to be considered in this area as well as the topic of explaining recommendations. The same goes for the inclusion of novelty in recommendations in order to avoid the filter bubbles mentioned in Section 2.1.2.5, which are especially a problem for content-based filtering systems but not for knowledge-based or collaborative systems.

The third common challenge lies in totally new recommendation tasks, such as making a new recommendation every week, reciprocal recommendations (for example in dating apps), ranking recommendations instead of predicting specific ratings, guided navigation, and user action interpretation so that every user action has an effect on the recommendations.

As for the common issues, all recommendation techniques that are based on learning, specifically collaborative, content-based, and demographic recommenders, suffer from the cold-start problem, which affects the handling of new items or new users. For example, in collaborative filtering systems, new items can not be recommended to users as long as they haven't been rated by anyone. While content-based systems are inherently protected against the new item problem, they are affected by the new user problem the same way collaborative filtering systems are, since a user without ratings has not yet expressed any preferences. (Burke, 2007) Implicit feedback and user demographics may mitigate this new user problem to a certain degree. (Koren, Bell, and Volinsky, 2009)

2.1.5 Evaluation Methods

A simple online user study can help make the decision as to what algorithm works best, but is highly subjective by its very nature. The easy variant of simply having users choose the best algorithm and then rank them by their number of votes should therefore be put aside and replaced with a more detailed measurement allowing the improvement of the system in a more selective way. Gunawardana and Shani (2015) elaborates on a total of 13 desired properties of recommender systems and their evaluation methods for selecting a certain algorithm, which are summarized in the following subsections.

2.1.5.1 Prediction Accuracy

Accuracy is one of the most widely used properties because of the assumption that a user would want the system to be as accurate as possible in its predictions. Tests can easily be done offline since the accuracy is mostly independent of the user interface. Several measurements can be made depending on the prediction task, which can be the prediction of ratings (the rating a user would give an item), usage (whether the user would use an item or not), or ranking (an ordered list of items the user would might like).

2.1.5.2 Coverage

Coverage deals with the long-tail of an item set, specifically the recommendation of the majority of items that have only a few ratings each, as opposed to the head or the few items with a large amount of ratings. Item coverage is usually measured by the proportion of items a recommender system can recommend while user coverage can also be measured analogously.

2.1.5.3 Confidence

Trust in its own recommendation or prediction is known as a recommenders' confidence. It is usually measured through the probability of a predicted value being true and can be done with offline experiments by repeatedly testing the outcome of a prediction while hiding a few of the user's other ratings each time.

2.1.5.4 Trust

As opposed to confidence, trust refers to the user's trust in the system and can be measured in online experiments by asking the users for their opinion on the recommendation quality. A recommender system may recommend items the user already liked, which does not provide any direct value for the user, but may raise his trust in recommendations of previously unknown items.

2.1.5.5 Novelty

Recommending unknown items to the user is measured by novelty and can be done in either an online user study or an offline experiment. In offline experiments, a number of user ratings past a certain time will be hidden and the system will be rewarded for recommending a hidden item. Additionally, a few items before that point in time can also be hidden. If the system then recommends one of those hidden items, it will be punished. Another way of measuring a system's novelty can be accomplished by rewarding the system for accurately recommending unpopular items. It is always important to consider the system's accuracy though, because recommending an irrelevant item is worthless even if it is new to the user.

2.1.5.6 Serendipity

Serendipity is a measure of how surprising a recommendation is to the user and should always be balanced with accuracy. In other words, serendipity is the number of relevant item features that are new to the user and thus not obvious. Again, this property can be evaluated in a user study or in an offline experiment by measuring the distance of a successful recommendation to already rated items in a collaborative filtering approach, or to the user profile in a content-based approach.

2.1.5.7 Diversity

Diversity is yet another property that has a trade-off with accuracy and is measured by the distance between items in a list of recommendations. The system is rewarded for recommending diverse items and punished if it recommends items that are too similar to one another. The point of this is to provide the user with as many diverse items as possible so that many of the user's interests can be satisfied.

2.1.5.8 Utility

The utility for a user or the system can be measured in many ways. For example, the utility function could be optimized in terms of revenue for the system's company. Another possibility, in terms of utility for a user, would be to consider a recommendation of a 5-star movie to be better than the recommendation of a 4-star movie. Evaluation is simply done by measuring the utility in an offline experiment or in an online user study. In the example of improving revenue, the online study could compare the change in revenue for different users and algorithms. In the example of optimizing user utilities, the online study might become more difficult as users usually have a hard time assigning utilities to outcomes.

2.1.5.9 Risk

Some systems imposing certain risks, such as a stock recommendation system, might need a separate measurement for that. Usually this is done by also considering the variance of the utility, which can be multiplied with either a positive or negative factor, depending on whether the system should reward higher risk or lower risk.

2.1.5.10 Robustness

Robustness is about the recommendation stability in the presence of fake information, which attempts to change others' recommendations, and can be measured by the amount of information required in order to actually change the recommendation.

2.1.5.11 Privacy

It is import for most users that their expressed preferences remain private. While privacy may come at the expense of recommendation accuracy, it is usually bad if the system reveals the private information of even a single user. Evaluation can be of a theoretical nature, as in considering all cases under which private information may be disclosed, or practical, as in counting the users of certain algorithms with leaked private information.

2.1.5.12 Adaptivity

Adaptivity refers to the change of interest in certain items in general or for certain users and the algorithms capability of adapting to the new situation in a timely manner. Once again, adaptivity comes at the expense of accuracy. In a way, this stays in contrast to the robustness of a system as new items can only be recommended with a certain amount of new information. The change for a specific user instead can be evaluated by measuring the distance between a list of recommendations before and after adding new information.

2.1.5.13 Scalability

Since data used in recommender systems usually grows over time, scalability is an import point to consider and is usually measured by monitoring the speed and resource consumption of the system after changing the volume of the data. Again, this measurement comes at the cost of accuracy. Other measurements could be the throughput (the number of recommendations a system can do per second) or the latency (the time required for making a recommendation).

2.2 Recommender System Libraries

This section contains an overview of a few selected recommender system libraries, which attempt to support researchers in finding answers to their questions. They do so by providing a framework that includes a range of implemented recommendation algorithms and also allows for the easy addition of new ones. Moreover, they may provide means for their evaluation and ultimately comparison with already existing algorithms. Because open-source software provides several advantages, such as the low cost and the opportunity to personally examine provided code for correctness etc., the focus of this section lies solely on such libraries. Table 2.2 lists these libraries together with some basic information and the recommendation approaches supported by the inclusion of at least one algorithm in that specific category. Although there are many other more or less interesting frameworks, most of them are either not being actively developed anymore, focus on one specific aspect of recommender systems, are not available via an open-source license, or simply lack proper documentation.

Name	MyMediaLite	PREA	LensKit	Mahout
License	GPL 3	FreeBSD	LGPL 2.1	Apache 2.0
Last update	2015-12-31	2014-06-05	2015-11-10	2016-06-13
Programming language	C#	Java	Java	Java
Recommendation Approaches				
Baselines	✔	✔	✔	-
Content-based	✔	-	-	-
Collaborative filtering	✔	✔	✔	✔
Demographic	✔	-	-	-
Knowledge-based	-	-	-	-
Community-based	✔	-	-	-
Hybrid	✔	-	-	-

Table 2.2: Overview of various recommender system libraries and offered recommendation approaches.

2.2.1 MyMediaLite

MyMediaLite is an open source library of recommender system algorithms developed by Gantner, Rendle, Freudenthaler, et al. (2011) at the University of

Hildesheim. Its target audience include both researchers in the area, as well as business users searching for an existing framework to use or build upon. This is also reflected by the website's list of users, which includes a large number of universities and also a few commercial users such as the BBC[1]. (Gantner, Rendle, Drumond, et al., 2015) The site also offers an introduction, a few examples and a quite thorough API documentation. Since the library is written in C#, and due to the free .NET implementation Mono, it runs on many major platforms such as Windows, OS X and Linux. Additionally, it can be called from other programming languages like Python or Ruby via the .NET specific implementations IronPython and IronRuby respectively.

The library does offer a very broad range of supported recommender approaches. These include several baseline algorithms, such as those described by the Netflix Grand Prize winners, which take biases for users, items, time and frequencies into account. (Koren, 2009; Koren, 2010) Collaborative filtering techniques make up the major part of the framework by offering the usage of either explicit feedback (for example star ratings on a scale of 1 to 5) in case of rating prediction, or positive-only implicit feedback (for instance purchase actions) in case of item prediction. Among the collaborative filtering methods are simple slope one predictors as outlined by Lemire and Maclachlan (2005), various k-nearest neighbor models including Amazon's by Linden, B. Smith, and York (2003), an algorithm involving simultaneous clustering of users and items as proposed by George and Merugu (2005) and more sophisticated latent factor models. The latter include the Probabilistic Matrix Factorization (PMF) algorithm by Salakhutdinov and Mnih (2007) and a log-linear model for dyadic prediction using latent features as described by Menon and Elkan (2010), which can both be parallelized based on the idea of Gemulla et al. (2011) and updated online as depicted by Rendle and Schmidt-Thieme (2008). Further algorithms are comprised of a matrix factorization technique using factor-wise learning by Bell, Koren, and Volinsky (2007) and the SVD++ algorithm introduced by Koren (2008). Since some algorithms can also take item or user attributes into account, content-based filtering as well as demographic methods are also supported. With gSVD++, there is also a hybrid recommender approach integrated, which is based on the SVD++ algorithm and adds metadata awareness to it. (Manzato, 2013) Last but not least, the framework also offers a community-based model

[1]http://www.bbc.co.uk/rd/projects/sibyl-recommender-system

(arguably also a hybrid model) that adds trust propagation to existing matrix factorization techniques. (Jamali and Ester, 2010)

2.2.2 PREA

Another open source library for recommender system algorithms is PREA, which is short for *Personalized Recommendation Algorithm Toolkit*. It was developed by J. Lee, Sun, and Lebanon (2012b) at the Georgia Institute of Technology and attempts to provide an easy mechanic for comparing various algorithms, thus it is geared more toward researchers. The website offers a comprehensive overview of the implemented features, a tutorial on all the major functions, a documentation embedded into the website as well as an API documentation made with Javadoc. (J. Lee, Sun, and Lebanon, 2014) Since the framework was written in Java, cross-platform usability is also provided.

The authors have implemented several baseline algorithms, memory-based collaborative filtering techniques and a large number of latent semantic models using matrix factorization. Among the collaborative filtering algorithms is a user-based one as described by Adomavicius and Tuzhilin (2005) and Su and Khoshgoftaar (2009) as well as an item-based one as proposed by Sarwar et al. (2001). Also, a few implemented extensions dealing with default voting and inverse user frequency as expressed by (Breese, Heckerman, and Kadie, 1998) can be found. The implemented Slope One algorithm illustrated by Lemire and Maclachlan (2005) concludes the available collaborative filtering methods.

According to a study conducted by the framework's authors, matrix factorization techniques provide the highest accuracy among those offered. (J. Lee, Sun, and Lebanon, 2012a) This is clearly reflected by the large amount of variants they offer with the PREA toolkit, namely regularized SVD by Paterek (2007), Non-negative Matrix Factorization (NMF) by D. D. Lee and Seung (2000), Probabilistic Matrix Factorization (PMF) by Salakhutdinov and Mnih (2007), Bayesian Probabilistic Matrix Factorization (BPMF) by Salakhutdinov and Mnih (2008) and Non-linear Probabilistic Matrix Factorization (NLPMF) by Lawrence and Urtasun (2009). Furthermore, the authors provide their rather recent Local Low-Rank Matrix Approximation (LLORMA) methods as outlined by J. Lee, Kim, et al. (2013) and J. Lee, S. Bengio, et al. (2014) in addition to other state-of-the art algorithms, which are Fast Nonlinear Principal Component

Analysis (NPCA) by Yu et al. (2009) and a rank-based recommender by Sun, Lebanon, and Kidwell (2011).

2.2.3 LensKit

LensKit was developed by M. D. Ekstrand, Ludwig, et al. (2011) for the purpose of preventing researchers from wasting time on reimplementing well-known algorithms. The idea is that researchers can easily compare their own algorithms against carefully implemented versions of the prior state-of-the-art algorithms. According to the authors, the framework's design was driven by three goals, namely modularity, clarity, and efficiency. Since it is implemented in Java, it runs on a Java Virtual Machine (JVM), and in succession is also accessible from other languages like Python and Ruby via their respective Java specific implementations Jython and JRuby. The website covers all the basics that are to be expected, such as a documentation of the framework, the implemented algorithms and evaluation methods, as well as a comprehensive list of published research that has used the software in all kinds of forms. (M. Ekstrand et al., 2016)

In addition to the obligatory baseline algorithms, the framework offers highly-customizable and performant implementations of the original automatic user-user collaborative filtering algorithm as expressed by Resnick, Iacovou, et al. (1994), an item-item version of collaborative filtering as described in Sarwar et al. (2001) and Deshpande and Karypis (2004), and an SVD-like matrix factorization collaborative filtering algorithm using gradient descent as proposed by the Netflix prize winner in Funk (2006). More recently, the *Slope One* algorithm was implemented as suggested by Lemire and Maclachlan (2005). For the future, the authors originally planned to implement content-based and hybrid algorithms, which unfortunately are not available as of today. Still, the framework is being actively developed and existing algorithms are constantly improved.

2.2.4 Mahout

Apache Mahout is a scalable and robust machine learning library that includes some recommendation algorithms. It is maintained by The Apache Software Foundation (2016) and seems to be targeted towards a more practically orientated audience. The unique part of this framework is that some of the algorithms support Apache Hadoop, specifically its MapReduce programming model, and the more recent Apache Spark for distributed computing. Although the official website feels a little unorganized, as does the documentation, the authors offer a list of related books, articles, tutorials, coursework, talks and many other background materials.

As for the available algorithms, the framework's recommender part consists solely of collaborative filtering algorithms. These include user-based and item-based neighborhood models as well as a few matrix factorization algorithms. The latter include models using Alternating Least Squares (ALS) with either explicit or implicit data as described by Hu, Koren, and Volinsky (2008) and Zhou et al. (2008), a simplified version of the SVD++ algorithm as proposed by Koren (2008) and a parallel Stochastic Gradient Descent (SGD) implementation, which can be traced back to Recht et al. (2011).

2.3 Recommendation Algorithms

This section covers a selected range of recommendation algorithms implemented by the *MyMediaLite* framework. All the algorithms are trained using the given (user, item, rating) triples, enabling them to predict a previously unknown rating for a specific item and user. Some of the algorithms also take additional input, which will be mentioned separately. The purpose of this section is to give the reader of this thesis an understanding of how the algorithms work in general, how the training and prediction is done, and what hyperparameters are subject to a tuning process.

2.3.1 Baselines

2.3.1.1 Random

The *Random* predictor simply uses a random value between the boundaries of the rating scale for prediction. It does not have a separate training process or hyperparameters to be tuned.

2.3.1.2 GlobalAverage

The *GlobalAverage* predictor calculates the average of all ratings in the given dataset and uniformly uses this global average for prediction. The training process consists solely of the averaging of the ratings and does not require hyperparameters to be tuned.

2.3.1.3 UserAverage

Prediction in the *UserAverage* baseline works by calculating the average of ratings on a user basis, which also makes up the training process together with calculating the global average of all ratings, which is used as a fallback mechanism for new users. Again, no hyperparameters are involved in this predictor.

2.3.1.4 ItemAverage

Similar to the *UserAverage* predictor, the *ItemAverage* calculates the average of ratings on an item basis, while the rest works completely analogously.

2.3.1.5 UserItemBaseline

It is quite frequent that certain users tend to give generally higher or lower ratings than others and the same also applies to the ratings of certain items. An improvement on the *GlobalAverage* predictor can therefore be made by incorporating said biases into the prediction of a rating. The implementation of this predictor is based on the baseline estimate described by Koren (2010) with a few differences, where one is the support of several iterations of alternating optimization instead of just one. The system tries to solve the following least squares problem:

$$\min_{b_*} \sum_{(u,i)\in K} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_u \|b_u\|^2 + \lambda_i \|b_i\|^2 \tag{2.2}$$

where r_{ui} is user u's rating for item i, K the set of (u,i) pairs for which r_{ui} is known, μ the global average, b_u the user bias, b_i the item bias, and λ_u and λ_i the regularization constants to combat overfitting by shrinking the estimates towards the baseline default.

During the training phase, in order to solve the least squares problem, the biases are alternately calculated by applying the Alternating Least Squares (ALS) method as follows:

$$b_u = \frac{\sum_{i:(u,i)\in K}(r_{ui} - \mu - b_i)}{\lambda_i + |\{i \mid (u,i) \in K\}|}, \qquad b_i = \frac{\sum_{u:(u,i)\in K}(r_{ui} - \mu - b_u)}{\lambda_u + |\{u \mid (u,i) \in K\}|} \tag{2.3}$$

Prediction of a rating $\hat{r}_{ui}$ for user u and item i is then done by simply adding the biases to the global average as follows:

$$\hat{r}_{ui} = \mu + b_u + b_i \tag{2.4}$$

Hyperparameters to optimize for this algorithm include the regularization constants λ_u and λ_i, as well as the number of iterations for the ALS method.

2.3.2 Content-Based

2.3.2.1 ItemAttributeKNN

The *ItemAttributeKNN* represents a content-based recommendation algorithm as it uses binary item attributes as the basis. Its implementation is based on the neighborhood model depicted by Koren (2010). Due to the underlying *UserItemBaseline* algorithm, it again includes an iterative process of alternating optimization. Koren (2010) states that this algorithm became very popular due to its intuitiveness and the relatively simple implementation. Further, predictions can easily be reasoned to the user and new ratings are taken into account as soon as the user enters them.

During the training phase, the underlying baseline algorithm is trained as explained in Section 2.3.1.5. Afterwards, similarities between all item pairs are calculated using the binary item attributes as input. Several similarity measures are available for selection including the Jaccard index, the cosine similarity and conditional probabilities.

In the event of a new user or item, the algorithm falls back to the *UserItemBaseline* algorithm. Otherwise, prediction is done by first selecting k neighboring items that have the highest positive correlation. For each found item, the correlations are then multiplied by the difference of the user's rating and the baseline estimate of the *UserItemBaseline* algorithm, which accounts for user and item biases as explained in Section 2.3.1.5. These weighted ratings are then summed up and divided by the sum of the weights (the correlations) and eventually used as prediction for the given user and item. Ratings are thereby restricted to the boundaries of the rating scale. More formally:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k(i;u)} s_{ij}(r_{uj} - b_{uj})}{\sum_{j \in S^k(i;u)} s_{ij}} \tag{2.5}$$

where $\hat{r}_{ui}$ is the prediction for the rating of user u and item i, b_{ui} the baseline estimate, $S^k(i;u)$ the k items rated by u which are most similar to i, and s_{ij} the similarity between item i and item j.

Hyperparameters for this predictor are the chosen similarity measure and the number of neighbors k. The *UserItemBaseline* requires the usual optimization as explained in Section 2.3.1.5.

2.3.3 Collaborative Filtering

2.3.3.1 ItemKNN

ItemKNN is exactly the same implementation as the *ItemAttributeKNN* (in fact *ItemAttributeKNN* is derived from *ItemKNN*), the only difference being the correlation calculations and their basis, which are the ratings in this case. Section 2.3.2.1 contains further details on the implementation.

Similarity measurements available for this predictor, such as the Pearson correlation coefficient, are implemented using a regularization parameter for shrinking the estimates towards the baseline defaults as follows:

$$s_{ij} = \frac{n_{ij}}{n_{ij} + \lambda} p_{ij} \tag{2.6}$$

where s_{ij} is the similarity measure, n_{ij} the number of users who co-rated items i and j, λ the regularization parameter, and p_{ij} the Pearson correlation coefficient.

The shrinkage parameter λ would then be subject to hyperparameter optimization for this predictor. Also the *UserItemBaseline* requires the usual optimization as explained in Section 2.3.1.5.

2.3.3.2 SlopeOne

The *SlopeOne* algorithm is implemented as presented in the paper of Lemire and Maclachlan (2005) using their *Weighted Slope One* scheme. The authors attempt to provide a solid algorithm for real-world situations, as it is supposed to be easy in implementation, be updateable on-the-fly, possess an efficient query time, provide valid ratings to users with only a few ratings, and is accurate within reason. In order to be able to predict a rating for an item i, the system finds users who also rated item i and accounts for average differences in common ratings of other items.

The training phase therefore consists of the calculation of the average differences in ratings between all item pairs and the frequencies of ratings for each pair as follows:

$$\text{dev}_{i,j} = \sum_{u \in S_{i,j}(U)} \frac{r_{ui} - r_{uj}}{|S_{i,j}(U)|} \tag{2.7}$$

where $\text{dev}_{i,j}$ is the average rating deviation between item i and j, r_{ui} and r_{uj} the ratings of user u for item i and j respectively, and $S_{i,j}(U)$ the set of all users who rated both items i and j.

For new users and items, the global average is again used as a fallback mechanism. Prediction of a rating $\hat{r}_{ui}$ for a user u and item i is otherwise done as follows:

$$\hat{r}_{ui} = \frac{\sum_{j \in S(u)-\{i\}} (\text{dev}_{i,j} + r_{uj})|S_{i,j}(U)|}{\sum_{j \in S(u)-\{i\}} |S_{i,j}(U)|} \tag{2.8}$$

where $S(u)$ is the set of all items rated by user u.

No hyperparameters are involved in this algorithm and therefore it requires no separate optimization.

2.3.3.3 MatrixFactorization

Matrix factorization techniques generally combine good scalability, accuracy and flexibility. The *MatrixFactorization* algorithm found in *MyMediaLite* is based on the elucidations of Koren, Bell, and Volinsky (2009) and implements a basic matrix factorization model including the global average as bias. In order to learn the latent factor vectors for users and items, the system uses Stochastic Gradient Descent (SGD) as described by the Netflix prize winner in Funk (2006), which is supposed to provide ease of implementation and a fast running time. Specifically, the system attempts to minimize the regularized squared error on the set of known ratings K as follows:

$$\min_{q*,p*} \sum_{(u,i) \in K} (r_{ui} - \mu - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.9}$$

where r_{ui} is user u's rating for item i, μ the global average, q_i the item factors vector, p_u the user factors vector and λ the regularization constant to avoid overfitting.

The training phase of the algorithm therefore iteratively loops over all ratings, attempts to predict this rating, computes the resulting error as stated in equation 2.10 and finally adapts the parameters by a magnitude proportional to γ in the opposite direction of the gradient as shown in equation 2.11.

$$e_{ui} = r_{ui} - (\mu + q_i^T p_u) \tag{2.10}$$

$$p_u = p_u + \gamma(e_{ui} q_i - \lambda p_u), \qquad q_i = q_i + \gamma(e_{ui} p_u - \lambda q_i) \tag{2.11}$$

To predict the rating $\hat{r}_{ui}$ of user u for item i, the system again takes the global average as a fallback for new items or users. In all other cases, it takes the global average μ and adds the dot product of the user factors vector p_u and the item factors vector q_i as stated in equation 2.12. Results will further be restricted by the boundaries of the given rating scale.

$$\hat{r}_{ui} = \mu + q_i^T p_u \tag{2.12}$$

This method has multiple hyperparameters to be tuned: the learning rate γ an optional decay for it, the regularization constant λ, the number of factors to be used, and the number of iterations.

2.3.3.4 BiasedMatrixFactorization

The *BiasedMatrixFactorization* implementation in *MyMediaLite* is based on the paper of Salakhutdinov and Mnih (2007), who demonstrate an alternative approach to the linear factor model described in Section 2.3.3.3. *Probabilistic Matrix Factorization* assumes that given the user and item features, the distribution over the corresponding ratings is given by a Gaussian distribution with the mean being the dot product between user and item feature vectors plus some noise represented by the standard deviation σ^2.

$$P(r_{ui}|p_u, q_i, \sigma^2) = \mathcal{N}(r_{ui}|g(q_i^T p_u), \sigma^2) \tag{2.13}$$

r_{ui} is thereby the rating for user u and item i, p_u the user factors vector, q_i the item factors vector, and $\mathcal{N}(x|\mu, \sigma^2)$ the probability density function with mean μ and variance σ^2. Further, since the Gaussian model can predict ratings outside of the allowed range, the dot product of the user and item factors vector are passed through the logistic function $g(x) = \frac{1}{1+e^{-x}}$ in order to keep them in the interval $[0, 1]$. This of course requires the input ratings to be mapped to the interval $[0, 1]$ as well.

In addition, the user and item feature vectors for N users and M items are initialized with zero-mean Gaussian priors:

$$P(p|\sigma_p^2) = \prod_{u=1}^{N} \mathcal{N}(p_u|0, \sigma^2 \mathrm{I}), \qquad P(q|\sigma_q^2) = \prod_{i=1}^{M} \mathcal{N}(q_i|0, \sigma^2 \mathrm{I}) \tag{2.14}$$

While Salakhutdinov and Mnih (2007) do not include any biases within their model, Gantner, Rendle, Freudenthaler, et al. (2011) add again a global, user and item bias to it. Maximizing the posterior probability estimation is then equivalent to minimizing the following on the set of known ratings K:

$$\begin{aligned} \min_{q*,p*,b*} \frac{1}{2} \sum_{(u,i)\in K} \Big(r_{ui} - g(\mu + b_u + b_i + q_i^T p_u)\Big)^2 \\ + \frac{\lambda_u}{2}\Big(\sum_{u=1}^{N} \|p_u\|^2 + \|b_u\|^2\Big) + \frac{\lambda_i}{2}\Big(\sum_{i=1}^{M} \|q_i\|^2 + \|b_i\|^2\Big) \end{aligned} \tag{2.15}$$

where μ is the global average, b_u the user bias, b_i the item bias, and λ_u and λ_i the regularization constants.

Again, as in the *MatrixFactorization* algorithm described in Section 2.3.3.3, learning is done using Stochastic Gradient Descent while iterating over all observed ratings. Error calculation is done as follows:

$$e_{ui} = r_{ui} - g(\mu + b_u + b_i + q_i^T p_u) \tag{2.16}$$

Adjustment of the latent factors is given by Equation 2.18, while bias adjustment is given by Equation 2.17. Bias adjustment has an additional learning rate γ_b, which is bound to the overall learning rate γ, and an additional regularization constant λ_b, which is bound to the user and item biases λ_i and λ_u.

$$\begin{aligned} b_u &= b_u + \gamma_b\gamma\Big(e_{ui}g'(\mu b_u + b_i + q_i^T p_u) - \lambda_b\lambda_u b_u\Big), \\ b_i &= b_i + \gamma_b\gamma\Big(e_{ui}g'(\mu b_u + b_i + q_i^T p_u) - \lambda_b\lambda_i b_i\Big) \end{aligned} \tag{2.17}$$

$$\begin{aligned} p_u &= p_u + \gamma\Big(e_{ui}g'(\mu b_u + b_i + q_i^T p_u)q_i - \lambda_u p_u\Big), \\ q_i &= q_i + \gamma\Big(e_{ui}g'(\mu b_u + b_i + q_i^T p_u)p_u - \lambda_i q_i\Big) \end{aligned} \tag{2.18}$$

where $g'(x)$ is the derivative of the logistic function $g'(x) = \frac{e^x}{(1+e^x)^2}$.

Prediction of a rating $\hat{r_{ui}}$ for user u and item i is finally done as follows:

$$\hat{r}_{ui} = g(\mu + b_u + b_i + q_i^T p_u) \tag{2.19}$$

The *BiasedMatrixFactorization* requires several hyperparameters to be tuned: the regularization terms λ_b, λ_u and λ_i, the learning rates γ_b, γ and an optional decay, the number of factors, and the number of iterations. Optimization of the learning rate decay is no longer important, due to the SGD implementation in this algorithm also supporting the *bold driver* algorithm, which automatically adjusts the learning rate.

2.3.3.5 SVDPlusPlus

SVDPlusPlus is implemented as proposed in the paper of Koren (2008). It is based on the *MatrixFactorization* algorithm described in Section 2.3.3.3 and integrates the users' implicit preferences for items into the system, which improves accuracy especially for those users who have not provided much explicit feedback. The implicit feedback is thereby modeled as a separate latent factors vector y_j for each item j. Optimization is carried out using SGD and solving the following least squares problem for all known ratings in the set K:

$$\begin{aligned} \min_{q_*, p_*, y_*, b_*} \sum_{(u,i) \in K} \left(r_{ui} - \mu - b_u - b_i - q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) \right)^2 \\ + \lambda \left(\lambda_b b_u^2 + \lambda_b b_i^2 + \|q_i\|^2 + \|p_u\|^2 + \sum_{j \in N(u)} \|y_j\|^2 \right) \end{aligned} \tag{2.20}$$

where r_{ui} is the rating of user u for item i, μ the global average, b_u the user bias, b_i the item bias, q_i the item factors vector, p_u the user factors vector, and y_j the implicit feedback factors vector. The difference to *MatrixFactorization* is made up by the combination of the user factors vector and the implicit point of view as expressed by the term $\left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$. $N(u)$ thereby contains all items for which user u provided an implicit preference.

During training, SGD is applied by iterating over all known ratings in K, calculating the error given by Equation 2.21 and adjusting the biases as well as the factors by a magnitude proportional to γ in the opposite direction of the gradient as shown in Equation 2.22 and Equation 2.23 respectively. Further, overfitting is battled by the regularization constant λ, which is applied to each optimization target. For the adjustment of the biases, Gantner, Rendle, Freudenthaler, et al. (2011) again add a separate learning rate γ_b, bound to the overall learning rate, and regularization constant λ_b, bound to the overall regularization constant.

$$e_{ui} = r_{ui} - \mu + b_u + b_i + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) \tag{2.21}$$

$$b_u = b_u + \gamma_b \gamma \left(e_{ui} - \lambda_b \lambda b_u \right), \qquad b_i = b_i + \gamma_b \gamma \left(e_{ui} - \lambda_b \lambda b_i \right) \tag{2.22}$$

$$\begin{aligned}
p_u &= p_u + \gamma\left(e_{ui}q_i - \lambda p_u\right), \\
q_i &= q_i + \gamma\left(e_{ui}\left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j\in N(u)} y_j\right) - \lambda q_i\right), \\
y_j &= y_j + \gamma\left(e_{ui}|N(u)|^{-\frac{1}{2}} q_i - \lambda y_j\right)
\end{aligned} \tag{2.23}$$

Once training is done, the recommender can predict a rating $\hat{r}_{ui}$ for user u and item i as follows:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T\left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j\in N(u)} y_j\right) \tag{2.24}$$

Hyperparameters to be adjusted are comprised of the regularization constants λ and λ_b, the learning rates γ, γ_b as well as an optional decay, the number of factors, and finally the number of learning iterations.

2.3.4 Hybrid

2.3.4.1 GSVDPlusPlus

The *GSVDPlusPlus* algorithm is implemented as described by Manzato (2013). It is based on the *SVD++* algorithm and incorporates item attributes such as genres, which the paper's authors denote as the set $G(i)$, thus providing the name *gSVD++*. For this set of item attributes, a metadata factors vector x_g is introduced, which contains the factors for possible item descriptions. The rest of the algorithm works completely analogously using again SGD to solve the following least squares problem:

$$\begin{aligned}\min_{q_*,p_*,x_*,y_*,b_*} \sum_{(u,i)\in K} \Bigg(& r_{ui} - \mu - b_u - b_i \\ & - \Big(q_i + |G(i)|^{-\alpha} \sum_{g\in G(i)} x_g\Big)^T \Big(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j\in N(u)} y_j\Big)\Bigg)^2 \\ & + \lambda\Big(\lambda_b b_u^2 + \lambda_b b_i^2 + \|q_i\|^2 + \|p_u\|^2 + \sum_{g\in G(i)} \|x_g\|^2 + \sum_{j\in N(u)} \|y_j\|^2\Big)\end{aligned} \tag{2.25}$$

where r_{ui} is the rating of user u for item i, K the set of (u,i) pairs for which r_{ui} is known, μ the global average, b_u the user bias, b_i the item bias, q_i the item factors vector, p_u the user factors vector, x_g the metadata factors vector, and y_j the implicit feedback factors vector. $G(i)$ is further the set of descriptions for item i, which is taken into account only if metadata is available. This is done by setting the parameter α to 1 or 0, depending on the availability of metadata, thus making the algorithm identical to *SVD++* if set to 0.

During the training phase, SGD is applied by iterating over all ratings in K and calculating the error given by Equation 2.26. Using this calculated error, the biases and latent factors are adjusted by moving them in the opposite direction of the gradient as depicted in Equation 2.27 and Equation 2.28 respectively. Factors for metadata are further updated for all $g \in G(i)$, and for implicit feedback for all $j \in N(u)$. Overfitting is also again battled by the regularization constant λ, which is applied to each optimization target.

$$\begin{aligned} e_{ui} = r_{ui} - \Bigg(& \mu + b_u + b_i \\ & + \Big(q_i + |G(i)|^{-\alpha} \sum_{g\in G(i)} x_g\Big)^T \Big(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j\in N(u)} y_j\Big)\Bigg)\end{aligned} \tag{2.26}$$

$$b_u = b_u + \gamma_b\gamma\Big(e_{ui} - \lambda_b\lambda b_u\Big), \qquad b_i = b_i + \gamma_b\gamma\Big(e_{ui} - \lambda_b\lambda b_i\Big) \tag{2.27}$$

$$
\begin{aligned}
p_u &= p_u + \gamma \left(e_{ui} \left(q_i + |G(i)|^{-\alpha} \sum_{g \in G(i)} x_g \right) - \lambda p_u \right), \\
q_i &= q_i + \gamma \left(e_{ui} \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) - \lambda q_i \right), \\
x_g &= x_g + \gamma \left(e_{ui} |G(i)|^{-\alpha} \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) - |G(g)|^{-1} x_g \right), \\
y_j &= y_j + \gamma \left(e_{ui} |N(u)|^{-\frac{1}{2}} \left(q_i + |G(i)|^{-\alpha} \sum_{g \in G(i)} x_g \right) - |N(j)|^{-\frac{1}{2}} y_j \right)
\end{aligned}
\tag{2.28}
$$

Prediction of a rating $\hat{r}_{ui}$ for user u and item i is then carried out as follows:

$$
\begin{aligned}
\hat{r}_{ui} = {} & \mu + b_u + b_i \\
& + \left(q_i + |G(i)|^{-\alpha} \sum_{g \in G(i)} x_g \right)^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)
\end{aligned}
\tag{2.29}
$$

Hyperparameters to be adjusted are identical to the *SVDPlusPlus* algorithm and are thus comprised of the regularization constants λ and λ_b, the learning rates γ, γ_b as well as an optional decay, the number of factors, and the number of learning iterations.

2.4 Knowledge Discovery Process

Finding useful information or knowledge in the ever-growing amount of available data in various research areas calls for a unified process: the *Knowledge Discovery in Databases Process* or simply *Knowledge Discovery Process (KDP)*. Fayyad, Piatetsky-Shapiro, and Smyth, 1996b define this process as

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

or more detailed as

> “the process of using the database along with any required selection, preprocessing, subsampling, and transformations of it; to apply data mining methods (algorithms) to enumerate patterns from it; and to evaluate the products of data mnining to identify the subset of the enumerated patterns deemed ‘knowledge’.”

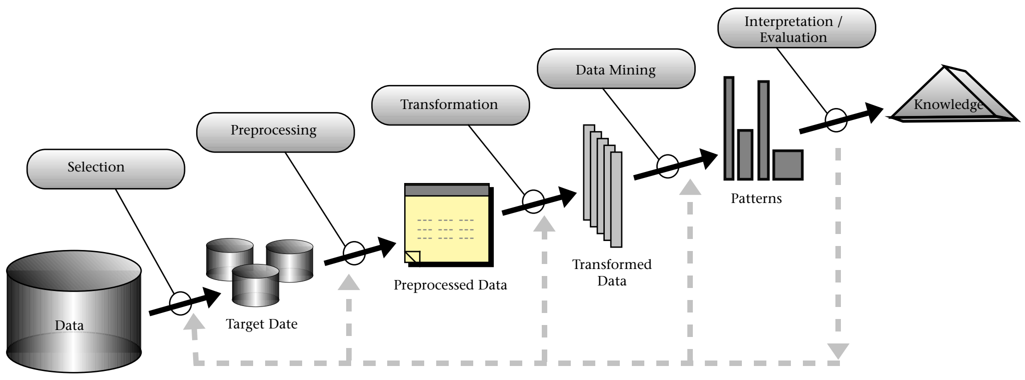


Figure 2.1: The *Knowledge Discovery Process (KDP)* and its five major steps: selection, preprocessing, transformation, data mining and interpretation/evaluation. The process is iterative and may contain loops as depicted by the dashed arrows. Depiction taken from Fayyad, Piatetsky-Shapiro, and Smyth (1996a).

Figure 2.1 is a graphical illustration of said process and its five major steps: selection, preprocessing, transformation, data mining and interpretation/evaluation. These steps are to be understood iteratively and may thus contain loops between any of the steps. It is further noteworthy that, according to Cios et al. (2007), there are several other models that will not be considered further here. Instead, the focus lies on the model of Fayyad, Piatetsky-Shapiro, and Smyth (1996b) and Fayyad, Piatetsky-Shapiro, and Smyth (1996c), which will be discussed and applied to the field of recommender systems using additional information from Geyer-Schulz and Hahsler (2002).

2.4.1 Selection

After having a clear understanding of the application domain and with a goal in mind, the first major step in the KDP is the selection of a target data set. Data can come from many sources and also be retrieved in many ways, for example by crawling a website. In the case of recommender systems, the purpose of the selection process is clearly to have enough usable information as input for the system.

2.4.2 Preprocessing

Second comes the data cleaning and preprocessing step, which includes removing unnecessary data, dealing with missing values, and handling of outliers. For recommender systems, this could include the removal of anything but the (user, item, rating) triples, outlier detection within the ratings, removal of duplicate ratings and the like.

2.4.3 Transformation

Next, the preprocessed data is transformed into a format useful for the following data mining step. It also includes the extraction of relevant features to represent the data and a possible application of dimensionality reduction. In the case of recommender systems, this may include finding valuable implicit data and transforming it into a proper format.

2.4.4 Data Mining

At the heart of the knowledge discovery process lies the data mining step. With the process goal in mind and having found algorithms that properly match these goals, a number of algorithms are chosen in order to search for patterns within the data. These patterns could be classification rules, decision trees, regression models, trends, clustering, and others. In a recommender system, the data mining part could be described as learning the parameters of the underlying model using the given dataset.

2.4.5 Interpretation/Evaluation

Finally, the patterns found have to be made understandable for humans and be integrated into the productive system or simply documented and reported. It is possible that the retrieved knowledge conflicts with previously believed knowledge, which has to be resolved in this step. A recommender system inherently interprets the patterns for the user by showing him the results as ratings or recommendations. Eventually, the system's developer should also evaluate the system using any of the properties explained in Section 2.1.5.

2.5 The Steam Platform

Steam is the social entertainment platform of Valve Corporation, which was founded in 1996 by Gabe Newell. Starting with the award winning video games Half-Life and Counter-Strike, a former modification for it, the company felt the necessity for an easier update mechanism for the Counter-Strike client, which was previously distributed via FTP. The solution, Steam, has since become the world's largest online gaming platform with over 4500 pieces of content, namely games from Valve and third-party companies as well as non-gaming applications. More recently, Valve released live broadcasting, music, movies, their own operating system *SteamOS* and various hardware. Valve further claims to have about 125 million active users with more than 9.5 million concurrent players at peaks, making up 2 billion minutes of playtime per day. Although no exact numbers are known, the company is valued between 2 and 4 billion dollars. (Chiang, 2011; Dunkle, 2015; Valve Corporation, 2016b)

The Steam store is available via the web[2], but most of Steam's features are bound to the client depicted in Figure 2.2. Next to the store itself, the client provides access to the users application library and the community where users can socially interact with each other, become friends and of course play games together. This social network can be especially interesting for various recommender algorithms. As for the applications, Steam does not only track which game is owned by what users, but also the time users are spending on them, which is yet another appealing feature of the platform as it provides domain

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

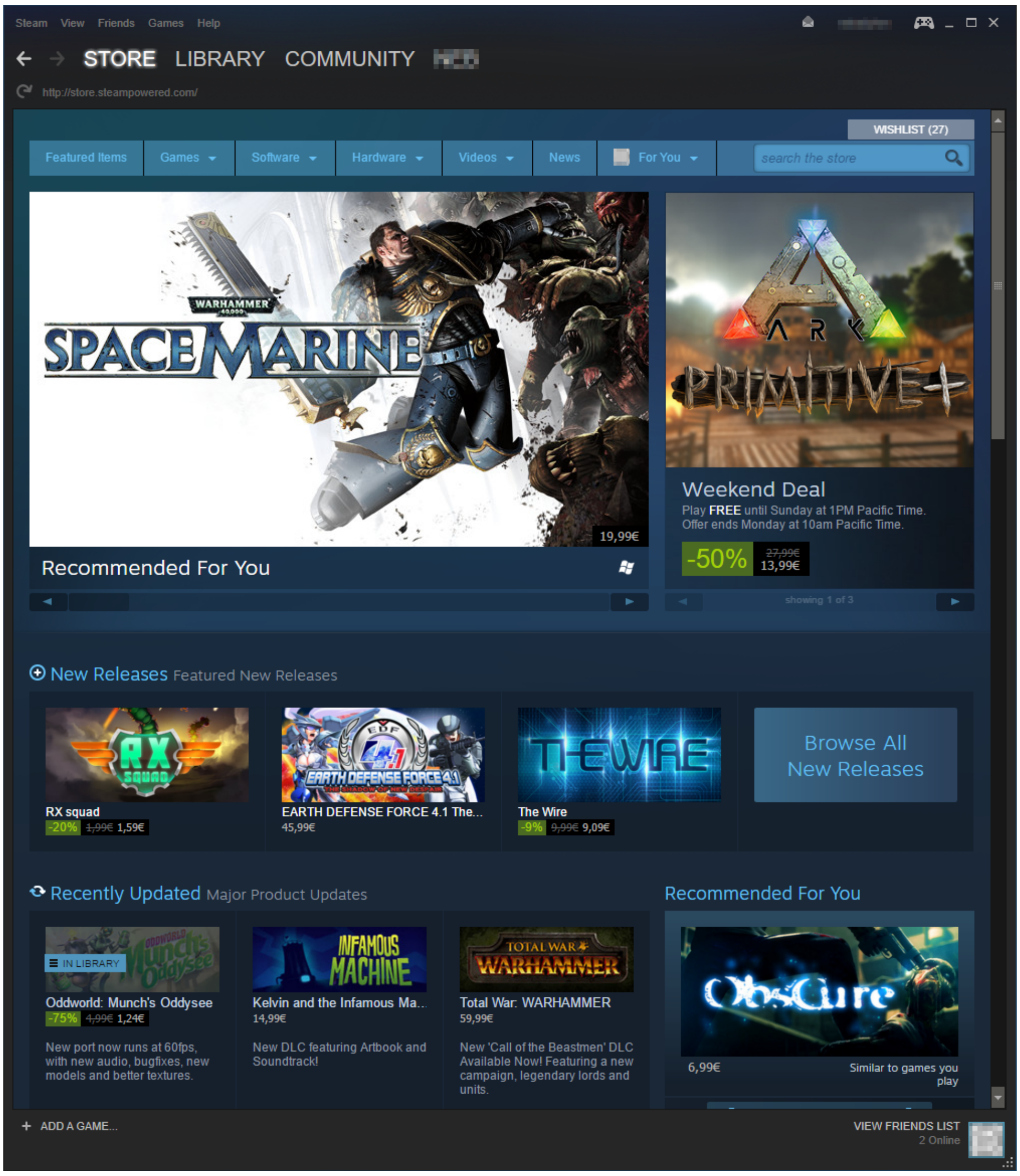

Steam View Friends Games Help
STORE LIBRARY COMMUNITY
http://store.steampowered.com/
WISHLIST (27)
Featured Items
Games
Software
Hardware
Videos
News
For You
search the store
Recommended For You
19,99€
Weekend Deal
Play FREE until Sunday at 1PM Pacific Time.
Offer ends Monday at 10am Pacific Time.
-50% 27,99€ 13,99€
New Releases Featured New Releases
RX squad -20% 1,99€ 1,59€
EARTH DEFENSE FORCE 4.1 The... 45,99€
The Wire -9% 9,99€ 9,09€
Browse All New Releases
Recently Updated Major Product Updates
Oddworld: Munch's Oddysee -75% 4,99€ 1,24€
New port now runs at 60fps, with new audio, bugfixes, new models and better textures.
Kelvin and the Infamous Ma.. 14,99€
New DLC featuring Artbook and Soundtrack!
Total War: WARHAMMER 59,99€
New 'Call of the Beastmen' DLC Available Now! Featuring a new campaign, legendary lords and units.
Recommended For You
6,99€
Similar to games you play
ADD A GAME...
VIEW FRIENDS LIST
2 Online

Figure 2.2: The store view of the Steam client with obfuscated personal information. Screenshot taken by the author.

specific implicit information of the users preferences. Further, applications can be reviewed by the users and rated within a binary scale, specifically a thumbs-up or thumbs-down, thus providing explicit feedback that can be utilized as the basis for the majority of recommendation algorithms.

The client, more precisely the store, also offers recommendations in various ways. Besides the obligatory advertisement for top sellers and new releases on top of the store page, individual recommendations are made through one of three distinct features: the *Discovery Queue*, *Steam Curators*, and the *Recommendation Feed*. To the best of the author's knowledge, the details on the technologies behind these components are unavailable to the public. Nevertheless, the following list describes each of the features as accurately as possible in accordance with Valve Corporation (2016a).

- The *Discovery Queue* is a queue of 12 titles personalized for each user and a mixture of recommended titles, new titles, and popular releases. After browsing through the queue, users can start a new queue made up of 12 entirely new titles again. According to Schreier (2014), the Discovery Queue prioritizes popular titles, previously unseen by the user.
- *Steam Curators* are individuals, organizations, and groups that manually recommend content to the users, who can follow selected curators in order to see their recommendations.
- The *Recommendation Feed*, last but not least, is a nearly endless list of recommendations based on the titles a user and his or her friends have been playing. Yet again, the list also contains titles that are simply well-reviewed. Schreier (2014) also mentions that these recommendations are mainly based on user-applied tags and accordingly games with similar tags.

3 Related Work

As of today, the domain of video games in recommendation systems is relatively unexplored. Yet, video games are played by millions of people worldwide, its industry is worth billions of dollars and it heavily competes with the movie and music industries. According to Egenfeldt-Nielsen, J. H. Smith, and Tosca (2016), the global revenues of the video game industry reached \$64.9 billion in 2014, positioning it right between the movie industry (\$90 billion) and the music industry (\$20.97 billion). Additionally, while the music industry struggles to keep its revenues at all, the annual growth rate between 2013 and 2018 of the video game industry is expected to be 9.6%, compared to 4.5% in the movie industry. Entertainment Software Association (ESA) (2016) further reports that video games are frequently played by persons of all ages, both male and female.

Sifa, Bauckhage, and Drachen (2014) claim to be the first to apply recommender systems to the domain of digital games. They present two approaches to Top-N recommendation systems, where the user is presented a list of N recommendations, namely a matrix factorization and a user-based neighborhood model operating in reduced dimensions. Both models are based on archetypal analysis, a method similar to cluster analysis, thus clustering the users into k archetypes. The data used for this analysis is comprised of implicit feedback, specifically information on game ownership and playtimes, with the ultimate goal of recommending those games, which have the highest predicted playtime. The authors compare their algorithms to several baselines and an item-based neighborhood model, which they were able to beat.

Microsoft revealed in the paper by Koenigstein et al. (2012) how the recommendation system in the Xbox Live Marketplace works. As input for their algorithm, Microsoft infers like/dislike ratings based on game ownership. The positive ratings consist of the games a user owns, whereby the user's playtimes are utilized in order to remove games that were hardly ever played. Further, they generate an equal number of negative ratings by randomly picking games a user does not own, which is done in proportion to their popularity. Prediction is then carried out using a probabilistic matrix factorization approach including item biases.

Cosley et al. (2003) examine various effects of recommender system interfaces as well as different rating scales. Although they hesitate to draw conclusions based on the MAE results obtained from common collaborative filtering algorithms, they found that the prediction accuracy drops in relation to a rising granularity of the rating scale. More interestingly, rating scales with 5 or 6 stars performed about the same, while binary and also 10-star rating scales performed worse.

The rating scale offered by a recommendation system also affects user satisfaction according to Sparling and Sen (2011). The paper's authors found that a 5-star rating scale is strongly preferred over a binary rating scale and even more preferred over a unary or 100-point scale. They also believe that their findings apply to many different domains and further recommend designers of recommender systems to carefully evaluate different rating scales before deployment. Not only does the rating scale affect user satisfaction, it also affects the way a user experiences items. For further research, the authors suggest the topic of adaptive rating scales, which change from smaller ratings scales for fresh users to larger scales for more experienced users. The idea is that, as a system learns from the user, more fine-grained information would yield better results.

While other researchers already found that user ratings are noisy, which could limit the predictive power of a recommender system, Kluver et al. (2012) further explored how much preference information is contained in ratings as well as predictions within different rating scales. The authors found that there is a "sweet spot" in rating scale granularity, which balances between input preference information and rating noise. The results are similar to the results of Sparling and Sen (2011), with the binary rating scale requiring remarkably more ratings per user to deliver the same information a 5-star rating scale achieves with fewer ratings. Once again, there is little to no reason to pick a 100-point rating scale over a 5-star scale.

With the above research in mind, this thesis explores the domain of video games by comparing various recommendation algorithms. An attempt at inferring explicit user ratings from implicit feedback is made in order to compare them to actual explicit ratings using the selected recommendation algorithms. The implicit feedback will further be used to compare different rating scales, specifically binary to quinary rating scales, while using the same recommendation algorithms.

4 Materials and Methods

This chapter contains the materials and methods used within this thesis. First, the development environment, programming languages, and libraries utilized are listed in order to provide reproducibility of the results following in Chapter 5. Second, the knowledge discovery process used for conducting the experiments in order to answer the research questions is described. This contains the following major steps in the process: data selection, preprocessing, transformation, mining and evaluation as well as a thorough analysis of the dataset.

4.1 Development Environment

The following is a brief overview of the relevant hard- and software used for performing the experiments described in Section 4.2.

- Processor
  - Intel Core i5 @ 3.40 GHz
- Memory (RAM)
  - 32 GB
- System type
  - 64-bit Operating System, x64-based processor
- Operating System
  - Windows 10 Pro
- Integrated Development Environment (IDE)
  - Microsoft Visual Studio Enterprise 2015

The subsequent list contains programming languages and important libraries used for varying purposes.

- Data selection
  - Python 64-bit 2.7
  - Scrapy 1.0.3[1]
- Data preprocessing
  - Python 64-bit 3.5
- Data analysis
  - Python 64-bit 3.5
  - Python 64-bit 2.7
  - Snap.py 1.2[2]
- Data transformation
  - R 3.2.4
  - Python 64-bit 3.5
- Data mining and evaluation
  - C# 6.0
  - .NET Framework 4.5.2
  - MyMediaLite 3.11[3]
  - R 3.2.4
  - scmamp 0.2.5

4.2 Knowledge Discovery Process

In this section, the major steps of the knowledge discovery process are depicted. Since this is an iterative process that may contain loops, as stated by Fayyad, Piatetsky-Shapiro, and Smyth (1996b), the process steps contained in the following subsections may overlap sometimes. Furthermore, a separate data analysis step was inserted between the data preprocessing and transformation steps in contemplation of providing an adequate understanding of the dataset.

[1]http://scrapy.org/
[2]https://snap.stanford.edu/snappy/
[3]http://mymedialite.net/

4.2.1 Selection

In order to carry out the experiments, a comprehensive dataset in the domain of video games is required. The Steam platform is very well suited for this task, as it provides a large catalog of games, a massive user base and most importantly, binary user ratings as well as playtime for the games.

The first task in the knowledge discovery process was, therefore, the data acquisition from the Steam website. Valve offers a Web API that can be used to retrieve data in different formats such as JSON or XML, which substantially reduces the necessary effort. (Valve Developer Community, 2016) At the time of writing this thesis, some of the required data, mainly the user reviews for the games and thus also the ratings, were not accessible via the API. Most API features also require registration and are limited in certain ways, with the major limitation being a restriction to 100 000 calls to the API per day. (Valve Corporation, 2010) Due to these reasons, a combination of API usage and classical web scraping was applied. It goes without saying that the website's *Robots Exclusion Standard (robots.txt)* was respected, although it is only a convention according to Mitchell (2015).

Furthermore, there is no way to get an accurate list of users of the Steam platform via API or the website, especially since not all user profiles are publicly available. The list of users was thus extracted from the results of the *ReviewCrawler* described in Section 4.2.1.2. As a consequence, this method resulted in all users within the dataset having at least one review or rating respectively.

Eventually, four web crawlers were implemented in Python under the usage of Scrapy, an open source framework for crawling websites. Besides the obvious crawling of URLs and their content, Scrapy also provides the opportunity to extract only relevant and proper information that has to be defined earlier in order to store it in whatever format necessary. The framework was thus utilized to combine a part of the data preprocessing with the crawling process. Specifically, only the most important data from the HTML pages was extracted and written to CSV files on the fly. The following sections describe each crawler and the data retrieved during February 2016.

4.2.1.1 AppCrawler

The *AppCrawler* was implemented to retrieve a list of apps and their details from the Steam store. First, the API was utilized to get a full list of applications. This application list was then used in a second step to initialize crawling the details for each app, again making use of the API. As a side note, these two API methods were not listed at Valve's developer community wiki and were found via the forums instead. Table 4.1 lists only the most interesting attributes retrieved via the API, although all others were kept as well.

Attribute	Data type	Description
type	String	The type of app.
name	String	The app's title.
steam_appid	Integer	The app's unique identifier.
about_the_game	String	A detailed description of the game.
developers	List(String)	A list of the developers.
publishers	List(String)	A list of the publishers.
metacritic	Integer	The Metascore from Metacritic (`http://www.metacritic.com/`).
categories	List(Integer, String)	A list of integer and string pairs containing the category ID and its description. Categories describe the features of an app, such as "Single-player", "Multi-player", "Steam Cloud" etc.
genres	List(Integer, String)	A list of integer and string pairs containing the genre ID and its description.
recommendations	Integer	The total number of positive ratings for this app.
release_date	Date	The release date.
is_free	Boolean	Whether the app is available for free.

Table 4.1: Attributes of an app, crawled using the *AppCrawler*.

4.2.1.2 ReviewCrawler

After crawling the app list and details, all application types but games were removed in a preprocessing step, leaving 7580 games in the dataset. This was mainly necessary due to this thesis focusing on recommending only video games to the users. Among others, the removed types include unreleased apps, which were not allowed to be rated by the users yet, Downloadable Content (DLC), which always requires the base game and would thus only complicate the recommendation process, and private apps, which were hidden from the Steam store presumably due to them no longer being available for purchase. Details are listed in Table 4.2.

Type	Count	Percent
Game	7580	33.09%
DLC	6421	28.03%
Private App	5430	23.71%
Movie	1765	7.71%
Demo	983	4.29%
Advertising	257	1.12%
Unreleased App	249	1.09%
Video	180	0.79%
Mod	32	0.14%
Hardware	7	0.03%
Total	22 904	100.00%

Table 4.2: Count of application types, retrieved using the *AppCrawler*.

The *ReviewCrawler* was then initialized with the previously filtered IDs of games in order to scrape reviews of each app, the attributes of which are depicted in Table 4.3. The reviews for each game were directly accessed through Steam's game specific web page, as there was no API available for this task.

4.2.1.3 FriendsCrawler

The *FriendsCrawler* was used to get the relationships between users. The crawler was thereby initialized with the list of users retrieved by using the

Attribute	Data type	Description
app_id	Integer	The app's unique identifier.
steam_id	Integer	The user's unique identifier.
date_posted	Date	The date this review was posted.
rating	Integer	The rating given by the user: either positive (2) or negative (1).
playtime_forever	Integer	The number of minutes the user spent playing this game.
content	String	The content of the review.
helpful_ratings	Integer	The number of users who found this review helpful.
total_ratings	Integer	The total number of users who rated this review.
funny_ratings	Integer	The number of users who found this review funny.
content_url	String	The URL pointing to this specific review.

Table 4.3: Attributes of a review, crawled using the *ReviewCrawler*.

ReviewsCrawler. The relationships presented in Table 4.4 were then extracted from Steam's user specific web pages.

Attribute	Data type	Description
steam_id	Integer	The user's unique identifier.
friend_id	Integer	The friend's unique identifier.

Table 4.4: Attributes of a relationship, crawled using the *FriendsCrawler*.

4.2.1.4 GamesOwnedCrawler

To get the games owned by each user, the *GamesOwnedCrawler* was utilized. Similar to the *FriendsCrawler*, this crawler was initialized with the list of users obtained by using the *ReviewCrawler*. The data depicted in Table 4.5 was then retrieved by accessing Steam's user specific web pages.

Attribute	Data type	Description
app_id	Integer	The app's unique identifier.
steam_id	Integer	The user's unique identifier.
playtime_forever	Integer	The number of minutes the user spent playing this game.
playtime_2weeks	Integer	The number of minutes the user spent playing this game within the last two weeks only.
last_played	Integer	The last time the user has played this game.

Table 4.5: Attributes of the games owned by a user, crawled using the *GamesOwnedCrawler*.

4.2.2 Preprocessing

In the data preprocessing step, several actions were executed in order to clean the dataset of unnecessary information. The major part was already done during the crawling process with the help of the Scrapy framework as explained in Section 4.2.1. This includes removing the HTML code surrounding the data as well as extracting the relevant information and transforming it into the desired format. For example, reviews within the Steam store are annotated with either a thumbs-up symbol in combination with the label "Recommended", or a thumbs-down symbol in combination with the label "Not Recommended", which were transformed into 2 or 1 respectively. Further data preprocessing done with Scrapy includes transformation of varying date representations and numbers.

Also already mentioned in Section 4.2.1.2, all applications but games were removed during the data selection process. Last but not least, the relationships between users were cleaned from duplicate entries, as the relationships are bidirectional, and entries where one entity was not present in the user list retrieved by crawling the reviews.

4.2.3 Analysis

To get a feeling for the composition of the dataset, it was analyzed thoroughly before proceeding with the next steps. Table 4.6 shows some basic statistics of the Steam dataset, while Table 4.7 contains structural properties of the network spanned by the relationships between users. It should be mentioned that the diameter and effective diameter of this network are approximations based on 1000 random nodes using the Snap.py tool developed by Leskovec and Sosič (2014). It is noteworthy that the potential implicit data for a recommender algorithm, the number of games owned and hours played by a user, differs from the explicit data, the total ratings, by two and three orders of magnitude respectively. Also, more than 80% of the ratings given by the users are positive which could either be due to user bias or the games being actually mostly good.

	Count
Users	2 206 672
Games	7580
Total ratings	4 976 058
Positive ratings	4 132 048
Negative ratings	844 010
Games owned	199 435 959
Hours played	3 479 153 098

Table 4.6: Basic statistics of the Steam dataset.

Table 4.8 further shows some descriptive statistics for various data within the Steam dataset. Since the data appears to be right-skewed and heavy-tailed, an attempt to fit power law distributions was made using the Python implementation of Alstott, Bullmore, and Plenz (2014). The results are depicted in Figure 4.1 and Figure 4.2 as complementary cumulative distribution functions (CCDF) of the data and show that all of the data is indeed heavy-tailed and that an exponentially truncated power law is a good fit for all data, except for the hours played per user, which is best fit by a log-normal distribution. The curves in Figure 4.1e look heavily distorted and a glance at the corresponding entry in Table 4.8, specifically the maximum value for hours played per user,

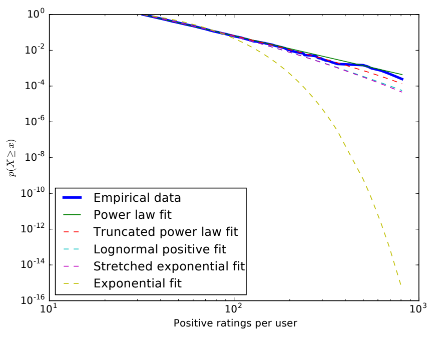
Nodes	2 206 672
Edges	5 185 136
Zero degree nodes	612 086
Nonzero degree nodes	1 594 586
Nodes in largest connected component	1 537 021
Edges in largest connected component	5 151 791
Average clustering coefficient	0.063
Global clustering coefficient	0.0319
Number of triangles	1 033 960
Diameter	18
Effective diameter	7.6648

Table 4.7: Structural properties of the network spanned by the relationships between users within the Steam dataset. The diameter and effective diameter are approximations based on 1000 random nodes.

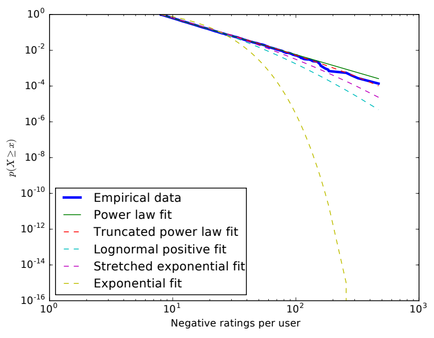
certainly raises some questions. Seen realistically, 664 968 hours or roughly 76 years spent playing, does sound impossible, especially considering the fact that Valve did not begin tracking playtime before early 2009 according to Valve Developer Community (2016). Clearly, this number is caused by outliers in the playtime data, which are subject to data cleaning when using playtime data as input for a recommender algorithm.

	Min	Q1	Median	Q3	Max	Skewness	Kurtosis
Positive ratings per user	0	1.0	1.0	2.0	811	30.2415	3273.1375
Negative ratings per user	0	0.0	0.0	0.0	469	55.0328	9957.5216
Total ratings per user	1	1.0	1.0	2.0	841	29.2907	2582.7896
Games owned per user	0	16.0	50.0	109.0	7027	9.0819	164.3865
Hours played per user	0	267.0	1038.0	2259.0	664 968	31.6006	7706.0332
Friends per user	0	0.0	2.0	6.0	819	6.7118	141.862
Positive ratings per game	0	10.0	41.0	192.0	93 797	16.5713	400.1691
Negative ratings per game	0	3.0	14.0	59.0	24 570	23.0333	814.4457
Total ratings per game	0	16.0	61.0	262.0	97 876	15.5054	344.2644
Ownerships per game	0	934.0	4460.0	18 805.0	1 391 197	7.0134	75.4816
Hours played per game	0	878.0	6560.0	39 383.0	327 204 943	37.4231	1577.4309

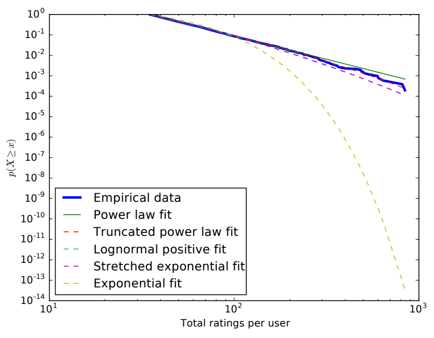
Table 4.8: Descriptive statistics of the Steam dataset.



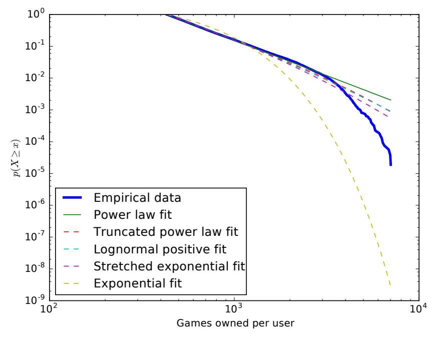
(a) Positive ratings per user



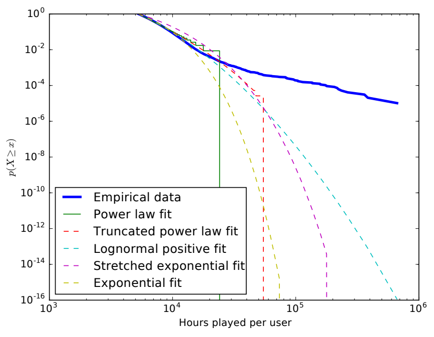
(b) Negative ratings per user



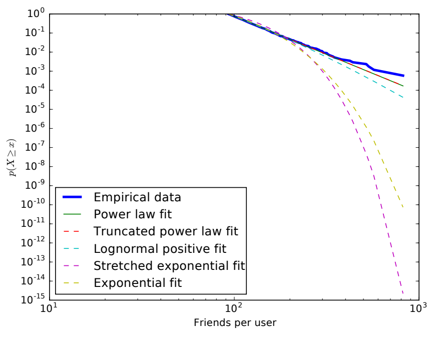
(c) Total ratings per user



(d) Games owned per user

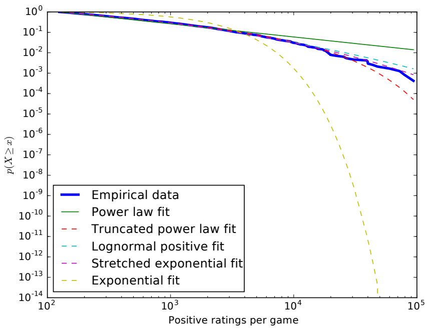


(e) Hours played per user

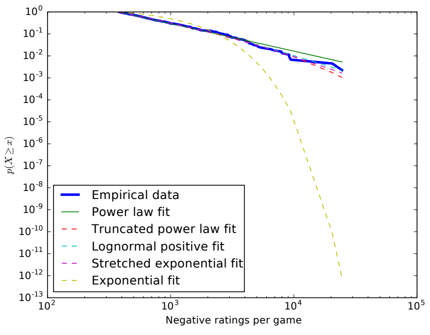


(f) Friends per user

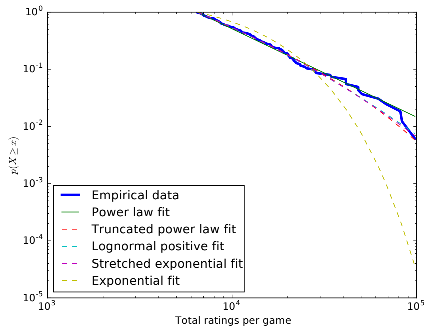
Figure 4.1: Complementary cumulative distribution functions (CCDF) of various user related data within the Steam dataset. An exponentially truncated power law is a good fit for the data in Figures a, b, c, d, and f, while a log-normal distribution considering only positive random variables is the best fit for the data in Figure e.



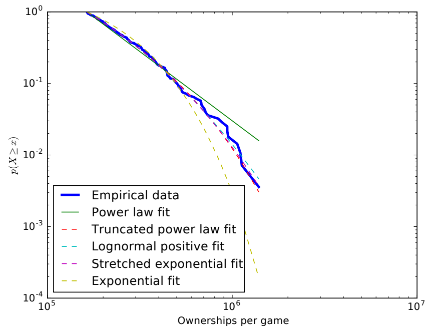
(a) Positive ratings per game



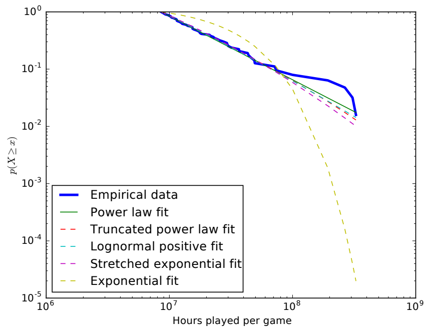
(b) Negative ratings per game



(c) Total ratings per game



(d) Ownerships per game



(e) Hours played per game

Figure 4.2: Complementary cumulative distribution functions (CCDF) of various game related data within the Steam dataset. An exponentially truncated power law is a good fit for the data in figures a, b, c, d, and e.

4.2.4 Transformation

The data transformation described in this section consists of sampling a subset of the dataset retrieved earlier and generating the input files for the recommender algorithms used in Section 4.2.5. The latter is further split into the generation of the vital explicit and implicit rating data files and other input files that are used for specific algorithms only.

4.2.4.1 Sampling

Due to the dataset being very large and the resource-intensive, possibly not optimal, implementations of the algorithms to be tested, sampling is a good strategy to circumvent these issues, while maintaining the original properties of the dataset. (Amatriain and Pujol, 2015) In order to make all the algorithms as comparable as possible, the network spanned by the relationships between users within the Steam dataset was used as a starting point for the sampling process.

Considering the fact that the Steam network has been growing over time, having a sample that looks similar to what the network used to look like at any given time seemed appropriate. Leskovec and Faloutsos (2006) compare various sampling strategies for this back-in-time sampling goal and found the *Forest Fire Model* to perform best. Therefore, the network was sampled using the algorithm described in Leskovec, Kleinberg, and Faloutsos (2005) with a forward burning probability of 20% and a sample size of 1%, resulting in a sample consisting of 22 070 users (nodes) and 37 375 relationships (edges) between them.

4.2.4.2 Explicit Rating Data File

An attempt at answering the first and second research questions requires a dataset containing the explicit ratings of the users sampled in Section 4.2.4.1. The rating data was extracted from the files retrieved by the various crawlers described in Section 4.2.1, which was a straight forward task. Table 4.9 shows the most important statistics of the generated file.

Users	Games	Ratings	1 Stars	2 Stars
22 070	4358	67 321	10 338	56 983

Table 4.9: Statistics of the explicit rating data file used as input for the recommender algorithms.

4.2.4.3 Implicit Rating Data Files

Answering the second and third research question calls for a dataset containing explicit ratings that are derived from implicit data, specifically the time that users spent on playing each game. As a preprocessing step, all game times of zero were treated as unrated and thus removed from the dataset. The remaining game times were then binned on a logarithmic scale with the upper limit being the 99th percentile (476 hours) of all game times for each user and game. This upper limit dealt with the outliers described in Section 4.2.3 and further gave quite realistic values for the boundaries of the logarithmic bins and the number of ratings in each bin.

Moreover, to answer research question three, datasets for binary to quinary rating scales were created by adapting the number of bins. The resulting statistics of the generated files can be seen in Table 4.10. It can be seen that, while using the same number of sampled users as a starting point, the amount of ratings generated using implicit data far surpasses the explicit data while also covering more games. Additionally, there is a substantial difference in the ratio of 1 and 2 stars between the explicit and the implicit dataset.

Max. Rating	Users	Games	Ratings	1 Star	2 Stars	3 Stars	4 Stars	5 Stars
2	20 570	6887	1 553 844	684 628	869 216	0	0	0
3	20 570	6887	1 553 844	224 068	969 739	360 037	0	0
4	20 570	6887	1 553 844	95 751	588 877	669 646	199 570	0
5	20 570	6887	1 553 844	36 194	335 019	643 629	404 526	134 476

Table 4.10: Statistics of the implicit rating data files used as input for the recommender algorithms.

4.2.4.4 Additional Input Files

Several additional input files for usage within specific recommender algorithms were created: an item attribute file, a user relation file, and an implicit data file. The latter two were created in a straight forward manner: the user relation file simply contains the relationships between the sampled users and the implicit data file contains the games owned by those users.

On the other hand, the item attribute file is more complex and consists of a list of pairs, namely an app ID and an assigned attribute ID. Attribute IDs were thus extracted from the app details described in Table 4.1 and the following attributes: “is_free”, publishers, genres and categories. While the “is_free” attribute already was a binary feature, the others had to be transformed first. This means that each publisher, genre and category was given an ID, which in turn was assigned to the games involving them. For example the app ID 70 (Half-Life) was connected to 5 attributes, including attribute ID 2 and 10, which map to the genre “action” and the category “Single-player” respectively.

Furthermore, while these attributes were selected manually using domain knowledge and were deemed appropriate for the purpose of providing an initial input for a few specific recommender algorithms, this attribute selection step could obviously be subject to more sophisticated methods in order to remove attributes with little information gain and add attributes with high information gain. Further sources for good features could be the user assigned tags for each game, and/or the application of Term Frequency-Inverse Document Frequency (TF-IDF) on the game descriptions.

4.2.5 Data Mining

For the purpose of data mining in the recommender system, the MyMediaLite library was chosen because the algorithms implemented cover the most diverse recommendation approaches among the libraries discussed in Section 2.2. Further it provides almost all the required utility, including easy input of rating files, splitting the datasets for cross-validation, calculating evaluation metrics, and even some form of hyperparameter optimization.

As for the recommender algorithms themselves, several diverging algorithms were chosen. Since the Steam network provides hardly any demographic user data, this recommendation approach was out of the question. The following list contains the chosen algorithms and their underlying approach:

- Baselines
  - Random
  - GlobalAverage
  - UserAverage
  - ItemAverage
  - UserItemBaseline
- Content-based
  - ItemAttributeKNN
- Collaborative filtering
  - ItemKNN
  - SlopeOne
  - MatrixFactorization
  - BiasedMatrixFactorization
  - SVDPlusPlus
- Hybrid
  - GSVDPlusPlus

4.2.5.1 Approach

As Amatriain and Pujol (2015) state, the dataset should be split into three subsets: training, validation, and test. The training dataset is thereby used for fitting the model, the validation set for hyperparameter optimization, and the test set for evaluating the accuracy.

First, the algorithm trains its parameters on the data of the training set, while the data in the test set remains unseen. Then, the accuracy is measured by predicting the ratings of the unseen test set and calculating the error to the actual ratings. This ensures that the algorithm is evaluated on how well it generalizes, as testing on the training set would result in biased accuracy.

Salzberg (1997) further explains that everything done to modify or train the algorithm has to be done in advance of seeing the test set, otherwise the results may also be biased. Therefore, hyperparameter tuning is done before measuring the accuracy, again by testing on different unseen data, specifically the validation set.

Since the originally huge dataset was heavily reduced in size using sampling, the data might not be enough for getting an accurate estimate of the algorithms' general performances. Therefore, the above procedure is extended by applying 10-fold cross-validation, where essentially the training and testing process is repeated 10 times using different parts of the dataset each time. The following process is based on the proposal of Salzberg (1997) with a few adaptions and applied on each algorithm and each dataset constructed as described in Section 4.2.4:

1. Split the dataset D into k disjoint and approximately equal sized subsets $D_1, ..., D_k$, where $k = 10$.
2. For $i \in [1, k]$, do the following:
   a) Create a test set $D_i^{test} = \{D_i\}$.
   b) Create a validation set $D_i^{valid} = \{D_{i-1} | D_{-1} \overset{\text{def}}{=} D_k\}$
   c) Create a training set $D_i^{train} = \{D - D_i^{test} - D_i^{valid}\}$
3. Optimize hyperparameters as described in Section 4.2.5.2 by applying cross-validation, where the D_i^{train} subsets are used for training and the D_i^{valid} subsets for testing.
4. Using the best hyperparameters found in the previous step, train on each set D_i^{train} and test on the corresponding set D_i^{test} while calculating the Root Mean Squared Error (RMSE).
5. Store the results of each fold for further evaluation as described in Section 4.2.6.

Figure 4.3 graphically depicts the process described above. It shows that every subset is used exactly once for validation and once for testing, while the others are used for training.

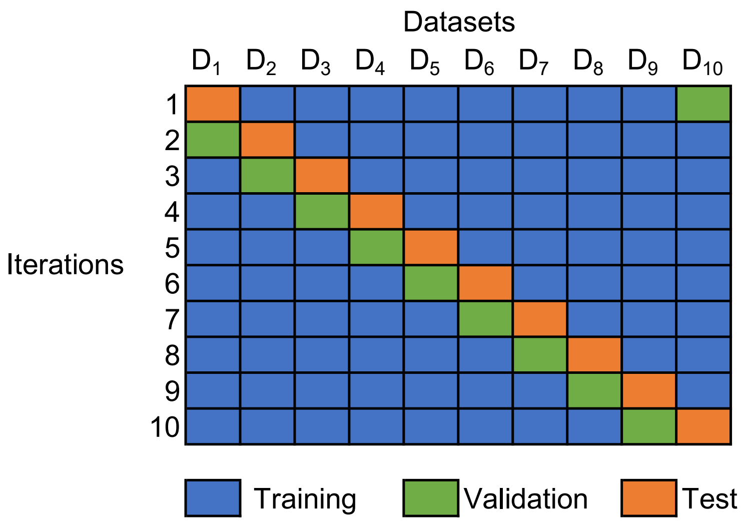


Figure 4.3: 10-Fold Cross-Validation with Training, Validation and Test Subsets. The dataset is split into 10 subsets $D_1, ..., D_10$, whereby in each iteration 8 subsets are used for training, 1 is used for validation, and 1 is used for testing. Hyperparameter optimization is done by cross-validation with the validation sets used for testing. Finally, using the best hyperparameters found, the model is again cross-validated using the test sets for evaluating the RMSE. Image drawn by the author.

4.2.5.2 Hyperparameter Optimization

As already described in Section 2.3, most of the algorithms have hyperparameters that need to be tuned. A hyperparameter is basically a control knob that has to be adjusted in order for the actual learning algorithm to perform well. Y. Bengio (2012) defines a hyper-parameter for a learning algorithm A as

> “a variable to be set prior to the actual application of A to the data, one that is not directly selected by the learning algorithm itself.”

There are many ways to optimize such hyperparameters, including wrapping another learning algorithm around this problem and create a so-called “hyper-learner”, which could potentially require its own hyper-parameters to be adjusted. If the hyper-learner has no hyper-parameters itself, it is considered to be a “pure” learning algorithm. Since this process can take a long time, and is not within the focus of this thesis, a simple combination of manual search and grid search was applied. Grid search simply exhaustively tries all possible combinations for hyper-parameters by applying cross-validation as described in Section 4.2.5.1 in order to determine the ones that perform best. All possible combinations thereby refers to a finite set of values specified by a human.

Therefore, in a first step, a manual search for good hyper-parameters was applied. The found values were then extended by a range of neighboring values, which were then passed on to grid search. Also, for most of the hyper-parameters such as regularization constants, testing values on a log scale given by 10^x, where $x \in [-6, 1]$, was deemed sufficient for the purpose of the experiments. Though, it should be noted that there is most certainly a little more potential contained in this hyper-parameter optimization step.

4.2.6 Interpretation/Evaluation

The approach described in Section 4.2.5.1 contains the calculation of the Root Mean Squared Error (RMSE), which was deemed to be more appropriate than other measures such as the Mean Absolute Error (MAE). Within a recommender system, these measures represent the error between the predicted and the known ratings. The MAE is a linear score and therefore weights all differences equally in the average, while the RMSE punishes large errors more due to it being a quadratic score. This difference was expected to be of importance, especially in the case of various rating scales, which is why the selected learning algorithms were optimized with respect to the RMSE. Further, in order to compare the results of the various rating scales, these error measures need to be normalized first. Gunawardana and Shani (2015) define these measures as depicted in Equation 4.1 and Equation 4.2.

$$\text{RMSE} = \sqrt{\frac{1}{|K|} \sum_{(u,i)\in K} (\hat{r}_{ui} - r_{ui})^2} \qquad \text{NRMSE} = \frac{\text{RMSE}}{r_{max} - r_{min}} \tag{4.1}$$

$$\text{MAE} = \sqrt{\frac{1}{|K|} \sum_{(u,i)\in K} |\hat{r}_{ui} - r_{ui}|} \qquad \text{NMAE} = \frac{\text{MAE}}{r_{max} - r_{min}} \tag{4.2}$$

where r_{ui} is the rating of user u for item i, K the set of (u, i) pairs for which r_{ui} is known, and $\hat{r}_{ui}$ the prediction. NRMSE and NMAE further are the normalized variants of the measures, while r_{max} and r_{min} represent the maximum and minimum rating within the given scale.

While some of the algorithms may obviously perform better than others, as the error measures defined above can show, some may also perform very similarly in terms of accuracy. Drawing reliable conclusions when comparing these algorithms is another challenging task, which is usually done by applying a hypothesis test such as a paired t-test. In fact, there is a lot of discussion in the literature regarding statistical validity of hypothesis tests, also specifically within the field of machine learning.

In order for hypothesis tests to deliver reliable results, a relevant number of measurements is required. Salzberg (1997) warns about repeating cross-validation on the same data set as it would not produce valid statistics due to the trials being highly interdependent. However, Nadeau and Y. Bengio (2003) propose a correction for the variance estimation in such a resampling scenario, specifically multiplying σ^2 with $\frac{1}{n} + \frac{n_2}{n_1}$, where n_1 is the fraction of data used for training and n_2 the fraction of data used for testing. Bouckaert (2003) and Bouckaert and Frank (2004) further propose a 10 times repeated 10-fold cross-validation approach where accuracies calculated on each of the 100 folds are treated as separate measurements and used for estimating the mean and variance. According to the authors, this approach has not only an appropriate Type I error and a low Type II error, but also provides high replicability.

Another challenge arises when comparing multiple algorithms and multiple datasets. This is due to the repetition of t-tests also raising the chance of finding statistical significance. Therefore, these tests have to be corrected, for example by applying the Bonferroni correction. Although Salzberg (1997) mentions that this approach lacks the power of better tests and is overly radical, the application of other methods such as the ANOVA test is left unexplored. Demšar (2006) seizes this thought and analyzes several statistical tests usable for the comparison of two or more classifiers on multiple datasets, concluding that the Friedman test is safer to use than parametric tests such as the ANOVA since it does not assume normal distributions or homogeneity of variance.

Hence, with Research Question 1 in mind, the Friedman test and a subsequent post-hoc test, specifically the Nemenyi test for comparing all classifiers to each other, was deemed appropriate. The RMSE results of all algorithms for all datasets were thus aggregated in order to rank the algorithms tested. For this purpose, the R-package *scmamp* was utilized with results being presented in Chapter 5.

5 Results

This chapter contains the results of the experiments outlined in Chapter 4. The goal of this thesis was the execution of the *Knowledge Discovery Process* (KDP) on recommender systems in the domain of video games, specifically the Steam platform, in order to answer the research questions stated in Chapter 1. The Normalized Root Mean Squared Error (NRMSE) was calculated for each algorithm and dataset, thus allowing for a comparison in terms of performance, and further providing the foundation for answering all research questions. The results for each algorithm are presented separately in the first part of the chapter, which is followed by an overview of all algorithms and the results of the Friedman and Nemenyi tests.

5.1 Algorithms

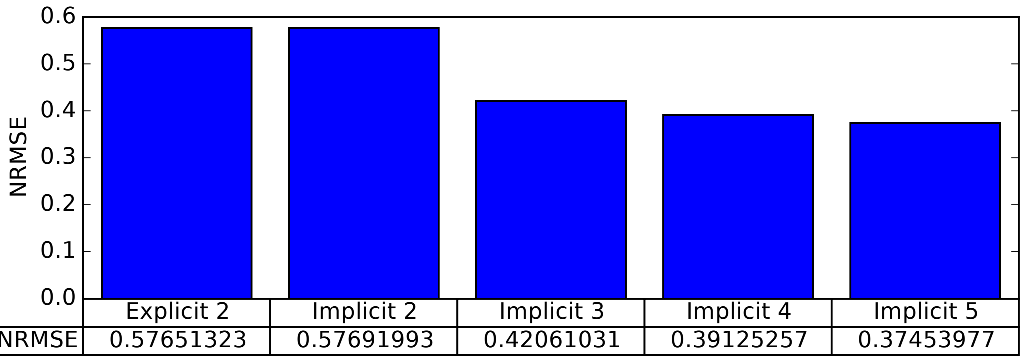
For each algorithm tested, the following information is presented:

- The Normalized Root Mean Squared Error (NRMSE) per dataset in the form of a bar plot, which also includes the exact values in tabular form.
- For algorithms requiring hyperparameters to be tuned: the best hyperparameter settings found per dataset.

5.1.1 Baselines

5.1.1.1 Random

Figure 5.1 shows the NRMSE measurements for the *Random* algorithm of all datasets tested.

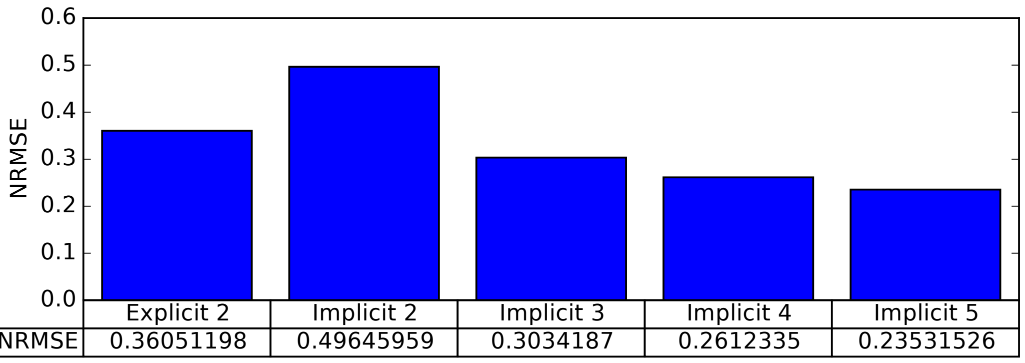


	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.57651323	0.57691993	0.42061031	0.39125257	0.37453977

Figure 5.1: Barplot of the NRMSE measurements for the *Random* algorithm.

5.1.1.2 GlobalAverage

Figure 5.2 shows the NRMSE measurements for the *GlobalAverage* algorithm of all datasets tested.

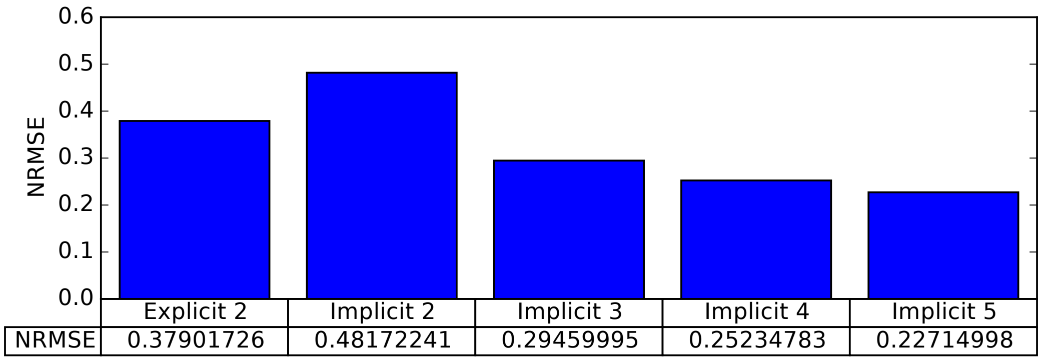


	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.36051198	0.49645959	0.3034187	0.2612335	0.23531526

Figure 5.2: Barplot of the NRMSE measurements for the *GlobalAverage* algorithm.

5.1.1.3 UserAverage

Figure 5.3 shows the NRMSE measurements for the *UserAverage* algorithm of all datasets tested.

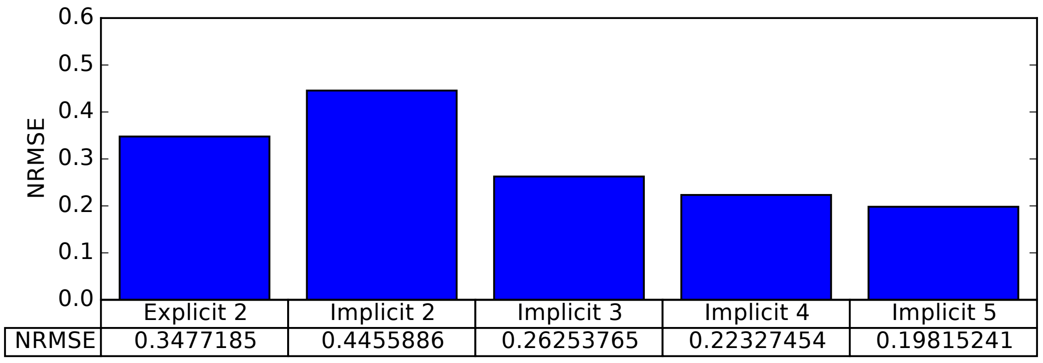


	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.37901726	0.48172241	0.29459995	0.25234783	0.22714998

Figure 5.3: Barplot of the NRMSE measurements for the *UserAverage* algorithm.

5.1.1.4 ItemAverage

Figure 5.4 shows the NRMSE measurements for the *ItemAverage* algorithm of all datasets tested.

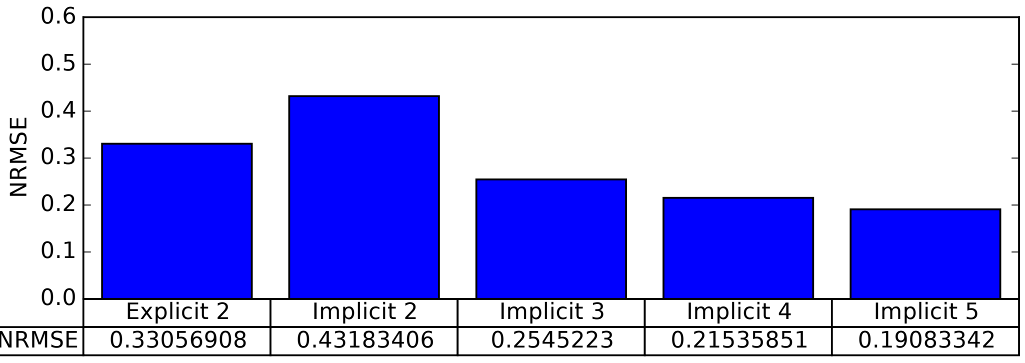


	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.3477185	0.4455886	0.26253765	0.22327454	0.19815241

Figure 5.4: Barplot of the NRMSE measurements for the *ItemAverage* algorithm.

5.1.1.5 UserItemBaseline

Figure 5.5 shows the NRMSE measurements for the *UserItemBaseline* algorithm of all datasets tested. Table 5.1 further shows the best hyperparameters for each dataset.



	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.33056908	0.43183406	0.2545223	0.21535851	0.19083342

Figure 5.5: Barplot of the NRMSE measurements for the *UserItemBaseline* algorithm.

	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
reg_u	7	11	12	11	11
reg_i	4	1	1	1	1
num_iter	15	10	10	10	10

Table 5.1: Best hyperparameters for the *UserItemBaseline* algorithm.

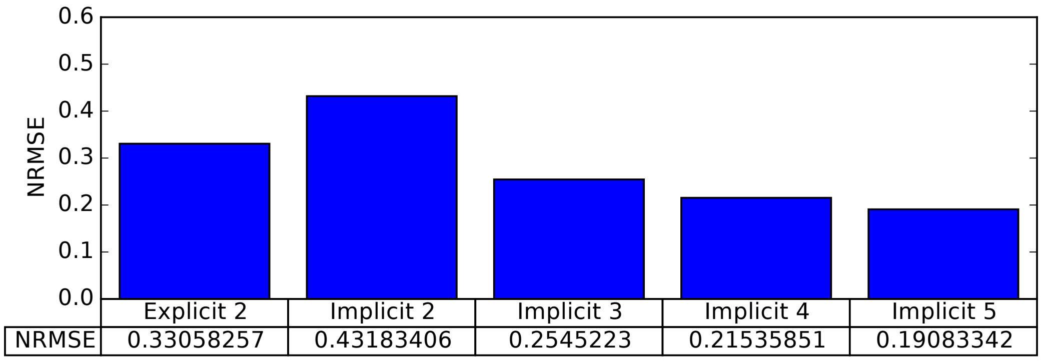
5.1.2 Content-Based

5.1.2.1 ItemAttributeKNN

Figure 5.6 shows the NRMSE measurements for the *ItemAttributeKNN* algorithm of all datasets tested. Table 5.2 further shows the best hyperparameters for each dataset.

	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
k	50	50	50	50	50
correlation	Conditional-Probability	Conditional-Probability	Conditional-Probability	Conditional-Probability	Conditional-Probability
weighted_binary	True	True	True	True	True
alpha	0	0	0	0	0
reg_u	7	11	11	11	11
reg_i	4	1	1	1	1
num_iter	15	10	10	10	10

Table 5.2: Best hyperparameters for the *ItemAttributeKNN* algorithm.



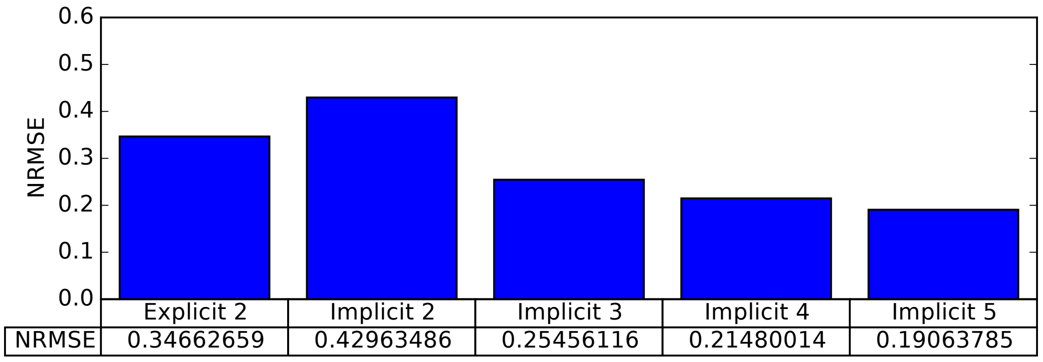
	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.33058257	0.43183406	0.2545223	0.21535851	0.19083342

Figure 5.6: Barplot of the NRMSE measurements for the *ItemAttributeKNN* algorithm.

5.1.3 Collaborative Filtering

5.1.3.1 ItemKNN

Figure 5.7 shows the NRMSE measurements for the *ItemKNN* algorithm of all datasets tested. Table 5.3 further shows the best hyperparameters for each dataset.



	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.34662659	0.42963486	0.25456116	0.21480014	0.19063785

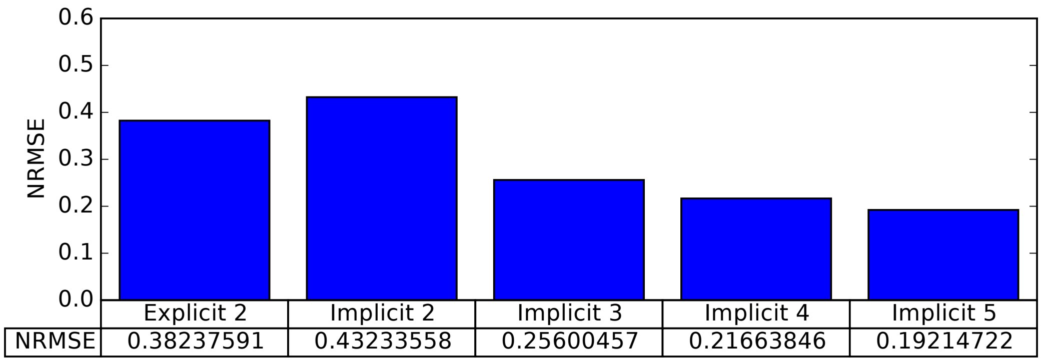
Figure 5.7: Barplot of the NRMSE measurements for the *ItemKNN* algorithm.

	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
k	50	50	50	50	50
correlation	Pearson	RatingCosine	RatingCosine	RatingCosine	RatingCosine
weighted_binary	False	False	False	False	False
alpha	10	1	1	1	1
reg_u	7	11	12	11	11
reg_i	4	1	1	1	1
num_iter	15	10	10	10	10

Table 5.3: Best hyperparameters for the *ItemKNN* algorithm.

5.1.3.2 SlopeOne

Figure 5.8 shows the NRMSE measurements for the *SlopeOne* algorithm of all datasets tested.

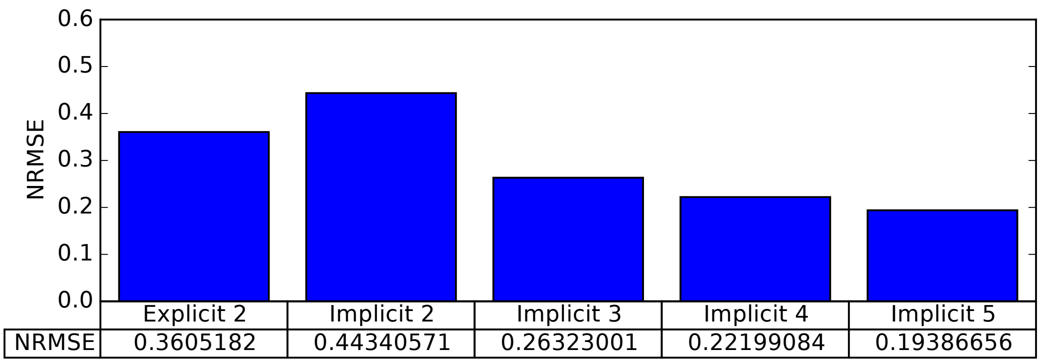


	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.38237591	0.43233558	0.25600457	0.21663846	0.19214722

Figure 5.8: Barplot of the NRMSE measurements for the *SlopeOne* algorithm.

5.1.3.3 MatrixFactorization

Figure 5.9 shows the NRMSE measurements for the *MatrixFactorization* algorithm of all datasets tested. Table 5.4 further shows the best hyperparameters for each dataset.



	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.3605182	0.44340571	0.26323001	0.22199084	0.19386656

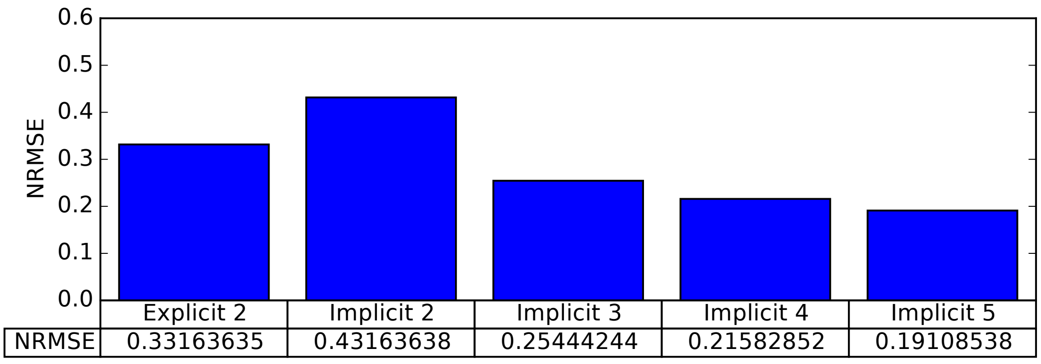
Figure 5.9: Barplot of the NRMSE measurements for the *MatrixFactorization* algorithm.

	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
num_factors	10	50	50	50	10
regularization	2	0.01	0.01	0.01	0.01
learn_rate	0.005	0.01	0.01	0.01	0.01
decay	0.99	0.99	0.99	0.99	0.99
num_iter	30	20	15	10	15

Table 5.4: Best hyperparameters for the *MatrixFactorization* algorithm.

5.1.3.4 BiasedMatrixFactorization

Figure 5.10 shows the NRMSE measurements for the *BiasedMatrixFactorization* algorithm of all datasets tested. Table 5.5 further shows the best hyperparameters for each dataset.



	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.33163635	0.43163638	0.25444244	0.21582852	0.19108538

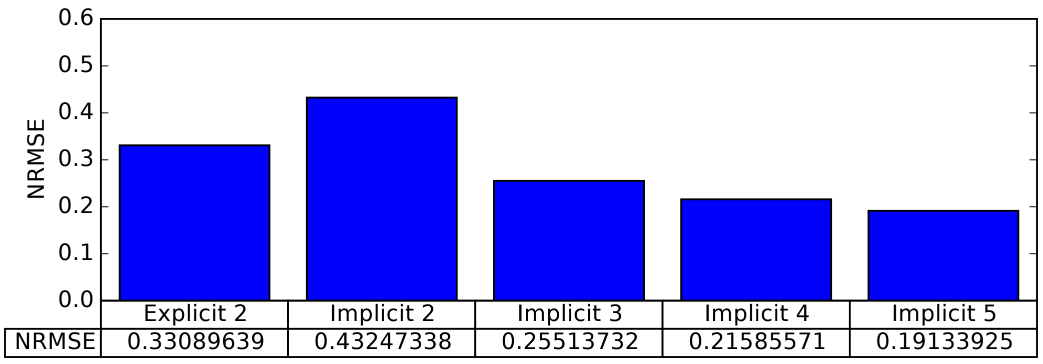
Figure 5.10: Barplot of the NRMSE measurements for the *BiasedMatrixFactorization* algorithm.

	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
num_factors	50	10	10	10	10
bias_reg	0.001	0.001	0.01	0.01	0.1
reg_u	4	3	1	1	1
reg_i	0.1	0.001	1E-05	1E-05	1E-05
frequency_regularization	False	False	False	False	False
learn_rate	0.5	0.5	0.1	0.01	0.01
bias_learn_rate	1	0.25	0.25	1	1
decay	0.99	0.99	0.99	0.99	0.99
num_iter	10	35	20	20	10
bold_driver	True	True	True	True	True
loss	RMSE	RMSE	RMSE	RMSE	RMSE
max_threads	1	1	1	1	1
naive_parallelization	False	False	False	False	False

Table 5.5: Best hyperparameters for the *BiasedMatrixFactorization* algorithm.

5.1.3.5 SVDPlusPlus

Figure 5.11 shows the NRMSE measurements for the *SVDPlusPlus* algorithm of all datasets tested. Table 5.6 further shows the best hyperparameters for each dataset.



	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.33089639	0.43247338	0.25513732	0.21585571	0.19133925

Figure 5.11: Barplot of the NRMSE measurements for the *SVDPlusPlus* algorithm.

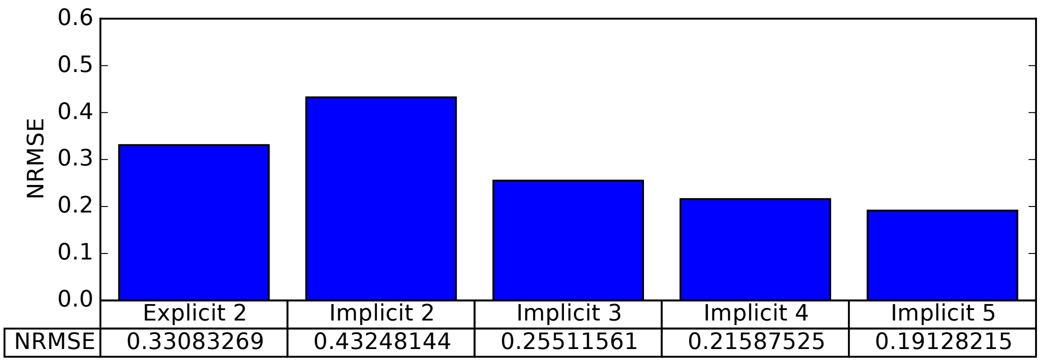
	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
num_factors	10	10	10	10	10
regularization	4	4	7	7	4
bias_reg	0.001	1E-05	1E-05	0.0001	0.0001
frequency_regularization	False	False	False	False	False
learn_rate	0.01	0.01	0.01	0.01	0.01
bias_learn_rate	1	0.5	0.5	0.5	0.5
decay	0.99	0.99	0.99	0.99	0.99
num_iter	15	195	215	25	230

Table 5.6: Best hyperparameters for the *SVDPlusPlus* algorithm.

5.1.4 Hybrid

5.1.4.1 GSVDPlusPlus

Figure 5.12 shows the NRMSE measurements for the *GSVDPlusPlus* algorithm of all datasets tested. Table 5.7 further shows the best hyperparameters for each dataset.



	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
NRMSE	0.33083269	0.43248144	0.25511561	0.21587525	0.19128215

Figure 5.12: Barplot of the NRMSE measurements for the *GSVDPlusPlus* algorithm.

	Dataset				
Hyperparameter	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
num_factors	10	10	10	10	50
regularization	7	4	7	7	5
bias_reg	0.0001	1E-05	1E-05	0.0001	1E-05
frequency_regularization	False	False	False	False	False
learn_rate	0.01	0.01	0.01	0.01	0.01
bias_learn_rate	1	0.5	0.5	0.5	0.5
decay	0.99	0.99	0.99	0.99	0.99
num_iter	15	200	205	25	25

Table 5.7: Best hyperparameters for the *GSVDPlusPlus* algorithm.

5.2 Overview

Table 5.8 shows the NRMSE Measurements for all algorithms, which were also used as input for the Friedman test. The algorithms in the table are sorted by their average ranks based on all datasets tested. Figure 5.13 finally shows the Critical Difference (CD) diagram for the Nemenyi post-hoc test in succession to the Friedman test, which showed a p-value of 1.047e-06. The diagram graphically depicts the average ranks of the algorithms with groups of algorithms being connected, if there is no significant difference in terms of performance.

	Dataset				
	Explicit 2	Implicit 2	Implicit 3	Implicit 4	Implicit 5
UserItemBaseline	0.33056908	0.43183406	0.2545223	0.21535851	0.19083342
ItemKNN	0.34662659	0.42963486	0.25456116	0.21480014	0.19063785
ItemAttributeKNN	0.33058257	0.43183406	0.2545223	0.21535851	0.19083342
BiasedMatrixFactorization	0.33163635	0.43163638	0.25444244	0.21582852	0.19108538
GSVDPlusPlus	0.33083269	0.43248144	0.25511561	0.21587525	0.19128215
SVDPlusPlus	0.33089639	0.43247338	0.25513732	0.21585571	0.19133925
SlopeOne	0.38237591	0.43233558	0.25600457	0.21663846	0.19214722
ItemAverage	0.3477185	0.4455886	0.26253765	0.22327454	0.19815241
MatrixFactorization	0.3605182	0.44340571	0.26323001	0.22199084	0.19386656
UserAverage	0.37901726	0.48172241	0.29459995	0.25234783	0.22714998
GlobalAverage	0.36051198	0.49645959	0.3034187	0.2612335	0.23531526
Random	0.57651323	0.57691993	0.42061031	0.39125257	0.37453977

Table 5.8: NRMSE Measurements for all algorithms. The algorithms are sorted in order of their average ranks based on all datasets tested.

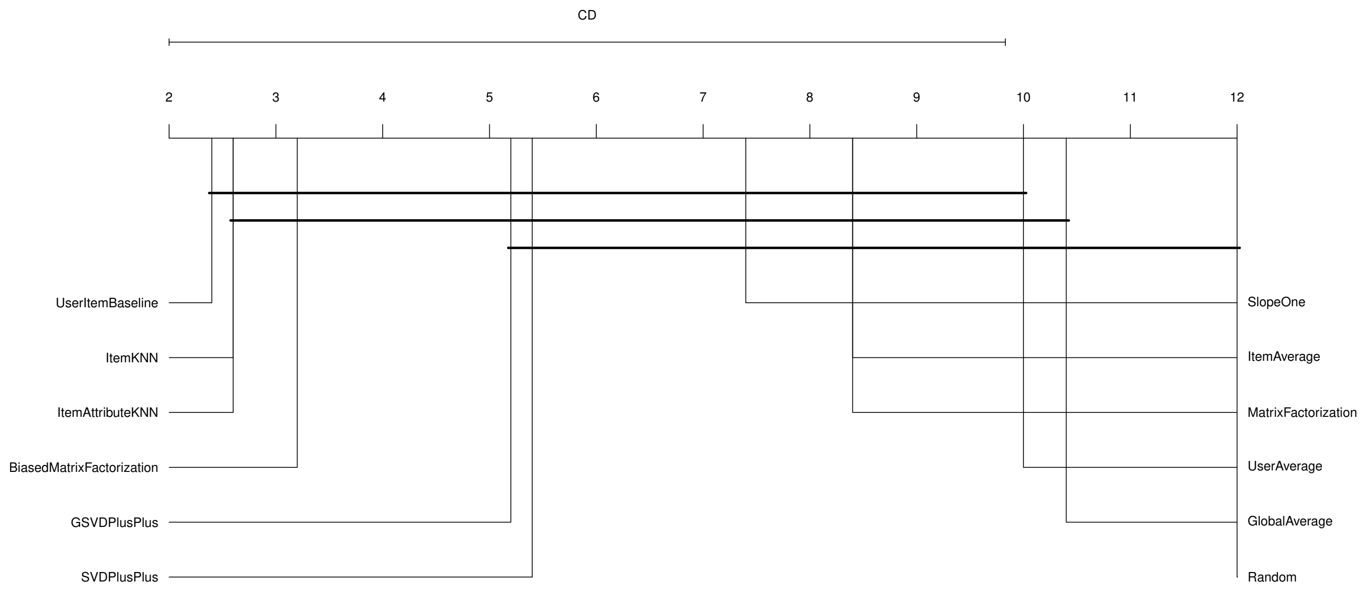


Figure 5.13: Critical Difference (CD) diagram for the Nemenyi post-hoc test in succession to the Friedman test. The critical difference is displayed above the horizontal axis. The numbers in this axis represent the average ranks of the algorithms with 1 being the best and 12 being the worst. If there is no significant difference in terms of performance, the groups of algorithms are connected.

6 Discussion

This chapter contains the discussion of the results presented in Chapter 5, while answering the research questions raised in Chapter 1. All research questions are specifically targeting the domain of video games and specifically the Steam platform.

RQ 1 How well are various recommender algorithms performing in comparison to each other?

The NRMSE results in Section 5.2, especially Figure 5.13, show that the *UserItemBaseline* algorithm is a very strong baseline and overall the best algorithm tested in terms of prediction accuracy. For the *Explicit 2* dataset, no other algorithm was able to beat the baseline. For the implicit datasets, the *ItemKNN* algorithm was superior in most cases with only the *BiasedMatrixFactorization* algorithm beating all others in the case of the *Implicit 3* dataset. Overall, the KNN methods are a close second to the *UserItemBaseline* algorithm, closely followed by the *BiasedMatrixFactorization* algorithm. While the other collaborative filtering methods *SVDPlusPlus* and *GSVDPlusPlus* give a mediocre performance, the basic *MatrixFactorization* algorithm as well as the rest of the baseline algorithms lag behind.

Looking at the good performance of the *ItemAverage* baseline, it can be assumed that the majority of items are rated similarly, which also explains the good performance of the *UserItemBaseline* algorithm. The distributions of the ratings shown in Table 4.9 and Table 4.10, specifically, clearly emphasize this. Interestingly enough, the KNN methods were not able to beat the *UserItemBaseline* in some of the datasets. The otherwise good performance of these methods can be traced back to the *UserItemBaseline* being directly integrated into these methods.

Although the mentioned rankings give a tendency of what algorithms work best in the domain of video games and specifically the Steam platform, the Nemenyi post-hoc test shows no significant difference between the first 10 algorithms. In order to draw final conclusions, more datasets or repetitions of the experiments would be necessary. Further, most of the algorithm's hyperparameters were tuned in a rather simple manner. It is thus quite possible that some of the algorithms might perform better if more sophisticated hyperparameter tuning strategies are applied. Then again, the simplistic *UserItemBaseline* provides good results even without expensive hyperparameter optimization as do the KNN and *BiasedMatrixFactorization* algorithms. During the execution of the experiments, the matrix factorization techniques also showed the expected advantage over the KNN methods in terms of system resources and runtime performance, making the *BiasedMatrixFactorization* algorithm a good overall choice next to the *UserItemBaseline*, whereby the latter is not as heavily personalized as the former.

RQ 2 Can implicit data, specifically game times for each user, be utilized as an adequate replacement for explicit ratings?

The bar plots contained in Section 5.1 show the same behavior for all the algorithms tested. While there is a performance drop between the *Explicit 2* and *Implicit 2* datasets, the rest of the implicit datasets show a drastic performance increase compared to the explicit dataset. RQ2 can thus definitely be answered positively.

It should be noted that the creation of explicit ratings based on game times is subject to tuning as well. The chosen method used logarithmic bins to convert the game times into ratings with the 99th percentile being the upper limit. Shifting this limit might, for example, lead to very different results. Further, as Koenigstein et al. (2012) show, the datasets might also be augmented with randomly generated negative ratings while the positive ratings simply result from the more frequently played games owned by each user. Another thing that could be respected is game and user specific biases amongst the game time. A hardcore player might play games for a longer time compared to a casual player, while both have the same positive or negative experience of these games. Analogously, this might also be the case for the games themselves. One game might be fully experienced in a short amount of time, while others might offer content that can be exciting for hundreds of hours. Despite the gap in game time, players might have the same feeling for both kinds of games.

RQ 3 Considering rating scales from binary to quinary, which rating scale is to be preferred in terms of accuracy?

Again considering the bar plots of Section 5.1, the performance increases in relation to the size of the rating scale for all algorithms tested. While there is a large gap between the performances of the *Implicit 2* and Implicit 3 datasets, the performance increase becomes smaller for the subsequent datasets and thus rating scales. Intuitively, this is reflected by the severity of an error in prediction. For example, if the prediction is off by one star in the case of a binary rating scale, the item is rated with the exact opposite of the true rating. A larger rating scale is more forgiving in such scenarios, as a prediction of 4 stars is still a good recommendation, even if the true rating would be 5 stars.

Other scientists found similar results with completely different settings as outlined in Chapter 3. It is thus expected, that a further extension of the rating scale would not lead to a significant improvement in performance. For the domain of video games and specifically the Steam platform, the recommended size of the rating scale is therefore the classic 5-star rating scale.

7 Conclusion

This thesis dealt with recommender systems in the domain of video games and specifically the Steam platform. Chapter 1 gave an introduction to the field and illustrated the lack of research for recommender systems in the domain of video games and subsequently the Steam platform, for which the existing recommendation approach is unknown to the public. Further, the difference in rating scales between various platforms such as Amazon, Netflix, YouTube, and Steam, as well as the diverse ways for attaining implicit feedback was pointed out. Based on these observations, the following research questions were stated.

RQ 1 How well are various recommender algorithms performing in comparison to each other?

RQ 2 Can implicit data, specifically game times for each user, be utilized as an adequate replacement for explicit ratings?

RQ 3 Considering rating scales from binary to quinary, which rating scale is to be preferred in terms of accuracy?

Chapter 2 gave an introduction to recommender systems consisting of a brief history tracing the roots back to the late 1970s, an explanation of basic terms and concepts, different recommendation techniques, challenges and issues as well as evaluation methods. The chapter further provided an overview of various recommender libraries, with the focus being on open-source software. Next, selected recommendation algorithms offered by the *MyMediaLite* library including baselines, content-based, collaborative filtering and hybrid methods were explained. In order to lay the foundation for the process of answering the research questions, the Knowledge Discovery Process was presented from the viewpoint of a recommender system. Last but not least, a brief presentation of the Steam platform and its existing recommendation approaches was given.

Related research was discussed in Chapter 3, depicting the few works dealing with the domain of video games as well as the analysis of varying rating scales and the effect on users.

The major part of this thesis was presented in Chapter 4, which described the necessary steps for answering the research questions. The chapter started with the presentation of the development environment, continuing with the treatise on the Knowledge Discovery Process. In the selection step, data was retrieved by crawling the Steam website during February 2016 and preprocessed using Python. A thorough analysis of the dataset gave a feeling for its characteristics and showed that most of the data follows a truncated power law. The transformation step of the KDP contained sampling of the data using the *Forest Fire Model* and the creation of the necessary input files for the data mining process. This step then outlined the approach for testing the selected recommendation algorithms, which included 10-fold cross validation with separated validation and test sets for hyperparameter optimization and evaluation respectively. Finally, the interpretation step dealt with the importance of drawing reliable conclusions. The Friedman test and the subsequent Nemenyi post-hoc test was deemed fitting in order to find significant differences in the performance of the tested recommendation algorithms.

Chapter 5 in due course presented the results of the conducted experiments. For each algorithm, a table of the NRMSE measurements obtained for each dataset as well as a corresponding bar plot was shown. Further, for algorithms utilizing hyperparameters, the best ones found were shown as well. The chapter concluded with an overview of the results that were also used as input for the Friedman test, for which the post-hoc Nemenyi test was graphically displayed by utilizing a Critical Difference diagram.

Discussion of the results and answering the research questions was then done in Chapter 6. It was explained that no significant difference between the first 10 algorithms tested could be found, although the results indicated that the *UserItemBaseline* as well as the KNN methods and the *BiasedMatrixFactorization* performed best. After considering the personalization of the recommender algorithms as well as the required system resources and runtime performances, the *BiasedMatrixFactorization* algorithm was deemed as a good overall choice. The bar plots of Section 5.1 further indicated that it is indeed possible to use game times as a replacement for explicit ratings and that a quinary rating scale, the classic 5-star rating scale, is recommended in the domain of video games and specifically the Steam platform.

Further research could deal with optimizing the conversion of implicit feedback into explicit ratings as well as further optimization of the hyperparameters, which is often neglected in many works that deal with recommender systems. Also, the *SocialMF* algorithm contained in the *MyMediaLite* library could be tested by utilizing the already crawled trust network of the Steam platform. Since it builds upon the *BiasedMatrixFactorization* algorithm, the performance in terms of accuracy might thus further improve.

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 Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749. DOI: 10.1109/TKDE.2005.99 (cit. on pp. 6, 7, 10, 20).

Alstott, Jeff, Ed Bullmore, and Dietmar Plenz (2014). “powerlaw: A Python Package for Analysis of Heavy-Tailed Distributions.” In: *PLoS ONE* 9.1, pp. 1–11. DOI: 10.1371/journal.pone.0085777. URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0085777 (cit. on p. 50).

Amatriain, Xavier and Josep M. Pujol (2015). “Data Mining Methods for Recommender Systems.” In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. 2nd ed. Boston, Massachusetts, USA: Springer US. Chap. 7, pp. 227–262. ISBN: 9781489976376. DOI: 10.1007/978-1-4899-7637-6_7 (cit. on pp. 54, 57).

Bell, Robert, Yehuda Koren, and Chris Volinsky (2007). “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems.” In: *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’07)*. (San Jose, California, USA), pp. 95–104. ISBN: 9781595936097. DOI: 10.1145/1281192.1281206 (cit. on p. 19).

Bengio, Yoshua (2012). “Practical Recommendations for Gradient-Based Training of Deep Architectures.” In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. 2nd ed. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg. Chap. 19, pp. 437–478. ISBN: 9783642352898. DOI: 10.1007/978-3-642-35289-8_26 (cit. on p. 60).

Bouckaert, Remco R. (2003). “Choosing Between Two Learning Algorithms Based on Calibrated Tests.” In: *Proc. Twentieth International Conference on Machine Learning (ICML’03).* (Washington, D.C., USA), pp. 51–58. ISBN: 9781577351894. URL: http://www.aaai.org/Papers/ICML/2003/ICML03-010.pdf (cit. on p. 61).

Bouckaert, Remco R. and Eibe Frank (2004). “Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms.” In: *Proc. 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD’04).* (Sydney, Australia), pp. 3–12. ISBN: 9783540247753. DOI: 10.1007/978-3-540-24775-3_3 (cit. on p. 61).

Breese, John S., David Heckerman, and Carl Kadie (1998). “Empirical Analysis of Predictive Algorithms for Collaborative Filtering.” In: *Proc. 14th Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI’98).* (Madison, Wisconsin, USA), pp. 43–52. ISBN: 155860555X. URL: https://dslpitt.org/uai/papers/98/p43-breese.pdf (cit. on p. 20).

Burke, Robin (2002). “Hybrid Recommender Systems: Survey and Experiments.” In: *User Modeling and User-Adapted Interaction* 12.4, pp. 331–370. DOI: 10.1023/A:1021240730564 (cit. on p. 12).

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, Germany: Springer Berlin Heidelberg. Chap. 12, pp. 377–408. ISBN: 9783540720799. DOI: 10.1007/978-3-540-72079-9_12 (cit. on pp. 9, 11–13).

Chiang, Oliver (2011). “The Master of Online Mayhem.” In: *Forbes.* URL: http://www.forbes.com/forbes/2011/0228/technology-gabe-newell-videogames-valve-online-mayhem.html (visited on 07/07/2016) (cit. on pp. 1, 37).

Cios, Krzysztof J., Roman W. Swiniarski, Witold Pedrycz, and Lukasz A. Kurgan (2007). “The Knowledge Discovery Process.” In: *Data Mining: A Knowledge Discovery Approach.* Boston, Massachusetts, USA: Springer US, pp. 9–24. ISBN: 978-0-387-36795-8. DOI: 10.1007/978-0-387-36795-8_2 (cit. on p. 35).

Cosley, Dan, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl (2003). “Is Seeing Believing?: How Recommender System Interfaces Affect Users’ Opinions.” In: *Proc. SIGCHI Conference on Human Factors in*

Computing Systems (CHI'03). (Ft. Lauderdale, Florida, USA), pp. 585–592. ISBN: 1581136307. DOI: 10.1145/642611.642713 (cit. on p. 41).

Davidson, James, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath (2010). "The YouTube Video Recommendation System." In: *Proc. 4th ACM Conference on Recommender Systems (RecSys'10)*. (Barcelona, Spain), pp. 293–296. ISBN: 9781605589060. DOI: 10.1145/1864708.1864770 (cit. on p. 1).

Demšar, Janez (2006). "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *The Journal of Machine Learning Research* 7, pp. 1–30. URL: http://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf (cit. on p. 62).

Deshpande, Mukund and George Karypis (2004). "Item-Based Top-N Recommendation Algorithms." In: *ACM Transactions on Information Systems (TOIS)* 22.1, pp. 143–177. DOI: 10.1145/963770.963776 (cit. on p. 21).

Dunkle, Mike (2015). "Content Delivery Summit 2015. Steam – Past, Current, Future." In: *Content Delivery Summit 2015*. (New York, New York, USA). URL: http://conferences.infotoday.com/documents/234/2015CDNSummit-Keynote-Valve.pdf (visited on 07/08/2016) (cit. on p. 37).

Egenfeldt-Nielsen, Simon, Jonas Heide Smith, and Susana Pajares Tosca (2016). *Understanding Video Games. The Essential Introduction*. 3rd ed. New York, New York, USA: Routledge. ISBN: 9781138849815 (cit. on p. 40).

Ekstrand, Michael D., Michael Ludwig, Joseph A. Konstan, and John T. Riedl (2011). "Rethinking the Recommender Research Ecosystem: Reproducibility, Openness, and LensKit." In: *Proc. 5th ACM Conference on Recommender Systems (RecSys'11)*. (Chicago, Illinois, USA), pp. 133–140. ISBN: 9781450306836. DOI: 10.1145/2043932.2043958 (cit. on p. 21).

Ekstrand, Michael D., John T. Riedl, and Joseph A. Konstan (2011). "Collaborative Filtering Recommender Systems." In: *Foundations and Trends in Human–Computer Interaction* 4.2, pp. 81–173. DOI: 10.1561/1100000009. URL: http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf (cit. on pp. 6, 7, 10, 11).

Ekstrand, Michael, Daniel Kluver, Lingfei He, Jack Kolb, Michael Ludwig, and Yilin He (2016). *LensKit Recommender Toolkit*. URL: http://lenskit.org (visited on 07/14/2016) (cit. on p. 21).

Entertainment Software Association (ESA) (2016). *The 2016 Essential Facts About the Computer and Video Game Industry.* URL: http://essentialfacts.theesa.com/Essential-Facts-2016.pdf (visited on 07/27/2016) (cit. on p. 40).

Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996a). “From Data Mining to Knowledge Discovery in Databases.” In: *AI Magazine* 17.3, pp. 37–54. DOI: 10.1609/aimag.v17i3.1230 (cit. on p. 35).

Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996b). “Knowledge Discovery and Data Mining: Towards a Unifying Framework.” In: *Proc. Second International Conference on Knowledge Discovery and Data Mining (KDD’96).* (Portland, Oregon, USA), pp. 82–88. ISBN: 9781577350040. URL: https://www.aaai.org/Papers/KDD/1996/KDD96-014.pdf (cit. on pp. 34, 35, 44).

Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996c). “The KDD Process for Extracting Useful Knowledge from Volumes of Data.” In: *Communications of the ACM* 39.11, pp. 27–34. DOI: 10.1145/240455.240464 (cit. on p. 35).

Funk, Simon (2006). *The Evolution of Cybernetics - A Journal. Netflix Update: Try This at Home.* URL: http://sifter.org/~simon/journal/20061211.html (visited on 07/14/2016) (cit. on pp. 21, 27).

Gantner, Zeno, Steffen Rendle, Lucas Drumond, and Christoph Freudenthaler (2015). *MyMediaLite Recommender System Library.* URL: http://www.mymedialite.net (visited on 07/14/2016) (cit. on p. 19).

Gantner, Zeno, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme (2011). “MyMediaLite: A Free Recommender System Library.” In: *Proc. 5th ACM Conference on Recommender Systems (RecSys’11).* (Chicago, Illinois, USA), pp. 305–308. ISBN: 9781450306836. DOI: 10.1145/2043932.2043989. URL: http://www.ismll.uni-hildesheim.de/pub/pdfs/Gantner_et_al2011_MyMediaLite.pdf (cit. on pp. 18, 29, 31).

Gemulla, Rainer, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis (2011). “Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent.” In: *Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’11).* (San Diego, California, USA), pp. 69–77. ISBN: 9781450308137. DOI: 10.1145/2020408.2020426 (cit. on p. 19).

George, Thomas and Srujana Merugu (2005). “A Scalable Collaborative Filtering Framework based on Co-clustering.” In: *Proc. 5th IEEE International*

Conference on Data Mining (ICDM'05). (Houston, Texas, USA), pp. 625–628. ISBN: 0769522785. DOI: 10.1109/ICDM.2005.14 (cit. on p. 19).

Geyer-Schulz, Andreas and Michael Hahsler (2002). "Comparing Two Recommender Algorithms with the Help of Recommendations by Peers." In: *WEBKDD 2002 – Mining Web Data for Discovering Usage Patterns and Profiles. 4th International Workshop.* (Edmonton, Canada). Berlin, Heidelberg, Germany, pp. 137–158. ISBN: 9783540203049. DOI: 10.1007/978-3-540-39663-5_9 (cit. on p. 35).

Goldberg, David, David Nichols, Brian M. Oki, and Douglas Terry (1992). "Using Collaborative Filtering to Weave an Information Tapestry." In: *Communications of the ACM* 35.12, pp. 61–70. DOI: 10.1145/138859.138867 (cit. on pp. 1, 5).

Gomez-Uribe, Carlos A. and Neil Hunt (2016). "The Netflix Recommender System: Algorithms, Business Value, and Innovation." In: *ACM Transactions on Management Information Systems (TMIS)* 6.4, 13:1–13:19. DOI: 10.1145/2843948 (cit. on pp. 1, 6).

Gunawardana, Asela and Guy Shani (2015). "Evaluating Recommendation Systems." In: *Recommender Systems Handbook.* Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. 2nd ed. Boston, Massachusetts, USA: Springer US. Chap. 8, pp. 257–297. ISBN: 9780387858203. DOI: 10.1007/978-0-387-85820-3_8 (cit. on pp. 14, 61).

Hill, Will, Larry Stead, Mark Rosenstein, and George Furnas (1995). "Recommending and Evaluating Choices in a Virtual Community of Use." In: *Proc. 1995 Conference on Human Factors in Computing Systems (CHI'95).* (Denver, Colorado, USA), pp. 194–201. ISBN: 0201847051. DOI: 10.1145/223904.223929 (cit. on p. 6).

Hu, Yifan, Yehuda Koren, and Chris Volinsky (2008). "Collaborative Filtering for Implicit Feedback Datasets." In: *Proc. 8th IEEE International Conference on Data Mining (ICDM'08).* (Pisa, Italy), pp. 263–272. ISBN: 9780769535029. DOI: 10.1109/ICDM.2008.22 (cit. on p. 22).

Jamali, Mohsen and Martin Ester (2010). "A Matrix Factorization Technique with Trust Propagation for Recommendation in Social Networks." In: *Proc. 4th ACM Conference on Recommender Systems (RecSys'10).* (Barcelona, Spain), pp. 135–142. ISBN: 9781605589060. DOI: 10.1145/1864708.1864736 (cit. on p. 20).

Jannach, Dietmar, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich (2011). *Recommender Systems. An Introduction.* New York, New York, USA: Cambridge University Press. ISBN: 9780521493369 (cit. on pp. 4, 5).

Karypis, George (2001). "Evaluation of Item-Based Top-N Recommendation Algorithms." In: *Proc. 10th International Conference on Information and Knowledge Management (CIKM'01).* (Atlanta, Georgia, USA), pp. 247–254. ISBN: 1581134363. DOI: 10.1145/502585.502627 (cit. on p. 6).

Kluver, Daniel, Tien T. Nguyen, Michael Ekstrand, Shilad Sen, and John Riedl (2012). "How Many Bits Per Rating?" In: *Proc. 6th ACM Conference on Recommender Systems (RecSys'12).* Dublin, Ireland, pp. 99–106. ISBN: 9781450312707. DOI: 10.1145/2365952.2365974 (cit. on p. 41).

Koenigstein, Noam, Nir Nice, Ulrich Paquet, and Nir Schleyen (2012). "The Xbox Recommender System." In: *Proc. 6th ACM Conference on Recommender Systems (RecSys'12).* (Dublin, Ireland), pp. 281–284. ISBN: 9781450312707. DOI: 10.1145/2365952.2366015 (cit. on pp. 40, 76).

Koren, Yehuda (2008). "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model." In: *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08).* (Las Vegas, Nevada, USA), pp. 426–434. ISBN: 9781605581934. DOI: 10.1145/1401890.1401944 (cit. on pp. 19, 22, 30).

Koren, Yehuda (2009). *The BellKor Solution to the Netflix Grand Prize.* URL: http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf (visited on 07/14/2016) (cit. on p. 19).

Koren, Yehuda (2010). "Factor in the Neighbors: Scalable and Accurate Collaborative Filtering." In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4.1, 1:1–1:24. DOI: 10.1145/1644873.1644874 (cit. on pp. 19, 24, 25).

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems." In: *Computer* 42.8, pp. 30–37. DOI: 10.1109/MC.2009.263 (cit. on pp. 14, 27).

Krulwich, Bruce (1997). "Lifestyle Finder. Intelligent User Profiling Using Large-Scale Demographic Data." In: *AI Magazine* 18.2, pp. 37–46. DOI: 10.1609/aimag.v18i2.1292 (cit. on p. 11).

Lamprecht, Daniel, Florian Geigl, Tomas Karas, Simon Walk, Denis Helic, and Markus Strohmaier (2015). "Improving Recommender System Navigability Through Diversification: A Case Study of IMDb." In: *Proc. 15th International Conference on Knowledge Technologies and Data-driven Busi-*

ness (i-KNOW'15). (Graz, Austria), 21:1–21:8. ISBN: 9781450337212. DOI: 10.1145/2809563.2809603 (cit. on p. 6).

Lawrence, Neil D. and Raquel Urtasun (2009). "Non-linear Matrix Factorization with Gaussian Processes." In: *Proc. 26th International Conference on Machine Learning (ICML'09).* (Montreal, Quebec, Canada), pp. 601–608. ISBN: 9781605585161. DOI: 10.1145/1553374.1553452 (cit. on p. 20).

Lee, Daniel D. and H. Sebastian Seung (2000). "Algorithms for Non-negative Matrix Factorization." In: *Proc. 14th Conference on Neural Information Processing Systems (NIPS'00).* (Denver, Colorado, USA), pp. 556–562. URL: http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf (cit. on p. 20).

Lee, Joonseok, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer (2014). "Local Collaborative Ranking." In: *Proc. 23rd International Conference on World Wide Web (WWW'14).* (Seoul, Korea), pp. 85–96. ISBN: 9781450327442. DOI: 10.1145/2566486.2567970 (cit. on p. 20).

Lee, Joonseok, Seungyeon Kim, Guy Lebanon, and Yoram Singer (2013). "Local Low-Rank Matrix Approximation." In: *Proc. 30th International Conference on Machine Learning (ICML'14).* (Atlanta, Georgia, USA), pp. 82–90. URL: http://jmlr.org/proceedings/papers/v28/lee13.pdf (cit. on p. 20).

Lee, Joonseok, Mingxuan Sun, and Guy Lebanon (2012a). "A Comparative Study of Collaborative Filtering Algorithms." In: arXiv: 1205.3193 (cit. on p. 20).

Lee, Joonseok, Mingxuan Sun, and Guy Lebanon (2012b). "PREA: Personalized Recommendation Algorithms Toolkit." In: *The Journal of Machine Learning Research* 13, pp. 2699–2703. URL: http://www.jmlr.org/papers/volume13/lee12b/lee12b.pdf (cit. on p. 20).

Lee, Joonseok, Mingxuan Sun, and Guy Lebanon (2014). *PREA (Personalized Recommendation Algorithms Toolkit).* URL: http://prea.gatech.edu/ (visited on 07/14/2016) (cit. on p. 20).

Lemire, Daniel and Anna Maclachlan (2005). "Slope One Predictors for Online Rating-Based Collaborative Filtering." In: *Proc. 2005 SIAM International Conference on Data Mining (SDM'05).* (Newport Beach, California, USA), pp. 471–475. ISBN: 9780898715934. DOI: 10.1137/1.9781611972757.43. URL: http://lemire.me/fr/documents/publications/lemiremaclachlan_sdm05.pdf (cit. on pp. 19–21, 26).

Leskovec, Jure and Christos Faloutsos (2006). "Sampling from Large Graphs." In: *Proc. 12th ACM SIGKDD International Conference on Knowledge*

Discovery and Data Mining (KDD'06). (Philadelphia, Pennsylvania, USA), pp. 631–636. ISBN: 1595933395. DOI: 10.1145/1150402.1150479. URL: http://cs.stanford.edu/people/jure/pubs/sampling-kdd06.pdf (cit. on p. 54).

Leskovec, Jure, Jon Kleinberg, and Christos Faloutsos (2005). "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations." In: *Proc. 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05).* (Chicago, Illinois, USA), pp. 177–187. ISBN: 159593135X. DOI: 10.1145/1081870.1081893. URL: https://www.cs.cornell.edu/home/kleinber/kdd05-time.pdf (cit. on p. 54).

Leskovec, Jure and Rok Sosič (2014). *Snap.py: SNAP for Python, a General Purpose Network Analysis and Graph Mining Tool in Python.* URL: http://snap.stanford.edu/snappy (visited on 05/25/2016) (cit. on p. 50).

Linden, Greg, Brent Smith, and Jeremy York (2003). "Amazon.com Recommendations. Item-to-Item Collaborative Filtering." In: *IEEE Internet Computing* 7.1, pp. 76–80. DOI: 10.1109/MIC.2003.1167344 (cit. on pp. 1, 6, 19).

Manzato, Marcelo Garcia (2013). "gSVD++: Supporting Implicit Feedback on Recommender Systems with Metadata Awareness." In: *Proc. 28th Annual ACM Symposium on Applied Computing (SAC'13).* (Coimbra, Portugal), pp. 908–913. ISBN: 9781450316569. DOI: 10.1145/2480362.2480536 (cit. on pp. 19, 32).

Menon, Aditya Krishna and Charles Elkan (2010). "A Log-Linear Model with Latent Features for Dyadic Prediction." In: *Proc. 10th IEEE International Conference on Data Mining (ICDM'10).* (Sydney, Australia), pp. 364–373. ISBN: 9780769542560. DOI: 10.1109/ICDM.2010.148 (cit. on p. 19).

Mitchell, Ryan (2015). *Web Scraping with Python.* Sebastopol, California, USA: O'Reilly Media, Inc. ISBN: 9781491910290 (cit. on p. 45).

Nadeau, Claude and Yoshua Bengio (2003). "Inference for the Generalization Error." In: *Machine Learning* 52.3, pp. 239–281. DOI: 10.1023/A:1024068626366 (cit. on p. 61).

Netflix, Inc (2009). *Netflix Prize.* URL: http://www.netflixprize.com (visited on 07/09/2016) (cit. on p. 6).

Nguyen, Tien T., Pik-Mai Hui, F. Maxwell Harper, Loren Terveen, and Joseph A. Konstan (2014). "Exploring the Filter Bubble: The Effect of Using Recommender Systems on Content Diversity." In: *Proc. 23rd International*

Conference on World Wide Web (WWW'14). (Seoul, Korea), pp. 677–686. ISBN: 9781450327442. DOI: 10.1145/2566486.2568012 (cit. on pp. 6, 9).

Paterek, Arkadiusz (2007). "Improving Regularized Singular Value Decomposition for Collaborative Filtering." In: *Proc. KDD Cup and Workshop 2007.* (San Jose, California, USA). URL: https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/Regular-Paterek.pdf (cit. on p. 20).

Rajaraman, Shiva (2009). *Five Stars Dominate Ratings.* URL: https://youtube.googleblog.com/2009/09/five-stars-dominate-ratings.html (visited on 07/07/2016) (cit. on pp. 2, 6).

Recht, Benjamin, Christopher Re, Stephen Wright, and Feng Niu (2011). "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent." In: *Proc. 25th Conference on Neural Information Processing Systems (NIPS'11).* (Granada, Spain), pp. 693–701. URL: http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf (cit. on p. 22).

Rendle, Steffen and Lars Schmidt-Thieme (2008). "Online-updating Regularized Kernel Matrix Factorization Models for Large-scale Recommender Systems." In: *Proc. 2008 ACM Conference on Recommender Systems (RecSys'08).* (Lausanne, Switzerland), pp. 251–258. ISBN: 9781605580937. DOI: 10.1145/1454008.1454047 (cit. on p. 19).

Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl (1994). "GroupLens: An Open Architecture for Collaborative Filtering of Netnews." In: *Proc. 1994 ACM Conference on Computer Supported Cooperative Work (CSCW'94).* (Chapel Hill, North Carolina, USA), pp. 175–186. ISBN: 0897916891. DOI: 10.1145/192844.192905 (cit. on pp. 6, 21).

Resnick, Paul and Hal R. Varian (1997). "Recommender Systems." In: *Communications of the ACM* 40.3, pp. 56–58. DOI: 10.1145/245108.245121 (cit. on pp. 1, 5).

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2015a). *Recommender Systems Handbook.* 2nd ed. Boston, Massachusetts, USA: Springer US. ISBN: 9781489976376. DOI: 10.1007/978-1-4899-7637-6 (cit. on pp. 1, 8, 12).

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2015b). "Recommender Systems: Introduction and Challenges." In: *Recommender Systems Handbook.* Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. 2nd ed. Boston, Massachusetts, USA: Springer US. Chap. 1, pp. 1–34. ISBN: 9781489976376. DOI: 10.1007/978-1-4899-7637-6_1 (cit. on pp. 9, 11, 12).

Rich, Elaine (1979). "User Modeling via Stereotypes." In: *Cognitive Science* 3.4, pp. 329–354. DOI: 10.1207/s15516709cog0304_3. URL: http://dx.doi.org/10.1207/s15516709cog0304_3 (cit. on p. 5).

Salakhutdinov, Ruslan and Andriy Mnih (2007). "Probabilistic Matrix Factorization." In: *Proc. 21st Conference on Neural Information Processing Systems (NIPS'07)*. (Vancouver, British Columbia, Canada), pp. 1257–1264. URL: http://papers.nips.cc/paper/3208-probabilistic-matrix-factorization.pdf (cit. on pp. 19, 20, 28, 29).

Salakhutdinov, Ruslan and Andriy Mnih (2008). "Bayesian Probabilistic Matrix Factorization Using Markov Chain Monte Carlo." In: *Proc. 25th International Conference on Machine Learning (ICML'08)*. (Helsinki, Finland), pp. 880–887. ISBN: 9781605582054. DOI: 10.1145/1390156.1390267 (cit. on p. 20).

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. DOI: 10.1023/A:1009752403260 (cit. on pp. 58, 61, 62).

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl (2001). "Item-based Collaborative Filtering Recommendation Algorithms." In: *Proc. 10th International Conference on World Wide Web (WWW'01)*. (Hong Kong, Hong Kong), pp. 285–295. ISBN: 1581133480. DOI: 10.1145/371920.372071 (cit. on pp. 6, 20, 21).

Schreier, Jason (2014). "Steam Is Getting A Massive Overhaul." In: *Kotaku*. URL: http://kotaku.com/steam-is-getting-a-massive-overhaul-1637818112 (visited on 07/08/2016) (cit. on p. 39).

Shardanand, Upendra and Pattie Maes (1995). "Social Information Filtering: Algorithms for Automating &Ldquo;Word of Mouth&Rdquo." In: *Proc. 1995 Conference on Human Factors in Computing Systems (CHI'95)*. (Denver, Colorado, USA), pp. 210–217. ISBN: 0201847051. DOI: 10.1145/223904.223931 (cit. on p. 6).

Sifa, Rafet, Christian Bauckhage, and Anders Drachen (2014). "Archetypal Game Recommender Systems." In: *Proc. 16th LWA Workshops: KDML, IR and FGWM (LWA'14)*. (Aachen, Germany), pp. 45–56. URL: http://ceur-ws.org/Vol-1226/paper10.pdf (cit. on p. 40).

Sparling, E. Isaac and Shilad Sen (2011). "Rating: How Difficult is It?" In: *Proc. 5th ACM Conference on Recommender Systems (RecSys'11)*. (Chicago, Illinois, USA), pp. 149–156. ISBN: 9781450306836. DOI: 10.1145/2043932.2043961 (cit. on p. 41).

Su, Xiaoyuan and Taghi M. Khoshgoftaar (2009). “A Survey of Collaborative Filtering Techniques.” In: *Advances in Artificial Intelligence* 2009. DOI: 10.1155/2009/421425 (cit. on p. 20).

Sun, Mingxuan, Guy Lebanon, and Paul Kidwell (2011). “Fast Nonparametric Matrix Factorization for Large-scale Collaborative Filtering.” In: *Proc. 14th Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS’11).* (Ft. Lauderdale, Florida, USA). URL: http://jmlr.csail.mit.edu/proceedings/papers/v15/sun11a/sun11a.pdf (cit. on p. 21).

The Apache Software Foundation (2016). *Apache Mahout: Scalable Machine Learning and Data Mining.* URL: http://mahout.apache.org (visited on 07/14/2016) (cit. on p. 22).

Valve Corporation (2010). *Steam Web API Terms of Use.* URL: http://steamcommunity.com/dev/apiterms (visited on 05/25/2016) (cit. on p. 45).

Valve Corporation (2016a). *Steam Discovery Update. A Smarter Storefront, Personalized Just for You.* URL: http://store.steampowered.com/about/newstore (visited on 07/08/2016) (cit. on p. 39).

Valve Corporation (2016b). *Valve.* URL: http://www.valvesoftware.com/company (visited on 05/25/2016) (cit. on p. 37).

Valve Developer Community (2016). *Steam Web API.* URL: https://developer.valvesoftware.com/w/index.php?title=Steam_Web_API&oldid=195954 (visited on 05/25/2016) (cit. on pp. 45, 51).

Yu, Kai, Shenghuo Zhu, John Lafferty, and Yihong Gong (2009). “Fast Nonparametric Matrix Factorization for Large-scale Collaborative Filtering.” In: *Proc. 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’09).* (Boston, Massachusetts, USA), pp. 211–218. ISBN: 9781605584836. DOI: 10.1145/1571941.1571979 (cit. on p. 21).

Zhou, Yunhong, Dennis Wilkinson, Robert Schreiber, and Rong Pan (2008). “Large-Scale Parallel Collaborative Filtering for the Netflix Prize.” In: *Proc. 4th International Conference on Algorithmic Aspects in Information and Management (AAIM’08).* (Shanghai, China), pp. 337–348. ISBN: 9783540688655. DOI: 10.1007/978-3-540-68880-8_32 (cit. on p. 22).