University of Bergen
Department of Informatics

Master thesis

Scaling the scales

A suggested improvement to
IBM's Intelligent Recommendation Algorithm

Author
Magnar Myrtveit

Supervisor
Jan Arne Telle

November 2014

Abstract

Recommender systems appear in a large variety of applications, and their use has become very common in recent years. As a lot of money can be made by companies having a better recommender system than their competitors, much of the research behind the best recommendation algorithms is proprietary and has not been published. We suggest an improvement to a graph-based collaborative filtering recommendation algorithm developed at IBM Research and published in “KDD ’99 Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery data mining” in 1999: the Intelligent Recommendation Algorithm (IRA) [1].

We start by giving an overview of the field of recommender systems, how they work, and how they can be utilized. Next we give a detailed description of the graph-theoretical ideas IRA is built upon, and take an in-depth look at how the algorithm works. We then present a suggested improvement to IRA, called Scaling the scales. In Scaling the scales, customers using differently sized ranges of the rating scale can still be used to predict each other. We give a short overview of the design behind the implementation of our recommender system, which is available at GitHub [23], as well as BORA [3], where this thesis is published. The implementation uses a modular software design to allow for developing extensions to the recommendation algorithms we have implemented, as well as deploying other recommendation algorithms in our system.

Next we compare the recommendation quality of IRA and Scaling the scales using leave-one-out cross-validation. This method works by hiding a known rating, predicting what this rating should be, and comparing the correct and predicted ratings. Experiments on four different datasets were run. Our results indicate that Scaling the scales give slightly smaller errors than IRA when predicting ratings. A bigger improvement is that Scaling the scales calculates predictions in many cases where IRA is unsuccessful.

Lastly, ideas for further improvement of Scaling the scales are presented.

Acknowledgments

First and foremost, I would like to thank my supervisor, Jan Arne, for all help and support. I would not have been able to write this thesis without his feedback and expertise.

I would also like to thank the Algorithms group as a whole, for providing an environment that is both social and educational. The table tennis matches, Friday lunches, and trips to Winter School in Algorithms have been an important part of my time as a master student in the Algorithms group.

A special thank you goes to Martin for getting me interested in the field of algorithms.

I am honored to have been competing alongside Simen and Johan in various programming competitions. In addition to being educational, it has always been a great pleasure, and I believe the programming competitions will be among my best memories from my time at the university.

I would like to thank my mom and dad, Astrid and Magne, for providing me with a playful and good learning environment when i was young. And a special thank you goes to my dad for introducing me to programming when I was still in primary school.

Lastly, I would like to thank my sister, Solbjørg, both for providing valuable feedback on this thesis, but even more so for taking care of me and making sure I didn't starve to death during the last month of writing this thesis.

Bergen, November 2014
Magnar Myrtveit

Contents

Chapter 1

Introduction to recommender systems

When you want to read a book, but don't know which one to choose, you may ask a knowledgeable friend for a recommendation. The better your friend knows your taste in books, the more likely he or she will be to recommend a book that you actually like. Recommender systems are based upon this idea. By building an understanding of what a user likes and dislikes, the recommender system can identify items or content that the user is likely to be interested in.

The two main purposes of recommender systems are generating sales (or other desirable user actions) and dealing with information overload on part of the user, as described in *Introduction to recommender systems handbook* [28]. If it takes a person looking for a new camera too long to identify a camera fitting his or her needs in a large web shop, this potential customer might move on to try another web shop. A good recommender system can increase sales by quickly identifying items of high interest to the user. Also, a recommender system can generate new sales by introducing customers to products they did not even know they were interested in, for example by recommending items that other customers with similar interests have bought. By dealing with the information overload that can otherwise hit the user, the recommender system can increase customer satisfaction. In a large online newspaper, every reader will probably not have interest in all the articles written. If a reader finds many of the articles uninteresting, he or she might start considering the newspaper unsatisfactory. By knowing what the reader usually finds interesting, a recommender system can discriminate between relevant and irrelevant articles, so that the articles likely to be of high interest are the ones presented most prominently to the reader.

Recommender systems can provide clear benefits to both users and content providers, and the business value of a good recommender system is supported by both research (see the article by Garfinkel et al. [12] for one example among many) and common practice. Because the idea behind recommender systems is generic, recommender systems can be deployed and utilized in many different domains. There is, however, no general recommender system suitable for providing good recommendations in every scenario. In some domains each user is mainly interested in one specific type of items, and a suitable recommender system might base the recommendations on the specific details of the item contents. Such systems are called *content-based* recommender systems. In other cases a content-agnostic recommender system that does not need any information on the content of an item might be better. Such approaches are called *collaborative* recommender systems. These systems rely solely on identifying users having similar interests, and recommend the items that are popular amongst these users. Most recommender systems are based on one of these two approaches, or a combination of the two. Let's have a closer look at how these two main approaches to recommender systems work.

1.1 Collaborative filtering

The basic content-agnostic recommender system approach is called collaborative filtering. It is based on the assumption that if two users had similar taste in the past, it is likely that they will continue to share the same interests in the future. Let's look at a store selling videos as an example. If customer a and customer b have bought many of the same videos in the past, and customer a recently bought a new video, then it is likely that customer b also would be interested in this new video. This approach was first introduced by Goldberg et al. [13], and is called collaborative filtering because the users implicitly collaborate on making the recommendations. Through actions like rating or buying items, the users inform the system that other users with similar interests might also be interested in these items.

The big advantage of collaborative filtering is that there is no need for any information about the recommended items. The convenience of not having to provide metadata for items makes setting up the system easy, but also poses some drawbacks. For instance, how should the recommender system deal with new users? When a user has not rated or bought more than a few items, the recommender system might not have sufficient data to confidently consider this user's interests as similar to any other user's. Also, when there are few registered users, the likelihood of there being users considered to have similar taste is smaller. And what about new items that are added to the system? When nobody has rated or bought an item, the collaborative filtering algorithm has no way of knowing who might be interested in it. This results in recently deployed collaborative recommender systems performing poorly, since items cannot be recommended due to lack of data about users and their interests. This is called the cold-start problem and is discussed for example by Schein et al. [30].

1.2 Content-based filtering

Content-based recommender systems require information on the contents of each item. In order to predict a rating, the content-based recommender system compares the information about the contents of an item with a profile weighing the importance of different properties. Let's again look at the video store example described in the previous section, now using a content-based recommender system. If a customer in the past has bought "The Da Vinci Code", the recommender system might recommend other movies in the "mystery" or "thriller" genre, movies directed by Ron Howard or starring Tom Hanks, or other movies based on the works of Dan Brown. This is because the recommender system will have created a profile for the customer, valuing these properties of "The Da Vinci Code" highly.

The preference profile used by a content-based recommender system is very similar to the user history used by a collaborative recommender system. The difference is that while the collaborative approach builds up a profile of items the user is interested in, the content-based approach constructs a profile of item *attributes* the user is interested in. This can be done through actions like rating or buying items, as in a collaborative recommender system, or the system can ask the user to explicitly rank different attributes in order to build up a user specific taste profile.

By looking for items matching a taste profile instead of considering the actions of users with similar taste when making recommendations, content-based recommender systems overcome some of the drawbacks of collaborative recommender systems. For instance, content-based recommender systems have no cold-start problem: any item can be recommended if item attributes are available—there is no need for an item to have been bought or rated by a user. This is great for systems with few users, systems where new items are frequently added, or for recently deployed recommender systems.

In contrast to collaborative recommender systems, content-based recommender systems have the drawback of requiring well-organized attribute information about the items. Sometimes such metadata is available in a database along with the item, or can be extracted automatically from text documents. In many domains, however, attributes concerning subjective qualities such as “ease of use” and “aesthetically pleasing” could be useful. Such attributes are hard to extract automatically, and manually entering such information can be an expensive and error prone process. In order to overcome this cost issue, a possibility is to let the users enter metadata about the items. This does however not solve the problem with subjective properties, where users might have conflicting opinions. Different users might also use different words to describe the same quality or feature. This can make the user-provided metadata less valuable, as similar qualities might not be recognized as such by the recommender system [2].

In order to increase sales and customer satisfaction, a good recommender system should be able to recommend new, relevant, and interesting items to the users. In some domains content-based recommender systems often fail to achieve this goal, as the recommendations tend to either be too general (recommending all the best-selling items in a category), or too narrow (recommending all items matching a specific attribute), as discussed for example by Linden et al. [22]. Because of this, collaborative recommender systems tend to be more popular than pure content-based ones.

1.3 Privacy and censoring

In addition to positive aspects such as increased sales and customer satisfaction, deploying a recommender system can also pose risks that should be taken into consideration. When creating recommendations based on the actions of other users, the issue of privacy immediately arises. The recommender system, like any system, should not reveal the behavior and interests of individual users without the user’s consent. We will not discuss the topic of privacy in recommender systems further in this thesis, but recommend an article on privacy risks in collaborative filtering by Calandrino et al. [6]. Differential privacy aims to provide means to overcome the privacy issue in statistical databases. For a thorough introduction to differential privacy we refer to an article by Dwork and Roth [9].

A different issue can arise when dealing with information overload; recommender systems can hinder users from getting a full overview of the information available. If the recommender system in a large newspaper considers a user as mainly interested in articles presenting a conflict from one side, fewer articles presenting the issue from the other side might be presented to the user. As a consequence, the user might get an unbalanced picture of the issue. It might be correct that the user sympathizes with one side of the conflict, and prefer articles supporting his or her own view. However, it can be argued that, the user might be better off with a recommendation for an article presenting the conflict from the other side to get a fuller picture. Pariser has made a highly recommended TED Talk [25] on this topic, in addition to his book *The filter bubble: What the Internet is hiding from you* [26].

1.4 Applications and considerations

Recommender systems can answer many different questions and queries, making them useful in numerous applications and scenarios. Let's look at some of the queries recommender systems can answer, and how the answers to these queries can be useful.

1. *What is the likelihood that a specific user would be interested in a specific item?* This likelihood can be expressed in different ways, depending on the recommender system and scenario. In some cases one might be looking for the likelihood of the user buying the item. In other scenarios the predicted rating the user would give to the item is more interesting.

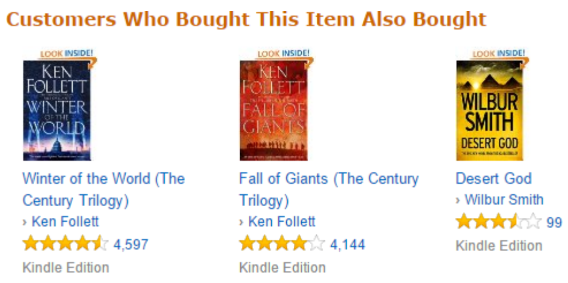
2. *Give a list of the items a specific user or set of users is most likely to be interested in.* This can be amongst all items, or narrowed down to items within a category or sharing some attribute, such as a set of promotional items. The possibility to specify more than one user can be very useful, for example if a group of friends want a recommendation for a movie that they would all enjoy.

3. *Give a list of the users who are most likely to be interested in a specific item or a list of items.* This can be amongst all users, or narrowed down to users matching some criterion, for example based on demographics or geographical information. This can be useful for example when sending out emails with special offers for promotional items.

4. *Give a list of the items that are most similar to a specific item or a list of items.* In a content-based recommender system, similarity would be based on similarity between the attributes of the items. In a collaborative recommender system, the similarity can be based on which items users are likely to be interested in if they are interested in the specified item or set of items. This query can be very useful for generating lists such as "Users buying this item also bought..." and "You might also be interested in...", as seen in Figure 1.

5. *Give a list of the users who have the most similar taste to a specific user or set of users.* In a collaborative recommender system, the similarity will depend on users having rated (or other user actions) several of the same items. In a content-based recommender system this is not necessary, as the preference profiles can be similar regardless of the users having rated many items in common. This can be useful for creating chat groups of users with similar interests. Of course introducing users to each other poses privacy and anonymity issues that must be handled with care.

The first query in this list is the most basic query a recommender system can answer, and many other queries can be answered by subsequently answering this basic query [1]. For example, the second query could be answered by repeatedly running the first query on the specified user(s) and all items in question, and then return the items most likely to be of interest to the user(s). The third query could similarly be answered by repeatedly running the first query on the specified item(s) and all users in question, and then return the users most likely to be interested in the item(s). Of course, many of the queries can also be reversed from "most likely" to "least likely". Some users might for example be interested in a list of the items they are least likely to enjoy. For the rest of this thesis, we will be concentrating on answering the basic query of predicting the rating a specific user would give to a specific item.


Customers Who Bought This Item Also Bought

Winter of the World (The Century Trilogy)
› Ken Follett
4,597
Kindle Edition

Fall of Giants (The Century Trilogy)
› Ken Follett
4,144
Kindle Edition

Desert God
› Wilbur Smith
99
Kindle Edition

Figure 1: A list of books recommended by Amazon, based on customers having bought "Edge of Eternity" by Ken Follet.

Another type of question a recommender system might be able to answer is why a user gets a specific recommendation. Being able to support the recommendations with reasons why a user might like an item can effectively improve the user's trust in the recommender system, as well as the user's willingness to adopt the recommendation, as discussed by Bilgic and Mooney [4]. See also the work by Herlocker et al. [18] for more information on how recommender systems can give explanations to support the recommendations. Figure 2 shows an example of explained recommendations in Spotify.

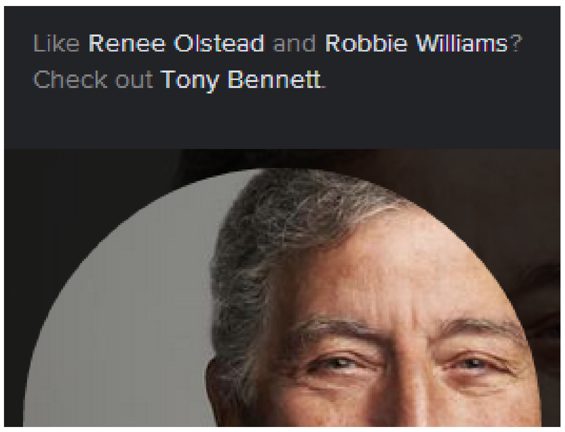

Like Renee Olstead and Robbie Williams?
Check out Tony Bennett.

Figure 2: Music recommendation by Spotify, explained and supported by related artists.

Depending on the domain characteristics, such as the number of users and items, a recommender system can be *memory-based* or *model-based*. Memory-based recommender systems keep the entire dataset in memory and calculate the recommendations in real-time. This requires the recommendation algorithm to be fast and the dataset to not be extremely large. Model-based recommender systems do precalculations on the dataset offline, resulting in a simpler model representing the entire dataset. Since more of the calculations have already been performed when using the recommender system, this allows for larger datasets. On the other hand, the recommendations might be of lower quality, as some of the data is thrown away during the precomputations. Also, model-based recommender systems often have more difficulty supporting the recommendations with explanations, as the model might not preserve the data required for this. The recommender systems looked at in this thesis are memory-based, but provide the possibility of using slightly stale data when calculating predictions. This can improve the speed of the recommender system by reusing earlier calculations, in exchange for slightly less accurate predictions. This will be discussed briefly in Section 4.1.

In many use cases, collaborative recommender systems might work very well when considering only a subset of the items, but less so when considering the entire set of items. A reason for this is that users might have similar taste when it comes to one category, for example books, while having conflicting opinions in other categories, for instance when choosing which music to listen to. Hence, a collaborative recommender system would be able to identify that the users have similar taste if only considering the subset of the items that are books, while not being able to find any similarity when looking at the entire item set. There are several approaches that seek to overcome this problem. One can for example use a hierarchical classification of the items. If two users do not seem to have similar taste when considering all items in a category, the recommender system can go down one level in the hierarchy, and check for similar taste when considering the items in each subcategory.

1.5 Graph-based collaborative filtering

This thesis will focus on collaborative recommender systems. Many different techniques can be applied by the recommendation algorithms in such systems. Some are *user-based*, and rely on identifying users with similar taste. Others are *item-based*, and provide predictions based on similarity between items instead of similarity between users. This similarity is not based on content, as discussed in Section 1.2, but rather that items are similar if users tend to have similar opinions about them. Linden et al. [22] give an interesting introduction to an item-based collaborative recommender system used by Amazon. The recommendation algorithms discussed in this thesis are user-based.

The first collaborative recommender systems were based on computations on matrices. When dealing with user ratings, such recommender systems represent the data as a matrix where each column corresponds to an item, and each row to a user. Cell (a, i) in the matrix contains the rating user a has given item i. How similar taste two users a and b have, can for example be calculated as the angle between the vectors represented by row a and row b in the matrix. How likely a user a is to like an item i can be calculated by looking at the values in column i for the rows corresponding to users having similar taste as user a.

In recent years, graph-based recommender systems have become increasingly popular. In these systems, the data is modeled as a graph instead of as a matrix. The graph-representation allows for transitivity between user tastes to be exploited when calculating predictions. Huang et al. [19] describe a graph-based recommendation algorithm where both items and users are vertices in an undirected graph. If a user a likes an item i, there is an edge between vertex a and vertex i. Now, how likely a user b is to like an item j can be calculated based on the length of the shortest path from b to j. The recommendation algorithms discussed in this thesis are graph-based, but quite different from the method proposed by Huang et al. [19].

No matter which techniques a recommender system is based upon, a few qualities are essential for the recommender system to be successful:

Speed Many recommender systems provide their recommendations over the Internet. A difference in loading time of a few seconds can decrease the customer satisfaction substantially [35]. A recommender system must therefore be able to provide accurate recommendations tremendously fast.

Scalability The algorithm must perform well on both small and large datasets. Large web shops, such as Amazon, can contain several hundred million items and users that the recommender system must be able to deal with.

Learning curve[1] The data a recommender system bases it's calculations on will

[1]A learning curve shows the relationship between experience and learning. A learning curve with a steep start represents rapid progress. Curiously, the common expression "a steep learning curve" is often used in everyday language when describing something that is hard to learn, contrary to the true meaning of a steep learning curve. In the case of recommender systems, the learning curve shows the relationship between available data and recommendation quality. Hence a good (or steep) learning curve indicates that the recommender system is able to calculate accurate predictions without requiring a lot of user data.

often be very sparse. When there are many items, most users will only have expressed an option on a small fraction of them. This should not prevent the recommender system from providing good recommendations for the users.

We will in Chapter 3 go in depth on a specific graph-based collaborative filtering algorithm, called *IRA*, and attempt to improve it. For good resources on more information on recommender systems in general, we recommend the books by Jannach et al. [21] and Ricci et al. [28].

Chapter 2

Clarifications and definitions

Throughout the rest of this thesis, knowledge about basic graph theory and basic graph algorithms such as *breadth-first search* will be assumed. For more information on graph theory and algorithms, the book by Dasgupta et al. [8] is a good starting point. Basic knowledge about set theory, such as binary relations, will also be assumed. A good resource on this and other topics in abstract algebra can be found in the book by Fraleigh [11].

In many of the illustrations in this thesis, items from very different product groups are used. This is for illustrative purposes, and not because the recommender system necessarily should consider items from unrelated product groups when identifying users with similar taste.

There are different ways to measure how well a user likes an item. In some domains the measure may be explicit, such as a user giving a rating or buying an item. In other domains the time spent reading an article or listening to a song might be a more applicable measure for how well the item is liked. For the rest of this thesis we will consider numeric user ratings as the measure of how well a user likes an item, but the concepts and algorithms discussed would also apply if a different measure was chosen.

2.1 Notations and definitions

Every recommender system must have a set of items, which we'll denote by I. The recommender system must also have a set of users, which we'll denote by U. The users can rate the items. The ratings must be constrained to lie within a set of valid rating values, called the *rating scale*, which we'll denote by S. We'll denote the largest valid rating value (the largest value in S) by $s_{max} \in S$. All the rating scales in this thesis will consist of consecutive integers, and be on the form $S = \{1, \ldots, s_{max}\} \subsetneq \mathbb{Z}^+$. When user $a \in U$ gives a rating to item $i \in I$, we'll denote this rating by $r_{a,i} \in S$. Every user can give ratings to multiple items, but only one rating to each item. We'll denote the set of items that user $a \in U$ has rated by $I_a \subseteq I$.

For easier reference, we have that

- I is the set of items
- U is the set of users
- s_{max} is the largest valid rating value
- $S = \{1, \ldots, s_{max}\}$ is the set of valid rating values
- $I_a \subseteq I$ is the set of items rated by user $a \in U$
- $r_{a,i} \in S$ is the rating given to item $i \in I_a$ by user $a \in U$

Considering the example in Figure 3, we have that

- $I = \{game, picture, book, cd\}$
- $U = \{a, b\}$
- $S = \{1, \ldots, 7\}$

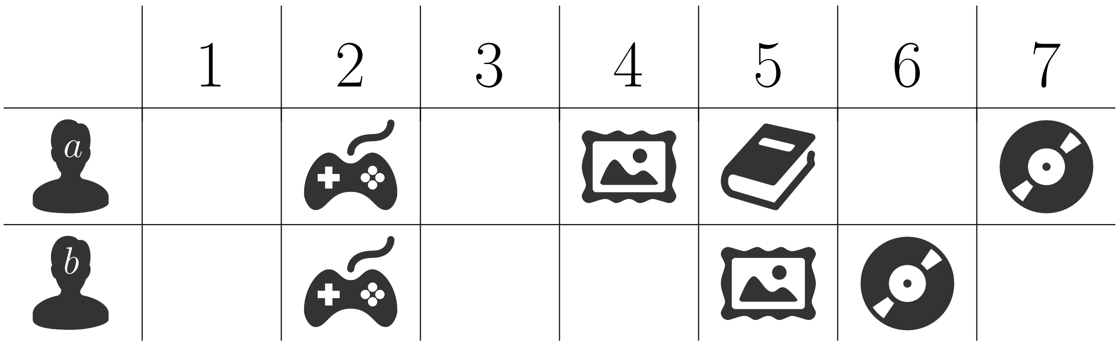


Figure 3: An example showing the use of I, U, and S. Two users have rated items on a scale from 1 to 7. User a has given the ratings 2, 4, 5, and 7 to the items *game*, *picture*, *book*, and *cd*. User b has given the ratings 2, 5, and 6 to the items *game*, *picture*, and *cd*.

2.2 Clarifications for pseudocode

To improve pseudocode readability, the following assumptions and decisions apply to all upcoming algorithms.

- Standard mathematical operations and notations are used in the pseudocodes. For example $|n|$ gives the absolute value of n if n is a number, and the number of elements in n if n is a set or a list. Binary operators such as $\cap$ and $\setminus$ can be used on sets, while $\in$ can be used on sets and lists.
- $\leftarrow$ is an assignment operator, while $=$ checks for equality.
- The same value can be assigned to multiple variables by having a comma separated list of the variables the value should be assigned to before the assignment operator $\leftarrow$, and the value that should be assigned to the variables after the assignment operator.
- Lists can be appended to while being iterated through. The appended elements will then also be iterated through.
- Items in arrays can be accessed through the []-operator. For example *container* $[i]$, where *container* is an array, will give the element at position i in *container*.
- All variables defined in Section 2.1 are available in the algorithms. For example, $r_{a,i}$ will be available as the rating user a has given item i. Similarly, $|I_a \cap I_b|$ is available as the size of the set of items both user a and user b have rated.
- When a variable has not been defined, its value will be `null`. For example, $r_{a,i} = \texttt{null}$ for $i \notin I_a$.
- Functions can return multiple values, but each function will always return the same number of values. Multiple return values are denoted by a comma separated list. When assigning the returned values to variables, the function call is assigned to a comma separated list of variables. The number of variables will always be the same as the number of values the function returns. The first return value is assigned to the first variable, etc.
- $\frac{0}{0}$ is an indeterminate form, and hence has no defined value. For simplicity, we define that $\frac{0}{0} = 1$ in our pseudocodes. (This is used in Algorithm 6.)

We assume the following functions to be available in the algorithms:

- APPEND(*container*, *elm*), where *container* is a list, adds *elm* to the end of the list.

- REVERSE(*container*), where *container* is a list, reverses the list such that subsequent iterations will iterate through the elements of the list in the opposite order.
- MAX(a, b), where $a, b \in \mathbb{R}$, returns the largest of a and b.

2.2.1 Variables and functions specific to this thesis

Variables and functions introduced later in this thesis are available in the algorithms appearing after the variable introductions. The variables that will be used in pseudocodes apart from the ones defined in Section 2.1 follow.

- *hort_common*, introduced in Definition 1.
- *hort_frac*, introduced in Definition 1.
- t_diff_{max}, introduced in Definition 2.
- P_a, where $a \in U$, introduced in Definition 3.
- *max_path_length*, introduced in Section 3.2, prior to Algorithm 3.
- $t_{a \to b}$, where $a, b \in U$, defined in Equations (1) and (3). When this is used in a pseudocode, it refers to the version of it minimizing $t\_diff_{a \to b}$ (defined in Equations (2) and (4)).

In addition, the following functions will be used by the algorithms:

- TRANS-FUNC-IRA(*sign*, *offset*) returns a callable function $t : \mathbb{R} \to \mathbb{R}$ such that $t(r) = sign \cdot r + offset$.
- TRANS-FUNC-SCALING$(min_a, max_a, min_b, max_b)$ returns a callable function $t : \mathbb{R} \to \mathbb{R}$ such that $t(r) = (r - min_a) \cdot \frac{max_b - min_b}{max_a - min_a} + min_b$. (Note that $\frac{0}{0} = 1$ in pseudocodes, as previously defined.)

Chapter 3

IBM's Intelligent Recommendation Algorithm (IRA)

In the article "Horting hatches an egg: A new graph-theoretic approach to collaborative filtering", published in "KDD '99 Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery data mining" in 1999, Charu C. Aggarwal, Joel L. Wolf, Kun-Lung Wu and Philip S. Yo, all working at the IBM T. J. Watson Research Center at the time, introduced a new approach to collaborative recommender systems [1]. The method they introduced, called the Intelligent Recommendation Algorithm (IRA), is the one we will attempt to improve in this thesis. In their article, Aggarwal et al. [1] compared IRA to two other recommender systems, namely Firefly [33] and LikeMinds [14]. These recommender systems also use collaborative filtering, but they are not graph-based. In this thesis, we focus on graph-based collaborative filtering recommender systems, and hence will not go into details on Firefly and LikeMinds. In this chapter we will have a detailed look at how IRA works. Our presentation will attempt to follow a pedagogically sound path, starting with easy examples, and ending with a detailed algorithm described in pseudocode.

3.1 Identifying compatible users

Being a collaborative filtering system, IRA relies on identifying users with compatible taste, and provides predictions based on the *compatibility* between users. A naive way to identify users with compatible taste could be to, for every pair of users, look at all the items both users have rated and compare the ratings given to these items. One could use, for instance, the *average difference* between the ratings to determine how compatible the users are to each other. Let's try this approach on the example in Figure 3 (in Section 2.1).

The items both users have rated are *game*, *picture*, and *cd*. Both users have given *game* the rating 2, so the difference in rating for this item is 0. For both *picture* and *cd*, the difference in rating is 1. Using the average difference as *compatibility measure*, we get a compatibility between the two users of $\frac{0+1+1}{3} = \frac{2}{3}$. Whether this is a good or bad compatibility will vary between use cases. In particular, the size of the scale will play a big role in determining what is a good compatibility. If the scale has size 2, for example, a compatibility of $\frac{2}{3}$ is very bad, as the users would be in complete disagreement more often than they agree with each other. If the scale had size 100, however, a compatibility of $\frac{2}{3}$ could be very good. Since *book* is only rated by one of the users, this item is not used in the calculation of the compatibility between the users. Obviously, other measures for compatibility than the average difference can be used, for example adjusted cosine similarity, Spearman's rank correlation coefficient, or the mean squared difference [28].

This straightforward approach makes a lot of sense assuming two users can have compatible taste only if they have low average difference when rating the same items. But IRA takes the notion of compatible taste two steps further than this. Firstly, what if one user is in general more positive than the other and always gives higher ratings? Users could still agree on what they like and what they dislike, even though they do not agree on the exact rating an item should receive. Secondly, and more radically, two users who always rate totally opposite could still predict each other's behavior. Instead of assuming that users can only have compatible taste when having low average difference between the ratings given the same items, IRA attempts to discover a *transformation function* that can make the users compatible.

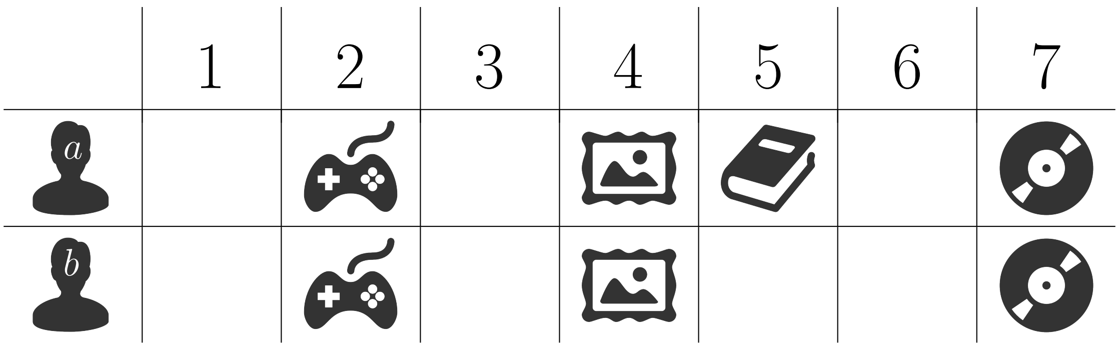


Figure 4: An example showing the ratings of two users such that the average difference equals zero. See Figure 3 in Section 2.1 for details on how to interpret the figure.

If, when applying the transformation function to one of the users' ratings, the transformed ratings have low average difference compared to the ratings of the other user, the users are said to have compatible taste. Let's look at the example in Figure 4.

In this example, the items both users have rated are *game*, *picture*, and *cd*, and for all of them, the users have given the same rating. Hence a good transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ from a rating by user a to a predicted rating for user b could be $t_{a \to b}(r_a) = r_a$. When looking at only $i \in I_a \cap I_b$, we see that $t_{a \to b}(r_{a,i}) - r_{b,i} = 0$ in all cases. Hence we get that the average difference $t\_\mathit{diff}_{a \to b}$ between the transformed ratings by user a and the ratings by user b equals 0, which means that the users have compatible taste. Based on user a's rating of *book*, we can predict that user b would give *book* the rating $p_{a \to b, book} = t_{a \to b}(r_{a,book}) = r_{a,book} = 5$.

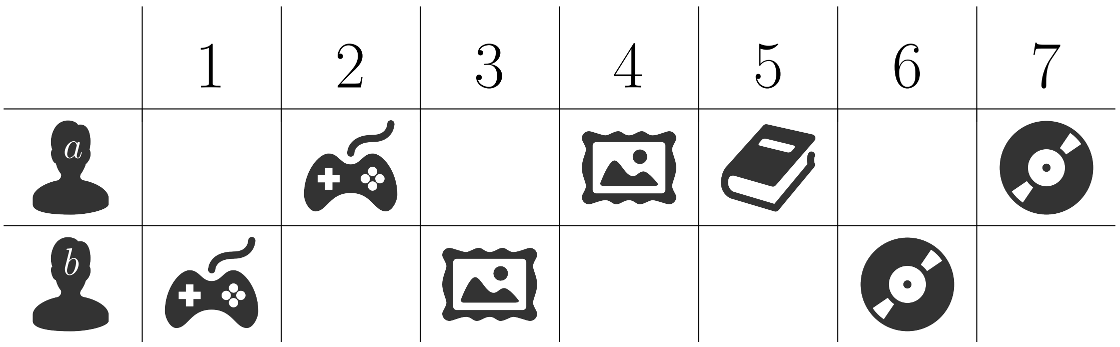


Figure 5: An example showing the ratings of two users such that a transformation function with offset −1 gives an average difference equal zero. See Figure 3 in Section 2.1 for details on how to interpret the figure.

Consider now Figure 5. In this example, the items both users have rated are *game*, *picture*, and *cd*, and for all of them, user b has a rating of value 1 less than user a. Hence a good transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ from a rating by user a to a predicted rating for user b could be $t_{a \to b}(r_a) = r_a - 1$, which we could call an offset of −1. When looking at only $i \in I_a \cap I_b$, we see that $t_{a \to b}(r_{a,i}) - r_{b,i} = 0$ in all cases. Hence we get that the average difference $t\_\mathit{diff}_{a \to b}$ between the transformed ratings by user a and the ratings by user b equals 0, which means that the users have compatible taste. Based on user a's rating of *book*, we can predict that user b would give *book* the rating $p_{a \to b, book} = t_{a \to b}(r_{a,book}) = r_{a,book} - 1 = 4$.

In addition to identifying compatible users whose ratings are offset, as in the previous example, the IRA algorithm can also identify users whose ratings are more or less opposite of each other as compatible. This is based on the assumption that two users with opposing taste can still provide information on whether the other user will like an item or not. If you like action movies and dislike romantic comedies, while your friend dislikes action movies and likes romantic comedies, even though you do not have the same taste in movies, one could expect that if your friend likes a movie of any of these two types, you would probably not like it, and if your friend dislikes it, you will probably like it.

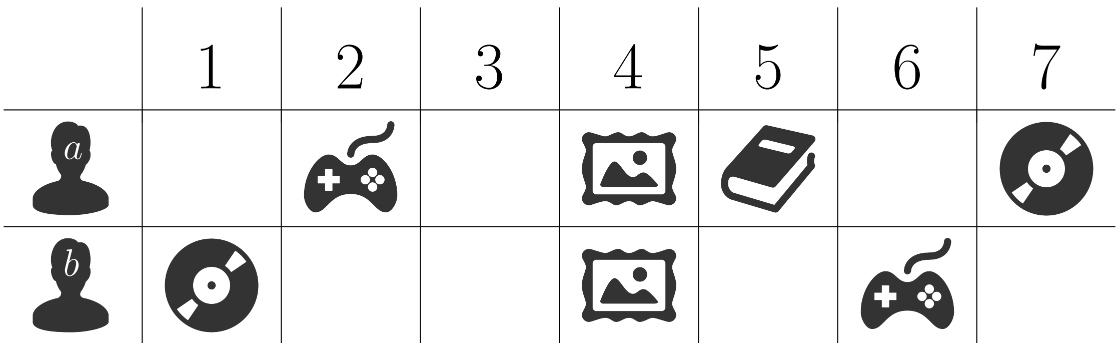


Figure 6: An example showing the ratings of two users such that a transformation function inverting the ratings around the value four, gives an average difference equal zero. See Figure 3 in Section 2.1 for details on how to interpret the figure.

Consider Figure 6. In this example, the items both users have rated are *game*, *picture*, and *cd*. We see that the ratings by user b are in a sense the opposite of those by user a. User b's ratings are flipped around 4 compared to the ratings by user a. Hence a good transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ from a rating by user a to a predicted rating for user b could be $t_{a \to b}(r_a) = -r_a + 8$. When looking at only $i \in I_a \cap I_b$, we see that $t_{a \to b}(r_{a,i}) - r_{b,i} = 0$ in all cases. Hence we get that the average difference $t\_diff_{a \to b}$ between the transformed ratings by user a and the ratings by user b equals 0, which means that the users have compatible taste. Based on user a's rating of *book*, we can predict that user b would give *book* the rating $p_{a \to b, book} = t_{a \to b}(r_{a,book}) = -r_{a,book} + 8 = 3$.

We are now ready to specify the details of how IRA defines a compatibility measure based on the concept of a transformation function. To find the transformation function that makes two users a and b as compatible as possible, the IRA algorithm chooses the version of

$$t_{a \to b}(r_a) = sign \cdot r_a + offset \tag{1}$$

where $sign \in \{-1, 1\}$, $offset \in \mathbb{Z}$, that minimizes

$$t\_diff_{a \to b} = \sum_{i \in I_a \cap I_b} |t_{a \to b}(r_{a,i}) - r_{b,i}| \tag{2}$$

If $t\_diff_{a \to b}$ is not greater than some predetermined threshold, we say that user a and user b have compatible taste.

Algorithm 1 Algorithm for finding the best IRA transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ for predicting ratings for user b, based on ratings by user a. See Section 2.2 for general clarifications for pseudocodes.

```
Input: Two users $a, b \in U$
Output: Two values:
    1. The IRA transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ minimizing $t\_diff_{a \to b}$
    2. The minimized $t\_diff_{a \to b}$
function BEST-TRANS-FUNC($a, b$)
    $best\_sign \leftarrow 0$
    $best\_offset \leftarrow 0$
    $best\_diff \leftarrow \infty$

    for all $sign \in \{-1, 1\}$ do                    ▷ All feasible transformation functions
        for all $offset \in \{-s_{max}, \ldots, 2 \cdot s_{max}\}$ do
            $diff \leftarrow 0$
            for all $i \in I_a \cap I_b$ do                ▷ Calculate total difference
                $diff \leftarrow diff + |(sign \cdot r_{a,i} + offset) - r_{b,i}|$
            end for
            if $diff < best\_diff$ then                ▷ New best transformation function
                $best\_sign \leftarrow sign$
                $best\_offset \leftarrow offset$
                $best\_diff \leftarrow diff$
            end if
        end for
    end for

    $avg\_diff \leftarrow best\_diff / |I_a \cap I_b|$          ▷ Calculate average difference
    $t \leftarrow$ TRANS-FUNC-IRA($best\_sign$, $best\_offset$)
    return $t$, $avg\_diff$
end function
```

Algorithm 1 shows pseudocode for finding the transformation function $t_{a \to b}$ that minimizes $t\_diff_{a \to b}$. This is done by exhaustive search—every feasible version of $t_{a \to b}$ is tried, and the best one is chosen. Feasible means that the transformation function maps at least one valid rating value to a value within the rating scale: $\exists\, r \in S \mid t_{a \to b}(r) \in S$. When $sign = 1$, $t_{a \to b}(r) = r + offset$. $1 \leq r \leq s_{max}$, so $offset \in \{1 - s_{max}, \ldots, s_{max} - 1\}$ all yield feasible transformation functions. When $sign = -1$, $t_{a \to b}(r) = -r + offset$. $1 \leq r \leq s_{max}$, so $offset \in \{2, \ldots, 2 \cdot s_{max}\}$ all yield feasible transformation functions. By trying all combinations of $sign \in \{-1, 1\}$

and *offset* $\in \{-s_{max}, \ldots, 2 \cdot s_{max}\}$, we know that all feasible versions of $t_{a \to b}$ have been tried. This algorithm has time complexity $O(|S| \cdot n)$, where $n = |I_a \cap I_b|$. Generally the size of the rating scale is not very large, and the number of items both users have rated is also generally not very large, hence this algorithm should run very fast. As mentioned in Section 4.3, there are also other ways to calculate the best transformation function, giving alternative running times, for example $O(n \cdot \log n)$, where $n = |I_a \cap I_b|$.

In addition to the users having compatible taste, IRA has another condition that must be satisfied before one user's ratings can be used to predict ratings for another user, namely the concept of *horting*[1].

Definition 1. User $b \in U$ *horts* user $a \in U$ if there either is a large number of items they both have rated

1. $|I_a \cap I_b| \geq hort\_common$

or user a has rated a large portion of all the items user b has rated

2. $\frac{|I_a \cap I_b|}{|I_b|} \geq hort\_frac$

where $hort\_common \geq 1$ and $hort\_frac \leq 1$ are predetermined thresholds.

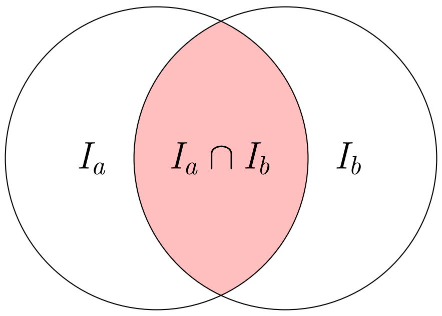


Figure 7: The first condition for horting: $|I_a \cap I_b| \geq hort\_common$. $I_a \cap I_b$ (the set of items both users have rated) is colored red.

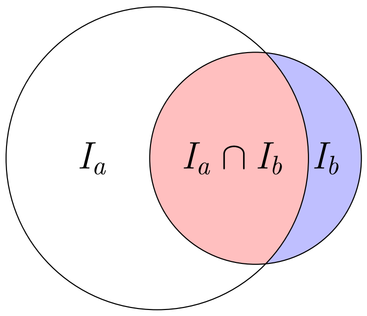


Figure 8: The second condition for horting: $\frac{|I_a \cap I_b|}{|I_b|} \geq hort\_frac$. $I_a \cap I_b$ (the set of items both users have rated) is colored red, while I_b (the set of items user b has rated) is the union of the blue area and the red area.

The horting condition attempts to improve the prediction quality by preventing the algorithm from basing predictions on users whose compatibility is based on very few data points. If, for example, two users have only rated one item in common, there will always exist a transformation function making the two users completely

[1]The word horting was introduced by Aggarwal et al. [1]. It is based on the noun "cohort", but "co" was omitted since horting describes a property that is not necessarily symmetric.

compatible. Basing the further prediction process on this compatibility, however, would very often give poor results.

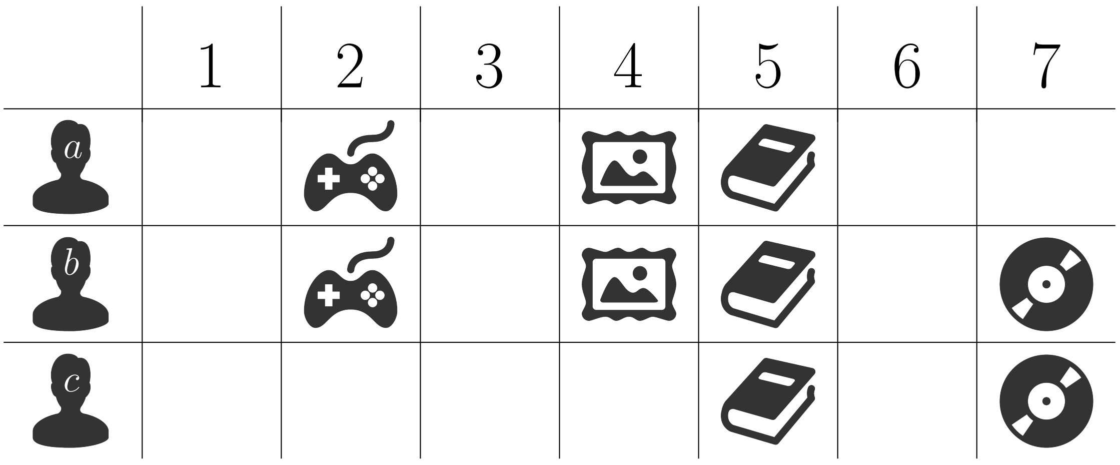


Figure 9: An example showing that horting is not generally symmetric or transitive. See Figure 3 in Section 2.1 for details on how to interpret the figure.

Consider Figure 9 for an example showing that horting is not symmetric or transitive. Let $hort\_common = 3$ and $hort\_frac = \frac{3}{4}$. We have that $|I_a \cap I_b| = 3 \geq hort\_common$, so a horts b and b horts a. $|I_b \cap I_c| = 2 < hort\_common$, hence the first condition of horting between user b and user c is not satisfied. $\frac{|I_b \cap I_c|}{|I_b|} = \frac{2}{4} < hort\_frac$, so b does not hort c. $\frac{|I_b \cap I_c|}{|I_c|} = \frac{2}{2} \geq hort\_frac$, so c horts b. $|I_a \cap I_c| = 1 < hort\_common$, hence the first condition of horting between user a and user c is not satisfied. $\frac{|I_a \cap I_c|}{|I_a|} = \frac{1}{3} < hort\_frac$, so a does not hort c. $\frac{|I_a \cap I_c|}{|I_c|} = \frac{1}{2} < hort\_frac$, so c does not hort a. We have that horting is not symmetric, as c horts b, while b does not hort c. We have that horting is not transitive, as c horts b and b horts a, while c does not hort a.

We are now ready to define the *prediction* relationship.

Definition 2. User $a \in U$ *predicts* user $b \in U$ if

1. user b horts user a (see Definition 1)

and the error in the transformation function between the users is small

2. $t\_diff_{a \to b} \leq t\_diff_{max}$

where $t\_diff_{a \to b}$ is the minimized difference between $t_{a \to b}(r_{a,i})$ and $r_{b,i}$ for $i \in I_a \cap I_b$, as defined in Equation (2), and t_diff_{max} is some predetermined threshold. (Note the opposite order of prediction compared to that of horting.)

Definition 3. P_a is the set of all users predicting a. $P_a = \bigcup_{u \in U} (u \mid u \text{ predicts } a)$.

Because horting is a condition for prediction (but in the opposite order), prediction inherits horting's properties of not being symmetric or transitive. This can easily be seen when considering Figure 9. Let $t\_diff_{max} = 0$. We have that the transformation function $t(r) = r$ yields an average difference of zero between all user pairs. Hence, the second condition in Definition 2 for one user to predict another is satisfied for all user pairs. Let $hort\_common = 3$ and $hort\_frac = \frac{3}{4}$, as in the example following Figure 9. Since c horts b, while b does not hort c, we have that b predicts c, while c does not predict b, and prediction is not symmetric. Because c horts b and b horts a, while c does not hort a, we have that a predicts b and b predicts c, while a does not predict c, and prediction is not transitive.

Algorithm 2 shows pseudocode for determining whether there is a prediction relation from user a to user b, and if so, finding the best transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$. This is done by first checking that at least one of the two conditions for user b to hort user a is satisfied. If user b horts user a, the function described in Algorithm 1 is called to find the best transformation function from user a to user b. If this transformation function has a small enough error, user a predicts user b, and the transformation function is returned.

Algorithm 2 Algorithm for determining whether there is a prediction relation from user a to user b, and if so returning the transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$. See Section 2.2 for general clarifications for pseudocodes.

```
Input: Two users $a, b \in U$
Output: The best transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ if $a$ predicts $b$, or
    null if $a$ does not predict $b$
 1: function RELATE($a, b$)
 2:     if $|I_a \cap I_b| < hort\_common$ and $|I_a \cap I_b|/|I_b| < hort\_frac$ then
 3:         return null                        ▷ The horting condition is not satisfied
 4:     end if
 5:
 6:     $t, t\_diff \leftarrow$ BEST-TRANS-FUNC($a, b$)
 7:
 8:     if $t\_diff > t\_diff_{max}$ then
 9:         return null      ▷ The transformation function is not accurate enough
10:     end if
11:
12:     return $t$                              ▷ Return the transformation function
13: end function
```

3.2 Creating predictions by transitivity

Now that we have seen how IRA determines the prediction relations between the users, we want to show how IRA uses these to actually predict how well a user will like an item he or she has not rated yet. To predict which rating user a would give item i, we can look at the users who predict user a and see if any of them have rated item i. Let's say we find such a user b. We can then predict that user a would give item i the rating $p_{b \to a,i} = t_{b \to a}\left(r_{b,i}\right)$. If there are several users who predict user a and has rated item i, we can take the average of the calculated predictions as the final prediction $p_{a,i}$.

The problem with this intuitive approach is what happens when there are no users who both predict user a and have rated item i: a prediction for user a and item i cannot be calculated! To overcome this problem, IRA models the prediction relations as a directed graph. Every user corresponds to a vertex in the graph, and for every prediction relation, an edge is added from the predicted user's vertex to the predicting user's vertex. With this graph, IRA can exploit some sort of transitivity in the predictions. Even though the prediction relation is not transitive (z predicts y and y predicts x does not imply that z predicts x, according to Definition 2 and seen in Figure 9), z's rating can be transformed into a predicted rating for y, which in turn can be transformed into a predicted rating for x! So, to create the prediction, IRA performs a *breadth-first search* from user a, searching for a user who has rated item i. When such a user has been found, its rating can be transformed back to a predicted rating for user a, even if the user does not predict user a directly. Consider the example in Figure 10.

Stage 0 We want to predict how well the red user a will like an item μ. The graph shows the users as vertices, and the directed edges correspond to prediction relations. If there is an edge from user x to user y, it means that y predicts x. Note that not all edges are bidirectional. For example, f predicts d, while d does not predict f. The reason for this is that the prediction relation is not in general symmetric, as seen in Figure 9 and the following explanation. The users with rings around them (b, c, f, and i) are the users who have rated item μ.

Stage 1 A breadth-first search is performed from user a, searching for a user having rated item μ (the users with rings around them). During the first stage, the users colored blue (d, e, and h) are discovered, as these have distance 1 from a. None of these users have rated item μ, so the breadth-first search continues from the discovered users.

Stage 2 During the second stage of the breadth-first search, the users colored

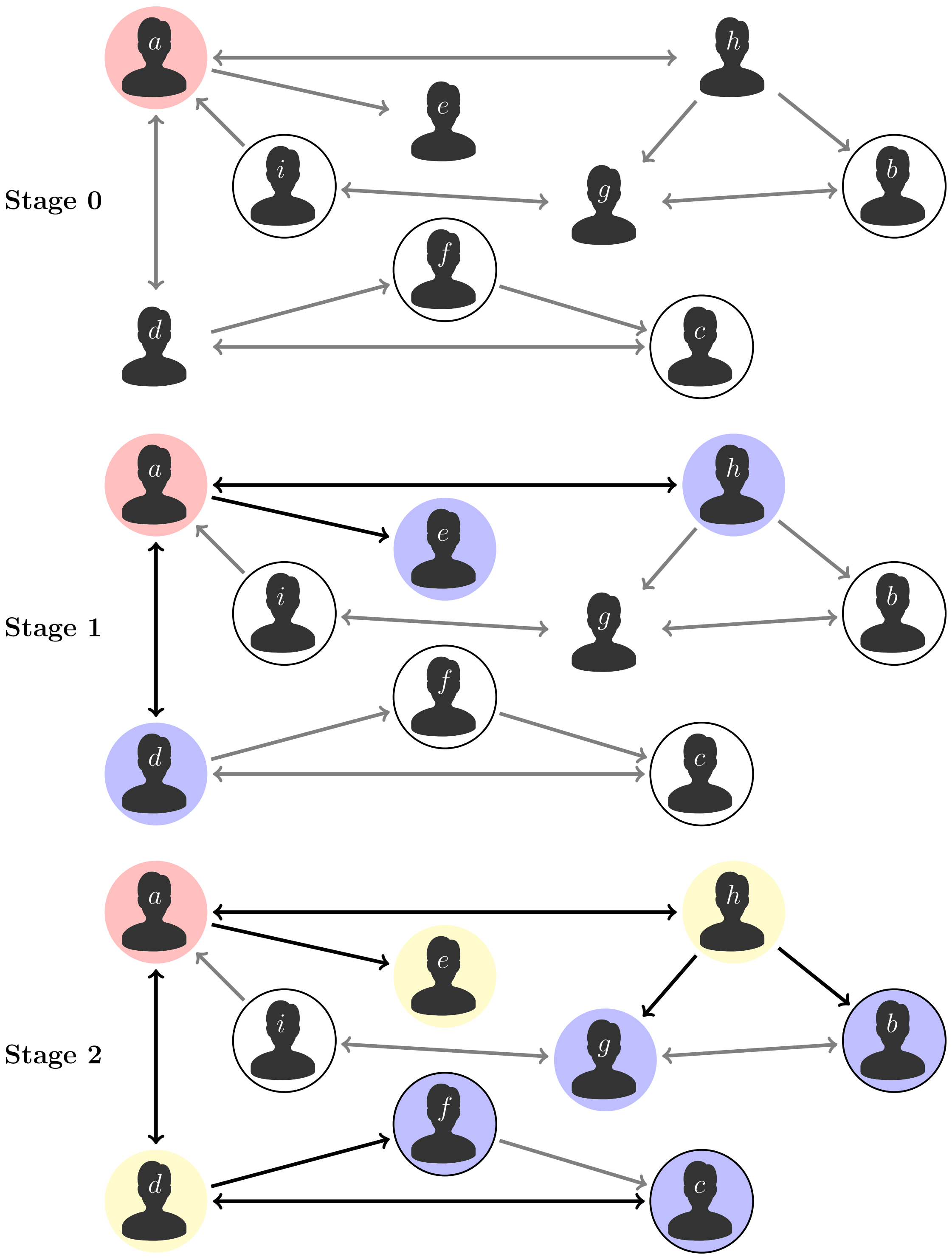


Figure 10: A breadth-first search searching from user *a* for a user having rated a specific item (users with rings around them) in a prediction graph.

blue (b, c, f, and g) are discovered, as these have distance 2 from a. The users colored yellow (d, e, and h) are the users that have been discovered in previous stages. The breadth-first search is looking for a user having rated item μ (the users with rings around them). Since such a user has been found (b, c, and f), the breadth-first search is terminated.

When the breadth-first search has found one or more users that have rated item μ, these ratings are used to predict which rating user a will give to the item. In this example, three shortest paths from user a to a user having rated μ were found: $a \to d \to c$, $a \to d \to f$, and $a \to h \to b$. Each arrow $x \to y$ represents a prediction relation between the head y and the tail x, and each such prediction relation has an associated transformation function $t_{y \to x} : \mathbb{R} \to \mathbb{R}$. For each shortest path $w \to x \to \cdots \to y \to z$, where z has rated μ, user z's rating $r_{z,\mu}$ can be transformed into a predicted rating $p_{z \to y \to \dots \to x \to w,\mu} = t_{z \to y} \circ t_{y \to \dots} \circ \cdots \circ t_{\dots \to x} \circ t_{x \to w}\left(r_{z,\mu}\right)$ for user w. The average of these predicted ratings is used as the final predicted rating $p_{w,\mu}$. In our example we have $p_{c \to d \to a,\mu} = t_{c \to d} \circ t_{d \to a}\left(r_{c,\mu}\right)$, $p_{f \to d \to a,\mu} = t_{f \to d} \circ t_{d \to a}\left(r_{f,\mu}\right)$, and $p_{b \to h \to a,\mu} = t_{b \to h} \circ t_{h \to a}\left(r_{b,\mu}\right)$. We now take the average of these $p_{a,\mu} = \frac{p_{c \to d \to a,\mu} + p_{f \to d \to a,\mu} + p_{b \to h \to a,\mu}}{3}$ as the final prediction for how well user a will like item μ. Note that this value is not necessarily in S. This can very often be useful, as it says something about the confidence of the prediction. A predicted rating of 13.1 indicates a higher likelihood of the item getting the rating 13 than if the predicted rating was 12.5, even though they both round to 13. One can easily constrain $p_{a,\mu}$ to be within S by rounding it to the nearest value in S, if this is more desirable.

As the breadth-first search goes through more stages, and the distance from the source of the search increases, the chance of the predicted rating being accurate decreases. Every prediction relation has a chance of being erroneous, and each transformation function can even introduce an average error up to t_diff_{max}. Hence, the longer the path of relations used to calculate the prediction is, the less accurate the prediction will be. Because of this, IRA, introduces a depth limit *max_path_length* on the breadth-first search. When reaching a distance further than *max_path_length* from the source, the breadth-first search is terminated. The algorithm then returns as if a path to a user having rated the item in question could not be found, and as result a prediction cannot be calculated.

Algorithms 3, 4, and 5 show pseudocode for predicting the rating a specific user would give to a specific item.

Algorithm 3 calls the code in Algorithm 4 to search for users having rated the item. Then the code in Algorithm 5 is called to transform the ratings by the users found to predicted ratings for the user in question. Finally the average of the predicted

Algorithm 3 Algorithm for calculating a predicted rating for a specific user and item. See Section 2.2 for general clarifications for pseudocodes.

```
Input: A $user \in U$ and an $item \in I$
Output: A prediction for the rating $user$ would give to $item$, or
    null if no prediction could be calculated
 1: function PREDICT($user$, $item$)
 2:     $L, D, M \leftarrow$ PREDICT-BFS($user$, $item$)        ▷ Perform breadth-first search
 3:     $R \leftarrow$ PREDICT-BACKTRACK($L, D, M$)              ▷ Perform backtracking
 4:
 5:     if $\varnothing = R$ then
 6:         return null                                      ▷ No prediction could be calculated
 7:     end if
 8:
 9:     $rating \leftarrow 0$
10:     for all $r \in R$ do
11:         $rating \leftarrow rating + r$                    ▷ Calculate the sum of the predictions
12:     end for
13:
14:     return $rating/|R|$                                  ▷ Return the average of the predictions
15: end function
```

ratings is returned as the final predicted rating. If no predicted rating could be calculated, `null` is returned.

Algorithm 4 performs a breadth-first search, searching from the user in question for users having rated the item in question. The pseudocode shows a fairly standard depth-limited breadth-first search, so, along with the comments in the pseudocode, the algorithm should be self-explanatory and will not be further explained here. Let u be the user in question, and i be the item in question. If there are no users at distance less than or equal to max_path_length from user u having rated item i, the empty list is returned. If we let $d \leq max\_path\_length$ be the shortest distance from user u to a user having rated item i, the list of all users at distance d from user u having rated item i is returned. Note that if user u has rated item i, the distance from user u to a user having rated item i is 0, and the list containing only user u is returned.

Algorithm 5 transforms the ratings given by the users found by the breadth-first search back to predicted ratings for the user in question. This is done by iterating through the users in the opposite order of which they were visited by the breadth-first search. This means that the users furthest from the user in question will be

Algorithm 4 Algorithm searching from a specific user for users having rated a specific item. See Section 2.2 for general clarifications for pseudocodes.

Input: A *user* $\in U$, and an *item* $\in I$
Output: Three values:

1. An ordered list of the users visited during the search
2. An array with distances from *user* to other users
3. A list of the users closest to *user* having rated *item*

```
1:  function PREDICT-BFS($user$, $item$)
2:      $L \leftarrow [user]$                                  ▷ List of users to visit
3:      $D \leftarrow [\infty, \dots, \infty]$  ▷ Array of size $|U|$ with distance from $user$ to other users
4:      $M \leftarrow []$                                      ▷ List of users having rated $item$
5:      $D[user] \leftarrow 0$
6:      $depth\_limit \leftarrow max\_path\_length$            ▷ Set the depth limit for the search
7:
8:      for all $u \in L$ do                                   ▷ Perform the breadth-first search
9:          if $D[u] \leq depth\_limit$ then
10:             if null $= r_{u,item}$ then                    ▷ User $u$ has not rated $item$
11:                 for all $n \in P_u$ do                     ▷ Users predicting user $u$
12:                     if $\infty = D[n]$ then                ▷ User $n$ has not been visited before
13:                         $D[n] \leftarrow D[u] + 1$
14:                         APPEND($L, n$)                     ▷ Put $n$ in list of users to visit
15:                     end if
16:                 end for
17:             else                                           ▷ User $u$ has rated $item$
18:                 $depth\_limit \leftarrow D[u]$             ▷ Update the depth limit for the search
19:                 APPEND($M, u$)                             ▷ Add $u$ to list of users having rated $item$
20:             end if
21:         end if
22:     end for
23:     return $L, D, M$
24: end function
```

visited first. For each user u we visit, we check all the users predicting user u that are further away from the user in question than user u, and transform any ratings we have calculated for those users to predicted ratings for user u. After iterating through all the users, we will have a list containing one predicted rating for each shortest path from the user in question to any of the users discovered by the breadth-first search. This list is returned as the list of predicted ratings for the user and item in question. Note that if no users were discovered by the

Algorithm 5 Algorithm for transforming ratings found in a breadth-first search from a specific user back to predicted ratings for that user. See Section 2.2 for general clarifications for pseudocodes.

```
Input: A $user \in U$, and an $item \in I$
    An ordered list $L$ or users visited in a breadth-first search
    An array $D$ with distances from $user$ to other users
    A list $M$ of the users closest to $user$ having rated $item$
Output: A list of predicted ratings for $user$ and $item$
 1: function PREDICT-BACKTRACK($user, item, L, D, M$)
 2:     $R \leftarrow [[], \ldots, []]$                ▷ Array of size $|U|$ with lists of ratings
 3:     for all $u \in M$ do                            ▷ Users having rated $item$
 4:         APPEND($R[u], r_{u,item}$)                  ▷ Add rating to $u$'s list of ratings
 5:     end for
 6:     REVERSE($L$)      ▷ Reverse the list, so iteration becomes last-in-first-out
 7:     for all $u \in L$ do                            ▷ Perform the backtracking
 8:         for all $n \in P_u$ do                      ▷ Users predicting $u$
 9:             if $D[n] + 1 = D[u]$ then               ▷ $n$ is reachable from $u$
10:                 for all $p \in R[n]$ do
11:                     ▷ Transform rating for $n$ to rating for $u$ and add it to $u$'s list
12:                     APPEND($R[u], t_{n \rightarrow u}(p)$)
13:                 end for
14:             end if
15:         end for
16:     end for
17:     return $R[user]$                                ▷ List of predicted ratings for $user$
18: end function
```

breadth-first search, the returned list will be empty.

Together, Algorithms 3, 4, and 5 perform a depth-limited breadth-first search with backtracking, and the combined time complexity is that of a standard breadth-first search: $O(n + m)$, where n is the number of vertices (users), and m is the number of edges (prediction relations). Because the graph will normally be very sparse, combined with the depth limit *max_path_length*, the algorithms will usually run very fast, and hence predictions can be calculated with great speed.

Chapter 4

Improving IRA—Scaling the scales

The article "Horting hatches an egg: A new graph-theoretic approach to collaborative filtering" by Aggarwal et al. [1], which introduced IRA, has been cited more than 450 times, according to Google Scholar [31]. Despite this high number of citations, we have not found in the literature any attempts of further developing and extending IRA to make it perform even better. In this chapter we suggest such an improvement, which we have called "Scaling the scales".

4.1 Scaled scales

A natural extension to the assumption that users with compatible taste can rate offset and/or inverse in relation to each other, is that users can use differently sized ranges of the scale. One user might be very effusive, and often give top or bottom ratings. Another user might find that everything is more or less ok. This user, who seldom loves or hates an item, will naturally not use the top or bottom ratings very frequently. In this scenario, the two users can actually agree with each other on what is good and what is bad—they are just using differently sized ranges to express the difference between good and bad. Note that the transformation function used by IRA as described in the previous chapter, would not capture this type of taste compatibility between a pair of users.

This is the idea behind the suggested improvement to IRA implemented in this thesis. We create an algorithm that, in addition to identifying users who rate offset or inverse compared to each other, also attempts to identify users who rate on differently sized ranges of the scale. The relationships between users having either of these tree types of compatible taste (or any combination of them), are then considered when creating predictions. Consider the example in Figure 11.

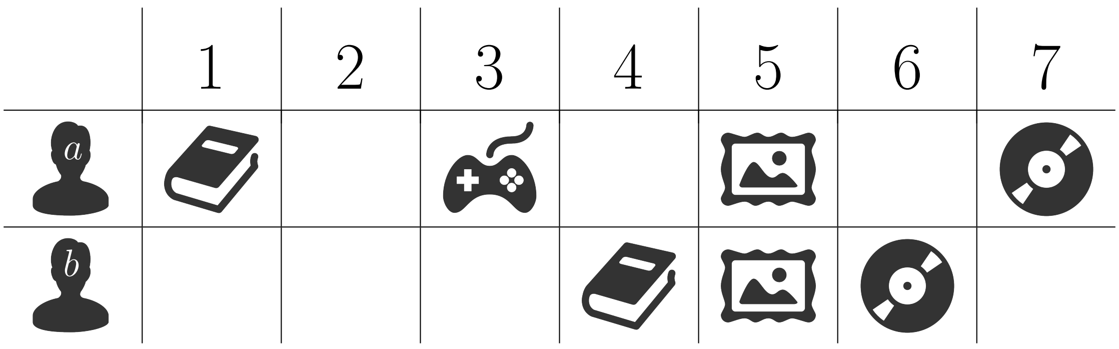


Figure 11: An example showing the ratings of two users such that a transformation function halving the size of the ratings scale gives an average difference equal zero. See Figure 3 in Section 2.1 for details on how to interpret the figure.

The items both users have rated are *game*, *picture*, and *cd*. Both users have given *picture* the rating 5, so the difference in rating for this item is 0. For both *game* and *cd*, the difference in rating is 1. For *game*, user b has given a higher rating than user a, while user b has rated *cd* lower compared to user a.

By using IRA's method for identifying users with compatible taste, we get that the transformation function $IRA\_t_{a \to b} : \mathbb{R} \to \mathbb{R}$ that minimizes the average difference $IRA\_t\_diff_{a \to b}$ between the transformed ratings by user a and the ratings by user b is $IRA\_t_{a \to b}\left(r_a\right) = r_a$. We get that the minimized $IRA\_t\_diff_{a \to b} = \frac{1+0+1}{3} = \frac{2}{3}$. This might seem like an ok average difference and transformation function. However, if we look closely at the figure, we see that user b's ratings look like a compressed version of user a's ratings. Can it be that user a is more effusive than user b, and hence uses a larger part of the scale? Our algorithm will identify this case as both users having the same opinion on the rating 5, while above and below 5, user a's rating scale is twice as large as user b's scale. In other words, the distance from 5 is twice as large for user a compared to user b's ratings. The final transformation function from user a to user b will be $t_{a \to b}\left(r_a\right) = \left(r_a - 3\right) \cdot \frac{1}{2} + 4$. Using this transformation function we get that the average difference $t\_diff_{a \to b}$ between the transformed ratings by user a and the ratings by user b equals 0.

If we want to predict which rating user b would give *book* based on user a's rating of *book*, IRA's transformation function would give us a predicted rating $IRA\_p_{a \to b, book} = IRA\_t_{a \to b}\left(r_{a,book}\right) = r_{a,book} = 1$, while our transformation function would give the prediction $p_{a \to b, book} = t_{a \to b}\left(r_{a,book}\right) = \left(r_{a,book} - 3\right) \cdot \frac{1}{2} + 4 = 3$. This is quite a large difference on a scale of size 7!

To find the transformation function that makes two users a and b as compatible as possible, our algorithm chooses values for the variables min_a, max_a, min_b, and max_b, giving the functions

$$\begin{aligned} t_{a \to b}\left(r_a\right) &= \begin{cases} \left(r_a - min_a\right) \cdot \frac{max_b - min_b}{max_a - min_a} + min_b & \text{if } min_a \neq max_a \text{ and } min_b \neq max_b \\ \left(r_a - min_a\right) + min_b & \text{if } min_a = max_a \text{ and } min_b = max_b \end{cases} \\ t_{b \to a}\left(r_b\right) &= \begin{cases} \left(r_b - min_b\right) \cdot \frac{max_a - min_a}{max_b - min_b} + min_a & \text{if } min_a \neq max_a \text{ and } min_b \neq max_b \\ \left(r_b - min_b\right) + min_a & \text{if } min_a = max_a \text{ and } min_b = max_b \end{cases} \end{aligned} \tag{3}$$

where $min_a, max_a, min_b, max_b \in S$ such that

$$t\_diff_{a \to b} = \max\left(\sum_{i \in I_a \cap I_b} \left|t_{a \to b}\left(r_{a,i}\right) - r_{b,i}\right|, \sum_{i \in I_a \cap I_b} \left|t_{b \to a}\left(r_{b,i}\right) - r_{a,i}\right|\right) \tag{4}$$

is minimized. Note that the functions are only defined when $max_a - min_a$ and $max_b - min_b$ are either both equal 0, or both different from 0. In other words, the functions are always invertible, and, in fact, $t\_diff_{a \to b}$ and $t\_diff_{b \to a}$ are the inverse of each other.

The rationale behind minimizing the maximum of the average difference between user b's ratings and the transformed ratings by user a and the average difference between user a's ratings and the transformed ratings by user b, instead of just minimizing the average difference between user b's ratings and the transformed ratings by user a, is to avoid accepting relations due to the size of the scale being decreased by the transformation function. When $|max_b - min_b| < |max_a - min_a|$, the difference between $t_{a \to b}\left(r_{a,i}\right)$ and $r_{b,i}$ will be smaller than the difference between $t_{b \to a}\left(r_{b,i}\right)$ and $r_{a,i}$, for $i \in I_a \cap I_b$. This is not necessarily due to $t_{a \to b}$ being more accurate than $t_{b \to a}$, but rather that $t_{a \to b}$ maps onto a smaller range than $t_{b \to a}$, and hence the differences are scaled down accordingly. A difference of 1 on a scale from 2 to 5 has the same relative size as a difference of 2 on a scale from 1 to 7, as $\frac{5-2}{1} = \frac{7-1}{2}$. Consider Figure 12, which illustrates this effect.

If we let $min_a = 1$, $max_a = 7$, $min_b = 4$, $max_b = 6$, and $i \in I_a \cap I_b$, we get that the average difference between $t_{a \to b}\left(r_{a,i}\right)$ and $r_{b,i}$ is $\frac{0+\frac{2}{3}+0}{3} = \frac{2}{9}$, while the average difference between $t_{b \to a}\left(r_{b,i}\right)$ and $r_{a,i}$ is $\frac{0+2+0}{3} = \frac{2}{3}$. Hence it seems that $t_{a \to b}$ is more accurate than $t_{b \to a}$. If we, instead of comparing the average differences as absolute values, compare them relative to the size of the scaled scales, we see that the average differences actually are equal. $\frac{\frac{2}{9}}{6-4} = \frac{1}{9}$, and $\frac{\frac{2}{3}}{7-1} = \frac{1}{9}$. In Scaling the

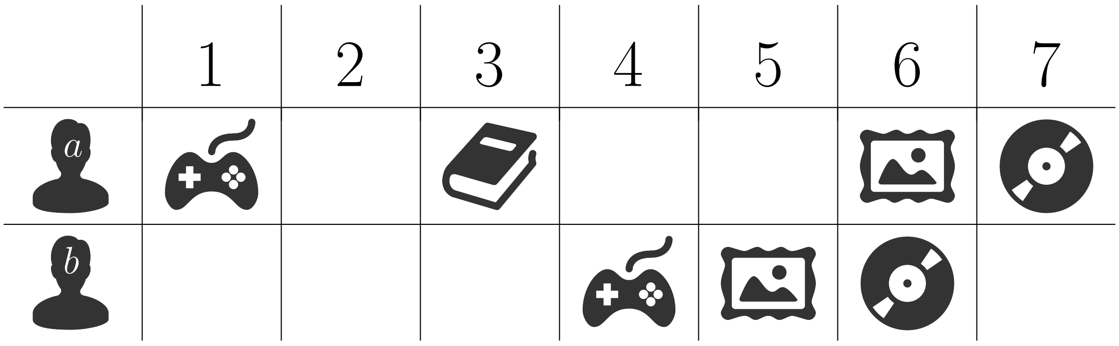


Figure 12: An example showing that decreasing the size of the rating scale results in decreased average difference. See Figure 3 in Section 2.1 for details on how to interpret the figure.

scales, we therefore choose to regard two users as having compatible taste only if both transformation functions yield small average differences[1]. Because scaling is not involved in IRA, an IRA transformation function will always give the same average difference as its inverse function. Therefore this issue of scaled average differences does not apply to IRA.

Rating on differently sized ranges of the scale is a generalization of both rating offset and rating inverse. Therefore this transformation function will identify users as having compatible taste when they are rating in any combination of offset or inverse, as IRA does, as well as when they are rating on differently sized ranges of the scale[2]. Here are some examples of how our transformation function covers any combination of scaled, offset, and inverse rating scales:

Offset If user a generally rates 2 values higher than user b, IRA would give the transformation function $IRA\_t_{a \to b}(r_a) = r_a - 2$. The transformation function

[1]We have also tried variants of our algorithms which do not have this constraint, and some of them are included in our implementation. See Section 4.3 for more details.

[2]There are actually two transformation functions that, in theory, can be used by IRA and not by Scaling the scales, namely $t(r) = -r + 2$ and $t(r) = -r + 2 \cdot s_{max}$. Using these functions, all ratings are transformed to ratings 1 (bottom of the scale) or lower and s_{max} (top of the scale) or higher, respectively. We do not regard it as reasonable that two users have compatible taste if such a function is the best transformation function between them. In addition, these functions will never be the only best transformation functions, as they will never give a smaller average difference than the function $t(r) = r$. Therefore we chose to not include these transformation functions in Scaling the scales. If we wanted to, the functions could have been included by for example introducing a separate $sign \in \{-1, 1\}$ variable in the Scaling the scales transformation functions.

given by our algorithm could[3] be $t_{a\to b}(r_a) = (r_a - 2)\cdot\frac{1-0}{3-2} + 0 = (r_a - 2)$.

Inverse If user a generally rates exactly opposite of user b on a scale of size 13 (so they agree on the midpoint of the scale), IRA would give the transformation function $IRA\_t_{a\to b}(r_a) = -r_a + 14$. The transformation function given by our algorithm could be $t_{a\to b}(r_a) = (r_a - 5)\cdot\frac{8-9}{6-5} + 9 = -r_a + 14$.

Offset + inverse If user a generally rates opposite of user b on a scale of size 13, but is also more negative and rates 1 lower than user b, IRA would give the transformation function $IRA\_t_{a\to b}(r_a) = -r_a + 15$. The transformation function given by our algorithm could be $t_{a\to b}(r_a) = (r_a - 7)\cdot\frac{7-8}{8-7} + 8 = -r_a + 15$.

Scaled By letting $\frac{max_b - min_b}{max_a - min_a} \neq \pm 1$, scaling can be added to any of the above cases.

Other than BEST-TRANS-FUNC, Scaling the scales can use the same algorithms as IRA when calculating predictions. So all algorithms described in Chapter 3 except for Algorithm 1 also apply for Scaling the scales. Algorithm 6 shows pseudocode for the Scaling the scales version of BEST-TRANS-FUNC. It's job is to find the Scaling the scales transformation function $t_{a\to b}$ that minimizes $t\_diff_{a\to b}$. This is done by exhaustive search—every feasible version of $t_{a\to b}$ is tried, and the best one is chosen. Feasible means that both users' rating scales are assumed to be within the valid rating scale: $1 \leq min_a, max_a, min_b, max_b \leq s_{max}$.

We have that

$$\begin{aligned}
t_{a\to b}(r_a) &= (r_a - min_a)\cdot\frac{max_b - min_b}{max_a - min_a} + min_b \\
&= (r_a - min_a + max_a - max_a)\cdot\frac{max_b - min_b}{max_a - min_a} + min_b \\
&= (r_a - max_a)\cdot\frac{max_b - min_b}{max_a - min_a} + (max_a - min_a)\cdot\frac{max_b - min_b}{max_a - min_a} + min_b \\
&= (r_a - max_a)\cdot\frac{max_b - min_b}{max_a - min_a} + max_b - min_b + min_b \\
&= (r_a - max_a)\cdot\frac{max_b - min_b}{max_a - min_a} + max_b \\
&= (r_a - max_a)\cdot\frac{min_b - max_b}{min_a - max_a} + max_b
\end{aligned}$$

Therefore, every instance where $max_a < min_a$ will be covered by an instance where

[3] In Scaling the scales, transformation functions with different parameters can behave identically. For example $t_{a\to b}(r_a) = (r_a - 2)\cdot\frac{1-0}{3-2} + 0 = (r_a - 3)\cdot\frac{2-1}{4-3} + 1$. Hence the exact values of the parameters in the transformation function cannot be uniquely determined in all cases. However, the behavior of the transformation function is uniquely determined in any case.

Algorithm 6 Algorithm for finding the best Scaling the scales transformation function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ for predicting ratings for user b, based on ratings by user a. See Section 2.2 for general clarifications for pseudocodes.

```
Input: Two users $a, b \in U$
Output: Two values:
    1. The Scaling the scales trans. function $t_{a \to b} : \mathbb{R} \to \mathbb{R}$ minimizing $t\_diff_{a \to b}$
    2. The minimized $t\_diff_{a \to b}$
1: function BEST-TRANS-FUNC($a, b$)
2:     $best\_min_a, best\_min_b \leftarrow 0$
3:     $best\_max_a, best\_max_b \leftarrow s_{max}$
4:     $best\_diff \leftarrow \infty$
5:     for all $min_a \in \{1, \dots, s_{max}\}$ do        ▷ All feasible transformation functions
6:         for all $max_a \in \{min_a, \dots, s_{max}\}$ do
7:             for all $min_b \in \{1, \dots, s_{max}\}$ do
8:                 for all $max_b \in \{1, \dots, s_{max}\}$ do
9:                     if ($max_a = min_a$ and $max_b = min_b$)
                       or ($max_a \neq min_a$ and $max_b \neq min_b$) then
10:                        $diff_{a \to b}, diff_{b \to a} \leftarrow 0$
11:                        $scale_{a \to b} \leftarrow (max_b - min_b)/(max_a - min_a)$
12:                        $scale_{b \to a} \leftarrow (max_a - min_a)/(max_b - min_b)$
13:                        for all $i \in I_a \cap I_b$ do        ▷ Calculate total difference
14:                            $diff_{a \to b} \leftarrow diff_{a \to b} + |(r_{a,i} - min_a) \cdot scale_{a \to b} + min_b - r_{b,i}|$
15:                            $diff_{b \to a} \leftarrow diff_{b \to a} + |(r_{b,i} - min_b) \cdot scale_{b \to a} + min_a - r_{a,i}|$
16:                        end for
17:                        if MAX($diff_{a \to b}, diff_{b \to a}$) $< best\_diff$ then
18:                            $best\_min_a \leftarrow min_a$        ▷ New best trans. function
19:                            $best\_max_a \leftarrow max_a$
20:                            $best\_min_b \leftarrow min_b$
21:                            $best\_max_b \leftarrow max_b$
22:                            $best\_diff \leftarrow$ MAX($diff_{a \to b}, diff_{b \to a}$)
23:                        end if
24:                    end if
25:                end for
26:            end for
27:        end for
28:    end for
29:    $avg\_diff \leftarrow best\_diff \,/\, |I_a \cap I_b|$        ▷ Calculate average difference
30:    $t \leftarrow$ TRANS-FUNC-SCALING($best\_min_a, best\_max_a, best\_min_b, best\_max_b$)
31:    return $t, avg\_diff$
32: end function
```

$min_a < max_a$, and we only have to consider the cases where $min_a \leq max_a$, as we have done in Algorithm 6.

The time complexity of this algorithm is $O\left(|S|^4 \cdot n\right)$, where $n = |I_a \cap I_b|$. Because the rating scale and the number of items both users have rated generally both are fairly small, this algorithm runs quite fast. There are ways to improve the running time compared to the quite straightforward algorithm presented in Algorithm 6, and some such optimizations have been used in our implementation. It runs, however, considerably slower than IRA's algorithm for finding the best transformation function between two users.

4.2 Offline vs. real-time computations

As discussed in Section 1.5, it is important that the process of calculating predictions (called the *predicting process*) is very fast. The process of finding the best transformation function between two users (called the *relating process*) does not necessarily have to be invoked when calculating predictions, and hence its running time might not be that crucial. If we assume that the dataset is static, and does not change, the relating process only has to be run once for each pair of users. After this, the predicting process can calculate predictions by simply searching in the prediction graph constructed by the relating process. If changes are made to the dataset (for example when users rate items, or when new users or items are added), however, the predicting process will not take these changes into account before the relating process has been run again to update the prediction graph.

The highest prediction quality would be obtained by running the relating process whenever changes are made to the dataset, in order to always keep the prediction graph up to date. This might come with an unsatisfactory speed penalty. To increase the speed of the system, it might be acceptable to use slightly stale data when calculating predictions. For example, the relating process can be run offline, updating the prediction graph at regular intervals, while the predicting process runs in real-time, using the most recent data available for calculating predictions. This makes for a very fast system, as the predicting process never has to wait for the relating process. The prediction quality, on the other hand, might suffer, since the data used when calculating predictions is not always up to date. Another approach, which is implemented by Aggarwal et al. [1], gives a tradeoff between speed and prediction quality. In addition to running the relating process offline at regular intervals, this approach makes sure that the relations concerning the user for which the prediction should be calculated are up to date before the predicting process is started. The data is still not always totally up to date, but at least the relations where the user in question is the predicted user are up to date. This should give higher prediction quality, with only a small time penalty. An in-depth explanation of this approach can be found in the article by Aggarwal et al. [1].

4.3 Implementation details

> *"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."*
>
> — C.A.R. Hoare, winner of the 1980 Turing Award

Even though it would be great, we do not consider our software design so simple that the absence of deficiencies is obvious. Nor do we consider our implementation so simple that it is easy to see exactly how it works without any extra explanation besides the source code. This section will therefore give a short introduction to the design of our recommender system. For a further study of our implementation, we refer to the source code itself, which is available at GitHub [23], as well as BORA [3], where this thesis is published.

When implementing our recommender system, we wanted the development and deployment of different extensions and approaches to be easy, without having to rewrite big parts of the system. The recommender system should also be able to handle different datasets with ease. Therefore we chose to let the recommender system consist of four components: `Core`, `Parser`, `Relator`, and `Predictor`.

- The `Core` stays the same for all the recommender system approaches, and takes care of the communication between the other components.
- The `Parser` is different for every dataset. Its job is to know the format of the dataset on which the recommender system should be run, and translate it into something that can be understood by the other components.
- The `Relator`'s responsibility is to discover users with compatible taste, and is different for every recommendation algorithm.
- The `Predictor` calculates predictions. This is done based on the prediction relations calculated by the `Relator`. By providing the `Predictor` with a few parameters, such as the depth limit *max_path_length* for the breadth-first search, we can use the same `Predictor` for all the different recommendation

algorithms we have implemented. It can, however, be changed to accommodate recommender systems behaving differently from the algorithms we have tested in this thesis.

Our recommender system works by the `Core` first asking the `Parser` to parse the dataset. The `Parser` constructs objects for each user, item and rating in the dataset. When the parsing is done, the `Relator` is asked to identify all pairs of users such that one of the users (called the *predicting* user) predicts the other (called the *predicted* user). This is done by looking at the ratings given to the items which both users have rated, as discussed in Sections 3.1 and 4.1. For each such pair of users, a relation object is created, containing information on the transformation function used to transform ratings by the predicting user to ratings for the predicted user. Each user has a list of all relations in which they are the predicted user. The `Predictor` calculates predictions by searching in the graph consisting of user vertices and directed relation arcs from predicted users to predicting users.

Every `Parser` must implement the same interface in order to be used by our recommender system. We have implemented `Parser`s for the four different datasets we will use in Section 5.2. New `Parser`s can easily be created for using the recommender system on other datasets.

In our implementation, the abstract class `CollaborativeFilter` contains most of the functionality needed by the `Relator` and the `Predictor`. We have included four different classes extending `CollaborativeFilter` in our implementation. These classes give the functionality of the recommendation algorithms IRA [1], Scaling the scales, Firefly [33] and LikeMinds [14]. As mentioned in Section 3, Firefly and LikeMinds were used when comparing the performance of IRA to other recommender systems in the article by Aggarwal et al. [1]. Because we focus on graph-based collaborative recommender systems, Firefly and LikeMinds are not discussed in this thesis. But to show that our software design allows for recommendation algorithms taking other approaches than IRA and Scaling the scales, we have included the implementations of Firefly and LikeMinds.

To tailor how the recommender system should behave, the `Predictor` part of `CollaborativeFilter` takes as argument an instance of a class describing how the search in the graph should be performed. We have only implemented one such class, as the same one could be used for all four of the `CollaborativeFilter`s implemented. Similarly, the `Relator` part of `CollaborativeFilter` takes as argument an instance of a class called `Transformer`, whose task is to create the functions for transforming ratings by one user into predicted ratings for another. We have made ten different classes implementing `Transformer`. Four of them provide the functionality used by IRA, Scaling the scales, Firefly and LikeMinds (items 1, 3, 9, and 10 in the following list), while one (item 2) provide alternative running times for IRA. The remaining five classes provide slightly different approaches to scaled scales than the one presented in Section 4.1. These approaches were implemented in order to pick the algorithm providing the best prediction quality for use in Scaling the scales. Brief descriptions of the different classes extending `Transformer` follow.

1. `InverseOffsetMinimized` provides an implementation of the function for identifying users with compatible taste used by IRA [1]. This is an implementation of the code shown in Algorithm 1, which is an exhaustive search algorithm with time complexity $O(|S| \cdot n)$, where $n = |I_a \cap I_b|$.

2. `InverseOffsetAnalyzed` provides a different implementation of the function for identifying users with compatible taste used by IRA. This algorithm finds the optimal *offset* by looking at the median of the differences. In our implementation, the algorithm for finding the median can be chosen from three alternatives, giving different running times for the function:

   (a) By sorting, in time $O(n \cdot \log n)$, where $n = |I_a \cap I_b|$.

   (b) By a random selection algorithm in expected time $O(n)$, where $n = |I_a \cap I_b|$.

   (c) By a deterministic selection algorithm [5] in guaranteed time $O(n)$, where $n = |I_a \cap I_b|$.

   In our experience, the sorting algorithm usually is the fastest one in practice.

3. `InverseScaledInvertible` provides an implementation of the function for identifying users with compatible taste used by Scaling the scales. This is an implementation of the code shown in Algorithm 6, but with a few optimizations to make it run faster. It is an exhaustive search algorithm, and has time complexity $O\left(|S|^4 \cdot n\right)$, where $n = |I_a \cap I_b|$.

4. `ScaledInvertible` provides an algorithm similar to number 3. The main difference is that this algorithm does not identify users rating opposite of each other as having compatible taste. In our experiments, this algorithm gave lower prediction quality than `InverseScaledInvertible`, which we chose to use in Scaling the scales.

5. `InverseScaled` provides an algorithm similar to number 3. The main difference is that this algorithm does not consider the average difference of the inverse transformation function when identifying users with compatible taste,

as discussed in Section 4.1. Instead, user a and user b have similar taste if there is a transformation function with small average difference between the transformed ratings by user a and the ratings by user b, regardless of there being a transformation function with small average difference in the opposite direction. In our experiments, this algorithm gave lower prediction quality than `InverseScaledInvertible`, which we chose to use in Scaling the scales.

6. `Scaled` provides an algorithm similar to number 5. The main difference is that this algorithm does not identify users rating opposite of each other as having compatible taste. In our experiments, this algorithm gave lower prediction quality than `InverseScaledInvertible`, which we chose to use in Scaling the scales.

7. `InverseScaledTwoWay` provides an algorithm similar to number 3. The main difference is that this algorithm does not require the transformation functions $t_{a \to b}$ and $t_{b \to a}$ to be inverse of each other. For user a and user b to have compatible taste, the algorithm still requires that both $t_{a \to b}$ and $t_{b \to a}$ give small average differences. But the functions can be any valid Scaling the scales transformation functions, and not necessarily the inverse of each other, as in number 3. In our experiments, this algorithm gave lower prediction quality than `InverseScaledInvertible`, which we chose to use in Scaling the scales.

8. `ScaledTwoWay` provides an algorithm similar to number 7. The main difference is that this algorithm does not identify users rating opposite of each other as having compatible taste. In our experiments, this algorithm gave lower prediction quality than `InverseScaledInvertible`, which we chose to use in Scaling the scales.

9. `ConstrainedPearsonR` provides an implementation of the function for identifying users with compatible taste used by Firefly [33]. This algorithm has time complexity $O(n)$, where $n = |I_a \cap I_b|$.

10. `AgreementScalar` provides an implementation of the function for identifying users with compatible taste used by LikeMinds [14]. This algorithm has time complexity $O(n)$, where $n = |I_a \cap I_b|$.

In addition to the recommender system itself, we have included a system for generating synthetic datasets, as well as a system for computing statistics for datasets, relations and predictions in our implementation.

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

— Bill Gates, co-founder of Microsoft

The number of lines of code in a project is seldom a useful measure, and is usually not a good indicator for neither quality nor progress. It can, however, be a fun measure to look at when comparing projects, and can also provide some insight in regard to the amount of work put into a program. Our implementation contains more than 6,000 lines of code, and uses around 50 different classes and interfaces. If we were to include the source code in an appendix to this thesis, the appendix would be more than 150 pages long, or about twice as long as the thesis itself!

Chapter 5

Experimental results

Some people argue that the quality of a recommender system cannot be measured, due to there being too many objective functions. There are, however, many methods attempting to calculate such quality measures. Most traditional methods rely on performing offline experiments using an existing dataset. There are several limitations to such offline evaluation methods, as there is no way to know how users would react to the recommendations. Because of this, a better method might be to actually deploy several recommender systems on a real live system for doing performance comparison, as this allows for measuring true user satisfaction [32]. Many companies use this approach for identifying the recommender systems that gives the best performance for their particular applications. The comparison can for example be done by dividing the user base into several groups, where each group of users is served recommendations from different recommender systems. The results for how the different user groups respond to the recommendations can then be used for comparing the different recommender systems, and for tweaking them for optimal performance.

For this thesis, we do not have access to a live system on which we can test different recommender systems, and hence we must resort to using an offline approach for performance comparison. We will use a technique called *leave-one-out cross-validation* for comparing the performance of IRA and Scaling the scales. This method is based on comparison between predicted ratings, calculated based on a dataset, and the real ratings in the dataset. For more information on how recommender systems can be evaluated, the article by Herlocker et al. [17] is the authority in the field. Another interesting article is written by Olmo and Gaudioso [24], who present a new evaluation approach that takes the interactivity of the recommender system into account.

In the article by Aggarwal et al. [1], IRA is only tested on synthetic data, but the authors state that the algorithm "will be tested in a real user trial in the near future". We have contacted the authors, and received replies from Wolf and Wu. They replied that, presumably due to lack of real data, such a trial was never deployed. We have not found in the literature any published work testing the IRA algorithm on real datasets. Hence this thesis might present the first published results on how IRA performs on real data. Many organizations release datasets based on real data for use in research on recommendation algorithms. We will run experiments on datasets provided by GroupLens Research and the 2$^{\text{nd}}$ International Workshop on Information Heterogeneity and Fusion in Recommender Systems.

In addition to using real datasets, we would like to compare the performance of Scaling the scales and IRA on similar datasets as were used in the article by Aggarwal et al. [1]. The synthetic datasets used in the article were crafted towards the features of the IRA algorithm. We want to test how Scaling the scales performs on such datasets, and also how the IRA algorithm performs on datasets crafted towards Scaling the scales. We start by describing how we have created the synthetic datasets used in this thesis.

5.1 Dataset generation

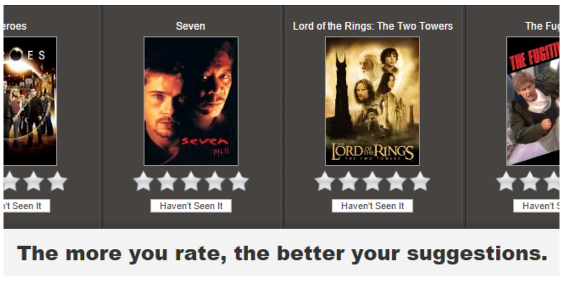

Seven
Lord of the Rings: The Two Towers
Haven't Seen It
Haven't Seen It
The more you rate, the better your suggestions.

Figure 13: Netflix asking their users to rate movies, in order to improve the recommendations.

When generating a dataset, we first choose the values of the rating scale $S = \{1, \ldots, s_{max}\} \subsetneq \mathbb{Z}^+$. Secondly, we specify the sizes of the set of users U and the set of items I in our dataset. Every user has a chance $uc \leq 0.5$ of being a contrarian, and rate opposite of normal users. Let $U_{contra} \subseteq U$ be the set of users chosen to be contrarians. The items are divided into two parts; one small set of items $I_{hot} \subseteq I$ called the "hot set", and the rest of the items $I_{cold} \subseteq I$, called the "cold set". Users are more likely to have rated the items in the hot set than the items in the cold set. The idea behind this is that the hot set will increase commonality between the users (users have rated many of the same items), while the cold set increases the coverage of the items rated (all items have been at least a few times) [1]. Having a hot set of items which users are likely to have rated increases the probability that two users have rated many of the same items, and hence we can identify their taste compatibility. It is not unrealistic that such hot sets can exist in many applications. This is both due to some items being generally more popular than others, but also because a recommender system can "push" users towards rating specific items. As seen in Figure 13, the movie website Netflix, for example, ask their users whether they have seen specific movies, and if so, how they would rate them. This would be a great opportunity to ask users to rate items in the hot set. But it is equally important that the users rate the items in the cold set. If very few users have rated an item, it might not be possible to find a path from a user we want to predict a rating for to any of the few users having rated the item. So users could also be asked to rate items in the cold set, in order to ensure that the system will be able to calculate predictions for those items.

Next we choose how many items ri the users have rated on average, as well as how many of these are items from the hot set ri_{hot} and from the cold set ri_{cold}. We can now determine the probability $pr_{a,i}$ for user a to have rated item i:

$$pr_{a,i} = \begin{cases} \frac{ri_{hot}}{|I_{hot}|} & \text{if } i \in I_{hot} \\ \frac{ri_{cold}}{|I_{cold}|} & \text{if } i \in I_{cold} \end{cases} \tag{5}$$

Every item i is now assigned an average rating m_i, chosen randomly from a uniform distribution over the rating scale. Specific user ratings will ensure that the average of all ratings given to item i becomes more or less m_i, as described in Sections 5.1.1 and 5.1.2.

The next step is to determine the specific ratings the various users have given. So far, the generation process has been identical for all our synthetic datasets. But when determining the specific user ratings the processes are different, depending on whether the dataset should be crafted towards the IRA algorithm, or towards the features of Scaling the scales. This is because the two algorithms exploit different aspects of how users behave, and hence, in order to bring out these differences, the datasets should be based on different models for user behavior.

5.1.1 Ratings crafted towards IRA

The following describes the datasets that were used to test the IRA algorithm by Aggarwal et al. [1]. The IRA algorithm assumes that users can have compatible taste while rating offset and/or inverse compared to each other. To create a dataset with users matching this assumption, we first assign every user a an offset addend o_a randomly chosen from a normal distribution with mean 0. This offset addend represents how positive or negative the user is when giving a rating, compared to the average user.

When creating the dataset, user a will have given item i a rating with probability $pr_{a,i}$. The average rating given to item i has already been specified, and should have the value m_i. Given that user a is chosen to give a rating to item i, this rating $r_{a,i}$ is calculated as follows. First a random rating rr_i for item i is chosen from a normal distribution with mean m_i. Then user a's offset addend o_a is added to the random rating, and if $a \in U_{contra}$ the rating is reversed. Finally, the rating is constrained to lie within the rating scale. In formulas this yields

$$r_{a,i} = \begin{cases} \max(1,\, \min(s_{max},\, rr_i + o_a)) & \text{if } a \notin U_{contra} \\ \max(1,\, \min(s_{max},\, s_{max} + 1 - (rr_i + o_a))) & \text{if } a \in U_{contra} \end{cases} \tag{6}$$

5.1.2 Ratings crafted towards Scaling the scales

In addition to letting users rate offset and/or inverse compared to each other, Scaling the scales also identifies users rating on differently sized ranges of the scale as compatible. To create a dataset with users matching this assumption, every user a is assigned a value s_a, representing the size of the scale the user typically uses when rating. s_a is chosen randomly from a uniform distribution over the set $\{0, \ldots, s_{max} - 1\}$. Next, the user is assigned a value $s_{from,a}$, representing where the scale the user typically uses when rating is located on the full rating scale. $s_{from,a}$ is chosen randomly from a uniform distribution over the set $\{1, \ldots, s_{max} - s_a\}$. We let $s_{to,a} = s_{from,a} + s_a$. The values on the rating scale between $s_{from,a}$ and $s_{to,a}$ represent the part of the scale that the user typically uses when rating. If $a \in U_{contra}$, we swap $s_{from,a}$ and $s_{to,a}$, so that $s_{to,a} \leq s_{from,a}$.

When creating the dataset, user a will have given item i a rating with probability $pr_{a,i}$. The average rating given to item i should be m_i. Given that user a is chosen to give a rating to item i, this rating $r_{a,i}$ is calculated as follows. First a random rating rr_i for item i is chosen from a normal distribution with mean m_i. Then the random rating is multiplied by the relative size of the scale that user a typically uses when rating. If user a is a contrarian, the rating will now automatically have been reversed, as the relative size of the scale is negative. Now $s_{from,a}$ is added to the rating, before the rating is constrained to lie within the rating scale. As a formula this can be expressed as

$$r_{a,i} = \max\left(1,\, \min\left(s_{max},\, s_{from,a} + (rr_i - 1) \cdot \frac{s_{to,a} - s_{from,a}}{s_{max} - 1}\right)\right) \tag{7}$$

5.2 Performance comparison

As mentioned in the introduction to this chapter, there are many different ways to evaluate recommender systems. In this thesis, we will compare how well IRA and Scaling the scales fare when giving predictions using an approach called *leave-one-out cross-validation*. This method works by hiding a known rating, predicting what that rating should be, and comparing the correct and predicted ratings.

Algorithms 7 and 8 show pseudocode for the leave-one-out algorithm. Algorithm 7 first checks whether the user in question has rated the item in question. If this is not the case, there is no rating to hide, and the prediction is calculated directly by calling the basic prediction function in Algorithm 3. If the user has rated the item, the rating must be hidden. When a rating is hidden, the prediction relations concerning the user having given the rating might change. For example, the user might not have enough rated items in common with other users any longer, or the transformation function giving the smallest average difference might be another when the hidden rating is not considered. Therefore the prediction relations must

Algorithm 7 Algorithm for predicting which rating a specific user would give to a specific item. If the user has already rated the item, the rating is hidden and the prediction calculated as if the user had not rated the item. See Section 2.2 for general clarifications for pseudocodes.

```
Input: A user $a \in U$, and an item $i \in I$
Output: Two values:
    1. The original rating, null if no original rating
    2. The predicted rating, null if no rating could be predicted
 1: function LEAVE-ONE-OUT($a, i$)
 2:     if $i \notin I_a$ then        ▷ $a$ has not rated $i$, calculate the prediction and return
 3:         return null, PREDICT($a, i$)
 4:     end if
 5:
 6:     $I_a \leftarrow I_a \setminus \{i\}$                ▷ Hide $i$ from the items $a$ has rated
 7:     RE-RELATE($a$)          ▷ Recalculate prediction relations to and from $a$
 8:     $p \leftarrow$ PREDICT($a, i$)
 9:     $I_a \leftarrow I_a \cup \{i\}$                ▷ Unhide $i$ in the items $a$ has rated
10:     RE-RELATE($a$)          ▷ Recalculate prediction relations to and from $a$
11:
12:     return $r_{a,i}, p$
13: end function
```

Algorithm 8 Algorithm for recalculating the prediction relations to and from a user. See Section 2.2 for general clarifications for pseudocodes.

```
Input: A user $a \in U$
1: function RE-RELATE($a$)
2:     for all $u \in U$ do
3:         $t_{a,u} \leftarrow$ RELATE($a, u$)      ▷ Update transformation function from user $a$
4:         $t_{u,a} \leftarrow$ RELATE($u, a$)      ▷ Update transformation function to user $a$
5:                                                ▷ $P_u$ and $P_a$ are implicitly updated
6:     end for
7: end function
```

be recalculated. This is done by calling the function in Algorithm 8, before the prediction can be calculated by calling the basic prediction function in Algorithm 3. Finally, the rating must be unhidden, and the prediction relations recalculated again. Both the original rating and the predicted rating are returned, so they can be compared and used when calculating statistics.

Algorithm 8 simply goes through every user u, and calls the function in Algorithm 2 to recalculate the prediction relation from the user in question to u and from u to the user in question. The callable transformation functions returned by the function in Algorithm 8 are stored globally, so they can be called by other functions. Note that, as prediction relations are removed and introduced, the sets P_a, where $a \in U$, are implicitly updated, as they always contain the set of current predictors.

When evaluating the recommender systems on different datasets, we collect statistics on the prediction results. The different statistics we collect are:

Success rate How many of the attempts to predict a rating were successful?

Success rate extremal This statistic is the same as the previous one, except that only the predictions where the real rating is an extremal value on the rating scale are considered. As mentioned in the works by Shardanand and Maes [33], these predictions are often the most important, as users are generally more interested in what they will love or hate, than in items they might perceive as average. The rating values considered to be extremal will be specified for each dataset.

Error The average difference between the predicted rating and the real rating.

Error extremal This statistic is the same as the previous one, except that only the predictions where the real rating is an extremal value on the rating scale are considered.

Prediction sources The average number of different users having rated the item in question that were found by the breadth-first search, and hence used as sources for the prediction.

Paths The average number of paths used to calculate the prediction. This is the same as the average number of shortest paths from the predicted user to a user having rated the item in question.

Path length The average distance from the predicted user to a user having rated the item in question in the prediction graph. This will never be larger than max_path_length.

Distinct edges On average, how many of the edges in the paths used to calculate the predictions are distinct? On average, there are **Paths** · **Path length** edges used to calculate the predictions. The same edge can be used in several different paths, and hence it is interesting to know how many distinct edges are used.

Common ratings The average of $I_a \cap I_b$ for all edges $a \rightarrow b$ in all paths used to calculate the final prediction. An edge will count multiple times towards this statistic if it appears in multiple prediction paths. To our understanding, this measure is the same as the *average intersection cardinality of directed path arcs* used by Aggarwal et al. [1].

Trans func diff The average of $t\_diff_{a \rightarrow b}$ for all edges $a \rightarrow b$ in all paths used to calculate the final prediction. This will never be larger than t_diff_{max}. An edge will count multiple times towards this statistic if it appears in multiple prediction paths. To our understanding, this measure is the same as the *average manhattan segmental distance of directed path arcs* used by Aggarwal et al. [1].

Relations The number of prediction relations identified by the recommender system. In other words, the number of user pairs such that one user predicts the other.

Inverse relations How many of the **Relations** are inverse relations, meaning that the users in the relation rate opposite.

Scaled relations How many of the **Relations** are scaled relations. This only applies to Scaling the scales. A relation $a \rightarrow b$ is scaled if $\frac{max_b - min_b}{max_a - min_a} \neq \pm 1$ in the transformation function $t_{a \rightarrow b} : \mathbb{R} \rightarrow \mathbb{R}$ giving the smallest average difference $t\_diff_{a \rightarrow b}$.

Graph density The graph density for the prediction graph, calculated as $\frac{|E|}{|U| \cdot (|U| - 1)}$, where $|U|$ is the number of users, and hence equals the number of vertices in the prediction graph, while $|E|$ is the number of **Relations** identified by the recommendation algorithm, and hence equals the number of edges in the prediction graph. This measure is always in the range $[0, \ldots, 1]$. A graph density close to 0 means that the graph is sparse (a graph with density 0 has no edges), while a graph density close to 1 means that the graph is dense (a graph with density 1 is a complete graph).

5.2.1 Dataset crafted towards IRA

For this test, we created our dataset as close to the dataset with 10,000 users described by Aggarwal et al. [1] as we were able to. Ideally, our implementation of the IRA algorithm should produce the same results on this dataset as those presented by Aggarwal et al. [1]. Unfortunately we were not able to achieve this. In particular we get more and shorter prediction paths, meaning that our prediction graph is denser than the one obtained by Aggarwal et al. [1]. If we change the parameters to make the prediction graph more sparse, the success rate is drastically decreased. It seems there must be a difference either between Aggarwal et al.'s implementation of the algorithm and our implementation, between the dataset used in the test published in the article and the dataset we have used, or a difference in how the statistics have been computed. There can be several explanations for this, as there are both details about the algorithm, dataset, and statistical analysis that are not transparent in the article by Aggarwal et al. [1]. Here are the explanations that, in our mind, are the most plausible. Some of them are formulated as questions, as we do not know the correct answers.

- Is the final predicted rating calculated by Aggarwal et al. [1] the average of the transformed ratings, or a weighted average? In our algorithm we use the normal average, as seen in Algorithm 3.
- Do Aggarwal et al. recalculate the prediction relations when a rating is hidden? In our algorithm we recalculate the prediction relations when a rating is hidden, as seen in Algorithm 7.
- In the datasets generated by Aggarwal et al., have the users rated ri items on average, or at least ri items? We assume that the users have rated ri items on average when we generate datasets, and have no lower limit on the number of items a user has rated.
- Details about the datasets, like how many users are contrarians and the standard deviations of the normal distributions, are not specified by Aggarwal et al.

- When presenting their results, Aggarwal et al. state that “Only experiments in which the algorithms were able to compute a rating were considered”. This statement is ambiguous, and can be interpreted as “only experiments in which all algorithms were able to compute a rating were considered”, or as “only experiments in which at least one algorithm were able to compute a rating were considered”. In addition, it is unclear whether the statement applies to all or only a few of the statistics presented in the article.

Although our results differ from what was obtained by Aggarwal et al. [1], it is still interesting to compare the performance of our implementation of IRA to the performance of our implementation of Scaling the scales. The question if scaling the rating scales has a positive effect on the prediction quality, can in any case be tested, regardless of whether Aggarwal et al. [1] used slightly different assumptions in their experiments than we have done in ours.

The dataset used in these experiments were generated as described in Sections 5.1 and 5.1.1, using the following parameters.

- The rating scale $S = \{1, \ldots, 13\}$, and $s_{max} = 13$
- The number of users $|U| = 10{,}000$
- The number of items $|I| = 5{,}000$, $|I_{hot}| = 50$, and $|I_{cold}| = 4{,}950$
- The average number of items rated $ri = 20$, $ri_{hot} = 10$, and $ri_{hot} = 10$
- The chance of a user being a contrarian $uc = 0.01$

This produced a dataset crafted towards the features of the IRA algorithm with the following properties.

Rating scale $\{1, \ldots, 13\} = S$

Number of items $5{,}000 = |I|$

Number of users $10{,}000 = |U|$

Number of ratings $201{,}120 = \sum_{u \in U} |I_u|$

In order to compare the prediction quality of IRA and Scaling the scales, we performed several experiments on this dataset, each using different parameters for the recommendation algorithms. 1,058 user-item pairs for which a rating existed in the dataset were randomly chosen. In every experiment, predictions were calculated for each of these pairs, using leave-one-out cross-validation. 500 of the user-item pairs involved extremal values, which were regarded as the ratings of value 1, 2, 3, 11, 12, and 13.

Comparative results

Consider Table 1, which shows statistics for two of the experiments run on this dataset. The first experiment uses parameters *hort_common* = 4, *hort_frac* = 1, and $t\_diff_{max} = 0.2$, and shows the best results we were able to obtain when minimizing the average error, while keeping a success rate of at least 90 %. In the second experiment, we tightened the constrains for horting, to see how this would effect the performance of the algorithms. We see that Scaling the scales is able to identify considerably more prediction relations than IRA. Note that Scaling the scales will always identify at least as many prediction relations as IRA. This is because the transformation functions identified by Scaling the scales is a superset of the transformation functions IRA can utilize, as discussed in Section 4.1. In the two experiments, we see an increase in prediction relations identified by Scaling the scales compared to IRA of 117 % and 42 %, respectively. This leads to Scaling the scales being able to compute predictions in many cases where IRA is unsuccessful, and particularly so in the second experiment, where the prediction graphs are more sparse. Scaling the scales also generally finds many more prediction sources and prediction paths to base the prediction calculations on. Scaling the scales gives

	$hort\_common = 4$		$hort\_common = 7$	
	$hort\_frac = 1$		$hort\_frac = 1$	
	$t\_diff_{max} = 0.2$		$t\_diff_{max} = 0.2$	
Algorithm	Scaling	IRA	Scaling	IRA
Success rate	99 %	99 %	42 %	36 %
Success rate extremal	99 %	98 %	43 %	36 %
Error	0.38	0.35	0.56	0.49
Error extremal	0.32	0.28	0.44	0.41
Prediction sources	30.21	16.69	3.25	2.66
Paths	65.32	28.09	4.23	3.17
Path length	1.36	1.45	2.61	2.80
Distinct edges	94 %	95 %	86 %	87 %
Common ratings	4.36	4.58	7.16	7.16
Trans func error	0.09	0.08	0.13	0.12
Relations	2,217,694	1,020,140	19,218	13,516
Inverse relations	2 %	2 %	2 %	2 %
Scaled relations	57 %	0 %	33 %	0 %
Graph density	0.02	0.01	0.0002	0.0001

Table 1: Performance comparison of IRA and Scaling the scales when creating predictions based on a dataset crafted towards the features of IRA.

Scaling the scales

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	87	12	2	0	0	0	0	0	0	0	0	0	0
2	1	70	25	1	1	0	0	0	0	0	0	0	0
3	1	15	68	13	2	0	0	0	0	0	0	0	0
4	1	0	17	67	14	1	0	0	0	0	0	0	0
5	0	0	0	16	72	9	2	0	0	0	0	0	0
6	0	0	0	1	19	67	11	0	0	0	1	0	0
7	0	0	0	0	2	21	70	7	0	0	0	0	0
8	0	0	0	0	0	1	18	61	18	1	0	0	0
9	0	0	0	0	0	2	5	8	60	26	0	0	0
10	0	0	0	0	0	0	0	0	24	60	16	0	0
11	0	0	0	0	0	0	0	0	2	24	59	12	4
12	0	0	0	0	0	0	0	0	0	5	10	69	15
13	2	0	0	0	0	0	0	0	0	0	0	14	84

IRA

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	90	8	2	1	0	0	0	0	0	0	0	0	0
2	3	70	25	1	0	0	0	0	0	0	0	0	0
3	0	10	74	15	0	1	0	0	0	0	0	0	0
4	1	0	11	69	18	1	0	0	0	0	0	0	0
5	0	0	0	11	67	18	2	1	1	0	0	0	0
6	0	0	0	0	16	65	19	0	0	0	0	0	0
7	0	0	0	0	0	19	73	8	0	0	0	0	0
8	0	0	0	0	0	0	15	61	24	0	0	0	0
9	0	0	0	0	0	0	2	18	66	15	0	0	0
10	0	0	0	0	0	0	0	2	22	64	13	0	0
11	0	0	0	0	0	0	0	0	2	27	61	10	0
12	0	0	0	0	0	0	0	0	0	3	23	64	10
13	0	0	0	0	0	0	0	0	0	0	0	18	82

Table 2: Prediction accuracy for Scaling the scales (top) and IRA (bottom) when using parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 0.2$ on a dataset crafted towards the features of IRA. Rows correspond to actual ratings, while columns correspond to predicted ratings.

slightly lower prediction quality than IRA, which is expected when the dataset is crafted to fit the model IRA is based upon. Relative to the total number of relations identified, both algorithms identify equally many inverse relations. The number of inverse relations is very low, which is expected as the probability of a user being a contrarian was set to 1 % when generating the dataset. In both experiments, a quite high number of the relations are identified to be scaled by Scaling the scales, considering that the dataset is generated without any users using scaled rating scales. This is probably due to the randomness of the normal distribution the ratings are generated around, in addition to ratings generated outside of the rating scale, and hence moved to the top or bottom valid rating value. This can lead to the ratings appearing to be on a scaled scale, which is identified by Scaling the scales.

Consider now Table 2. The table shows the relationships between real ratings and predicted ratings for the experiment using parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 0.2$. The rows correspond to real ratings, while the columns correspond to predicted ratings. Each cell (i, j) shows the percentage of the real ratings of value i that were predicted to be of value j. For example, for each real rating of value 2 that is predicted to be a rating of value 3, cell $(2, 3)$ in the table is increased. Since each cell shows a percentage relative to the number of real ratings for that row, each row sums to approximately 100. Obviously, the diagonal matrix with 100 in every cell along the diagonal, would be the best result. We see that IRA generally performs slightly better than Scaling the scales, but both algorithms perform very well, with the average value along the diagonal being close to 70. This means that the algorithms predicted the correct rating in about 70 % of the cases. Scaling the scales give slightly larger errors than IRA. While IRA generally predicts to the three or four columns closest to the diagonal, Scaling the scales generally predicts to the four or five columns closest to the diagonal. Scaling the scales also has some outliers, in particular 2 % of the ratings that actually were of value 13 were predicted to be of value 1!

5.2.2 Dataset crafted towards Scaling the scales

The dataset used in these experiments were generated as described in Sections 5.1 and 5.1.2, using the following parameters.

- The rating scale $S = \{1, \ldots, 13\}$, and $s_{max} = 13$
- The number of users $|U| = 10{,}000$
- The number of items $|I| = 5{,}000$, $|I_{hot}| = 50$, and $|I_{cold}| = 4{,}950$

- The average number of items rated $ri = 20$, $ri_{hot} = 10$, and $ri_{hot} = 10$
- The chance of a user being a contrarian $uc = 0.01$

This produced a dataset crafted towards the features of the Scaling the scales algorithm with the following properties.

Rating scale $\{1, \ldots, 13\} = S$

Number of items 5,000 $= |I|$

Number of users 10,000 $= |U|$

Number of ratings 201,407 $= \sum_{u \in U} |I_u|$

In order to compare the prediction quality of IRA and Scaling the scales, we performed several experiments on this dataset, each using different parameters for the recommendation algorithms. 1,006 user-item pairs for which a rating existed in the dataset were randomly chosen. In every experiment, predictions were calculated for each of these pairs, using leave-one-out cross-validation. 279 of the user-item pairs involved extremal values, which were regarded as the ratings of value 1, 2, 3, 11, 12, and 13.

Comparative results

Consider Table 3, which shows statistics for two of the experiments run on this dataset. The first experiment uses parameters $hort\_common = 5$, $hort\_frac = 1$, and $t\_diff_{max} = 0.2$, and shows the best results we were able to obtain when minimizing the average error, while keeping a success rate of at least 90 %. In the second experiment, we tightened the constrains for horting and relaxed the constraints for the average difference of the transformation function, to see how this would effect the performance of the algorithms. As in the experiments with the dataset crafted towards IRA, we see that Scaling the scales is able to identify considerably more prediction relations than IRA. In the two experiments, we see an increase in prediction relations identified by Scaling the scales compared to IRA of 80 % and 81 %, respectively. This leads to Scaling the scales being able to compute predictions in many cases where IRA is unsuccessful, and particularly so in the second experiment, where the prediction graphs are more sparse. Scaling the scales also generally finds many more prediction sources and prediction paths to base the prediction calculations on. Scaling the scales gives slightly higher prediction quality than IRA, which is expected when the dataset is crafted to fit the model Scaling the scales is based upon. Relative to the total number of relations identified by the algorithms, Scaling the scales identifies noticeably more inverse relations than IRA,

	$hort\_common = 5$ $hort\_frac = 1$ $t\_diff_{max} = 0.2$		$hort\_common = 8$ $hort\_frac = 1$ $t\_diff_{max} = 0.6$	
Algorithm	Scaling	IRA	Scaling	IRA
Success rate	97 %	96 %	51 %	46 %
Success rate extremal	97 %	97 %	54 %	48 %
Error	0.26	0.29	0.40	0.55
Error extremal	0.27	0.30	0.48	0.69
Prediction sources	25.41	15.58	9.89	5.17
Paths	49.60	26.87	14.92	7.56
Path length	1.40	1.47	1.94	2.10
Distinct edges	93 %	94 %	83 %	83 %
Common ratings	5.20	5.24	8.25	8.27
Trans func error	0.14	0.12	0.33	0.33
Relations	1,483,792	826,350	63,780	35,262
Inverse relations	5 %	2 %	4 %	1 %
Scaled relations	50 %	0 %	70 %	0 %
Graph density	0.01	0.008	0.0006	0.0004

Table 3: Performance comparison of IRA and Scaling the scales when creating predictions based on a dataset crafted towards the features of Scaling the scales.

and the amount of inverse relations is surprisingly high (5 % and 4 %), considering that the probability of a user being a contrarian was set to 1 % when generating the dataset. In both experiments, a high number of the relations are identified to be scaled by Scaling the scales, which is expected considering that the dataset is generated with many users rating on scaled rating scales.

Consider now Table 4. The table shows the relationships between real ratings and predicted ratings for the experiment using parameters $hort\_common = 5$, $hort\_frac = 1$, and $t\_diff_{max} = 0.2$. We see that Scaling the scales generally performs slightly better than IRA, but both algorithms perform very well, predicting the correct rating in about 80 % of the cases (the average value along the diagonal). Both algorithms give very small errors. We do not see any outliers, as we saw in the experiment using the dataset crafted towards IRA.

Scaling the scales

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	88	13	0	0	0	0	0	0	0	0	0	0	0
2	6	78	17	0	0	0	0	0	0	0	0	0	0
3	0	4	85	8	3	0	0	0	0	0	0	0	0
4	0	0	4	85	11	1	0	0	0	0	0	0	0
5	0	0	0	10	76	12	2	0	0	0	0	0	0
6	0	0	0	0	6	88	4	1	0	0	0	0	0
7	0	0	0	0	0	12	83	5	0	0	0	0	0
8	0	0	0	0	0	2	4	87	7	0	0	0	0
9	0	0	0	0	0	0	0	10	88	2	0	0	0
10	0	0	0	0	0	0	0	0	7	79	13	1	0
11	0	0	0	0	0	0	0	0	0	11	77	10	1
12	0	0	0	0	0	0	0	0	0	3	16	76	5
13	0	0	0	0	0	0	0	0	0	0	11	11	78

IRA

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	83	13	4	0	0	0	0	0	0	0	0	0	0
2	11	65	24	0	0	0	0	0	0	0	0	0	0
3	0	5	83	11	0	1	0	0	0	0	0	0	0
4	0	0	5	80	15	0	0	0	0	0	0	0	0
5	0	0	1	7	78	13	2	0	0	0	0	0	0
6	0	0	0	0	7	83	10	0	0	0	0	0	0
7	0	0	0	0	0	9	84	7	0	0	0	0	0
8	0	0	0	0	0	1	8	80	10	1	0	0	0
9	0	0	0	0	0	0	0	13	81	5	0	0	0
10	0	0	0	0	0	0	0	1	10	75	11	3	0
11	0	0	0	0	0	0	0	0	0	14	77	9	0
12	0	0	0	0	0	0	0	0	0	3	21	76	0
13	0	0	0	0	0	0	0	0	0	0	0	22	78

Table 4: Prediction accuracy for Scaling the scales (top) and IRA (bottom) when using parameters $hort\_common = 5$, $hort\_frac = 1$, and $t\_diff_{max} = 0.2$ on a dataset crafted towards the features of Scaling the scales. Rows correspond to actual ratings, while columns correspond to predicted ratings. See Section 5.2.1 for details on how to interpret the tables.

5.2.3 Real dataset: MovieLens 100k

GroupLens [15], a research lab at the Department of Computer Science and Engineering at the University of Minnesota, has a long history of research on recommender systems. They provide a movie recommendation service called MovieLens. MovieLens has hundreds of thousands of registered users, and GroupLens has collected and made available some rating datasets from this service. We will now see how IRA and Scaling the scales perform when providing recommendations based on one of these datasets, called "MovieLens 100k" [16]. This dataset has the following properties.

Rating scale $\{1, \dots, 5\} = S$

Number of items $1{,}682 = |I|$

Number of users $943 = |U|$

Number of ratings $100{,}000 = \sum_{u \in U} |I_u|$

In order to compare the prediction quality of IRA and Scaling the scales, we performed several experiments on this dataset, each using different parameters for the recommendation algorithms. 1,043 user-item pairs for which a rating existed in the dataset were randomly chosen. In every experiment, predictions were calculated for each of these pairs, using leave-one-out cross-validation. 284 of the user-item pairs involved extremal values, which were regarded as the ratings of value 1 and 5.

Comparative results

Consider Table 5, which shows statistics for two of the experiments run on this dataset. The first experiment uses parameters $\mathit{hort\_common} = 4$, $\mathit{hort\_frac} = 1$, and $\mathit{t\_diff}_{max} = 0.8$, and shows the best results we were able to obtain when minimizing the average error, while keeping a success rate of at least 90 %. In the second experiment, we tightened the constrains for both horting and the average difference of the transformation function, to see how this would effect the performance of the algorithms. As expected, we see that Scaling the scales is able to identify more prediction relations than IRA, but the increase is considerably smaller than in the synthetic datasets, as seen in Sections 5.2.1 and 5.2.2. In both experiments, we see an increase in prediction relations identified by Scaling the scales compared to IRA of about 2 %. Since this increase is so small, the algorithms perform almost identically in both experiments. Scaling the scales has a slightly higher success rate, and a slightly lower average error. Relative to the total number

	$hort\_common = 4$ $hort\_frac = 1$ $t\_diff_{max} = 0.8$		$hort\_common = 11$ $hort\_frac = 1$ $t\_diff_{max} = 0.4$	
Algorithm	Scaling	IRA	Scaling	IRA
Success rate	100 %	100 %	77 %	76 %
Success rate extremal	100 %	100 %	77 %	76 %
Error	0.58	0.59	0.77	0.79
Error extremal	0.83	0.85	0.87	0.88
Prediction sources	55.36	55.05	4.33	4.15
Paths	107.79	104.89	4.71	4.49
Path length	1.03	1.03	1.90	1.90
Distinct edges	99 %	99 %	84 %	84 %
Common ratings	52.48	52.57	17.46	17.57
Trans func error	0.69	0.69	0.34	0.34
Relations	310,302	305,610	2,924	2,856
Inverse relations	25 %	24 %	16 %	17 %
Scaled relations	7 %	0 %	3 %	0 %
Graph density	0.35	0.34	0.003	0.003

Table 5: Performance comparison of IRA and Scaling the scales when creating predictions based on the dataset “MovieLens 100k”

of relations identified, the two algorithms identify more or less the same number of inverse relations. A low number of the relations are identified to be scaled by Scaling the scales in both experiments. The reason for the number of scaled relations being so small can be attributed to the rating scale being so small. There is reason to think that users are more likely to only use a smaller part of the scale if the rating scale is very large. When rating on a scale of size 3, for example, users might give items they don't like the rating 1, items that are ok the rating 2, and items they like the rating 3, and hence use the whole scale. If the rating scale had size 100, however, it is harder to decide which ratings to give. A user might never like an item so much that it gets the rating 100, and the full size of the rating scale is not utilized by the user. We believe that the likelihood of users rating on scaled scales increases as the size of the rating scale increases.

Consider now Table 6. The table shows the relationships between real ratings and predicted ratings for the experiment using parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 0.8$. We see that both algorithms perform very similarly, with Scaling the scales performing slightly better concerning extremal values (1 and 5). Interestingly, both algorithms tend to give predictions of higher

Scaling the scales

	1	2	3	4	5
1	17	43	33	7	0
2	1	25	54	20	1
3	0	3	60	36	0
4	0	1	17	79	3
5	0	0	3	66	31

IRA

	1	2	3	4	5
1	13	39	39	9	0
2	1	24	53	22	1
3	0	3	61	35	0
4	0	1	17	79	3
5	0	0	4	66	30

Table 6: Prediction accuracy for Scaling the scales (left) and IRA (right) when using parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 0.8$ on the dataset "MovieLens 100k". Rows correspond to actual ratings, while columns correspond to predicted ratings. See Section 5.2.1 for details on how to interpret the tables.

value than the real rating. The sum below the diagonal is around 90, while the sum above the diagonal is around 200 for both algorithms. Both algorithms predict the correct rating in about 42 % of the cases (the average value along the diagonal).

5.2.4 Real dataset: HetRec 2011

Several real datasets were released in the framework of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) [7] at the 5th ACM Conference on Recommender Systems. The focus of this workshop was raising awareness of the potential of using multiple sources of information in recommender systems. We will now see how IRA and Scaling the scales perform when providing recommendations based on one of these datasets, combining information from both IMDb [20], RottenTomatoes [10] and a dataset released by GroupLens [15], containing 10,000,000 ratings. This dataset [27] has the following properties.

Rating scale $\{1, \dots, 10\} = S$[1]

Number of items $10{,}197 = |I|$

Number of users $2{,}113 = |U|$

Number of ratings $855{,}598 = \sum_{u \in U} |I_u|$

[1]The actual rating scale in this dataset is from 0.5 to 5, with steps of 0.5. For simplicity, we have multiplied all ratings in the dataset by 2, and hence turned the rating scale into a scale from 1 to 10, with steps of 1, which fits the model described in Section 2.1.

In order to compare the prediction quality of IRA and Scaling the scales, we performed several experiments on this dataset, each using different parameters for the recommendation algorithms. 1,017 user-item pairs for which a rating existed in the dataset were randomly chosen. In every experiment, predictions were calculated for each of these pairs, using leave-one-out cross-validation. 246 of the user-item pairs involved extremal values, which were regarded as the ratings of value 1, 2, 9, and 10.

Comparative results

Consider Table 7, which shows statistics for two of the experiments run on this dataset. The first experiment uses parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 1.4$, and shows the best results we were able to obtain when minimizing the average error, while keeping a success rate of at least 90 %. In the second experiment, we relaxed the constrains for horting and tightened the constraints for the average difference of the transformation function, to see how this would effect the performance of the algorithms. Again, we see that Scaling the

	$hort\_common = 4$, $hort\_frac = 1$, $t\_diff_{max} = 1.4$		$hort\_common = 3$, $hort\_frac = 1$, $t\_diff_{max} = 0.5$	
Algorithm	Scaling	IRA	Scaling	IRA
Success rate	96 %	95 %	14 %	11 %
Success rate extremal	95 %	94 %	22 %	15 %
Error	1.09	1.09	0.78	0.87
Error extremal	1.56	1.58	0.75	0.98
Prediction sources	111.20	111.14	6.69	4.47
Paths	112.77	112.71	7.10	4.48
Path length	1.06	1.06	1.19	1.20
Distinct edges	98 %	98 %	93 %	93 %
Common ratings	217.66	218.13	5.52	5.91
Trans func error	1.26	1.27	0.34	0.37
Relations	1,389,822	1,376,540	61,332	45,076
Inverse relations	18 %	15 %	36 %	31 %
Scaled relations	7 %	0 %	49 %	0 %
Graph density	0.31	0.31	0.01	0.01

Table 7: Performance comparison of IRA and Scaling the scales when creating predictions based on the dataset “HetRec 2011”

scales is able to identify more prediction relations than IRA. In the first experiment, we see an increase in prediction relations identified by Scaling the scales compared to IRA of only 1 %. Since this increase is so small, IRA and Scaling the scales perform almost identically. Scaling the scales has a slightly higher success rate, and a slightly lower average error for extremal values. In the second experiment, we see an increase in prediction relations identified by Scaling the scales compared to IRA of 36 %. This leads to Scaling the scales being able to compute predictions in many cases where IRA is unsuccessful. Scaling the scales also generally finds many more prediction sources and prediction paths to base the prediction calculations on. In the second experiment, Scaling the scales gives noticeably higher prediction quality than IRA. Relative to the total number of relations identified by the algorithms, Scaling the scales identifies slightly more inverse relations than IRA. We are not certain about what might cause this, but one explanation we find reasonable is that users with different taste are more likely to use differently sized ranges of the scale. When two users have so different taste that they rate opposite, they are more likely to use scaled scales than when they have more similar taste, and only rate offset compared to each other. This would make Scaling the scales able to identify more inverse relations than IRA, also relative to the total number of relations they identify. In the first experiment, a small fraction of the relations are identified to be scaled by Scaling the scales, while the fraction is much larger in the second experiment.

Consider now Table 8. The table shows the relationships between real ratings and predicted ratings for the experiment using parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 1.4$. We see that the algorithms perform similarly. As in the experiment in Section 5.2.3, both algorithms tend to give predictions of higher value than the real rating. In particular, there are so few predictions of value 1 that the first column in the table is completely empty for both algorithms! Both algorithms tend to make occasionally large errors, for example 4 % of the ratings of value 2 were predicted to be of value 9. Both algorithms predict the correct rating in around 23 % of the cases (the average value along the diagonal).

Scaling the scales

	1	2	3	4	5	6	7	8	9	10
1	0	14	14	14	29	21	7	0	0	0
2	0	13	4	29	17	8	8	17	4	0
3	0	0	18	36	18	18	0	9	0	0
4	0	0	2	15	20	34	19	3	3	3
5	0	0	3	5	18	37	26	11	0	0
6	0	0	0	3	9	33	41	10	3	1
7	0	0	0	1	2	18	44	28	6	1
8	0	0	0	1	2	5	40	43	9	0
9	0	0	0	0	0	4	17	49	28	3
10	0	0	0	0	4	0	8	29	39	20

IRA

	1	2	3	4	5	6	7	8	9	10
1	0	14	14	7	29	29	7	0	0	0
2	0	13	4	29	17	8	8	17	4	0
3	0	0	18	36	18	18	0	9	0	0
4	0	0	2	14	22	32	20	3	3	3
5	0	0	3	5	17	36	29	11	0	0
6	0	0	0	3	9	34	41	10	3	1
7	0	0	0	1	2	18	44	28	6	1
8	0	0	0	1	2	5	39	44	9	0
9	0	0	0	0	0	4	17	48	29	3
10	0	0	0	0	2	1	8	31	38	20

Table 8: Prediction accuracy for Scaling the scales (top) and IRA (bottom) when using parameters $hort\_common = 4$, $hort\_frac = 1$, and $t\_diff_{max} = 1.4$ on the dataset "HetRec 2011". Rows correspond to actual ratings, while columns correspond to predicted ratings. See Section 5.2.1 for details on how to interpret the tables.

Chapter 6

Analysis and discussion of the experimental results

In this chapter, the results presented in the previous chapter will be analyzed. The strengths and weaknesses of Scaling the scales in comparison to IRA will be highlighted, and the tangible benefits and drawbacks discussed.

6.1 Success rate

Our experiments show that Scaling the scales gives a noticeable improvement over IRA's ability to predict ratings. As discussed in Section 4.1, Scaling the scales always identifies at least as many prediction relations as IRA. This leads to Scaling the scales being able to successfully predict ratings in many cases where IRA is unsuccessful.

This improvement is most prominent when prediction graphs calculated by the algorithms are sparse. In some cases, in particular when the recommender system has recently been deployed, the prediction graph is likely to be sparse. When few users and ratings have been registered, the lack of data makes recommender systems less likely to identify many users with similar taste. This is part of the cold start problem, discussed in Section 1.1. The improved ability to identify prediction relations leads us to think that Scaling the scales should have less of a cold start problem than IRA[1]. Note that no experiments were run on datasets representing a

[1]Scaling the scales only provides a potential improvement of the part of the cold start problem concerning not being able to identify users with similar taste. Scaling the scales does not attempt to solve the problems of users not having rated many items or items not having been rated.

recently deployed system. Our experiments show an increased success rate over IRA on datasets where the prediction graph is sparse due to using tight conditions for similarity in user taste. Whether this effect also applies to datasets where the prediction graph is sparse due to the system being recently deployed will have to be tested in a separate experiment using an appropriate dataset.

When the prediction graph calculated by IRA is less sparse, Scaling the scales provide less improvement in success rate. In these cases, IRA already has a more or less universal success rate.

6.2 Prediction quality

In all our experiments, except for when the dataset was crafted towards IRA (Section 5.2.1), we see that Scaling the scales calculates predictions of slightly higher quality than IRA. The improvements are generally quite small, and whether they are results of random variation or if they would provide a tangible benefit in a real live system is difficult to discern. Having said that, also small improvements can substantially increase the value of a recommender system. In a large webshop, for instance, a small improvement leading to a 1‰ increase of users buying the recommended items could result in significantly increased earnings. As discussed in the introduction to Chapter 5, running tests on a real live system is required to confidently decide which recommender system provides the highest recommendation quality in a particular application.

6.3 Fallibility

As seen in Section 5.2.1, Scaling the scales seems more prone to making large mistakes when calculating predictions than IRA. A reason for this can be users wrongly identified as having compatible taste through scaled transformation functions. Consider the example in Figure 14, where two users are identified as having compatible taste by Scaling the scales, even though the likelihood of the users actually having compatible tastes does not seem very high.

The items both users have rated are *game*, *picture*, and *cd*. User *a* has given both *game* and *picture* the rating 2, while user *b* has given those items the rating 1. Almost no matter which ratings the users have given the *cd*, Scaling the scales will be able to find a transformation function giving an average difference of 0

	🎮	⚽	🖼	🎸	📖	💿	📺
a	2		2		5	3	
b	1		1			7	

Figure 14: An example showing the ratings of two users whose ratings are not diverse. The rating scale goes from 1 to 7. The value in the cells represent the rating given by the row user to the column item.

between the users[2]. In this example, Scaling the scales will identify $t_{a \to b}(r_a) = (r_a - 2) \cdot \frac{7-1}{3-2} + 1 = (r_a - 2) \cdot 6 + 1$ as a good transformation function from a rating by user a to a predicted rating for user b. When looking at only $i \in I_a \cap I_b$, we see that $t_{a \to b}(r_{a,i}) - r_{b,i} = 0$ in all cases. Hence we get that the average difference $t\_diff_{a \to b}$ between the transformed ratings by user a and the ratings by user b equals 0, which means that the users have compatible taste. Based on user a's rating of *book*, we can predict that user b would give *book* the rating $p_{a \to b, book} = t_{a \to b}(r_{a,book}) = (r_{a,book} - 2) \cdot 6 + 1 = 19$. This does not seem like a very likely rating on a scale from 1 to 7! Ideas on how this problem could be avoided are presented in Section 7.2.1.

6.4 Synthetic vs. real data

Our experiments show a significant difference in prediction quality when using synthetic and real data. While both algorithms provide accurate predictions when using synthetic datasets, as seen in Sections 5.2.1 and 5.2.2, the prediction quality is considerably lower when using real datasets, as seen in Sections 5.2.3 and 5.2.4. Not surprisingly, it seems that real data must be used when analyzing the real performance of the algorithms, while synthetic datasets can be useful for confirming correctness of the implementation. An algorithm performing badly on data crafted towards the algorithm might indicate that there is an error in either the generated data or the implemented algorithm. On real data we see that both Scaling the scales and IRA tend to give predictions of too high rating value.

[2]The exceptions are if user a has given the rating 2, while user b has given a rating different from 1, or when user a has given a rating different from 2, while user b has given the rating 1.

Particularly in Section 5.2.4, we see that neither algorithm predict any ratings to be of value 1. It would be interesting to investigate what might cause this, as, to our understanding, there is nothing in the algorithms indicating that the predictions calculated should be overly positive. Our understanding is also confirmed by the experiments on synthetic data, where we do not see the trend of predicted values being too high.

6.5 Rating scale sizes

In Section 5.2.3, we argued that Scaling the scales can provide greater improvements over IRA when using a large rating scale. Studies have shown that the reliability of data collected in surveys does not increase substantially if the number of choices is increased beyond seven [29]. This undermines the usefulness of the increased improvement provided by Scaling the scales: If large rating scales are not used, due to there being no benefits from using them, there is also no benefit from having an algorithm with increased performance for large rating scales. Another issue arising when increasing the size of the rating scale is the increased running time of the algorithm.

6.6 Running times

As discussed in Section 4.1, the algorithm for identifying users with compatible taste runs much slower for Scaling the scales than for IRA. In Section 4.2 we argued that this might be acceptable, as this algorithm does not necessarily have to be run when calculating predictions in real-time. Which running times are acceptable must be determined for each particular application, but Scaling the scales definitely performs worse than IRA in this aspect.

Chapter 7

Conclusion and further research

7.1 Summary

In this thesis, a suggested improvement to IBM's Intelligent Recommendation Algorithm (IRA), introduced by Aggarwal et al. [1] in 1999 has been presented. The suggested improvement, called Scaling the scales, is based on the assumption that users rating on differently sized ranges of a rating scale can be used to predict each other. This is a simple and natural extension of the concept of users having compatible taste, already used by IRA.

Experiments have been run using both synthetic and real datasets. To our knowing, this thesis presents the first published results on how IRA performs when calculating predictions based on real data. Our experiments show that Scaling the scales, in many cases, give noticeable improved success rate over IRA when calculating predictions. This is most prominent when the predictions are calculated based on sparse prediction graphs. Scaling the scales also gives a slight improvement in prediction quality in all experiments except for the one crafted towards the IRA algorithm. Whether this improvement is enough to give any benefit in a real live system is difficult to discern from our results, and experiments on a real live system would be needed. That Scaling the scales does not show any decline in prediction quality except in the experiment crafted towards the IRA algorithm is, however, encouraging.

Improvements usually do not come without a cost, and the cost of Scaling the scales lies in its running time. The process of identifying users with compatible taste is considerably slower for Scaling the scales than for IRA. In many applications this might not pose a problem, but the increase in running time is a clear disadvantage of Scaling the scales.

Scaling the scales is also more prone to make big mistakes than IRA. Ideas for further improving the algorithm to overcome this issue by reducing the chances of identifying users with compatible taste erroneously will been presented in Section 7.2.1. Finally, aiming to increase the prediction quality as well as the usefulness of the predictions provided by the algorithm, two further ideas for improvement will also be presented in the next section.

7.2 Further research

This section presents three ideas for further improvement of IRA and Scaling the scales. The first idea introduces new constraints for users to have compatible taste, and aims to overcome the problem described in Section 6.3. The two other ideas concern the process of calculating predictions in the prediction graph, and aim to increase prediction quality and usefulness. We would like to emphasize that these ideas suggest improvements to IRA, as well as Scaling the scales. For instance in systems with particularly high requirements for speed, an improved version of IRA which does not introduce the time complexity of scaled rating scales can be of interest.

7.2.1 Rating diversity

As seen in Section 6.3, problems can arise when transformation functions are based on only a few different rating values. The horting condition, introduced in Definition 1, ensures that two users have rated some minimum number of items in common in order for them to predict each other. In some cases, especially when using Scaling the scales, this condition is not enough to confidently find a good transformation function between the users, as seen in Figure 14 and the following discussion. Some ideas for additional constraints that could reduce this problem are presented below. All of them look at two users a and b, and suggest constraints that should be satisfied in order for the users to predict each other.

- The ratings given by user a to items not rated by user b should not be transformed to values (far) beyond the rating scale by the function $t_{a \to b}$. The same applies in the opposite direction.

- For user a and user b to predict each other (through a scaled transformation function), both users must have used at least k different values on the rating scale when looking at the items both users have rated. This constraint could apply to scaled transformation functions only (the condition does not have to be satisfied for transformation functions that are not scaled), or to all transformation functions. We imagine that setting $k = 3$, and using it for scaled transformation functions only, would make for a good constraint in many cases.
- Transformation functions that enlarge or reduce the rating scale more than k times should not be considered. One can argue that the likelihood of user a and user b having compatible taste is small if user a's rating scale is for example more than three times larger than user b's rating scale. By setting $k = 3$, no such transformation functions would be considered.

The problems that might arise when transformation functions are based on only a few different rating values is most prominent with Scaling the scales, as errors can be scaled up to huge dimensions, as seen in Figure 14 and the following description. The problem is, however, also present in IRA. Consider, for instance, two users a and b and the items both users have rated ($I_a \cap I_b$). If user a has given the same rating to all of the items, and user b have given the same rating to all of the items, IRA would find a transformation function giving an average difference equal 0 by using an offset. It is, however, hard to decide how the users behave in relation to each other when giving ratings other than that one value the transformation function is based on. For instance, they may rate opposite, or maybe the ratings given by user a are bottom ratings, while the ratings given by user b are top ratings. In this case all other ratings than the bottom rating will be transformed to a value outside the rating scale when transforming from user a to user b. Hence the constraints just presented can also be used to improve IRA.

7.2.2 Rating the ratings

When predicting a rating for user a and item i, IRA and Scaling the scales perform a depth-limited breadth-first search in the prediction graph, looking for users having rated item i. As demonstrated in Section 3.2, the search returns all paths of length k from user a to users having rated item i, where $k \leq max\_path\_length$ is the shortest distance from user a to any user having rated item i. The ratings by the users found in the search are transformed along the paths, and finally the average of the transformed ratings is used as the predicted rating.

This suggested improvement is called "Rating the ratings", and changes the way predictions are calculated. In Rating the ratings, every prediction relation $a \to b$ is assigned a confidence value $0 \leq c_{a\to b} \leq 1$. This value is based on measures like the number of items both user a and user b have rated ($|I_a \cap I_b|$) and the average difference between the transformed ratings by user a and the ratings by user b ($t\_diff_{a\to b}$). The confidence value indicates the reliability and accuracy of the transformation function $t_{a\to b}$. A confidence of 1 indicates high reliability and accuracy, while a confidence of 0 indicates low reliability and accuracy. When predicting a rating for user c and item i, Rating the ratings searches in the prediction graph from user c using a *weighted shortest path search*, such as *Dijkstra's algorithm*. This means that prediction relations with high confidence will be explored earlier than prediction relations with low confidence. As the confidences lie between 0 and 1, the confidence of a path can be expressed as the product of the confidences of each edge in the path[1]. In order to not end up with only one prediction path, namely the one with highest confidence, the search should be continued further than to the first user having rated item i that is found. The search could for example continue until all paths have confidence below some threshold. Note that this threshold does not have to be fixed, but can be relative to the confidence of the first prediction path found. After transforming the ratings along the paths back to user c, each transformed rating is assigned the confidence (or rating, hence "Rating the ratings") of the path it was calculated along. The final prediction can now be calculated using a weighted average of the transformed ratings, where the weight of each transformed rating corresponds to its confidence value. We believe this will provide higher prediction quality than using the simple average of the transformed ratings.

It is worth to note that this suggested improvement will lead to a slightly increased running time for the predicting process of the algorithms, since a weighted shortest path search has higher time complexity than an unweighted shortest path search, such as breadth-first search. We do not consider it likely that the increased running time will pose any problem in most applications, as Dijkstra's algorithm is extremely fast. This is especially true when searching on sparse graphs that should not be completely searched, as is the case in most recommender systems.

7.2.3 "You will either love or hate this"

In IRA and Scaling the scales, the average of the transformed ratings is used as the final predicted rating. This might work well in many cases. In others, however,

[1]Usually, Dijkstra's algorithm explores low weighted edges first, and weights are added together along the paths. In Rating the ratings, the algorithm must be modified to explore edges of high confidence before edges of low confidence, and multiply the confidences along the paths.

valuable information might be lost when not paying attention to the individual transformed ratings.

Let's look at some examples showing how a simple analysis of the transformed ratings can provide valuable information about the prediction.

- If all or most transformed ratings indicate more or less the same predicted rating, the likelihood of the prediction being correct is high. Also, the likelihood of the prediction being correct increases as the total number of transformed ratings increases.
- If the transformed ratings indicate many different predictions, the likelihood of the prediction being correct is low, hence it has low reliability. Even if the average value of the transformed ratings is high, this might not be a good item to recommend. When expecting something to be average, an average item would meet expectations. Items expected to be good, but perceived as less good, lead to disappointments. Hence a reliable prediction of value 9 can in many cases indicate a better recommendation than a prediction of value 10 with low reliability. Even though the latter is more likely to be of high interest to the user, it is also more likely to disappoint the user.
- When dealing with controversial topics or content, users who normally have similar taste can be in complete disagreement. Whether a user will like or dislike an item with such content can be hard to predict, both in everyday life and for recommender systems. If the transformed ratings contain many values both near the bottom and the top of the scale, this might indicate an item with controversial content. Instead of predicting a rating of the average value in the middle of the scale, a recommendation saying "You will either love or hate this" might be more informative, valuable, and titillating for the user.

Note that we do not present any concrete suggestions on how an analysis of the transformed ratings should or could be done. We only state that performing such an analysis could lead to higher prediction quality and usefulness.

Bibliography

[1] Charu C Aggarwal, Joel L Wolf, Kun-Lung Wu, and Philip S Yu. "Horting hatches an egg: A new graph-theoretic approach to collaborative filtering". In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 1999, pp. 201–212.

[2] Anasse Bari, Mohamed Chaouchi, and Tommy Jung. *Predictive Analytics For Dummies.* John Wiley & Sons, 2014.

[3] The University of Bergen. *BORA - Bergen Open Research Archive.* URL: `https://bora-uib-no.pva.uib.no`.

[4] Mustafa Bilgic and Raymond J Mooney. "Explaining recommendations: Satisfaction vs. promotion". In: *Beyond Personalization Workshop, IUI.* Vol. 5. 2005.

[5] Manuel Blum, Robert W Floyd, Vaughan Pratt, Ronald L Rivest, and Robert E Tarjan. "Time bounds for selection". In: *Journal of computer and system sciences* 7.4 (1973), pp. 448–461.

[6] Joseph A Calandrino, Ann Kilzer, Arvind Narayanan, Edward W Felten, and Vitaly Shmatikov. """ You Might Also Like:" Privacy Risks of Collaborative Filtering". In: *Security and Privacy (SP), 2011 IEEE Symposium on.* IEEE. 2011, pp. 231–246.

[7] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. "2nd Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011)". In: *Proceedings of the 5th ACM conference on Recommender systems.* RecSys 2011. Chicago, IL, USA: ACM, 2011.

[8] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms.* McGraw-Hill, Inc., 2006.

[9] Cynthia Dwork and Aaron Roth. "The algorithmic foundations of differential privacy". In: *Theoretical Computer Science* 9.3-4 (2013), pp. 211–407.

[10] Flixter. *Rotten Tomatoes: Movies — TV Shows — Movie Trailers — Reviews.* URL: `http://www.rottentomatoes.com`.

[11] John B Fraleigh. *A first course in abstract algebra.* Pearson Education India, 2003.

[12] Robert Garfinkel, Ram D Gopal, Bhavik K Pathak, Rajkumar Venkatesan, and Fang Yin. "Empirical analysis of the business value of recommender systems". In: *Available at SSRN 958770* (2006).

[13] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. "Using collaborative filtering to weave an information tapestry". In: *Communications of the ACM* 35.12 (1992), pp. 61–70.

[14] Dan R Greening. "Collaborative filtering for Web marketing efforts". In: *Recommender Systems, Papers from the 1998 Workshop.* 1994, pp. 53–55.

[15] GroupLens. *GroupLens: Social Computing Research at the University of Minnesota.* URL: `http://www.grouplens.org`.

[16] GroupLens. *MovieLens Datasets.* URL: `http://www.grouplens.org/datasets/movielens`.

[17] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. "Evaluating collaborative filtering recommender systems". In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 5–53.

[18] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. "Explaining collaborative filtering recommendations". In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work.* ACM. 2000, pp. 241–250.

[19] Zan Huang, Hsinchun Chen, and Daniel Zeng. "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering". In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 116–142.

[20] IMDb. *IMDb - Movies, TV, Celebrities.* URL: `http://www.imdb.com`.

[21] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction.* Cambridge University Press, 2010.

[22] Greg Linden, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering". In: *Internet Computing, IEEE* 7.1 (2003), pp. 76–80.

[23] Magnar Myrtveit. *GitHub - Scaling the scales.* URL: `https://github.com/Stadly/Scaling-the-scales`.

[24] Félix Hernández del Olmo and Elena Gaudioso. “Evaluation of recommender systems: A new approach”. In: *Expert Systems with Applications* 35.3 (2008), pp. 790–804.

[25] Eli Pariser. “Beware online “filter bubbles””. In: *TED Talks, March* (2011). URL: `http://www.ted.com/talks/eli_pariser_beware_online_filter_bubbles`.

[26] Eli Pariser. *The filter bubble: What the Internet is hiding from you.* Penguin UK, 2011.

[27] 5th ACM Conference on Recommender Systems. *2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011).* URL: `http://www.grouplens.org/datasets/hetrec-2011`.

[28] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook.* Springer, 2011.

[29] Robert Rosenthal and Ralph L Rosnow. *Essentials of behavioral research: Methods and data analysis.* Vol. 2. McGraw-Hill New York, 1991.

[30] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. “Methods and metrics for cold-start recommendations”. In: *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval.* ACM. 2002, pp. 253–260.

[31] Google Scholar. *Horting hatches an egg: A new graph-theoretic approach to collaborative filtering.* URL: `http://scholar.google.no/scholar?cites=4418358089889456011` (visited on 2014-10-14).

[32] Guy Shani and Asela Gunawardana. “Evaluating recommendation systems”. In: *Recommender systems handbook.* Springer, 2011, pp. 257–297.

[33] Upendra Shardanand and Pattie Maes. “Social information filtering: algorithms for automating “word of mouth””. In: *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM Press/Addison-Wesley Publishing Co. 1995, pp. 210–217.

[34] VisualPharm. *Icons8 - Free Windows 8 Icons.* URL: `http://icons8.com`.

[35] Sean Work. “How Loading Time Affects Your Bottom Line”. In: *KISSmetrics, April* (2011). URL: `https://blog.kissmetrics.com/loading-time`.