UNIVERSITY OF TWENTE.

Making people matches using Supervised Machine Learning algorithms

Master of Science Thesis

Nils van Kleef

Supervisors Universiteit Twente:

- dr.ir. R.W. Poppe
- prof.dr. D.K.J. Heylen
- dr. M. Poel

Supervisors Paiq BV:

- ir. F.C. van Viegen

28 April 2014

Voorwoord

Online dating wordt steeds meer geaccepteerd door de samenleving. Mensen kiezen er steeds vaker voor om via een dating site mogelijk een partner te vinden dan door de kroeg in te gaan en daar vrouwen aan te spreken. De voordelen zijn een veel grotere verscheidenheid aan mogelijke partners en de laagdrempeligerheid van contact leggen. Natuurlijk zijn er ook nadelen zoals het veel meer afstandelijker contact en de mogelijkheid makkelijk en snel iemand weg te klikken, waardoor mensen ook steeds minder snel tevreden zijn en maar door blijven gaan met zoeken. De algemene trend is nog steeds dat dating sites steeds vaker worden gebruikt.

Mijn afstudeeronderzoek gaat over recommender systems. Het systeem kan idealerwijs foto's aanbevelen aan gebruikers aan de hand van hun stemgedrag op andere foto's, waarbij gebruik wordt gemaakt van wat andere gebruikers van die foto's vinden. Dit zijn dan hopelijk foto's die deze gebruiker ook leuk zal vinden. Met mijn onderzoek hoop ik een radertje aan het matchingsalgoritme van Paiq toe te voegen en deze weer te verbeteren. Als hierdoor al een paar mensen succesvol met elkaar in contact zijn gebracht ben ik al tevreden met de praktische toepasbaarheid van mijn onderzoek..

Mijn Master Thesis had niet mogelijk geweest zonder een aantal mensen. Bij deze bedank ik Ronald Poppe voor zijn begeleiding, en ook voor zijn geduld en altijd erg nuttige feedback. Ik wil Frank van Viegen en de andere mensen van Paiq bedanken voor de ondersteuning en de goede tijd die ik bij hen heb gehad.

Op het bij elkaar brengen van mensen in de liefde!

Nils van Kleef
Enschede, 25 april 2014

Contents

1 Introduction

This chapter illustrates the background to the research in this thesis, the problem statement and the structure of this thesis.

1.1 Background

1.1.1 Paiq.nl

The website Paiq.nl is a dating site that first came online on 2 June 2005, created by two former computer science graduate students of the University of Twente: Frank van Viegen and Jelmer Feenstra. Its office is located in the city center of Enschede, and there are still some ties with the university, for example by hiring students as part-time employees.

Imagine the following scenario: on a dating site with profiles, it is often up to the men to initiate contact with women. Because profiles usually have photos of the person and because in general men are very much visually oriented, 80% of the men contact the top 20% of best looking women. These women get lots of messages, and respond to those coming from the top 20% of best looking men, getting conversations going, and perhaps dating in the real world too. The rest of the women get few messages and the large group of men gets few responses, if any at all, leaving most people disillusioned with online dating and leaving it altogether within several months.

While the numbers in the above scenario are made up, a scenario like the one described often happens with dating sites using profiles. Paiq tries to circumvent this problem by presenting a new way of dating online, and thereby became part of a new generation of online dating sites. Doing away with profiles, users just fill out a couple of questionnaires so that the system gets an idea of what kind of person the user is and what his or her preferences are. The system then matches users with similar personalities and preferences with an artificial intelligence. It is a self-learning system that by using feedback has learned to use combinations that in practice seem to work well. This is keeping in mind that even though one of the dating paradigms is ‘opposites attract’, there has to be enough similarity between people for dating to work.

1.1.2 The quest for photo suggestions

Prospective partners’ physical appearance (beauty) is the single strongest predictor of attraction for people [18]. Paiq uses some simple numeric information about a person’s appearance to improve its matching algorithm such as age, weight and height, and matching dating preferences, as well as the average rated attractiveness of photos, but this leaves out a lot about a person liking someone else’s appearance.

A user hopes to date someone whose desirability in looks is just a bit above their own

desirability. Someone with desirability far above their own will not want to date them, while it is less satisfactory for a user to date someone with desirability below their own. This would create a paradox, except that people have certain preferences for looks of a potential partner. For example, some people prefer partners with blond hair, while others prefer those with brown hair. This means that a person, who is judged on average to be okay looking, might be good looking (have a better desirability) to a group of people. The implication is that people want to date someone who is better than average looking, illustrated with another example: a person that is judged to be on average a 7 will want to date another person of preferred gender who is judged to be on average a 7 as well, while they both judge each other to be an worth an 8.

Paiq has a photo rating app built into their website. Users can rate photos of faces of their preferred sex for dating here. The way it works is that a user is shown a new photo and has to compare it against a stack of photos the user already rated. The user is asked to insert the photo into the stack by means of a binary search algorithm, using “is leuker dan” (is cuter than), “vergelijkbaar” (comparable) and “minder leuk” (less cute) buttons, where the “vergelijkbaar” button is treated as having found a match in the binary search algorithm. The photo is then inserted into this stack at that position, with lesser liked photos below it and better liked photos above it. Next, another new photo is presented to the user, resulting in a final stack of fifty photos. Of course it starts off with two new photos, one of which is considered to already be in the stack.

The photo rating app creates a wealth of data not only about users’ looks in photos in general (average photo ratings), but also for a specific person (that person’s preferences for looks). An algorithm is needed that can exploit these ratings to create predictions for what photos a user might like, and use that to offer the user photo suggestions.

1.2 Problem Statement

Recommendation systems make the following assumption: user behavior in the past can be used to predict user behavior in the future. Some of the better known uses of recommender systems are on the websites Amazon.com and Netflix.com. These websites take into account previous information collected about the users on a (small) subset of information or products such as buys, views, and ratings, and use these to recommend possibly interesting items to them [2]. The assumption is that there is a correlation between items that the recommender systems can use, implying that for example a person that buys a fork has a big chance of also wanting to buy a related product such as a knife or a plate.

Recommender systems such as the aforementioned ones have been further improved based on another assumption, namely that not only items can be related, but also that people can be related. They have the following train of thought: if two photos are related, and one of them has been rated highly by a person, this means that we can likely recommend the other photo to this person too. However, it can be that two photos do not seem to be related, but when taking a group of people and looking at their preferences they suddenly appear to be. If we can find out

how people are related to other people concerning their photo preferences, and what photos hold their preference, we can make even better recommendations. This is based on the assumption that these relations between people exist: not only are photos correlated, but also users, and that this information is hidden in the ratings data: ratings from the same photo but from different users are correlated. People rate differently when compared to some groups of people, but similar to yet other groups of people. This is further explored in the literature chapter in the discussion of recommender systems.

Another assumption made specifically for this research is that we only have rating information to base our predictions on. Most other datasets also have other data to use, such as date of rating, buys and views, as explained in the paragraph above about Amazon.com and Netflix.com. The Paiq dataset, containing only ratings, is a lot sparser and has a far larger number of users and items than some of the datasets used in other scientific literature. Not only that, but the set of available values is pretty random as well, as opposed to for example the Netflix dataset of user's movie ratings, which has a set of well known movies that many of their users will have rated. Thus, some of the most important challenges that this research faces in creating recommendations based on the Paiq dataset encounters are sparsity, scalability and noise-related problems. These challenges make up the novelty of this research and will be thoroughly explored in the following chapters.

Something else to consider when predicting ratings, is that one approach can lead to a lower MAE because it predicts ratings more conservatively than another approach. If the conservative predictions are the reason, another approach might be more desirable. Especially in recommender systems, one would like to have a small amount of none-conservative ratings of high confidence, rather than a high amount of many conservative ones. This also goes for Paiq: when looking to create recommendation for a user, Paiq would rather have a couple of photos with high predicted rating as well as high confidence than higher accuracy overall but a smaller amount of obvious matches. In other words: Paiq wants to predict 9s and 10, and not 5s to 7s.

1.3 Thesis structure

First, the research intends to find out what types of recommendation algorithms are suitable. Then it needs to find suitable machine learning approaches that belong to these types. It looks to find a baseline method we can use as a metric against which the other algorithms can be tested. Next, the experiment is done where first the selected algorithms are finetuned and then tested to find out which one is most accurate. They are also matched up against the baseline algorithm to see whether the extra computational costs are worth it. Then, the ratings are split into different subsets depending on their height, and matched against each other and the baseline method. Finally, the results are examined: does the noise in the dataset prove to be a problem for the machine learning algorithms?

This thesis has the following structure:

- Chapter 2, literature, details the literature research on recommender systems that was done for this thesis, and covers dataset problems, metrics, classifying recommendation methods and specific recommendation approaches.
- Chapter 3, experimental setup, contains the setup of this experiment, with the research questions, dataset, metrics and implementation details of the methods used.
- Chapter 4, parameter estimation, shows how the parameters of the recommender algorithms used were selected.
- Chapter 5, results and discussion, shows the results of the experiments, and discusses these results.
- Chapter 6, conclusion, answers the research questions, presents possible options for future work and suggests recommendations to Paiq.

2 Literature

This chapter details the literature research that was done for this thesis, and covers scientific datasets, dataset problems, metrics, classifying recommendation methods and specific recommendation approaches

2.1 Datasets

2.1.1 Scientific datasets

There are a lot of datasets that are used in the scientific world to test recommendation algorithms. These datasets are often based on a customer-product principle. They can be used to measure the performance of recommendation algorithms against other recommendation algorithms in the scientific world, providing grounds for testing the algorithms in a scientific way.

Some of the standard ones used are the so called ‘MovieLens’, ‘BookCrossing’ and ’Joker’ datasets. A more recent one is the Netflix dataset. These datasets are discussed below.

2.1.1.1 The Netflix dataset

The most comparable dataset to the Paiq dataset is the Netflix dataset of the Netflix Prize Challenge started in October 2006 [58]. This competition was set up by Netflix as a competition worth 1 million dollars, with the goal of coming up with a recommendation system that beat the RMSE (root mean squared error) of their own basic algorithm by 10% [17,24,26,27,58], which started in October 2006 and ended in September 2009. The importance of this challenge for the field recommendation algorithms is also that it made researchers compare their approaches and algorithms against others that had been published maybe only some months before, instead of often many years before, speeding up scientific research in this field greatly.

This dataset that has been used a lot in scientific research in the past years. The following list shows some information about the Netflix training dataset:

- 100,480,507 ratings
- 480,189 users
- 17,770 items
- Total number of possible ratings: 8,529,600,000
- Missing data ~= 98,822%

Using this dataset for comparison might not be that relevant because all (consecutive) research on this dataset has been optimized to this dataset as much as possible with little regards to computational costs. It can however give an insight into the type of algorithms that work on such a large dataset with a huge amount of missing data.

2.1.2 Dataset problems

In recommender systems, a problem often occurs that is called data sparsity [1,2,3,8]. This happens when there are so many users and items that only few ratings are available from those users for items. This can make it very hard for the recommendation algorithm to find similarity between users. Even though recommendation systems often have to work with little data, there has to be a sufficient mass of data to actually make correct predictions and recommendations. Coverage is the percentage of items that the recommendation system can make good predictions for [8]. If an item has not been rated a sufficient number of times, it could lead to the item only very rarely being recommended, even though the few users that did rate it gave it very high ratings.

A way of handling data sparsity is to also take into account user profile information when calculating user similarity [1]. Users within the same age range, gender, location, education level could be considered to be more similar to each other, extending the collaborative filtering techniques with “demographic filtering” [1]. Another example is the use of popularity characteristics of products [3], but this could lead to popular items being recommended more often, where with the Paiq dataset, recommendation of less popular photos is more interesting.

A sub-problem of data sparsity called cold-start problems are problems that occur with recommending items that no one (or only few users) have yet rated [1,11] or recommending items for users that have not rated any items yet (or only very few) [1]. They are often again subdivided into the new user problem and new item problem [1]. A possibility for handling this is the use of a combination of general recommendations and user specific recommendations, using the first for new(er) users and the second for users that have rated a sufficient number of items.

Scalability is another problem recommendation systems with large and growing datasets have to deal with. Especially memory-based collaborative filtering techniques have the problem that when the number of users and items continues to grow, this causes the computational cost of the recommendation algorithm to grow even faster [2]. Introducing model-based collaborative filtering techniques where a model is created from the available data to base predictions on, help combat this problem [8].

Users are human beings and are fickle in their ratings of items: the repeatability of users and their own ratings is notoriously low. Not only can users change in their preferences, but their mood can also lead to giving more positive or more negative ratings. Ratings can change by as much as 40% of the rating scale from day to day [12], suggesting that when working with model-based methods, a low-rank approximation of data is probably better at generalizing data than perfectly reconstruction the data with a medium-rank model [12]. Furthermore, it can be said that any recommendation algorithm cannot be more accurate than the variance in its users’ ratings [13].

The synonymy limitation occurs when two (very) similar items with different names are in the database. Correlation-based systems do not see that these items are essentially the same [2,8]. When a large number of synonymy exists in a system, this can have a very noticeable impact on performance. Some solutions include manually going through all items and removing duplicates (passing over the already collected data to the remaining item), or using automatic methods such as a thesaurus, or using dimensionality reduction techniques, that essentially try to group together similar products to construct recommendations [8].

In any dataset, some users exist that have little similarity to any group of people. They are called gray sheep if they sometimes agree and sometimes disagree with any existing group, and black sheep if they have no similarity to any group at all [8]. Even considering that for black sheep it is almost impossible to make recommendations regardless of the system used, hybrid recommendation systems can be used as a way of approaching the gray sheep problem, taking an optimal mix of content-based and collaborative filtering recommendation algorithms, which are explained in section 2.3.4.

Any dataset contains the problem of noisy data [8], which is meaningless or even detrimental data when used for data mining. In terms of this research, this can be defined as data not helping with creating accurate recommendations, or even detracting from them. For example, when using clustering, noise can be defined as any point not belonging to a cluster [14]. Most collaborative filtering algorithms have a way of dealing with noise. In this setting, the main noise will probably come from photos rated differently by the system than the user intends, due to the uniformity of the ratings applied to sets of photos, as explained in section 3.2.1.

Shilling attacks [8] are deliberate attempts by parties to influence recommendation systems by having certain items (their products) recommended more often than other items (those of competitors) [9,10]. With creating profiles and giving biased ratings, this often helps increase sales of their products. Both push attacks (upping ratings) and nuke attacks (lowering ratings) can be used [10]. The problem is that they often have to be detected manually, but research has been done that investigate the use of statistical metrics to detect these attacks [9,10]. This research will not go more in depth into this problem: this is not relevant to this research, as it takes a lot of effort by a user to increase their own rating, and it is assumed that they will not want to go through this effort.

2.2 Metrics

Evaluation of recommendation algorithms is usually done with either *coverage* or *accuracy* [1,8,15]. Coverage is the percentage of items for which the system can make any prediction at all [1,15]. The accuracy metric is subdivided into two subtypes: statistical and decision-support [1,15]. Statistical accuracy metrics are evaluation of the predicted values against the true values, usually by means of testing the trained algorithm against a test set or validation set. Decision-support accuracy metrics determine how good a recommender system is at predicting

high-relevance items [1,15].

The statistical accuracy metric that is very often used in the scientific world is the MAE (Mean Absolute Error), which for example is used in the references [1,2,8,15]. This one divides the total absolute error of all predictions by the number of predictions made. Another one is the RMSE (Root Mean Square Error), which is used for example in the Netflix Prize challenge [8,17,58]. It takes the total root of the sum of each individual error squared. This is like the MAE, but gives more weight to large errors [13].

Even more important metrics for this research might be the ones falling in the decision-support accuracy type discussed next. Classification accuracy is the percentage of examples correctly classified as having a rating either negative or positive as seen from the average rating [16]. Recall is the percentage of positive examples classified as positive, and precision is the percentage of examples classified as positive that are actually positive [16]. Because often an increase in recall leads to a decrease in precision and vice versa, another metric resembling an optimized mean between recall and precision called F-score, F-measure or the F1 metric is often used [1,25]. Here especially precision might be important, because the recommendations should be as good as possible.

The ROC (Receiver Operating Characteristic) curve is a graph in which the recall and inverse recall (true positive rate and false positive rate) are plotted against each other. These curves can be used to explore trade-offs between true positives and false positives [1,8,15].

2.3 Classifying recommendation methods

This section discusses the four different classes of recommendation methods: active and passive systems, explicit and implicit measurements, memory-based and model-based algorithms, and collaborative, content-based and hybrid filtering systems [1,3,8].

Recommender systems have been researched since the mid-1990s [1], when research started explicitly on finding recommendations based only on ratings of items by users. Research in recommendation algorithms borrows heavily from the fields of information retrieval and information filtering [1,3]. With the introduction of the computer into workplaces, companies have been able to easily store large amounts of customer data. This data was reason for some to look into whether they can be used to improve sales. This led to even more research in recommendation algorithms.

Different kinds of recommendation methods exist. Whether recommendations are user-specific or general (active versus passive recommendation systems), the way ratings are gathered (explicit and implicit measurements), the way recommendations are created from the data (memory-based against model-based algorithms) and what kind of data is used (collaborative, content-based and hybrid filtering) all lead to different sorts of approaches to recommendation algorithms. This section discusses each briefly, together with the relevancy for this research's

specific recommendation problem.

2.3.1 Active and passive systems

The Active filtering systems make user-specific recommendations [4]. They analyze the user's behavior and make recommendations based on that specific user's preferences and past behavior. The advantages are of course user-specific recommendations, with the disadvantage that the algorithms needed to implement this are heavier computation-wise, the user will need to be identified by the system and the system will need to have enough information about the user (for example ratings) to make a good recommendation.

Passive filtering systems, such as Amazon recommending items that other customers bought together with an item that the current customer intends to buy, make general recommendations for its users [4]. They take the available data of all users and make recommendations based on for example the average ratings and (current) popularity of items. It does not matter to the system who the current user is, its recommendations will be the same. The advantage is that they are easy to implement and do not have a lot of the problems that active systems face. The disadvantage is of course that they do not take into account specific users and their specific preferences and past behavior. These systems are often used for websites where no user-specific rating information is (yet) available, or social news and entertainment sites such as reddit.com.

The focus of this research is active filtering systems, where user-specific recommendations are made. Passive filtering could in theory be used when not enough information is available to make use of active filtering, but that is not relevant for this research.

2.3.2 Explicit and implicit measurements

Two types of user behavior measurements that can be used are implicit and explicit measurements [4,5]. The difference between each is in the way ratings of items are obtained from the user.

Explicit measurements are done by requiring direct user input [5]. When users rate an item, they explicitly tell the system what they think about it, which can often be used to make reliable predictions. This will ensure a high cost to the user [6,7], and benefits are not always apparent [6].

When users browse a website, they also give a lot of information about themselves and their preferences and behavior that they do not directly input into the system. Some of the important implicit measurements can be used to make implicit ratings for creating recommendations. Statistics such as number of mouse clicks, time spent on a webpage, or scrolling on a webpage can be used, of which time spent on a page and the amount of scrolling appear to be good indicators for interest [5]. When proper implicit ratings can be obtained, they can help circumvent the problem of users saying something different (explicit ratings) from what they actually want

[5].

The problem is that implicit ratings are more difficult to work with than explicit ratings; and they are not suitable for every recommendation system. It could work for a recommendation system for a news website, where users read a lot of articles and only rate a few, but probably not as well for a system where a user has to rate every item that he or she sees.

Both types of ratings could also be combined, perhaps leading to an effective answer to the sparse data problem for collaborative filtering that states that it requires a certain number of ratings for every item and user to provide accurate predictions [5,7].

This paper will focus on explicit measurements, as the Paiq dataset consists of only explicit measurements.

2.3.3 Memory-based and model-based algorithms

Looking at memory-based algorithms (sometimes called “heuristic-based” algorithms [1,3]), they use the entire user-item dataset every time when creating predictions. A couple of advantages of memory-based techniques are easy implementation when designed for use with small data sets, new data can be added easily and incrementally, and they scale well with items that have been rated many times.

Model-based algorithms instead use a training subset of the dataset to create a model of the user ratings and a test set to check its validity. When creating a recommendation, the model is used instead of the entire dataset, most of the time leading to faster prediction times using less memory. Some other advantages of model-based algorithms are less issues with scalability, and sparsity problems [3] as discussed in the problems section in the previous section. Some algorithms create all sets of predictions all at once, while most memory-based algorithms only create a single prediction per run or only for one user. Model-based algorithms could lead to a reduction in quality of its predictions as opposed to memory-based ones, often trading performance for scalability. However, model-based algorithms can often offer better generalization of the training data, so that they sometimes perform better than memory-based algorithms. Because both could be valid options, both will be discussed later on.

2.3.4 Collaborative, content-based and hybrid filtering systems

The two types of filtering recommendation systems differ in the focus and the (amount of) information available for the system to base its recommendations on. Collaborative (sometimes called user-based) filtering [1] is based on finding similarity between users, based on rating information. If similarity between some users is high, collaborative filtering supposes that the rating of users with high similarity says something about missing ratings, which should also be rather similar. The user is recommended items that people with similar tastes and preferences liked in the past.

Content-based (sometimes called item-based) filtering is based on similarity of items, which themselves often consist of textual information [3]. The recommendation algorithms take information from the content of items and make recommendations based on the similarity of them. It looks at items that the user rated highly in the past and then finds items that can be seen as similar. For a movie recommendation, such an algorithm could consider information such as its director, starring roles and genre. Some of the challenges of content-based recommendations are limited content analysis (because of limited keywords), overspecialization and new user problems [1,3]. Also, more information is needed on the items, and for every item either a way of automatically inputting that information has to be found, or that information has to be input into the system manually. If a lot of information is known about users instead of the items, that information could be used instead of the information on items.

Hybrid filtering combine the collaborative and content-based recommendation methods as a way of avoiding the disadvantages of both [3]. It may mean combining the results of separate methods, adding characteristics of one to the other, or implementing a method that uses characteristics of both [1]. Hybrid filtering techniques could overcome the disadvantages of both collaborative filtering and content based recommenders, but are a lot more complex and expensive in implementation.

For this paper, the assumption is made that there is only rating information available, which means that the collaborative filtering methods will be looked into. Both content-based and hybrid recommendation methods will not be considered in more detail in the remainder of this paper.

2.4 Specific recommendation approaches

This section will detail a few specific recommendation algorithms that are likely to be suitable for tackling the dataset. These algorithms will be suitable for active systems, use explicit measurements, and will use a collaborative filtering method because of reasons that are explained in the problem statement. Both a memory-based and model-based approach will be looked at.

2.4.1 Selection criteria

The selection criteria on which the choice for a specific recommendation approach is made are:

1. Use in the domain
2. Relevance for problem statement

Something to keep in mind when selecting approaches is that simple collaborative filtering algorithms can be almost as effective as the best ones when grading them in terms of utility in certain restricted settings [1,19], and that getting the basics right is probably at least as important as tweaking the models [27].

Three of the better known models used in recommendation approaches, as well as being among the most important ones for the winner of the Netflix challenge are the neighborhood-based kNN,

the matrix factorization-based SVD and the neural network-based RBM [52,69]. They can be seen as some of the better approaches to both memory-based and model-based, and when combined yield even better results [17,27]. Both will be discussed separately below as well as a way of combining their results. Specific attention will be given to variations of the algorithms that help improve the data scalability and sparsity problems that will probably occur when applying these approaches to the Paiq dataset.

2.4.2 Nearest neighbor-based approach (kNN)

Good examples of memory-based recommendation algorithms are the collaborative filtering algorithms that check all users for similarity compared to the one the prediction is made for, and then use nearest-neighbor techniques to create a recommendation [4]. The name “nearest neighbors” applies to the users that are most similar to the user the recommendation is calculated for, and the *k* stands for the number of closest neighbors that are used in creating the prediction. A distance measure is used to instantiate the similarities between users. Most of the time a *k* nearest neighbor (kNN) approach is used, but sometimes a threshold-based approach is used instead, where all the users with a similarity above a certain value are taken when calculating a prediction [50]. These algorithms are easy to implement and make sense logically: the group of people with in average a voting behavior in the past similar to a user will probably have a voting behavior similar to that user in the future as well.

2.4.2.1 Calculating predictions

The input needed for using kNN to predict user ratings are existing user ratings of items, where for each rating its corresponding user and the item are known. Calculating predictions for users consists of these steps [21,28,34]:

1. Normalize the rating data
2. Calculate the user’s similarity ratings with other users
3. Select a subgroup of those other users to create the recommendation with (the *k* nearest neighbors)
4. Calculate the interpolation weights of the other users
5. Calculate a prediction by taking a sum of weighted ratings from the selected subgroup of users

The way these steps are filled in and which algorithms are used exactly can vary per implementation. This means that for an actual implementation, for each of these steps choices have to be made: how to normalize the data, choosing which similarity measure (distance of the user to neighbors) to use and how many of the closest neighbors are involved in calculating similarity ratings, and the way that the interpolation weights (how much weight is given to each individual neighbor for calculating the prediction) are determined. The first normalization step involves removing certain data-skewing effects such as some users having a higher or lower average rating than another user, while still having the same preferences towards items. This can also prevent some (extreme) users from being weighted too heavily [28,34]. Calculating the interpolation weights (which sum up to 1) also involves doing a normalization on the influence of

other users on the final ratings, preventing some extreme users or users that have a large number of ratings from weighing too heavily on the final result [28,34]. The similarity ratings are often recalculated once in a while to new include new rating information.

The two simplest ways in which similarity is calculated between users are the weighted sum and the adjusted weighted sum [1]. With the weighted sum, the similarity is calculated as sum of all distances between the ratings of all items two users have in common. The adjusted weighted sum takes into account that users can use ratings differently, which the weighted sum does not: one user could for example consistently rate items a point lower than another user, and adjusts for this by taking instead both distance measures from the average ratings of the items they have both rated. This is the reason the adjusted weighted sum is widely used instead of the original weighted sum, as it also presents a way of normalizing the data [28,34].

A few other commonly researched ways of measuring similarity are [1,22,23,46,48]:

- Mean squared differences algorithm
- Cosine based algorithm
- Pearson r algorithm
- Constrained Pearson r algorithm
- Item-item (or artist-artist) algorithm

The computation of correlation in correlation-based approaches such as kNN is $O(m^2*n)$ [25,41], where m is the number of photos and n the users.

The mean squared differences algorithm calculates the mean squared differences between two users by looking at the items both users rated. This is a modification of the weighted sum algorithm, putting greater emphasis on the magnitude of the errors by squaring the difference in ratings between two users. After calculating all differences for one user and all other users, a threshold is then set where all other users with greater dissimilarity are discarded. The inverse dissimilarity scores of the remaining users are used as weights to calculate the prediction.

Calculating similarity with a cosine based algorithm treats two users as vectors in a (number of items rated by both users)-dimensional space, where the similarity between the two users is obtained by calculating the cosine angle between the two [1]. It is a variant of the inner product which is a standard similarity calculation and is also often used in collaborative filtering, but some research suggests that correlation based similarity algorithms perform better than similar cosine based ones [47], probably because cosine similarity is not invariant, unlike the correlation-based one described below.

The Pearson r algorithm calculation is another variant of the inner product calculation and effectively builds on the cosine based approach. This algorithm had already been developed in the 1880s by Karl Pearson as a measurement of the strength of linear dependence between two variables. It is called a correlation-based approach and uses a Pearson r correlation coefficient, now for measuring user similarity. Each calculated user similarity is a number between -1 and

+1, implying either negative or positive user similarity, or no similarity at all. One important property of the Pearson correlation is that it is invariant: if all values scale or shift (for example x to x+1), other similarity algorithm values would change, but the Pearson correlation stays the same. This is good because similarity looks at users rating items in the same way, not that they are the same in an absolute manner.

The above Pearson r algorithm uses both positive and negative similarity when calculating a prediction. Positive similarity between two users implies that both will like and dislike about the same items (e.g. they both like horror movies and dislike other movies). However, this does not imply that a negative similarity between two users means that one likes what the other dislikes and vice versa, but only that what one likes, the other might not like (e.g. one likes horror movies but dislikes other movies, while the other user likes action movies and dislikes other movies, meaning both users dislike movies such as drama and comedy). The constrained Pearson r algorithm was thought up to take this into account: only when both users rate an item either negatively or positively the similarity will change, with the change being an increase. This also leads to only positive similarity ratings. Some research suggests that placing a restriction on similarity scores between users to only be nonnegative actually improves prediction accuracy [28,34], making this one better than the regular Pearson r algorithm.

When making a prediction, the item-item algorithm looks at the other items a user has rated and looks at how much these items resemble the item the prediction is made for. This method was considered as an alternative because with most datasets the number of users far outweighed the number of items, meaning that looking at item-item instead of user-user relations could be more effective because on average a single item would have more ratings than a single user and thus contain more information for making predictions. This approach uses one of the other algorithms (e.g. item-item correlation or cosine similarities between item vectors) to calculate actual similarity [35]. Scientific literature is not conclusive about whether it is clever to look at item-item algorithms: either well designed user-user and item-item algorithms will have equivalent performance [23], or item-item algorithms might be (significantly) more effective than user-user algorithms [34,35]; and item-item based algorithms seem to scale better to large datasets [35]. The reasons given for this better performance is: that there are typically many more users in the system than items [24,34]. The Paiq dataset has more items than users, as almost every user will have uploaded at least a few photos. Therefore in this case, primarily looking at user-user relations is probably the more suitable approach.

Just like with many other recommendation algorithms, it seems that when looking at results there is a tradeoff between the accuracy of predictions and the number of predictions that can be calculated by taking a smaller or greater number of nearest neighbors for creating the prediction [22]. This difference can be influenced by increasing or decreasing the threshold of the similarity ratings of the included users.

2.4.2.2 Advantages and disadvantages

As mentioned before, the advantage of using kNN for making recommendations is that it is

easier to implement than most other recommendation algorithms that fall within the scope of this research. Furthermore, the workings and underlying mathematics are easy to understand, and new data can easily and incrementally be added. Also, this approach scales well with items that have been rated many times.

A problem can be found in the calculation times that a large and increasing dataset will cost, as nearest neighbor approaches often have issues with scalability and sparsity of data [1,2,28,34]. Scalability is a computational cost issue, as the memory requirements become big by having to store all the data, or else having to calculate similarities at run-time takes a lot of time, as does updating the relevant similarities every time a user/item or even a rating is added. The sparsity problem is one of the main disadvantages of kNN: it is hard to find users that have the same items rated for users that have not rated a lot of photos, impacting performance.

Another disadvantage is that there is a chance of overfitting when tuning the algorithm that it becomes even better at predicting past behavior (data from the training set) but worse at predicting future behavior (tested with the test set).

2.4.2.3 Variations

There are some ways of making kNN more efficient in order to deal with scalability and sparsity issues, the first of which can be done by preprocessing the dataset by using some kind of dimensionality reduction technique [29,31]. These algorithms try to quickly determine (in the case of this research) which users belong to the group of *k* nearest neighbors, some of them finding approximations, reducing the computational cost by a large amount [31].

Scientific research on preprocessing the dataset by applying clustering techniques that already exist to kNN recommendation methods was done to examine their effect on computational cost, and sparsity and scalability issues, with the aim of decreasing them [35,49,50,55,56,64,65,67]. Clustering groups sets of users (or items) in such a way that calculating similarity between users is divided into a lot of smaller problems, in a divide-and-conquer kind of manner. This makes them easier to compute as compared to the biggest computational step, helping with the scalability problem. This then means that when determining the similarity, only the similarity with the users belonging to that group has to be calculated, and not with every other user first. The problem of sparsity is decreased because for users with a small number of ratings, with clustering the number of other users that can be counted as neighbors will likely be increased.

The simplest and most computational cost-efficient way to calculate predictions with clustering is to just take the average of the cluster as prediction [35]. Clustering can be applied when preprocessing the ratings dataset for use in kNN to help deal with the sparsity and scalability problems [49,50]. A method is to first cluster users based on their ratings, and calculate a cluster center for each cluster of users. Similarity can then be found easily by looking at the distance of the user to the cluster center. A similar item clustering collaborative filtering algorithm is then used to calculate recommendations, decreasing the computational cost for computing recommendations because otherwise all predictions would have to be calculated [50]. This also

helps in combating the sparsity problem, and is one of the approaches most suitable for the Paiq dataset in dealing with the scalability problem, yet the sparsity problem makes clustering difficult.. Calculating the clusters themselves can take some time, but can in principle be done on a subset of the dataset. For users and items not in this subset, their place in the clusters can be determined during runtime, provided that they have rated some same items or users.

Some newer clustering approaches try to find user-item subgroups or subclusters, in line with thinking that some users may agree on a subset of items, but totally disagree on another subset of items [65]. Yet another somewhat similar approach first tries to find similar users and items by using a spectral clustering technique, and then using an iterative process to calculate the recommendation [64]. This technique uses eigenvector calculations to cluster the ratings, after which a prediction is calculated by using both. To illustrate the effectiveness of approximation methods, even though standard spectral clustering has a computational complexity of $O(n^3)$, in more recent years an iterative *k* means approximation method has been proposed to lower it to $O(k^3) + O(knt)$ [66]. The *k* stands for the number of clusters, and *t* for the number of iterations. This was done to lower its computational cost so that spectral clustering becomes viable for huge datasets. Apart from this research, a lot other research is and has been done on approximate clustering [67].

A lot of scientific and other research has been done into data structures to optimize different ways of searching through data. These found their way into recommender system research as a means of making data lookup quicker. Data structures such as kd-trees, vp-trees, mvp-trees, sphere/rectangle (SR)-trees, metric trees or ball-trees can be used to lower the computational cost of the nearest neighbor step of calculating user similarity, but these are only efficient up to a moderate number of dimensions, about 20, which is totally inadequate for the dataset in this research [30,32]. Options for high dimensional space are Locality Sensitive Hashing (LSH) and spill trees, including optimizations such as random projection, aggressive pruning and redundant search [29,30,33]. An advantage of spill-trees are that they are exact nearest neighbor approaches, whereas LSHs are approximations based on probability [30,33]. Distance-Based Hashing (DBH) is based on LSH, with its main advantage over LSH that DBH can be constructed in any space, while LSH can only be applied when locality sensitive families of hashing functions exist, which also shows its relevancy for this research [32].

All in all, kNN models are a good approach for recommendation algorithms, but can mainly have scaling issues and problems with sparse datasets.

2.4.3 Singular value decomposition based approach (SVD)

Singular value decomposition (SVD) started taking form in 1873 after decades of related research [43], but was first used in the early 90s as a dimensionality reduction technique for finding relevant text documents in the field of information retrieval [39,41]. Near the end of the 90s, this feature extraction technique was starting to see use in recommendation approaches working with ratings [1]. It was first tested to try to deal with some of the sparsity, scalability and

synonymy problems of other collaborative filtering mechanisms such as neighbor-based approaches: these methods often only calculate similarity between users when these users have rated the same items while with SVDs some users can be considered for similarity calculations even though they have no overlap in ratings of items with other users [2,25,38]. Furthermore, SVDs have a very fast online performance [2,25]: calculating the actual recommendations is called the online part and all calculations done before that (which is mainly calculating the SVD) is called the offline part. The SVD approach gained another increase in popularity during and after its successful use in winning the Netflix challenge.

Singular value decomposition is very closely related to principal component analysis [36] on a data matrix (the other being eigenvalue decomposition [2]). It can be used as a feature extraction, matrix factorization/approximation or dimensionality reduction technique [1,2,20,25,39], which means finding the smaller dataset where the other data derives from, or approximating the original matrix by factoring it into smaller matrices that approximate the former matrix when the factors are multiplied, or reducing the number of dimensions of a matrix (by doing the matrix factorization/approximation). This approach is a model-based approach, also sometimes referred to as latent factor model approach [24], as instead of looking at relationships between either items or users, it also looks at latent relations between users and items and transforms both to the same space so they become directly comparable [24,26]. This makes cross-comparisons possible, and thereby should reduce the sparsity problem present in other collaborative filtering techniques.

2.4.3.1 Calculating predictions

Calculating predictions using SVD requires having a set of user ratings of items where for each rating its corresponding user and item are known, similar to kNN. The steps required for obtaining the predictions are [25,39]:

1. Obtain the user-item ratings matrix *M*
2. Fill in the missing (empty) ratings of matrix *M*
3. Factorize/decompose this matrix *M* into three matrices: $M_{m*n} = U_{m*r} * S_{r*r} * V^T_{r*n}$ where *S* is the singular-value matrix
4. Calculate the predictions by finding the best lower-rank approximation matrix M_hat of Matrix M, for a number of values of k:
    a. Obtain singular-value matrix S_hat from S by discarding the last r-k singular values from S, where k < r
    b. Compute the lower rank approximation Matrix M_hat from the resulting S_hat: $M\_hat_{mxn} \sim= U_{n*(r-k)} * S\_hat_{(r-k)*(r-k)} * V^T_{(r-k)*n}$
    c. Test on a test set whether the resulting Matrix M_hat has better predictions than another computed Matrix M_hat with different rank
    d. Repeat the above sub-steps until the best lower-rank Matrix approximation M_hat can be selected, taking its now filled in entries as predictions

Just as with kNN, the manner in which these steps are done influence its final prediction results, often with tweaks made to steps to better suit the dataset they are used for. The choices that

have to be made here are: how to fill in the missing ratings of the matrix and choosing the metrics that determine which rank-r approximation will be the best one. Some of the more advanced and alternative options for using SVD will be discussed further below.

To obtain the user-item ratings matrix *M*, take the user-item ratings and input them into a matrix, with the users as its rows and the items as its columns, and the ratings as its entries. With a large number of users and items this can turn into an enormous matrix, thus the computation of the decomposition of the matrix will require a high computational cost (in both processing power and memory) [2,25,28,37] and is by far the costliest step of the approach. Time-wise it takes in the order of $O((m+n)^3)$, making classic SVD suffer from high scalability issues too [37]. On the other hand, once the matrix has been decomposed, it is very simple computation-wise to obtain the lower-rank matrix approximations, find the best one, and use it to gather recommendations, as stated further above [2,25]. Also, because the lower-rank matrix approximations create a full matrix with ratings and predictions, the predictions are not calculated one at a time as with kNN, but all at once. This means that storage-wise SVD is more efficient, where only the reduced matrices have to be stored with a storage cost of $(m*r)+(r^2)$ as opposed to $m*n$ for correlation-based approaches [25,39].

Another surmountable disadvantage of computing an SVD is that by definition there can be no empty entries in the matrix, requiring the filling in of missing ratings. This often happens either with a zero (which will make for more inaccurate predictions), or the corresponding (normalized) average user-rating or item-rating (corresponding row or column average) [2,25,28], of which using the item-rating seems to work better [25]. A more advanced way is using a combination of the global average and a deviation for both the item and the user [23,24], of which a variation is using the product average as a rating and then subtracting the user average, which is also a normalization technique [25]. Taking the user average is not relevant for the Paiq dataset, since all average user ratings will be somewhere around the average rating because of the way the ratings are calculated (see problem statement). Another manner suggests performing SVD iteratively while computing the missing values based on prior iteration results [25,45], but there is a chance of the imputations distorting the data, especially with sparse datasets, and it is still very expensive computation-wise and impractical for very large datasets [24,28,51].

Calculating the SVD itself can be done by reducing the filled ratings matrix to a bi-diagonal matrix, for example with Householder reflections, and then computing its SVD with an iterative method [68]. An advantage of only wanting to compute lower-rank approximations is that only the SVD up until a certain rank (the one that has to be tested) will have to be calculated. It basically boils down to finding the matrix factorizations that, when multiplied, have the smallest possible (mean squared) error between the original matrix and the newly created matrix approximation. Further information, discussion and mathematical background on SVDs can be found in [39,40,68].

Utilizing the SVD approach involves calculating the best lower-rank approximation from the full-rank matrix [1,2,36], removing unrepresentative or insignificant users (noise) in the process

[8,39]. As shown above in the steps needed to find the best lower-rank approximation matrix M_hat, this has to be done iteratively, i.e. the best rank cannot be determined beforehand. Choosing a different rank can significantly impact the general accuracy of the predictions [2,25]. The best rank is high enough so that all the important information is captured, but low enough so overfitting is prevented. A good lower-rank approximation of the data has a higher chance of being a better generalization than a medium-rank approximation that is a better reconstruction of the data, because of user fickleness. This can also mean that the lower-rank approximation matrix often better resembles user behavior than the original matrix itself [37,41].

Finding the best rank usually involves starting with rank 2 (k = 2) and moving up one in rank with each new iteration until the performance clearly starts getting worse instead of getting better. Usually the Frobenius norm $||M - M\_hat||_F$ (the difference between the square root of the sum of the absolute squares of the elements of the two matrices) is minimized to determine which matrix approximation *M_hat* is the best lower-rank approximation of the original ratings matrix [2,25,28,37]. This is called the Eckart-Young theorem [43].

2.4.3.2 Advantages and disadvantages

There are a couple of main advantages to using SVD approximation for recommender systems. The SVD approach better handles the online part of making predictions than kNN, making for faster predictions and better scalability for the recommendation part. That the SVD method can make use of latent relations between users and items helps with the sparsity problem and synonymy problem (where differently named and stored items essentially refer to the same item).

The SVD method also has a few drawbacks. The exact implementation details of the SVD are a bit harder to explain, especially in layman's terms. Another disadvantage is that computing the SVD requires a high computational cost in both processing power and memory, as a huge matrix has to be decomposed. This is only in the offline part, which can be done before putting the system online. Furthermore, a process has to be used to fill in the unknown matrix entries, resulting in extra computational time and complexity. This could mean that in this research' situation, the SVD is not a very good alternative for kNN after all.

2.4.3.3 Variations

Variations on SVD focus mostly on dealing with the heavy offline computational cost of SVDs or the filling in of missing ratings.

One way of making the matrix decomposition easier that is often applied is by using a reduced matrix instead of the full matrix, which is often sufficient in statistical applications and in this case for calculating predictions. The resulting SVD is called a thin SVD instead of a full SVD, which refers to the matrix decomposition of the original ratings matrix [2,12]. Only the n column vectors of U that correspond to the row vectors of V^T are calculated: the other column vectors are discarded. This leads to the matrix decomposition: $M_{m*n} \sim= U_{m*n} * S_{n*n} * V^T_{n*n}$. Calculating this thin SVD is a lot quicker than calculating the full SVD when $n << m$, meaning when the number

of items is a lot smaller than the number of users.

Calculating the SVD is the most costly step. When new rating, user and item data becomes available, it would normally have to be added to the original matrix, and the SVD would have to be recomputed to incorporate this fresh data into the predictions. Some research has focused on ways to avoid this costly computation by updating the SVD with the new data instead of recalculating it [2,12,37,39,41,44]. One method proposes updating the SVD by doing rank 1 updates, where one single column is modified or added to the original matrix per update [2,12,39]. These are fast because they use small-matrix operations. Another method is the SVD-update method, where orthogonality is retained [37,41,44]. A simpler version of the SVD-update is the folding-in technique, where new users and ratings are folded into the existing SVD [37,41], which is claimed by [37] to be slightly faster and more accurate than the rank 1 update method, and by [44] to be a lot faster but more inaccurate than the SVD-update. All these SVD update methods can be (slightly) inaccurate, and each additional update will increase the overall inaccuracy, so that its users should be wary of deviations too large from the predictions were the SVD to be derived in the standard way [37]. Most methods have a tradeoff between accuracy and computational cost, but their exact effects on those metrics can often only be seen after applying the methods.

An alternating factorization approach is suggested in [2], which tries to find two factors that best represent the original matrix by alternatingly updating each factor and thereby minimizing the error between the original matrix and the multiplication of the factors. Its researchers claim this approach has better accuracy than the SVD-update technique, but it has problems with a large number of missing ratings.

Newer research to SVD has focused on avoiding having to fill in missing ratings to make the rating matrix dense, because of the added computational cost and data distortion that might happen with inaccurate methods of data imputation. Instead, only the available rating data is used, while avoiding overfitting by using regularization [24,28,42,51,69,71]. These approaches minimize the error of the factors versus the original matrix. One of these approaches is using an alternating-least-squares (ALS) method [24,51]: systems that can use parallelization and systems centered on implicit data [51].

Another newer SVD approach is the in the Netflix competition often used stochastic gradient descent, brought to attention by Simon Funk [24,51,70]. This method loops through all ratings and for each rating predicts a value, calculates the error of the prediction compared to the actual value, and adjusts the factors by a magnitude proportional to a certain chosen value times the opposite direction of the gradient. The main advantages of this method are its easiness in implementation and relatively fast run time, and it is widely used in the Netflix competition over more conventional SVD methods [51]. Good results are reported with optimized parameters.

2.4.4 Restricted Boltzmann machine based approach (RBM)

The third and final method that is looked at in this literature chapter is the restricted Boltzmann machine algorithm (RBM). The RBM was originally invented by Paul Smolensky [76]. It became more widely used in the 2000s, when Geoffrey Hinton and others thought of ways to massively increase its learning speed. This model-based approach has been used for problems such as dimensionality reduction, classification and recommendation algorithms and as a stackable basis for deep belief networks (DBNs) [75,77]. It was one of the algorithms used in the winning entry for the Netflix Prize, where the winners lauded the algorithm for its accuracy and its relative tolerance for different parameter settings [17].

The RBM is a generative stochastic neural network (consisting of bidirectionally connected stochastic units). The network consists of visible and hidden units separated into two layers, with weights in-between, and biases (offsets) linking to all units. A Boltzmann machine is a neural network with the restriction that it has only one layer of hidden units. And a *restricted* Boltzmann machine refers to there being no connections in between visible and hidden units [73], making them independent from each other, as shown in figure 2.4.4.a. Both reasons make learning for the RBM easier and less time consuming than for other more elaborate neural networks [75], as they are not recurrent and thereby having more efficient training algorithms available for them.

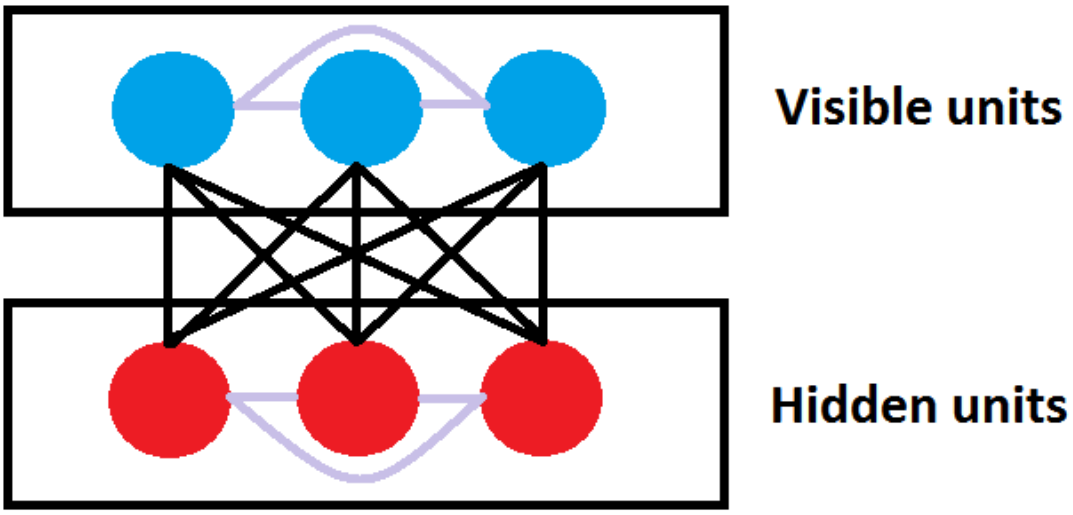

Figure 2.4.4.a: RBM, with the existing connections in black, and the lost connections in grey.

2.4.4.1 Calculating predictions

Calculating predictions for users with RBM is done with the following steps [73,74,75,78,79]:

1. Initiate the weights and biases
2. Choose a number of training epochs and do contrastive divergence (CD) to train the weights:
    a. Initialize the visible units with the given ratings from a training case
    b. Calculate the probabilities of the hidden units
    c. Calculate the probabilities of the visible units, given the calculated probabilities of

the hidden units
    d. Calculate the probabilities of the hidden units again
    e. Repeat steps c and d for each rank of CD
    f. Adjust the weights of the connections where the visible units have changed
    g. Adjust the biases of the visible and hidden units
    h. Repeat the above sub-steps until all training cases from the training set have been used for training
3. Calculate the predictions (the online part) for a user by:
    a. Setting the visible units for the user
    b. Doing steps 2b and 2c once
    c. Use those calculated probabilities to get the score as the prediction

The RBM has two layers of units: a number of visible units and a number of hidden units. All units are binary, meaning they can have one of two states: 0 (off) and 1 (on). Each visible unit represents an item, so the total number of visible units is equal to the number of items. The hidden units make up the latent factors the network tries to learn, and these will have to be set semi-arbitrarily and the RBM tested to find their optimal amount.

Each user has one RBM and makes up exactly one training case, but each RBM has the same number of hidden units and the weights are shared between all RBMs. The weight sharing means that when two users have rated some similar items, the relevant weights will be used for the RBM corresponding with the first user, and after that the same weights will be used for the RBM corresponding with the second user (for the ratings of the similar items). This allows the RBM to learn weights from a different network each time, but remembering the weights from when they were last updated [73]. Each unit also has an associated bias that contains generalized information about it (the bias can be seen as belonging to a unit, or the bias can be seen as a separate entity with a separate connection for all units). The weights and biases are initialized in semi-random fashion to avoid local minima.

The recent widely accepted way of training an RBM is the CD method, an approximate gradient descent based method [73,74]. The visible units of the network are initialized with a training case from the training set, making sure that the chain will already be close to having converged to its final distribution. Then, *k* steps of Gibbs sampling (with a small *k*) are performed and the results are used to train the weights and biases. The *k* stand for the rank of the CD, which is generally written as CD-k.

Gibbs sampling consists of steps 2b to 2e in the above mentioned steps for calculating predictions. The probabilities of the hidden units are calculated using the entries of the training case, their biases and the connection weights. Then, the hidden units are set and the visible units are calculated again using the states of the hidden units combined with their biases, and connection weights (the reconstruction step). Then the hidden units are calculated yet again. The last two steps of calculating the visible units and hidden units are performed for each rank of CD [73,74,79].

RBM works with an energy function. The activation energy for a unit is the weight of all connections that the unit has with other units from the opposite class (for hidden and visible units). While calculating the activation energy, if there is a positive relation between two units, one unit tries to get the other unit to share its state. Where a negative relation between the units is present, one unit tries to get the other unit to take on its opposite state [73]. The total energy of a state is calculated by subtracting the weights of the connection between the visible and hidden units with the bias of the visible nodes and the bias of the hidden nodes.

The activation rule is the rule that determines whether a unit is activated (set to 1 instead of 0). A unit is activated when its energy is over a certain threshold. The probability of this happening is computed by calculating the sigmoid $1/(1+e^{-t})$, where t is the sum of the biases of the hidden and visible units and the weight of the connection times the unit that the state is calculated from [78,79].

The weights are updated in the following way [73,74,75,79]: first we have a measurement for each relevant hidden-visible unit combination, whether they are both on or both off (similar), or one is on and the other is off (dissimilar). This measurement is done twice during the CD: once the first time the probability of the hidden units are calculated (called the positive phase), and once after the last time this is done and CD is over (called the negative phase). The measurement of the positive phase is added and that of the negative phase is subtracted from the weight (modified by a learning rate). Thus, we reinforce the association between the units that we want the network to learn, while we diminish the association between the units that the network itself generates [78,79].

And finally, creating predictions is done as explained in the sub-steps of step 3 in the above mentioned steps for calculating the predictions, which is basically another 1-step Gibbs sampling: CD-1. This can be done in time linear in the number of hidden units.

2.4.4.2 Advantages and disadvantages

The RBM has some of the same advantages as the SVD (specifically the Funk SVD), because it is also a model-based approach: it handles the online part of making predictions really quickly. Once the network is trained, the predictions can be created by just calculating CD-1. The rest of the advantages of being a model-based approach like SVD have already been discussed in the SVD section.

A disadvantage is that even with the CD approach, the offline part of RBM remains quite computationally expensive, especially with big datasets: the number of visible units grows with an increasing number of photos, as there is one visible unit per photo. Furthermore, the RBM is at least as difficult as the SVD to understand and implement.

2.4.4.3 Variations

One of the variations of is using softmax visible units for a (very) small amount of discrete

ratings higher than 2 [73]. With this approach, each visible unit is split up into x sub-units, where each sub-unit stands for a possible rating. This changes the amount of available ratings from 2 (1 and 0) to x. Each sub-unit has a different probability for whether it is turned on or not, and the sub-unit with the highest probability will be turned on. The number of biases for the visible units is changed to x biases, where x stands for the number of values of the softmax units.

Another variation is continuous RBM, which can model continuous data [72]. This replaces the binary units with continuous ones, and changes the energy function to incorporate the change.

Another is addition can be taking into account that users can have used items, for example watched movies, but not have rated them. In this case, the types of movies a user watches says something about the types of movies he or she likes [73].

Conditional factored RBM [73] is a type of RBM where the weight matrix is factorized, because a huge amount of visible units, hidden units and ratings make for a very big weight matrix. The factorization can remove possible memory issues. The advantage is that a lot more hidden units can be used.

2.4.5 Blending methods

As stated before, the kNN, SVD and RBM approaches all have a different specialization in the relations they take into account. They are 3 of the 4 main approaches of the Netflix prize winning entry [17], and SVD and RBM were the only two methods that were implemented by Netflix itself. The kNN approach looks more at localized relations and the SVD and RBM approaches are better at looking at global relations [24,28,69]. RBM also finds slightly different relationships than the SVD [73]. This makes it interesting to look at all three methods when considering which one to use. It also led researchers to look for ways to combine multiple approaches to increase accuracy, obviously at the expense of computational cost. The idea is that, like when asking more people for directions, calculating predictions in multiple ways add up to more predictions than just calculating predictions one way.

In [42], SVD is post-processed with kNN by using SVD to find similarity ratings for items and then using kNN to create predictions using those SVD similarity ratings, but their results were not encouraging: their research already found other methods that performed better.

2.4.5.1 Calculating predictions

The most basic method of blending predictions of different methods together to get one better prediction is using the mean of all obtained predictions. A better approach to combining models works by taking a linear combination of the predictions of both models [24]. Another uses a factorization approach with included an overweight for items similar to the one for which the prediction is made, those of which are derived with nearest neighbor techniques [28]. It is also possible to blend results by using an ensemble of multiple methods [17,27]. Here, a ratings vector is regressed on the predictors of those method, leading to weights for each predictor.

This approach also suggests that overtraining or overfitting is helpful when blending, without giving reasons why.

Other suggestions for blending are made in [52], which shows that linear blending is not optimal. Binned blending is a blending method that applies learners to structured subsets. Neural networks and gradient boosted decision trees work even better, of which the first performed best in this research, and the second was used in the winning entry of the Netflix challenge entry [27].

2.4.5.2 Advantages and disadvantages

The accuracy of the recommendations can be improved beyond what any single approach would achieve, but this means an increase in complexity and computational time, because the predictions of multiple recommendation algorithms have to be calculated and then combined in a simple or more convoluted way.

3 Experimental setup

This chapter discusses the setup of the experiment. It starts with the research questions. Next, information is given about the dataset that is used by the recommender algorithms to make recommendations. The metrics that are used to compare the implemented methods are detailed, with the error and the baseline methods. It finishes with relevant implementation details of the methods used for the experiment: baseline methods, nearest neighbor, singular value decomposition and restricted Boltzmann machines.

3.1 Research Questions

Looking at the challenges and assumptions made in the introduction and the dataset problems and recommender approaches researched in the literature study, there are a couple of research questions that this research will address:

R1. Which of the researched machine learning approaches or baseline methods is best for creating recommendations on the Paiq dataset considering accuracy?
    a. Which of the researched machine learning approaches or baseline methods is best for creating predictions on the higher rating range, considering accuracy?
R2. Which of the researched machine learning approaches is best for creating recommendations on the Paiq dataset considering computational and memory costs?

The metrics used to compare accuracy are discussed further on in this chapter. The next chapter on parameter estimation will outline how validation of the parameters was done. The results and discussion chapter after that will discuss the MAE on the test set as well as the MAE of the methods on different rating ranges taken together in steps of 200, starting with a rating of 1 up to a rating of 1000. The computational and memory costs of the algorithms are discussed in this chapter as well.

3.2 Dataset

This section elaborates on details of obtaining the final Paiq dataset and the splitting into training, validation and test sets for the experiment.

3.2.1 The Paiq dataset

The final Paiq dataset consists of subjective ratings of photos by users. Some photos have been rated only a few times, and others have been rated many times. Some users have rated only a few photos, and others have rated a lot of photos, as shown in the chart below. The number of photos (83,513) exceeds the number of users (30,598) by a factor of almost three, making for an enormous dataset would all items be rated by all users. This used data is just a subset of the entire Paiq dataset, which as per 10-03-2014 contains 99,344,346 ratings from 164,323 users

about 414,813 photos. Making recommendations for this dataset means having to deal with some problems: a gigantic, but extremely sparse, dataset with a huge number of missing ratings, far bigger and sparser (especially the combination) than most datasets used in scientific research done on recommender systems.

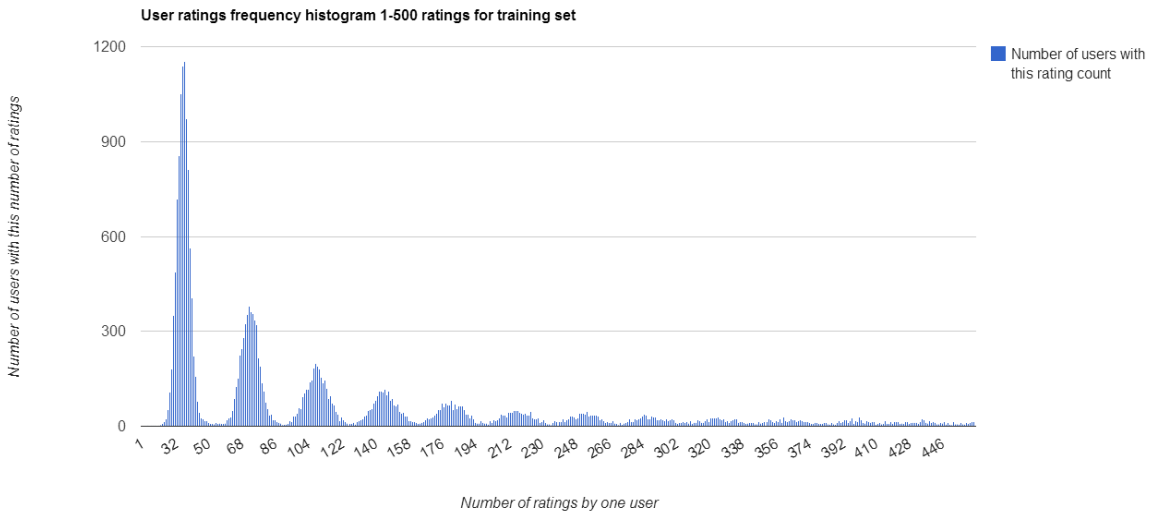

Chart 3.2.1.a: User ratings frequency histogram 1-500 ratings for training set

As chart 3.2.1.a shows, there is a peak at around 36 ratings by one user, and peaks again every next 36. A user rating set (one sitting of a user rating photos, as explained in the next paragraph) consists of 50 ratings, so most users have rated multiples of 50 photos. Of these ratings, 10% were semi-randomly taken for the validation and test sets, leaving on average 40 ratings per user rating set. After the original test set was lost, another 10% was semi-randomly taken away as part of a new test set, leaving on average 36 ratings per user ratings set.

The uniformity of the ratings might also influence the ratings themselves. This uniformity is because of the manner in which ratings are assigned between users and photos. When the users is done rating a user set of 50 photos, the system inserts uniform ratings from 0 to 1000 to the photos into the database for that user and those photos depending on the order of the photo set.

The distribution of photo ratings across the ratings as shown in chart 3.2.1.b is expected: the average rating of most photos is close to the global photo average of 501, and the count goes down as the line moves from the global photo average towards either end of the graph, with ratings 0 and 1000.

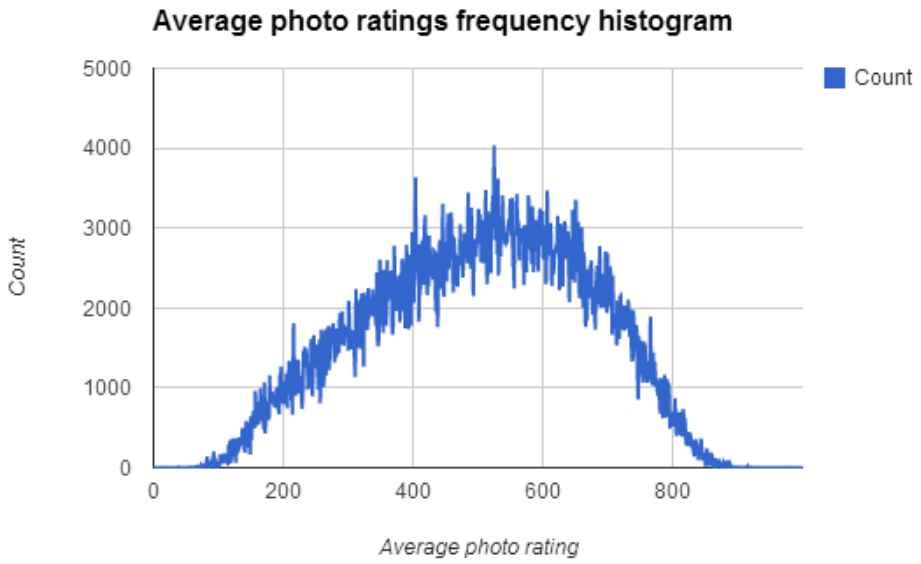

Chart 3.2.1.b: Average photo ratings frequency histogram

3.2.2 Noise removal

The first thing done with the dataset was noise and outlier removal. The following ratings were removed:

- All non-headshots
- All ratings consisting of men rating other men or women rating either men or other women
- All user rating sets with less than 40 items
- All with less than 1,8 click average per photo
- Users rating the same photo multiple times, using an average rating instead
- Photos with less than 5 total photo ratings

A user rating set will have to have at least 40 items, otherwise the rating set was not complete enough for the system in order to get a good rating assessment of the photos. When rating a photo, the user has to click to put the photo in the already existing list of photo ratings, as explained in the dataset subsection. When plotting the average number of clicks per photo for such a user rating set (the data itself is not available anymore), the resulting chart shows one clear top. If this would be two, when a user was not serious about rating the photos, that user rating set would have been close to the first top, and were he or she serious, the user rating set would be in the second top. Because this is not the case, no clear cutoff line for serious and not-serious user rating sets can be found, and it was chosen somewhat arbitrarily at an average of 1.8 clicks per photo.

This cleared up dataset will from here on be referred to as the (Paiq) dataset.

3.2.3 The final Paiq dataset

The final Paiq dataset consists of close to fourteen million (13,784,389) ratings. The final users/items combinations for the subsets are shown in table 3.2.4.a. The ratings distributions chart in chart 3.2.3.a shows the count for each rating in the training set. The ratings are clearly quite evenly divided among the scale of 10 to 1000.

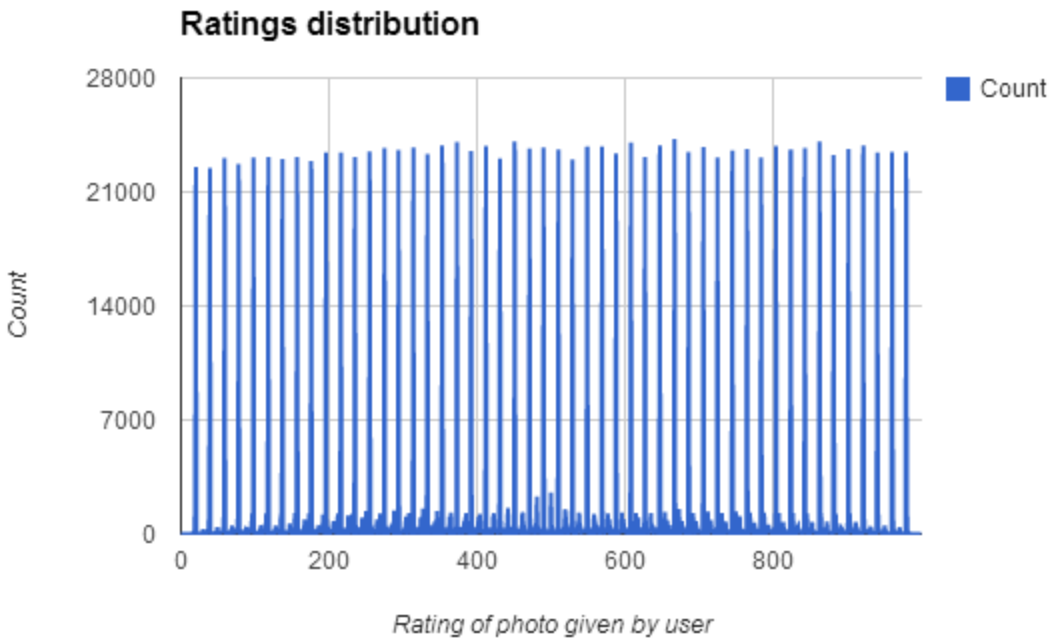

Chart 3.2.3.a: Ratings distribution of training set

3.2.4 Training, validation and test sets

The dataset was split into a training, validation and test set. As the data in sets were randomly taken from the full set and as they contain a huge number of ratings, the assumption can be made that the ratings distribution of the three sets are comparable.

The training set is used as input for the algorithm for training for the SVD and RBM approaches. This set has 99,568% missing ratings, meaning that from the total of possible ratings, only 0.432% of ratings can be used for training.

The validation set is used to determine the best set of parameters to minimize overfitting. Based on the training data and a specific parameter instantiation, a model is generated. This model is then validated on the validation set, which contains data other than the training set. The parameter instantiation with the best validation score are chosen.

The test set is used to calculate the final performance. This set is untouched previously and can

therefore find out the actual predictive power of the algorithm on new data.

The dataset was split into a training, validation and test set with 80%, 10%, 10% ratio. The validation and test sets were selected semi-randomly from the total dataset. Because of the randomness in the algorithm, the aforementioned ratios are very good approximation of the real ratios. The exact number of ratings and percentage of total dataset are shown in table 3.2.4.a.

The number of photos in the test set was 1,370,471, but the number of photos that could be predicted by the algorithms was only 1,370,349. This is because of the semi-random way in which the test set was obtained from the training and validation sets, having some photos in the test set but not in the training set. This set of photos was used as the final test set.

Set	**Number of ratings**	**Percentage of total dataset**
Training	11,035,267	80.06%
Validation	1,378,651	10.00%
Test	1,370,349	9.94%
Totals	*13,784,389*	*100%*

Table 3.2.4.a: Ratings numbers for subsets

3.3 Metrics

3.3.1 Mean absolute error

In literature, [49] states that MAE is often used in scientific research on recommender systems, and that it is easiest to understand and interpret. RMSE is the second most used metric. In [15], it concludes that using RMSE led to the same conclusion as using MAE. Based on the above reasons, this research uses MAE as an error metric.

3.3.2 Baseline methods

It is important to have a baseline to test the algorithms against. Two baseline methods were tested:

- All average (AA)
- Photo average (PA)

The AA is the most basic baseline rating method. It takes the average of all photo scores in the training set and uses that as a prediction for all photo scores. The AA is 501.077, so this method uses that average as prediction for all photos. The mean absolute error of this method only depends on the distribution of the photo ratings. As these ratings are more or less uniformly

distributed along the 10-1000 domain, this method will likely have a mean absolute error of around 240-250.

The PA method calculates the average rating per photo in the training set and then uses that value as prediction for its corresponding photos in the test set. Only items are taken into account here, while nothing is done with users. Because the user ratings are assigned uniformly by the Paiq software, the average score of each user will be around the global rating average of 501.077 and thus user average scores will not be used.

The difference between the PA and AA methods is that AA calculates the average rating of all photos while PA calculates the average rating per photo. This is more accurate than the AA method, because of the assumption that ratings for the same photo but from different users are correlated: photos that are generally liked better have a greater chance of being liked by another person as well, and vice versa. Thus, the mean absolute error of this method should be lower than AA.

3.4 Implementation

The three methods that were discussed in depth in the literature section are also the ones that were implemented for the experiment. They are:

- (k-)Nearest neighbor (kNN)
- Singular value decomposition (SVD)
- Restricted Boltzmann Machines (RBM)

This section explains implementation choices made for each method, based on the literature study of the previous chapter.

3.4.1 Nearest neighbor (kNN)

K-nearest neighbor (kNN) is implemented as described in the literature.

Normalizing the ratings is not necessary because the ratings are already normalized by the ordering of the ratings done when a user rates the photos.

The problem of the training data scaling to use too much memory is handled by calculating similarity during runtime. When creating a prediction, the similarity of the user to all users that have rated that photo is calculated, by looking at the difference between the user's and another user's score on photos they have both rated. The chosen error metric is linear, therefore this kNN implementation uses a linear metric too.

After calculating the similarity with all relevant users, the *k* nearest neighbors (users with the highest similarity score) are selected. All these users' photo ratings, corrected by the user similarity, are then weighted averaged to get the final prediction.

For users with very little neighbors, this implementation could cause problems because of the lack of information on which to base the prediction, creating quite random predictions. In a practical application, this problem could be dealt with by having the photo averages have a part in the prediction as well, but since this research is trying to find out the performance of kNN in this particular case, this is not done, and solely all available neighbors up to *k* are used.

The greatest computational cost is finding the nearest neighbors. This would normally be done offline, but is done online in this implementation. This goes in the order of $O(m*p)$, where *m* is the number of users that rated that photo in the training set and *p* the number of photos that are rated by both users. This is the computational cost for all the user similarities required for all predictions, but of course the real values of *n* and *m* vary per prediction [1,8,23,29].

To calculate the actual prediction once the relevant user similarities are known, the computational costs are $O(k)$, where *k* is the number of nearest neighbors used for the prediction [1,8,23,29].

The greatest memory costs is *s*, where *s* is the number of entries in the training set. Other important memory costs are p^2, where p is the number of photos, when storing user similarity. In practical use, there will not be similarity between every user [1,8,23,29].

3.4.2 Singular value decomposition (SVD)

The method used for doing SVD is the gradient descent Funk-SVD method [70], an approach to finding the U and V matrices of the SVD as mentioned in the literature chapter.

The advantages of this method over normal SVD are:

- It works a lot faster than most other SVD calculation methods
- The ratings the algorithm uses can be chosen, leading to less perturbations

The normal SVD approach has a computational cost of $O((m+n)^3)$, as elaborated upon in the literature chapter. The implementation of the gradient descent solver of Funk-SVD has a computational cost equal to the main loop explained below of $O(k*p*q)$, where *k* is the rank, *p* the number of training epochs and *q* the number of entries in the training set [70]. The Funk-SVD working faster than the SVD means that it is easier to recalculate the U and V matrices once (a lot of) new rating data has been added that can and should be used.

That the ratings can be chosen means that it does not have to fill in the missing entries with zero or make other assumptions about them. This does not perturb the data, which will happen with regular SVD, especially with the extreme number of missing values in the Paiq dataset. Another advantage is that it only works on the existing items, so unlike the standard SVD calculation it does not have to fill in the missing entries.

The cost of calculating a prediction from the resulting reduced matrix is the same for both

methods: a dot product of the corresponding row and column, being *O(k)* [25,70]. Storage space requirements are also the same for both, being *(m+n)*k* for a reduced matrix with dimensions *m x n* and order *k*, for storing the features. Memory requirements are *s+((m+n)*k)*, where *s* is the total number of entries in the training set, and *(m+n)*k* is the storage space cost.

A disadvantage is that Funk-SVD is prone to overfitting, so using regularization mechanisms is important, as they help in preventing overfitting. Often with SVD, the values of the features will become very good at describing the training data but be a bad generalization of the data, so that with testing the validation data error will increase instead of decreasing. Regularization tries to take into account that for photos and users for which there are only a few observations (few entries in the training set), a few extreme values (outliers) will most likely not generalize to extreme values for most users.

Another disadvantage is that the method is sensitive to the order of the training data. This can be overcome by training multiple SVDs and combining their results, but doing this in a way that does not just increase the computational cost is not straightforward. Changing the data order also changes the ideal learning and regularization rates, as the first entries of the dataset have more influence on the training simply by being used for training before the rest of the data. Also, just randomly changing the data order does not seem to help as much as ordering it by some pseudo-logical criteria. Ordering the data by deviation of the ratings from the average rating, meaning taking the extreme ratings first and the averages ones later will probably not help, since some extreme ratings might be important for training the SVD, while others might slow learning down. Due to these reasons the tests were just carried out on the training set ordered by photo, and no extra SVDs were done and combined.

Getting predictions with Funk-SVD starts with initiating two feature sets, one for the rows and one for the columns of the original matrix. It takes arbitrary values for the features: either manually or semi-randomly chosen, close to zero. For this experiment, a global initial value for all features was used to make comparison testing easier.

Then for each training cycle, the algorithm derives the approximation error, in our case the MAE, on the predictions for each entry in the training set by taking the first rank column-vector U_1 and corresponding row-vector V_1 (the first feature). It tries to optimize these vectors as best as possible by doing gradient descent. This entails a step-by-step adjustment of the feature values so that the approximation error gets smaller each time, moving the error towards the derived approximation error minimum. It then moves on to the next pair of vectors U_2 and V_2 (the second feature), and tries to optimize those, and so on. The first vectors are the most significant ones, then the second ones, and so on. Each extra pair of vectors increases number of features (similar to the rank of the matrix decomposition) by 1. For clarification, the loops in the main training algorithm are shown in algorithm 3.4.2.a.

```
For (each training epoch)
      For (each feature)
            For (each rating in the training set)
                  Calculate a prediction and the error (MAE) associated wit
                  this prediction
                  Increase the value of the features by a small amount that
                  lowers the error
```

Algorithm 3.4.2.a: SVD main training algorithm loop

3.4.3 Restricted Boltzmann machines (RBM)

RBM is implemented in a straightforward manner from the details found in the literature study.

This research used RBMs with binary states, where 0 is used for dislike and 1 for like to get clear values . Every time a state is set for a rating in the training set, a random function chooses a number between 0 and 1000. If the number is bigger than the rating, the state is set to 0 and if it is smaller or equal to the rating it is set to 1.

Softmax visible units were not used because the number of softmax units per visible unit would have to be 1000 to cover each possible rating, making both computational time and memory constraints a problem. Continuous RBM was not used because of the large amount of training ratings probably already makes up for the RBM having binary units instead and losing out on some accuracy here.

The weights and biases are initialized with a Gaussian around 0.1, with a standard deviation of 1.0, as is often done in practical applications.

A CD (contrastive divergence) with rank 1, 3, 5 and 9 was used; the lowest one for the first 25% of the training epochs, and then each subsequent 25% of the training epochs the following CD is used: 3, 5 and 9. In practice, using rank 1 already seemed to work out well, but using the higher ranks should make the RBM even more accurate in its training [79].

The learning rate was halved after half of the training epochs, and then halved again after three-quarters of the training epochs. This is done because at that point the weights are already converging and the smaller improvements become more important [75].

To train the RBM, the computational cost is $O(h*e*s)$, where h is the number of hidden units, e the number of training epochs and s the number of entries in the training set [73,78,79].

To calculate a prediction: $O(v*h)$, where v is the number of visible units for that user that are used and h the number of hidden units. A translation from visible units to hidden units and back to visible units is made [73,78,79].

The memory costs for RBM are: *s+5*v+5*h*, where *s* is the number of entries in the training set, *h* is the number of hidden units, *v* the number of visible units (equal to the number of photos), [73,78,79]. Here, *v*h* stands for the weights, *v+h* for the biases, and the other *v+h* terms for the temporary storage needed to do contrastive divergence.

4 Parameter estimation

This chapter shows how the values of the parameters of the algorithms were selected. Per method it discusses the parameters that were trained, the validation of the parameters and finally states which values were chosen for the parameters.

The parameters are the values of the methods that can be varied and thereby either increase or decrease the MAE when changed. Each method's parameters were chosen based on this criterion.

For most of the parameters, there will be a tradeoff between performance and computation time. The higher the precision of the algorithm, the greater the computational cost required to get to this point. This research wants to find the settings where the improvement in precision is a large improvement over other settings, but the additional computational time is still acceptable. Finding this sweet spot is also what the validation phase is used for. The computational time of the optimally selected parameter settings should be acceptable, which is arbitrary and depends on the hardware and specific algorithm implementation used to calculate the predictions.

4.1 Nearest neighbor

4.1.1 Parameters

This is the parameter that was tested for kNN:

- Number of nearest neighbors *k*

The implementation of nearest neighbor used in this research has only one parameter that is changed, which is the number of nearest neighbors to use, also known as the k in the term kNN. Using a k that is too low will probably mean that the predictions will often lack information from other neighbors that are also very near. Using a k that is too high will probably add information from users that are too far away, meaning they have little nothing in common with the rating behavior of the user the prediction is calculated for.

Because of limitations to the available hardware and time constraints, 1/10th of the entries in the validation set were randomly selected and used for validation of the parameters. The test set run is done over the entire test set to make sure the final results are derived from the exact same set for all methods.

4.1.2 Validation of parameters

The chart containing error information on parameter settings is shown in chart 4.1.2.a.

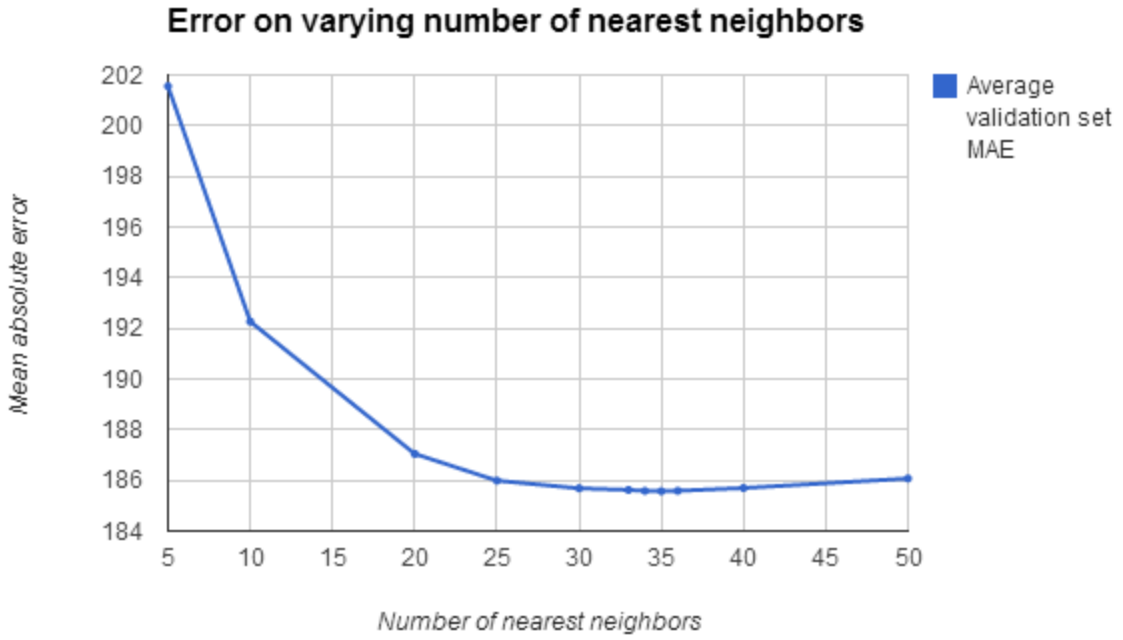

Chart 4.1.2.a: kNN error on varying number of nearest neighbors

The chart (4.1.2.a) shows that the best number of nearest neighbors to use is 35, which has a slightly lower MAE than 34 and 36. Using a higher number adds information from unwanted neighbors.

4.1.3 Selected parameters

The selected values for the parameters are:

- Number of nearest neighbors: 35

4.2 Singular value decomposition

4.2.1 Parameters

These are the parameters that were tested for SVD, with the used default values:

- Number of features: 40
- Initial value of features: 0.1
- Number of training epochs: 100
- Learning rate: 0.000001
- Regularization: 0.025

These default values were suggested in Funk-SVD references, for example in [70,80].

The number of features corresponds to the number of properties that try to generalize the data,

which is the number of different categories that the SVD tries to categorize the ratings into. Probably when adding more features, the mean absolute error decreases, but at some time all meaningful properties have been described, and adding more features does not really decrease the error in a noticeable way, but does increase computational time.

The features are updated by doing multiplications over previous values, starting of course with the initial value. If the initial values are set too low or too high, it will take a long time for the model to find its settings with the smallest MAE.

A training epoch is one pass of all features over all training data. Per rating in the training set, each feature is adjusted a bit in the direction of the error. Using more training epochs would essentially decrease the error each time, but after some time each extra training epoch is redundant because the minimum error has already been reached, and because it will cause more overfitting. Not having enough training epochs will stop the training before the feature settings with minimal error have been found.

The learning rate is the rate by which the values of the features change each training run. If it is set too low it will increase computational time and if it is set too high the features can pass over the global minimum.

With regularization, when training a user feature-photo feature combination, the prior value of the other feature is taken into account and subtracted from the learning rate. A low regularization rate will not prevent overfitting as much, while a too high regularization rate will negate learning too much and just generalize all features.

The default values of the parameters gives the following mean absolute error:

- Validation set: 188.256

4.2.2 Validation of parameters

Chart 4.2.2.a shows the error that comes with varying the number of features.

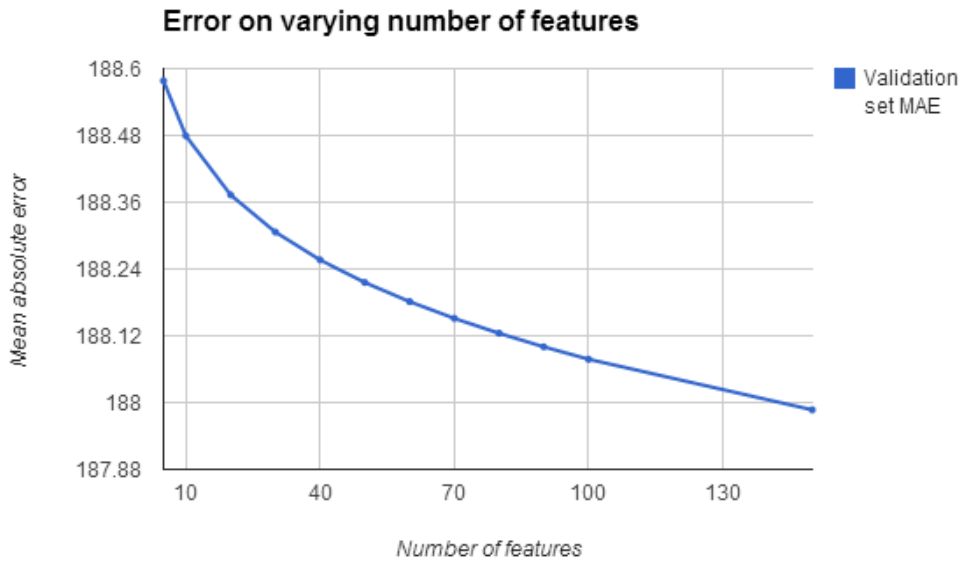

Chart *4.2.2.a: SVD error on varying number of features*

As can be seen in the graph, the MAE declines by increasing the number of features used in the SVD, as expected. However, this is a converging process, meaning that the decrease in mean absolute error gets smaller with every extra added feature. The best tradeoff between MAE and computational cost was set on 50 features. Overfitting might happen at some point when using even more features, but this was not researched.

Chart 4.2.2.b shows the error when varying the initial value of the features.

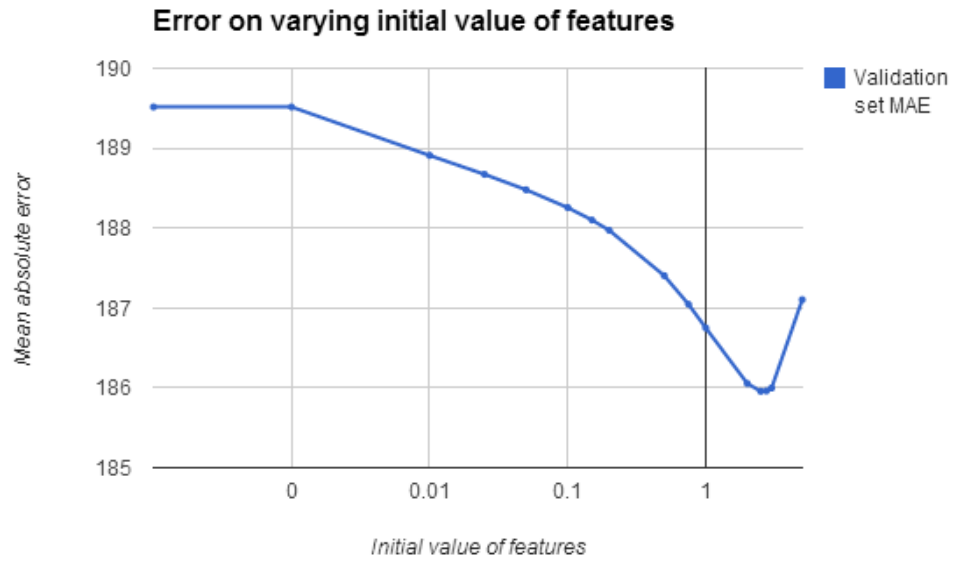

Chart 4.2.2.b: SVD error on varying initial value of features

The best initial value of features seems to be 2.5, looking at the graph. This is where validation set MAE is at its lowest. When the initial value is set lower or higher, there are not enough training epochs to get to a lower MAE, meaning that for 200 training epochs 2.5 is the optimal value of the initial features.

When changing the number of training epochs, the mean absolute error changes as shown in chart 4.2.2.c.

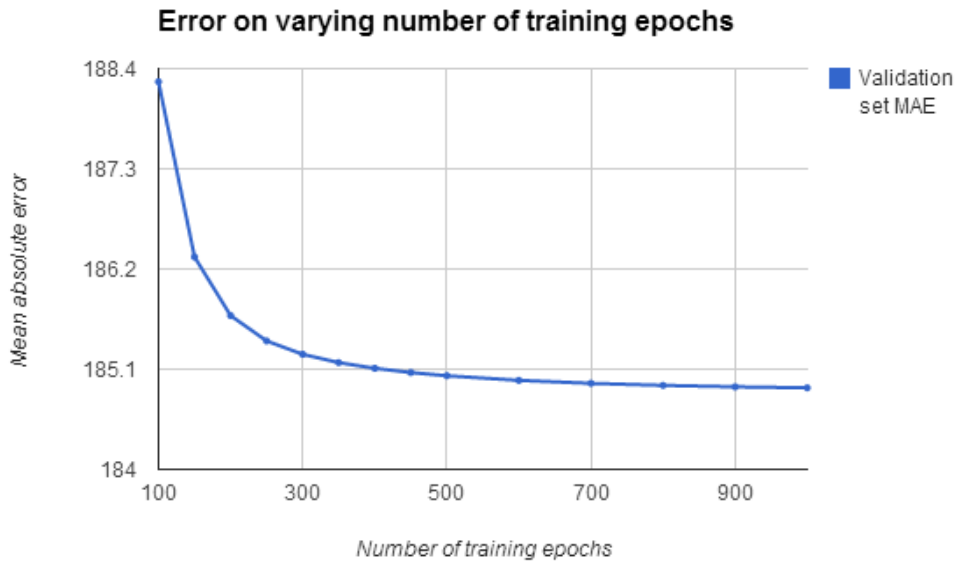

Chart 4.2.2.c: SVD error on varying number of training epochs

As with the number of features, the mean absolute error keeps decreasing as an asymptote when increasing the number of training epochs. This is because adding more training epochs gives the algorithm better time to converge to the global minimum. It was expected that after some time overfitting would occur, but this is not the case. This can mean that the data in the training set is a really good generalization of the data in the validation set, or that this method creates a generalization of the data based on inherent properties between ratings in both the training and validation sets, which likely could be the average photo scores. This furthermore means having to find a sweet spot here instead of finding the minimum and taking that as the selected parameter setting. Here, 600 training epochs were chosen, because the improvement of adding another 100 training epochs (and having 700 training epochs total) only improves the final resulting mean absolute error is less than 0.05.

Chart 4.2.2.d presents the error that belongs to varying the learning rate.

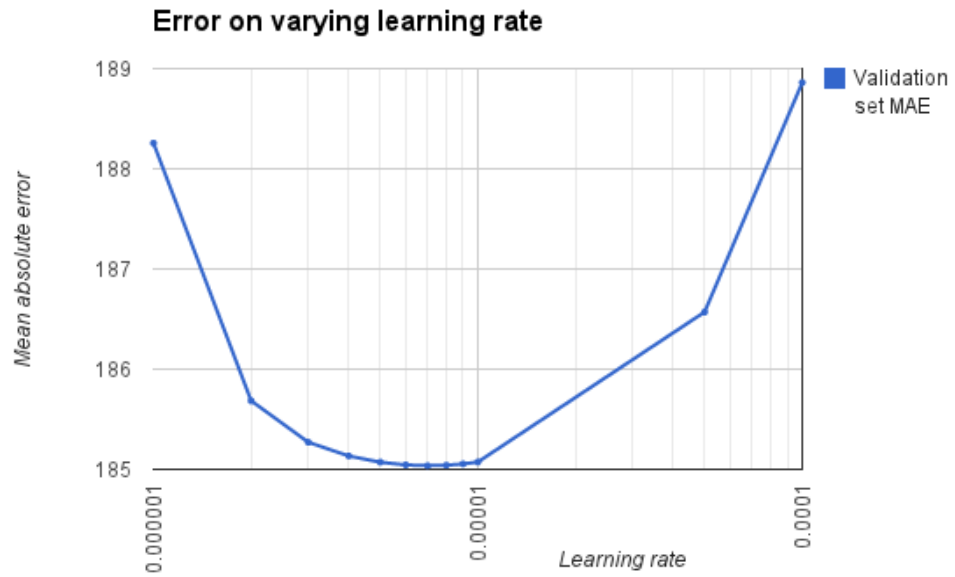

Chart 4.2.2.d: SVD error on varying learning rate

With the learning rate a minimum can be found that can be used as the best setting. The number 0.000007 was chosen as the optimized learning rate.

Chart 4.2.2.e shows the mean absolute error on varying the regularization:

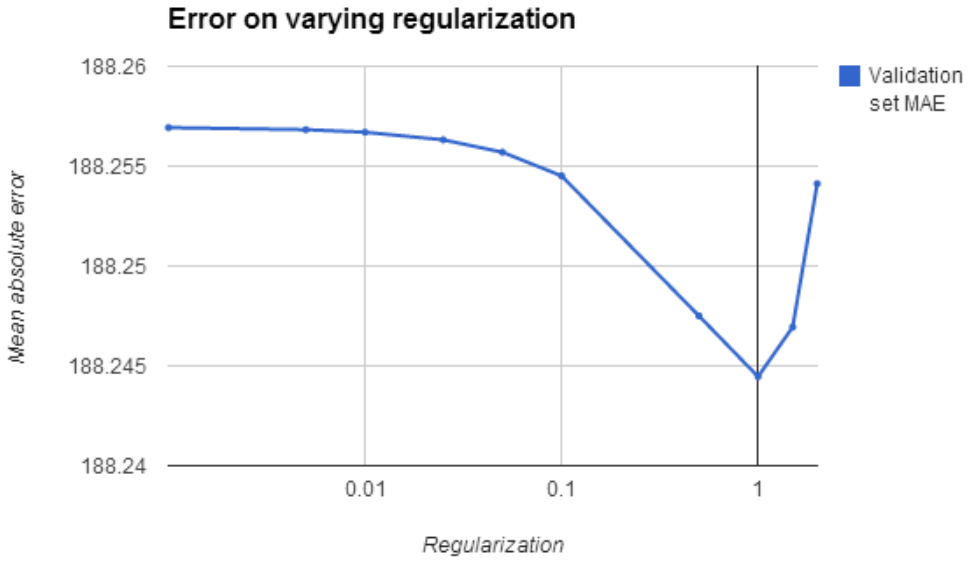

Chart 4.2.2.e: SVD error on varying regularization

The regularization is a parameter where there is a minimum, at 1. This is the optimized parameter setting taken for testing. Since the differences are small, it seems that regularization

has little effect on the outcome of the SVD. Why this is will be further explained in the discussion in the next chapter.

4.2.3 Selected parameters

The selected values for the parameters are:

- Number of features: 50
- Initial value of features: 2.5
- Number of training epochs: 600
- Learning rate: 0.000007
- Regularization: 1

4.3 Restricted Boltzmann machines

4.3.1 Parameters

These are the parameters that were tested for RBM, with the used default values:

- Number of hidden units: 20
- Number of training epochs: 200
- Learning rate: 0.001

As with the SVD approach, the parameter values were adapted from practical references such as [78,79,80], mixed with own insights about the dataset and computational costs.

The hidden units are comparable to the features of the SVD. It is expected that adding more hidden units leads to a decrease in MAE, but having too many hidden units only adds unneeded computational time.

Just as with SVD, adding more training epochs decreases the MAE up to a certain point, but after that it will increase MAE because of overtraining on the training set. For the RMB, because contrastive divergence is used in four steps, the number of training epochs used in the model is also tested in multiples of four.

The consequences of tweaking the learning rate is also the same as SVD: setting it too low makes learning slow, and setting it too high makes it unable to get close to the global minimum in MAE.

The RBM was trained five times with the default values of the parameters. These settings give the following averaged mean absolute error:

- Validation set: 189.827

4.3.2 Validation of parameters

The semi-random initiation of RBM causes a slight variance in parameter testing results. To

counter this, all tests were carried out five times and their average MAE was taken as the MAE of that model.

Chart 4.3.2.a shows the error that comes with varying the number of hidden units.

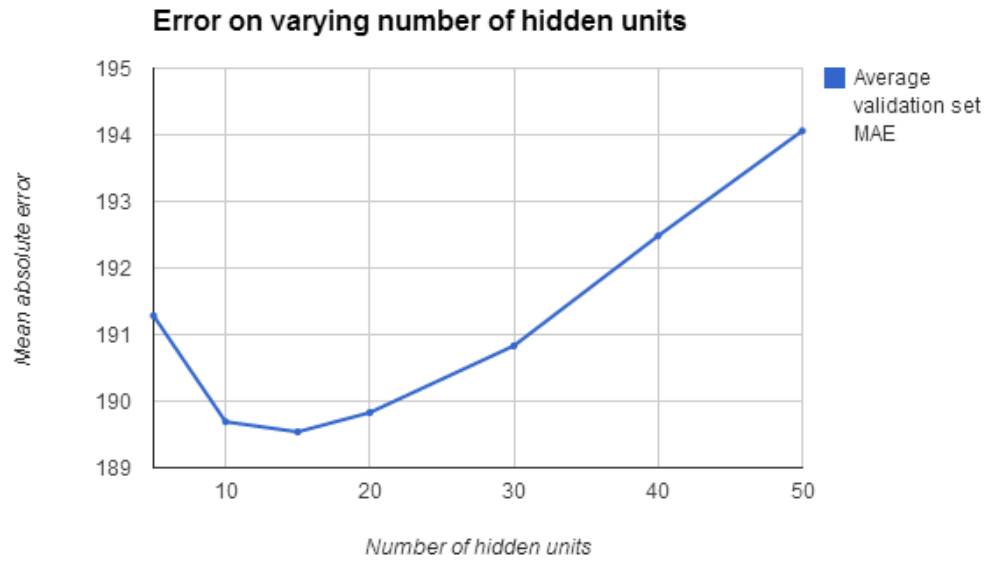

Chart 4.3.2.a: RBM error on varying number of hidden units

This chart shows that increasing the number of hidden units until 15 decreases the MAE. Increasing the number of hidden units further increases the MAE, meaning that 15 hidden units best describes the underlying data, and using more hidden units overfits on the validation set.

Chart 4.3.2.b shows the error on varying the number of training epochs.

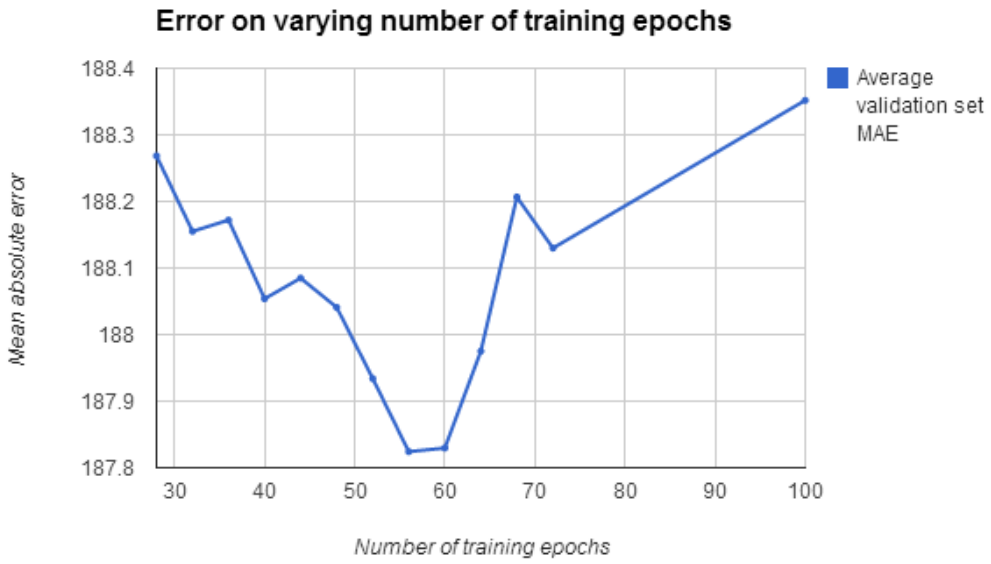

Chart 4.3.2.b: RBM error on varying number of training epochs

The average MAE on the default number of training epochs of 200 is 189.827. This is not in the chart to make the chart more focused on the number of training epochs with the lowest MAE. The spikiness in the chart can be attributed to the semi-random initiation of RBM. To get clearer results and a possibly smoother graph, more parameter tests could have been run. The selected value of the parameter is 56.

The learning rate parameter tests are shown in chart 4.3.2.c.

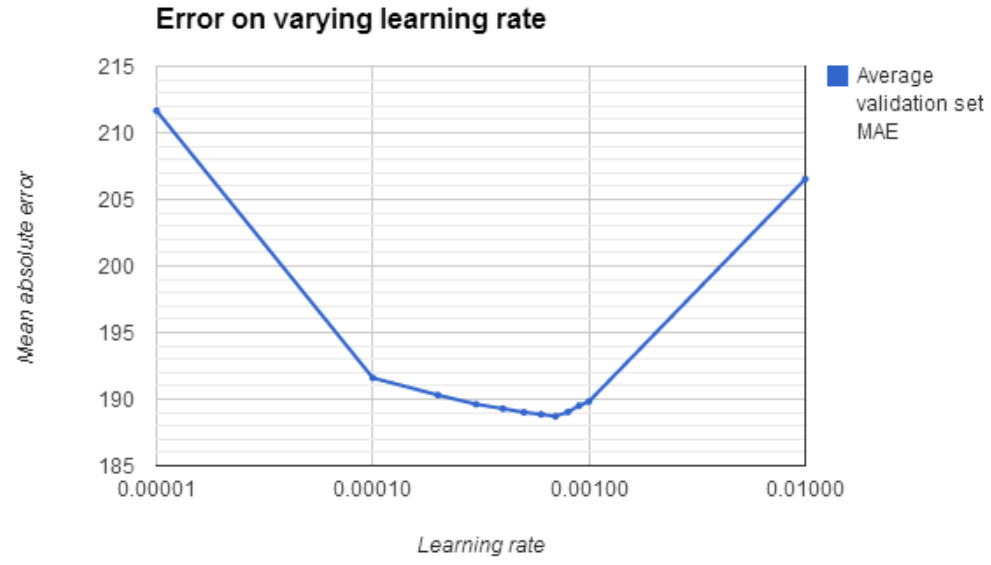

Chart 4.3.2.c: RBM error on varying learning rate

The best on the learning rate is 0.0007. It is important to note that the learning rate was halved after 50% of the training epochs were done, and again halved after 75% was done, as explained in implementation details of RBM in the experimental setup chapter.

4.3.3 Selected parameters

The selected values for the parameters are:

- Number of hidden units: 15
- Number of training epochs: 56
- Learning rate: 0.0007

5 Results and discussion

The first part of this chapter shows the results of the experiments, while the second part discusses these results.

5.1 Results

This section describes the results of the predictions with the selected parameters on the test set and shows several charts and metrics concerning the testing data.

For each method, the distribution of the ratings, the averaged MAE of the predictions and the MAEs of the prediction over the real ratings are shown. Then, a scatter plot shows about 1/500 randomly chosen predictions (which is 2741 predictions) plotted against the real ratings for those predictions. Finally, the ratings scale of 1-1000 is subdivided into five equal categories, starting with 1-200, and the last being 801-1000, and shows the averaged MAE for all the predicted and real ratings in those categories.

5.1.1 All average ratings distribution

Some stats for the AA baseline method on the test set are:

- MAE: 241.255
- Average prediction: 501 (rounded to full integer)

5.1.1.1 Ratings distribution

The ratings distribution for the AA is not that interesting, since all predictions were the same rating. The same goes for its error distribution over predicted ratings, which is equal to the test set error of 241.255 for the rating 501.077. The average error of the AA predictions over the real ratings are shown in chart 5.1.2.1.a, showing an expected average error that is equal to the absolute value of (501.077 - the real rating).

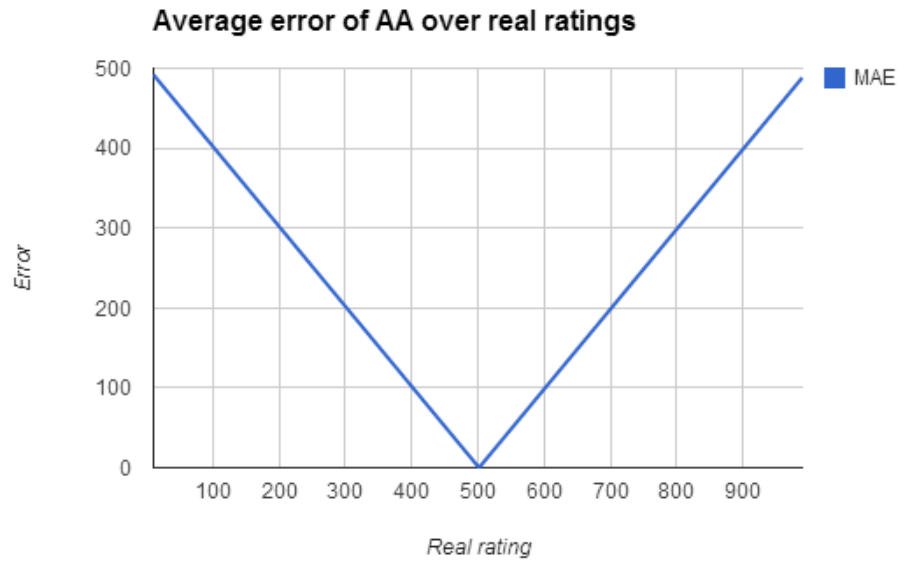

Chart 5.1.2.1.a: Average error of AA over real ratings

5.1.1.3 Rating range errors

When merging a range of ratings per 200 ratings, the following average errors on the predicted ratings are shown in table 5.1.1.3.a.

Predicted ratings range	**Averaged MAE per range**	**Number of ratings**
1-200	0	0
201-400	0	0
401-600	241.255	1,370,471
601-800	0	0
801-1000	0	0
Average / Total	*241.255*	*1,370,471*

Table 5.1.1.3.a: AA predicted ratings range errors and count

This is obvious, because all predictions are made at the global photo average.

And the average errors on the real ratings are in table 5.1.1.3.b.

Real ratings range	Averaged MAE per range	Number of ratings
1-200	393,019	261,509
201-400	197.824	279,268
401-600	49.307	284,216
601-800	194.529	276,959
801-1000	389.988	268,519
Average / Total	*241.255*	*1,370,471*

Table 5.1.1.3.b: AA real ratings range errors and count

5.1.2 Average photo ratings distribution

Some stats for the PA baseline method on the test set are:

- MAE: 186.862
- Average prediction: 501.176

5.1.2.1 Ratings distribution

The distribution of predicted ratings for the PA is equal to the average photo ratings frequency histogram shown in chart 3.2.1.b, as it uses the average photo ratings as predictions.

The average error of PA predictions produce chart 5.1.2.1.b.

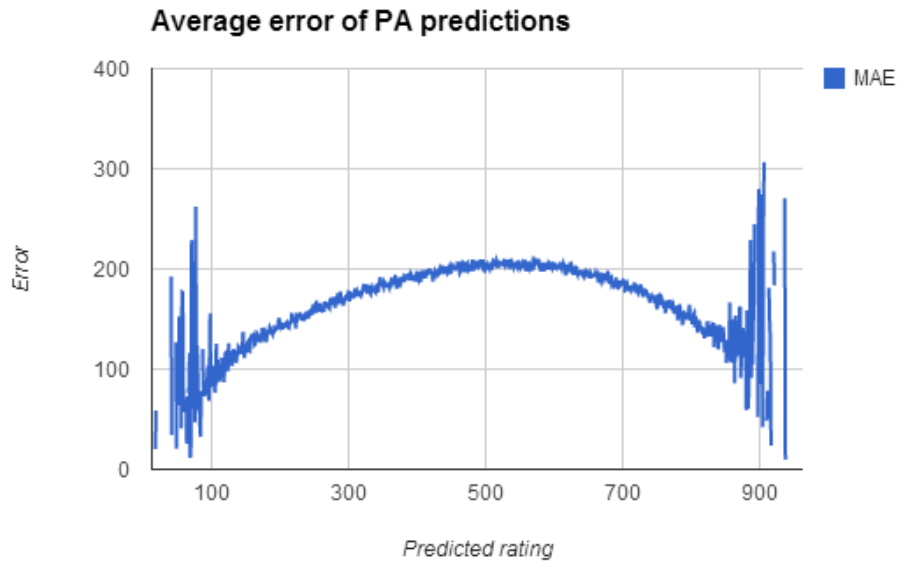

Chart 5.1.2.1.b: Average error of PA predictions

A total of seven merged ratings with total count eleven were taken out to improve clarity of the above chart:

- A MAE of 910 on one 51 rating
- A MAE of 894 on one 894 rating
- A MAE of 678 on two (averaged) 903 ratings
- A MAE of 685 on one 959 rating
- A MAE of 520 on two (averaged) 961 ratings
- A MAE of 669 on one 973 rating
- A MAE of 705 on three (averaged) 982 ratings

The chart shows that the error tops between the ratings of 450 and 600, and tapers off at both sides. The absolute outliers can have a high or low error, or anything in between, due to these having a relatively low count.

And the average error of PA over real ratings:

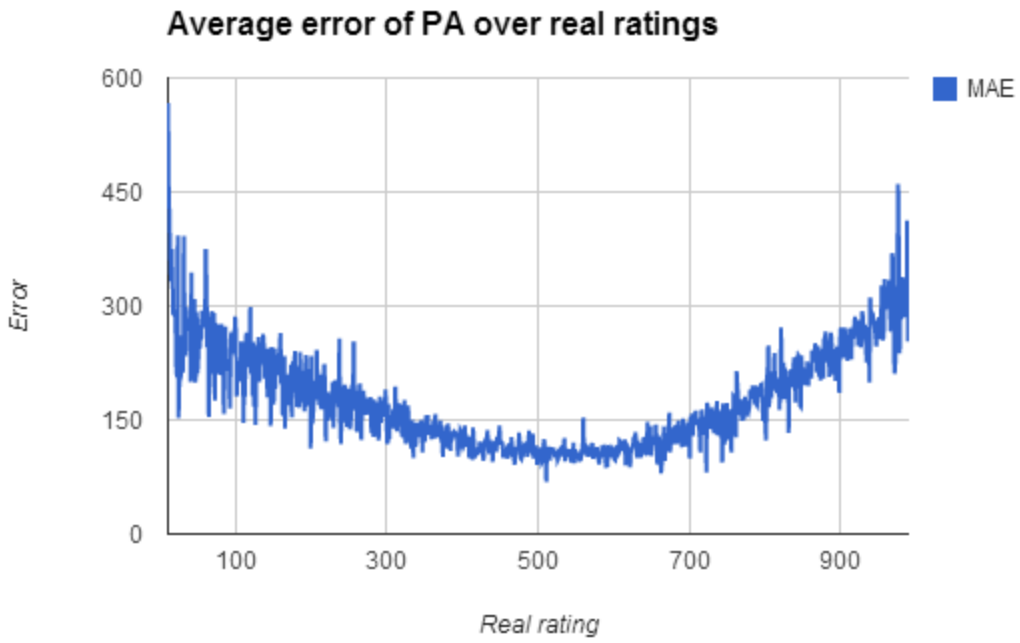

Chart 5.1.2.1.c: Average error of PA over real ratings

The above chart shows that most photos that had a true rating close to the global photo average of 501 had the least error in prediction, and photos further away have on average a greater error.

5.1.2.2 Scatter plot

Chart 5.1.2.2.a shows the scatter plot for PA. The most important thing it shows it that the main predicted ratings are at least somewhat equal to the real rating. There are few low predicted

ratings for the higher real ratings, and not many high predicted ratings for the low real ratings.

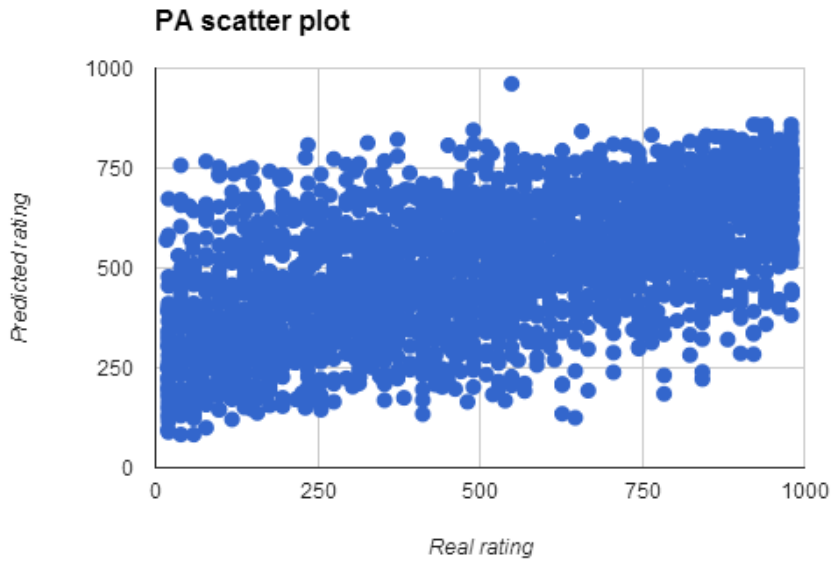

Chart 5.1.2.2.a: PA scatter plot of predicted ratings vs real ratings

5.1.2.3 Rating range errors

When merging a range of ratings per 200 ratings, the average errors on the predicted ratings as displayed in table 5.1.2.3.a are found.

Predicted ratings range	**Averaged MAE per range**	**Number of ratings**
1-200	127.819	52,011
201-400	174.161	339,384
401-600	201.942	556,776
601-800	186.832	400,717
801-1000	139.823	21,461
Average / Total	*186.857*	*1,370,349*

Table 5.1.2.3.a: PA predicted ratings range errors and count

And the average errors on the real ratings are shown in table 5.1.2.3.b.

Real ratings range	Averaged MAE per range	Number of ratings
1-200	252.365	261,487
201-400	164.156	279,243
401-600	115.004	284,190
601-800	153.445	276,928
801-1000	257.210	268,501
Average / Total	*186.862*	*1,370,349*

Table 5.1.2.3.b: PA real ratings range errors and count

5.1.3 Nearest neighbor

Some stats for the selected values of the parameters on the kNN method are:

- Training set MAE: 181.334
- Test set MAE: 182.552
- Average prediction on test set: 504.942

5.1.3.1 Ratings distribution

The distribution of predicted ratings for kNN shown in chart 5.1.3.1.a shows a mountainous curve. Most ratings are around the global photo average of 501 and the number of ratings tapers off towards either side. The highest count of a predicted rating with 2.561 predictions is at a rating of 597.

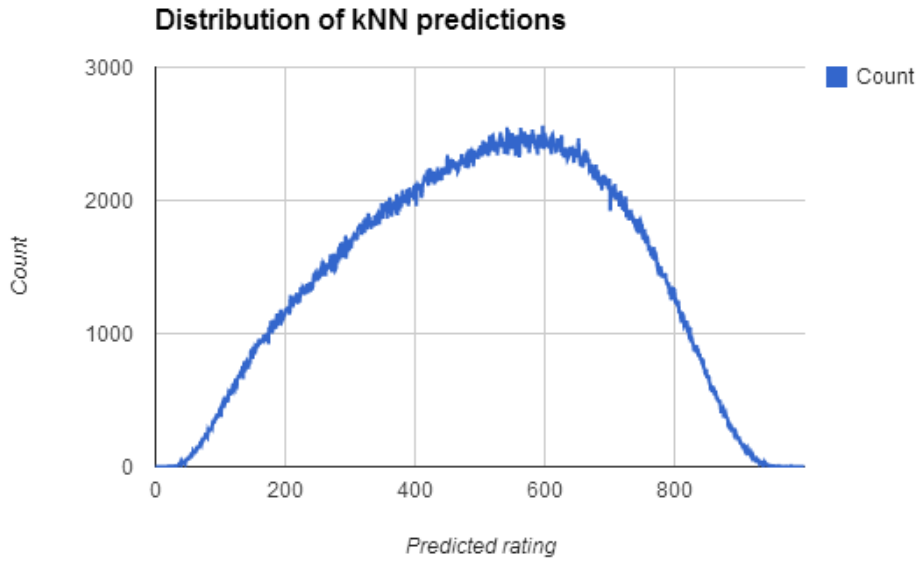

Chart 5.1.3.1.a: Distribution of kNN predictions

The average error of kNN predictions produce chart 5.1.3.1.b. The errors are distributed in a hill in somewhat the same way again as that of the PA. A total of six merged ratings with total count nine were taken out to improve the clarity of the chart:

- A MAE of 768 on one 17 rating
- A MAE of 618 on two (averaged) 972 ratings
- A MAE of 510 on one 401 rating

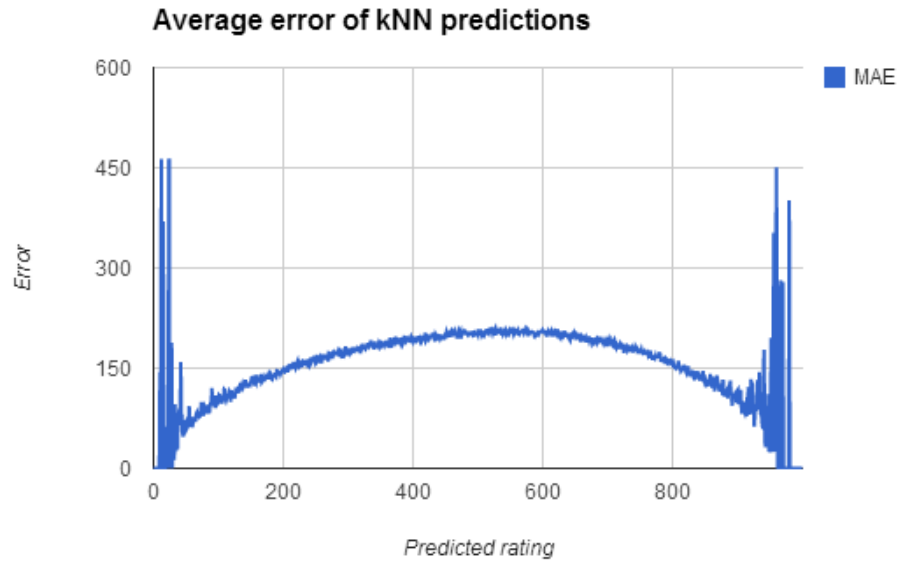

Chart 5.1.3.1.b: Average error of kNN predictions

And the average error of kNN over the real ratings are in chart 5.1.3.1.c. This chart shows that the algorithm's predictions have a higher error on the real lower and higher ratings compared to the middle ratings around 500.

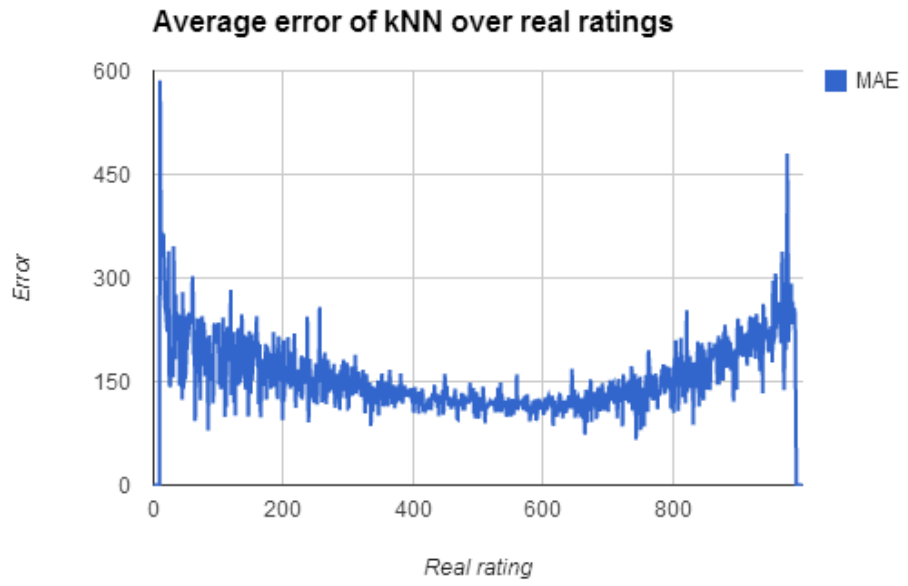

Chart 5.1.3.1.c: Average error of kNN over real ratings

5.1.3.2 Scatter plot

Chart 5.1.3.2.a shows the scatter plot for kNN. This scatter plot is quite similar to the one for PA.

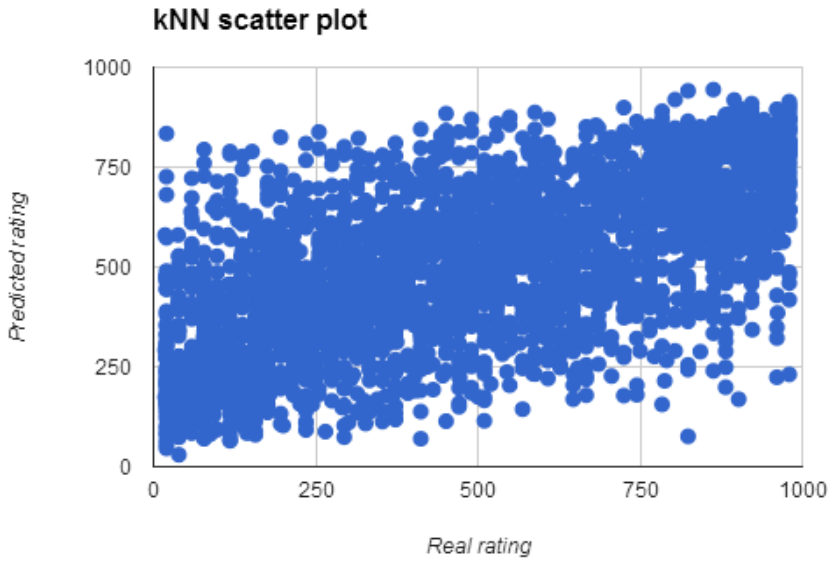

Chart 5.1.3.2.a: kNN scatter plot of predicted ratings vs real ratings

5.1.3.3 Rating range errors

When merging a range of ratings per 200 ratings, the following average errors on the predicted ratings are found as shown in table 5.1.3.3.a.

Predicted ratings range	Averaged MAE per range	Number of ratings
1-200	124.441	94,733
201-400	176.075	332,700
401-600	201.529	466,403
601-800	187.882	403,315
801-1000	136.893	73,125
Average / Total	*182.553*	*1,370,276*

Table 5.1.3.3.a: kNN predicted ratings range errors and count

And the average errors on the real ratings are in table 5.1.3.3.b.

Real ratings range	Averaged MAE per range	Number of ratings
1-200	233.431	261,467
201-400	169.620	279,231
401-600	133.403	284,176
601-800	154.731	276,917
801-1000	227.176	268,485
Average / Total	*182.553*	*1,370,276*

Table 5.1.3.3.b: kNN real ratings range errors and count

5.1.4 Singular value decomposition

Some stats for the selected values of the parameters on the SVD method are:

- Training set MAE: 183.268
- Test set MAE: 185.264
- Average prediction on test set: 500.701

5.1.4.1 Ratings distribution

The distribution of predicted ratings for the SVD shown in chart 5.1.4.1.a shows a similar mountainous curve to that of the kNN. The SVD has its highest count of a predicted rating with 3.030 predictions ending up at 553.

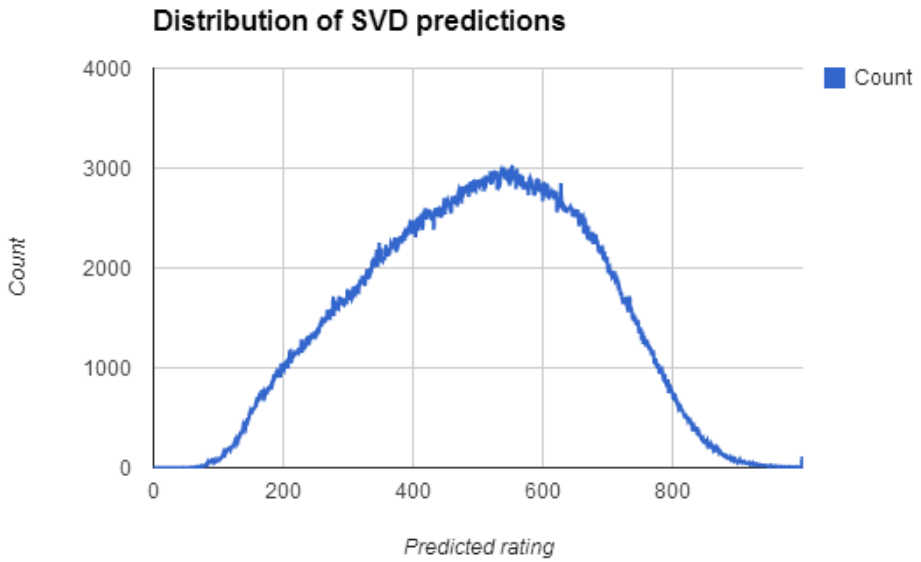

Chart 5.1.4.1.a: Distribution of SVD predictions

The average error of SVD predictions produce chart 5.1.4.1.b. A total of 2 merged ratings with total count 2 were taken out to improve clarity of the chart:

- A MAE of 544 on one 44 rating
- A MAE of 913 on one 48 rating

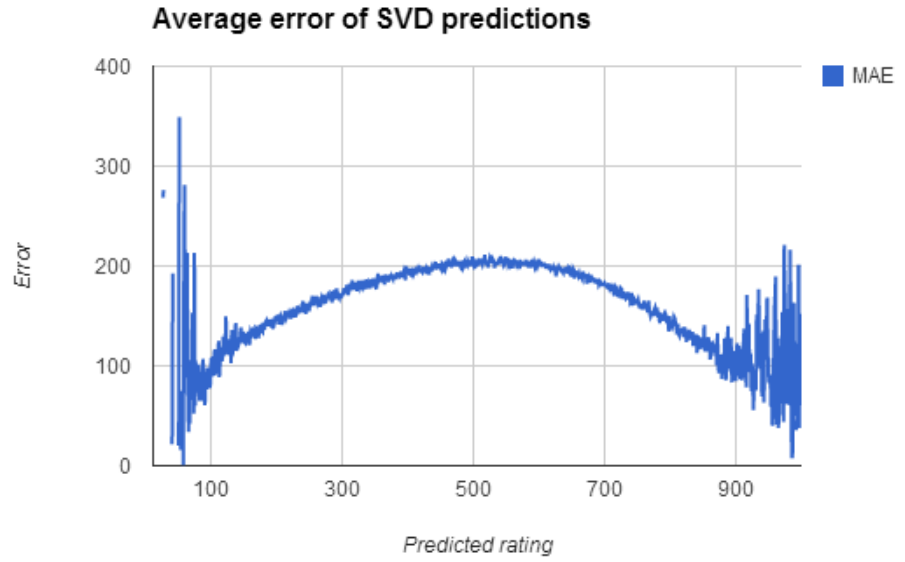

Chart 5.1.4.1.b: Average error of SVD predictions

And the average error of SVD over the real ratings are in chart 5.1.4.1.c. This chart shows the same thing for SVD as for kNN: a higher error on the real lower and higher ratings compared to

the middle ratings around 500.

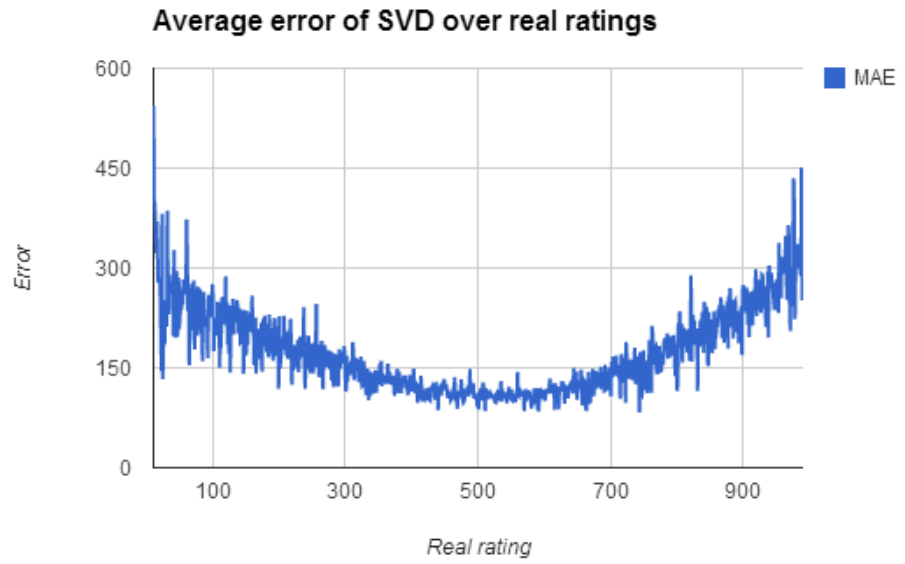

Chart 5.1.4.1.c: Average error of SVD over real ratings

5.1.4.2 Scatter plot

Chart 5.1.4.2.a shows the scatter plot for SVD. This one is again quite similar to the ones for PA and kNN.

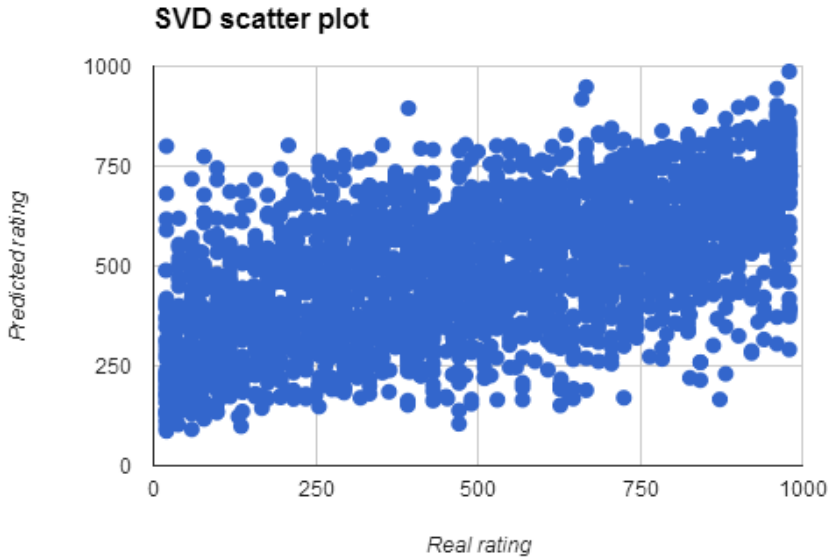

Chart 5.1.4.2.a: SVD scatter plot of predicted ratings vs real ratings

5.1.4.3 Rating range errors

When merging a range of ratings per 200 ratings, the following average errors on the predicted ratings are found as shown in table 5.1.4.3.a.

Predicted ratings range	**Averaged MAE per range**	**Number of ratings**
1-200	131.966	54,741
201-400	175.511	345,379
401-600	201.358	550,048
601-800	183.661	386,367
801-1000	127.656	33,814
Average / Total	*185.263*	*1,370,349*

Table 5.1.4.3.a: SVD predicted ratings range errors and count

And the average errors on the real ratings are in table 5.1.4.3.b.

Real ratings range	**Averaged MAE per range**	**Number of ratings**
1-200	248.209	261,487
201-400	162.039	279,243
401-600	115.930	284,190
601-800	154.921	276,928
801-1000	252.797	268,501
Average / Total	*185.264*	*1,370,349*

Table 5.1.4.3.b: SVD real ratings range errors and count

5.1.5 Restricted Boltzmann machines

When taking the averaged error of five test runs, some stats for the selected values of the parameters on the RBM method:

- Training set MAE: 185.026
- Test set MAE: 187.438
- Average prediction on test set: 501.662

For the ratings distribution and ratings range errors, one run was arbitrarily selected to go

in-depth into.

5.1.5.1 Ratings distribution

The distribution of predicted ratings for the RBM as shown in chart 5.1.5.1.a also shows a similar mountainous curve to that of the kNN and SVD. For RBM the highest amount of predictions with predictions is 3.172 predictions ending up at 524.

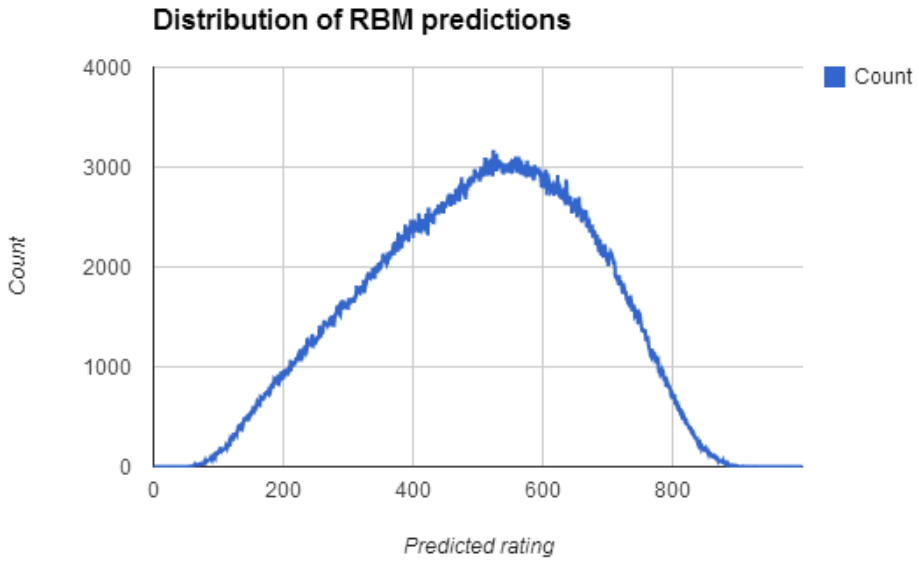

Chart 5.1.5.1.a: Distribution of RBM predictions

The average error of RBM predictions produce chart 5.1.5.1.b. The errors are distributed in a hill in somewhat the same way again as that of the other methods. No ratings were taken out in this chart.

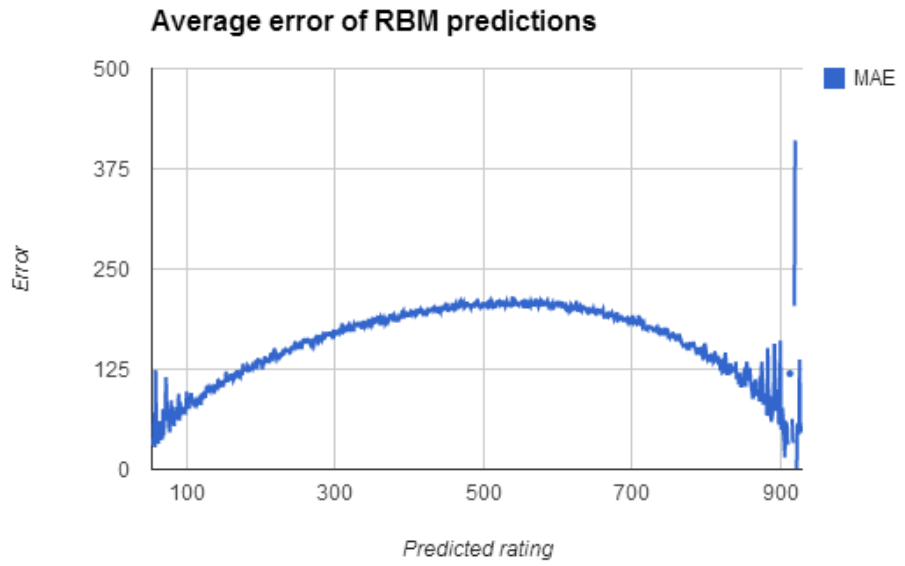

Chart 5.1.5.1.b: Average error of RBM predictions

And the average error of RBM over the real ratings are in chart 5.1.5.1.c. It shows the same thing for RBM as the similar SVD chart did for kNN: a higher error on the real lower and higher ratings compared to the middle ratings around 500.

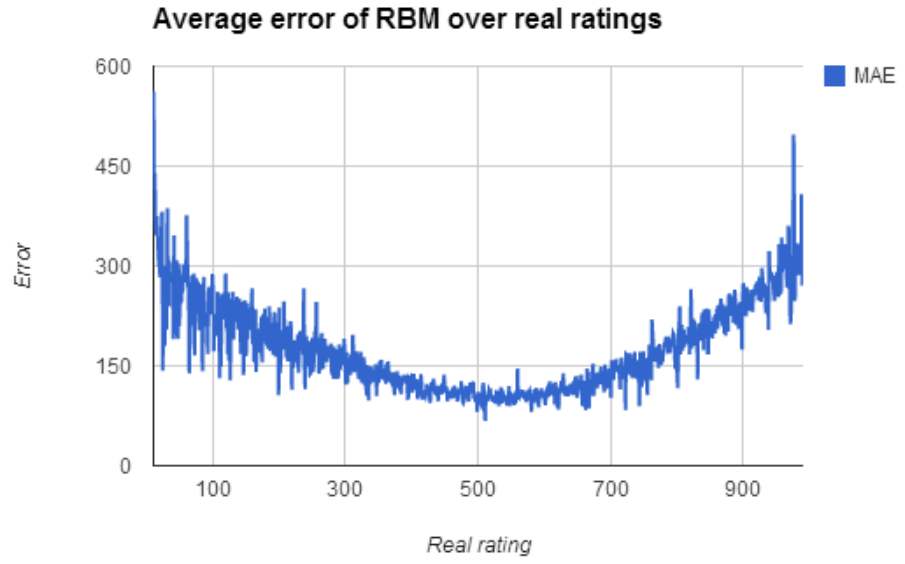

Chart 5.1.5.1.c: Average error of RBM over real ratings

5.1.5.2 Scatter plot

Chart 5.1.5.2.a shows the scatter plot for RBM. This chart is again quite similar to the scatter plots for PA, kNN and SVD.

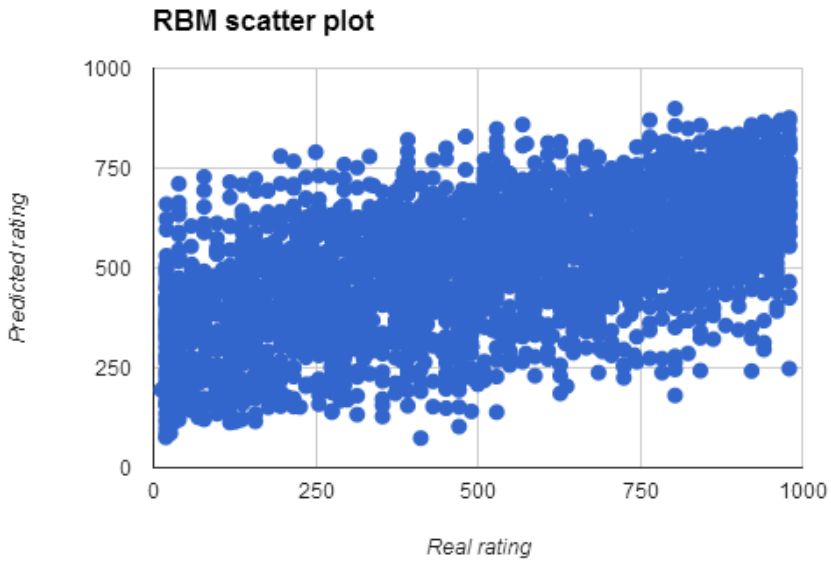

Chart 5.1.5.2.a: RBM scatter plot of predicted ratings vs real ratings

5.1.5.3 Rating range errors

When merging a range of ratings per 200 ratings, the following average errors on the predicted ratings are found as shown in table 5.1.5.3.a.

Predicted ratings range	**Averaged MAE per range**	**Number of ratings**
1-200	113.771	55,530
201-400	172.502	331,659
401-600	203.903	561,592
601-800	187.250	397,777
801-1000	128.226	23,791
Average / Total	*185.829*	*1,370,349*

Table 5.1.5.3.a: RBM predicted ratings range errors and count

And the average errors on the real ratings are in table 5.1.5.3.b.

Real ratings range	Averaged MAE per range	Number of ratings
1-200	252.188	261,487
201-400	164.865	279,243
401-600	114.498	284,190
601-800	152.531	276,928
801-1000	256.289	268,501
Average / Total	*187.438*	*1,370,349*

Table 5.1.5.3.b: RBM real ratings range errors and count

5.2 Discussion

Because PA is a far better baseline method than AA, and has the same linear computational and memory costs as PA, AA is not considered as one of the relevant methods that will be discussed in this section, whereas PA, kNN, SVD and RBM are.

In any machine learning recommender system, after some time overfitting will occur during training, but this is not the case in this experiment. This can mean that the data in the training set is a good generalization of the data in the validation set, or that this method creates a generalization of the data based on inherent properties between ratings in both the training and validation sets, which likely could be the average photo scores. This is probably due to noise in the dataset and that the models are not powerful enough to model the training set. The training set MAE for the approaches are barely better than the test set MAE, and are not even close to 0.

5.2.1 Overall performance

The performance of all relevant methods (thus meaning PA, kNN, SVD and RBM, but excluding AA) is close when purely looking at their MAE on the test set. Table 5.2.1.a shows the MAE for all the important methods. It shows that kNN is the best performer here, but only 4.130 points better than the next best method, and only 2.712 points better than the worst method.

	PA	kNN	SVD	RBM
MAE	186.862	**182.552**	185.264	187.438

Table 5.2.1.a: MAE for all important methods

Table 5.2.1.b shows the average predicted rating for all important methods. The average prediction of SVD and RBM only differs by 0.193, while the average kNN prediction of 504.942 differs with 3.28 points from the nearest method (RBM). In a blend, mixing the predictions of kNN

and either SVD or RBM would probably work better than mixing SVD and RBM.

	PA	**kNN**	**SVD**	**RBM**
Average rating	501.176	504.942	500.701	501.662

Table 5.2.1.b: Average predicted rating for all important methods

The likely reason that the machine learning approaches work only one to five MAE points better than the PA baseline method is due to the dataset that was used. The dataset is constructed of data that does not have clearly specified ratings per photo by a user as in other recommendation systems, but instead is made up of batches of fifty photos that are put in order of preference by the user and then given somewhat arbitrary ratings by the software. Furthermore, the user will often not carefully put the photos in order but might only do this with photos he or she finds interesting, often resulting in more arbitrary ratings from the user for most photos. There is simply too much noise in the dataset to make the machine learning approaches perform clearly better than the PA baseline method. This might mean that only the low and high ratings are actually meaningful, which will be discussed in the next section.

The reasons stated in the above paragraph also made it unlikely for a blending method to lower the MAE in a way that made up for the noise in the data set. Therefore, blending was not experimented with any further in this research after the literature study.

5.2.2 Ranges of ratings distribution

Table 5.2.2.a shows the predicted ratings range for all relevant methods. This table shows that even though kNN is overall a better performer than SVD and RBM (as concluded in the discussion of the overall performance of the methods), RBM performs best in the 1-200 range, making RBM the top choice when looking to make predictions in the 801-1000 range.

Predicted ratings range	**PA**	**kNN**	**SVD**	**RBM**
1-200	127.819	124.441	131.966	**113.771**
201-400	174.161	176.075	175.511	**172.502**
401-600	201.942	201.529	**201.358**	203.903
601-800	186.832	187.882	**183.661**	187.250
801-1000	139.823	136.893	**127.656**	128.226
Average / Total	*186.857*	***182.553***	*185.263*	*185.829*

Table 5.2.2.a: Predicted ratings range for all relevant methods

Table 5.2.2.b shows the number of ratings for all important methods for the ratings ranges. Even though SVD (and RBM) are the best performers in the 801-1000 ratings range, they have less than half the number of predictions in that ratings range of kNN.

Predicted ratings range	**PA**	**kNN**	**SVD**	**RBM**
1-200	52,011	**94,733**	54,741	55,530
201-400	339,384	332,700	**345,379**	331,659
401-600	556,776	466,403	550,048	**561,592**
601-800	400,717	**403,315**	386,367	397,777
801-1000	21,461	**73,125**	33,814	23,791
Average / Total	*1,370,349*	*1,370,276*	*1,370,349*	*1,370,349*

Table 5.2.2.b: Number of ratings for ratings ranges for all relevant methods

As the focus of this research is to get as many high quality predictions in the high rating ranges of 801-1000 as possible, the overall best performers here is SVD. RBM is a close second, with their MAE only differing by 0.57. However, RBM has almost 30% less predictions than SVD, meaning that SVD seems to be the better algorithm to use when looking at both quality and quantity of the predictions. KNN can also be used when needing more predictions while sacrificing some quality.

5.2.3 Computational cost

This section compares both the offline (preliminary work) and the online (calculating the actual predictions using the preliminary work) computational costs of the relevant methods. For information on the terms used in the table see the implementation details of the methods in the experimental setup chapter.

Table 5.2.3.a shows the offline computational cost of the relevant methods. PA is clearly the fastest method, whereas the offline performance of both SVD and RBM were close. The kNN approach did not have an offline computational cost.

	PA	**kNN**	**SVD**	**RBM**
Offline Computational cost	***O(s-p)***	-	*O(k*p*q)*	*O(h*e*s)*

Table 5.2.3.a: Offline computational cost for all relevant methods

In practice, the online computational costs are much more important, as these have to be

calculated quickly when needing predictions in a small timeframe. These are shown in table 5.2.3.b, showing that PA is clearly the fastest here. The SVD and RBM methods' performances are quite close to each other, with the slight edge to SVD as even though three terms are of SVD is to the third, in practice the RBM is slightly slower. The problem with kNN is that it scaled way worse because it does the entire calculation online per separate prediction, while the SVD and RBM do a lot of work in the offline computation. And while SVD and RBM were by far fast enough in practice with calculating predictions after the offline computations were done, for kNN all of the calculations were done in the online part. This means that the best machine learning algorithms in this category here are SVD and RBM, over kNN.

Keep in mind that in table 5.2.3.b the term *k* in the order for kNN and SVD stand for different things (number of nearest neighbors versus order of the matrix decomposition).

	PA	**kNN**	**SVD**	**RBM**
Online Computational cost	***O(1)***	*O((m*p)+(k))*	*O(k)*	*O(v*h)*

Table 5.2.3.b: Online computational cost for all relevant methods

For kNN, calculating the nearest neighbors could be done beforehand, but this means a lot of redundant work because most user similarity that would have been calculated cannot be used to create the predictions for the exact photo-user combinations that were in the test set. This was not doable for the hardware used so was not done in this experiment.

5.2.4 Memory cost

As can be seen in table 5.2.4.a, PA is also the best method when considering the memory cost. The most expensive costs between the offline and online part of the algorithms are shown. SVD and RBM were both efficient enough in terms of memory so their cost was never an issue. The memory costs of kNN were a problem on the experiments, since kNN requires a lot more specialized data structures to store the sets for fast retrieval using hash maps. Iterators could be used for SVD and RBM to walk over the training entries.

	PA	**kNN**	**SVD**	**RBM**
Memory cost	***p***	$2*s + p^2$	*s+((m+n)*k)*	*s+5*v+5*h*

Table 5.2.4.a: Memory costs for all relevant methods

The storage cost is not big enough to be an issue for the training and test sets in this research.

6 Conclusion

This chapter answers the research questions, presents possible options for future work and suggests recommendations to Paiq.

6.1 Answering the research questions

Having done the evaluation of the methods in the previous chapter, the research questions are answered below.

R1. Which of the researched machine learning approaches or baseline methods is best for creating recommendations on the Paiq dataset considering accuracy?

The kNN method is the best machine learning approach for creating recommendations on the Paiq dataset considering accuracy, as it has the lowest overall MAE of 182.552. It is slightly better than the other methods, only differing with the highest overall MAE of RBM by 4.886. The PA baseline method is best to use if either computational cost or memory costs are an issue.

a. Which of the researched machine learning approaches or baseline methods is best for creating predictions on the higher rating range, considering accuracy?

SVD is the best machine learning approach for this research question, as it has the lowest MAE of 127.656 on the 801-1000 rating range. The MAE of RBM on this rating range is only slightly higher (128.226) and is therefore a good alternative to SVD. It is best to use kNN if the MAE of 136.893 is deemed to be low enough, giving over twice the amount of predictions that any of the other methods do.

R2. Which of the researched machine learning approaches is best for creating recommendations on the Paiq dataset considering computational and memory costs?

The answer is two approaches: both the computational and memory costs of SVD and RBM are quite similar, with a slight edge to the SVD due to computational cost.

6.2 Future work

This section shows some possible approaches for future work:

- Refine the machine learning approaches:
    - Using refinements such as a continuous RBM (CRBM) that replaces the binary stochastic unit by a continuous one, exchanging its deterministic behavior for binary-stochastic behavior [72].
- Change the dataset:

    - Remove more noise from the database'.
    - Only train on the low/high scoring ratings.
- Change the type of machine learning algorithm:
    - Use a ranking machine learning algorithm, one that orders the predictions instead of calculating hard errors.
- Change the rating system:
    - Using a different rating system where the user's input is directly converted to a photo rating should clear up a lot of noise. The entire dataset will have to be rebuilt though.

6.3 Recommendations to Paiq

This research has shown that using a baseline recommender is almost as effective MAE-wise as machine learning methods because of the noise in the Paiq dataset, but that the SVD method provides the best quality predictions in the higher rating ranges. Therefore this research suggests that if Paiq were to implement an algorithm, it implements SVD and uses it for its predictions in the higher rating range, or kNN if needs more predictions in the higher rating ranges than SVD can give. Taking into account the scalability issues of kNN as well as its lower performance on the 801-1000 ratings range, the recommendation is to not use kNN, unless the higher number of predictions in that ratings range for kNN is required. RBM can be used as an alternative for SVD.

Other options are to change the photo rating app to either:

- Let people straight up give ratings from 1 to 10.
- Pair photos against another, and using all ratings information for a person to give the photos a rating, instead of just using a set of 50 photos.

References

1. Gediminas Adomavicius and Alexander Tuzhilin; Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions; IEEE Transactions on Knowledge and Data Engineering archive, Volume 17 Issue 6, June 2005, Page 734-749
2. Carme Julià, Angel D. Sappa, Felipe Lumbreras, Joan Serrat, Antonio López; Predicting Missing Ratings in Recommender System: Adapted Factorization Approach; International Journal of Electronic Commerce, Winter 2009-10, Vol. 14, No. 2, pp. 89-108; M.E. Sharpe, Inc. Copyright 2009
3. Kangning Wei, Jinghua Huang, Shaohong Fu; A survey of e-commerce recommender systems; Tsinghua University and Beijing Normal University, China
4. Kangas, S.; Collaborative Filtering and Recommendation Systems; Research Report TTE4-2001-35
5. Mark Claypool, Phong Le, Makoto Waseda and David Brown; Implicit Interest Indicators; IUI'01, January 1417, 2001, Santa Fe, New Mexico, USA.
6. Diane Kelly, Jaime Teevan; Implicit Feedback for Inferring User Preference: A Bibliography; Newsletter ACM SIGIR Forum Homepage archive Volume 37 Issue 2, Fall 2003 Pages 18 - 28
7. Douglas W. Oard, Jinmook Kim; Implicit Feedback for Recommender Systems; Published in 1998, pages 81-83. Technical Report WS-98-08. AAAI Press, in AAAI Workshop on Recommender Systems, Madison, WI.
8. Xiaoyuan Su and Taghi M. Khoshgoftaar; A Survey of Collaborative Filtering Techniques; Journal Advances in Artificial Intelligence archive Volume 2009, January 2009 Article No. 4 Hindawi Publishing Corp. New York, NY, United States
9. Lam, S.T.K., Riedl, J.; Shilling Recommender Systems for Fun and Prot; Journal Advances in Artificial Intelligence archive Volume 2009, January 2009 Article No. 4 Hindawi Publishing Corp. New York, NY, United States
10. Chirita, P-A., Nejdl, W., Zamfir, C.; Preventing shilling attacks in online recommender systems; WIDM'05 Bremen, Germany
11. Schein, A. I., A. Popescul, L. H. Ungar, and D. M. Pennock; Methods and metrics for cold-start recommendations; In Proc. of the 25th Annual Intl. ACM SIGIR Conf., 2002
12. Brand, M.; Fast online SVD revisions for lightweight recommender systems; In <<ed.>> SIAM Conference on Data Mining. Montreal: <<publisher>> 2003, pp. 37–46.
13. Herlocker, J. L., J. A. Konstan, L. G. Terveen, and J. T. Riedl; Evaluating Collaborative Filtering Recommender Systems; ACM Transactions on Information Systems, 22(1):5-53, 2004
14. Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu; A density-based algorithm for discovering clusters in large spatial databases with noise; 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)
15. Herlocker, J. L., J. A. Konstan, A. Borchers, and J. Riedl; An algorithmic framework for performing collaborative filtering; In Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99). 1999
16. Mooney, R. J. and L. Roy; Content-based book recommending using learning for text categorization; In ACM SIGIR'99. Workshop on Recommender Systems: Algorithms and Evaluation, 1999
17. Robert M. Bell, Yehuda Koren and Chris Volinsky; The BellKor solution to the Netflix Prize; Netflix Prize (2007), related to ICDM'2007 and KDD-Cup '2007 papers
18. Shanhong Luo and Guangjian Zhang; What Leads to Romantic Attraction - Similarity, Reciprocity, Security, or Beauty - Evidence From a Speed-Dating Study; Journal of Personality, Vol. 77, No. 4.,

pp. 933-964
19. Kumar, R., P. Raghavan, S. Rajagopalan, and A. Tomkins; Recommendation Systems: A Probabilistic Analysis; Journal of Computer and System Sciences, 63(1): pp42-61, 2001
20. Abdelwahab, Amira Sekiya, Hiroo; Matsuba, Ikuo; Horiuchi, Yasuo; Kuroiwa, Shingo; Nishida, Masafumi; An Efficient Collaborative Filtering Algorithm using SVD-free Latent Semantic Indexing and Particle Swarm Optimization; International Conference on Natural Language Processing and Knowledge Engineering, 2009. NLP-KE 2009. 24-27 Sept. 2009
21. Jon Herlocker, Joseph A. Konstan, John Riedl; An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms; Journal, Information Retrieval archive, Volume 5 Issue 4, October 2002, Pages 287 - 310
22. Shardanand, U. and P. Maes; Social information filtering: Algorithms for automating ‘word of mouth’; In Proc. of the Conf. on Human Factors in Computing Systems, 1995
23. Yehuda Koren and Robert Bell; Advances in Collaborative Filtering; Recommender Systems Handbook, 2011, pp 145-186
24. Yehuda Koren; Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model; Proceeding KDD '08 Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining Pages 426-434
25. Sarwar, M.; Karypis, G.; Konstan, J.; and Riedl, J.; Application of dimensionality reduction in recommender system - a case study; In Workshop on Web Mining for E-Commerce. Boston: , 2000, pp. 133–151
26. Katz, G.; Shani, G.; Shapira, B.; Rokach, L.; Using Wikipedia to Boost SVD Recommender Systems; arXiv preprint arXiv:1212.1131; Dec 5, 2012
27. Koren, Y.; The BellKor Solution to the Netflix Grand Prize; August 2009
28. Bell, R.M.; Koren, Y.; Volinsky, C.; Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems; Conference on Knowledge Discovery and Data Mining, KDD’07, August 12–15, 2007, San Jose, California, USA
29. Andoni, A.; Indyk, P.; Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions; COMMUNICATIONS OF THE ACM January 2008/Vol. 51, No. 1
30. Liu, T.; Moore, A.W.; Gray, A.; Yang, K.; An Investigation of Practical Approximate Nearest Neighbor Algorithms; In proceedings of Neural Information Processing Systems(NIPS 2004), Vancouver, BC, Canada, 2004
31. Beis, J.S.; Lowe, D.G.; Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces; 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 17-19 Jun 1997, Page(s): 1000 - 1006
32. Athitsos, V.; Potamias, M.; Papapetrou, P.; Kollios, G.; Nearest Neighbor Retrieval Using Distance-Based Hashing; Proceedings of IEEE International Conference on Data Engineering (ICDE), 7-12 April 2008, Pages: 327 - 336
33. Slaney M., Casey M.; Locality-Sensitive Hashing for Finding Nearest Neighbors; IEEE Signal Processing Magazine, Volume 25, Issue 2, p.128–131 (2008)
34. Bell, R.M.; Koren, Y.; Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights; ICDM '07 Proceedings of the 2007 Seventh IEEE International Conference on Data Mining, Pages 43-52
35. Sarwar, B., G. Karypis, J. Konstan, and J. Riedl; Item-based Collaborative Filtering Recommendation Algorithms; In Proc. of the 10th International WWW Conference, 2001
36. Kostelich, E.; Kuhl, D.; Introduction to SVD and Applications; MSRI Climate Change Summer School, July 18, 2008
37. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J.; Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems; 5th International Conference on Computer

and Information Technology (ICCIT), 2002
38. Billsus, D.; Pazzani, M.; Learning Collaborative Information Filters; In 15th International Conference on Machine Learning. Madison, 1998, pp. 46–54.
39. http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/svd.html; Dimensionality Reduction and the Singular Value Decomposition; Recommender systems, A Computer Science Comprehensive Exercise Carleton College, Northfield, MN
40. http://www.igvita.com/2007/01/15/svd-recommendation-system-in-ruby/; Grigorik, I.; SVD Recommendation System in Ruby; 1 January 2007
41. Deerwester, S.; Dumais, S.T.; Furnas, G.W.; Landauer, T.K.; Harshman, R.; Indexing by Latent Semantic Analysis; Journal of the American Society for Information Science (1986-1998); Sep 1990; 41, 6
42. Paterek, A.; Improving regularized singular value decomposition for collaborative filtering; Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining (2007), pp. 39-42
43. Stewart, G.W.; On the Early History of the Singular Value Decomposition; SIAM Review, Vol. 35, No. 4 (Dec 1993), pp. 551-566
44. Berry, M.W.; Dumais, S.T.; O'Brien, G.W.; Using Linear Algebra for Intelligent Information Retrieval; SIAM Review, Vol. 37, Issue 4, Dec. 1995, pp. 573 - 595
45. Kim, D.; Yum, B.-J.; Collaborative Filtering Based on Iterative Principal Component Analysis; Expert Systems with Applications, Volume 28, Issue 4, May 2005, Pages 823–830
46. Shardanand, U.; Social Information Filtering for Music Recommendation; CHI '95 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pages 210-217
47. Elnaz B.; Comparing accuracy of cosine-based similarity and correlation-based similarity algorithms in tourism recommender systems; ICMIT 2008. 4th IEEE International Conference on Management of Innovation and Technology, 21-24 Sept. 2008, pages 469 - 474
48. http://brenocon.com/blog/2012/03/cosine-similarity-pearson-correlation-and-ols-coefficients/; O'Connor, B.; Cosine similarity, Pearson correlation, and OLS coefficients
49. Sarwar, B. M.; Karypis, G.; Konstan, J. A.; Riedl, J.; Recommender Systems for Large-Scale E-Commerce: Scalable Neighborhood Formation Using Clustering; 5th International Conference on Computer and Information Technology (ICCIT), 2002
50. Gong, S.J.; A Collaborative Filtering Recommendation Algorithm Based on User Clustering and Item Clustering; JOURNAL OF SOFTWARE, VOL. 5, NO. 7, JULY 2010, PAGES 745 - 752
51. Koren, Y.; Bell, R.; Volinsky, C.; Matrix Factorization Techniques for Recommender Systems; IEEE Computer Magazine, Volume 42, Issue 8, p.30-37 (2009)
52. Jahrer, M.; Combining Predictions for Accurate Recommender Systems; KDD '10 Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, Pages 693-702
53. Zhang, Z.; Cuff, P.; Kulkarni, S.; Iterative Collaborative Filtering for Recommender Systems with Sparse Data; 2012 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), Santander, Spain, 23-26 Sept. 2012, pp 1 - 6
54. Huang, Z.; Zeng, D.; Chen, H.; A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce; Intelligent Systems Journal, IEEE, Sept.-Oct. 2007,Volume: 22, Issue: 5, pps: 68 - 78
55. Kim, T.H.; Yang, S.B.; An Effective Recommendation Algorithm for Clustering-Based Recommender Systems; AI'05 Proceedings of the 18th Australian Joint conference on Advances in Artificial Intelligence, Pages 1150-1153, Springer-Verlag Berlin, Heidelberg, 2005
56. Yao, Z.; Zhang, Q.; Item-Based Clustering Collaborative Filtering Algorithm under High-Dimensional Sparse Data; International Joint Conference on Computational Sciences and Optimization. CSO

2009, 24-26 April 2009, Volume: 1, pps. 787 - 790
57. Miller, B. N., I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl; MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System; In Proceedings of the International Conference on Intelligent User Interfaces, Miami, Florida, 2003
58. www.netflixprize.com; Netflix; Netflix prize website
59. http://www.grouplens.org/node/12; GroupLens; MovieLens Data Sets
60. http://www.informatik.uni-freiburg.de/~cziegler/BX/; Institut für Informatik, Universität Freiburg; Book-Crossing Dataset
61. Ziegler, C.; McNee, S.; Konstan, J.; and Lausen, G.; Improving recommendation lists through topic diversification; In International World Wide Web Conference. Chiba, 2005, pp. 22–32
62. http://eigentaste.berkeley.edu/dataset; Berkeley; Anonymous Ratings from the Jester Online Joke Recommender System
63. Goldberg, K.; Roeder, T.; Gupta, D.; and Perkins, C.; Eigentaste: A constant time collaborative filtering algorithm; Information Retrieval Conference, 4 (2001), 133–151
64. Abdelwahab, A.; Sekiya, H.; Matsuba, I.; Horiuchi, Y.; Kuroiwa, S.; Collaborative filtering based on an iterative prediction method to alleviate the sparsity problem; iiWAS '09 Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services, Pages 375-379, ACM New York, NY, USA
65. Xu, B.; Bu, J.; Chen, C.; Cai, D.; An Exploration of Improving Collaborative Recommender Systems via User-Item Subgroups; WWW '12 Proceedings of the 21st international conference on World Wide Web, Pages 21-30, ACM New York, NY, USA
66. Yan, D.; Huang, L.; Jordan, M.I.; Fast Approximate Spectral Clustering; KDD '09 Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data minin, Pages 907-916, ACM New York, NY, USA, 2009
67. Balcan, M.F.; Blum, A.; Gupta, A.; Approximate Clustering without the Approximation; SODA '09 Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, Pages 1068-1077, Society for Industrial and Applied Mathematics Philadelphia, PA, USA
68. Klema, V.C.; Laub, A.J.; The Singular Value Decomposition: Its Computation and Some Applications; IEEE Transactions on Automatic Control, Apr 1980, Massachusetts Institute of Technology, Cambridge, MA, USA, Volume: 25, Issue: 2, pages: 164 - 176
69. Bell, R.M.; Koren, Y.; Lessons from the Netflix Prize Challenge; ACM SIGKDD Explorations Newsletter - Special issue on visual analytics, Volume 9 Issue 2, December 2007, Pages 75-79 , ACM New York, NY, USA
70. http://sifter.org/~simon/journal/20061211.html; Funk, S.; Netflix Update: Try This at Home; Monday, December 11, 2006
71. Salakhutdinov, R.; Mnih, A.; Probabalistic Matrix Factorization; ICML ’08: Proceedings of the 25th International Conference on Machine Learning
72. Chen, H.; Murray, A.F.; Continuous restricted Boltzmann machine with an implementable training algorithm; IEE Proc.-Vis. Image Signal Process., Vol. 150, No. 3, June 2003
73. Salakhutdinov, R.; Mnih, A.; Hinton, G.; Restricted Boltzmann Machines for Collaborative Filtering; 24th International Conference on Machine Learning, Corvallis, OR, 2007
74. Carreira-Perpinan, M.A.; Hinton, G.; On Contrastive Divergence Learning
75. Fischer, A.; Igel, C.; An Introduction to Restricted Boltzmann Machines; L. Alvarez et al. (Eds.): CIARP 2012, LNCS 7441, pp. 14–36, 2012
76. Smolensky, P.; Information processing in dynamical systems: Foundations of harmony theory; Rumelhart, D.E., McClelland, J.L. (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations, pp. 194–281
77. Hinton, G.E., Salakhutdinov, R.R.; Reducing the dimensionality of data with neural networks;

Science 313(5786), 504–507 (2006)
78. http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/; Chen, E.; Introduction to Restricted Boltzmann Machines
79. http://deeplearning.net/tutorial/rbm.html; Restricted Boltzmann Machines (RBM)
80. https://github.com/grouplens/lenskit/wiki/FunkSVD; Ekstrand, M.; Github grouplens / lenskit