# Categorised Neighborhood-based Collaborative Filtering

Marcel Jongsma

July 2016

## Abstract

*Bell and Koren produced an efficient and swift Collaborative Filtering algorithm, which improved the original Netflix algorithm by addressing some of the main concerns which were present. We copied the majority of this algorithm and included an extension. The question is whether this extension will provide an improvement to the existing algorithm.*

*The existing model makes use of the so-called jointly derived interpolation algorithm, we left out factorisation which was used in Bell and Koren (2007) and included a categorisation of the ratings. In short this means that user ratings labeled as 'high', 'middle' or 'low', will only be compared to similar 'high', 'middle' or 'low' user ratings. This is in contrast to the original where they are compared to all ratings. The results will be compared to our own reproduction of the existing algorithm by making use of the Root Mean Squared Error term. Lastly, we will discuss the results and shortcomings and make suggestions for future research.*

## Introduction

During daily life people rely a lot on recommendations from all kinds of sources. For instance, a friend recommends you a telephone that he himself uses; a certain product is recommended by your favorite television personality; or an online website recommends you products to buy. Where the former are forms of recommendation that are subjective, the latter is actually a programmed recommendation. It is therefore possible to optimise these recommendations of an online retailer. That is exactly what this research will focus on. More precisely, this research will try to find an algorithm that finds optimal recommendations for a dataset given by Netflix. Netflix tries to predict user ratings for movies as good as possible in order to properly recommend content. One of the first recommendation systems is the Tapestry system (Goldberg et al., 1992). It was the first system to make use of Collaborative Filtering (CF). A term used for filtering systems that makes use of past users' behavior to filter content. It is further elaborated on in many papers, in this paper we will have a look at the paper: 'Improved Neighborhood-based Collaborative Filtering' (Bell and Koren, 2007). We will use this and try to find an extension that might further improve the solution that was given.

The reason that CF is so important for companies can be explained quite easily. If unknown ratings of customers can be predicted with the least error, recommendations can be made according to these predictions. The better these recommendations are, the more satisfied a customer will be. Furthermore, because of the rise of big data, more and more information is available which should be put to good use. CF is a good example of large amounts of data being put to good use.

In this paper we will first describe the data and how it is handled, afterwards we will explain the methodology of the algorithm that will be put to use. Furthermore, the extension will be explained. Thereafter there will be a section for the results and finally we will discuss the outcomes and make conclusions.

### 1.1 Collaborative Filtering

First of all, we will further elaborate on CF. 'The fundamental assumption is that if users A and B rate k items similarly, they share similar tastes, and hence will rate other items similarly.' (Goldberg et al., 2001) In most cases CF makes use of a database of preferences, there is a list of $m$ users and $n$ items. Each user has rated a subset of the items. it is then possible to predict the unknown ratings of a user $u$ by using the rating information of $k$ other users which have rated similar on other items as user $u$. (Su and Khoshgoftaar, 2009)

CF can actually be divided in two sub-methods, namely **Memory-based Collaborative Filtering Algorithms** (which is the method that will be used for this paper) and **Model-based Collaborative Filtering Algorithms**. The latter actually makes item recommendations by developing a model of user ratings. Algorithms within this method take a probabilistic approach. The prediction is in most cases the expected value of the probabilities. (Sarwar et al., 2001)
So by using CF we can predict the preferences of users by looking at like-minded users and basing the predictions on these users. These predicted preferences are then used to make recommendations. One of the main occurring problems with CF is the sparsity issue. There are not enough rating points to make proper interpolation possible. Goldberg et al. (2001) solved this problem by creating a 'gauge' set, this was a set of movies that users had to rate regardless of having watched the movie or not. These ratings were based on a description of the movie. Of course the ratings will not be as accurate as they normally would be, however, this created a very dense subset, which can be used for better interpolation.
In our paper we will not use a gauge set and the consequences of sparsity might pose a problem, which we will try to solve by copying the jointly derived interpolation model as explained in Bell and Koren (2007). We will make use of the Netflix contest prize data and try to predict the ratings as good as possible.

## Data

Our data is a data set from Netflix that was published for the Netflix Prize competition. It was published in 2006 and includes a training set that consists of 100,480,507 ratings that 480,189 users gave to 17,770 movies, this training set is set up in the following way: It contains text files for every movie with the Movie IDs ranging from 1 to 17770. Inside these movie files you can find customer IDs, their ratings and the date on which the rating was made. The Customer IDs range from 1 to 2649429, with gaps in between. The ratings are on a scale from 1 to 5 and the dates are of the format YYYY-MM-DD. The data set also contains a probe set, which is a subset of the training set. It contains about 1,400,000 ratings and is structured as one big text file. Where you can find pairs of movies and users. The probe set can be used to test the model that has been created with the training set. Furthermore, the data set includes a qualifying set with movie IDs, Customer IDs and Dates to make predictions that can be submitted to Netflix to participate in the contest. This paper does not participate in the contest, as the contest has finished. Therefore, we can ignore the qualifying set.

In this paper the data is being handled by using the software Matlab. Because of the size of the data set it is difficult to handle the data in an efficient way and because a large data set makes running times very long, in this paper there was chosen to use a subset of the data set. All the users were included, however, only a subset of the movies was taken. The first 2000 movies were taken in the subset and as mentioned earlier, the probe set was also part of the training set, but since we want to use this as a test set, we removed the probe set from the training set. So from here on after the probe and training set are two seperate data sets, which do not overlap.

It is not possible to load in all the data at once, because there will not be enough memory available to make this possible. Instead, files are temporarily opened and useful information is taken out, once done, the file will be closed again. To work in this way efficiently, the first step was to create a file for every user, with rating information. Having a file for every user and every movie made it possible to program everything that was necessary.

During the rest of the paper, different users will be referred to with the subscripts $u$ and $v$, whereas different movies will be referred to with subscripts $i, j$ and $k$. So a rating from user $u$ on movie $i$ will be denoted as $r_{ui}$.

| *No. of ratings* | *Frequency* |
|---|---|
| 0 | 66318 |
| 1-5 | 120440 |
| 6-10 | 66897 |
| 11-20 | 77397 |
| 21-50 | 93456 |
| 51-100 | 41313 |
| 101-200 | 12824 |
| 201-400 | 1431 |
| 401-800 | 93 |
| 801+ | 14 |

**Table 1:** *Frequency of ratings per user*

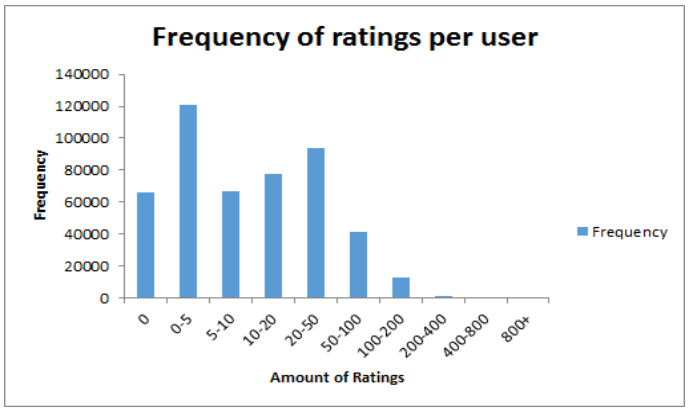


**Figure 1:** *Frequency of ratings per user*

If we take a closer look at the data of our subset we come to some important findings. For instance, if we look at table 1 and figure 1 we can see that more than 10% of the users in our subset has not given any rating at all and therefore has no information that can be used. Furthermore, about half of the users in the subset have not rated more than 20 movies. Sparsity already is an issue regarding CF, however, with this subset we will have even more difficulties on this matter. We have to take these facts into account when looking at the results later on.

| *No. of ratings* | *Frequency* |
|---|---|
| 0 | 0 |
| 1-50 | 2 |
| 51-100 | 116 |
| 101-200 | 419 |
| 201-500 | 424 |
| 501-1000 | 267 |
| 1001-2000 | 187 |
| 2001-5000 | 245 |
| 5001-10000 | 128 |
| 10001+ | 212 |

**Table 2:** *Frequency of ratings per movie*

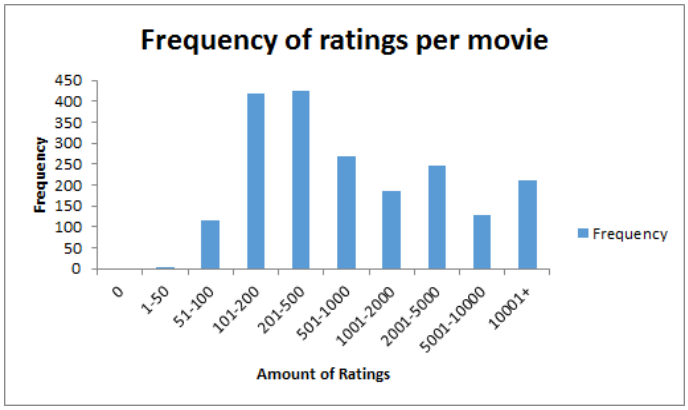


**Figure 2:** *Frequency of ratings per movie*

Similarly, we can have a look at table 2 and figure 2 to see that the movies do have sufficient support. There is not a movie that has not been given any ratings. Here it seems that there is plenty of information per movie. This is of course not a strange result given that all the users are included in the subset. So when we encounter sparsity problems, it would come from the users not having enough ratings.

# Methodology

## 3.1 Normalisation

To be able to compare ratings across users and movies, different rating behaviour has to be taken into account before using these ratings in the prediction algorithm. Therefore, we have to remove the 'global effects' (Bell and Koren, 2007). These are movie or user characteristics that influence the ratings. The first global effects that are addressed, are the user effect and the movie effect. To explain: certain users might rate more optimistic, whereas others rate lower on average. The same applies for movies: some movies have higher average rating than others. This effect can be estimated after subtracting the overall mean from the ratings and regress the residual as follows:

$$r_{ui} = \theta_u x_{ui} + \epsilon_{ui} \tag{1}$$

Where $\theta$ is either a parameter per movie or per user and the dependent variable $x_{ui}$ is equal to 1 for the movie and user effects. To prevent overfitting, the estimated $\hat{\theta}$ will be shrunk towards a common value:

$$\frac{n_u \theta_u}{n_u + \alpha} \tag{2}$$

$n$ is the amount of ratings by user $u$ and $\alpha$ is a constant which is determined by using cross-validation. The shrinkage is based on a Bayesian perspective and the derivation of this formula can be found in Bell and Koren (2007) on page 10.

From here on new global effects will always be estimated on the residual after removing previous global effects. Following, time-related global effects will also be taken into account for. Sometimes user and movie behaviour changes over time. For instance, a movie can get higher ratings just after its release compared to a year later. Equivalently, a user might be more positive at the first rating, but get more critical as time drags on. Therefore, in this paper there will be normalised for the following time effects: $user \times time(user)^{\frac{1}{2}}$, $user \times time(movie)^{\frac{1}{2}}$, $movie \times time(user)^{\frac{1}{2}}$ and $movie \times time(movie)^{\frac{1}{2}}$. Where time(user) stands for the time between the current rating and the first rating that user has given. Similarly, time(movie) is the time between current rating and the first rating that was given to that movie.

The effects will be calculated step by step, and at every step the previous effects will be subtracted from the existing residuals. The newly formed residual will be the dependent variable and the explanatory variable ($x$) will be the square root of time since first rating of movie/user, where time is in days.

If these user and movie time effects have been dealt with, we can finally normalise for the average rating of a movie or user and the support of a movie or user. With these we mean the average of the ratings that have been given to a movie or given by a user and the amount of ratings that a movie has received or a user has given. If we rewrite these descriptions into $\theta \times x$ again, we end up with: *user* $\times$ *movie average*, *user* $\times$ *movie support*, *movie* $\times$ *user average* and *movie* $\times$ *user support*.

### 3.2 Neighborhood model

After removing the global effects from the ratings, we are left with normalised ratings $r_{ui}$. Then we will further work from an item-perspective. To predict the unknown rating of a specific user on item $i$, we will find the $K$ most similar items to $i$ and base the prediction on these $K$ items by using interpolation. We will call the set of K neighbouring items for item $i$ and user $u$ $N(i, u)$. A prediction of a rating will therefore look as follows:

$$\hat{r}_{ui} = \sum_{j \in N(i,u)} w_{ij} r_{uj} \tag{3}$$

To be able to predict the weights $w_{ij}$ we will have a look at all users that have rated movie $i$ and also rated movies that are in $N(i, u)$, so theoretically we will use perfect neighbours that rated all the $K$ similar movies to $i$. An adjustment will be made later. We have to estimate all weights at the same time to take the inter dependencies into account (Bell and Koren, 2007). By doing this our optimisation problem will look like this:

$$\min_{w} \sum_{v \neq u} \left( r_{vi} - \sum_{j \in N(i;u)} w_{ij} r_{vj} \right)^2 \tag{4}$$

In this equation every user $v$ that has rated $i$ and the items in $N(i; u)$ will be added in the equation. If we differentiate this equation and write it in matrix form we will get the following notation:

$$Aw = b \tag{5}$$

Where $A$ is a $K \times K$ matrix defined as

$$A_{jk} = \sum_{v \neq u} r_{vj} r_{vk} \tag{6}$$

and $b$ a $K \times 1$ vector defined as:

$$b_j = \sum_{v \neq u} r_{vj} r_{vi} \tag{7}$$

The problem that arises with this method is the sparsity issue. In this formulation we only take a look at the users that have rated all the $K$ similar items, so we leave out a lot of information. As a solution we can take the information based per item. So every user that has rated $i$ and $j$ will be put in the equation. Because not every pair of movies has the same amount of information the elements of the $A$ matrix have to be averaged on the support, which results to the following formulation:

$$\bar{A}_{jk} = \frac{\sum_{v \in U(j,k)} r_{vj} r_{vk}}{|U(j,k)|} \tag{8}$$

$$\bar{b}_j = \frac{\sum_{v \in U(i,j)} r_{vj} r_{vi}}{|U(i,j)|} \tag{9}$$

Here $U(j,k)$ is the set of all users that rated both $j$ and $k$. To improve the current definition even more, we can also adjust the averages that are based on little support, by shrinking them to a common value, a baseline value. The baseline value is the average of all the values in the $\bar{A}$ matrix, where we do make a distinction between diagonal values and non-diagonal values. Diagonal values tend to be a bit higher, since they will always be the sum of positive values. Non-diagonal values on the other hand can be the sum of positive and negative values, since after normalisation ratings can also be negative. Our formulation will then look as follows:

$$\hat{A}_{jk} = \frac{|U(j,k)|\bar{A}_{jk} + \beta \cdot avg}{|U(j,k)| + \beta} \tag{10}$$

$$\hat{b}_j = \frac{|U(i,j)|\bar{b}_j + \beta \cdot avg}{|U(i,j)| + \beta} \tag{11}$$

The $\beta$ in these equations controls the extent of the shrinkage. And the value that is chosen for $\beta$ in this paper is 500. So now we have defined our best estimates for $A$ and $b$ our final formulation will look like this:

$$\hat{A}w = \hat{b} \tag{12}$$

This derivation is based on and entirely similar to the derivation from Bell and Koren (2007). The resulting weights that are gained from this regression are multiplied with corresponding $r_{uj}$ with $j \in N(i,u)$ to get the 'normalised' prediction $\hat{r}_{ui}$, which we then de-normalise to get our real prediction.

### 3.3 Predictions

Now that the model is defined, we will shortly explain how the ratings are actually predicted. The first step is to create the $K \times K$ matrix $A$. When looking at user $u$ and item $i$ we select the $K$ most correlated movies to $i$, which user $u$ has rated. This can be done by looking at the Pearson Correlation coefficients between $i$ and other movies $j$. However, the Pearson correlation does not give the most accurate value (as some pairs $i$ and $j$ do not have enough observations to calculate a correct correlation value). Therefore, the Pearson Correlation coefficient will also be shrunk based on its support (Exact definition can be found in Section 3.4) and the resulting value will be used here. So now we have chosen the $K$ most correlated movies using the adjusted Pearson Correlation coefficient, we can create the matrix $A$ that is used for calculation. Similarly, the $b$ vector can also be created.

The most simple solution to predict the value of $r_{ui}$ is by using Least Squares (LS). However, as mentioned in Bell and Koren (2007), giving a restriction to $w$ to be non-negative, slightly improved the solution. So non-negativity constraint is also added in. We then end up using the build in matlab function 'lsqnonneg'.

After calculating the weights, we can can multiply them with the corresponding movie residuals $r_{uj}$, with $j \in K$. The resulting $r_{ui}$ then has to be de-normalised by adding the mean rating over the data set and adding back the $\hat{\theta}$ that were calculated during normalisation.

We will test the model on the probe set, however, as we have seen in the Data section, a lot of users do not have any ratings to base a prediction on. Therefore, excluding these users yields better results. This proved to be vital while evaluating the model performances.

### 3.4 Extension

As extension this paper took a look at the $A$ matrix that was in the formulation described in Section 3.2 and made a slight adjustment. Before we elaborate on this, lets first shortly recap how the $A$ matrix is build up. Every element $j, k$ of the $A$ matrix is the sum over all users that rated item $j$ as well as item $k$, this user could have given item $j$ any rating between 1 and 5, and this has no connection to the user for whom we will predict the rating. So the idea of the extension is that we do have a look at the rating of the user and will take this into account when making the $K \times K$ matrix $A$.

To make a connection between element $j, k$ of $A$ and for instance user $u$, we can look at the rating that the user has given for movie $j$. If this is high, we should compare it to other users that have given a high rating to item $j$ as well. And we can do this by filtering the $A$ matrix for only high ratings of $j$. So it will look like this:

$$\bar{A}h_{jk} = \frac{\sum_{v \in H(j,k)} r_{vj} r_{vk}}{|H(j,k)|} \tag{13}$$

Equivalently to $\bar{A}h$ being the 'high' category, $\bar{Am}$ and $\bar{A}l$ will represent the 'middle' and 'low' category respectively. H(j,k) will be the set of user that rated $j$ above a certain threshold to make it a $'$high$'$ rating and rated both $j$ and $k$. Note that $k$ could have any rating. In this way we can create full $A$ matrices for different rating levels. In this paper 3 levels were chosen: high, middle and low. Note that the thresholds are chosen by making sure that every category has about one third of the total data in it. So we divide the full $A$ matrix in three separate matrices. Similarly, the $b$ vector will be made in an equivalent way.

After we have defined $A$ and $b$ we have to implement them in the prediction algorithm. Normally for given user $u$ and a movie rating $i$ that has to be predicted, we look at the movies that user $u$ has rated and select the $K$ most correlated with $i$. In the extension, however, we look at the residual ratings that user $u$ has given to other movies. Take for instance movie $j$ that has a residual rating categorised as a $'$high$'$ rating, first we check whether the $Ah$ matrix has enough information input, so that it can be use full. This is done by choosing a $\gamma$ value, which indicates the least amount of observations an element of a category matrix needs before it can be included. If the $Ah$ matrix has enough support, we take the correlation factors of the $'$high$'$ category between movie $i$ and $j$. And compare this value with the other correlation factors, which also might be from the high, middle and low categories if enough input.

We end up with a list of correlations and just like the regular algorithm, we will choose the most correlated movies from this list. Now we construct the $A$ matrix, but in a slightly different way. We have our list of movies that will be in the $A$ matrix, $j \in K$. Per row $j$ all elements $k \in K$ will be chosen from the category that $j$ was in. So if $j$ was in the $'$high$'$ category, the whole row $j$ will be created by taking it from the $Ah$ matrix. Now adding all rows up we get an final $A$ matrix which is made up of different rating categories. The reason that the whole row is taken from the $'$high$'$ matrix is because row $A_j$ times the weights will determine $b_j$ and $b_j$ is also from the $'$high$'$ category. Therefore, all elements that determine $b_j$ should be from the $'$high$'$ category

Interesting to note is that the resulting matrix will not be symmetric any more, this will not influence the weights, since they are calculated using LS, which does not require a matrix to be symmetric.

### 3.5 Preprocessing

For both the regular algorithm and the extension it is time-efficient to calculate some of the values beforehand. Furthermore, we still have to define the correlation coefficients that are used to choose the $K$ most correlated movies. As explained earlier a Pearson-Correlation coefficient will be taken between every movie $i$ and $j$ and this value will be shrunk to a more common value. Therefore, we end up with the following definition:

$$s_{ij} = \frac{|U(i,j)|}{\sum_{u \in U(i,j)} (r_{ui} - r_{uj})^2 + \alpha} \tag{14}$$

An $\alpha$ value of 10 was used in this research. This might not be the optimal value, however, test to find an optimal $\alpha$ value were not performed out in this paper.
The $s_{ij}$ values will be precomputed for every available combination of $i$ and $j$, so that every value can be easily accessed.

The values of $\hat{A}$ and $\hat{b}$ should be precomputed for the original and extension algorithm. The definition of these can be found in Sections 3.2 and 3.4. Note that only $\hat{A}$ has to be precomputed, as the values of $\hat{b}$ can be retrieved from the precomputed $\hat{A}$ matrix. Also, recap that the $avg$ value that is added in the $\hat{A}$ matrix for shrinkage differs among diagonal and non-diagonal elements.

# Results

## 4.1 Normalised Residuals

As discussed in Section 3.1 we will normalise the results for multiple "global effects". To check the effectiveness of these global effects we calculate the Root Mean Squared Error (RMSE) of the residuals $r_{ui}$ of the probe set. The RMSE is calculated exactly as the definition says: It is the root of the average squared error. These errors are the residuals $r_{ui}$. If we predict the RMSE while step by step removing the "global effects" from the probe set we come to the following results:

| | $\alpha = 5$ | $\alpha = 25$ | $\alpha = 50$ | $\alpha = 150$ | $\alpha = 500$ | $\alpha = 1000$ | $\alpha = 2000$ | $\alpha$ | Impr. |
|---|---|---|---|---|---|---|---|---|---|
| Overall mean | **1,1266** | | | | | | | NA | NA |
| Movie effect | 1,0722 | **1,0721** | 1,0721 | 1,0726 | | | | 25 | 0,0545 |
| User effect | **1,0366** | 1,0432 | 1,0500 | | | | | 5 | 0,0355 |
| Us × T(Us)$^{1/2}$ | 1,1136 | 1,0484 | 1,0400 | 1,0364 | **1,0362** | 1,0364 | | 500 | 0,0004 |
| Us × T(Mo)$^{1/2}$ | 1,0452 | 1,0382 | 1,0372 | 1,0365 | 1,0363 | 1,0363 | 1,0363 | NA | NA |
| Mo × T(Mo)$^{1/2}$ | **1,0358** | 1,0358 | 1,0358 | 1,0358 | 1,0359 | | | 5 | 0,0004 |
| Mo × T(Us)$^{1/2}$ | **1,0350** | 1,0350 | 1,0350 | 1,0350 | 1,0351 | | | 5 | 0,0008 |
| Us × Mo avg | 1,0418 | 1,0365 | 1,0357 | 1,0352 | **1,0350** | 1,0350 | 1,0350 | 500 | 0,0000 |
| Us × Mo supp | 1,2160 | 1,0501 | 1,0394 | 1,0356 | 1,0351 | **1,0350** | 1,0350 | 1,000 | 0,0000 |
| Mo × Us avg | 1,0353 | 1,0353 | 1,0353 | 1,0353 | 1,0353 | 1,0353 | 1,0353 | NA | NA |
| Mo × Us supp | 1,0352 | 1,0352 | 1,0352 | 1,0352 | 1,0352 | 1,0352 | 1,0352 | NA | NA |

**Table 3:** *RMSE improvements on probe set*

In the Table, Us stands for User, Mo stands for Movie and T means Time. The improvements on the RMSE of the probe set are seen here by using multiple $\alpha$ values for shrinkage. As mentioned in the Data section already, the normalisation was trained on the training set with the probe set excluded. Therefore, we can see this as the cross-validation results. The second to last column shows the chosen $\alpha$ for the best result. The last column shows the improvement since the previous RMSE. Note that the Overall mean does actually not include any $\alpha$, rather it is just the RMSE calculated by removing only the mean rating over the data set.

First thing that we can see, is that the RMSE does not go below the value 1. Unlike the results in Bell and Koren (2007), where the optimal RMSE (after removing the last global effect) is equal to 0,9657. Our result is of course undesirable, it is not strange however. Since we only use about 10% of the data, moreover, only 10% of the movies that were available in the training set. This causes about 90% less information per user. Further, as we exclude information of the user, the time since first rating, average rating and support are actually not true values anymore, and are for most users based on too little observations to be reliable. Therefore, almost all effects besides the main effects produce no to almost no improvement on the RMSE, since there is user information included in most of the global effects.

So finally we were able to reduce the RMSE by normalising to a value of 1,0350. This is exactly 0,0697 worse than Bell and Koren.

### 4.2 Precomputing Extension values

After normalising the data, we end up with a rating distribution as seen in figure 2. We can see that it appears to be normally distributed around zero, although it seems that the right side is a bit steeper than the left side. Following this figure we have to choose the thresholds for the extension categories. Since ratings are normalised the interpretation gets difficult, however, we will try to find thresholds that split up the data evenly in three categories, such that there is enough support for each category $\bar{A}$ matrix.

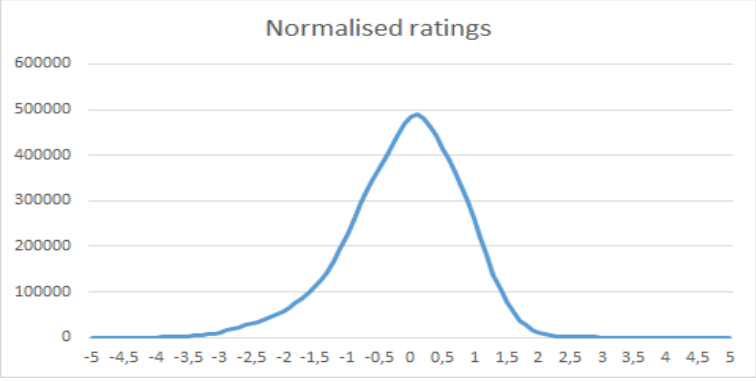


**Figure 3:** *Normalised Ratings*

If we take the first threshold to be at -0.5 and the second one at 0.5 we divide the data up as follows: the low category will contain a little bit more than 28% of the data, the middle category will contain about 43% of the data and the high category will also contain a little bit more than 28% of the data. Similarly to this result we will also use the thresholds -0.5 and 0.5 for the residuals where only the

main effects were removed and the residuals with only the mean rate removed. Results in Section 4.3 will be calculated with the thresholds as they were defined here.

### 4.3 Neighborhood model performances

In the methodology section we have defined two models in order to predict the ratings. In this section the results will be compared. First of all we study the results of the jointly derived interpolation algorithm as described in Bell and Koren (2007). Recall that the results were measured by predicting the values of the probe set, but by leaving out the users which have no ratings to base the prediction on. Furthermore, K = 5 and K = 10 were added to the results, since K = 35 and K = 50 produced rather odd results.

| Data normalisation | K = 5 | K = 10 | K = 20 | K = 35 | K = 50 |
|---|---|---|---|---|---|
| none (raw scores) | 1,0336 | 1,0307 | 1,0311 | 1,0315 | 1,0314 |
| Double centering | 0,9880 | 0,9845 | 0,9848 | 2,3511 | 3,2146 |
| Global effects | 0,9862 | 0,9844 | 0,9832 | 3,1836 | 2,2007 |

**Table 4:** *Jointly derived interpolation results*

We can see here again that our results are not as good as the results from Bell and Koren (2007), this is mainly caused by the fact that we have a subset, as stressed before. Another thing we can see is that for the normalised results (Double centering and Global effects), K = 35 and K = 50 produce weirdly high RMSEs. This could be caused by including movies in the prediction algorithm that have no actual correlation to the predicted movie anymore. However, this is in contradiction with the raw score, which performs consistently over the different values of K.

Secondly, we defined our extension model, with the use of different categories. It required twice as much precomputing time, but prediction times were only slightly higher than the jointly derived interpolation model.

| Data normalisation | K = 5 | K = 10 | K = 20 | K = 35 | K = 50 |
|---|---|---|---|---|---|
| $\gamma = 1$ | | | | | |
| none (raw scores) | 1,2093 | 1,2226 | 1,1447 | 1,1217 | 1,1123 |
| Double centering | 0,9907 | 0,9919 | 1,0007 | 1,0178 | 1,0339 |
| Global effects | 0,9889 | 0,9901 | 0,9985 | 1,0159 | 1,0348 |
| $\gamma = 5$ | | | | | |
| none (raw scores) | 1,2092 | 1,2222 | 1,1444 | 1,1216 | 1,1122 |
| Double centering | 0,9910 | 0,9925 | 1,0018 | 1,0191 | 1,0356 |
| Global effects | 0,9892 | 0,9906 | 0,9994 | 1,0170 | 1,0362 |
| $\gamma = 20$ | | | | | |
| none (raw scores) | 1,2097 | 1,2242 | 1,1426 | 1,1208 | 1,1109 |
| Double centering | 0,9918 | 0,9936 | 1,0037 | 1,0213 | 1,0381 |
| Global effects | 0,9900 | 0,9919 | 1,0014 | 1,0196 | 1,0394 |
| $\gamma = 50$ | | | | | |
| none (raw scores) | 1,2152 | 1,2348 | 1,1414 | 1,1189 | 1,1090 |
| Double centering | 0,9922 | 0,9945 | 1,0049 | 1,0229 | 1,4332 |
| Global effects | 0,9906 | 0,9927 | 1,0027 | 1,0211 | 1,0467 |

**Table 5:** *Extension results for a variety of* $\gamma$

In table 5 we can see the RMSE results of the extension over different values of $\gamma$. Recall that $\gamma$ is the minimum number of observations that are needed before a specific category $\hat{A}$ matrix will be used for prediction. The value of $\gamma$ is arbitrary, we chose to have a look at the values 1, 5, 20 and 50 to see if excluding less dense movies combinations from being categorised might improve the results. As can be seen, these values of $\gamma$ did not bring any improvement to the model, moreover, the raw scores perform worse than predicting with the mean rate in most cases using the extension. The normalised results, however, perform rather well with low K. Almost as good as the jointly derived interpolation algorithm. For higher K this result gets worse, but interestingly enough, the values of K = 35 and K = 50 perform better using the extension algorithm. This might be caused by a programming error in the original algorithm. In our results, lower $\gamma$ values produce better RMSEs. More specifically, using only categorised $\hat{A}$ matrices produced the best result. This could indicate that the higher RMSE values of the extension are caused by taking the categorised matrices of movie combinations with higher observations, whereas movie combinations with high observations perform better without being categorised. But that is just speculating and should be tested.

## Discussion

By looking at our results, we can conclude that the extension defined in this setting did not provide any improvement to the existing model. The extension came close to matching the original models RMSEs. The loss of information by categorising was bigger than the predicting improvement of defining these categories.

Another thing that has to be noted is that handling the data in an efficient way is a big task on its own. Encountering several problems along the way that makes programming the results difficult. This can be largely seen back in the extension that was made in this paper. The precomputing time is significantly higher than the regular algorithm. Furthermore, since we only used a subset, this might pose a bigger issue when using the whole data set. This leads to the biggest limitation of our research, which is the fact that only a subset of the data is used. Consequently, a lot of data is lost. One of the main existing problems with Collaborative Filtering is the sparsity issue. By taking a subset this issue will only increase. To get competitive RMSE results all the data and the factorisation method should also be included in the algorithm.

So the extension did not bring any observable improvements, but by altering the $\gamma$ value or the thresholds that make up the categories, there might be some improvement possible by using this method. Moreover, if we have another look at table 5 we see that RMSE gets worse when basing the predictions on more neighbouring movies. An adjustment for future research could be to only categorise the most correlated movies and use the regular $\hat{A}$ matrix for the remaining neighbors. An implementation of this extension on bigger scale and testing for different input options might even yield a slight improvement compared to Bell and Koren (2007).

## References

R. M. Bell and Y. Koren. Improved neighborhood-based collaborative filtering. *ATT Labs*, 2007.

D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Association for Computing Machinery*, 1992.

K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *IEOR and EECS Departments, University of California*, 2001.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. *Department of Computer Science and Engineering, University of Minnesota*, 2001.

X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Department of Computer Science and Engineering*, 2009.