Internal Report 2012-20 August 2012

Universiteit Leiden
Opleiding Informatica

An Investigation into Recommendation Algorithms

Floris Kleyn

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

An Investigation into Recommendation Algorithms

Floris Kleyn

August 2012

Supervisors:
Michael S. Lew
Erwin Bakker

Contents

1 Introduction

Recommender systems are widely used on the internet today. Web shops try to recommend items to their customers, movies are recommended by video rental companies and websites try to recommend music to their visitors based on what they have listened to. The focus in this paper lies on recommending movies to users. There is probably nobody who likes to waste his time on a movie he does not like, but due to the enormous amount of movies around it is hard to select a good movie all by yourself. Of course, people could search for reviews of movies and try to make an educated guess if they would like it. But it would be much easier if someone just told them what they would like. Recommender systems try to do just that.

There are generally two possible approaches to provide these recommendations. Recommender systems can be content-based or collaborative-based, although a hybrid approach, combining both methods, is also possible. Content-based filtering is often based on features of the objects that need to be recommended (e.g. genre, place of events or plot for movies) and the recommendations are based on the user's previously rated items. Collaborative filtering does not depend on a good breakdown of items into features but uses the opinions of other users. Recommendations are done based on what other, similar, users liked. The following example, using the data in Table 1, will illustrate this. The goal is to predict the rating Simon would give to the movie Die Hard. If the predicted rating is high, then it could be recommended to Simon because in that case he will probably like that movie. All users are compared with Simon to determine if Simon would like Die Hard. Kate does not have any rating in common with Simon and all here ratings differ substantially with Simon's, Peter has only one rating in common and John all but one. John is the user who is the most similar to Simon and because John rated Die Hard with an 8, the prediction is that Simon would also rate Die Hard with an 8.

	Alien	Memento	Fight Club	Snatch	Die Hard
John	6	9	9	7	8
Kate	3	7	6	6	5
Peter	7	7	8	8	6
Simon	6	9	9	8	?

Table 1: Example data

To provide recommendations, one could also just simply predict a like or a dislike for a certain item and recommend those items the user would probably like. But by predicting the actual rating, and presenting that rating to the user when he looks up that movie, a user could determine if that rating is high enough for him to watch it. This will prevent the user from watching movies he would probably hate, but allows him to try also movies that are not highly recommended. Highly recommended movies could be presented to the user in a short list, and for that it is not required to predict his actual rating.

2 Related Work

In the Introduction both content based filtering and collaborative filtering were mentioned. Only collaborative approaches are examined in this paper due to the generally low performance of content based filtering approaches [13]. There are different approaches on how to accomplish collaborative filtering for recommendations, which can be grouped in the four clusters mentioned below. One can look at user-to-user collaborative filtering or item-to-item collaborative filtering. These two approaches share a similar basis: all the available data is directly stored in memory and the all data can be used to provide recommendations. These methods are therefore memory-based. This as opposed to model-based approaches where the

data must be modified first. The model is constructed based on the example data and after construction the model will be capable of providing recommendations. The fourth method is the hybrid approach mentioned in the introduction, these use content based filtering to improve collaborative filtering methods and can be either memory-based or model-based.

User-to-user collaborative filtering approaches use users who are similar to the test user to provide recommendations for him. A lot of these methods do not differ too much from each other. The main difference is how they determine if a user is similar to another user, although there are of course some other differences as well. The creators of the first recommender system, Tapestry [7], who came up with collaborative filtering, allowed users to rate messages or attach keywords to it and other users could then filter out messages by searching for messages a particular user liked. GroupLens [14] provided personal recommendations to its users for Usenet articles. Similarity between users was determined by calculating the Pearson coefficient between their ratings. Vector similarity was used by [5]. It is a method that is often used to measure how similar documents are by counting word frequencies in each document and use a cosine function to determine the similarity of the two vectors. Universal queries were discussed in [8], where users must rate some fixed items based on a description to create a set of items that everyone has rated to prevent that the data set is too sparse to do predictions.

Item-to-item collaborative filtering approaches are very similar to user-to-user approaches, only they search for similarities between items and use users to determine how similar the items are. A benefit of an item based approach is that, unlike a user based approach, the computational work does not increase when the number of users grows. Different techniques for computing the item-to-item similarities were examined by [15]. These techniques were also used in user based approaches (e.g. Pearson coefficient, vector similarity) and only needed some small adaptations for usage in an item based recommender system. The same methods were also used by [9, 6] but they also used conditional probability-based similarity: the probability a user would purchase (like) a certain item if it already purchased (liked) an item or set of items in the past. All the described methods report equal or better recommendations and faster computation time than the user-to-user approaches.

Model-based collaborative filtering approaches are often based on machine learning algorithms, a branch of artificial intelligence. A model is generated based upon available data and this model is used to predict future ratings, and thereby capable of recommending movies to users. It was suggested by [4] that collaborative filtering tasks could be seen as classification problems, which showed that machine learning algorithms could be applied to the field of recommender systems. Neural networks were used by [11] to predict a like or a dislike for a movie. They used user based neural networks as well as item based neural networks. A User Community Model (UCM) was presented by [2]. They clustered users into communities and searched for patterns within those communities. This method was later used by [1] to improve performance by using a Bayesian modeling approach to create a Bayesian UCM.

It is also possible to use additional information to increase performance. This can be accomplished by combining collaborative filtering with content based filtering techniques, creating a hybrid approach. Such a hybrid approach was examined by [12]. User profiles were created based on features of the movies they rated and these profiles were used to select users who are more similar to each other (user-based). Information from Wikipedia[1] was used by [10] to improve the similarity calculation between movies (item-based) by using text, movie category and links. Another way to implement a hybrid approach was done by [3]. They proposed a perceptron-like training algorithm where collaborative information was combined with content information (model-based). All these hybrid approaches led to better results than the individual, content-based or collaborative, approaches.

[1] http://www.wikipedia.org

This paper discusses user-to-user collaborative filtering approaches (mainly k-NN and majority voting) and a model-based approach (neural networks). The implementation of the neural networks will be quite similar to [11], but instead of just predicting a like or dislike, it is tried to predict the actual rating.

3 Data set and performance measures

This paper focuses on examining different methods for recommending movies. A MovieLens[2] data set from GroupLens Research from the university of Minnesota was used for testing the different methods. GroupLens collected this data from its users of MovieLens, a movie recommendation website[3]. For the results described in this paper the MovieLens 1M Data Set was used, which contains 1.000.209 ratings from 6.040 users for 3.706 movies.

Each rating consists of the user who gave the rating, the movie that was rated and the rating the user gave to that movie (all numeric values). Movies are rated on a 5-star scale (whole star ratings only). Stars are represented with the numbers 1 to 5, and a higher number (i.e. more stars) means a higher appreciation of the movie by the user. Two indications about how well different methods perform are used in this paper. The correctness is measured by how many ratings from the test cases match with the predicted rating:

$$\text{Correctness } (\%) = \frac{c}{n} \cdot 100 \text{ with } c = \sum_{i=1}^{n} \begin{cases} 1 & \text{if } p_i = a_i \\ 0 & \text{if } p_i \neq a_i \end{cases}$$

where p_i is the predicted rating (after it is rounded to the nearest integer to correspond with the possible ratings), a_i the actual rating and n the number of test cases. Predictions are always integers between 1 and 5 to match the possible ratings. The error is defined by:

$$\text{Error} = \frac{e}{n} \text{ with } e = \sum_{i=1}^{n} |p_i - a_i|$$

where p_i is the predicted rating, a_i the actual rating and n the number of test cases. The predictions are rounded to the nearest integer so that it reflects the number of stars the prediction differs from the actual rating.

4 Collaborative filtering based on averaging ratings and majority voting

The first two methods that will be examined are averaging ratings and majority voting, which are both quite self-explanatory. The first method averages the rating from a group of users and uses that value as the predicted rating, and the second method uses the most frequently occurring rating from a group of users for this.

4.1 Test set

To measure the performance of the aforementioned methods, some test cases were required. Around 5% (50.480) of all the ratings from the MovieLens 1M data set were extracted at random to create a separate test set, which is disjoint from the remaining data, the base set. There were 8 cases where the only rating for a movie was in this test set and these were removed because in those cases it is not possible to do any predictions, creating a test set containing 50.472 cases.

[2] http://www.grouplens.org/node/73/
[3] http://www.movielens.umn.edu

4.2 Nearest neighbors

According to [14], personalized ratings increase performance of the recommender system. The recommendations should be based on a subset of the entire database to achieve this, otherwise each user will receive the same recommendations. Which subset is used depends on the similarity a user has with other users. When a user seems to be very similar to another user in his rating behavior, he will be part of that subset. To determine the subset, the method of nearest neighbors is used. In this case, the nearest neighbor is the user who has a the most similar movie taste as the reference user. The shorthand notation for this method is called k-NN, where NN stands for nearest neighbor and k for the amount of neighbors and so the two approaches to recommend movies will be called k-NN-A (i.e. averaging ratings from k neighbors) and k-NN-MV (i.e. majority voting with k neighbors).

A distance metric is used to determine how similar a neighbor's movie taste is when compared to another user. The smaller the distance between two users, the more similar they are. There are many ways how the distance can be computed, here there is chosen to determine the distance by computing the city block distance between two users. This distance is calculated with the following formula:

$$d_1(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} |p_i - q_i|$$

where $\mathbf{p} = (p_1, p_2, .., p_n)$ and $\mathbf{q} = (q_1, q_2, .., q_n)$. $\mathbf{p}$ and $\mathbf{q}$ are users, stored as vectors containing their movie ratings. So for each test case, all users are compared with the user from the test case and for each movie their ratings are compared and their difference summed, which results in the distance between those two users.

However a neighbor is only useful if he has rated the movie the prediction is calculated for. So it is possible that there are users who have a smaller distance (i.e. are more similar) to the test user who can not be used as neighbors because they have not rated that particular title. The consequence is that not only every user has a different set of nearest neighbors, but that in many cases the k-NN set differs for the same user when a prediction is done for another movie.

First, there was some starting point needed to select further settings. There has been chosen to start with all users in the database and deal with subsets later. This will create predictions that do not depend on the test user, and this will therefor allow us to do a comparison to see if personalizing the predictions will increase performance. Because every available neighbor from the data set was used, these methods were called ∞-NN-A and ∞-NN-MV. The results that were gathered are in Table 2. It is clearly visible that ∞-NN-MV performs much better than ∞-NN-A. This is not really a surprise if all data is used, because it is likely that the ratings that are the most frequent in the base set will also be the most frequent in the test set. Intuitively, using the averaged rating instead of the the majority vote will therefore result in a lower amount of correct predictions.

	∞-NN-A	∞-NN-MV
Correctness (%)	30.6	40.6
Error	0.89	0.77

Table 2: Performance comparison of the ∞-NN-A and ∞-NN-MV

Of course it is possible that there are movies where the highest occurring rating is not unique. For the results presented in Table 2, no prediction was done in those cases. Only in around 1% arose this problem. Two solutions to overcome this were tested. It is possible to fall back on k-NN-A if there is no rating which has the majority vote, or pick one of

the ratings which has equal occurrences. Two possibilities have been examined in the last case: selecting the lowest of the two ratings with equal occurrences and selecting the highest one. The results are in Table 3. In almost all of these cases only two ratings had the same

	Correctness (%)	
	Overall	Problem cases
k-NN-A	40.9	26.4
Lowest rating.	40.9	30.0
Highest rating	40.9	32.9

Table 3: Performance comparison between different methods for cases where there is no majority vote

occurrence ratio, but there were some cases where even three ratings had equal occurrences. No separate testing has been done on those cases because they were very rare. In such cases the middle of the three ratings is chosen (which is the second rating).

Preliminary results suggest that it is better to select the higher rating over the lower one. But the amount of test cases might not be sufficient to conclude so. The three used strategies perform almost equally well on the entire test set. Due to the low occurrence of equal frequencies, the difference is only visible when looked at the problematic cases. The overall performance increased slightly, with 0.3%. The benefits of an other strategy in case of a draw is low, but selecting the highest rating when this occurs seems to be the best method.

4.3 Dealing with missing values

The approach of calculating the distance is straightforward as long as both users have rated the same movies. But there are many cases where only one or neither of them has rated a certain movie. It is possible to calculate the city block distance when both users have not rated a title, it will be just zero. But this might not be the best solution. The other case, in which only one them has rated a movie, is more problematic. Calculating the distance over a missing value is a bit strange. It also means that the distance between a missing value from one user and a low rating from another is smaller than between a missing value and a high rating, which is not logical because it is simple not known it that user will like or dislike the movie he did not rate. These cases could be ignored when calculating the distance, but some other solutions to treat these missing values are also examined.

By adding a small penalty (instead of calculating the city block distance) to cases where only one or neither of them has rated a movie, the predictions improved. To determine the height of the penalty, a random selection of 10% from the test set was selected to speed up the testing process. This smaller test set was compared with the large one, to see if it would perform about equally well and thereby determine if any conclusions based on the small set could be valid. The differences between the large and small set were marginal, see Table 4, so it did not seem to be a problem to use a smaller set. Before the penalty could be

	Large set (50472 cases)	Small set (4963)
Correctness (%)	40.9	40.3
Error	0.77	0.78

Table 4: Performance comparison between the large and small test set

determined, it was necessary to find a suitable value for k, the number of neighbor's that should be used. The number of neighbors was varied between 1 and 40 and the distance only calculated if both users rate the movie. Although the best results were obtained with 28

neighbors, the difference between 28 and 15 neighbors was found small enough (Table 5) to use just 15 neighbors, enabling much faster calculations. There are two possible penalties

	28-NN-MV	15-NN-MV
Correctness (%)	37.5	36.7
Error	0.84	0.87

Table 5: Performance comparison between 15-NN-MV and 28-NN-MV on the small test set without a penalty

that can be applied: a penalty for cases where both users did not rate a title and a penalty in cases that only one of them rated a title. To determine the penalties, both penalties were varied between 0 and 5 with increments of 0.5. The best results were obtained when both penalties have around the same value (see the diagonal in Table 6a) with the best results when both penalties are 1. These results were then used for further testing: The penalties were varied between 0.5 and 1.4 with increments of 0.1. Again it turned out that the highest percentages were achieved on the diagonal (Table 6b). The best results were achieved when both penalties are 0.9. it is likely that the penalty depends on the sparsity of the data. If the sparsity increases or decreases (by either adding more movies or more ratings), the penalty that is required for the best results might change. A different penalty might yield to better results if a different data set is used. Because k-NN-MV method already outperformed k-NN-A method by a very large margin, there has not been done separate testing for the penalty for k-NN-A and thus the same penalty was used for both methods.

	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
0.0	36.7	36.6	37.7	37.9	37.6	37.9	**38.1**	37.6	37.8	37.5
0.5	**39.0**	**39.6**	39.3	39.2	38.6	38.1	38.3	38.0	37.8	37.8
1.0	38.9	39.5	**44.1**	39.7	39.4	38.2	38.2	37.8	38.1	37.7
1.5	38.3	38.8	39.5	**42.2**	40.9	39.6	38.2	38.1	38.2	38.1
2.0	38.0	38.3	38.8	39.5	**41.8**	**41.5**	40.4	38.9	38.2	37.8
2.5	38.0	38.3	38.4	38.8	39.1	**41.5**	**41.7**	40.4	39.9	38.6
3.0	37.7	37.8	37.9	38.3	38.9	39.0	41.2	**41.5**	40.8	39.2
3.5	37.6	37.7	38.0	38.1	38.4	38.7	39.0	40.8	**41.3**	41.0
4.0	37.5	37.5	37.9	37.9	38.4	38.4	38.4	39.1	40.3	**41.0**
4.5	37.5	37.6	37.8	37.9	38.3	38.4	38.4	38.2	39.0	**40.3**

(a)

	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4
0.5	**39.6**	39.2	39.1	**39.6**	39.5	39.3	39.5	39.4	39.0	39.2
0.6	**40.4**	**41.6**	40.3	40.0	40.2	40.0	39.2	39.5	39.6	39.3
0.7	40.0	41.1	**43.1**	41.0	40.5	40.9	40.2	40.4	40.0	39.4
0.8	40.1	40.3	41.2	**43.3**	42.3	42.0	41.0	40.6	40.5	40.3
0.9	39.9	40.3	40.6	41.2	**44.2**	42.7	41.5	40.9	39.8	40.1
1.0	39.5	39.9	40.0	40.4	41.6	**44.1**	43.6	42.7	41.9	40.5
1.1	39.5	39.5	39.9	40.2	40.0	41.0	**44.0**	**43.8**	42.8	42.0
1.2	39.4	39.4	39.5	39.3	40.0	40.2	41.1	**43.8**	**43.9**	43.3
1.3	39.0	39.1	39.3	39.6	39.7	39.8	39.8	40.9	42.7	**44.0**
1.4	38.9	38.8	39.1	39.2	39.6	39.7	39.8	40.0	40.9	**42.4**

(b)

Table 6: Performance comparison between different penalty combinations (applied to 15-NN-MV). The highest percentages of each row and each column are bold. Columns correspond to the penalty for one missing value and rows to two missing values.

The use of a penalty for cases where one or both of the users did not rate a movie increased the performance substantially (Table 7). After it was decided how missing values should be treated for the distance calculation, the optimal number of neighbors was determined. For both the k-NN-A as the k-NN-MV the number of neighbors has been varied between 1 and 50 to determine the best amount when the penalty of 0.9 is applied. k-NN-A performs best with a very low number of neighbors, and the highest success rate is achieved by just using the rating from the nearest neighbor as the predicted rating. However, this resulted in a large average mistake. By selecting the average of the nearest two neighbors, the correctness stayed the same but the erro decreased. The 32-NN-MV performed the best, but one might argue that the difference between 32-NN-MV and, for example, 18-NN-MV is so small that it is better to select less users to decrease the computational time. Still, k-NN-MV performs better. Even when just a small number of users is selected to calculate the average, the ratings of neighbors are too widespread, which results in a prediction that does not correspond to the actual rating. The results mentioned above are in Table 8.

	Always city block	No penalty	Penalty of 0.9
Correctness (%)	38.7	37.8	44.0
Error	0.82	0.85	0.71

Table 7: Performance comparison between different methods that deal with missing values

	1-NN-A	2-NN-A	18-NN-MV	32-NN-MV
Correctness (%)	39.3	39.3	44.3	44.6
Error	0.80	0.76	0.70	0.70

Table 8: Performance comparison between different number of neighbors

4.4 Deviation in ratings

To find ways to increase the performance, it was sorted out how well the methods performed on predicting ratings for movies which have a high deviation in the received ratings. For some movies it is easier to do predictions because most people give it the same rating. However, there are also movies where the ratings are much more mixed.

To shine some light on the performance for movies with a high and low deviation, the standard deviation was calculated for every movie. After that, 20 percent of the test set was tested: the 10 percent of test cases with the highest standard deviation and the 10 percent with the lowest standard deviation. These sets were tested with the settings that yielded in the highest results, which means that the 2-NN-A and 32-NN-MV were used with a penalty of 0.9. There is a very large difference in how good the predictions are for movies with a low

		Highest STDEV	Lowest STDEV	Entire test set
2-NN-A	Correctness (%)	31.4	46.7	39.3
	Error	0.95	0.61	0.76
32-NN-MV	Correctness (%)	36.1	54.5	44.6
	Error	0.90	0.55	0.70

Table 9: Performance comparison between the normal test set and the sets with ratings with a low and high standard deviation

deviation and with a large deviation (Table 9). Even when only looked at a limited number

of neighbors, the variance in ratings is too large to do decent predictions. To see if it would be better to decrease to number of neighbors in such cases, again the number of neighbors was varied between 1 40 for the k-NN-MV. Further decreasing the number of neighbors, with the idea of eliminating part of the variance, did not help to increase the performance: the best performance was still at around 30 neighbors.

Two methods were tried to improve the success performance on movies with a high deviation in ratings. Instead of the previous used method to compute the neighbors, different methods were used to select one neighbor. The first method was to select the user who has the largest amount of equal ratings with the user where the prediction is for. The other method was to select the user with the highest amount of overlap in the movies they have seen. So instead of comparing the ratings, there was only checked if they had both rated the movie. The idea behind these strategies was that for a high deviation, just looking at similar ratings or a similar watch pattern might be sufficient to do a good prediction. The results, however, were very poor with correctness percentages at around 25%.

5 Collaborative filtering based on neural networks

A high improvement in performance, when compared to k-NN methods, was found by [11]. They used neural networks for recommending movies. They only predicted a like or dislike of movies and only used test cases in which the user highly liked or disliked a movie. Less clear opinions were ignored. Here we check if neural networks can also be used to boost performance when a prediction of a rating is required and where test cases are not ignored based on rating.

Neural networks are capable of finding patterns and learning complex relationships between its inputs and outputs. Neural networks are a subclass of the machine learning algorithms, which is a branch of artificial intelligence. There are many different neural networks. Here is chosen for the multilayer perceptron (MLP), a feed-forward (cycle-free) neural network. A MLP is a directed graph consisting of an input layer, hidden layer(s) and an output layer. Each layer consists of nodes, the neurons. These neurons have an activation function which determine its output value. Each layer is fully connected to the next one. By training the network on input examples the weights on those connections is adapted and by changing the weights, the output is influenced. In the training stage, the network learns from and adapts to the training data. It tries to match its output with the desired output value from the train set. The performance of the neural network is measured by how well it performs on the test set, which should be disjoint from the train set.

Just as described in [11], neural networks were created for a single user. Its inputs will be the ratings from the nearest neighbors for a certain movie. The goal of a neural network is be able to predict future ratings of the user after the network is trained, by having learned what kind of movies the user likes and dislikes.

5.1 Determining the settings

To create the neural network, the FANN[4]-library was used. This library provides many settings to optimize the neural network. To determine which settings to use for the neural network, different data sets than the one used for the k-NN-MV and k-NN-A were required because a neural network would only do recommendations for one particular user and therefor the data set for each network had to be user specific. Users who rated less than 40 titles were excluded, as well as movies with less than 100 ratings. The first choice was made because sufficient train and test cases were needed to be able to create a good model for a user. The second choice was made because if a movie has hardly any ratings, almost all inputs of the neural network would be zero while only real ratings, and not missing values,

[4]Fast Artificial Neural Network - http://leenissen.dk/fann/wp/

are wanted as inputs. Which will make it quite hard to find some good working settings because the inputs are pretty much useless.

To find the best settings for the neural network, a group of 30 users were selected at random from the reduced data set. A neural network was created for each user and for each user 20 test cases were extracted from that set. The train set is composed of all the other ratings of a user. Each of the 30 networks was trained and tested 20 times, creating a total of 600 neural networks. The performance of a network differs substantially after one completed training phase, so by training the network multiple times an average performance could be determined.

Each train and test case consists of 100 inputs and 5 outputs. The inputs are the ratings the 100 nearest neighbors gave to a certain movie. There has been chosen for a much larger amount of neighbors than used in the k-NN methods. This was done because the constraint was dropped that a neighbor must have rated the movie of the test case. It was dropped because the neural network needs the same neighbors as input for every input case. By keeping the constraint, the neighbors in the k-NN set will not be the same for different movies and then an input would be associated with different neighbors. The outputs of the network correspond to the rating the user gave to that movie, the first output corresponding to a rating of 1, the second to 2 and so forth. The output with the highest value was used as the prediction. Inputs were between 0 (i.e. a missing value) and 5 and outputs between 0 and 1.

Before tuning the settings of the neural network, the penalty that was used for calculating the distance between users was examined. This was done because the requirement was dropped that a neighbor rated the title where the prediction was for. The penalty was varied between 0.7 and 1.9. However, it was assumed that the penalty for both cases (see Section 4.3), should be the same, just as it was for the k-NN methods. The results of varying the penalty are in Table 10. After this, a different training method was tried: incremental training instead of batch training. Incremental training changes the weights of the neural network after each input as where batch training changes its weights after each time it cycles through all the train data. The incremental training was tried for a penalty of 1.4 and for a penalty of 1.9. Although the 1.9 penalty outperformed the 1.4 penalty on batch-training, the 1.4 penalty performed much better on incremental training. Therefore, for the rest of the experiments, a penalty of 1.4 was used.

Penalty	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	2.1
Correctness (%)	35.7	35.6	35.0	36.8	37.0	37.2	37.4	37.5	36.1	36.6	36.7	37.0	37.6	37.6	37.3
Error	0.96	0.95	0.95	0.93	0.93	0.93	0.92	0.92	0.95	0.94	0.94	0.91	0.92	0.91	0.92

Table 10: Performance comparison between different penalty values

The algorithm trains on the data as long as the mean square error (MSE) is above a certain value or the train set has been passed trough the neural networks a certain amount of times (1000). To prevent overtraining, some testing has been done on using a larger value for the MSE. For the previous tests a MSE of 0.001 was used, which is a very low error. Therefore, there has been experimented with a higher MSE, although it became quickly clear that it should be below 0.1. Otherwise the neural network reached the desired MSE when there still was enough room for further learning without overfitting to the train data. The difference in performance for different values for the MSE were small, but using a higher value than the previous value of 0.001 clearly improved the results (Table 11). Now the steepness of the activation function of the neurons in the hidden and output layer was changed. With a higher steepness, the activation function will go faster from its minimum to its maximum

MSE	0.0001	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.12	0.15
Correctness (%)	38.4	39.2	40.0	39.5	39.5	39.4	39.1	39.5	38.2	37.3
Error	0.88	0.87	0.84	0.85	0.86	0.86	0.86	0.87	0.88	0.88

Table 11: Performance comparison between different error levels

value. A lower steepness yielded to better results (Table 12). Also the learning rate was varied. The best results were achieved when the learning rate was low (Table 13). But with a low learning rate, the training of the network is slower and thus takes it more time to reach the desired MSE. The final parameter that was varied was the amount of hidden neurons. The number of neurons should be relatively small, because a network with less neurons will be faster to train. However, if it is too small, the predictive power of the network will also drop. The results in Table 14 suggest that it is better to have a large amount of neurons, although the performance does not really increase after 30 neurons. The difference is so small that it probably is not significant.

Steepness	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Correctness (%)	41.4	40.8	40.4	40.8	39.8	39.8	38.8	38.9	39.0	38.7
Error	0.80	0.81	0.83	0.83	0.85	0.85	0.87	0.87	0.87	0.87

Table 12: Performance comparison for different steepness values of the activation function

Learning rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Correctness (%)	42.7	42.3	41.9	41.8	41.2	41.8	41.6	41.7	41.3	41.8
Error	0.78	0.79	0.79	0.79	0.81	0.80	0.81	0.80	0.81	0.80

Table 13: Performance comparison of different learning rates

5.2 Performance

Because only a small test set was used, chances are that these settings only work well on the used test data. To draw any conclusions about the performance of neural networks a new test set was created. This time, 50 users were selected and the users with less than 60 ratings were removed, reducing the number of users to 36. The limit of 100 ratings for a movie was removed. For each user, again 20 test cases were extracted from all of their ratings. Because of the used settings, and in particular the lower steepness, the variance in the performance of each network was strongly reduced. Therefore, instead of 20 training and testing cycles per user, only 7 were done. After that, the network that performed best on the test data was selected. The neural network with the highest correctness was chosen, and if there were multiple networks with the same correctness, the one with the lowest error was picked. After this, the network was validated with the validation set, a set which also contained 20 ratings and that was disjoint from both the train and the test set. The usage of a separate validation sets made sure that a valid comparison between the neural networks and the k-NN methods could be made: an average performance over multiple networks per user is not very useful as a result because only one network per user is desired for practical use.

Although the results in Table 14 suggested that it would be better to use a large number of neurons, the validation was also done with a smaller number of hidden neurons. Instead of using 30 neurons, which seemed to be a good amount of neurons, also networks with only 12 hidden neurons were trained, tested and validated. It turned out that the results

Hidden neurons	5	10	15	20	25	30	35	40	45	50
Correctness (%)	41.7	42.0	42.3	43.1	43.0	43.6	43.4	43.7	43.2	43.4
Error	0.80	0.79	0.79	0.78	0.77	0.76	0.77	0.77	0.77	0.76

Table 14: Performance comparison for different amounts of hidden neurons

are quite equal (Table 15), but with less neurons the training stage is much faster and therefore preferred. The performance drop (Table 15) between the results on the test set and the validation set is logical. Which of the seven networks was selected for validation was determined by its performance on the test set. It is unlikely than, that it would perform even better on the validation set. Although we now have a neural network that performs

	30 hidden neurons		12 hidden neurons	
	test	validation	test	validation
Correctness (%)	49.3	44.2	49.4	44.6
Error	0.68	0.74	0.66	0.73

Table 15: Performance comparison between the best performing networks on the test and validation data

quite well, better performance should not be too hard to achieve. The settings were selected by a sequential process, first selecting the penalty and followed by the mean square error, steepness, learning rate and the number of hidden neurons. So not all combinations of settings were tried, and there are still many settings that could be varied. There has not been done any testing with changing the amount of inputs (i.e. neighbors), so changing that might also improve the performance.

6 Comparison of the different methods

To compare the k-NN methods with the neural networks, all the validation sets that were used for the neural networks were combined into one test set and then tested on the 2-NN-A and 32-NN-MV (Table 16). 32-NN-MV outperformed the average performance of the neural networks as well as 2-NN-A The performance from 32-NN-MV on this small test set is also higher than was earlier achieved. However, the test set is much smaller and therefore less representative than the previously used set. Also, only users who rated at least sixty movies are in the test set, which might make it easier to do predictions.

	2-NN-A	32-NN-MV	Neural networks (12 hidden neurons)
Correctness (%)	39.2	47.2	44.6
Error	0.73	0.67	0.73

Table 16: Performance comparison between different methods

7 Conclusions

Different methods for predicting ratings to provide recommendations were examined. k-NN-A does not seem to be a very suitable method for recommending movies but better results might be obtained by adding weights to neighbors, creating a weighted average. k-NN-MV performs much better in recommending movies and does not require a vast amount of neighbors for good performance on the MovieLens database. When k-NN-MV is used

to give a prediction for every item that is searched for by the user, its prediction will be often slightly off. But k-NN-MV will, due to the small error in its predictions, succeed in providing a number of highly recommended movies that were selected only for that user, based on his personal movie taste. Further research can be done on predictions for movies with a high deviation in its ratings because the examined methods did not improve results and there is a lot of performance to gain there.

The much higher results from neural networks when compared to k-NN that were published by [11] could not be reproduced. Although they only predicted a like or dislike and did not try to predict the actual rating a user would give to a movie, it was expected that also rating prediction should improve. However, our preliminary results, based on the MovieLens database, suggest otherwise. Further research in the usage of neural networks for recommender systems is warranted.

Although the neural networks were outperformed by the simpler k-NN-MV method, there is still much room for improvement and further testing. With some more research it might perform better than k-NN-MV. A lot of the parameters can be further finetuned by changing them with smaller increments or varying other settings provided by the FANN-library that were left at default or not used at all. Also, the number of neighbors that were used for the neural networks could be examined and even the selection of those neighbors could be changed by using a different method to compute the similarity between users. The number of neurons in the hidden layer requires some further research because at first the gathered results suggested that more neurons would give better performance, but later results suggested that the same performance could be achieved with far less neurons in the hidden layer. All neural networks that were created were multilayer perceptrons, but there are many different, more advanced, neural networks around which might lead to better results.

Neural networks are just one of many approaches in the larger field of machine learning algorithms. The performance of neural networks shows that machine learning algorithms can be used for recommender systems and there is still a lot of research possible to find good working machine learning algorithms for recommender systems.

References

[1] N. Barbieri, G. Costa, G. Manco, and R. Ortale. Modeling item selection and relevance for accurate recommendations: a bayesian approach. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 21–28. ACM, 2011.

[2] N. Barbieri, M. Guarascio, and G. Manco. A probabilistic hierarchical approach for pattern discovery in collaborative filtering data. In *Proc. SDM Conf*, 2011.

[3] J. Basilico and T. Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first international conference on Machine learning*, page 9. ACM, 2004.

[4] D. Billsus and M.J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 54, page 48, 1998.

[5] J.S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[6] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[7] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[8] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[9] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM, 2001.

[10] G. Katz, N. Ofek, B. Shapira, L. Rokach, and G. Shani. Using wikipedia to boost collaborative filtering techniques. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 285–288. ACM, 2011.

[11] M. Kim, E. Kim, and J. Ryu. A collaborative recommendation based on neural networks. In *Database Systems for Advanced Applications*, pages 425–430. Springer, 2004.

[12] P. Melville, R.J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 187–192. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[13] M.J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5):393–408, 1999.

[14] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[15] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.