Universiteit Leiden
Opleiding Informatica

Video Recommendation

A comparison between collaborative filtering algorithms

Name:	Kevin van der Wijden
Studentnr:	s1073192
Date:	10/06/2014
1st supervisor:	Dr. M.S. Lew
2nd supervisor:	Dr. E.M. Bakker

BACHELOR THESIS

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

Abstract

Video recommendation is used all over the world. Websites like Netflix and IMDb use this to recommend movies to their visitors. In video recommendation, collaborative filtering algorithms are used. Collaborative filtering algorithms find patterns in datasets. These patterns can be used to make predictions. There are different kinds of collaborative filtering algorithms. In this document we will look into the model-based and memory-based collaborative filtering algorithms. The minimum distance classifier it used as memory-base collaborative filtering algorithm. Classic neural networks and dynamic neural networks are used as model-based collaborative filtering algorithms. How well do these different algorithms perform compared to each other? And what is the difference between a classic and a dynamic neural network?

Contents

1 Introduction

Video recommendation is used a lot nowadays. Websites like Netflix and YouTube use video recommendation to be able to recommend movies, similar to the ones that the user already has seen. There are several methods to do these recommendations. Some examples are found in [3], [7], [11] and [16]. These kind of recommendations are not only done for movies, see [6] and [8] for examples This document is based on the Netflix method for video recommendation as found in [7], [11] and [16].

From 2006 till 2009 Netflix wanted to improve their recommendation system. To achieve this they started an annually competition, called the Netflix Grand Prize. This was an open competition. The team that created the best method, which has to be better than the Netflix method, would win $1.000.000. Several methods have lead to a better performance. Some of these methods are described in [7], [11] and [16].

In [16] they used a cluster with 34 nodes, each having 4GB memory. Those nodes were grouped under a head node with 8GB of memory. When reading this a question raised; how would normal computers perform on algorithms similar to the ones used for the Netflix Grand Prize? In this document some collaborative filtering algorithms are tested and compared using a computer with an Intel i7-4700MQ (2.4GHz) processor and 16GB working memory.

Collaborative filtering algorithms are the algorithms used in video recommendations. These algorithms use information patterns among multiple users to gain information. The Netflix Grand Prize is based on video recommendation, which leads to the fact that in [7], [11] and [16] collaborative filtering algorithms were used to form a better solution.

By filtering datasets corresponding items are used to find patterns or information [2]. Applied on video recommendation the collaborative filtering algorithms are used to make predictions. Similarities in movie ratings are used to get a predicted rating for another movie.

Collaborative filtering algorithms work as follows. First, users rate items following a scale. For this research a scale from 1 till 5 is used. Next, the algorithm matches one user with another user, which is having the most similar ratings. Finally, using these similar ratings, the algorithm finds the highest rating of a not yet rated movie [2], [15]. In this way predictions can be made based on all ratings in the dataset, and not only based on the users own ratings.

There are different kinds of collaborative filtering algorithms: memory-based, model-based and hybrid [2]. Memory-based algorithms directly use the given dataset to find similarities between users. Memory-based algorithms can use these similarities to make predictions. The model-based algorithms are first trained on a trainingset to find patterns. Those patterns are stored and will be used to process the real data. When the real data is processed, the predictions are made based on the found patterns. Hybrid algorithms combine the memory-based and the model-based algorithms. In this research only the first two algorithms are used.

The main question is how well these algorithms perform compared to each other. This is what is being researched in this document. To find an answer on the main question, several collaborative filtering algorithms are runned using the MovieLens database [12]. While running the algorithms, the root mean squared error (RMSE) will be calculated. The root mean squared error gives the mean error for a user over all movies. The RMSE is also used in [16], where similar comparisons are made. This error is calculated using the following formula:

$$\sqrt{\frac{\sum_{t=1}^{n}(X_t - Y_t)^2}{n}} \tag{1}$$

Where n is the amount of movies in the dataset, X_t is the predicted value and Y_t is the real value.
The calculation of the global RMSE, which is the sum of the RMSE divided by the amount of users, is shown in the following formula:

$$\frac{\sum_{x=1}^{m} \sqrt{\frac{\sum_{t=1}^{n}(X_x t - Y_x t)^2}{n}}}{m} \tag{2}$$

Where m is the amount of users in the dataset, n is the amount of movies in the dataset, $X_x t$ is the predicted value for movie t for user x and $Y_x t$ is the real value for movie t for user x.

This document is organized as follows: in Section 2 the dataset is described, followed by some information about notations in this paper. The different algorithms are described in Section 3 and 4. Following, the outcome of running the algorithms using the dataset is in Section 5. A final conclusion is made in Section 6.

2 Dataset & Notation

The dataset used for this research is created from the MovieLens database. The MovieLens database is build by the GroupLens group. This database contains users which have rated movies. As found in the MovieLens documentation [12], there are three different sizes of databases available. These databases contain a different amount of users and movies as listed in Table 1. *Note: the numbers in Table 1 differ from the numbers in the MovieLens documentation. This is because rounded numbers are used in the Movielens documentation.*

Table 1: Amount of users and movies in the MovieLens databases

Database name	Amount of users	Amount of movies
MovieLens 100k	934	1682
MovieLens 1M	6040	3952
Movielens 10M	71567	10681

To be able to get reliable predictions, the difference between rated and unrated movies should be as small as possible. This difference should be as small as possibile, because the more ratings are given the preciser the predictions will be. After doing some test the 1M database appeared to give the best results. Even though the 1M database gives the best results when used in total, experiments have shown that the creation of a subset is needed. The results when using the total 1M database, on the collaborative filtering algorithms (for the algorithms, see Chapter 3 and 4) are worse than with random guessing. Random guessing will lead to an average error of 2.5 for each prediction, while running the algorithms on the whole 1M database will result in errors ≥ 3. To reduce this error, the difference between rated and unrated movies has to become smaller. Therefore a subset is created where the users have rated $\geq 25\%$ of the movies. For the rest of this paper this subset will be referred to as the dataset.

For the model-based collaborative filtering algorithm a training set and test set are needed. Those sets are created as follows; the training set is created by picking 75% of the dataset, and the test set is the other 25% of the dataset. For each movie a training set and test set is created. Those created sets will be constant for all experiments, and will be referred to as the training set and test set.

To improve the readability of this document, from now on some abbreviations are made:

- CF for collaborative filtering
- MDC for minimum distance classifier
- NN for neural network, NNs for neural networks
- RMSE for root mean squared error

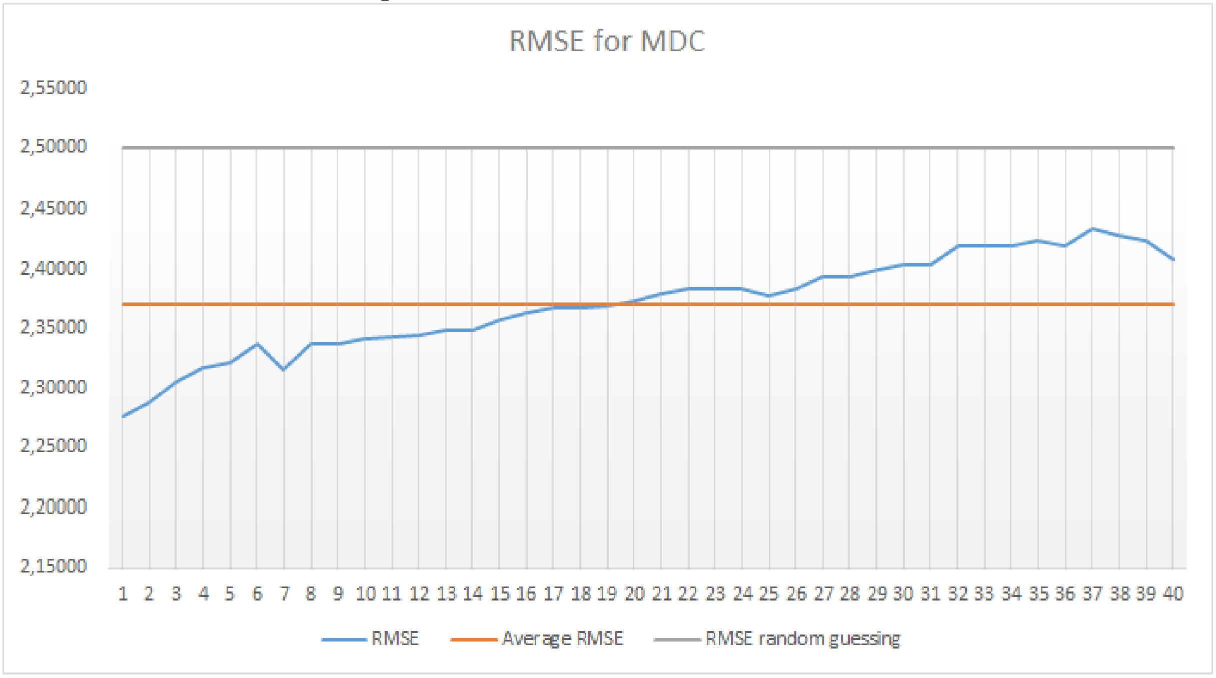
3 Memory-based Collaborative Filtering Algorithm

A well known memory-based CF algorithm is the MDC. The MDC finds for user A the nearest user B. The nearest user B is defined as the user that has the smallest difference in ratings compared to user A. The specific MDC used here is called k-NN, which is an abbreviation for k-NearestNeighbour. This classifier finds the 1 till k nearest neighbours for a user. In Figure 1 is shown what happens to the RMSE when variating k. Following the MDC in pseudocode:

Algorithm 1 The MDC in pseudocode

```
setCheck                                   ▷ To set the checked status of all users to 0
for $i = 1$;$i \leq k$;$i++$ do
    $best = \#Movies * 5$                  ▷ Maximum possible distance
    for $j = 1$;$j \leq \#Users$;$j++$ do
        $current = 0$                      ▷ Variable to save the current distance
        for $x = 1$;$x \leq \#Movies$;$x++$ do
            if $ratings[x] \leq ratings[i][x]$ then ▷ Where ratings[] are the ratings given by
a user and rating[$x$][] are the ratings given by a user $x$
                $current+ = ratings[i][x] - ratings[x]$
            else
                if $ratings[x] \geq ratings[i][x]$ then
                    $current+ = ratings[x] - ratings[i][x]$
                end if
            end if
        end for
        if $current \leq best \&\& check[j] \neq 1$ then
            $best = current$
            $latest = j$
        end if
    end for
    $check[latest] = 1$                    ▷ To exclude the last found best neighbour from the dataset
end for
```

Figure 1: MDC with different k-value.



As shown in Figure 1 the RMSE will globally increase when k is increased. This is logically derived from the fact that the higher the k is, the further away the neighbour is. When neighbours are further away, the larger distance will lead to a higher RMSE. This is because bigger differences between ratings lead to a bigger distance between users. The RMSE is only calculated by using already known ratings. So if an user has not rated a movie, this movie will not be used for the calculation of the RMSE.

The average RMSE for the MDC is $2,37092$. To be able to do a good comparison the result of the MDC need to be better than the result of random guessing. In Figure 1 is shown that this is accomplished. There is no need to compare CF algorithms that perform worse than random guessing. Because in those cases, random guessing leads to a better result and is easier to implement.

4 Model-based Collaborative Filtering algorithms

As described in [14] a NN is a network of neurons, this is a small model of the human brain. Within the representation of these networks a neuron is a node, and a connection between neurons is an edge. Each neuron has an input, an activation function and an output. The input is calculated by the input function described in formula 3. Using the input the neuron calculate an output with the activation function. When an activation function has a hard threshold, the NN is called a perceptron. When the activation function has a soft threshold or logistic function, the NN is called a sigmoid perceptron.

The edges between neurons have a weight. These weights are used to calculate the input of the receiving neuron, as shown in Algorithm 3. During the training of the NN the weights will change based on an error value. This error value is a value based on the difference between the predicted value an the value that should be given. There are two error values; the first to update the weights from input to hidden neurons, and the second is to update the weights from the hidden neurons to the output neuron(s). There are two algorithms for updating the weights, repectively Algorithm 4 and 5. The first one is for updating the weights from the input neurons to the hidden neurons. The second one is for updating the weights from the hidden neurons to the output neuron(s) [14]. The updating of the weights is done during the training phase of the NN. In the training phase the NN updates the weights in such manner that the output is as close as possible to the actual rating. The actual rating is the rating given by a user for a movie. Training is done with data of which the input and output is known. In this way it is possible to calculate an error. This error is the difference between predicted value and the actual value. This error is used to update the weights as shown in Algrotithms 4 and 5. Once trained the NNs are ready to make predictions with new datasets. These datasets only contain input information. So an output has to be calculated using this input.

$$\sum_{i=1}^{n} W_{i,j} * a_i \tag{3}$$

Where W_i, j is the weight from neuron i to neuron j and a_i is the activation value of neuron i.

$$W_{i,j} = W_{i,j} + \alpha * a_i * \triangle_j \tag{4}$$

Where $W_{i,j}$ is the weight from input neuron i to output neuron k, α is the step size (learning speed),a_i is the input of neuron i and $\triangle_j$ is the error value as described before.

$$W_{j,k} = W_{j,k} + \alpha * a_j * \triangle_k \tag{5}$$

Where $W_{j,k}$ is the weight from hidden neuron j to output neuron k, α is the step size (learning speed), a_j is the input of neuron j and $\triangle_k$ is the error value as described before.

NNs can have several layers: an input layer, possibly hidden layers and an output layer. NNs with only an input and output layer can solve the basic boolean functions AND, OR and NOT. More difficult problems have to be solved by NNs with a hidden layer[14].

In this research a NN is build for every movie. So there is a NN for each individual movie of the dataset. This means that $\#movies$ NNs have to be trained. It is done this way because creating a NN for each individual movie make the NNs less complex, than when a large NN is created for all movies. Instead of creating a NN with $\#movies$ outputs, $\#movies$ NNs are created with only one output. The training of all these NNs takes some time as described in Subsection 4.1.

Because NNs are already widely researched and used, serveral libraries can be found to directly implement NNs. A library that is suitable for the problem of this research is the FANN library. FANN is an abbreviation for Fast Artificial Neural Network. The FANN library is well documented and has a lot of different possibilities for creating NNs. The FANN library makes it possible to create a normal NN and a dynamic NN. Those NNs will be used for this research and are described in the following Subsections.

4.1 Normal Neural Network

A normal NN is a NN which is build like defined in the introduction of Section 4. These kind of NNs consist of an input layer, a hidden layer with a certain amount of neurons and an output layer containing one neuron. This is called a normal NN because the amount of input neurons, hiddens neurons and output neurons are known. For this particular case the following configuration is used: $\#inputneurons = \#movies-1$, $\#hiddenlayers = 1$, $\#hiddenneurons = 3$,

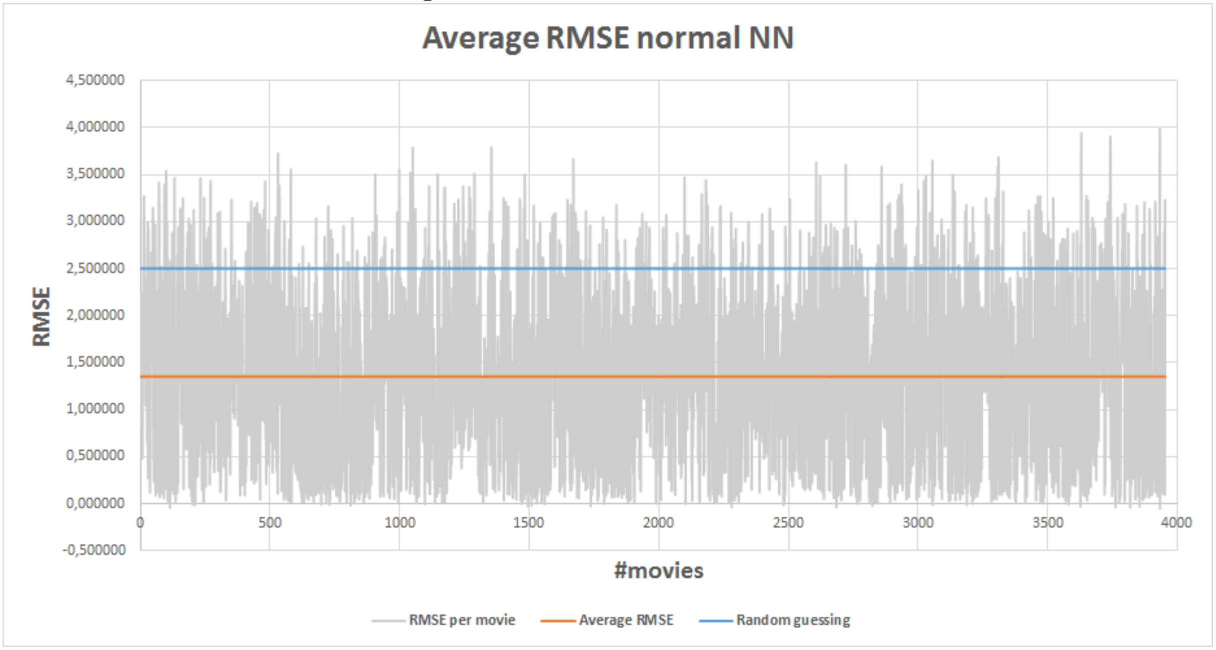
$\#outputneurons = 1$ and $\#epochs = 500000$. The amount of input neurons is set like this because training is only needed for al ratings of an user, except for the rating of the movie which is going to be predicted. As defined in [4] an epoch is one pass trough the training set. A training is allowed to do a maximum amount of epochs. This maximum is set to $\#epochs$. The NN will not always reach this maximum because when the desired error, here set to 0.0001, is reached the training will automatically stop.

The training of a normal NN takes a lot of time. In [4] they list two major problems why this training takes so long. The first problem, called '*the step sized problem*'[4], explains that a normal NN takes small steps when optimizing the weights. In each iteration small changes are made to the weights to reach a local, in most situation also the global, minimum. These small steps are the cause of the long duration of the training. When fast training is needed, the step size has to be as big as possible. With these big steps, the minimum is reached much faster. If the step size is too big there is a chance that the minimum will never be reached and that the output of the NN will not be reliable.

Second is "*the moving target problem*" [4]. Because there is no global co-operation between the neurons in a NN, the neurons are trying to solve the problem themselves. This is a hard task, because they only see their inputs and the error signal that is propagated back from the output [4]. The environment of the neurons is constantly changing. When neurons think they have reached their best solution, the environment changes and their best solution results into a worse output. This way the training takes very long, untill all neurons are trained such that a local minimum is reached. There are some ways to overcome or reduce these two problems, which is adapted to the Dynamic NNs described in the following subsection.

Nevertheless once the networks are trained, the generation of output is fast. After the training is done, the created neural networks will be tested with the test set. From this testing the RMSE can be calculated. The RMSE is calculated only with known values, this means that if there are unrated movies within the test set that the NN cannot calculate a reliable prediction. The FANN library has implemented this by ouputting a value representing that the NN is predicting for an instance of the data which has no real value. The result of the testing (after training) of the normal neural network is shown in the following Figure 2

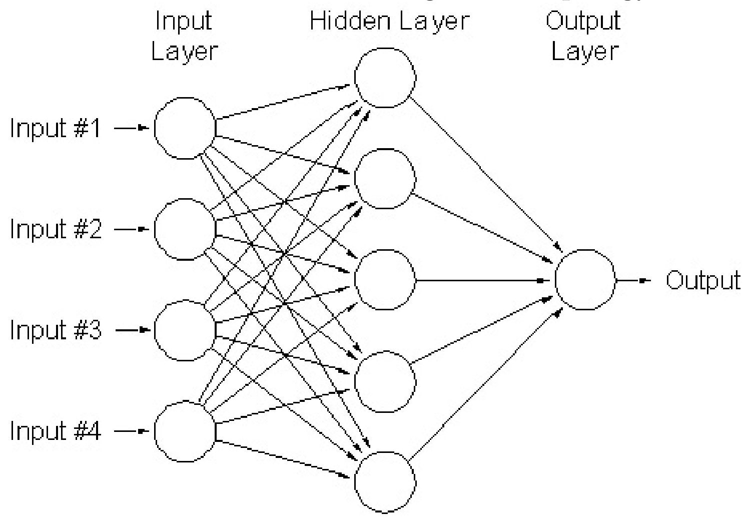
Figure 2: RMSE for normal NN



Note: there are some negative RMSE values as a result from insufficient data to train the NN properly. These values are not used for the calculation of the average RMSE.

The next Figure 3 is an example of the topology of a normal NN with four input neurons, one hidden layer with five neurons and one output neuron.

Figure 3: Topology normal NN



4.2 Dynamic Neural Network

Like described in the previous subsection, the normal NN comes with two major problems which extends the execution time for the training of the normal NN. The dynamic neural network reduce these two problems and this results in a much smaller execution time for the training. To achieve this, the dynamic NN uses Cascade-Correlation [4].

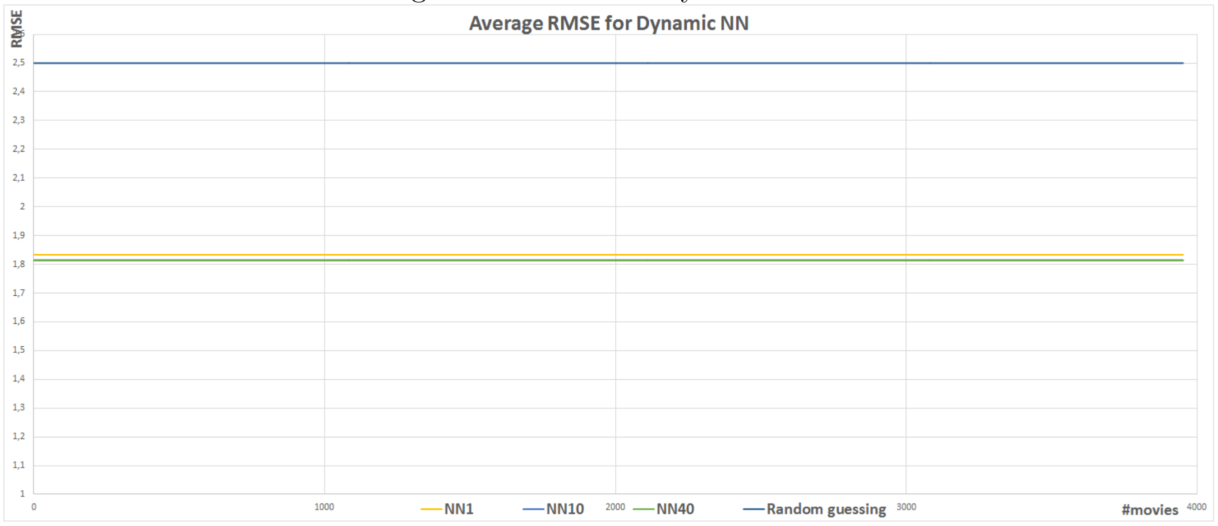
According to [4], Cascade-Correlation combines two ideas. The first idea is that hidden neurons will be added to the network one at a time. Once added they will not be changed.

Second is a different learning algorithm which works as follows. It creates and installs the new hidden units. The correlation between the error signal and the output of the newly installed hidden unit has to be as high as possible (maximized) [4]. The dynamic NN will be created as follows. To start there are only input and output neurons. Then these input and output neurons are trained like a normal neural network using RPROP learning algorithm as described in [13]. Once a minimal error is reached and no further training is possible, there needs to be decided if the neural network works good enough or that the remaining error is still too big. If the error is small enough, the NN is accepted. When the error is still too big, hidden units will be added one by one to reduce this error.

The adding of the hidden units will be done as follows [4]. First a candidate unit will be created. This candidate unit will get trained using a few instances of the training set, so the input and output weights of this candidate unit are set. The candidate unit is linked to all other units in the NN, which makes it more of a unity than all individual neurons in a network. There is a good co-operation along the whole NN, which results in the fact that much lesser training of the neurons is needed. Due to this the time needed to train a dynamic NN compared to normal NN is clearly lesser.

Because the NN is created dynamically, we have chosen to vary the amount of times the *fann_cascadetrain_on_data* function is called, to see if the result changes. The result is expected to change because there will always be a remaining error which can be reduced. The improvement is made by adding new hidden units. The result of this experiment is shown in the following Figure 4.

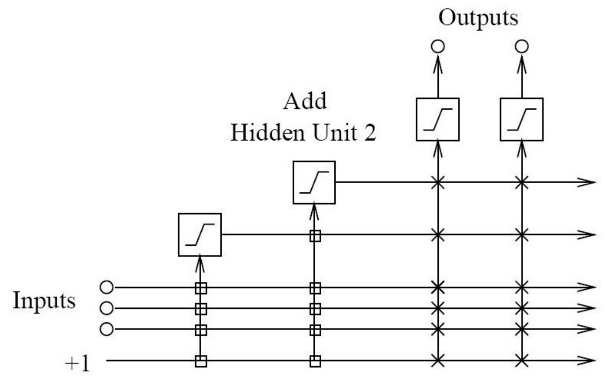
Figure 4: RMSE for dynamic NN



Where NNx are the NNs that are trained x times. Because there is a very small difference between the values of NN10 and NN40, respectively 1.814513 and 1.813125, only the line of NN40 is visible in Figure 4. The performance of the dynamic NN is better than random guessing like the previous CF algorithms. This makes it suitable for the comparisons done in the following chapter.

Because a dynamic NN is created on the run, the topology of the network will be different from the topology of a normal NN like shown in Figure 3. Following is an example of a topology of a dynamic neural network with three input neurons, two hidden neurons (which is also the amount of hidden layers) and two output neurons.

Figure 5: Topology dynamic NN



Where the vertical lines sum all incoming activation. Boxed connections are frozen, X connections are trained repeatedly. Figure 5 is taken from [4]. The vertical line having the +1 in front is a so called bias node. Bias nodes are used to store a certain threshold. This threshold is saved in these bias nodes such that during the training of the network, the adaptation of the weights is independent of the threshold. Bias nodes have the same definition for normal NNs. This topology is clearly different than the one in Figure 3. This is the result of the method used for creating a dynamic NN. Each hidden neuron also has connections with the other hidden neurons. The hidden neurons in a normal NN only have a connection with their input and output neurons and not with other neurons in the NN.

5 Comparison

In this section a comparison between the different CF algorithms is made.

For this research the RMSE is calculated for each individual CF algorithm. No combinations of CF algorithms are made, which results in a higher RMSE. Nevertheless it is interesting to see how the different CF algorithms perform on a normal computer.

CF algorithms are already widely researched, therefore there are already comparisons made, some examples are found in [5].

5.1 Normal NN & Dynamic NN

As shown in Table 2 the normal NN performs better than the dynamic NN. This difference in performance is due to the difference in training. The normal NN is trained intensively on the training set. The training of the normal NNs takes significantly longer than the training of the dynamic NNs, respectively serveral days and serveral hours. This difference in time is also caused by different ways of determine when to stop the training. Like described in Section 4 the neurons in a dynamic NN are added with the weights set. Those weights are not changed during further evolving of the networks. Opposite to this are the normal NNs where all neurons are already set. The training is based on the updating of all the weights beween the neurons. Therefore different ways of determine the conditions to stop the training are needed. For the normal NN the stop conditions are as follows: or the maximum amount of epochs is reached (500000) or, the minimum error is reached (0.001). For the dynamic NN a special function is used to determine if the training has to stop. Such functions are included in the FANN library, the specific function used is called *FANN_STOPFUNC_BIT*. This function stops the training when a bit error level is exceeded. A bit error is defined as follows; when an output neuron differs more then a certain error there is one bit error. The bit error is set to 0.9.

Table 2: Average RMSE

Algorithm	Average RMSE
Normal NN	1.352817
Dynamic NN1	1.833228
Dynamic NN10	1.814513
Dynamic NN40	1.813125
Random guessing	2.5

5.2 Neural networks & MDC

In this subsection both NNs are compared to the MDC. In Section 3 and 4 two different kinds of CF algorithms are described. In those sections is described that the two kinds of CF algorithms use different ways of computing a prediction. MDC needs all data, or a subset with all k closest neighbours, to be able to compute a prediction. On the other hand, once trained the NNs only need the actual data of one user to be able to compute a prediction. The training of the NNs take some time, but once trained the results will be generated significantly faster than with MDC. Due to this NNs are more logically to get implemented in a video recommendation application than a MDC. The results can be calculated on the run with the NNs, and the MDC takes too long to be runned in runtime. Nevertheless the comparison made here is not based on execution time, but on the actual results. This is because we want to know how well these algorithms perform to eachother, when generating results as good as possible.

As shown in Table 3, the NNs perform better than the MDC. This is as expected; the NNs get an intensive training before results are calculated, while the MDC directly runs on a subset of the data containing the ratings of k users. As shown in Section 3, it makes no sense to make k really big because the higher k will be, the higher the RMSE will get. Due to this the subset used for MDC will be smaller than the training set used for the NNs. This results in a more precise prediction by the NNs.

Table 3: Average RMSE

Algorithm	Average RMSE
Normal NN	1.352817
Dynamic NN1	1.833228
Dynamic NN10	1.814513
Dynamic NN40	1.813125
MDC	2.37092
Random guessing	2.5

6 Conclusion

As shown in the previous chapter, NNs perform better than the MDC due to a different handling of the data. NNs use all the dataset while MDC only uses a subset of the dataset. Also, NNs get trained on the dataset, after which it can predict, while the MDC immediately uses the data to make a prediction. These things lead to a better performance of the NNs.

There is also a difference between the two NNs, normal and dynamic, as discussed in Chapter 4. The two NNs handle their training in a different way. They differ in the stop criterion of the training and this leads to different results, see Chapter 5.

The main contribution of this work was the comparison between the MDC, a normal NN and a dynamic NN on a normal computer. Comparisons have been made before, like seen in [5]. Those comparisons are usually run on clusters, like found in [16]. To give an idea of the difference between the results of normal computers and clusters we look into the methods of [7], [11] and [16]. Here are different CF algorithms combined in a specific order and executed one after another to give a prediction. The calculation is done on a dataset which contains 1% of all possible ratings. This compared to the at least 25% in the dataset used for the experiments in this work. The less ratings a dataset has, the harder it is to make predictions. Eventhough it is harder to make predictions on that dataset, the combined algorithms will reach a RMSE of 0.8563 or lower, as described in [16].

7 Future work

For the future it is interesting to combine the different algorithms to get better results. It would be nice to do this combining of algorithms on a normal computer, liked used in the experiments done is this document. In this way it is possible to compare the results created with dedicated computers, like used in [16], with the results created on a normal computer.

In [7], [11] and [16] is already described that the combining of algorithms will improve the predictions. The order in which the algorithms are combined is important. An algorithm may perform better on the outcome of algorithm A, than on the outcome of algorithm B. It would be interesting to test different orders and different methods of combining.

It would be a challenge to produce a CF algorithm library which is dedicated for usage on normal computers. The used CF algorithms in this work, were normal implementations without optimizations. When adjusted for normal computers the performance could be improved, or the same performance can be reached in less time. This is to improve the efficiency of the algorithms.

According to [9], Netflix uses 76897 unique labels for movies. In the MovieLens database, the 10M sized one, there are labels available. These labels are given by the users [12]. The movies can be grouped using those labels. It would be interesting to see if the outcome of the CF algorithms will improve if only the ratings for movies with the same label are used, instead of the ratings of all movies.

References

[1] Achal Augustine and Manas Pathak. User rating prediction for movies. *Technical report, University of Texas at Austin*, 2008.

[2] John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[3] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The YouTube Video Recommendation System. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, New York, NY, USA, 2010. ACM.

[4] Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Carnegie Mellon University, Computer Science Department Technical Paper no. 1938*, 1989.

[5] Zan Huang, Daniel Zeng, and Hsinchun Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22(5):68–78, 2007.

[6] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: Applying Collaborative Filtering to Usenet News. *Commun. ACM*, 40(3):77–87, March 1997.

[7] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[8] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, Jan 2003.

[9] Alexis Madrigal. How Netflix Reverse Engineered Hollywood. *The Atlantic, January*, 12, 2014.

[10] Steffen Nissen and others (open source). FANN. n.d. `http://http://leenissen.dk/fann/wp/authors/`.

[11] Martin Piotte and Martin Chabbert. The pragmatic theory solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[12] GroupLens Research. MovieLens. n.d. `http://grouplens.org/datasets/movielens/`.

[13] Martin Riedmiller and Heinrich Braun. RPROP - A Fast Adaptive Learning Algorithm, 1992.

[14] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Englewood Cliffs, 1995.

[15] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[16] Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, 2009.