Universiteit Leiden
Opleiding Informatica

Combined Neural Networks

for Movie Recommendation

Name: Ruben Buitelaar

Date: 24/08/2015

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

Combined Neural Networks for Movie Recommendation

Ruben Buitelaar

Liacs, Leiden University,
Niels Bohrweg 1, Leiden, The Netherlands

Abstract. Movie Recommendation is something millions of people make use of everyday. Predictions are made by various sophisticated data mining and artificial intelligence techniques. In this thesis we propose an original method for predicting movie ratings. This method is based on combining neural networks. Basic neural networks predictors are trained individually on training data, and their results are used as input for other neural networks. This combination could provide a better rating than the individual network. We look at the setup of the basic neural network, together with the design decisions faced during the building of this network. We compared the predictions made by this method to a baseline predictor. The baseline we use is k-Nearest Neighbors, a data mining algorithm. The original approach did not show improvement compared to the k-Nearest Neighbor algorithm. The basic neural network performed slightly worse than using the average rating of the movie as a prediction.

Table of Contents

1 Introduction

With the advancement of the internet, it has become a reality that we have a lot of different choices at our disposal. As the number of options grows, it becomes harder to choose. This is especially true for online video distributors as Netflix and YouTube, who have thousands of hours of video available for the user.

In recent years recommender systems have found numerous ways into our lives. Recommender systems are software systems which recommend a subset of items to the user, based on previously exhibited behavior. The user now has an easier time making a choice, because the system has identified the taste of the user. To make a subset of items, the system predicts how much the user will like the item. For example a movie recommender system will predict the rating a user would hypothetically give to a movie. In this research we will be making our own original recommender system, which attempts to correctly predict the ratings a user gives to movies.

There are two primary techniques used in recommender systems. Content-based filtering is using information about the similarity of items to make predictions. If you have liked action movies in the past, the system will recommend you other action movies. You are matching items to items. Content-based filtering can work well when you do not have a lot of data generated by the users, because you only work of the information (meta data) of items.

Collaborative filtering is the concept of using data generated by other users to make a prediction for a specific user. You are matching users to users. Many different data mining techniques can be used to achieve this. Collaborative filtering works well when you have sufficient data generated by the users. This also requires that the user you are making a prediction for, has generated enough data to compare with other users.

A hybrid approach involves combining these two techniques to make a prediction. This combination usually yields the best results in real-world applications, because both techniques have different advantages and can supplement each others weak points.

In this thesis, we are going to compare an original approach to a frequently used and easy to implement data mining algorithm, k-Nearest Neighbors. The original approach is based on a network of neural networks. We are going to elaborate on the design decisions faced during the construction of the neural network.

2 Relevant Work

There has been a lot of research into recommender systems. Recommender systems are essentially predicting human behavior, and this is something that has great commercial value. Companies like Amazon[1] and YouTube[2] do a lot of research on recommender systems. A lot of different approaches for recommender systems have been researched[3, 4]. Because of their predicting power through learning, neural networks have been used since the beginning of recommender

systems[5]. In an effort to improve their own recommender, Netflix promised a 1 million dollar prize[6] for the team that could come up with a system that would improve Netflix' recommender by 10 percent. This reignited a search for better working models[7]. After almost three years, the 10 percent increase was realized. This shows how much effort was needed to further increase the accuracy of the predictions. The winning team was an ensemble of three teams, banding together at the final moments to ensure victory. It consisted of Team BellKor[8], Team BigChaos[9] and Team Pragmatic Theory[10]. In annual progress reports the participating teams explained their intermediate results. All teams achieve good results by blending the results of different predictors. Accurately blending the results is explained in this[11] paper by the same people from Team BigChaos. Neural networks are also used in predicting the ratings, a form of neural networks called Restricted Boltzmann Machines[12] have been used with great success in collaborative filtering. Our approach is based on a simple form of neural network, the multilayer perceptron with back-propagation[13]. The goal is to chain these simple networks, combining their knowledge to improve results. Chaining neural networks to improve classification has been done[14, 15], but not in the same way we propose.

3 Data set

The data is provided by the GroupLens Research Group[16], a research group at the University of Minnesota. Different data sets are made available for research. For this research the MovieLens 1M data set[17] is used. The set consists of 1,000,209 ratings of 3 elements; a user, a movie, and a rating value from 1 to 5 (whole numbers only). The 1,000,209 ratings are made by 6040 users on approximately 3900 movies.

3.1 Characteristics

A quick look shows that 1,000,209 ratings, 3900 movies, and 6040 users results in an average of 256 ratings/movie or 166 ratings/user. For our intended use, we would like to make the data set denser. Movies with a low amount of ratings are weakly connected and cannot be trained properly. Reducing the amount of movies would also greatly reduce the computational costs for training the neural networks. We chose to only use the movies with more than 200 ratings. This allowed us to train the networks faster, but is still a large part of the original set of ratings.

Rating	1	2	3	4	5
Frequency (%)	5.6161	10.7534	26.1156	34.8883	22.6266

Table 1. Frequency of different ratings

n	1+	50+	100+	200+	300+	400+	500+
Movies	3592	2499	2006	1419	1052	796	616
Ratings	999886	976884	940727	854146	763706	674443	593583

Table 2. Movies with more than n ratings.

3.2 Train & Test Set

After we have reduced the set to 1419 movies with each more than 200 ratings we split the data set into a training set and a test set. The training set consists of the ratings the recommender systems knows the outcome of, these are the known ratings. The test set consists of ratings the system has no knowledge of. For these ratings we can compare the prediction made by the system to the outcome that is known to us but not to the predictor. The data set is split 80% to 20% between training set and test set, respectively. This split is done beforehand and kept, so the Neural Networks can train on the training set and save their configuration, without rebuilding the next time.

4 Measuring the Accuracy

The performance of different recommender systems has to be compared. Choosing performance criteria will be always be subjective. We could favor a predictor which will be spot on most of the time but will be off by a large margin in some cases. Or we could favor a predictor which will be slightly off all the time. Even the stability[18] of the ratings could be used as a criteria. We introduce three methods of measuring the accuracy.

$$e_i = |target_i - predicted_i|$$

Mean Absolute Error (MEA)
This error is the average of the absolute errors, the absolute error being the difference between the outcome and the prediction.

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Root Mean Squared Error (RMSE)
This error is the root of the Mean Squared Error, which is the average of the squared errors.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (e_i)^2}{n}}$$

Correctness
This error is the percentage of cases correctly predicted. This can only be done if the predictor outputs an integer.

It is important to note that different methods produce different results. The RSME relatively favors smaller differences (<1) and punishes bigger differences (>1) because of the squaring of the error. A predictor which produces a high correctness does not necessarily produce a low RSME, because one big error will be punished heavily. We will use all these predictors, with the most important being the RMSE. This performance criteria was also used for the Netflix Prize.

5 *k*-Nearest Neighbors

The k-Nearest Neighbors algorithm is the predictor we use to create a baseline for comparison to our original approach. The algorithm works by comparing a user to all the other users and calculating the distance between them. The k users that are the closest to the user and have rated the movie you are predicting for are the k nearest neighbors. The average rating for the movie from the neighbors is the resulting prediction. The distance is between users p and q is defined by calculating the difference in ratings both users have rated:

$$d(u_1, u_2) = \frac{\sum_{v=1}^{n} |r(u_1, v) - r(u_2, v)|}{n}$$

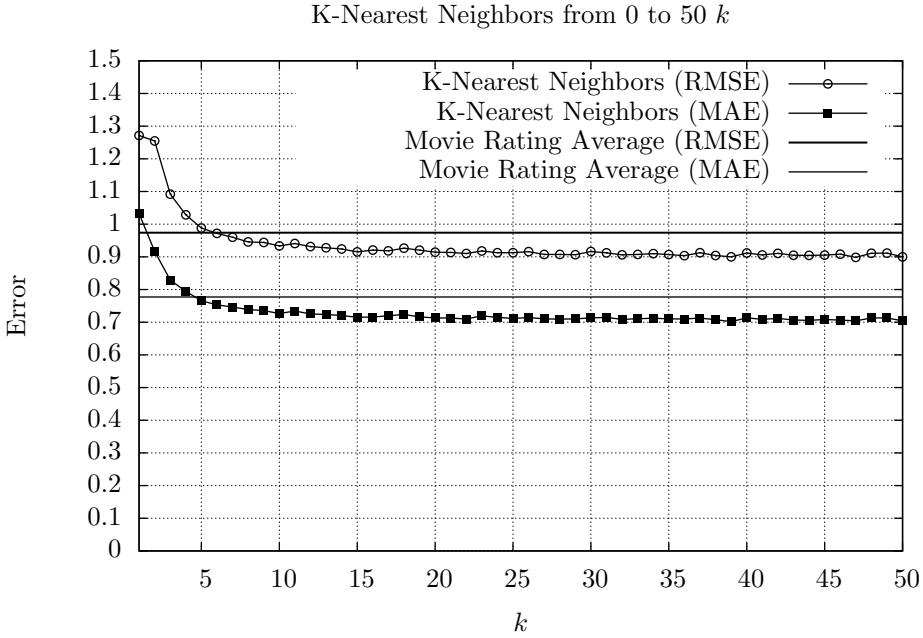
$r(u, v)$ is the rating of user u on movie v
n is the set of movies both users have rated$v$

Where n is the number of movies both users have rated. This results in every user having having a distance to another user. The k-nearest are the k users having the shortest distance to the user. The size of k influences the result. A low k gives the users that are the closest to the user, but a small set is also more prone to fluctuations. If k were infinite, the result would be the same as the average rating of a movie. The rating for user u on movie v predicted by k-NN is defined by:

$$r(u, v) = \frac{\sum_{i=1}^{k} r(n_i, v)}{k}$$

$r(u, v)$ is the rating of user u on movie v
n_i is the i-th neighbor that has rated movie v

We have to optimize the prediction by finding the right value for k. Here is a plot of the results:



We can see in this plot that for very low values of k the predictions are sub-optimal. The average of the few neighbors is not consistent enough. Around $k = 5$ the predictions get better than the average movie rating. The RMSE continues to drop and stalls around $k = 20$. This value is still easy to compute so we will compare our approach to 20-Nearest Neighbors.

6 Neural Network

In machine learning, neural networks (NN's or nets) have been used to solve problems that are hard to solve using rule-based programming. NN's learn by repeating a lot of training problems. Each time the net will attempt to produce the output from the input. The produced output is compared to the actual output and weights of the network are adjusted according to the back-propagation algorithm. Each cycle of the above is called an epoch. The supervised learning makes the network strong at learning more complex and non linear behavior. Because of these characteristics neural networks are successfully used by recommender systems. The winner of the Netflix Prize for predicting movie ratings, BellKor's Pragmatic Chaos, successfully employed a variant of neural networks called Restricted Boltzmann Machines as one of their prominent predictors. They also used NN's to blend the predictions made by the many different predictors they used. Our neural network will be based on a multilayer perceptron, which is a feed-forward network mapping the input data on a set of outputs. For every movie we create a separate neural network. This network will act as a predictor for that specific movie. All nets are trained separately with the training data

available for that movie. The input layer consists of the set of all movies except the movie of the network. Each movie input is represented by a probability distribution. Because the amount of inputs is over one thousand, we cannot afford to use a large number of hidden nodes. Both the number of calculations and the memory needed to store all the weights would become unfeasible. Different configurations were tried and showed 10 to be a sufficient number of hidden nodes. The output layer of is formed by 5 nodes. These nodes correspond to a probability distribution from 1 to 5.

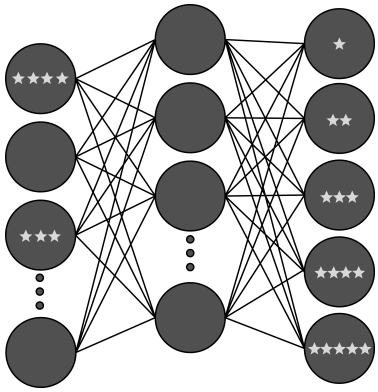


Fig. 1. From left to right; input layer, hidden layer and output layer.

6.1 Filling in the inputs

The input set consists of the ratings from a user for all the movies. Because a user has not rated all the movies the known rating set is only a subset of the required input set.

$$RatingSet_{user} \subseteq InputSet_{network}$$

A couple of options to fill in the inputs were explored and tested.

Only Known Inputs
Only using the known inputs, leaving the unrated movies unpopulated. This is a very simple option, essentially ignoring the unrated movies. However, this can produce odd behavior as the network is trained with only subset of the inputs filled, which is not the normal way for training neural networks.

Average Rating
The unknown inputs are filled in by taking the average movie rating. The idea

is to take an approximation of how the user would have rated that movie. The problem here would be that because a large part of the inputs are filled with approximated values, the known input of the user has less of an effect on the whole.

***k*-Nearest Neighbor Rating**
The unknown inputs are filled by doing a simple k-NN algorithm on the user. The same idea of using an approximation of a user's rating applies here. It also has the same issue of using approximations next to known ratings. We chose to use the average rating to fill in the unknown input. This gave better results than using only the known inputs, but is still very simple. Using the k-NN required more complexity and did not seem to improve the results.

6.2 Momentum

Back-propagation is used to train the neural network. To accelerate convergence, a momentum term is introduced. The momentum adds a fraction of the the previous weight update to the current. This prevents the network from converging to a local minimum in the early stages of training. The momentum is updated by combining the weight update with the last momentum.

$$W_{ij} = (\alpha * input_j * (\Delta Out_i * (1 - mFactor) + m_i * mFactor))$$

$$m_i = m_i * 0.8 + \Delta Out_i * 0.2$$

ΔO_i is the output delta
m_i is the momentum for output i
$mFactor$ is the variable for the influence of the momentum

In the beginning the momentum factor should be bigger, this allows the training to be more exploratory, as opposed to settling in a local minimum. The momentum is decreased during training, so the system will eventually settle in the minimum found.

6.3 Probability Distributions

Probability distributions allow us greater flexibility and provide more information than an ordinary number value. Distribution A: The probabilities of ratings $\{1, 2, 3, 4, 5\}$ are $\{0.3, 0.1, 0.2, 0.1, 0.3\}$. A holds more information then an average value of 3 would, because it also shows there is a higher probability of 5 and 1 opposed to the rest. A distribution also allows us greater flexibility in calculating a value from a distribution. The distributions form a integral part of the constructed neural network. The function to determine the actual value resulting from a distribution is very influential in the performance of the network. The function is first used to get a usable value as input to the network. It is important because the output of the network is also a probability distribution. We

calculate the overall performance using this output, so it is necessary we have a good translation function in order for the output value to be be as accurate as possible. We created and evaluated multiple functions.

Random Chance
This method is based on random chance. A random integer is picked according to the probability it has to be correct. Let distribution A be $\{0.5, 0.2, 0.2, 0.1, 0.0\}$ for $\{1, 2, 3, 4, 5\}$. If we would run this function the odds of returning 1 is 50%, for 2 it is 20% etc. This function does not work very well because if we look at distribution A it still has a 10% chance of returning 4 while that would make no sense according to the distribution.

Max Value
This is a very simple deterministic function that returns the integer with the highest probability. The reasoning is that the highest probability is the most likely to occur. This does not take into account that this could also result in a high error if the highest probability does not occur. Let A be $\{0.3, 0.3, 0.0, 0.0, 0.4\}$ for $\{1, 2, 3, 4, 5\}$. The Max Value function will always return 5 for this distribution. We can see this will result in a large error because in 60% the result will be 1 or 2, which both are far from 5. The behavior of this function can therefore be described as optimistic.

Average Value
The Average Value function is a simple deterministic function that returns the weighted average of the probabilities. If we would let A be $\{0.2, 0.2, 0.2, 0.2, 0.2\}$ the average would be 3.0. We can take B $\{0.1, 0.1, 0.1, 0.2, 0.5\}$ which results in

$$1 * 0.1 + 2 * 0.1 + 3 * 0.1 + 4 * 0.2 + 5 * 0.5 = 3.9$$

This function generally works good because it has a very cautious nature, it always takes all the probabilities into account. The downside is that this function is not very optimistic. Is the 10% chance in B realistic or is it just noise left from the neural network?

Average Value from Squared Probability
A variation on the Average Value function, this function is a more optimistic attempt. Instead of taking the probability at face value, it is instead squared first before being add together. The effect of this is that a 0.1 probability becomes 0.01, and 0.5 becomes 0.25. The 0.1 loses 90% of its weight, while 0.5 loses only 50%. The same distribution B $\{0.1, 0.1, 0.1, 0.2, 0.5\}$ from above results in

$$\frac{0.1^2 * 1 + 0.1^2 * 2 + 0.1^2 * 3 + 0.2^2 * 4 + 0.5^2 * 5}{0.1^2 + 0.1^2 + 0.1^2 + 0.2^2 + 0.5^2} = 4.59375$$

This function is thus more optimistic by adding weight to higher probabilities and marginalizing lower probabilities.

Testing the different functions
The different functions are tested by comparing the error rate over a large amount of predictions. For this test we compared the error rate on trained neural networks for over 100000 predictions.

	Random Chance	Max Value	Average Value	Average Value Squared
RMSE	1.50852	1.65905	1.01336	1.01637
MAE	1.15067	1.30292	0.81594	0.79605

Table 3. Comparing different functions

We can clearly see the average value and average value squared functions result in the the lowest error rates. The RMSE is lower for the average value and the MAE is lower for the average value squared. It is interesting to see the different functions being the best for different performance criteria. The correctness is not measured here because the average and average squared are not rounded. The best correctness was found by rounding the average value squared. The conclusion we can draw from this test is that the average value squared function makes riskier prediction than the average value function, because the RMSE is higher but the MAE is lower. In further tests we will use the average value for the RMSE and the average value squared for the MAE. For the correctness we will use the rounded average value squared.

7 Combining Neural Networks

When we have our basic neural networks we can chain them together. If one neural network can make reasonable predictions for a movie given the user's input set, then combining their knowledge could result in a even better prediction. The thought is that if the network can make predictions based on partially known input it would make better predictions with the whole input known. So we could use the predictions made by network A to fill in the rating for movie A in network B, if you were to make a prediction for movie B.

Fig. 2. Simplified schema of basic neural network, input left and output right.

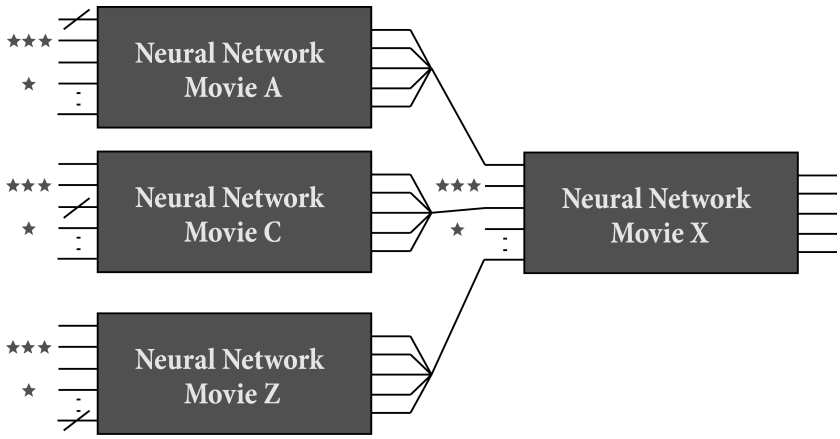
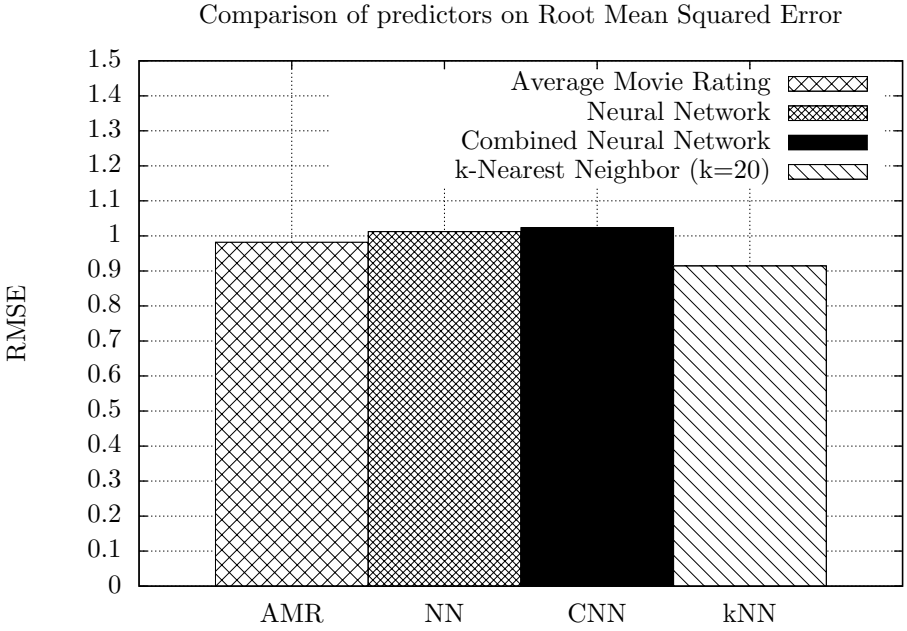


Fig. 3. Combined Neural Network, note that the ratings for B and D are known.

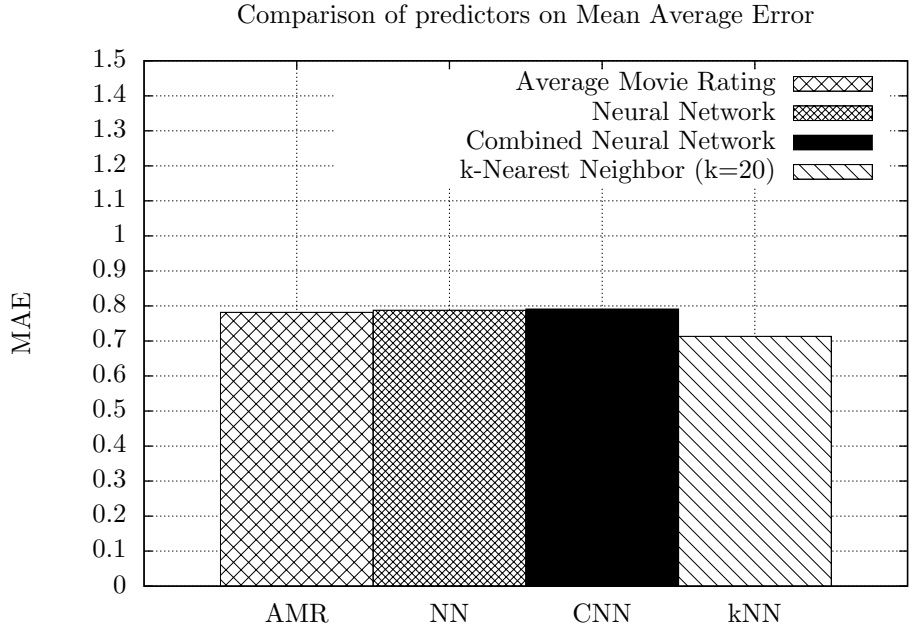
The network for movie A only uses the ratings from the users that have rated movie A. Using chained neural networks would allow us to make prediction using information from all the ratings. Chaining the networks this way should improve the predictions over using a single network. However, the prediction made by a single network should be a good approximation of the user's unknown rating. If this is not the case, the final network will work on badly approximated ratings, which will have a negative influence on the result.

8 Results

In this section we compare the different predictors to each other. All predictors have the same training set as known knowledge, and make predictions for the same test set. The predictors are tested by making large number of predictions and calculating their error rate. The performance criteria are the RMSE, MAE and correctness. The outcome of the tests can also be seen in table 2.



In this plot we can see the RMSE for the predictors. The k-NN algorithm performs the best, with a RMSE of 0.91424. The Average Movie Rating performs slightly better than the Neural Network predictor. The Combined Neural Networks performs slightly worse than the NN predictor.



In this plot we can see the MAE for the predictors. The predictors perform relatively the same on this metric. The differences between the predictors are a bit smaller, but small differences in MAE usually compare to bigger differences in RMSE.

	AMR	NN	CNN	k-NN (k=20)
RMSE	0.98218	1.01212	1.02370	0.91424
MAE	0.78165	0.78782	0.79120	0.71321
Correctness (in %)	37.932	36.309	36.760	43.672

Table 4. Comparison of predictors

We can see in the results that the k-NN algorithm has the best performance overall. It proves to have a better RMSE, MAE and correctness. The other three predictors are almost equal in performance, but the AMR is a slightly better predictor than the neural network. The CNN approach is comparable in performance to the NN. It has a slightly higher RMSE and MAE, but a better correctness. However, this difference is so small that we cannot draw any conclusions. Based on the performance of the NN compared to the AMR, it is expected that the CNN performs worse than the NN. This is because the CNN works on ratings generated by the NN. The NN itself works on ratings provided by the AMR. In these results we can see the NN produces worse output than the AMR, and therefore the CNN has worse input than the NN.

9 Conclusion

After analyzing the results conclusions can be drawn. The Neural Network predictor does not perform as well as the k-Nearest Neighbor algorithm. It even performs slightly worse than taking the average rating of a movie. In theory the network should be trained to the point where it knows which ratings on specific movies correlate to a rating for the network's movie. This is currently not noticeably happening. A possible explanation for this is lack of training data. Because the NN's are split on movie, we are essentially also splitting the data set. The reduced training set consists of 682705 ratings. The reduced movie set has 1419 movies, thus 1419 neural networks are created which on average only have 481 different input sets as training. Because not every movie is in the input set, the network is not trained often enough on every movie.

The combined neural network is based on the same neural network, only with different input values used. Because the performance of the basic network is not good enough, the combined network will not produce results better than the basic network. The effectiveness of linking the output of one neural network to the input of the other can only be properly tested when the output of one neural network is better than the standard input used. This is not the case because the NN performs worse than the AMR.

A neural network in the form used here would likely not be able to improve on existing methods. The difference in the RMSE (≈ 0.1) is too big to close with simple optimizations. Even with a larger data set it is not likely that this neural network based approach would succeed in besting k-NN. Most likely a change in set-up of the neural network is needed to improve beyond other methods.

10 Future Work

Despite this neural network testing worse than other methods, NN's cannot be written off. Other forms of neural networks have been proven to work for collaborative filtering. This setup, with a network per movie, does not seem to be a good recommender. It is possible that with a change in structure and/or training it would work better, allowing the network to really understand the underlying logic. Further improvements can be made by analyzing the ratings. People rate movies in a different way, some people are more likely to rate movies only 1's or 5's and other rate more subtle 2's or 4's. Does one user's 5 compare to another user's 5? Analyzing the rating pattern of a user can determine the quality of a rating. This can then be accounted for by adjusting the input ratings. These so called global effects can be partially accounted for[19]. Hybrid systems, using both rating information as information about the movies, are very interesting. Could meta information be incorporated into the structure of the neural network?

References

1. G. Linden, B. Smith and J. York. *Amazon.com Recommendations: Item-to-item Collaborative Filtering*. IEEE Internet Computing 7, pages 7680, (2003)
2. J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. *The YouTube video recommendation system*. In Proceedings of the fourth ACM conference on Recommender systems, ACM, pages 293-296, (2010)
3. G. Adomavicius and A. Tuzhilin.*Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. Knowledge and Data Engineering, IEEE Transactions on, Vol. 17, Iss. 6, pages 734-749, (2005)
4. X. Su and T.M. Khoshgoftaar. *A survey of collaborative filtering techniques*. Advances in artificial intelligence, 2009, Vol. 4, Iss. 13, (2009)
5. M. Rocha, P. Cortez and J. Neves. *Evolution of neural networks for classification and regression*. Neurocomputing, Vol. 70, Iss. 16, pages 2809-2816, (2007)
6. Website of the Netflix Prize, `http://www.netflixprize.com/`
7. H. Zhou and K. Lange. *Rating Movies and Rating the Raters Who Rate Them*. The American Statistician, Vol. 63, Iss. 4, pages 297-307, (2009)
8. Y. Koren. *The BellKor Solution to the Netflix Grand Prize*. Netflix prize documentation, `http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf`(2009)
9. A. Töscher and M. Jahrer. *The BigChaos Solution to the Netflix Prize*. Netflix prize documentation, `http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf` (2008)
10. M. Piotte and M. Chabbert. *The Pragmatic Theory Solution to the Netflix Grand Prize*. Netflix prize documentation, `http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf`
11. A. Töscher, M. Jahrer and R. Legenstein. *Combining Predictions for Accurate Recommender Systems*. `http://www.commendo.at/UserFiles/commendo/File/kdd2010-paper.pdf` (2010)
12. R. Salakhutdinov, A. Mnih, and G. E. Hinton. *Restricted Boltzmann Machines for Collaborative Filtering*. In ICML, pages 791-798, (2007)
13. D.E. Rumelhart, G.E. Hinton and R.J. Williams. *Learning Internal Representations by Error Propagation*. `http://www.cs.toronto.edu/~hinton/absps/pdp8.pdf` (1986)
14. K. Zaamout and J.Z. Zhang. *Improving Neural Networks Classification through Chaining*. LNCS, Vol. 7553, pages 288-295, (2012)
15. D. Mitchell and R. Pavur. *Using modular neural networks for business decisions*. Management Decision, Vol. 40, Iss. 1, pages 58-63, (2002)
16. GroupLens Research Group, `http://grouplens.org/`
17. MovieLens Data Set, `http://grouplens.org/datasets/movielens/`
18. G. Adomavicius and J. Zhang. *Stability of recommendation algorithms*. ACM Transactions on Information Systems. Vol. 30, Iss. 4, Art. 23, (2012)
19. R. M. Bell and Y. Koren. *Scalable collaborative filtering with jointly derived neighborhood interpolation weights*. In Data Mining, ICDM 2007, Seventh IEEE International Conference on, pages 43-52, (2007)