MovieMatch
A Collaborative Filtering System

Omer Meshar, Data Mining
University of Tel-Aviv
July 2002

Abstract

MovieMatch is a collaborative filtering system, which enables users, to rate and get predicted rates for movies. The system gathers the users' ratings and compares them with other users in the database. It finds the closest neighbors for the user, and predicts the user's rating for a certain movie, based on his neighbors' ratings of that movie. This algorithm is close to the standard nearest-neighbor technique [1]. I tested this algorithm to find the number of neighbors which gives the best Normalized Mean Absolute Error (NMAE) and compared to the POP and AverageShift algorithms discussed later on.

1. Introduction

The networked world contains a vast amount of data. Visitors face the arduous task of retrieving information that matches their preferences. The term "Collaborative Filtering" (CF) describes techniques that use the known preferences of a group of users to predict the unknown preferences of a new user; recommendations for the new user are based on these predictions [11]. Other terms that have been proposed are "Social information filtering" [14], and "Recommender system" [12]. In each case, users *collaborate* in the sense that each rating improves the performance of the overall system. The fundamental assumption is that if users A and B rate k items similarly, they share similar tastes, and hence will rate other items similarly. Approaches differ in how they define a "rating," how they define k, and how they define similarly.
A CF algorithm should be both accurate (the recommended objects should subsequently receive high ratings) ,and efficient in terms of computational complexity. A CF database represents n users and m items available for rating and recommendation. In most existing CF algorithms, online computation scales linearly with n. *MovieMatch* is one of them. Each prediction of a rating is done by looking at every available user in the database, finding the closest users to the current logged-in user, and predicting the rating based on these users' ratings on the movie at hand.

This paper is organized as follows: Section 2 reviews related work. Section 3 introduces the MovieMatch algorithm. Section 4 describes the implementation of *MovieMatch*. Section 5 proposes the normalized Mean Absolute Error (NMAE) metric and compares performance of several algorithms on the dataset in terms of accuracy and efficiency. Section 6 reviews the results and discusses future work.

2. Related Work

In this section we review only a small sample of the papers on Collaborative Filtering. Rich [13] is considered an early reference. There is a long history of patents related to CF, ranging from [8] in 1989 to [5] in 2000. In 1992, D. Goldberg et. al.

coined the term "collaborative filtering" in the context of a system for filtering email using binary category flags [6].

Shardanand and Maes [14] designed a collaborative filtering system for music (Ringo) and experimented with a number of measures of distance between users, including Pearson correlation, constrained Pearson correlation, and vector cosine. They compare four different recommendation algorithms based on the Mean Absolute Error of predictions. All of their neighborhood-based algorithms require time linear in the number of users.

GroupLens is a pioneering and ongoing effort in collaborative filtering [12, 9, 10, 2 and 7]. The GroupLens team initially implemented a neighborhood-based CF system for rating Usenet articles. They used a 1-5 integer rating scale and computed distance using Pearson correlations.

Breese et. al. [3] classify collaborative filtering algorithms into two classes: Memory-based and Model-based. Memory-based algorithms operate over the entire user database to make predictions. The most common memory-based models are based on the notion of nearest-neighbors, using a variety of distance measures. Model-based systems are based on a compact model inferred from the data. In this framework MovieMatch would be considered Memory-based. Breese et. al. compare a number of algorithms including Bayesian clustering and decision-tree models. They show that Bayesian network and correlation models are the best-performing but do not discuss computational complexity.

Delgado [4] takes an agent-based approach to CF, developing several algorithms that combine ratings data with other sources of information such as the geographic location of the user. Weighted majority voting is used to combine recommendations from different sources. In their recent paper, Herlocker et. al. [7] divide neighbor-based CF algorithms into three steps:
i) weighting possible neighbors, ii) selecting neighborhoods and iii) producing a prediction from a weighted combination of neighbors ratings. They explore alternative methods for each step and propose Spearman (rank-based) correlation weighting as an alternative to Pearson correlations and a "significance weighting" based on the number of items two users have rated in common. To compute predictions they find that subtracting global means improves performance, while conversion to Z-scores does not.

O'Connor and Herlocker discuss the option of using clustering within a CF system, both for better efficiency and performance [15].

3. The MovieMatch Algorithm

3.1 Notation and Terminology

U	set of all users in database
J	set of all items in database
n	number of users $\lvert U \rvert$
m	number of items $\lvert J \rvert$
$r_{\min}, r_{\max}$	min and max rates respectively
r_{ij}	raw rating of item j by user i; $r_{ij} \in [r_{\min}, r_{\max}] \cup 0$
$\bar{r}_i$	average item rating for user i

p_{ij} predicted rating of item j for user i

3.2 Nearest Neighbor Algorithms

The MovieMatch algorithm is based upon the Nearest-Neighbors Algorithm (NN(x)), where x defines the number of neighbors taken into account, when predicting the rate. These algorithms are the ones most widely referenced in the literature.

The formula generally used to find the predicted rating p_{ij} for user i and item j is

$$p_{ij} = \bar{r}_i + \kappa \sum_{p=1}^{l} w(i,p)\left(r_{pj} - \bar{r}_p\right)$$

where κ is a normalizing factor ensuring that the absolute value of the weights sum to 1, and l is the number of neighbors we consider in our prediction (or the number of users available that rated the item in question – whichever is smaller). The weights $w(i,p)$ can reflect distances, correlations, or similarities between user i and user p that have rated the same items.

I used for the MovieMatch algorithm, a weight most commonly used - the Pearson Correlation Coefficient (PCC) between users i and p:

$$w(i,p) = \frac{\sum_j \left(r_{ij} - \bar{r}_i\right)\left(r_{pj} - \bar{r}_p\right)}{\sqrt{\sum_j \left(r_{ij} - \bar{r}_i\right)^2 \sum_j \left(r_{pj} - \bar{r}_p\right)^2}}$$

where the summations over j include items that both user i and p have rated in common [1,4,14]. I found out that when calculating the averages of the users in the PCC, it is better to take the average on the ratings of each user on the common items only, and not take the whole average of each user's ratings. κ in this case and implementation, is:

$$\kappa = 1 \Big/ \sum_{p=1}^{l} \left| w(i,p) \right|$$

This is actually a weighted NN algorithm, which gives to a closer user, a bigger part of the predicted rating. In order to find out the value of x to be used in the system (the number of neighbors to be taken account when predicting), I built a test which examined the performance of a range of x (from 1 to 160), the results are discussed further in the Results section (5).

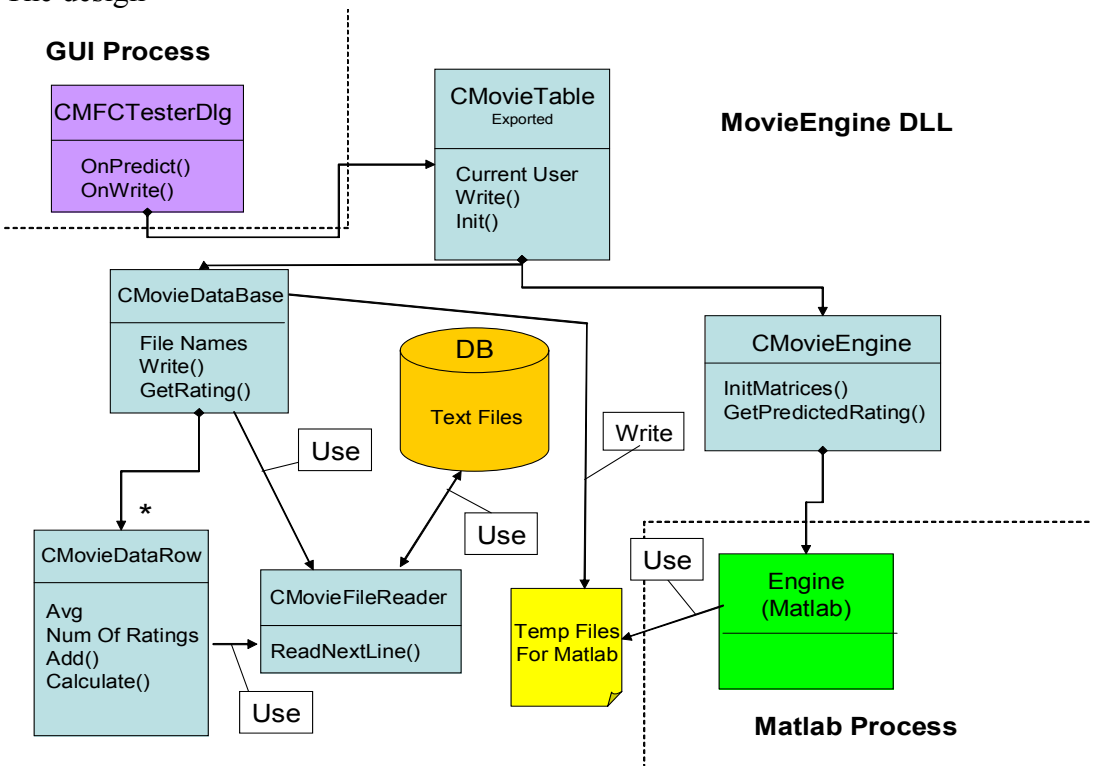
4. The MovieMatch Implementation

The system is implemented in *C++* and *MFC*, whereas the engine itself is implemented in *Matlab*. The database is based on the database distributed on the web by *GroupLens* [2] and consists of 100,000 ratings from 943 users on 1682 movies. Each user has rated at least 20 movies, each rating is an integer between 1 and 5, 1 being the least favored rate and 5 being the best favored rate. A new user will be given the opportunity to rate at least 15 movies, before the system can begin to predict

ratings for him/her. After that, the user can see his ratings, see the predicted ratings coming from the algorithm and so on.

The initialization of the system takes a while – this includes reading the database from the files, putting the information in the memory and writing it to temporary files, which in turn are read by the *Matlab* engine. Whenever a new user logs in, his/her corresponding weights are calculated and entered as well to the *Matlab* engine for later use when predicting.

The design -



5. Results

5.1 Normalized Mean Absolute Error (NMAE)

The error metric used most often in the CF literature is the Mean Absolute Error (MAE) [14, 3, and 11]. If p_{ij} is the prediction for how user *i* will rate item *j*, the MAE for user *i* is defined as

$$MAE = \frac{1}{c}\sum_{j=1}^{c}\left|r_{ij} - p_{ij}\right|$$

where *c* is the number of items user *i* has rated. MAE for a set of users is the average MAE over all members of that set. Since our numerical rating scale gives ratings over the range [1:5], we normalize to express errors as percentages of full scale: Normalized Mean Absolute Error is:

$$NMAE = \frac{MAE}{r_{\max} - r_{\min}}$$

NMAE is the expected percentage of error in our predictions. For example, if the NMAE is 0.20, then the predicted rating is expected to be, on average, off by a fifth of the scale away from the actual rating. Herlocker et al. [7] discusses a variety of other error measures. In the Appendix, I consider NMAE from a theoretical perspective.

5.2 The MovieMatch Test

I tested the MovieMatch algorithm with a changing number of neighbors to get the best number of neighbors to use in the system. The algorithm is also tested against the POP algorithm (see 5.3) and my own AverageShift algorithm (see 5.4). The results can be viewed in the figures 1-2 below.

5.3 The POP Algorithm

The simplest recommendation algorithm is to treat all users as coming from the same global cluster and to base recommendations for all users on global mean ratings. I used this "POP" algorithm as my control case. (Note: The name "POP" is taken from [3]). POP predicts ratings for every movie based on its global average. I use the training set to compute the global average and the test set to evaluate the predictions. POP yields NMAE of 0.209.

5.4 The AverageShift Algorithm

Another simple way of recommending is taking the overall average rating (for all users and movies), calculating the current user's average and the questioned movie's average, and predicted rating is the sum of the overall average, and the differences between it and the averages calculated :
OverallAvg + (*UserAvg*-*OverallAvg*) + (*MovieAvg*-*OverallAvg*)
This technique is straight forward: we would like to have the predicted rating to be influenced both by the user's average on other items, and by the item's average from other users. For example, if the user's ratings are above the mean of ratings, we would anticipate that his next rating will be above average as well. The same goes to a movie, therefore their difference from the overall rating are considered in the prediction. The AverageShift algorithm is just as efficient as the POP algorithm, making the prediction in $O(1)$ time, if the averages are already calculated. Looking closely, you can see that it's basically the same prediction, while adjusted by the current user's average prediction. AverageShift yields NMAE of 0.1914.

5.5 The Test

For the test, I took the data from the database and split it up to 80%/20% parts, the first being the training data set and the second the test data set. This was done 5 times, each time with a disjoint test part from the others. The training sets were used as the data for the system, and the test sets were used for comparing the predicted ratings of the system, and the real rating recorded in the set. Each test included predictions made by the POP algorithm, the AverageShift algorithm and by NN(x) (Nearest-Neighbor) algorithm, where x is the number of neighbors used in the prediction. I ran the test 5 times, one for each data set, and calculated the average NMAE, for each algorithm.

As can be seen in figure 1, the NN(1) algorithm is doing worse than the POP algorithm, but soon enough, with NN(4)... and more, the NN algorithm predicts much better and gets as low as 0.1914 where x=70-90.

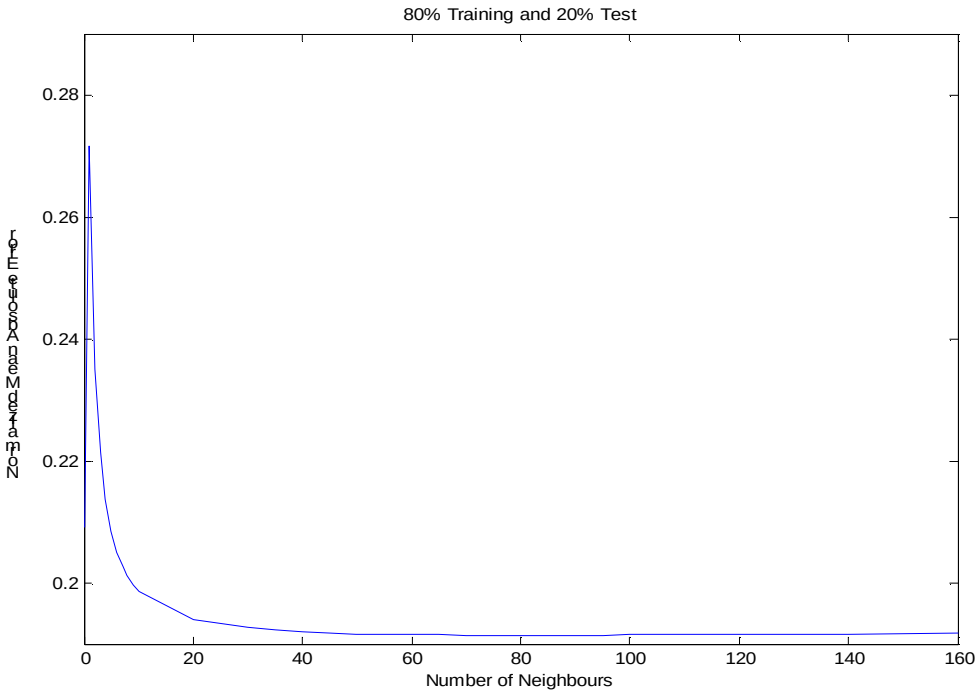


Figure 1 Average NMAE for POP (Number of Neighbors = 0) and for NN(x).

As can be seen in the following table (Figure 2), the POP algorithm does better than the 1-4 NN, but worse than the others. The best performing algorithm was when x = 70-90, performing at 0.1914 NMAE. When increasing x further, the NMAE slowly increases. The AverageShift algorithm does the same as the best NN algorithm, but is a much more efficient.

Algorithm	Average NMAE
POP	0.2090
AverageShift	0.1914
NN 1	0.2716
NN 3	0.2214
NN 5	0.2084
NN 10	0.1987
NN 30	0.1927
NN 70	0.1914
NN 100	0.1915
NN 160	0.1918

Figure 2 Average NMAE for POP, AverageShift and for NN(x).

It seems from the results, that the best algorithm to use is the AverageShift algorithm. In spite of that, I chose to implement in my system the NN (70) algorithm, which takes into account only close users with relatively similar opinions to the current user, and not the global average rating of all users.

6. Discussion

In this section I would like to discuss some comparisons with related work, and to discuss what can be done to improve the algorithm/performance.
The Normalized Mean Absolute Error (NMAE) values we got using the MovieMatch system indicates, that predicted ratings values will be within roughly 20% of the true ratings values. This means that items with predicted ratings well above the mean for a new user will, in many cases, correspond to desirable items for that user. This of course suggests that a system like *MovieMatch* will be of use to a user wanting to get a suggestion for a movie.

It is interesting to note that these accuracies are comparable with those reported by Herlocker et al [7] and Eigentaste [1]. Herlocker et al reported NMAE of 0.192 to 0.207 on a database much like our own, a database of movies and ratings between 1 and 5. Eigentaste reports NMAE on a completely different data set (jokes), but its NMAE can be easily compared, and is between 0.187 for NN (80) and Eigentaste algorithms, and 0.203 for POP. Looking closer, you can see that the improvement of NN(x) to POP is about 0.017 in both Eigentaste and *MovieMatch*.

The POP (global mean) algorithm offers a useful baseline for accuracy (see Appendix on other baseline comparisons). In terms of NMAE, the POP algorithm performs reasonably well, as other researchers have found [7]. It is computationally efficient but completely ignores differences between users. Nearest-neighbor methods offer improved accuracy, unless not enough neighbors are considered (e.g. NN (1)), which makes individual recommendations highly susceptible to noise.
If the 70 nearest neighbors are considered (NN (70)), noise is reduced and accuracy improves about 8.5% over POP, but at the cost of considerable online computational as the number of users grows. Of course it may be possible to pre-process the user group to select or create a small number of representative users ("mentors") to keep n small. This could be one way of improving the efficiency of this algorithm, but it may cause accuracy problems. Another way to improve efficiency is clustering the users or movies, so that each prediction will look at the respective cluster and not at the whole dataset [15]. This clustering should be done at the pre-processing stage, and updated once in a while.

Appendix: NMAE for random predictions.

In the appendix, I will try to compare the CF performance to random guessing. This is based upon the same comparison made in the appendix of Eigentaste [1].
In order to compare with random guessing, I use the Uniform distribution model to estimate NMAE analytically. Let X be the user's rating and Y be the predicted rating. Let X and Y be independent uniform random variables on the interval [1, 5]. The probability distribution of the error, X-Y, is a triangular function over the range [-4, 4].

Taking the absolute value, folds this function onto the positive axis. Normalizing to integrate to 1, the MAE density function, $|X-Y|$, is $f(x) = 0.5 - 0.125x, 0 \le x \le 4$
The expected value for the MAE of $|X-Y|$ is

$$E[MAE] = \int_0^4 (0.5 - 0.125x)x\,dx = 1.334$$

Normalizing over the range of values, NMAE = 0.333. That is, if actual and predicted values are uniformly distributed, we'd expect the random predictions, on average, to be off by a third of the scale. (In the comparison made in [1], the same result was found, which is expected, because the only difference is the range of values, which doesn't matter in the uniform model.) As we can see, all of our algorithms do better than the random guess algorithm.

References

[1] *Eigentaste: A Constant Time Collaborative Filtering Algorithm* which talks about "*Jester*", an online joke recommending system. The paper discusses also nearest-neighbor techniques.
See: http://www.ieor.berkeley.edu/~goldberg/pubs/eigentaste.pdf

[2] GroupLens database.
See: http://www.cs.umn.edu/Research/GroupLens/

[3] Breese, Heckermen, and Kadie. *Empirical analysis of predictive algorithms for collaborative filtering*. Microsoft Research Technical Report, (MSR-TR-98-12), October 1998.

[4] Joaquin A. Delgado. *Agent-Based Information Filtering and Recommender Systems on the Internet.* PhD thesis, Nagoya Institute of Technology, February 2000.

[5] Alexander Chislenko et al. US patent 6092049: Method and apparatus for efficiently recommending items using automated collaborative filtering and feature guided automated collaborative filtering. 18 July 2000.

[6] David Goldberg, David Nichols, Brian Oki, and Douglas Terry: *Using collaborative filtering to weave an information tapestry.*
Communications of the ACM, 35(12):61-70, 1992.

[7] Jonathan Herlocker, Joseph Konstan, Al Borchers,
and John Riedl. *An algorithmic framework for performing collaborative filtering.* In Proceedings of the SIGIR.ACM, August 1999.

[8] John Hey. US patent 4870579: *System and method of predicting subjective reactions*. 26 September 1989.

[9] J.A. Konstan and K. Bharat: *Integrated personal and community recommendations in collaborative filtering*. In CSCW Workshop, 1996.

[10] Joseph Konstan, Bradley Miller, David Maltz, Jonathan Herlacker, Lee Gordon, and John Riedl: *Grouplens: Applying collaborative filtering to usenet news.* Communications of the ACM, 40(3):77-87,March 1997.

[11] D.M. Pennock and E. Horvitz: *Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach.* In IJCAI Workshop on Machine Learning for Information Filtering, Stockholm, Sweden, August 1999. International Joint Conference on Artificial Intelligence.
[12] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl: *Grouplens: An open architecture for collaborative filtering of netnews.*
In Proceedings of the ACM Conference on Computer Supported Cooperative Work, 1994.

[13] Elaine Rich: *User modeling via stereotypes*. Cognitive Science, 3:335-366, 1979.

[14] U. Shardanand and P. Maes: *Social information filtering: Algorithms for automating word of mouth*. In ACM Conference on Computer Human Interaction (CHI), 1995.10

[15] Mark O'Connor and Jon Herlocker: *Clustering Items for Collaborative Filtering*. Dept. of Computer Science and Engineering, University of Minnesota MN.