On-line Learning of User Preferences in Recommender Systems

Joaquin Delgado and Naohiro Ishii
Nagoya Institute of Technology
Dept. of Intelligence & C.S.
Gokiso-cho, Showa-ku, Nagoya 466-8555
JAPAN
{jdelgado,ishii}@ics.nitech.ac.jp

Abstract

Recommender Systems, sometimes called Collaborative Filtering, are learning systems that make use of a database representing multi-user preferences over items (e.g. a **score**(*user*,*item*) matrix), to try to predict the preference towards new items or products regarding a particular user. The objective is to filter and present to the user the options that are most attractive for him; i.e. he would probably like the most. This paper is about on-line algorithms for performing such task. In particular, we introduce a new algorithm for on-line learning of user preferences called Voting-by-Category Weighted Majority Prediction (VC-WMP) algorithm. In a nutshell, it proposes a.) Clustering of items by categories in order to reduce the dimensionality and sparseness of the score matrix; and b.) A variant of a majority vote learner with selection of voters based on the correlation of user profiles. We then give some experimental results comparing our results to other approaches.

1 Introduction

Recently, in the Artificial Intelligence (AI) and Information Retrieval (IR) community, there has been a great deal of work on how to develop smart applications in scenarios such as the Internet and the World-Wide Web, as emerging platforms for distributed intelligent systems. Notions of personalized Search Engines, Intelligent Software Agents, and *Recommender Systems* have gained large acceptance among users for the task of assisting them in searching, sorting, classifying, filtering and sharing the vast amount of information now available on the Web.

Most of the information filtering software agents now available, such as e-mail filters, push technology and content-based filtering systems, makes use of an interface, trough which the system prompts the user to explicitly enter his preferences, desires and intentions in the form of attribute-values or rules. Although this process can provide very precise results when done properly, it can be both time consuming and sensitive to input errors. It also requires the user to have a good understanding of domain knowledge and actions performed by the agents. The manual updating turns out to be much more difficult when the requirements change. From the system point of view, it is desirable to know as much as possible from the users so that the system can provide useful results from the first actions of the agents. However, the user should be willing to provide only limited time for creating the specification of his user profiles. Moreover, user's interests may change over time, making the profiles difficult to maintain. This is where Recommender Systems play a very important and complementary role.

Recommender Systems are learning systems that use a database about multi-user preferences over items, to try to predict the preference towards new items or products with regard to a particular user. The objective is to filter and present to the user the options that are most attractive for him; i.e. he would probably like the most. These systems usually do not maintain complete descriptions or profiles of the users' for mining their interests. Instead, it uses a *rating schema* in order to produce statistical learning or prediction of ratings (scores) of new items or classes of items. The system then bases its analysis on the user's previously captured ratings and the ratings of the *other users in the system* over the same item/class (see Figure 1). Recommendations are then extracted from this analysis that reflects the social patterns and analogies among the different users within the system.

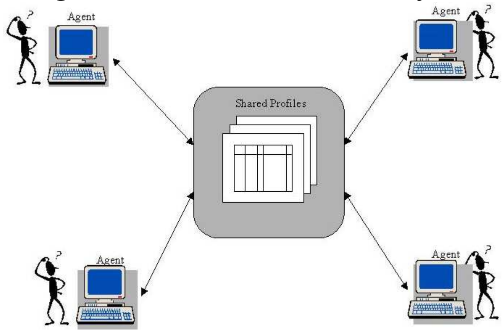

Figure 1. Recommender Systems Users share their *interests*, and obtain *recommendations* based on what other users have to say about related *items*.

This process, known in the literature as *Collaborative Filtering* [Resnick et. al 1994] or *Social Filtering* [Shardanand and Maes 1995], was originally proposed for the automation of the "word-of-mouth" type of phenomena, being very suitable for entertainment evaluation (e.g. recommendations of movies, CD, books, etc.). Since their first appearance these systems have evolved into more sophisticated *Recommender Systems*[1] that present different levels of automation, ranging from explicit entry of the rates to automatic generation of the rates based on the observation of users' behavior. Several ways of combining Content-based and Collaborative-based information for Recommender systems have been proposed in [Delgado, Ishii, and Ura 1998], [Delgado and Ishii, 1998] and [Basu, Hirish and Cohen, 1998].

Although increasingly gaining commercial acceptance, little has been published about the core algorithms used in Recommender Systems.

In this paper, we investigate the topic of learning algorithms in Recommender Systems. In particular, we concentrate on *on-line learning,* as we believe it is more faithful to the idea of learning by making use of recommendations, thought as a continuous, adapting and interactive process. We also introduce a new on-line algorithm for learning of user preferences under this context, called Voting by Category Weighted Majority Prediction (VC-WMP) algorithm, which proposes a.) Clustering of items by categories in order to reduce the dimensionality and sparseness of the **score**(*user*,*item*) matrix, and b) A variant of a majority vote learner based on the correlation of user profiles. We then give some experimental results comparing our results to other approaches.

Learning in Recommender Systems

Learning of user preferences in Recommender Systems could be thought, in principle, as a joint learning problem for binary relations, in which each user is related to an item just in case he or she prefers it. A 0,1-valued matrix, in which the rows represent users and columns represent items (instances), can represent such a binary relation

$$\begin{array}{c} \\ U_1 \\ U_2 \\ \vdots \\ U_m \end{array} \begin{array}{c} \begin{array}{cccc} I_1 & I_2 & \dots & I_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \end{array}$$

Figure 2. M: The Joint Information Matrix.

We define this joint learning model as follows. We have an instance space X with an underlying unknown distribution, and a target concept class C:$X\rightarrow\{0,1\}$, representing the class of user preferences over a given domain D. Let **M** (see Figure 2) be the joint information matrix that represents the target binary function whose i,j-entry ($a_{i,j}$) represents the value of a target function $c_i(x_j)$, where $c_i\in C$ and $x_j\in X$, and $\mathbf{O}^t$ an observation matrix **O**, at time t, that in general satisfies $\mathbf{O}[i,j] = \mathbf{M}[i,j]$, whenever the i,j entry has been observed, and $\mathbf{O}[i,j]= *$ otherwise. Let $P(t)$: $\mathbf{O}^t\times\{i,j\} \rightarrow\{0,1\}$ be the prediction function for a given observation matrix **O** at time t, a column (instance) i and a row (user) j.

On-line or *incremental learning* is easily described in terms of the observation matrix **O** as follows. Starting initially with $\mathbf{O}^0$ whose elements are all $*$, at any given time t an arbitrary pair i,j is given to $P(t)$ that predicts its value as $\mathbf{M}^\wedge[i,j]$, based on the observation matrix $\mathbf{O}^t$. The learner is given the actual value of $\mathbf{M}[i,j]$, and $\mathbf{O}^t$ is updated (to $\mathbf{O}^{t+1}$) accordingly. The above process is repeated until the matrix is fully observed, namely $\mathbf{O}^t$=**M**.

Off-line or *batch learning,* although clearly not suitable for this type of ever-changing problem, can be thought as a one time learning process that receives as input a partially filled (usually small) matrix **O** as training data, then produces a prediction function P: $\{i,j\} \rightarrow\{0,1\}$ later used to predict the remaining unknown entries (test data). Note that in this case time is irrelevant, as well as the actual values $\mathbf{M}[i,j]$ corresponding to the test data.

In the literature, several ideas for implementing $P(t)$ have been proposed. It's worth mentioning the widely used Pearson-r/Correlation algorithm [Resnick et.al. 1994]. Several of well-known Machine Learning (ML) algorithms have also been implemented and evaluated for learning collaborative filters [Billsus and Pazzani 1998]. Inductive learning has also been explored in [Basu, Hirish and Cohen, 1998]. An interesting empirical analysis of several algorithms was made by [Breese, et.al. 1998], where the following classification of collaborative filtering algorithms was presented:

- Memory-based Algorithms
  - Correlation (Person-r). Extended with default voting and case amplification
  - Vector Similarity (tf-idf)
- Model-based Algorithms
  - Cluster Models (Naïve Bayes Classifier with EM for the estimation of the model structure)
  - Bayesian Network Model

As for evaluation criterias, *precision, recall* and *accuracy* are widely used. In [Breese, et.al. 1998], since the prediction was the actual score of the user instead of a more simple *like-dislike* (binary) score, two other evaluation metrics were proposed – one appropriate for individual item-by-item recommendations and another appropriate for ranked lists.

On-line Learning Algorithms

From a theoretical point of view, on-line algorithm falls into what machine learning and computational learning theory (COLT) calls the mistake-bound model. Sometimes called "Learning from experts advice", this on-line learning model is a continuous and interactive process, in

[1] We prefer this name opposed to *collaborative filtering* since in many cases there is no notion of explicit collaboration.

which each attribute is considered to be an "expert predictor" and is given a weight used to measure its confidence on the prediction task. In each trail, a valid instance $\vec{x}$ is presented to the algorithm, and each predictor gives its verdict using a certain Boolean testing function. The weighted majority output is

$$\arg\max_{R\in\{R_o,R_1\}} \sum_R a_i(\vec{x})w_i$$

where

$$R_o = \{a_i(\vec{x}) = 0; \forall i \leq n\} \quad R_1 = \{a_i(\vec{x}) = 1; \forall i \leq n\}$$

and $a_i(\vec{x})$ is the result of applying a basic Boolean test to the i-th component of $\vec{x}$, made of n attributes. Next, the correct label is shown to the algorithm that proceeds to update the weights of the experts denoted by w_i, following a strategy that punishes those that made mistakes and rewards or leaves unchanged the weights of those that were correct. The algorithm goes back to the prediction phase and loops.

Note that there is a subtle difference between Memory-based algorithms and On-line algorithms. The former neither punishes nor rewards weights used for the prediction after knowing the correct result. Instead, it calculates the weights solely based on the data available. The later does use a correcting scheme that reflects the actual performance of the algorithm.

Although on-line algorithms are not assured to converge (it might be because the target concept function changes in time!), there are guarantees on theoretical bounds on the number of mistakes they can make.

On the other hand, in this paper we refer to Model-based and Off-line learning indistinguishably.

Voting-by-Category WMP

In this section we describe our main contribution: the Voting by Category Weighted Majority Prediction (VC-WMP) on-line algorithm, proposed in this paper.

The original weighted majority prediction algorithm for learning binary relations was introduced by [Goldman and Warmuth, 1995], and later extended for recommender systems by [Nakamura, A. and Abe, N., 1998]. It makes its prediction $\mathbf{M}^{\wedge}[i,j]$ by weighted majority voting by all rows i' such that the entry $\mathbf{M}[i',j]$ in that same column has been observed, each weighted by the weight $w_{i,i'}$ representing the believed degree of similarity between the rows (users) i and i'. If the prediction of the algorithm was wrong, the weights are updated by multiplying those contributing to the correct value by $(2-\gamma)$ and those contributing to a wrong value by γ for some $0\leq\gamma<1$. Note that if $w_{i,i'}$ is initialized with 1, the update is equivalent to defining the weight $w_{i,i'}$ at each failed trial as $(2-\gamma)^{Gi,i'}\gamma^{Bi,i'}$ where Gi,i' is the number of times i' has voted for a correct value and Bi,i' the number of times it voted for a wrong value when predicting a value in row i.

This approach presents several problems, namely:

1. In some applications the dimensions of the observation matrix **O** may become unmanageable making impractical calculations over the whole matrix.
2. The weights $w_{i,i'}$, as described before, captures the degree of similarity between rows, based solely on the entries (votes) given by rows on the same column. Therefore, if the observation matrix is too sparse, i.e., few votes on the same column/item, which is not rare, it can lead to serious miscalculations. This problem occurs frequently in memory-based algorithms (e.g. in correlation) and it is treated as "missing values" for the off-line learning case.
3. Normally, **M** will consist of multi-value (non-binary) entries.
4. Similarity between rows is calculated only by using information from the column to which the entry its trying to predict belongs to. It is not aware of the scores given on other columns. In other words, it captures similarity between rows without considering information from the complete scoring pattern.

In order to cope with these problems first we need a way to reduce the dimensionality of the matrix, and by doing so, solve the sparse problem without distracting our attention from the basic idea of predicting recommendations based on other's votes. Later we need to adapt WMP as for accepting non-binary values for the voting. Finally, we should include a component in the weighted majority voting in order to consider similarities suggested by the scores given on other columns and maybe for the selection of voters based on their similarities.

In [Billsus and Pazzani 1998], the first point was already noticed. To solve the problem they used Singular Value Decomposition (SVD) as to reduce the dimensions of the 0,1-valued matrix required as input for the various ML algorithms reported in the paper.

We propose a rather simpler approach, better suited for on-line learning. Instead of using a matrix transformation we assume the existence of a small number of generated *clusters* or pre-existing *categories* to which items can be assigned. For example, in the *EachMovie*[2] database, used as data set for our experiments, each movie belongs to at least one of the 10 existing *genres* (e.g. action, romance, suspense, thriller, etc.). Once the categories of a given item have been determined[3], the matrix **O** can be converted straightforward into a matrix with fewer dimensions. In this case, the number of rows is the number of users of the system, but the number of columns now reflects the number of categories instead of the number of items. We now define an entry of this new matrix **O'** as the average over the scores of the items per each category for a given user; that is $\mathbf{O}'[i,c]$ = Avg_c ($\mathbf{O}[i,j]$), where c is the index of the category and j the movies of that category *seen* by the user.

Clusters were also used in [Breese, et al. 1998]. The main difference with their approach is that we do not use clusters while learning with an off-line algorithm, such as

[2] For more information see: http://www.research.digital/SRC/EachMovie/

[3] A movie may belong to several genres, for example *action* and *thriller*.

Naïve-Bayes Classifier, but rather use them in conjunction with an on-line learning algorithm, such as WMP. On the other hand, clustering techniques such as the Expectation-Maximization (EM) algorithm [Dempster, et. al 1977] could be considered for the generation of the categories if these do not exists beforehand.

As for the second problem, since we are more interested in predicting preference (like/dislike), instead of the numerical ratings itself, we consider the actual vote as a function $V(s)$ such that $V(s) = 1$ if they actual score s is above a given threshold τ, and $V(s) = 0$ otherwise.

Finally, for solving the third point introduce a *correlated coefficient* as a component for the voting. We use the reduced matrix **O**' for this purpose. Instead of using plain entries for determining positives and negative votes, we use the standard correlation between the active user i and the other users i' [Resnick et al., 1994], defined as:

$$correl(i,i') = \frac{\sum_c (a_{i,c} - \bar{a}_i)(a_{i',c} - \bar{a}_{i'})}{\sqrt{\sum_c (a_{i,c} - \bar{a}_i)^2 \sum_j (a_{i',c} - \bar{a}_{i'})^2}}$$

Where $a_{i,j}$ is the corresponding entry **O**'$[i,c]$. Then we compare its absolute value to a given threshold θ in order to decide if a vote is positive or negative. The rest that follows is the same as in the original WMP.

Now we describe our algorithm in pseudo-code:

VC-WMP(user *i*, item *j*, param γ, threshold θ) :-
  Initialization:
  For every i'<>i do $w_{i,i'} \leftarrow 1$
  Prediction:
  For every non observed entry *i*,*j* of **M** do {
    majority ← 0
    minority ← 0
    recommend ← False
    midpoint ← Midpoint_Of_the_Voting_Scale()
    $R_1 \leftarrow \{\varnothing\}$
    $R_0 \leftarrow \{\varnothing\}$
    c ← Largest_Category_Of(*j*)
    For every observed i'<>i do
      If (*correl*(i,i') >= θ and **O**'[*i*,*c*] > *midpoint*) {
        majority ← *majority* + ($w_{i,i'} \times$**O**'[*i*,*c*])
        $R_1 \leftarrow R_1 \cup \{i'\}$
      }
      Elseif ((*correl*(i,i')<= (- θ) and **O**'[*i*,*c*] > *midpoint*)
        or (*correl*(i,i') >= θ and **O**'[*i*,*c*] <= *midpoint*)) {
        minority ← *minority* +($w_{i,i'} \times$Max(**O**'[*i*,*c*],1- **O**'[*i*,*c*]))
        $R_0 \leftarrow R_0 \cup \{i'\}$
      }
    If (*majority*>=*minority*) { *recommend* ←True }
  Weight Update:
    If (*recommend* and *V*(**M**[*i*,*j*]) = 0)   /* False Pos. */
      For every $i' \in R_0$ do
        $w_{i,i'} \leftarrow w_{i,i'} (2-\gamma)$
      For every $i' \in R_1$ do
        $w_{i,i} \leftarrow w_{i,i'} \gamma$
    ElseIf (not *recommend* and *V*(**M**[*i*,*j*]) = 1) /* False Neg */
      For every $i' \in R_1$ *do*
        $w_{i,i'} \leftarrow w_{i,i'} (2-\gamma)$
      For every $i' \in R_0$ do
        $w_{i,i'} \leftarrow w_{i,i'} \gamma$
}

In **VC-MVP,** instead summing over weights, majority voting is performed over the sum of the weights multiplied by the value of the averaged score over the category of item j. That is, voting according to:

$$\arg\max_{R \in \{R_O, R_1\}} \sum_R w_{i,i} \times \mathbf{O'}[i',c]$$

If an item j belongs to several categories, we select c among them, as the category that holds the largest averaged score for the user i'. Largest_Category_Of(*j*) means exactly this in the pseudo-code.

	Like c	*Dislike c*
Highly Correlated	+	-
Poorly Correlated	-	?

Table 1. Voting Schema. Positive and Negative votes according to 2 dimensions: correlation and likeliness.

As shown in table 1, we consider as "positive" votes the ones that are issued by highly correlated users who "like" category c (who's averaged score for category c is above the midpoint of the rating scale). On the other hand, "negative" votes are issued by highly correlated users that happen to "dislike" c (averaged score below the midpoint) or poorly correlated users who happen to "like" c.

The idea here is very intuitive: similar users tend to like (and dislike) the same things, meanwhile dissimilar users usually have opposite tastes. We decided not to make any assumptions in the Poorly Correlated/Dislike case, since it was somewhat more controversial. We considered this last case and all the other alternatives, we haven't mentioned, as null votes. This selective voting reduces the number of rows participating in the majority voting and therefore reduces the computational complexity when comparing our algorithm to the original WMP algorithm. It also adapts according to the information available in the database, used to calculate the correlation between users.

Experiments and Results

We conducted several experiments to confirm the usefulness of our approach. Below we explain about the data and methodologies used and later report on some of the significant results.

Data and Methodology

We used for our experiments data from the *EachMovie* database available from the System Research Center of Digital Corporation Equipment [McJones, 1997]. This database was constructed during an 18-month period in which *EachMovie*, a movie recommender service, hosted 72,916 users and received ratings for 1,628 movies. The database contains scores from 6 levels (0.0, 0.2, 0.4, 0.6, 0.8, 1.0) to express preference.

In order to resemble as much as possible real recommender systems, even in the start-up phase, we considered the following.

1. The data used for our analysis was restricted to the first 2,000 users, since we are interested in handling conditions in which the ratio of users to items is low and it is less likely to have overlap of rated items. These 2,000 users provided ratings for 1,410 different movies.
2. The chronological order in which the rates are received is considered for the learning process. Since we use an on-line prediction algorithm, it is important to simulate the possible state of the database at the time the prediction is going to be made. For example, we cannot predict the vote of a user *i* for a movie *j* based on scores given to *j* by other users *after* the date of the actual prediction. This would be very unrealistic. Instead, we have to consider growing pattern of the database by date.

In order to compare to other approaches we used an experimental protocol similar to the one used in [Billsus and Pazzani 1998], complying with the aspects mentioned above:

- Select 20 randomly chosen users from the initial 2,000, but with the following constraints. First, only users with average score within [0.25, 0.75] are considered. Otherwise, scores that indicate high precision of our algorithm might be biased by the fact that there are some users in the database who either like/dislike everything or just gave rating to movies they like/dislike. Second, each user should have at least 80 movies rated in order to later average results. Next, order their entries by date and try to predict votes following this order just by using a "view" of the original database restricted by the date of the entry being predicted.

We ran experiments for VC-WMP with this protocol to evaluate overall performance on a per user basis. We used $\theta = 0.75$ as a threshold parameter for high correlation and $\tau = 0.7$, the mid-point between 0.6 and 0.8, as the threshold for the deciding function *V*, so only scores above this threshold are considered as liked by the user. For the evaluation we used classification *accuracy*, *precision* and *recall*, commonly used in IR.

Accuracy is the number of correct predictions regardless of the output (like/dislike), precision is the number of movies recommended by the algorithm that users actually liked, and recall the number of movies liked by the users that were recommended to them.

Summary and Results

Our experiments reported interesting results, as shown in Table 1. In particular, it is important to emphasize that both precision and recall are essentially high. At this point its important to remember that the system's goal is to provide high quality recommendations, not plain quantity, since this would overwhelm the users with options (and ratings to make). Little matters about non-recommended movies that aren't indeed considered by the user as interesting. We are also interested in systems that can improve with time and adapt to the changes of preferences shown by users. Although we only present the final (maximum) values obtained, both precision and recall showed slow but positive slopes as the number of entries augmented.

	Accuracy	*Precision*	*Recall*
Micro-averaged	75.86333	**77.5225**	**87.22757**
Macro-averaged	**76.15409**	77.09881	82.07003

Table 1. Summary of the Results. Micro-average is averaging over the whole set; Macro-average computes results on individual basis and averages them at the end.

Comparing these results to the ones presented by [Billsus and Pazzani 1998], using their SVD/Artificial Neural Network algorithm, over the same data set and very similar protocol, we obtained 76.154%, an improvement in the order of nearly 9% in accuracy against 67.9% obtained by them in the general case. If we calculate the *F-measure*[4] we obtained 82.0891%, against 68.8%, an improvement of more than 13%.

Although we cannot compare directly with the results given in [Breese, et.al. 1998], due to the difference in evaluation metrics and protocols, we noticed that all the algorithms employed in their experiments were of-line algorithms, with the exception of the correlation variant denoted as *CR+*. These algorithms cannot be easily applied on live data, and are only suited for static frameworks, which generally is not the case of real life Recommender Systems. In the *EachMovie* dataset, the off-line methods they proposed did not show significant improvement over the modified correlation method.

These results are encouraging and show that on-line algorithms can lead to improved predictive performance while conserving the on-line nature of Recommender Systems.

Conclusion

We have introduced a new on-line weighted majority prediction algorithm and a variation, for tackling the problem of quality recommendations in Recommender Systems by making use of correlated voting by category. After conducting basic experiments, we empirically demonstrated the usefulness of our approach, achieving high results while using standard metrics for information retrieval on a well known data-set. It is left for future research the theoretical upper and lower mistake bounds of our algorithms, as well as a complete formulation within the binary-relation learning model.

[4] F-measure = 2*(Precision*Recall) /(Presicion+Recall)

References

Basu, C., Hirsh, H, and Cohen, W., 1998. Recommendation as Classification: Using Social and Content-Based Information in Recommendation, in: Proceedings AAAI-98. 714-726. Menlo Park, Calif. AAAI Press.

Billsus, D. and Pazzani, M., 1998. Learning Collaborative Filters, in: *Proceedings of ICML'98,* 46-53. Morgan Kaufman Eds.

Breese, J., Heckerman, D., Kadie, C., 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI.

Delgado, J., Ishii N. and Ura, T., 1998. Content-based Collaborative Information Filtering: Actively Learning to Classify and Recommend Documents, in: Matthias Klush, Gerhard Weiss, eds., Cooperative Agents II, Proceedings/CIA'98 206-215. LNAI Series Vol. 1435, Springer-Verlag.

Delgado, J. and Ishii, N., 1998. Content + Collaboration = Recommendation, in Papers from the AAAI 1998 Workshop on Recommender Systems, 37-41. Technical Report WS-98-08, Menlo Park, Calif. AAAI Press.

Dempster, A., Laird, N., and Rubin,D., 1977. Maximum likelihood from incomplete data via the EM Algorithm. *Journal of the Royal Statistical Society*, B 39:1-38

Goldman, S. and Warmuth, M.. Learning Binary Relation using Weighted Majority Voting. *Machine Learning* 20, 1995, pp. 245-271

Goldman S., Rivest, R and Shapire R., 1993. Learning binary relations and total orders. *SIAM Journal of Computing*, 22(5): pp. 1006-1034

McJones P., 1997. EachMovie collaborative filtering data set. DEC Systems Research Center. http://www.research/digital.com/SRC/eachmovie

Nakamura, A. and Abe, N., 1998. Collaborative Filtering using Weighted Majority Prediction Algorithms in: *Proceedings of ICML'98*, 395-403. Morgan Kaufman Eds.

Resnick, P., Iacovou, N., Sushak, M., Bergstrom P. and Riedl J., 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews,in: Proceedings of the CSCW 1994 conference.

Shardanand, U. and Maes, P., 1995. Social Information Filtering: Algorithms for Automation of “Word of Mouth”, in: *Proceedings of ACM/CHI'95*. ACM Press.