Memory-Based Weighted-Majority Prediction

for Recommender Systems

Delgado, Joaquin and Ishii, Naohiro
Department of Intelligence & C.S., Nagoya Institute of Technology
Gokiso-cho, Showa-ku,
Nagoya 466-8885 JAPAN
+81 (52) 735-5473
{jdelgado,ishii}@ics.nitech.ac.jp

ABSTRACT

Recommender Systems are learning systems that make use of data representing multi-user preferences over items (e.g. **Vote** [*user*, *item*] matrix), to try to predict the preference towards new items or products regarding a particular user. User preferences are in fact the learning target functions. The main objective of the system is to filter items according to the predicted preferences and present to the user the options that are most attractive to him; i.e. he would probably like the most. We study Recommender Systems viewed as a pool of *independent* prediction algorithms, one per every user, in situations in which each learner faces a sequence of trials, with a prediction to make in each step. The goal is to make as few mistakes as possible. We are interested in the case that each learner has reasons to believe that there exists some other target functions in the pool that consistently behaves similar, neutral or opposite to the target function it is trying to learn. The learner doesn't know *a priori* which ones, but is willing to discover and use this information for the benefit of its own prediction. A simple, yet effective method is introduced for constructing a compound algorithm for each learner. Prediction is based on transformations over the original preferences, recorded as votes over the same items by other users, according to similarity patterns and updated weights, combining memory-based individual prediction and on-line weighted-majority voting. We prove mistake bounds for this algorithm that are closely related to the total loss of the best predictor in the pool.

Keywords

Recommender Systems, On-line learning, Memory-based algorithms, Collaborative Filtering, User Modeling.

1. INTRODUCTION

In general, the task in Recommender Systems is to predict the votes of a particular user (called the active user) over a given subject or item, for deciding its recommendation. The prediction is usually done from a database of user votes from a sample or population of other users. In Memory-based collaborative filtering algorithms [1], commonly used for Recommender Systems, the vote prediction of an active user (indicated with a subscript *a*) is done based on some partial information regarding the active user and a set if weights calculated directly from the entire user-vote database. It is assumed that the predicted vote of the active user for item *j*, $p_{a,j}$, is a weighted sum of the votes of the other users:

$$p_{a,j} = \bar{v}_a + \sigma \sum_{i=1}^{n} ws(a,i)(v_{i,j} - \bar{v}_i) \qquad (1.1)$$

where $\bar{v}_a, \bar{v}_i$ are the mean vote of user *a* and *i* respectively. And *n* is the number of users in the database with non-zero weights that have voted over item *j*. The weights $ws(a,i)$ express the *similarity* between each user *i* and the active user *a*. σ is a normalizing factor such that the absolute values of the weights sum to unity.

If we combine statistical formulation and the vector space model, then this similarity can be expressed in terms of the cosine or normalized inner (dot) product of two vectors V_a and V_i:

$$ws(a,i) = \frac{\sum_{j=1}^{m} V_{a,j} \times V_{i,j}}{\sqrt{\sum_{j=1}^{m} (V_{a,j})^2 \times \sum_{j=1}^{m} (V_{i,j})^2}} \qquad (1.2)$$

Where *m* is the total number of subjects or items rated both by user *a* and user *j*.

When the components of each vector are deviations from the mean of the votes, e.g. $V_{i,j} = (v_{i,j} - \bar{v}_i)$, equation 1.2 becomes the statistical *correlation coefficient* first used for collaborative filtering in [2]. On the other hand, if we use the Term Frequency – Inverse Document Frequency (TF-IDF) coefficient [4] as components, equation 1.2 becomes the *document similarity measure*, widely used in Information Retrieval (IR). In this case, an item is considered to be a term, a vote represents term frequency and a user is analogous to a vector representing a "document". Although there is no well founded theory for supporting these types of algorithms, several variations as well as empirical results have been shown useful for the collaborative filtering task [1, 3].

In this paper, we study Recommender Systems viewed as a pool of *independent* prediction algorithms, one per each user, in situations in which a learner faces a sequence of trials, with a prediction to make in each step, and the goal is to make as few mistakes as possible. We are interested in the case that each learner has reasons to believe that there exists some other target functions in the pool that consistently behaves similar, neutral or opposite to the target function it is trying to learn. The learner doesn't know *a priori* which ones, but is willing to discover and use this information for the benefit of its own prediction. A simple yet effective method is introduced for constructing a compound algorithm for each learner. Prediction is based on transformations over the original preferences, recorded as votes over the same items, according to similarity patterns and updated weights combining memory-based [1] individual prediction and on-line weighted-majority voting [5]. We prove mistake bounds for this algorithm that are closely related to the number total loss of the best predictor in the pool.

2. ON-LINE LEARNING ALGORITHMS

On-line algorithms fall into what computational learning theory (COLT) call the mistake-bound model. Sometimes called “Learning from experts advice”, the on-line learning model is a continuous and interactive process, in which there is a pool of algorithms each of which 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 algorithms, and each predictor gives its verdict from the set {0,1} (binary prediction). The weighted-majority output is:

$$\underset{R\in\{R_o,R_1\}}{\arg\max}\sum_R a_i(\vec{x})wc_i \qquad (2.1)$$

where $R_o=\{a_i(\vec{x})=0;\forall i\le n\} \qquad R_1=\{a_i(\vec{x})=1;\forall i\le n\}$

and $a_i(\vec{x})$ is the result of the prediction of the *i*-th algorithm out of *n* algorithms voting over $\vec{x}$. Next, the correct label is shown to the algorithm that proceeds performing a multiplicative update to the weights of the experts denoted by wc_i, following a strategy that punishes the weights of those algorithms 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.

As a natural extension of this notion for on-line prediction of collaborative filters with binary votes, we can define a binary prediction for the active user *a* over the item *j* is as:

$$p_{a,j}=\underset{R\in\{R_o,R_1\}}{\arg\max}\sum_R^n wc_{a,i}(v_{i,j}) \qquad (2.2)$$

where

$$R_o=\{v_{i,j}=0;\forall i\le n\} \quad R_1=\{v_{i,j}=1;\forall i\le n\}$$

Note that the weights, $wc_{a,i}$, are not based on the data present in the database. Instead, they are initialized non-negative numbers updated in each trial.

If votes and predictions are continuous, from the interval [0,1], then the following weighted sum prediction formula is used:

$$p_{a,j}=\frac{\sum_{i=1}^n wc_{a,i}(v_{i,j})}{\sum_{i=1}^n wc_{a,i}} \qquad (2.3)$$

For this case, the notion of mistake has to be replaced by a quantity that measures how far the prediction is from the correct label. In this paper we will use the absolute loss. If an algorithm predicts $p_{a,j}$ and the correct label is $v_{a,j}$ then the its loss in that trial is $|\, p_{a,j} - v_{a,j}|$; this definition applies both to algorithms in the pool and the master algorithm.

There are several ways one can perform the updates of the weights. Updates could be done on every trial or just in the ones where a mistake occurs. The most common is the standard Weighted-majority Algorithm [5] that updates the weights only when the prediction is wrong by multiplying those contributing to a wrong value by γ for some fixed value $0\le\gamma<1$. The Winnow algorithm [6] and variants also multiply the weights contributing to a correct value by a factor of $(2-\gamma)$. Note that in this case the update is equivalent to defining the weight $w_{a,i}$ at each failed trial as $(2-\gamma)^{Ga,i}\gamma^{Ba,i}$ where *Ga,i* is the number of times *i* has voted for a correct value and *Ba,i* the number of times it voted for a wrong value when predicting a value.

The mistake-bounded model, upon which these weighted-majority algorithms are built on, has a nice set of theoretical and empirical support in the literature [5, 6, 7, and 8]. Unfortunately, they present some serious problems when applied to collaborative filtering, namely:

1. They were originally designed for a pool of algorithms that are “experts” predicting the same target function, and the objective is to give more confidence (weight) to the sub-pool of algorithms that make less mistakes while punishing the erratic algorithms bringing benefits for the majority voting. Nothing is said about totally independent, possibly unrelated or opposite target functions, used as well for the prediction.
2. Votes are considered absolute values which may not be the case when different levels of similarity exists among the target functions in the pool and the target function being predicted. For example, in the binary case, if an algorithm *i* votes consistently with a target function that is totally opposite to the target function being learned, it would be helpful to consider the inverse function of the votes for the prediction.

Contrary to On-line learning algorithms, Memory-based algorithms, that calculate weights solely based on the data available, do consider the relativeness among votes. For example, when the correlation coefficient is used, weight *ws*(*a*,*i*), which expresses the degree of similarity between users, is a real number in the interval [-1,1]. An actual vote (deviation from the mean), that express the opinion of a user over a given item, can be a positive value, zero or a negative value. It easy to verify that the sign of the contribution of the vote in the final equation (Eq.1) is determined by the following table:

	$v_{i,j}>\bar{v}_i$	$v_{i,j}<\bar{v}_i$
$ws(a,i)>0$	+	-
$ws(a,i)<0$	-	+

Table 1. Voting Schema in Memory-Based Algorithms.

This can be interpreted in words as “similar users tend to like (and dislike) the same items, meanwhile dissimilar users usually have opposite tastes”, which is very intuitive.

To summarize this section, we would say that a weight *wc* in On-line algorithms represents the “confidence” on each expert’s prediction, whether a weight *ws* in Memory-based algorithms, represents “similarity” among the predictors and the active target function. Although related, the semantics behind the weights in both cases are not the same. On the other hand both are calculated from predictions done in the past. Thus, the former is done by updates done in each trial considering only the value of the weight in the previous trial and the actual prediction, whether the later is a calculation on all the historical data accumulated up to the actual trail.

A more interesting idea, explored in this paper, is the combination of both approaches and the proposal of a theoretically grounded and robust algorithm for collaborative filtering.

3. SIMILARITY-BASED PREDICTION

In this section we introduce the main contribution of this paper. We present the first version of an algorithm that combines characteristics of Memory-based prediction and On-line prediction. We call it *MbWM*, after Memory-based Weighted-majority algorithm. First, we will define an individual prediction as a function of the original vote and the memory-based similarity measure between users. Then we explain how weights are updated and finally we will prove some mistake bounds on this master algorithm for an active user *a*, closely related to the total lost of the best predictor in the pool.

Assumptions for *MbWM*: The predictions of the master algorithm, as well as the values of the target functions of the pool and the labels associated with the instances are assumed to be in [0,1].

3.1 Individual Prediction

Lets recall that each vote $v_{i,j}$ is in fact a value of the target function that represents user's preference, and that *it is not a prediction*. Thus, for realizing real on-line prediction we will have to convert this value into an individual prediction, denoted by $x_{a,i,j}$, that relates the active user *a*, user *i* and the subject *j*. This individual predictor is defined as:

$$x_{a,i,j} = \bar{v}_a + ws(a,i)(v_{i,j} - \bar{v}_j) \qquad (3.1)$$

The formula is, in essence, the same as equation 1.1 for n=1 (i.e., when the prediction relies on just one *other* user). Note that it captures the relativeness among users and votes depicted in Table 1. It also includes the means of both the active user and the user *i* in the calculation as in Memory-based algorithms. For the benefit of the theoretical analysis, users that cannot vote (e.g. they haven't rated item *j*) are given as default vote the value of their mean. From now on *n* refers to the total number of users in the pool.

We are now ready to integrate this concept into weighted-majority prediction. By substituting $v_{i,j}$ by $x_{a,i,,i,}$ in equation 2.3 we obtain:

$$p_{a,j} = \frac{\sum_{i=1}^{n} wc_{a,i}(x_{a,i,j})}{\sum_{i=1}^{n} wc_{a,i}} \qquad (3.2)$$

Let us introduce a new notation that will enable us to refer to values that occur in each trail in which an update step occurs. We use the term *update-trail t* to refer to the *t*-th trial in which an update step occurs. We assume that there is a total of *T* such trails. We assume that the master algorithm predicts an unknown preference for the active user *a* over an item or subject *j* and is applied to a pool of *n* users/predictors. Every trial corresponds to a different item or subject. This allows us to drop the *j* index, making reference only to the trail *t*. Let $x^{(t)}_{a,i}$ denote the prediction of the *i-th* user of the pool in update-trail *t*. Accordingly, let $wc^{(t)}_{a,i}$, $p^{(t)}_{a,i}$ and $v^{(t)}_{a}$ denote the weights, the prediction of the master algorithm, and the label, respectively. We assume that all initial weights $wc^{(1)}_{a,i}$ are positive. Let $s^{(t)}_a = \sum_{i=1}^{n} wc^{(t)}_{a,i}$, thus $s_a^{(1)} = wc_{a,init}$ and $s_a^{(T+1)} = wc_{a,fin}$.

3.2 Update step for *MbWM*

In *MbWM* we require updates to be done in each trail. That allows us to calculate exactly the total loss of the master algorithm, $m_a = \sum_{t=1}^{T}\left|p^{(t)}_a - v^{(t)}_a\right|$. In an update step each weight $wc^{(t)}_{a,i}$ is multiplied by some factor *F* that depends on β, $x^{(t)}_{a,i}$ and $v^{(t)}_{a}$:

$$wc^{(t+1)}(a,i) = Fwc^{(t)}_{a,i} \qquad (3.2)$$

Where *F* can be any factor that satisfies

$$\beta^{\left|x^{(t)}_{a,i} - v^{(t)}_a\right|} \le F \le 1-(1-\beta)\left|x^{(t)}_{a,i} - v^{(t)}_a\right| \qquad (3.3)$$

The following lemma implies that such a factor exists; in particular either the upper or the lower bound given for *F* can be chosen as the update factor.

Lemma 3.1: *For* $\beta \ge 0$ *and* $0 \le r \le 1$, $\beta^r \le 1+r(\beta-1)$.

Proof It is easy to check the inequality in the case $\beta = 0$. If $\beta > 0$ then the inequality follows the convexity of β^r as a function of *r* for any $\beta > 0$. Convexity implies that for $0\le r \le 1$, $\beta^r \le (1-r)\beta^0 + r\beta^1$, which is another way of writing the inequality.

The next lemma is the basic lemma used to derive bounds for the loss of the master algorithm *MbWM*.

Lemma 3.2: *Assume that* $w^{(1)}_{a,i}>0$ for $i=1,\ldots,n$. *Assume* $0\le\beta\le1$, $0\le v^{(t)}_a \le 1$ *and* $0\le x^{(t)}_{a,i} \le 1$ for $t=1,\ldots,T$ *and* $i=1,\ldots,n$. *Assume* $wc^{(t+1)}_{a,i} \le wc^{(t)}_{a,i}(1-(1-\beta)\left|x^{(t)}_{a,i} - v^{(t)}_a\right|)$ for $t=1,\ldots,T$ *and* $i=1,\ldots,n$. *Then if* $\beta = 0$ and $\left|p^{(t)}_a - v^{(t)}_a\right| = 1$ *for some j in* {1,…,T} *then* $wc_{a,f,in} = 0$. *Otherwise*

$$\ln\frac{w_{a,fin}}{w_{a,init}} \le \sum_{t=1}^{T}\ln(1-(1-\beta)\left|p^{(t)}_{a,i} - v^{(t)}_a\right|)$$

Proof First we deal with the case where β=0 and there is a trail *t* such that $\left|p^{(t)}_a - v^{(t)}_a\right| = 1$. This forces the use of update factors that make $wc^{(t+1)}(a,i) = 0$ for all *i*. Thus $w_{fin} = 0$, as desired. Where this is not the case, we have from the inequality 3.3 that

$$s^{(t+1)}_a \le \sum wc^{(t)}_{a,i}(1-(1-\beta)\left|x^{(t)}_{a,i} - v^{(t)}_a\right|) = s^{(t)}_a - (1-\beta)\sum_{i=1}^{n} wc^{(t)}_{a,i}\left|x^{(t)}_{a,i} - v^{(t)}_a\right|$$

By the triangle inequality, the above is bounded by

$$s^{(t)}_a - (1-\beta)\sum_{i=1}^{n} wc^{(t)}_{a,i}\left|x^{(t)}_{a,i} - v^{(t)}_a\right| = s^{(t)}_a - (1-\beta)\left|p^{(t)}_a s^{(t)}_a - v^{(t)}_a s^{(t)}_a\right|$$

Thus

$$s^{(t+1)}_a \le s^1_a \prod_{t=1}^{T}(1-(1-\beta)\left|p^{(t)}_a - v^{(t)}_a\right|$$

Taking logarithm on both sides gives the desired result. □

3.3 Proving Mistake Bounds

Recall that for *MbWM* all predictions and *labels* are allowed to be in [0,1] and that the prediction in trail *t* (that corresponds to item *j*) is described as $p^{(t)}_a$ or $p_{a,j}$ when referring to the item instead of the trail, as in equation 3.2. The following lemma gives an upper bound on the total loss m_a of *MbWM* for the active user *a*.

Lemma 3.3: *If the conditions of Lemma 5.2 are satisfied then*

$$\sum_{t=1}^{T}\left|p_a^{(t)}-v_a^{(t)}\right| \le \frac{\ln\dfrac{wc_{a,init}}{wc_{a,fin}}}{1-\beta}$$

Proof The lemma follows the observation that $\ln(1-(1-\beta)\left|p_a^{(t)}-v_a^{(t)}\right|) \le -(1-\beta)\left|p_a^{(t)}-v_a^{(t)}\right|$ and lemma 3.2. □

Theorem 3.1: *Let S be any sequence of instances and labels with labeles in* [*0,1*]. *Let* m_a *be the total loss of MbWM on the sequence S when applied to some active user a and a pool of prediction algorithms. Then,*

$$m_a \le \frac{\ln\dfrac{wc_{a,init}}{wc_{a,fin}}}{1-\beta}$$

Proof The theorem follows immediately from Lemma 3.3□

Corollary 5.1: *Assume that* Λ *is a pool of n prediction algorithms for the active user a, and that* m_i *is the loss of the i-th algorithm of the pool on a sequence S of instances with labeles in* [*0,1*]. *If we apply MbWM to pool* Λ *with equal initial weights, then it will have a total loss of at most*

$$\frac{\ln n + m_i \ln\dfrac{1}{\beta}}{1-\beta}$$

on the sequence S, for $1 \le i \le |\Lambda|$.

Proof This follows from the above theorem and the fact that $w_{a,init}$ = n and $w_{a,fin} \ge \beta^{m_i}$ □

4. RELATED WORKS

Weighted-majority binary prediction algorithms for recommender systems were first introduced by Nakamura and Abe in [9] based on the theory of learning binary relations using weighted-majority voting [10]. They see the prediction task as 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

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

Figure 1. M, The Information Matrix**.**

We will briefly explain their model. It consists of instance space *X* and a target concept class *C*: $X\rightarrow\{0,1\}$, representing the class of user preferences over a given domain *D*. **M** (see Figure 1) is the 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 **O**[*i,j*] = **M**[*i,j*], whenever the *i,j* entry has been observed, and **O**[*i,j*]= ∗ otherwise. *P*(*t*): $\mathbf{O}^t\times\{i,j\} \rightarrow\{0,1\}$ is 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 **M**[*i,j*], and $\mathbf{O}^t$ is updated (to $\mathbf{O}^{t+1}$) accordingly. It makes its prediction $\mathbf{M}^{\wedge}[i,j]$ by weighted-majority voting by all rows *i'* such that the entry **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-γ) and those contributing to a wrong value by γ for some $0\le\gamma<1$. The above process is repeated until the matrix is fully observed, namely $\mathbf{O}^t$=**M**.

We differ to their work in several aspects. In order to obtain an acceptable theoretical analysis they strictly followed the original idea of learning binary relations with weighted-majority voting. In this case the prediction task only considers one master algorithm for *the entire* matrix and not a pool of independent prediction algorithms. This, in principle, is denying the intrinsically parallel nature of the prediction process. They also restrict their analysis to the binary domain, converting into binary the scores and the predictions, originally real numbers from a given scale. Although they present somewhat interesting empirical results on artificial data when comparing to Memory-based algorithms (Pearson-r algorithm [2] and its variations), they fail to give conclusive results on real data. Finally, the also fail to realize that instead of a binary relation, *different* and *independent* (possibly opposite) target functions are being learnt. Just by using the simple weighted-majority schema there is not sufficient information for capturing relative cases such as “low vote and low confidence = high vote” which we do treat in this paper.

In previous work [11], we gave empirical support to a slightly different version of the *MbWM* algorithm presented in this paper. In that opportunity we did experiments on the *EachMovie*[1] database, but instead of using the raw votes over items, we used averaged votes over movie categories in order to avoid sparseness. We obtained encouraging results while comparing to Memory-based algorithms and other *off-line* Machine Learning algorithms for collaborative filtering, such as the ones presented in [12].

5. SUMMARY AND CONCLUSIONS

Weighted-Majority and Winnow are on-line learning algorithms whose forte is in “learning simple things really well”. These algorithms have advantages of being fast and incremental, of being able to quickly focus on relevant predictions, of adapting well to target concepts that change with time and having reasonable accuracy/coverage tradeoffs as well as a well founded theoretical base. On the other hand they were designed for a pool of algorithms predicting over the same target function. This makes them somewhat inefficient when using information from independent target functions which is the case of collaborative filtering and Recommender Systems. Recommender Systems are incremental by nature and already some attempts have shown that

---

[1]For more information on the *EachMovie* database see:

http://www.research.digital/SRC/EachMovie/

on-line learning algorithms perform well for the voting prediction task.

Memory-based algorithms have been widely used for collaborative filtering and recommender systems. Because of their statistical nature, they need great amount of data and time in order to work properly and give interesting results. They also have slow response to changes, and do not react accordingly to changes in the target function and do not have a strong theoretical background. Yet, a positive aspect of Memory-based algorithms is that they cover relative voting among independent target functions, by considering the similarity among the users and the covariance of their votes (at least when the correlation coefficient is used).

In this paper we present a novel combination of both approaches, looking forward to merge the benefits of both worlds. We presented *MbWM* (Memory-based Weighted-majority) a weighted-majority algorithm specifically tailored for recommender systems. We study Recommender Systems viewed as a pool Λ of *independent* prediction algorithms, one per every user in the database, in situations in which each learner faces a sequence of trials, with a prediction to make in each step. By defining an algorithm's individual prediction as a function of the original vote (target function) and the similarity measure between users we were able to combine both Memory-based and On-line prediction. Weighted-majority is performed for the prediction of the master algorithm for active user *a*, updating the weights in each trail. Finally, we will prove some upper bounds on the total loss of the master algorithm for a particular active user *a* in the order of $\boldsymbol{O}(\log|\Lambda| + m_b)$, proportional to the size of the pool and the total lost of the best predictor *b* with respect of user *a* in the pool.

As future work, experiments with the *MbWM* algorithm have to be performed over standard data sets such as *EachMovie* and Microsoft's *MS Web* [1], in order to obtain empirical support for the ideas and the theory presented in this paper. Furthermore, to continue theoretical work on Machine Learning and Recommender Systems seems to be very promising for the future of the field.

6. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] Salton, G., Buckley, C., 1990. Improving retrieval performance by relevance feedback, *Journal of the American Society for Information Science* Vol. 41, 288-297

[5] N. Littlestone and M.K.Warmuth. The Weighted-majority algorithm. *Information and Computation*, 108(2):212-261, 1994.

[6] Littlestone, N. Learning quickly when irrelevant attributes abound A new linear-threshold algorithm. *Machine Learning*, 2:285-318, 1988.

[7] Blum, A. On-Line Algorithms in Machine Learning (a survey). This is a survey paper for a talk given at the Dagstuhl workshop on On-Line algorithms (June '96)

[8] Blum, A. and Mitchell, T. Empirical support for Winnow and Weighted-Majority based algorithms: results on a calendar-scheduling domain. Machine Learning 26:5-23, 1997.

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

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

[11] Delgado, J., Ishii N., 1999. Online Learning of User Preferences in Recommender Systems. Proceedings of the IJCAI-99 Workshop on Machine Learning for Information Filtering.

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