Forgetting mechanisms for incremental collaborative filtering

João Vinagre[1] and Alípio Mário Jorge[1,2]

[1] DCC, Faculdade de Ciências, Universidade do Porto, Portugal
[2] LIAAD-INESC Porto L.A., Portugal
c0016001@alunos.fc.up.pt, amjorge@fc.up.pt

Abstract. Collaborative filtering (CF) has been an important subject of research in the past few years. Many achievements have been made in this field. However, many challenges still need to be faced, namely the ones related with scalability and predictive ability. In this paper we present and evaluate the impact of two forgetting mechanisms – sliding windows and fading factors – in user-based and item-based CF algorithms with implicit binary ratings. Using forgetting mechanisms, the system is able to forget older and potentially obsolete data, avoiding unnecessary memory and computing time consumption. Our results suggest that forgetting mechanisms reduce time and space requirements, improving scalability, while not significantly affecting the predictive ability of the algorithms.

Keywords: Collaborative filtering, recommender systems, web mining, data streams, forgetting

1 Introduction

Collaborative filtering (CF) has been successfully used in a large number of applications, such as e-commerce websites [9] and on-line communities in a series of domains [13,7,14]. However, some challenges are still present. Most of these systems deal with huge amounts of data and frequently suffer from scalability problems. Additionally, a collaborative filtering system should be able to efficiently process data on-line as it arrives, in order to keep the system up-to-date. However this poses two problems:

- Scalability: as new users and items enter the system, time and memory requirements increase. At some point, the data processing rate may fall below the data arrival rate.
- Accuracy: as new data elements add up, the weight of each individual data element decreases. This causes the system to become less and less sensitive to recent information.

In order to overcome these problems, forgetting mechanisms [15,8,11] can be implemented. When forgetting older data, it is possible to impose bounds in time and memory and maintain the system's sensitivity to recent data.

In this work, we present two forgetting mechanisms: sliding windows and fading factors. We look at user activity as a data stream [3,5,1] in which data elements consist of individual user sessions, each containing a set of implicitly rated items. Then we implement and evaluate forgetting mechanisms in non-incremental and incremental versions of CF algorithms with binary ratings.

2 Collaborative filtering with forgetting mechanisms

2.1 Collaborative filtering with sliding windows

The simplest way to implement sliding windows [15,2,1] in a user-based or item-based CF system is to rebuild the similarity matrix using data of a fixed size sequence-based window holding the N latest sessions. Each time a new session s_i is available, the window moves one session forward discarding session s_{i-1-N} and including s_i. Then the new window is used as learning data to build a new similarity matrix.

Non-incremental algorithms can be easily adapted to use sliding windows, since they recalculate the whole similarity matrix each time a new session is available. The additional task is to move the window forward and use that window to rebuild the matrix. It is important to note that non-incremental algorithms process individual sessions several times – as many as the length of the window – which is not an ideal way to deal with data streams [5].

For incremental algorithms the adaptation is not so simple because the similarity matrix is not rebuilt from scratch [12,10]. The similarity values corresponding to the items in each new session are updated, while other values are kept. In order to implement a sliding windows approach in incremental algorithms it is necessary, at each update, to remove from the similarity matrix the information that was added by the oldest session in the window. This requires that every session is processed twice and, as in the non-incremental case, memory is required to hold session data for all sessions in the window.

2.2 Collaborative filtering with fading factors

Sliding windows provide an effective but abrupt way to forget older data. In many cases, however, past data is not necessarily obsolete, and can contain valuable information [5,8]. Fading factors [6] provide a mechanism to *gradually* forget past data.

Using fading factors with incremental user-based and item-based algorithms can be implemented by multiplying the similarity matrix by a factor $\alpha < 1$ before each update. This causes similarities between items or users to continuously decrease through time unless they are reinforced by recent session data. If the similarity reaches a lower threshold value, it can be assumed to be zero. This method is simple to implement and requires a single scan of each session.

Fading factors can also be implemented in non-incremental algorithms if all considered sessions are kept in memory in the same order by which they

arrived. A function of the session index can then be applied when rebuilding the similarity matrix, giving less weight to older sessions. However, this poses the same problem of using sliding windows: each session is processed several times, which is undesirable.

3 Algorithms

Four algorithms are presented: two using a non-incremental approach using sliding windows and two incremental algorithms using fading factors. For each approach, an item-based and a user-based version of the algorithms are studied.

All algorithms take usage data as input. This data is processed to build a similarity matrix S that contains the similarities between all pairs of users – in the user-based version – or items – in the item-based version. Similarity between a pair of users (or items) is calculated using a simplified version of the cosine measure for binary ratings [10]. If U and V are the sets of items that users u and v evaluated then the user-based similarity between u and v is given by:

$$sim(u, v) = \frac{\#(U \cap V)}{\sqrt{\#U} \times \sqrt{\#V}} \tag{1}$$

For the item-based version, if I and J are the sets of users that evaluated items i or j, the similarity between i and j is given by:

$$sim(i, j) = \frac{\#(I \cap J)}{\sqrt{\#I} \times \sqrt{\#J}} \tag{2}$$

3.1 Sliding Windows

The Sliding Window approach basically considers the w most recent sessions to build the similarity matrix S. To illustrate, consider a sequence of the first w user sessions $\{s_1, s_2, ..., s_w\}$, each containing a set of items rated by one user. First, S is built from data in sessions $\{s_1, ..., s_w\}$. Then, for each new session s_a, the model is rebuilt with data from sessions $\{s_{a-w}, ..., s_a\}$, creating a window of data that *slides* through data as it arrives.

User-based - UBSW This is the non-incremental user-based algorithm with sliding window (UBSW).

- For each new session s_a (by active user u_a):
  - (Re)calculate S using window $\{s_{a-w}, ..., s_a\}$
  - Determine the activation weight W_i of each item i not yet seen by u_a
  - Recommend the n items with the highest activation weight

The activation weight W_i for item i is obtained as follows:

$$W_i = \frac{\sum_{\text{users in neighborhood of } u_a \text{ that evaluated } i} S[u_a, .]}{\sum_{\text{users in neighborhood of } u_a}} \tag{3}$$

Item-based - IBSW This is the non-incremental item-based algorithm with sliding window (IBSW).

- For each new session s_a (by active user u_a):
  - (Re)calculate S using window $\{s_{a-w}, ..., s_a\}$
  - Determine the activation weight W_i of each item i not yet seen by u_a
  - Recommend the n items with the highest activation weight

The activation weight W_i for item i is obtained as follows:

$$W_i = \frac{\sum_{\text{items in neighborhood evaluated by } u_a} S[i,.]}{\sum_{\text{items in neighborhood}} S[i,.]} \tag{4}$$

3.2 Fading factors

The incremental algorithms using fading factors simply multiply the similarity matrix S by a factor $\alpha \leq 1$ before updating them with the active session data. With $\alpha < 1$, at each new session, older sessions become less important. With $\alpha = 1$, older data weight is maintained. In order to incrementally update S we also need save in memory a frequency matrix F with the number of items co-rated by each pair of users. The principal diagonal in F gives us the number of items evaluated by each user – in the user-based case – and the number of users that evaluated each item – in the item-based case.

Forgetting is obtained multiplying matrices S and F by a fading factor $\alpha < 1$. When using $\alpha = 1$ no forgetting occurs. It is important that *both* matrices S and F are multiplied by α. Because similarities in S are calculated directly from values in F, forgetting must be reflected also in F. Otherwise, every time rows and columns in S were updated, no forgetting would occur for them. Also, if only F is multiplied by α, non-updated rows and columns in S would not be forgotten.

User-based - UBFF This algorithm (UBFF) is a modified version – using binary ratings and fading factors – of the user-based incremental algorithm originally presented in [12]. A database D of user sessions and the lists of known users $Users$ and known items $Items$ are maintained.

- Build matrices S and F from training data
- For each new session s_a (by active user u_a):
  - Update D, S and F
  - Determine the activation weight W_i of each item i not yet seen by u_a
  - Recommend the n items with the highest activation weight

The activation weight W_i is obtained as in UBSW. The update of D, S and F is performed as follows:

- Let I be the set of items that u_a evaluated in the active session.

- Add u_a to the list of known users, $Users$
- Add the new session to D
- Multiply all values in S and F by a fading factor α
- If u_a is a new user, add a row and a column to F and to S
- Update the row/column of F corresponding to u_a, using the new D
- Add new items in I to the list of known items, $Items$
- Update the row/column of S corresponding to user u_a:

$$S_{u_a,\cdot} = \frac{F_{u_a,\cdot}}{F_{u_a,u_a} \times F_{\cdot,\cdot}} \tag{5}$$

Item-based - IBFF This modified version of the incremental item-based algorithm presented in [10] (IBFF).

- Build S and F from training data
- For each new session s_a (by active user u_a):
  - Determine the activation weight W_i of each item i not yet seen by u_a
  - Recommend the n items with the highest activation weight
  - Update S and F using a fading factor α

The activation weight W_i is obtained as in IBSW. The matrices S and F are updated as follows:

- Let I be the set of items that u_a evaluated in the active session.
- Add u_a to the list of known users, $Users$
- Add new items in I to the list of known items, $Items$
- Multiply all values in S and F by a fading factor α
- For each new item, add a row and column to F and to S
- For each pair of items in I, (i, j), update $F_{i,j}$ to $F_{i,j} + 1$
- For each item i_a in I update the corresponding row (column) of S:

$$S_{i_a,\cdot} = \frac{F_{i_a,\cdot}}{F_{i_a,i_a} \times F_{\cdot,\cdot}} \tag{6}$$

4 Evaluation and results

4.1 Datasets

Four datasets are used in the experiments. Table 1 describes each dataset. Sessions with less than 2 items were removed from all datasets. In all datasets, every user performs exactly one session, meaning that each session corresponds to a different user. Datasets ART1 and ART2 are synthesized datasets with an abrupt change. Both ART1 and ART2 consist of identical sessions with 4 items. These sessions contain the items $\{a, b, c, d\}$ at the beginning and then the item d is replaced by a new item e. This change occurs at session index 400 in ART1 and session 500 in ART2.

Table 1. Description of the datasets used.

Dataset	Domain	Users	Items	Transactions
ART1	Artificial	800	5	3200
ART2	Artificial	2000	5	10000
ELEARN	E-learning website	509	295	2646
MUSIC	Music social network	785	3121	9128

4.2 Evaluation methodology

In all experiments the *all-but-one* protocol [4] was used. First, the dataset is split in a training set and a test set. For IBSW and UBSW, the training set is considered to be the first window. For IBFF and UBFF, an initial training set is used to build the initial matrices S and F. An item is randomly hidden from each session in the test set. Then, recommendations made to each user are evaluated as a hit if the hidden item is among the recommended items. Because one single item from each session is hidden, recall is either 1 (hit) or 0 (no hit), and precision is obtained dividing recall by the number of recommended items. For that reason we present accuracy results using recall only.

We observe the evolution of recall values throughout each experimental run. Computational time spent building or updating the matrices is provided for real world datasets ELEARN and MUSIC.

Evaluation parameters The following parameters must be set:

- k - number of nearest neighbors considered to compute recommendations;
- n - number of items in each recommendation;
- w - window size in percentage of the dataset length (IBSW and UBSW);
- α - the fading factor for IBFF and UBFF.

In all experiments the values $k = 5$ and $n = 5$ are used. These values have consistent results while not unnecessarily consuming too many resources.

For the incremental algorithms a window with length $w = 0.1$ is used in order to initially train the algorithm. This relatively small training set is used in order to observe the evolution of the algorithms' performance for as many sessions as possible. Four values of α were tested. Values close to 1 were chosen so that the fading curve was not too abrupt. The non-forgetting factor $\alpha = 1$ was used as reference to measure and compare the impact of forgetting with different factors.

Software All algorithms were implemented using R version 2.11.0 with the package *spam* version 0.21-0 to handle sparse matrices.

4.3 Experiments with synthesized datasets

It is possible to observe in Fig. 1 that using synthesized datasets the algorithms using forgetting mechanisms tend to recover faster from abrupt changes than

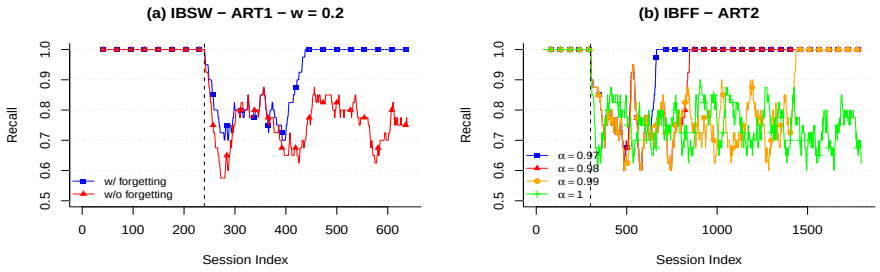

Fig. 1. Accuracy of IBSW with ART1 and IBFF with ART2. A moving average (n=40) is used to smoothen the lines. For IBSW w represents the percentage of sessions used as learning data. The vertical dashed line indicates the point where change occurs.

non-forgetting algorithms. As older data is forgotten, the initial conditions are not considered, providing a faster adaptation to new situations. Results for IBFF in Fig. 1 (b) illustrate the behavior of the algorithm using different fading factors: recovery is faster for lower values of α and slower for higher values of α. Without forgetting, none of the algorithms recover completely until the end of the test.

4.4 Non-incremental with sliding windows

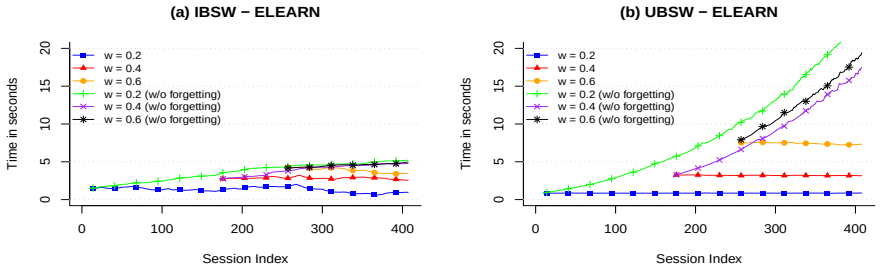

Fig. 2. Matrix rebuild time for IBSW and UBSW. A moving average (n=10) is used to smoothen the lines. w is the window size in percentage of total sessions.

In order to study the impact of forgetting in UBSW and IBSW, these algorithms are compared with their non-forgetting versions. These versions use *growing windows*, instead sliding windows. With a growing window, at each session s_i, all past sessions $\{s_1, ..., s_{i-1}\}$ are used to build S.

Figure 2 shows the time required by the algorithms to compute the similarity matrix S using dataset ELEARN. The algorithm without forgetting simply uses all known past data to build S. When using a sliding window, time is approximately constant for both algorithms, although some variability is noticeable in IBSW. When no forgetting occurs, time increases as more data is considered. This is an expected behavior, since the same number of sessions is used as learning data to build S at each step when using a fixed length window.

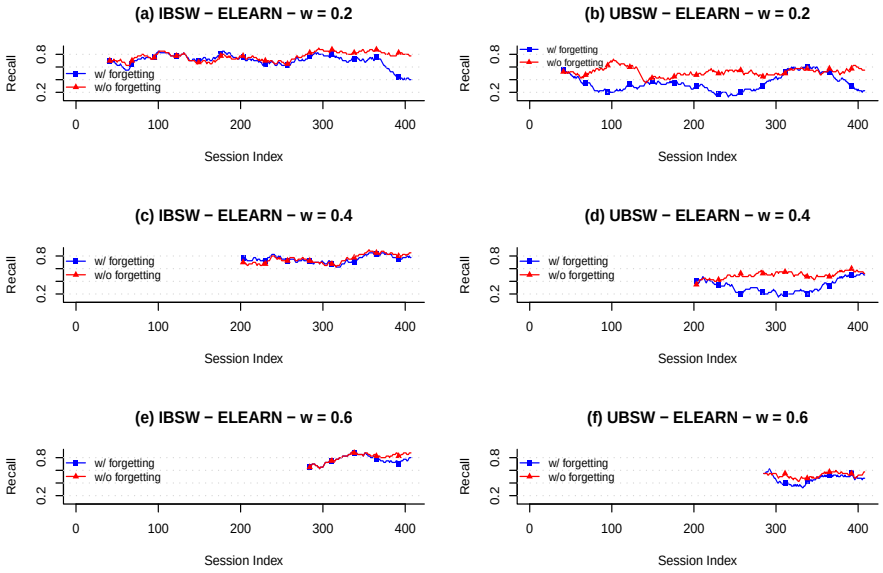

Fig. 3. Accuracy of IBFF and UBFF. A moving average (n=40) is used to smoothen the lines. A distinct graphic is shown for each window size (w). Without forgetting, w represents the percentage of initial sessions used as learning data.

When looking at the impact of sliding windows in accuracy (Fig. 3), it is clear that some loss occurs in most cases with forgetting algorithms, especially with UBSW. In some cases – for example, IBSW with $w = 0.4$ (Fig. 3 (c)) – accuracy remains very close with and without forgetting. Taking into consideration that time requirements are far lower for the forgetting versions, this loss – when it occurs – can be an acceptable trade-off for scalability.

4.5 Incremental with fading factors

Figure 4 shows the S update times for IBFF and UBFF using the two datasets – ELEARN and MUSIC. With the value $\alpha = 1$ no forgetting occurs, since S

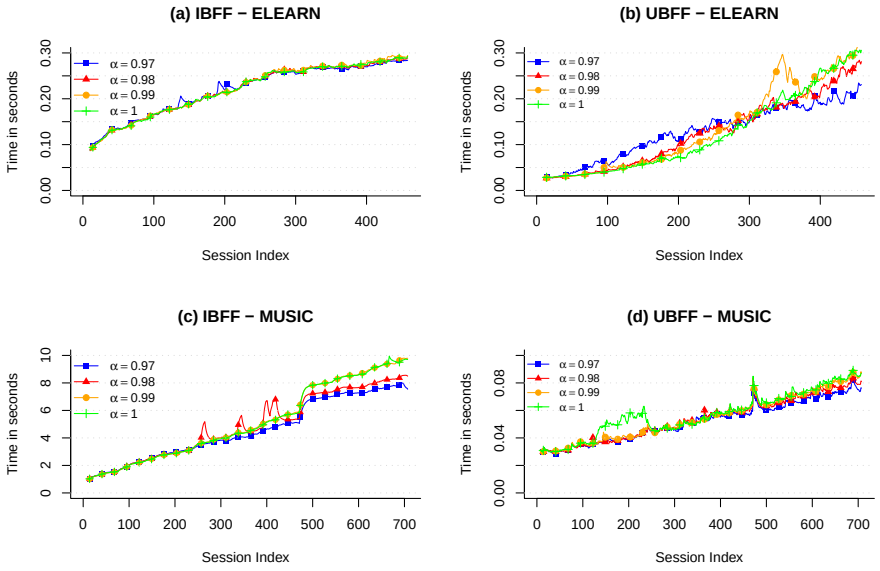

Fig. 4. Matrix update time for IBFF and UBFF. A moving average (n=10) is used to smoothen the lines.

is always multiplied by 1. One first observation is that update times for IBFF with the MUSIC dataset (Fig. 4 (a)) take up to 10 seconds, while in the other cases illustrated in Fig. 4, times are always less then half a second. This is explainable by the high number of items (3121) in this dataset. Using the same dataset, UBFF takes less than 0.1 seconds to update S. This illustrates the importance of choosing the right approach – user-based or item-based – according to the dataset. Observing the evolution of update times, there seems to be a linear increase as new sessions are considered. However, in all experiments with IBFF and UBFF, except IBFF with the ELEARN dataset, there is a slight time reduction using lower values for α. This happens because as older similarities become lower, the number of zeros in S tends to be greater with lower values of α. The package *spam* for R takes advantage of this increased sparsity by storing only values greater than zero, improving the efficiency of the algorithms.

When comparing the incremental algorithms' results with the ones obtained with the sliding window algorithms – using the ELEARN dataset –, it is clear that, in terms of time, incremental algorithms outperform non-incremental algorithms. This confirms results in [10]. In terms of accuracy, non-incremental algorithms tend to register greater losses when using small windows. Anyway, only windows $w > 0.4$ provide approximate accuracy to non-forgetting algorithms. In the incremental case, forgetting algorithms require less or equal time to their non-forgetting versions with minimal loss.

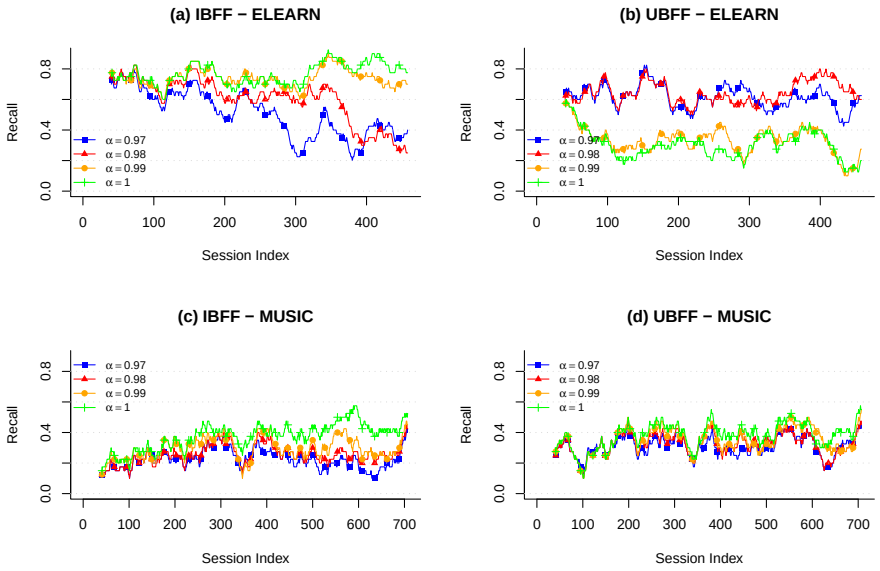

Fig. 5. Accuracy of IBFF and UBFF. A moving average (n=40) is used to smoothen the lines.

Looking at the predictive ability of the algorithms, in Fig. 5, it is possible to observe that lower factors tend to perform worse than higher factors. The exception is UBFF with the ELEARN dataset. In this case accuracy is significantly better with factors $\alpha = 0.97$ and $\alpha = 0.98$ than with higher factors. It is also noticeable that recall values for UBFF with the MUSIC dataset are very similar using several factors.

5 Conclusions

We have implemented and evaluated the impact of forgetting mechanisms in non-incremental and incremental collaborative filtering algorithms. Our results suggest that non-incremental algorithms that use sliding windows, when compared to their non-forgetting versions using a growing window, reduce computational requirements while not significantly reducing predictive ability. Results also suggest that incremental algorithms benefit from the use of fading factors, especially when using large datasets, although the fading factor approach has more subtle improvements in time requirements. It is also confirmed that incremental algorithms are more scalable than non-incremental algorithms.

In the future we intend to evaluate forgetting mechanisms in concept drift scenarios in order to make an in depth evaluation of the algorithms' reaction to

changes in data. Also, implementations in other languages and platforms will be tested to improve overall performance and allow larger scale experiments.

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Popa, L. (ed.) PODS. pp. 1–16. ACM (2002)
2. Babcock, B., Datar, M., Motwani, R.: Sampling from a moving window over streaming data. In: SODA. pp. 633–634 (2002)
3. Babu, S., Widom, J.: Continuous queries over data streams. SIGMOD Record 30(3), 109–120 (2001)
4. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: Cooper, G.F., Moral, S. (eds.) UAI. pp. 43–52. Morgan Kaufmann (1998)
5. Domingos, P., Hulten, G.: Catching up with the data: Research issues in mining data streams. In: DMKD (2001)
6. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: IV, J.F.E., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.) KDD. pp. 329–338. ACM (2009)
7. Hill, W.C., Stead, L., Rosenstein, M., Furnas, G.W.: Recommending and evaluating choices in a virtual community of use. In: CHI. pp. 194–201 (1995)
8. Koychev, I.: Gradual forgetting for adaptation to concept drift. In: Proceedings of ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning. pp. 101–106 (2000)
9. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing 7(1), 76–80 (2003)
10. Miranda, C., Jorge, A.M.: Item-based and user-based incremental collaborative filtering for web recommendations. In: Lopes, L.S., Lau, N., Mariano, P., Rocha, L.M. (eds.) EPIA. Lecture Notes in Computer Science, vol. 5816, pp. 673–684. Springer (2009)
11. Nasraoui, O., Cerwinske, J., Rojas, C., González, F.A.: Performance of recommendation systems in dynamic streaming environments. In: SDM. SIAM (2007)
12. Papagelis, M., Rousidis, I., Plexousakis, D., Theoharopoulos, E.: Incremental collaborative filtering for highly-scalable recommendation algorithms. In: Hacid, M.S., Murray, N.V., Ras, Z.W., Tsumoto, S. (eds.) ISMIS. Lecture Notes in Computer Science, vol. 3488, pp. 553–561. Springer (2005)
13. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: An open architecture for collaborative filtering of netnews. In: CSCW. pp. 175–186 (1994)
14. Shardanand, U., Maes, P.: Social information filtering: Algorithms for automating "word of mouth". In: CHI. pp. 210–217 (1995)
15. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning 23(1), 69–101 (1996)