Noname manuscript No.
(will be inserted by the editor)

Forgetting Techniques for Stream-based Matrix Factorization in Recommender Systems

Pawel Matuszyk · João Vinagre · Myra Spiliopoulou · Alípio Mário Jorge · João Gama

Received: date / Accepted: date

Abstract Forgetting is often considered a malfunction of intelligent agents, however, in a changing world forgetting has an essential advantage. It provides means of adaptation to changes by removing effects of obsolete (not necessarily old) information from models. This also applies to intelligent systems, such as recommender systems, which learn users' preferences and predict future items of interest.

In this work we present unsupervised forgetting techniques that make recommender systems adapt to changes of users' preferences over time. We propose eleven techniques that select obsolete information and three algorithms that enforce the forgetting in different ways. In our evaluation on real world datasets we show that forgetting obsolete information **significantly** improves predictive power of recommender systems.

Keywords Recommender Systems · Forgetting Techniques · Matrix Factorization · Data Stream Mining · Machine Learning

1 Introduction

Recommender systems learn users' preferences based on past data and predict future items of interest. A challenge in learning users' preferences lies in their

P. Matuszyk · M. Spiliopoulou
Knowledge Management and Discovery Lab
Otto von Guericke University,
Magdeburg, Germany
E-mail: {pawel.matuszyk | myra}@iti.cs.uni-magdeburg.de

J. Vinagre · A. M. Jorge
FCUP - Universidade do Porto,
LIAAD - INESC TEC,
Porto, Portugal
E-mail: jnsilva@inesctec.pt, amjorge@fc.up.pt

J. Gama
FEP - Universidade do Porto,
LIAAD - INESC TEC,
Porto, Portugal
E-mail: jgama@fep.up.pt

volatility. Preferences are not static in their nature - they undergo changes, i.e. concept drift, from the perspective of a predictive algorithm.

Adaptation to concept drift is an important aspect of every learning system that is a part of a changing environment. The fact that the environment of a recommender system (e.g. users' preferences) changes over time has been shown in past research [18,17,37,20,19]. However, there are two complementary ways of adapting to changes:

- incorporating new information into a model
- forgetting the obsolete information

The first option has been subject to much of recent research e.g. in the work on incremental matrix factorization [36,26,38,32,22]. Incorporating new information into models is used by stream-based methods. They work with the realistic assumption that users' feedback arrives one after another in a stream of ratings. Their biggest advantage is that they can incorporate new information in real time without discarding the entire model.

However, we argue that incorporating new information alone is not sufficient as an adaptation mechanism. Since users' preferences change over time, some of the users' feedback does not reflect the current preferences any more. Considering this obsolete feedback would distort users' profiles and negatively impact the predictive power of our models.

Therefore, next to incorporating new information, in this work we investigate the second way of adaptation - forgetting methods. They forget in a selective and controlled way to remove the impact of the obsolete information. These methods encompass selection strategies for finding the obsolete information (cf. Sec. 3) and algorithms implementing forgetting of selected information i.e. removing the effect of a rating from a model (cf. Sec. 4). Please note that an obsolete information does not need to be old. For instance, information about a user, who buys a present for someone else, is obsolete to this user's profile, even if it is new.

Our goal is *not* to show that our algorithm is better than a given other algorithm. Instead, we aim to show that selective forgetting of information improves the quality of recommendations as compared to no forgetting. Therefore, "no forgetting" is our main comparison baseline, next to other forgetting strategies existing in the literature already (cf. Sec. 2). In our experiments with explicit rating feedback and also with positive-only feedback, we show that **forgetting significantly improves predictive performance** of stream-based recommender systems, as compared to the first way of adaptation mechanism alone (incorporating new information with no forgetting).

As a representative of stream-based recommender systems we use an incremental matrix factorization algorithm BRISMF [36], which has the first adaptation mechanism already. Matrix factorization algorithms are considered state-of-the-art in recommender systems. While they belong to the class of collaborative filtering algorithms, they are not based on the notion of neighbourhood and similarity. We extended the BRISMF algorithm by our forgetting techniques. Since matrix factorization in recommender systems is an active research field, there are numerous recommendation algorithms based on it. Those algorithms encompass several extensions, e.g. for implicit feedback [13], time aspects (different than forgetting) [17,18], semi-supervised learning [26], active learning [16], etc. Therefore, it is not possible to test all of them in combination with our forgetting strategies. Thus, we use a possibly generic representative of the matrix factorization algorithms. Using a very specific extension of a matrix factorization algorithm would limit the generality of our results. By using a generic algorithm, such as BRISMF, whose internal working is common to nearly all matrix factorization algorithms in recommender systems,we ensure that our methods and results can be transferred to other algorithms.

Forgetting obsolete information has an additional advantage of allowing users to decide which information should be forgotten. From the perspective of data privacy this is highly desirable. A user could decide to exclude an item from her/his recommendation profile. This **could not** be done by a simple deletion of the item from the user record, since the impact of this item on the user profile would still reside in the user's preference model. Even though shared parameters of matrix factorization models are adjusted when incremental learning is performed and the relative importance of old information decreases as new information comes in, the impact of obsolete information still remains in the model. Our forgetting techniques solve this problem by discarding the impact of a rating from preference models.

In summary, our contributions are as follows:

– eleven forgetting strategies of two types and three alternative algorithms implementing them
– an evaluation protocol, which incorporates significance testing and an incremental measure of recall
– insights on forgetting with positive-only feedback and on rating feedback
– **significant improvement** of predictive power of recommender systems on seven out of eight datasets

This publication extends our work [25, 27], where we presented seven forgetting strategies and in [27] we also used the incremental recall measure. Here, in addition to our previous work, we propose four new forgetting strategies. We also formalize them and divide them into two categories with respect to their behaviour. We also propose two new algorithms that enforce forgetting on a stream and compare them with the remaining one from our previous work. In this work we also focus on the complexity of the algorithms. One of the algorithms is an approximative algorithm that shows lower complexity, while maintaining a similar level of quality. Furthermore, we introduce a new evaluation protocol that includes significance testing on streams and also considers runtime of algorithms. The proposed algorithms have also been extended by the ability to add new dimensions to the matrix. Due to new algorithms and evaluation protocol we executed more than 1040 new experiments.

The structure of this paper is as follows. In the next section we discuss related work. Sec. 3 explains our forgetting strategies for selecting obsolete information. In Sec. 4 we describe three algorithms that enforce forgetting of selected instances in different ways. Evaluation protocol and measures are mentioned in Sec. 5. In Sec. 6 we report our results. Finally, we conclude our work and discuss remaining issues in Sec. 7.

2 Related Work

In real world systems, data used by recommender systems has all the characteristics of a data stream. Data streams arrive on-line, at unpredictable order and rate, and they are potentially unbounded [1]. Once a data element is processed it must be discarded or archived, and subsequent access to past data gets increasingly expensive. Algorithms that learn from data streams should process data in one pass, at least as fast as data elements arrive, and memory consumption should to be independent from the number of data points [8]. Although recommendation is seldom approached as a data stream mining problem, the following contributions must be pointed out. Two of them apply generic data stream mining algorithms in recommender systems. In [21], Li et al. propose an approach to drifting preferences of individual users using the CVFDT algorithm [15]. This is a popular classification algorithm for high speed data streams. The CVFDT algorithm is used to build a decision tree for each item in the dataset, given the ratings of other highly correlated items. The ratings given by users to these correlated items are used to predict the ratings for the target item. The mechanics of CVFDT provides automatic adjustment to drifts in user interests, avoiding accuracy degradation. In [29] Nasraoui et al. use the TECHNO-STREAMS stream clustering algorithm [30], using a sliding window through user sessions to compute a clustering-based recommendation model.

Another possible approach is to adapt existing collaborative filtering algorithms to learn from data streams. One key aspect of most data stream mining algorithms is that they build models incrementally. Neighborhood-based incremental learning from ratings data for recommendation has been introduced in [31], and adapted for positive-only feedback in [28]. These two contributions use incremental cache structures to incrementally update similarities between users or between items as new ratings become available. Incremental matrix factorization for recommendation has been first studied in [33] using the fold-in method for Singular Value Decomposition (SVD) [2]. This is a method used in information retrieval to add new documents to large document-term matrices. However, because it violates the orthogonality required to maintain an accurate SVD, the decomposition still needs to be recomputed periodically from scratch. State-of-the-art factorization for recommender systems relies on optimization methods, mostly Stochastic Gradient Descent (SGD) [32]. Incremental versions of SGD based methods have been studied in [36, 22] for ratings and in [38] for positive-only data.

Forgetting past data is a model maintenance strategy frequently used in data stream mining. The underlying assumption is that some data points are more representative than others of the concept(s) captured by the algorithm. In recommender systems, forgetting is introduced by Koychev in a content-based algorithm [19]. The technique assigns higher weights to recent observations, forgetting past data gradually. This way, the algorithm is able to recover faster from changes in the data, such as user preference drifts. A similar approach is used with neighborhood-based collaborative filtering by Ding and Li in [7]: the rating prediction for an item is calculated using a time decay function over the ratings given to similar items. In practice, recently rated items have a higher weight than those rated longer ago. In [23], Liu et al. use the same strategy, additionally introducing another time decay function in the similarity computation, causing items rated closely together in time to be more similar than those rated far apart in time. Another contribution is made by Vinagre and Jorge in [37]. The authors use two different forgetting strategies with neighbourhood-based algorithms for positive-only data. Forgetting is achieved either abruptly, using a fixed-size sliding window over data and repeatedly training the algorithm with data in the window, or gradually, using a decay function in the similarity calculation causing older items to be less relevant. However, these techniques are only effective in the presence of sudden changes with a high magnitude global impact, i.e. for all users, and do not account for more subtle and gradual changes that occur on the individual users' level. In our experiments we also compare to those strategies.

In [18] Koren modelled users in a dynamic way, i.e. he tried to capture changes in latent user vectors by using linear regression and splines. However, his method was not able to recognize and forget obsolete information. Additionally, unlike our method, Koren's method is not able to work on streams of ratings. In 2014 Sun et al. introduced collaborative Kalman filtering [35]. This method also attempts to model changes in users' preferences, but similarly to Koren's method, it is not designed for streams of data and it does not forget any information. Chua et al. modelled temporal adoptions using dynamic matrix factorization [5]. Their approach is not stream-based, but works on chunks of data and combines models of single chunks using a linear dynamic system. Similarly to Sun et al., in [5] the authors also used Kalman filters.

Forgetting for incremental matrix factorization had not been studied before the work by Matuszyk and Spiliopoulou in [25] and our work in [27]. For readers unfamiliar with the concept of matrix factorization in recommender systems we refer to the seminal work by Takács et al. [36] and to the work by Koren et al. [17]. Due to space constraints we cannot explain those concepts here.

3 Forgetting Strategies

In this section we discuss how to select information that should be forgotten. We introduce the term of **forgetting strategies** as methods for selecting this obsolete information. They work in an unsupervised way, since there is no ground truth determining when a rating becomes obsolete.

We divide our strategies into two categories based on their output. The first category is **rating-based forgetting** (cf. Sec. 3.1). As the name suggests, all strategies of this type take a user-specific list of ratings as input and decide which ratings should be forgotten. Data returned by those strategies is a filtered list of ratings of a user. Forgetting strategies of the second type are based on latent factors from the matrix factorization model. The **latent-factor-based strategies** (cf. Sec. 3.2) modify a latent factor of a user or of an item in a way that lowers the impact of past ratings.

In total we present 11 forgetting strategies. The description of strategies in Secs. 3.1.1 - 3.1.4 and 3.2.1 - 3.2.3 comes from our work in [27].

3.1 Rating-based Forgetting

This category of forgetting strategies operates on sets of users' ratings. We define $R(u)$ as a set of ratings provided by the user u. Formally, a rating-based forgetting strategy is a function $f(R(u))$:

$$f : R(u) \rightarrow R(u)' \tag{1}$$

where $R(u)' \subseteq R(u)$. Furthermore, for each user we define a threshold of n ratings a user must have, before forgetting is applied. This threshold ensures that no new users are affected by forgetting and that no users or items are forgotten completely. Therefore, it is not possible that a badly chosen forgetting strategy forgets everything and is unable to make recommendations. In our experiments we use $n = 5$.

3.1.1 Sensitivity-based Forgetting

As the name suggests this strategy is based on sensitivity analysis. We analyse how much a latent user vector p_u changes after including a new rating of this user $r_{u,i}$ into a model. A latent user vector should always reflect user's preferences. With a new rating $r_{u,i}$ we gain more information about a user and we can adjust the model of his/her preferences (the latent vector p_u) accordingly. If the new rating is consistent with the preferences that we know so far of this particular user, the change of the latent vector should be minimal. However, if we observe that the latent vector of this user changed dramatically after training on the new rating, then this indicates that the new rating is not consistent with the past preferences of this user. Thus, we forget this rating, i.e. we do not update the user's latent vector using this rating, since it does not fit to the rest of user's preferences.

In real life recommenders this occurs frequently, e.g. when users buy items for other people as presents, or when multiple persons share one account. Those items are outliers with respect to preferences observed so far. Learning model based on those outliers distorts the image of user's preferences.

In order to identify those outlier ratings, we first store the latent user vector at time point t, denoted hereafter as p_u^t. Then, we simulate our incremental training on the new rating $r_{u,i}$ without making the changes permanent. We obtain an updated latent user vector p_u^{t+1}. Our next step is then calculating a squared difference between those two latent vectors using the following formula:

$$\Delta_{p_u} = \sum_{i=0}^{k} (p_{u,i}^{t+1} - p_{u,i}^{t})^2 \tag{2}$$

Furthermore, for each user we store and update incrementally at each time point the standard deviation of the squared difference. The notation $\overline{x}$ stands for a mean value of vector x and $SD(x)$ is standard deviation of this vector. If the following inequality is true, then it indicates that the new rating $r_{u,i}$ is an outlier:

$$\Delta_{p_u} > \overline{\Delta_{p_u}} + \alpha \cdot SD(\Delta_{p_u}) \tag{3}$$

Where α is a parameter that controls the sensitivity of the forgetting strategy. It is specific for every dataset and has to be determined experimentally.

3.1.2 Global Sensitivity-based Forgetting

Similarly to the previous strategy, Global Sensitivity-based Forgetting is also based on sensitivity analysis. However, in contrast to Sensitivity-based Forgetting, this strategy does not store the standard deviation of squared differences of latent vectors separately for each user but rather globally. Formally, it means that the inequality (3) needs to be changed into:

$$\Delta_{p_u} > \overline{\Delta} + \alpha \cdot SD(\Delta) \tag{4}$$

Where Δ is a squared difference of latent user vectors before and after including a new rating for all users.

3.1.3 Last N Retention

To the category of rating-based forgetting we also count the Last N Retention strategy by Matuszyk and Spiliopoulou [25]. This forgetting strategy uses sliding window, which often finds an application in stream mining and adaptive algorithms. Here however, a sliding window is defined for each user separately. LastNRetention means that the last N ratings in a stream of a user are retained and all remaining ratings are forgotten. If a user has fewer ratings than N, then forgetting will not be applied onto this user.

3.1.4 Recent N Retention

This forgetting strategy is also based on a user-specific sliding window [25]. However, here the size of a window is not defined in terms of ratings, but in terms of time. Therefore, unlike *last N retention* it considers the time that has passed between providing ratings. If $N = 3days$, then only ratings from the last three days are retained and all the remaining ones will be forgotten. This strategy can be also set up to use only ratings from e.g. the last session. In contrast to the Last N Retention, this strategy allows to define a lifetime of a rating, which is especially beneficial in highly dynamic environments, such as news recommendations.

3.1.5 Recall-based Change Detection

In many applications a change of preferences takes place gradually. However, there are applications where a change can happen abruptly and without prior indication. For instance, a person who becomes a parent is likely to start preferring items suitable for children. A recommender system that adapts gradually is not able to capture this abrupt change. Consequently, the importance of these new preferences will be underestimated for a long period of time. Therefore, we propose a forgetting strategy that detects a change in preferences and forgets all ratings from before the change. This is equivalent to resetting a single profile of the affected user without discarding the entire model.

In recall-based change detection, incremental recall (or a different incremental quality measure) for each user is monitored. A high drop in the quality measure indicates a change of preferences.

In more quantitative terms, let incremental recall for user u at timepoint t be denoted as $incrRecall_u^t$. A change is detected, if the following inequality is true:

$$incrRecall_u^t < \overline{incrRecall_u} - \alpha \cdot SD(incrRecall_u) \tag{5}$$

i.e. when the current recall of a user is lower than this user's mean recall by at least $\alpha \cdot SD(incrRecall_u)$. The parameter α controls the sensitivity of the detector.

3.1.6 Sensitivity-based Change Detection

Similarly to the previous forgetting strategy, this one is also a change detector. We assume that, if incorporating a new rating from the stream provided by a user changes the underlying user's model dramatically, then this new rating does not fit into the old model. Consequently, we conclude that the user's preferences have changed.

To express the relative change of a user's model over time we use the notion of local model sensitivity and the formula 2. Let $\Delta_{p_u}^{t}$ be the change of user's model after incorporating a new rating at timepoint t. A change is detected, if the following inequality holds:

$$\Delta_{p_u}^{t} > \overline{\Delta_{p_u}} + \alpha \cdot SD(\Delta_{p_u}) \tag{6}$$

i.e. when the change to the current model is higher than the standard deviation of all changes of this user multiplied with a control parameter α.

This strategy is similar to *Sensitivity-based Forgetting*, because it also uses the idea of an outlier-rating that does not fit into the learnt model. However, the key difference is that, here, we conclude that it is the model learnt so far that should be forgotten due to a concept shift and not the single outlier-rating.

3.2 Latent Factor Forgetting

Latent factor forgetting is the second type of our forgetting strategies. Unlike rating-based forgetting, this type of strategies operates directly on preference models and not on the ratings. In matrix factorization, preference models have form of latent user vectors, denoted as p_u, or latent item vectors, denoted as q_i. This type of forgetting is triggered when a new rating of the corresponding user u towards item i has been observed in a stream.

Formally, a latent factor forgetting strategy is a linear transformation of a latent user vector (cf. Eq. 7) or of a latent item vector (cf. Eq. 8):

$$\overrightarrow{p_u^{t+1}} = \gamma \cdot \overrightarrow{p_u^{t}} + \beta \tag{7}$$

$$\overrightarrow{q_i^{t+1}} = \gamma \cdot \overrightarrow{q_i^{t}} + \beta \tag{8}$$

The parameters γ and β are dictated by the strategies described in the following subsections. Since those strategies transform latent vectors, it is also not possible that users or items are forgotten completely. This could happen only if γ and β are equal to 0, which we do not allow in the following strategies.

3.2.1 Forgetting Unpopular Items

In this forgetting strategy unpopular items are penalized. Latent item vectors are multiplied with a value that is lower for unpopular items to decrease their importance in the prediction of interesting items. Formally, this strategy is expressed by the following formula:

$$\overrightarrow{q_i^{t+1}} = (-\alpha^{-|R(i)|} + 1) \cdot \overrightarrow{q_i^{t}} \tag{9}$$

$R(i)$ is the set of ratings for item i. α is a parameter that controls, how much the latent item vector is penalized. $(-\alpha^{-|R(i)|}+1)$ is an exponential function that takes low values for items with few ratings (for α values > 1). Additionally, this function has an advantage of being limited to value range of $[0, 1)$, for $\alpha > 1$.

3.2.2 User Factor Fading

User Factor Fading is similar to using fading factors in stream mining. In this strategy latent user factors are multiplied by a constant $\alpha \in (0, 1]$

$$\overrightarrow{p_u^{t+1}} = \alpha \cdot \overrightarrow{p_u^{t}} \tag{10}$$

The lower is this constant, the higher is the effect of forgetting and the less important are the past user's preferences.

3.2.3 SD-based User Factor Fading

As in user factor fading, this strategy alters latent user vectors. Hoverer, the multiplier here is not a constant but it depends on the volatility of user's factors. The assumption behind this strategy is that highly volatile latent vectors (the ones that change a lot), are unstable. Therefore, forgetting should be increased until the latent vectors stabilize.

Similarly to sensitivity-based forgetting, in this strategy we measure how much the latent user factor changed compared to the previous time point. We calculate again the squared difference between p_u^{t+1} and p_u^t and denote it as Δ_{p_u} (cf. Equation 2). Subsequently, we use the standard deviation of Δ_{p_u} in an exponential function:

$$\overrightarrow{p_u^{t+1}} = \alpha^{-SD(\Delta_{p_u})} \cdot \overrightarrow{p_u^{t}} \tag{11}$$

For high standard deviation of Δ_{p_u} the exponential function takes low values, penalizing unstable user vectors. The parameter α controls the extent of the penalty. For $\alpha > 1$ this function always takes values in the range $[0, 1)$.

3.2.4 Recall-based User Factor Fading

As in the previous strategy, users' latent vectors are also multiplied with a weight, which, here is based on user-specific recall. The idea is as follows: if a prediction model performs poorly for a user in terms of incremental recall, this forgetting strategy assumes that preferences of this user are changing. Therefore, forgetting should be amplified. In contrast, if the performance of the model is high, then forgetting is suppressed, so that a stable and well functioning model is not altered by the forgetting. To model this strategy we use the following formula:

$$\overrightarrow{p_u^{t+1}} = (-\alpha^{-incrRecall_u^t} + 1) \cdot \overrightarrow{p_u^{t}} \tag{12}$$

The exponential term $-\alpha^{-incrRecall_u^t} + 1$ takes high values for high recall values (for $\alpha > 1$). Therefore, if a model performs well, this term is close to 1, which makes the effect of the forgetting strategy low. Otherwise, a lower value of the exponential function increases the forgetting rate.

3.2.5 Forgetting Popular Items

This strategy is opposite to the one presented in Sec. 3.2.1. Here, popular items are penalized, so that their impact on the model is reduced. Forgetting popular items can be used to decrease the impact of mainstream products onto a preference model.

To achieve that, an exponential function is used that decreases the multiplier of the latent item vector (for $\alpha > 1$), when an item was rated by many users.

$$\overrightarrow{q_i^{t+1}} = \alpha^{-|R(i)|} \cdot \overrightarrow{q_i^{t}} \tag{13}$$

4 Enforcing Forgetting on a Stream of Ratings

Thus far, we have described how to select information that should be forgotten. In this section we discuss how the forgetting is implemented, i.e. how an impact of selected information can be removed from a model. We propose three alternative implementations. In those implementations we follow the stream setting. In this setting a model is updated incrementally. With every change of a model (adding or forgetting information) a preference model is updated locally. In matrix factorization algorithms adding or forgetting a rating eventually affects many latent features, as its influence can propagate onto other items and users. To account for that, the preference model would have to be re-learnt from the scratch every time a rating is added or forgotten. This, however, is not feasible due to excessive computation time and real-time requirements that are often imposed onto stream learning algorithms. Therefore, to make stream-mining algorithms applicable in real-world scenarios, it is a common practice to update only parts of a model.

This strategy of local updates is applied not only in recommender systems, as e.g. in [36], but also in many other stream mining algorithms. Hoeffding trees, such as CVFDT [14], and stream-based clustering algorithms, such as [3,30], apply the same strategy of partial updates when new data instances are observed (cf. also a recent survey on stream-mining [9]). CVFDT, for example, does not relearn the entire tree, when a new data instance is observed, but it adjusts a model only locally. We use the same strategy in our forgetting methods. Only the latent features that are directly affected, are updated locally without relearning the entire model. First however, we describe our baseline algorithm that does not use forgetting techniques. We use this algorithm in all our implementations and also in experiments as a comparison baseline.

4.1 Baseline Algorithm

We extend a state-of-the-art matrix factorization algorithm BRISMF (biased regularized incremental simultaneous matrix factorization) by Takács et al. [36]. Parts of the baseline description come from our previous work [27].

BRISMF is a batch method, however, Tákacs et al. also proposed an algorithm for retraining latent users features (cf. Alg. 2. in [36]) that can be used as a stream-based algorithm. Latent user features are updated as new observations arrive in a stream, ideally in real time. Since item features are not updated online (as dictated by Tákacs et al. [36]), the method requires an initial phase, in which the latent item features are trained. We adopted this procedure also in our extension of this algorithm.

4.1.1 Initial Phase

Similarly to [25] and [36], in this phase we use Stochastic Gradient Descent (SGD) to decompose the user-item-rating matrix R into two matrices of latent factors $R \approx P \cdot Q$, where P is the latent user matrix and Q the latent item matrix with elements $p_{u,k}$ and $q_{i,k}$ respectively. k is an index of the corresponding latent factor. In every iteration of SGD we use the following formulas to update latent factors [36]:

$$p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k}) \tag{14}$$

$$q_{i,k} \leftarrow q_{i,k} + \eta \cdot (predictionError \cdot p_{u,k} - \lambda \cdot q_{i,k}) \tag{15}$$

Where η is a learn rate of the SGD and λ is a regularization parameter that prevents SGD from overfitting. k stands for the number of latent dimensions.

4.1.2 Online Phase

Once the initial phase is finished the online phase is started. Here, evaluation and prediction tasks run in parallel (cf. Sec. 5 for more information on the evaluation setting). With every new rating arriving in a data stream the corresponding latent user factor is updated.

The pseudo code showing our extended incremental training approach is presented in Alg. 1. The algorithm takes as input the original rating matrix R, two factor matrices P and Q learnt in the initial phase and the aforementioned parameters required by the SGD. $r_{u,i}$ is most recent rating in a data stream that the algorithm uses to update the preference model, i.e. the latent user vector $\overrightarrow{p_u}$. The update is carried out by performing further iterations of SGD using the new rating. *optimalNumberOfEpochs* is determined during the initial phase. The complexity of updating a model with one rating is $O(E)$, where $E = optimalNumberOfEpochs$.

In the pseudo code two of our extensions to the BRISMF algorithm are visible (cf. line 3 in Alg. 1 and Alg. 2):

– extending dimensions of the matrix
– different initialization of new dimensions

Extending dimensions of the matrix is an essential feature for stream-based algorithms. In a stream new users and items are introduced into the system frequently. Those users and items do not appear in the user/item matrix from the training phase of the algorithm. In order to make predictions for those users we extend the original matrix by new rows or columns. In experiments with offline datasets this extension allows to reduce the number of missing predictions considerably.

Our second extension regards initialization of new dimensions in the matrix. According to Takács et al. [36] latent matrices are initialized with small random

Algorithm 1 Incremental Learning - Baseline

```
Input: $r_{u,i}, R, P, Q$
Parameters: $\eta, k, \lambda$, $optimalNumberOfEpochs$
1: $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
2: $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
3: Extend and initialize new dimensions($\overrightarrow{p_u}, \overrightarrow{q_i}, P, Q, R$)
4: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
5: evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
6: R.storeRating($r_{u,i}$)
7: $epoch = 0$
8: while $epoch < optimalNumberOfEpochs$ do
9:     epoch++
10:    $predictionError = r_{u,i} - \overrightarrow{p_u} \cdot \overrightarrow{q_i}$
11:    for all latent dimensions $k \neq 1$ in $\overrightarrow{p_u}$ do
12:        $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
13:    end for
14: end while
```

values centered around zero. In batch algorithms this type of initialization is not problematic, since after the initialization those values are overridden in multiple iterations of gradient descent that scans several times over a training set. In a data stream, however, scanning a training set multiple times is not possible. Ideally, stream-based algorithms are one-pass algorithms, which scan all data only once. Therefore, we adjust the initialization of new dimensions in the matrix to be more meaningful from the start. We initialize new latent dimensions with average values over all latent users/items. After the initial phase of the algorithm, the latent matrices already contain meaningful values, therefore, their averages are different from just random values. Additionally, we add a uniformly distributed random component Z as in [36] from a small range of $(-0.02, 0.02)$. Thus, a new user vector p_{new_user} and new item vector q_{new_item} are initialized as follows:

$$p_{new\_user,k} = \frac{1}{|U|} \cdot \sum_{i=1}^{|U|} p_{i,k} + Z \sim \mathcal{U}(-0.02, 0.02) \tag{16}$$

$$q_{new\_item,k} = \frac{1}{|I|} \cdot \sum_{i=1}^{|I|} q_{i,k} + Z \sim \mathcal{U}(-0.02, 0.02) \tag{17}$$

Where k is the index of a latent dimension, U is a set of all users and I a set of all items. Accordingly, new user and items are treated as average users/items until there is enough learning examples to make them more specialized. The pseudo code presenting the initialization of new dimensions is shown in Alg. 2.

Those two extensions are necessary adaptations of the BRISMF algorithm to the streaming scenario. However, our central contributions are the extensions to follow in the next subsections together with the forgetting strategies discussed before. Our goal is to investigate their impact.

4.2 Matrix factorization for Rating-based Forgetting

In this subsection we present incremental matrix factorization with rating-based forgetting in the online phase. This implementation extends the baseline algorithm

Algorithm 2 Extend and initialize new dimensions

```
Input: $\overrightarrow{p_u}, \overrightarrow{q_i}, P, Q, R$
 1: if $\overrightarrow{p_u}$ == null then // if $u$ is a new user
 2:     R.addNewUserDimension($u$)
 3:     $\forall k : p_{u,k} = \frac{1}{|U|} \cdot \sum_{i=1}^{|U|} p_{i,k} + Z \sim \mathcal{U}(-0.02, 0.02)$
 4:     P.addLatentUserVector($p_u$)
 5: end if
 6: if $\overrightarrow{q_i}$ == null then // if $i$ is a new item
 7:     R.addNewItemDimension($i$)
 8:     $\forall k : q_{i,k} = \frac{1}{|I|} \cdot \sum_{i=1}^{|I|} q_{i,k} + Z \sim \mathcal{U}(-0.02, 0.02)$
 9:     Q.addLatentItemVector($q_i$)
10: end if
```

from the previous subsection. Therefore, all differences in the performance between this algorithm and the baseline are due to our forgetting techniques. Those forgetting strategies can be also applied to different incremental matrix factorization methods analogously.

Alg. 3 shows pseudo code of our rating-based forgetting. First, we introduce a new notation, where $\overrightarrow{r}_{u*}$ is a vector of all ratings of user u. In line 6 such a user vector is retrieved from the matrix and in line 8 a rating-based forgetting strategy is called. Consequently, all rating from the user's vector that have been deemed obsolete by the forgetting strategy are removed. In the following line, the corresponding row in the matrix R is overridden with a new user vector that does not contain the obsolete ratings. This ensures that the removed ratings are not retrieved from the matrix R in the the next iteration of the algorithm. Subsequently, the user's latent vector $\overrightarrow{p_u}$ is retrained using all remaining ratings from the $\overrightarrow{r}_{u*}$ rating vector.

This procedure introduces a further loop into the algorithm, due to which its complexity of a model update rises to $O(E \cdot ||\overrightarrow{r}_{u*}||)$.

4.3 Matrix factorization for Latent Factor Forgetting

In Alg. 4 we present an implementation that uses latent factor forgetting strategies. In lines 6 and 7 a forgetting strategy is invoked to modify the corresponding latent user or item vectors.

Other than that, no further changes compared to the baseline algorithm are necessary. Latent factor forgetting does not require retraining on past ratings, therefore the complexity is here again at $O(E)$.

4.4 Approximation of Rating-based Forgetting

Since rating-based forgetting increased the complexity of updating a preference model to $O(E \cdot ||\overrightarrow{r}_{u*}||)$, we propose a faster approximative method of implementing this type of forgetting.

The implementation in Alg. 5 eliminates the need for the additional loop for retraining of the user latent vector on past ratings. Instead of this loop impact

Algorithm 3 Incremental Learning with Rating-based Forgetting

```
Input: $r_{u,i}, R, P, Q$
Parameters: $\eta, k, \lambda$, $optimalNumberOfEpochs$
 1: $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
 2: $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
 3: Extend and initialize new dimensions($\overrightarrow{p_u}, \overrightarrow{q_i}, P, Q, R$)
 4: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
 5: evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
 6: $\overrightarrow{r}_{u*} \leftarrow$ getUserRatings($R, u$)
 7: ($\overrightarrow{r}_{u*}$).addRating($r_{u,i}$)
 8: **rating-basedForgetting**($\overrightarrow{r}_{u*}$) **//obsolete ratings removed**
 9: $R$.overrideUserVector($\overrightarrow{r}_{u*}$) //obsolete ratings also removed from $R$
10: $epoch = 0$
11: while $epoch < optimalNumberOfEpochs$ do
12:     epoch++
13:     for all $r_{u,i}$ in $\overrightarrow{r}_{u*}$ do
14:         $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
15:         $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
16:         $predictionError = r_{u,i} - \overrightarrow{p_u} \cdot \overrightarrow{q_i}$
17:         for all latent dimensions $k \neq 1$ in $\overrightarrow{p_u}$ do
18:             $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
19:         end for
20:     end for
21: end while
```

Algorithm 4 Incremental Learning with Latent Factor Forgetting

```
Input: $r_{u,i}, R, P, Q$
Parameters: $\eta, k, \lambda$, $optimalNumberOfEpochs$
 1: $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
 2: $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
 3: Extend and initialize new dimensions($\overrightarrow{p_u}, \overrightarrow{q_i}, P, Q, R$)
 4: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
 5: evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
 6: $\overrightarrow{p_u} \leftarrow$ **latentForgetting**($\overrightarrow{p_u}$)
 7: $\overrightarrow{q_i} \leftarrow$ **latentForgetting**($\overrightarrow{q_i}$)
 8: $epoch = 0$
 9: while $epoch < optimalNumberOfEpochs$ do
10:     epoch++
11:     $predictionError = r_{u,i} - \widehat{r}_{u,i}$
12:     for all latent dimensions $k \neq 1$ in $\overrightarrow{p_u}$ do
13:         $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
14:     end for
15: end while
```

of learning upon a rating is stored in a *deltaStorage* (cf. line 26). The impact in form of δ_{p_u} results from subtraction of a latent user vector before and after learning. If at a later time point this rating has to be forgotten, the impact of this rating is retrieved from the *deltaStorage* (cf. line 13) and subtracted from the corresponding latent user vector (cf. line 14).

Update of the preference model upon a new rating is done in the same way as in the baseline algorithm. The complexity of an update is again $O(E)$. However, the procedure requires a higher memory consumption due to the *deltaStorage*.

Algorithm 5 Incremental Learning with Approximative Rating-based Forgetting

```
Input: $r_{u,i}, R, P, Q$
Parameters: $\eta, k, \lambda$, $optimalNumberOfEpochs$
1: $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
2: $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
3: Extend and initialize new dimensions($\overrightarrow{p_u}, \overrightarrow{q_i}, P, Q, R$)
4: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
5: evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
6: $\overrightarrow{r}_{u*} \leftarrow$ getUserRatings($R, u$)
7: ($\overrightarrow{r}_{u*}$).addRating($r_{u,i}$)
8: $remainingRatings = ratingBasedForgetting(\overrightarrow{r}_{u*})$//obsolete ratings removed
9: $R$.overrideUserVector($\overrightarrow{r}_{u*}$) //obsolete ratings also removed from $R$
10: $ratingsToBeForgotten = \overrightarrow{r}_{u*} - remainingRatings$
11: for all $rating_{u,i}$ in $ratingsToBeForgotten$ do
12:     $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
13:     $\delta_{p_u} = deltaStorage$.getUserVectorImpact($rating_{u,i}$)
14:     $\overrightarrow{p_u} \leftarrow \overrightarrow{p_u} - \delta_{p_u}$
15: end for
16: $epoch = 0$
17: $\overrightarrow{p_u}^{beforeUpdate} = \overrightarrow{p_u}$
18: while $epoch < optimalNumberOfEpochs$ do
19:     epoch++
20:     $predictionError = r_{u,i} - \overrightarrow{p_u} \cdot \overrightarrow{q_i}$
21:     for all latent dimensions $k \neq 1$ in $\overrightarrow{p_u}$ do
22:         $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
23:     end for
24: end while
25: $\delta_{p_u} = \overrightarrow{p_u}^{beforeUpdate} - \overrightarrow{p_u}$
26: $deltaStorage$.storeImpactOnUser($\delta_{p_u}$)
```

5 Evaluation Settings

In this section we describe how we evaluate our methods. Our evaluation protocol encompasses

- a method for splitting datasets for incremental matrix factorization
- incremental recall measure by Cremonesi et al. [6]
- parameter optimization
- significance testing

We applied this evaluation protocol to all 8 datasets used in our experiments (cf. Sec. 6).

5.1 Dataset Splitting

Our method operates on a stream of ratings. However, matrix factorization requires an initialization phase. According to the description of the BRISMF algorithms, which we adopted here with modifications (cf. baseline algorithm in Sec. 4.1), latent item features are trained only in the initial phase. Therefore, this phase is of even higher importance to the BRISMF algorithm.

Because our methods operate in two phases, we use the evaluation protocol from [27, 26], briefly explained hereafter. Figure 1 represents an entire dataset with with three parts. Part 1) is used for initial training in the batch mode. To evaluate training of latent factors on part 1), we use the part 2) of the dataset ("batch testing"). After the initial training is finished, our method changes into the streaming mode, which is its main mode.

In this mode we use prequential evaluation, as proposed by Gama et al. [10]. In the streaming mode, for each new rating, first, a prediction is made and evaluated and then this rating is used for updating the corresponding preference model. Due to this temporal separation of prediction and update procedures, separation of the training and test datasets is guaranteed. In our experiments we use the following split ratios for the datasets: 20% for batch training, 30 % for batch testing and 50 % for the stream mode.

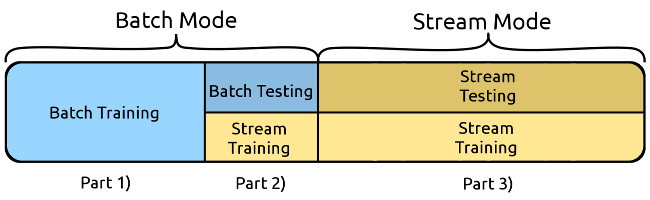

Fig. 1: Split of the dataset between the initialization and online phase (figure from [26]).

Since part 1) and 3) are used for training, part 2) of the dataset would represent a temporal gap in the training data. For stream-based methods that rely heavily on time aspects it is highly beneficial to maintain time continuity in the model. Therefore, we also use part 2) of the dataset for stream training. However, since it was used for batch testing, we don't use it as test set for the streaming mode.

5.2 Evaluation Measure

As quality measure we use incremental recall, as proposed by Cremonesi et al. [6]. It measures how often a recommender system can find a relevant item among random items in a stream. This measure should not be confused with the conventional recall. Also, while conventional precision and recall are complementary to each other and should always be considered together, it is not the case with the incremental recall and incremental precision. If the incremental recall was measured, then incremental precision can be derived from it and, therefore, it is redundant. For readers unfamiliar with this measure we refer to [6] and summarise the process of measuring it briefly.

First, in the process of measuring incremental recall, a relevance threshold is defined, above which items are considered relevant e.g. more than 4 out of 5 stars. If a relevant item is encountered in a stream, 1000 further items are chosen randomly. Those additional 1000 items are assumed to be irrelevant. All those 1000 random items and the relevant item are put into one set without indication of relevance. Subsequently, a recommender systems ranks all 1001 items from this set. If the relevant item has been ranked as one of top N items, then a hit is counted. The final value of incremental recall for a given N is calculated using the following formula:

$$incrementallRecall@N = \frac{\#hits}{|Testset|} \tag{18}$$

In our experiments we use $incrementalRecall@10$. In experiments with explicit rating feedback, the ranking made by the recommender systems is sorted with respect to the relevance score (highest first). In experiments with positive-only feedback, the value of 1 indicates the existence of feedback. Therefore, the ranking in this case is sorted with respect to the proximity of a predicted rating to 1.

Unlike in Secs. 3.1.5 and 3.2.4, where the incremental recall is measured for each user separately and used to control the extent of forgetting, here we use it as a global evaluation measure for the entire model.

We do not use the RMSE and MAE measures, because, unlike incremental recall, they do not consider any ranking of the items and they are based mostly on the great majority of non-relevant items.

5.3 Parameter Selection

Each of our forgetting strategies uses an additional parameter that needs to be set in advance (e.g. the size of a sliding window). To find the optimal values of the parameters we used a grid search algorithm that optimizes the average incremental recall for each dataset separately.

To avoid overfitting, the grid search was performed on a small sample from our datasets, called optimization set. The size of the optimization sets, expressed as a percentage of all users, ranged between 0.01 and 0.1 depending on the dataset size (cf. Tab. 1, column "Ratio of Users for Parameter Optimization"). Using the optimization set, we determined an approximately optimal parameter value. This value was then applied onto the holdout dataset, which is the remaining, bigger part of a dataset. The results reported in the next section are results on the holdout sets.

The remaining parameters used by matrix factorization and SGD were set to the following, approximately optimal values: number of dimensions = 40, learn rate $\eta = 0.003$ and regularization parameter $\lambda = 0.01$.

5.4 Significance Testing

To study the effect of our forgetting strategies we use significance testing. However, in the streaming scenario following all requirements of statistical tests is not a trivial task. One of the prerequisites of statistical tests is independence of observations of a random variable.

In our case the random variable is the average incremental recall. However, considering two measurements of incremental recall at timepoint t and $t + 1$ as independent would be wrong, since incremental recall is a cumulative measure. Therefore, quality at timepoint t affects the measurement of quality at timepoint $t + 1$. Consequently, the prerequisite of independent observations is violated.

As a solution to this problem we propose an alternative understanding of an observation. We partition every dataset into n disjoint, consecutive parts along the time dimension (imagine Fig. 1 n times along the time axis). Each of the parts is a sample of the entire dataset i.e. a sample from the same population. Since the samples are disjoint, they are also independent. As one observation we define the average incremental recall on one such sample. Consequently, on each dataset we observe n realisations of the random variable for each forgetting strategy. In our experiments we use $n = 10$, except for the ML100k dataset, where $n = 5$ due to its small size (cf. Tab. 1, column "Observations for Significance Testing").

Having n observations for each forgetting strategy and for our baseline, the "No Forgetting Strategy", we first test, if there is a significant difference among the strategies. For this purpose we use the Friedman test. It is more appropriate here than e.g. ANOVA, since it does not assume the normal distribution of the random variable. If the null hypothesis is rejected, then it can be assumed that there is a significant difference among the forgetting strategies.

If this is the case, we perform post-hoc tests to find out, which forgetting strategies are significantly better than our baseline, the "No Forgetting Strategy". Therefore, we use the Wilcoxon signed rank test (paired measurements with no assumption of normal distribution) with the following null hypothesis:

$$\overline{incr.Recall}_{StrategyX} = \overline{incr.Recall}_{NoForgettingStrategy} \tag{19}$$

and alternative hypothesis:

$$\overline{incr.Recall}_{StrategyX} > \overline{incr.Recall}_{NoForgettingStrategy} \tag{20}$$

where $\overline{x}$ denotes an average of the vector x.

If the null hypothesis is rejected, then the forgetting strategy X is significantly better than no forgetting. Since we test multiple hypothesis, we apply a correction for multiple testing according to the Hommel's method [34] to avoid the alpha error accumulation. In the next section we report the corresponding adjusted p-values and a summary of significant improvements.

6 Experiments

In this section we present results of our experiments on eight real-world datasets. We divide the results into ones showing the effects of the forgetting strategies (cf. Sec. 6.1) and ones showing the effects of our approximation of the rating-based implementation of forgetting (cf. Sec. 6.2). In total we conducted more than 1040 experiments on a cluster running a (Neuro)Debian operating system [11]. In those experiments we used datasets from Tab. 1. There are two types of datasets. The following description of them comes from our previous work [27].

The first type of datasets encompasses data with explicit rating feedback: MovieLens 1M and 100k[1] [12], a sample of 10 000 users from the extended Epinions [24] dataset and a sample of the same size from the Netfilx dataset. In this type of datasets our selection is limited, because many forgetting strategies require timestamp information.

The second type of datasets is based on positive-only feedback. Those datasets contain chronologically ordered user-item pairs in the form (u, i). Music-listen consists of music listening events, where each pair corresponds to a music track

[1] `http://www.movielens.org/`

Dataset	**Ratings**	**Users**	**Items**	**Sparsity**	**Ratio of Users for Parameter Optimiza-tion**	**Observa-tions for Signifi-cance Testing**
Music-listen	335,731	4,768	15,323	99.54%	0.1	10
Music-playlist	111,942	10,392	26,117	99.96%	0.1	10
LastFM-600k	493,063	164	65,013	95.38%	0.1	10
ML1M GTE5	226,310	6,014	3,232	98.84%	0.1	10
ML100k	100,000	943	1,682	93.7%	0.1	5
Netflix(10k users)	2,146,187	10,000	17,307	98.76%	0.01	10
Epinions (10k users)	1,016,915	10,000	365,248	99.97%	0.03	10
ML1M	1,000,209	6,040	3,706	95.53%	0.05	10

Table 1: Description of datasets. The column "Ratio of Users for Parameter Optimization" indicates what ratio of users was used to create an optimization dataset for the grid search. "Observations for Significance Testing" indicates the number of partitions of a dataset used as observations for significance testing.

being played by a user. Music-playlist consists of a timestamped log of music track additions to personal playlists. Contrary to Music-listen, Music-playlist contains *only* unique (u, i) pairs – users are not allowed to add a music track twice to the same playlist. Both Music-listen and Music-playlist are extracted from Palco Principal[2], an online community of portuguese-speaking musicians and fans. Furthermore, we also use a subset of the LasfFM[3] dataset [4] – LastFM-600k – and a binarized version of MovieLens 1M dataset that we call ML1M-GTE5 hereafter. In ML1M-GTE5 we assume that a rating value of 5 indicates a positive feedback. All remaining ratings have been removed and considered negative.

6.1 Impact of Forgetting Strategies

To show the impact of our forgetting strategies we compare them with the baseline algorithm from Sec. 4.1. This baseline employs no forgetting strategy. Therefore, we call it "No Forgetting Strategy" hereafter. In this subsection no approximation from Alg. 5 was used (for results using the approximation, see the next subsection).

First, we tested if the application of forgetting strategies has a significant impact on the quality of recommendations measured in incremental recall. For this purpose we used the Friedman rank sum test for each dataset separately as an omnibus test. The null hypothesis of this test states that all recall values are equal, no matter what forgetting strategy or no forgetting strategy was used.

In Tab. 2 we present results of the test on each dataset. The null hypothesis was clearly rejected on all datasets, which is indicated by low p-values. Consequently, we conclude that forgetting strategies make a significant difference in incremental recall values. Further in this section, we use post-hoc tests to find out which forgetting strategies are significantly better than no forgetting.

In Figure 2 we visualize the results of forgetting strategies on datasets with positive-only feedback. This figure contains box plots of incremental recall (higher values are better). Incremental precision in the streaming setting can be derived

[2] `http://www.palcoprincipal.com`

[3] `http://last.fm`

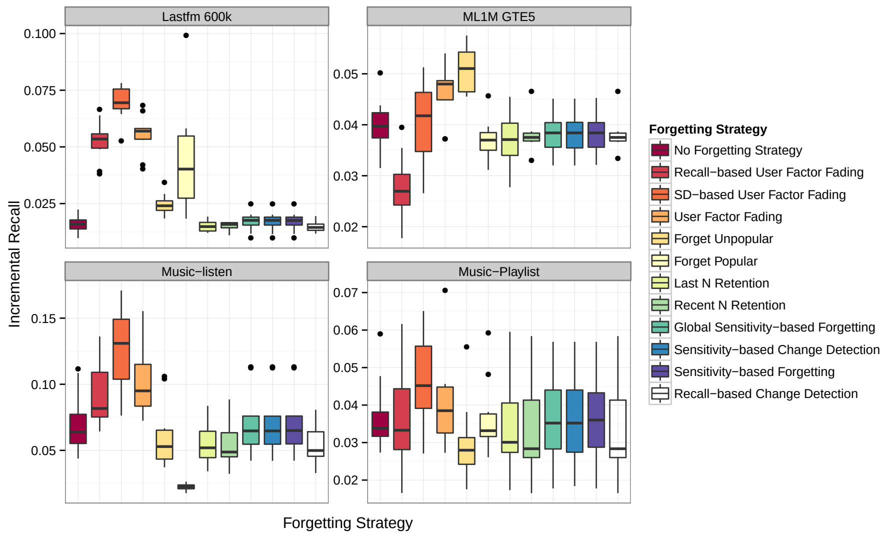

Fig. 2: Results of our forgetting strategies vs. "No Forgetting Strategy" (leftmost in the plots) on datasets with **positive-only feedback** (higher values are better). Latent factor based strategies are particularly successful.

Dataset	**p-value (Friedman rank sum test)**
Epinions Extended (10k users sample)	4.381e-16
Lastfm 600k	1.852e-04
ML1M GTE5	1.911e-09
ML1M	4.565e-11
ML100k	3.389e-09
Netflix (10k users sample)	1.735e-09
Music-listen	2.773e-05
Music-Playlist	7.246e-15

Table 2: Results of the Friedman rank sum test as an omnibus test for each dataset. Very low p-values indicate that forgetting makes a significance difference in the quality of recommendations.

from the recall measure and is, therefore, redundant (cf. [6]). Thus, we do not present the incremental precision.

Horizontal bars in each box in Fig. 2 represent medians of incremental recall from multiple partitions of a dataset (cf. Sec. 5). The hinges of each box represent the first and the third quartile of the distribution. Dots stand for outliers.

In the figure we, again, see that forgetting strategies have a great impact on the quality of recommendations as compared to the "No Forgetting Strategy" (leftmost in the plots). The latent factor forgetting strategies are particularly successful. On three out of four positive-only datasets the "SD-based User Factor Fading" was the best strategy. The "Forget Unpopular" strategy (also a latent factor forgetting strategy) performed the best on the MLGTE5 dataset only.

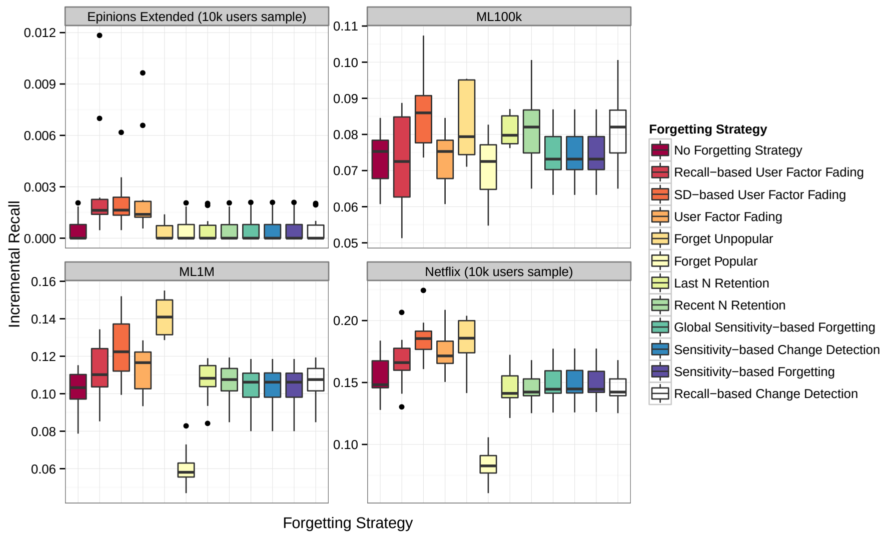

Fig. 3: Results of our forgetting strategies vs. "No Forgetting Strategy" (leftmost in the plots) on datasets with **explicit rating feedback** (higher values are better). Latent factor based strategies are particularly successful.

From this group of forgetting strategies the "Forget Popular" strategy is not recommendable. It performed better than the baseline on the Lastfm dataset only and otherwise considerably worse. The performance of rating-based strategies was generally worse than the one of the latent factor forgetting, often marginally different from the baseline.

In Tab. 3 we present the corresponding results of experiments with positive only feedback. The column "Param." describes the parameter setting for each forgetting strategy. The meaning of the parameter depends on the strategy itself (cf. Sec. 3). The parameter was determined by a grid search on a separate dataset not used for the final evaluation.

The column with mean incremental recall uses the following notation: $mean_{\pm std.\ deviation}$. The values of mean and standard deviation are based on multiple runs on different, consecutive parts of each dataset (cf. Sec. 5.4). The number of runs is indicated in Tab. 1 by column "Observations for Significance Testing". The forgetting strategy with the best value of mean incremental recall is marked in red.

P-values in the table refer to the Wilcoxon signed rank test (cf. Sec. 5.4). They are adjusted using Hommel's method to account for multiple testing. Values in bold font and a single asterisk indicate that the given strategy is better than the "No Forgetting Strategy" with significance level of 0.1. Values marked by two asterisks are significant at level of 0.05 and by three asterisks at level of 0.01. The column "Runtime" follows the same notation as the mean incremental recall. Values in each cell of this column represent a mean runtime in seconds on one partition of each dataset.

Forgetting Strategy	**Param.**	**Mean Incr. Recall**	**Adjusted P-value**	**Runtime (ms)**
Lastfm 600k				
No Forgetting Strategy	0	0.01583$_{\pm0.00361}$	-	65.84$_{\pm5.23}$
Recall-based User Factor Fading	**10^{12}**	**0.0525**$_{\pm0.00909}$	**0.00684*****	**74.11**$_{\pm7.78}$
SD-based User Factor Fading	**1.2**	**0.06953**$_{\pm0.00768}$	**0.00684*****	**64.21**$_{\pm4.91}$
User Factor Fading	**0.5**	**0.05541**$_{\pm0.00889}$	**0.00684*****	**64.31**$_{\pm3.2}$
Forget Unpopular	**2**	**0.02458**$_{\pm0.00465}$	**0.00684*****	**66.41**$_{\pm4.87}$
Forget Popular	**1.005**	**0.04436**$_{\pm0.02379}$	**0.00684*****	**66.24**$_{\pm5.11}$
Last N Retention	10	0.01496$_{\pm0.00241}$	0.88379	88.7$_{\pm9.4}$
Recent N Retention	1h	0.01519$_{\pm0.00183}$	0.88379	86.04$_{\pm7.31}$
Global Sensitivity-based Forgetting	0.1	0.01718$_{\pm0.00424}$	0.32032	55.99$_{\pm5.77}$
Sensitivity-based Change Detection	0.5	0.01716$_{\pm0.00426}$	0.32032	58.29$_{\pm3.48}$
Sensitivity-based Forgetting	10	0.01716$_{\pm0.00426}$	0.32032	58.14$_{\pm5.61}$
Recall-based Change Detection	0.05	0.01478$_{\pm0.00225}$	0.88379	1284.42$_{\pm196.36}$
ML1M GTE5				
No Forgetting Strategy	0	0.04005$_{\pm0.00501}$	-	9.04$_{\pm1.36}$
Recall-based User Factor Fading	10^{12}	0.02787$_{\pm0.00621}$	0.99903	9.38$_{\pm1.28}$
SD-based User Factor Fading	1.08	0.04075$_{\pm0.00795}$	0.99903	9.78$_{\pm2.46}$
User Factor Fading	**0.99**	**0.04627**$_{\pm0.00545}$	**0.00977*****	**8.94**$_{\pm1.38}$
Forget Unpopular	**1.2**	**0.05094**$_{\pm0.00452}$	**0.00977*****	**8.9**$_{\pm0.69}$
Forget Popular	1.00001	0.03706$_{\pm0.00403}$	0.99903	9.74$_{\pm2.41}$
Last N Retention	5	0.03692$_{\pm0.00517}$	0.99903	13.67$_{\pm2.06}$
Recent N Retention	1h	0.03803$_{\pm0.00342}$	0.99903	43.77$_{\pm13.41}$
Global Sensitivity-based Forgetting	0.1	0.03792$_{\pm0.00405}$	0.99903	11.3$_{\pm1.92}$
Sensitivity-based Change Detection	0.5	0.03789$_{\pm0.00408}$	0.99903	10.59$_{\pm1.61}$
Sensitivity-based Forgetting	10	0.03795$_{\pm0.00407}$	0.99903	10.81$_{\pm1.53}$
Recall-based Change Detection	0.05	0.03808$_{\pm0.00335}$	0.99903	45.06$_{\pm10.74}$
Music-Listen				
No Forgetting Strategy	0	0.07083$_{\pm0.02328}$	-	68.42$_{\pm4.09}$
Recall-based User Factor Fading	10^{12}	0.09178$_{\pm0.02308}$	0.58887	61.36$_{\pm5.35}$
SD-based User Factor Fading	**1.2**	**0.12713**$_{\pm0.02976}$	**0.01075****	**61.19**$_{\pm5.7}$
User Factor Fading	0.5	0.1033$_{\pm0.02649}$	0.18555	67.83$_{\pm8.67}$
Forget Unpopular	2	0.06078$_{\pm0.02505}$	1	66.58$_{\pm9.02}$
Forget Popular	1.005	0.02227$_{\pm0.00241}$	1	73.01$_{\pm9.58}$
Last N Retention	40	0.05588$_{\pm0.01745}$	1	101.73$_{\pm14.09}$
Recent N Retention	1 week	0.05527$_{\pm0.01885}$	1	224.31$_{\pm108.35}$
Global Sensitivity-based Forgetting	0.1	0.07082$_{\pm0.02473}$	1	34.99$_{\pm1.88}$
Sensitivity-based Change Detection	0.5	0.07082$_{\pm0.02476}$	1	36.8$_{\pm4.57}$
Sensitivity-based Forgetting	10	0.0709$_{\pm0.02473}$	1	35.97$_{\pm4.59}$
Recall-based Change Detection	0.05	0.05477$_{\pm0.01661}$	1	249.34$_{\pm93.92}$
Music-Playlist				
No Forgetting Strategy	0	0.03712$_{\pm0.00946}$	-	4.12$_{\pm0.9}$
Recall-based User Factor Fading	10^{12}	0.0366$_{\pm0.01358}$	0.99805	4.16$_{\pm0.75}$
SD-based User Factor Fading	**1.2**	**0.04708**$_{\pm0.01214}$	**0.05372***	**4.2**$_{\pm0.49}$
User Factor Fading	0.99	0.04058$_{\pm0.01232}$	0.99805	3.85$_{\pm0.63}$
Forget Unpopular	1.5	0.02972$_{\pm0.0109}$	0.99805	4.09$_{\pm0.91}$
Forget Popular	1.00001	0.03679$_{\pm0.00986}$	0.99805	4.1$_{\pm0.67}$
Last N Retention	40	0.03393$_{\pm0.01201}$	0.99805	10.94$_{\pm1.5}$
Recent N Retention	1 year	0.03333$_{\pm0.01231}$	0.99805	39.64$_{\pm48.78}$
Global Sensitivity-based Forgetting	0.1	0.03609$_{\pm0.01171}$	0.99805	4.2$_{\pm0.66}$
Sensitivity-based Change Detection	0.5	0.03604$_{\pm0.01171}$	0.99805	4.06$_{\pm0.75}$
Sensitivity-based Forgetting	10	0.03616$_{\pm0.0116}$	0.99805	4$_{\pm0.56}$
Recall-based Change Detection	0.05	0.03333$_{\pm0.01231}$	0.99805	38.75$_{\pm48.41}$

Table 3: Results on datasets with positive only feedback. Best value of incremental recall is marked in red. Asterisks indicate that a given strategy is significantly better than the no forgetting strategy (* at 0.1; ** at 0.05; *** at 0.01).

Forgetting Strategy	**Param.**	**Mean Incr. Recall**	**Adjusted P-value**	**Runtime (ms)**
Epinions Extended (10k users sample)				
No Forgetting Strategy	0	0.0005$_{\pm 0.00084}$	-	160.03$_{\pm 16.1}$
Recall-based User Factor Fading	**10^{10}**	**0.00307**$_{\pm 0.00357}$	**0.00879*****	**451.89**$_{\pm 38.52}$
SD-based User Factor Fading	**1.08**	**0.00219**$_{\pm 0.00163}$	**0.00879*****	**154.45**$_{\pm 11.68}$
User Factor Fading	**0.5**	**0.00274**$_{\pm 0.00297}$	**0.00879*****	**152.92**$_{\pm 14.87}$
Forget Unpopular	2	0.00035$_{\pm 0.00057}$	1	156.5$_{\pm 15.81}$
Forget Popular	1.00001	0.0005$_{\pm 0.00084}$	1	161.76$_{\pm 18.74}$
Last N Retention	10	0.0005$_{\pm 0.00084}$	1	192.48$_{\pm 24.29}$
Recent N Retention	4 weeks	0.0005$_{\pm 0.00084}$	1	2923$_{\pm 601.75}$
Global Sensitivity-based Forgetting	0.1	0.00052$_{\pm 0.00087}$	1	148.81$_{\pm 16.9}$
Sensitivity-based Change Detection	0.5	0.00052$_{\pm 0.00087}$	1	143.67$_{\pm 10.98}$
Sensitivity-based Forgetting	10	0.00052$_{\pm 0.00087}$	1	144.04$_{\pm 13.9}$
Recall-based Change Detection	0.05	0.00051$_{\pm 0.00085}$	1	5857.13$_{\pm 4815.21}$
ML100k				
No Forgetting Strategy	0	0.07336$_{\pm 0.0093}$	-	3.22$_{\pm 0.39}$
Recall-based User Factor Fading	10^{10}	0.07201$_{\pm 0.0155}$	1	3.24$_{\pm 0.19}$
SD-based User Factor Fading	1.04	0.08708$_{\pm 0.01319}$	0.1875	1.72$_{\pm 0.12}$
User Factor Fading	0.99999999	0.07336$_{\pm 0.0093}$	1	1.63$_{\pm 0.09}$
Forget Unpopular	1.5	0.08307$_{\pm 0.01151}$	0.1875	1.8$_{\pm 0.11}$
Forget Popular	1.00001	0.07039$_{\pm 0.01093}$	1	1.96$_{\pm 0.28}$
Last N Retention	10	0.08112$_{\pm 0.00476}$	0.1875	9.42$_{\pm 0.83}$
Recent N Retention	1h	0.08186$_{\pm 0.01331}$	0.1875	70.55$_{\pm 7.59}$
Global Sensitivity-based Forgetting	0.1	0.07462$_{\pm 0.00901}$	0.375	3.47$_{\pm 0.39}$
Sensitivity-based Change Detection	0.5	0.07462$_{\pm 0.00901}$	0.375	3.5$_{\pm 0.3}$
Sensitivity-based Forgetting	10	0.07462$_{\pm 0.00901}$	0.375	3.51$_{\pm 0.34}$
Recall-based Change Detection	0.05	0.08185$_{\pm 0.01331}$	0.1875	72.89$_{\pm 7.98}$
ML1M				
No Forgetting Strategy	0	0.10211$_{\pm 0.01104}$	-	32.41$_{\pm 7.35}$
Recall-based User Factor Fading	10^{12}	0.11249$_{\pm 0.01479}$	0.10547	63.63$_{\pm 16.63}$
SD-based User Factor Fading	**1.1**	**0.1244**$_{\pm 0.01697}$	**0.02051****	**33.65**$_{\pm 7.07}$
User Factor Fading	**0.99**	**0.11327**$_{\pm 0.01237}$	**0.00586*****	**33.41**$_{\pm 7.33}$
Forget Unpopular	**1.2**	**0.14088**$_{\pm 0.01014}$	**0.00586*****	**37.44**$_{\pm 9.3}$
Forget Popular	1.00001	0.06073$_{\pm 0.01053}$	1	37.52$_{\pm 8.69}$
Last N Retention	**10**	**0.10679**$_{\pm 0.01115}$	**0.00586*****	**64.18**$_{\pm 3.56}$
Recent N Retention	**1 week**	**0.10585**$_{\pm 0.01056}$	**0.00782*****	**939.05**$_{\pm 103.08}$
Global Sensitivity-based Forgetting	**0.1**	**0.1041**$_{\pm 0.01153}$	**0.01172****	**25.88**$_{\pm 2.03}$
Sensitivity-based Change Detection	**0.5**	**0.10412**$_{\pm 0.01154}$	**0.01172****	**25.95**$_{\pm 2.19}$
Sensitivity-based Forgetting	**10**	**0.10408**$_{\pm 0.01151}$	**0.01172****	**25.81**$_{\pm 2.07}$
Recall-based Change Detection	**0.05**	**0.10585**$_{\pm 0.01056}$	**0.00782*****	**951.31**$_{\pm 107.55}$
Netflix (10k users sample)				
No Forgetting Strategy	0	0.15455$_{\pm 0.0165}$	-	38.24$_{\pm 9.32}$
Recall-based User Factor Fading	10^{10}	0.16687$_{\pm 0.02184}$	0.64063	73.58$_{\pm 13.47}$
SD-based User Factor Fading	**1.02**	**0.18644**$_{\pm 0.01707}$	**0.00977*****	**39.67**$_{\pm 9.25}$
User Factor Fading	**0.99**	**0.17442**$_{\pm 0.0163}$	**0.00977*****	**38.16**$_{\pm 7.61}$
Forget Unpopular	**1.1**	**0.18341**$_{\pm 0.01988}$	**0.06153***	**47.64**$_{\pm 7.61}$
Forget Popular	1.00001	0.08305$_{\pm 0.0129}$	1	47.2$_{\pm 7.93}$
Last N Retention	5	0.1454$_{\pm 0.01463}$	1	78.61$_{\pm 12.89}$
Recent N Retention	1 year	0.14521$_{\pm 0.0125}$	1	1446.4$_{\pm 342.16}$
Global Sensitivity-based Forgetting	0.5	0.14935$_{\pm 0.01454}$	1	61.86$_{\pm 13.76}$
Sensitivity-based Change Detection	0.5	0.14933$_{\pm 0.01465}$	1	58.37$_{\pm 10.4}$
Sensitivity-based Forgetting	3	0.14942$_{\pm 0.01439}$	1	59.43$_{\pm 11.4}$
Recall-based Change Detection	0.05	0.14521$_{\pm 0.01251}$	1	1508.71$_{\pm 333.19}$

Table 4: Results on datasets with explicit rating feedback. Best value of incremental recall is marked in red. Asterisks indicate that a given strategy is significantly better than the no forgetting strategy (* at 0.1; ** at 0.05; *** at 0.01).

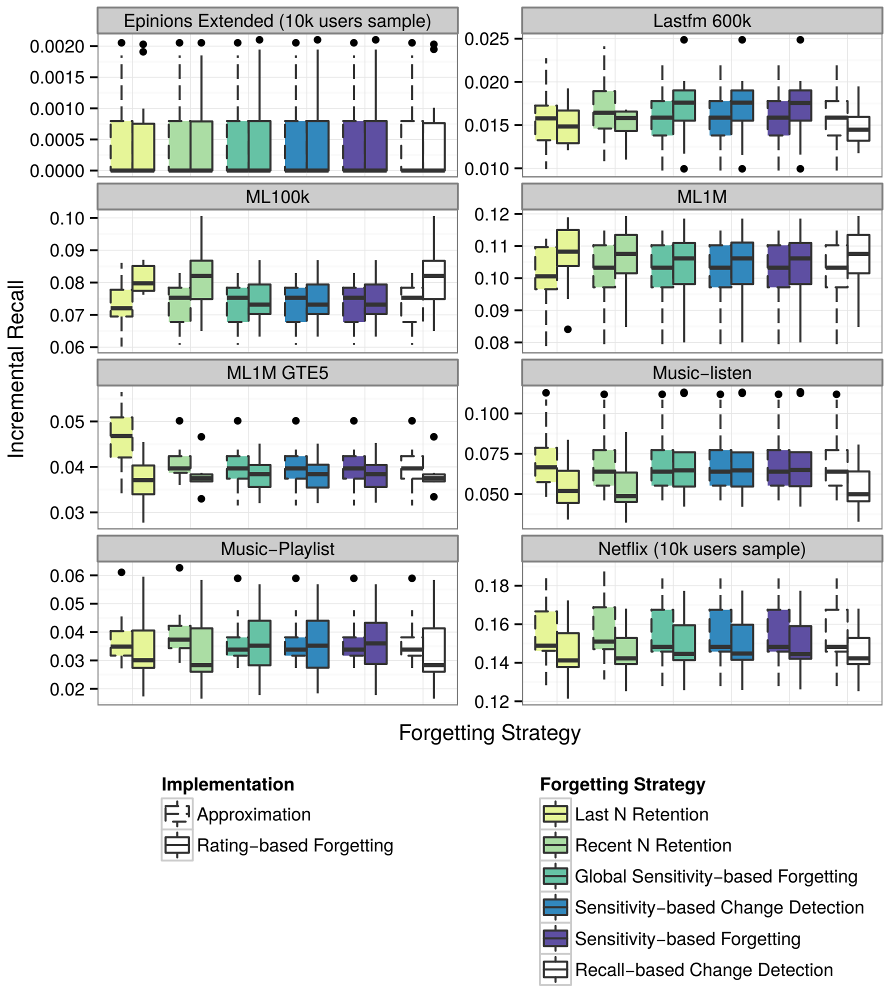

Fig. 4: Incremental recall of approximative rating based forgetting is similar to the non-approximative variant.

Tab. 3 shows that on all datasets with positive only feedback there is at least one forgetting strategy that is significantly better than no forgetting at the significance level better than 0.054. On three out of four datasets the best strategy was the SD-based User Factor Fading, on the remaining dataset it was the Forget Unpopular strategy.

Forgetting strategies brought on this type of datasets an improvement in the incremental recall of 118.18 % on average (as a result of comparison of the best strategy with the "No Forgetting Strategy"). Especially on the Lastfm 600k dataset recall improved from 0.01583 to 0.06953 . The median of improvement is 53,34%.

Not only did quality improve, the computation time decreased for the best strategy by 3,21 % on average. However, considering the high variance of runtime, this decrease is not substantial. Some strategies, e.g. "Recall-based Change Detection" showed higher computation time.

In Fig. 3 and in Tab. 4 we present analogous results of experiments with datasets with explicit rating feedback. Also here, the latent factor based strategies perform the best except for the "Forget Popular" strategy. In Tab. 4 we show improvements in quality of recommendations over the baseline. The significance level on three out of four of the datasets is better than 0.01. Only on the ML100k dataset no significant improvement could be shown. ML100k is a small dataset with only five observations, therefore, it is difficult to show significance on this dataset.

The improvement of quality due to forgetting reached 147,83 % on average (best strategy vs. no forgetting). The percentage is so high because of the extreme improvement on the Epinions dataset. The median of the improvement amounts to 19,67%. The runtime, when using the best strategy, increased on average by 48.62%, the median of the percentual runtime increase is 9,63%, though.

6.2 Impact of the Approximative Implementation

Since the rating-based forgetting strategies have a higher complexity than the latent factor based ones, we implemented also an approximative way of using them (cf. Sec. 4.4). This implementation stores a past impact of a rating and undoes it in an approximative way when the rating should be forgotten.

In Figure 4 we present the incremental recall values achieved by this approximative implementation (dashed line) in comparison to the original implementation of rating-based forgetting (solid line). The approximation performed similarly to the original implementation. In few cases it performed better (e.g. for the "Last N Retention" strategy on the ML1M GTE5 dataset). However, those cases are rather exceptional and no significant improvement can be concluded based on them. There are also cases with a decrease of the performance, such as on ML100k dataset.

Nevertheless, the goal of the approximation is to maintain a similar level of quality while decreasing the runtime. We present such a runtime comparison of those two implementations in Tab. 5. Values in the table represent median of a runtime of multiple runs of our algorithms. The values are grouped by dataset and implementation (approximation vs. the original rating-based implementation).

The approximation decreased the computation time for the strategies Last N Retention, Recent N Retention, Recallbased Change Detection. For the remaining three strategies the approximation often took more time despite lower complexity. This is explained by the fact that the approximative implementation of forgetting changes the latent model in a different way than the rating-based implementation does. Therefore, a forgetting strategy can select different ratings to forget depending on which implementation is used (approximative vs. rating-based). Consequently, a given forgetting strategy can decide to forget more, if the approximative implementation is used. Then, due to more storing and retrieving operations from the delta storage, the approximative implementation can require a longer computation time.

Consequently, we recommend the usage of the approximation only after prior testing of its behaviour with a given forgetting strategy.

Forgetting Strategy	**Approx.**	**Rating-based**	**Approx.**	**Rating-based**
	Epinions (10k users)		**Lastfm 600k**	
Last N Retention	106.22	185.60	74.74	91.48
Recent N Retention	235.61	3068.94	91.59	86.51
Global Sensitivity-based Forgetting	381.59	142.08	198.23	56.16
Sensitivity-based Change Detection	439.96	142.07	211.40	59.68
Sensitivity-based Forgetting	387.46	143.43	208.36	58.69
Recall-based Change Detection	375.18	3352.39	128.50	1255.23
	ML100k		**ML1M**	
Last N Retention	2.27	23 621.00	16.99	65.18
Recent N Retention	2.52	69.52	35.37	948.10
Global Sensitivity-based Forgetting	4.16	16 497.00	64.84	26.33
Sensitivity-based Change Detection	3.87	16 862.00	60.44	26.73
Sensitivity-based Forgetting	3.63	23 802.00	55.35	26.52
Recall-based Change Detection	4.30	72.60	46.72	955.87
	ML1M GTE5		**Music-listen**	
Last N Retention	9.41	42 627.00	49.13	100.29
Recent N Retention	9.22	45.27	51.63	187.98
Global Sensitivity-based Forgetting	9.66	33 909.00	47.86	35.11
Sensitivity-based Change Detection	10.22	42 411.00	55.00	36.17
Sensitivity-based Forgetting	9.58	34 608.00	55.75	35.06
Recall-based Change Detection	8.85	47.31	51.55	225.22
	Music-Playlist		**Netflix (10k users)**	
Last N Retention	3.61	43 405.00	63.72	77.04
Recent N Retention	3.82	42 571.00	81.58	1299.18
Global Sensitivity-based Forgetting	4.84	44 287.00	156.48	62.35
Sensitivity-based Change Detection	4.57	30 376.00	161.85	59.99
Sensitivity-based Forgetting	3.79	42 404.00	158.41	61.00
Recall-based Change Detection	3.40	19.22	153.33	1327.40

Table 5: Median runtime (in seconds) of the approximative and rating-based implementation of forgetting.

7 Conclusions

Before our work, adaptation to changes in recommender systems was implemented only by incorporating new information from a stream into a preference model. While this is a valid method of adaptation, it is not sufficient. In this paper, we have shown that forgetting obsolete information (additionally to incorporating new one) significantly improves predictive power of recommender systems.

We proposed eleven unsupervised strategies to select the obsolete information and three algorithms to enforce forgetting. In our experiments we used a state-of-the-art incremental matrix factorization algorithm, BRISMF [36], and extended it by the ability to forget information and to add new dimensions to the matrix.

Our forgetting strategies can be applied to any matrix factorization algorithm. Rating-based strategies are also applicable to neighbourhood-based methods.

Further, we proposed a new evaluation approach that includes significance testing. We conducted more than 1040 experiments on eight real world datasets with explicit rating feedback and positive only feedback. On seven out of eight of those datasets we observed a **significantly better performance** when using forgetting. On five of them the improvement was significant at level better than 0.01.

From all our forgetting strategies, the latent factor based ones were particularly successful in terms of quality of recommendations and in terms of computation time. On five out of eight datasets the "SD-based User Factor Fading" strategy achieved the best result, followed by the "Forget Unpopular" strategy with best result on two out of eight datasets. Rating-based forgetting strategies also showed significant improvements over the "No Forgetting" strategy, however, their improvement was not as high, as in the case of latent-based strategies. We remark that not all of the forgetting strategies achieved a significant improvement. The strategy that achieved highly significant improvement on the most datasets (6 out of 8) is the *SD-based user factor fading* strategy.

Rating-based forgetting strategies have higher complexity and, therefore, a higher runtime than latent factor based strategies. Therefore, we proposed an approximative implementation that maintains a similar level of incremental recall as the original, rating-based implementation. For half of the strategies it reduced the computation time considerably. For the other half the computation often took longer. Therefore, we recommend to use this approximation only in time-critical applications and after prior testing.

Our recommendation to practitioners is to, first, use the latent factor based forgetting strategies (e.g. the *SD-based user factor fading* strategy) due to their outstanding performance in terms of incremental recall and only slightly increased computation time. The rating-based implementation forgetting is the next best alternative.

Despite the significant improvements due to forgetting, our implementation also has limitations. One of them is related to the BRISMF algorithm. Since it does not incrementally update the latent item features, it still requires an occasional retraining to consider changes in the item features. This limitation is, however, more related to the learning algorithm itself than to forgetting strategies. Another limitation affects the rating-based forgetting strategies. Unlike the latent factor-based strategies, they need to maintain the rating matrix R in the main memory, from which our method selects the ratings to be forgotten. However, this requirement is also true for many conventional recommendation algorithms without forgetting.

In our future work we plan to investigate how to detect a sudden change in preferences of users (i.e. concept shift). The improvements due to forgetting indicate the existence of concept drift in users' preferences. However, a detection of sudden changes is extremely challenging and requires a different treatment compared to gradual changes.

Furthermore, we intend to investigate ensembles of forgetting strategies. It is possible to combine several forgetting strategies in one system. However, task of combining forgetting strategies is complex. The complexity results from the huge number of possible combinations of forgetting strategies. Also their interplay requires a thorough investigation. Fitting of parameters is also not trivial in an ensemble setting as forgetting strategies influence each other and need be tuned for each combination separately.

Acknowledgements

We would like to thank to the Institute of Psychology II at the University of Magdeburg for making their computational cluster available for our experiments.

This work is financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the FCT Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project POCI-01-0145-FEDER-006961.

References

1. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *PODS*, pages 1–16. ACM, 2002.
2. M.W. Berry, S.T. Dumais, and G.W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM review*, pages 573–595, 1995.
3. Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-Based Clustering over an Evolving Data Stream with Noise. In Joydeep Ghosh, Diane Lambert, David B. Skillicorn, and Jaideep Srivastava, editors, *SDM*. SIAM, 2006.
4. O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
5. Freddy Chong Tat Chua, Richard Jayadi Oentaryo, and Ee-Peng Lim. Modeling Temporal Adoptions Using Dynamic Matrix Factorization. In Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *ICDM*, pages 91–100. IEEE Computer Society, 2013.
6. Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *Proceedings of ACM RecSys*, RecSys '10, pages 39–46. ACM, 2010.
7. Yi Ding and Xue Li. Time weight collaborative filtering. In Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken, editors, *CIKM*, pages 485–492. ACM, 2005.
8. Pedro Domingos and Geoff Hulten. Catching up with the Data: Research Issues in Mining Data Streams. In *DMKD*, 2001.
9. João Gama. A Survey on Learning from Data Streams: Current and Future Trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012.
10. João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *KDD*, 2009.
11. Yaroslav O. Halchenko and Michael Hanke. Open is Not Enough. Let's Take the Next Step: An Integrated, Community-Driven Computing Platform for Neuroscience. *Front. Neuroinform.*, 2012.
12. F. Maxwell Harper and Joseph A. Konstan. The MovieLens Datasets: History and Context. *TiiS*, 5(4):19, 2016.
13. Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 263–272. IEEE Computer Society, 2008.
14. G. Hulten, L. Spencer, and P. Domingos. Mining Time-Changing Data Streams. *ACM SIGKDD*, 2001.
15. Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant, editors, *KDD*, pages 97–106. ACM, 2001.

16. Rasoul Karimi, Christoph Freudenthaler, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Towards Optimal Active Learning for Matrix Factorization in Recommender Systems. In *ICTAI*, pages 1069–1076. IEEE, 2011.
17. Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009.
18. Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.
19. I. Koychev. Gradual Forgetting for Adaptation to Concept Drift. In *ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning, Berlin, Germany*, pages 101–106, 2000.
20. I. Koychev and I. Schwab. Adapting to Drifting User's Interests. In *ECML 2000*, 2000.
21. Xue Li, Jorge M. Barajas, and Yi Ding. Collaborative filtering on streaming data with interest-drifting. *Intell. Data Anal.*, 11(1):75–87, 2007.
22. Guang Ling, Haiqin Yang, Irwin King, and Michael R. Lyu. Online learning for collaborative filtering. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2012.
23. Nathan Nan Liu, Min Zhao, Evan Wei Xiang, and Qiang Yang. Online evolutionary collaborative filtering. In *Proc. of the ACM RecSys*, 2010.
24. Paolo Massa and Paolo Avesani. Trust-aware bootstrapping of recommender systems. In *ECAI Workshop on Recommender Systems*, pages 29–33. Citeseer, 2006.
25. Pawel Matuszyk and Myra Spiliopoulou. Selective Forgetting for Incremental Matrix Factorization in Recommender Systems. In *Discovery Science*, volume 8777 of *LNCS*, pages 204–215. Springer International Publishing, 2014.
26. Pawel Matuszyk and Myra Spiliopoulou. Semi-supervised Learning for Stream Recommender Systems. In Nathalie Japkowicz and Stan Matwin, editors, *Discovery Science*, volume 9356 of *LNCS*, pages 131–145. Springer International Publishing, 2015.
27. Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. Forgetting Methods for Incremental Matrix Factorization in Recommender Systems. In *Proceedings of the ACM SAC*, SAC '15, pages 947–953, New York, NY, USA, 2015. ACM.
28. Catarina Miranda and Alípio Mário Jorge. Incremental Collaborative Filtering for Binary Ratings. In *Web Intelligence Conference Proceedings*, pages 389–392. IEEE Computer Society, 2008.
29. Olfa Nasraoui, Jeff Cerwinske, Carlos Rojas, and Fabio A. González. Performance of Recommendation Systems in Dynamic Streaming Environments. In *SDM*. SIAM, 2007.
30. Olfa Nasraoui, Cesar Cardona Uribe, Carlos Rojas Coronel, and Fabio A. González. TECNO-STREAMS: Tracking Evolving Clusters in Noisy Data Streams with a Scalable Immune System Learning Model. In *Proceedings of the IEEE ICDM 2003*, pages 235–242, 2003.
31. Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. Incremental Collaborative Filtering for Highly-Scalable Recommendation Algorithms. In *ISMIS 2005, Proceedings*, 2005.
32. Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07*.
33. Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of Dimensionality Reduction in Recommender System - A Case Study. In *In ACM WEBKDD Workshop*, 2000.
34. J P Shaffer. Multiple Hypothesis Testing. *Annual Review of Psychology*, 46(1):561–584, 1995.
35. John Z. Sun, Dhruv Parthasarathy, and Kush R. Varshney. Collaborative Kalman Filtering for Dynamic Matrix Factorization. *IEEE Trans. Signal Processing*, 62(14):3499–3509, 2014.
36. Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable Collaborative Filtering Approaches for Large Recommender Systems. *J. Mach. Learn. Res.*, 10, 2009.
37. João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18(4):271–282, 2012.
38. João Vinagre, Alípio Mário Jorge, and João Gama. Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback. In *UMAP*, pages 459–470, 2014.