Forgetting Methods for Incremental Matrix Factorization in Recommender Systems

Pawel Matuszyk
Otto von Guericke University
Magdeburg, Germany
pawel.matuszyk@iti.cs.uni-magdeburg.de

João Vinagre
FCUP - Universidade do Porto
LIAAD - INESC TEC
Porto, Portugal
jnsilva@inesctec.pt

Myra Spiliopoulou
Otto von Guericke University
Magdeburg, Germany
myra@iti.cs.uni-magdeburg.de

Alípio Mário Jorge
FCUP - Universidade do Porto
LIAAD - INESC TEC
Porto, Portugal
amjorge@fc.up.pt

João Gama
FEP - Universidade do Porto
LIAAD - INESC TEC
Porto, Portugal
jgama@fep.up.pt

ABSTRACT

Numerous stream mining algorithms are equipped with forgetting mechanisms, such as sliding windows or fading factors, to make them adaptive to changes. In recommender systems those techniques have not been investigated thoroughly despite the very volatile nature of users' preferences that they deal with. We developed five new forgetting techniques for incremental matrix factorization in recommender systems. We show on eight datasets that our techniques improve the predictive power of recommender systems. Experiments with both explicit rating feedback and positive-only feedback confirm our findings showing that forgetting information is beneficial despite the extreme data sparsity that recommender systems struggle with. Improvement through forgetting also proves that users' preferences are subject to concept drift.

Categories and Subject Descriptors

H.3.3 [**Information Search-Retrieval**]: Information Filtering

General Terms

Algorithms, Experimentation, Performance

Keywords

Recommender Systems, Matrix Factorization, Forgetting, Stream Mining

1. INTRODUCTION

Recommender systems learn users' preferences to predict and recommend a personalized set of items to a given user. Learning users' preferences is a challenging task. One of it's most demanding aspects is the constant change of the preferences and of the environment of a recommender system. The changes are driven by the evolution of users' taste and by external, unpredictable events. A successful recommender system has to account for those changes and adapt to them, ideally in real time, in order to reflect only the current interests of its users. The existence of change, which in stream mining is often referred to as concept drift or shift, implies that some of the data we use to build our prediction models are outdated. Using outdated information to build a prediction model leads to performance decline. Consequently, we should forget the outdated information in order to keep our models up-to-date with the current state of the environment and the current preferences of users.

In [9], Matuszyk and Spiliopoulou propose two strategies for selecting outdated information and show that forgetting such information leads to better predictive performance of a recommender system. In this paper, we extend their work by proposing new, more sophisticated forgetting strategies. We show that they further improve the predictive power of recommender systems for two types of user feedback.

Depending on the type of feedback users provide, recommendation problems can be formulated in two ways: rating prediction or item prediction. In a rating prediction problem the main task is to predict missing values in a *numeric* user-item ratings matrix. However, some systems employ positive-only user preferences. The user-item matrix can be seen then as a boolean value matrix, where $true$ values indicate a positive user-item interaction, and $false$ values indicate the absence of such interaction. In systems with positive-only feedback, the task is to predict unknown $true$ values in the ratings matrix, i.e. item prediction.

While most of the available literature in the field provides relevant research on the rating prediction problem, we think that issues specific to the item prediction are just as relevant. The main motivation for this is that while relatively few online systems have an implemented numeric rating feature, most systems use positive-only feedback.

The methods proposed in this paper are designed for and experimented in both settings – rating prediction and item prediction. To our knowledge, this is the first work that consolidates the validity of a recommendation algorithm in both of these two fundamentally distinct scenarios.

In summary, our contributions are as follows:

- We developed five new, more sophisticated forgetting strategies and divided them into two categories.
- We extended the evaluation protocol of Matuszyk and Spiliopoulou[9] by using an incremental measure of recall.
- We performed experiments with forgetting on positive-only feedback and on rating feedback.
- We showed that one of our new strategies outperforms the baseline with respect to incremental recall.
- We confirmed on in total eight datasets that forgetting information is beneficial in recommender systems.

This paper is structured as follows. In the next section we discuss related work. Section 3 explains the matrix factorization algorithm we used together with our extensions as well as our new forgetting strategies. Evaluation protocol and measures are mentioned in Section 4. In Section 5 we report about our results. Finally, we conclude our work and discuss remaining issues in Section 6.

2. RELATED WORK

Collaborative Filtering (CF) algorithms take usage data to compute models and make recommendations. These usage data share all the characteristics of a data stream. From this perspective, CF can be addressed as a data stream problem. However, most research in the field of recommender systems assume that data is static. In a real world system this kind of approach usually requires frequent batch retraining of models in order to keep models up to date, which potentially causes serious scalability problems, as the available data continuously grows over time. When processing data streams, key differences from static datasets need to be taken in consideration. Data elements arrive on-line at unpredictable rates and order. Also, it cannot be assumed that data streams fit in the working memory. Data elements must be discarded or archived at some time point.

One key aspect of most data stream mining algorithms is that they build models incrementally. Neighborhood-based incremental CF for ratings data has been introduced in [11], and adapted for positive-only feedback in [10]. These use incremental cache structures in order to be able to incrementally update a similarity matrix. Incremental matrix factorization for recommendation has been first studied in [13] using the Fold-in method for Singular Value Decomposition [1]. However, state-of-the-art factorization for recommender systems relies on optimization methods [12]. Incremental versions of these optimization methods have been studied in [14, 7] for rating prediction and in [16] for item prediction.

In [15], Vinagre and Jorge added forgetting capabilities to incremental neighbourhood-based algorithms for positive-only data. Forgetting is achieved either abruptly, using a fixed-size sliding window over data, or gradually, using a decay function in the similarity calculation. However, these techniques are only effective in the presence of a sudden global change in the feedback data, and do not account for more subtle changes that may occur, for instance, on the individual users' level. A similar approach is given in [4] with ratings data, in a batch algorithm.

In [6] Li et al. explicitly approach recommendation as a stream mining problem using the CVFDT algorithm for recommendation with user interest drifts. While this is an interesting approach, accuracy is only competitive when item taxonomies are added as an additional source of information.

3. SELECTIVE FORGETTING METHOD

We develop forgetting strategies for incremental matrix factorization. For our experiments we extend a state-of-the-art representative for this class of algorithms, the BRISMF algorithm (biased regularized incremental simultaneous matrix factorization) proposed by Tákacs et al. [14]. Nevertheless, our forgetting strategies are also applicable to different matrix factorization methods.

3.1 Matrix Factorization with Forgetting

The method proposed by Tákacs et al. in [14] is a batch method. However, the authors also proposed an algorithm for retraining latent users features (cf. Algorithm 2. in [14]) 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, 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.

Initial Phase.

Similarly to [9] and [14], 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 the latent factors [14]:

$$p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k}) \quad (1)$$

$$q_{i,k} \leftarrow q_{i,k} + \eta \cdot (predictionError \cdot p_{u,k} - \lambda \cdot q_{i,k}) \quad (2)$$

Where η is a learn rate of the SGD and λ is a regularization parameter that prevents SGD from overfitting.

Online Phase.

Once the initial phase is finished the online phase is started. Here, evaluation and prediction tasks run in parallel (cf. Section 4 for more information on the evaluation setting). With every new rating arriving in a data stream the corresponding latent user factor is updated. We extended the algorithm proposed in [14] and [9]. A pseudo code showing our extensions and the time point, when forgetting is triggered, is presented in Algorithm 1. As input the algorithm takes the original rating matrix R and two factor matrices P and Q that were learnt in the initial phase. Furthermore, SGD requires setting a few parameters e.g. η as the learning rate. λ is a regularization parameter and k stands for the number of latent dimensions. $r_{u,i}$ is the rating that appeared in a data stream and that the algorithm adapts to.

We developed two types of forgetting strategies that are called at three different places in the code. In line 7, a *rating-based forgetting* is triggered (cf. Subsection 3.2). Lines 14 and 15 call the second type of our strategies, *latent factor forgetting*, for latent user vectors and latent item vectors respectively (cf. Subsection 3.3). In the next subsection we provide a detailed explanation of those strategies.

Algorithm 1 Incremental Learning with Forgetting

```
Input: $r_{u,i}, R, P, Q, \eta, k, \lambda$
 1: $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
 2: $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
 3: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
 4: evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
 5: $\overrightarrow{r}_{u*} \leftarrow$ getUserRatings($R, u$)
 6: ($\overrightarrow{r}_{u*}$).addRating($r_{u,i}$)
 7: forgetRatings($\overrightarrow{r}_{u*}$) //old ratings removed
 8: $epoch = 0$
 9: while $epoch < optimalNumberOfEpochs$ do
10:   epoch++; //for all retained ratings
11:   for all $r_{u,i}$ in $\overrightarrow{r}_{u*}$ do
12:     $\overrightarrow{p_u} \leftarrow$ getLatentUserVector($P, u$)
13:     $\overrightarrow{q_i} \leftarrow$ getLatentItemVector($Q, i$)
14:     $\overrightarrow{p_u} \leftarrow$ latentForgetting($\overrightarrow{p_u}$)
15:     $\overrightarrow{q_i} \leftarrow$ latentForgetting($\overrightarrow{q_i}$)
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
```

3.2 Rating-based Forgetting

The first category of our forgetting strategies is based on filtering ratings of a user u. Before a latent vector p_u of a user is retrained using a new rating that occurred in a stream, the set of ratings belonging to the user $R(u)$ is filtered to exclude the outdated information. Here, we describe four strategies for selecting the outdated information. There is no ground truth that defines when an information is outdated. Nevertheless, we show that our strategies improve accuracy of a recommender system (cf. Section 5). In all of those strategies we have built in a threshold defining the minimum number of ratings a users needs to have for the forgetting methods to be used. We use a threshold of five ratings. Users with less than five ratings are not affected by our forgetting methods.

Sensitivity-based Forgetting.

As the name suggests, this strategy is based on sensitivity analysis. We analyse, how much the 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 of this particular user that we know so far, 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, 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:

$$\Delta_{p_u} = \sum_{i=0}^{k} (p_{u,i}^{t+1} - p_{u,i}^{t})^2 \tag{3}$$

Furthermore, for each user we store and update incrementally at each time point the standard deviation of the squared difference $SD(\Delta_{p_u})$. If the following inequality is true, then it indicates that the new rating $r_{u,i}$ is an outlier:

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

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.

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 globally. Formally, it means that the inequality (4) needs to be changed into:

$$\Delta_{p_u} > \alpha \cdot SD(\Delta) \tag{5}$$

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

LastNRetention.

To the category of rating-based forgetting we also count the *LastNRetention* strategy by Matuszyk and Spiliopoulou [9]. 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. We compare this strategy with our new forgetting strategies and present the results in Section 5.

RecentNRetention.

This forgetting strategy is also based on a sliding window by Matuszyk and Spiliopoulou [9]. However, here they defined the size of a user-specific window not in terms of ratings, but in terms of time. If $N = 3days$, then only ratings from the last three days are kept and all the remaining ones are forgotten. This strategy can be also set up to use only ratings from e.g. the last session. For completeness and comparison we also experiment with this strategy in our work.

3.3 Latent Factor Forgetting

This type of forgetting strategies is based on latent factors. After learning upon a new rating $r_{u,i}$ the corresponding user latent vector p_u or the corresponding item vector q_i are altered in a way dictated by one of the following strategies.

Forget 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:

$$\overrightarrow{q_i^{t+1}} = (-\alpha^{-|R(i)|} + 1) \cdot \overrightarrow{q_i^t} \tag{6}$$

$R(i)$ 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). An advantage of this function is that for values of $\alpha > 1$ it always takes values from the range $[0, 1)$.

We also experimented with penalizing *popular items*, however, this strategy was not successful. Therefore, we do not present it in detail here.

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 $\alpha \in [0, 1]$

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

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

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_u (cf. Equation 3). Subsequently, we use the standard deviation (SD) of Δ_u in an exponential function:

$$\overrightarrow{p_u^{t+1}} = \alpha^{-SD(\Delta_u)} \cdot \overrightarrow{p_u^t} \tag{8}$$

For high standard deviation of Δ_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 $[0, 1)$.

4. EVALUATION SETTINGS

Our method requires a short offline initialization phase; hence, we need different evaluation settings for the online and the offline phases. We adopt a framework from [9] that combines those two evaluation methods and we extend it by using an additional measure, incremental recall.

4.1 Evaluation Protocol

In Figure 1 we present the split of the dataset between online and offline phase of our method based on [9]. In the initial phase two subsets are used. The first subset is a training set ("Batch Train" in Fig. 1) and the second subset is a test set for the initialization set ("Batch Test"). The exemplary split ratios that we also used in our experiments are depicted in the figure. In the online phase prediction and evaluation run in parallel. This evaluation method is inspired by the prequential evaluation in stream mining by Gama et al. [5].

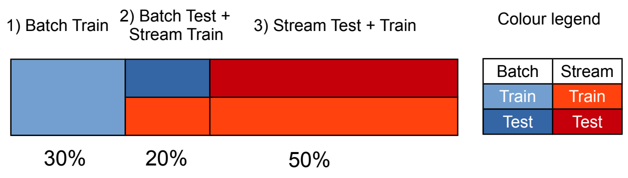

Figure 1: Split of the dataset between the initialization and online phase [9].

First, for a newly arrived rating in a data stream a prediction is made. Based on this prediction evaluation measures are updated and, finally, the model is updated using the new rating. This procedure guarantees that no prediction is made for a rating that has been seen before. However, the transition from the batch phase to the online phase is problematic. So far, datasets 1) and 3) from Figure 1 are used for learning. It means that dataset 2) represents a gap in the model. To avoid this gap, which is problematic for online methods, we use this part of the dataset also for online learning. Since the dataset 2) was used for testing already, we start the online evaluation on dataset 3).

4.2 Evaluation Measures

For our evaluation we use two measures. First of them is a sliding variant of RMSE (Root Mean Squared Error) [9].

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (r_{u,i} - \widehat{r}_{u,i})^2} \tag{9}$$

Where $\widehat{r}_{u,i}$ is a predicted value for $r_{u,i}$ and T is the test set of ratings. In our sliding variant of RMSE, T contains n most current ratings from the data stream. For our experiments we use $n = 500$.

Our second evaluation measure is incremental recall. For measuring it we use the methodology proposed by Cremonesi et al. in [3]. At each new rating in the data stream the measure is updated in the following way.
A prediction is made for the new rating $r_{u,i}$. Additionally to this prediction, 1000 further unrated items are selected randomly. Cremonesi et al. assume that those 1000 items are not relevant for the user. For all of them rating predictions are made and all the predictions are sorted. After the sorting the position p of $r_{u,i}$ is determined. If $p < N$, then a hit is counted. Finally, *incrementalRecall@N* is determined using the following formula:

$$incrementallRecall@N = \frac{\#hits}{|Testset|} \tag{10}$$

Dataset	Ratings	Users	Items	Sparsity
Music-listen	335,731	4,768	15,323	99,54%
Music-playlist	111,942	10,392	26,117	99,96%
LastFM-600k	493,063	164	65,013	95,38%
ML1M	1,000,209	6,040	3,706	95,53%
ML100k	100,000	943	1,682	93,7%
ML1M GTE5	226,310	6,014	3,232	98,84%
Netflix(1000 users)	216,329	1,000	10,825	98%
Epinions (10k users)	1,016,915	10,000	365,248	99,97%

Table 1: Description of datasets

In our experiments we use *incrementalRecall*@10. Since we experiment with two different kinds of feedback, we use two different sorting orders in the procedure of measuring recall. For the explicit rating feedback we sort the predictions with respect to the predicted rating value. For positive only feedback we assume a rating of 1 for existence of feedback. Therefore, the predictions are sorted with respect to proximity of the prediction to 1.

5. EXPERIMENTS

In our experiments we use two types of datasets. The first type encompasses data with explicit rating feedback: MovieLens 1M and 100k[1], a sample of 10 000 users from the extended Epinions [8] dataset and a sample of 1000 users from the Netfilx dataset. In this type of datasets our selection is limited, because many forgetting strategies require timestamp information. Table 1 provides some details on those datasets.

The second type of datasets consist positive-only feedback. Those datasets consist of a chronologically ordered set of user-item pairs in the form (u, i). Music-listen consists of music listening events, where each pair corresponds to a music track 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 [2] – LastFM-600k – and a binarized version of MovieLens 1M dataset that we call ML1M-GTE5 hereafter. In ML1M-GTE5 we assumed that a rating value of 5 indicates a positive feedback. All remaining ratings have been removed and considered as negative feedback. More information on those datasets is also presented in Table 1.

All of our forgetting strategies require a setting of exactly one parameter. Their performance depends strongly on the correct setting. To determine the approximately optimal value of the parameters we use a grid search over the parameter space. The remaining parameters we set to the following values: number of dimensions = 40, learn rate η = 0.003 and regularization parameter λ = 0.01. To show the benefit of using forgetting strategies, we compare their results against a baseline method – the same matrix factorization algorithm with no forgetting strategy.

In Figure 2 we show our results on four datasets with explicit rating feedback. The corresponding numerical values

[1]http://www.movielens.org/
[2]http://www.palcoprincipal.com
[3]http://last.fm

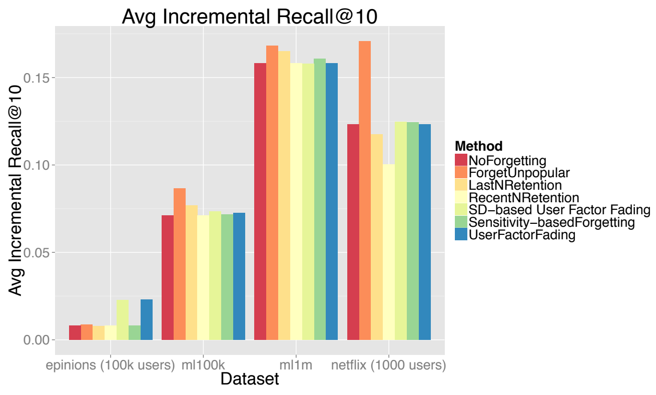

Figure 2: Best results achieved by each method with respect to the average incremental recall@10 using *rating feedback*, grouped by dataset. Higher results are better

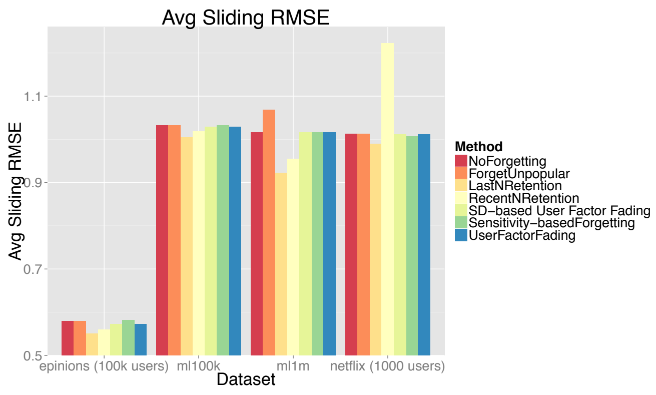

Figure 3: Best results achieved by each method with respect to the average incremental RMSE using *rating feedback*, grouped by dataset. Lower results are better.

are presented in Table 2. The entries in the Table are sorted with respect to the average incremental recall@10. Incremental recall, same as sliding RMSE, are online measures that monitor the performance over time. Hence, behind each of the results stands a curve of values representing the corresponding evaluation measure over time. However, due to the space constraints and for an easier comparison, we present only average values. On all four datasets our new strategies took the top positions. On the ML1M, Netflix and ML100k datasets the *ForgetUnpopular* strategy yielded the best result. On the Epinions dataset also one of our new forgetting strategies was the best, this time the *UserFactorFading* strategy. Here, also the *UserFactorFading* strategy was very successful.

For this type of user feedback we also provide measures with average sliding RMSE (cf. Figure 3). Opinions diverge about the importance of those two evaluation measures. Therefore, we provide both of them. However, in evaluation of top-N recommenders more importance is attributed to recall-based measures. In our evaluation methodology that we adopted from [3], measuring precision is not necessary, since it can be derived from the recall directly.

With respect to the average sliding RMSE, forgetting strategies also show an advantage (cf. Fig 3) compared to the baseline (no forgetting). Here, one of the strategies by Matuszyk and Spiliopoulou [9], *LastNRetention*, yielded the best results. Nevertheless, many of our new strategies also performed better than the baseline.

Forgetting Strategy	**Parameter**	**Avg. Incremental Recall@10**
ML1M		
ForgetUnpopular	1.02	**0.168368003**
LastNRetention	10	0.1651057374
Sensitivity-basedForgetting	0.5	0.1606939037
GlobalSensitivity-basedForgetting	0.01	0.1597336372
RecentNRetention	24h	0.1581627191
NoForgetting	–	0.1581627191
UserFactorFading	0.999999	0.1581627191
SD-based User Factor Fading	1.000001	0.1578246763
Netflix (sample 1000 users)		
ForgetUnpopular	1.1	**0.1707896384**
SD-based User Factor Fading	1.001	0.1246312781
Sensitivity-basedForgetting	0.1	0.1242929253
GlobalSensitivity-basedForgetting	0.1	0.1233110233
UserFactorFading	0.99999999	0.1232758314
NoForgetting	–	0.1232758314
LastNRetention	40	0.1175009409
RecentNRetention	1h	0.1002589166
ML100k		
ForgetUnpopular	1.1	**0.0865524865**
LastNRetention	10	0.076753316
SD-based User Factor Fading	1.001	0.0733544253
UserFactorFading	0.999	0.0725157144
Sensitivity-basedForgetting	1.35	0.0717012517
GlobalSensitivity-basedForgetting	0.7	0.0710979034
NoForgetting	–	0.0710803956
RecentNRetention	24h	0.0710803956
Epinions Extended (sample 10k users)		
UserFactorFading	0.5	**0.0230770405**
SD-based User Factor Fading	1.2	0.0226747025
ForgetUnpopular	4	0.0086745416
GlobalSensitivity-basedForgetting	0.5	0.0082184146
NoForgetting	–	0.0082184146
Sensitivity-basedForgetting	1.001	0.0082184146
RecentNRetention	24h	0.0081456019
LastNRetention	20	0.0077852762

Table 2: Best results of each method using *rating feedback* sorted w.r.t. avg. incremental recall@10 and grouped by dataset. Higher results are better. Best results in bold face.

In Figure 4 and Table 3 we present our results on datasets with positive-only user feedback. On the LastFM-600k dataset the *SD-based User Factor Fading* strategy performed the best, followed by *UserFactorFading*, both of which outperformed the baseline by a factor of approximately 2. On ML1M-GTE5 the strategies *ForgetUnpopular* and *SD-based User Factor Fading* clearly took the top positions. On Music-playlist dataset the *LastNRetention* strategy performed the best. The most extreme gain through forgetting methods is visible on the Music-listen dataset. The *SD-based User Factor Fading* strategy performed almost five times better than the baseline method. It reached an average incremental recall value of 0.177 as compared to 0.036 by the baseline. The *UserFactorFading* strategy has also shown a strong performance on this dataset.

For datasets with positive-only feedback we do not provide RMSE measures, since all ratings in those datasets are equal to one. Measuring prediction error in form of RMSE in this scenario would not be meaningful.

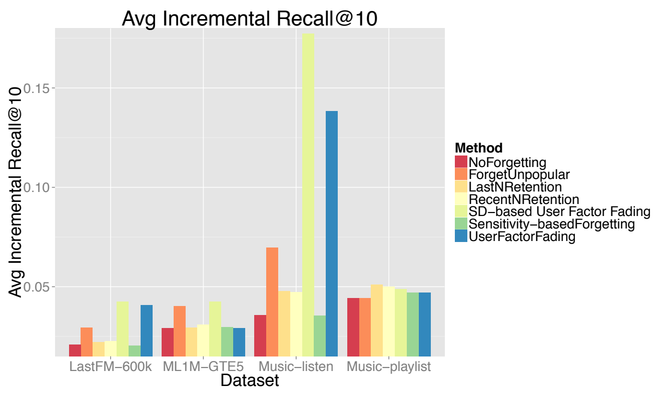

Figure 4: Best results achieved by each method with respect to the average incremental recall@10 using *positive-only feedback*, grouped by dataset. Higher results are better.

6. CONCLUSIONS

We investigated the problem of forgetting information in recommender systems for incremental matrix factorization. The goal of our research is to make recommender systems more adaptive to changes of users' preferences and of the environment. Those changes are innate to learning users' preferences, as they are subject to influence from outside and to intrinsic evolution of taste and interests. Our approach to improve the adaptivity of recommenders is by forgetting outdated information.

We developed five new forgetting strategies for incremental matrix factorization that heuristically distinguish between the current and outdated information in form of ratings. We extended a state-of-the-art algorithm, BRISMF [14], for usage of our strategies. We performed numerous experiments on eight datasets with two types of user feedback. To our knowledge, this is the first work that consolidates the validity of a recommendation algorithm on both types of user feedback: explicit ratings and positive-only feedback. We used two different evaluation measures for stream-based recommender systems, sliding RMSE and incremental recall. On all of those datasets we have shown that our forgetting strategies improve the predictive power of a recommender.

Our *ForgetUnpopular* strategy was particularly successful on datasets with explicit user feedback in form of ratings. Experiments with positive-only feedback have shown that on this type of data the *SD-based User Factor Fading* strategy yields extreme improvements. On the Music-listen dataset the result of this strategy was better than the baseline with no forgetting by a factor of nearly five.

An improvement of results by using forgetting techniques also proves that users' preferences are subject to concept drift. This is one of main challenges in stream mining that, as we have proven, also applies to recommender systems.

Since our forgetting strategies can differentiate between outdated and current information, they can also be used as

Forgetting Strategy	Parameter	Avg. Incremental Recall@10
Music-playlist		
LastNRetention	10	**0.0510906661**
RecentNRetention	3h	0.0499863333
SD-based User Factor Fading	1.01	0.0487104038
UserFactorFading	0.99	0.0471174833
Sensitivity-basedForgetting	0.1	0.0471037842
ForgetUnpopular	100	0.0443777217
NoForgetting	–	0.0443777217
GlobalSensitivity-basedForgetting	0.1	0.0443142059
LastFM-600k		
SD-based User Factor Fading	1.04	**0.0424265769**
UserFactorFading	0.99	0.040675639
ForgetUnpopular	1.4	0.0295259066
RecentNRetention	24h	0.0226743311
LastNRetention	10	0.0221790142
NoForgetting	–	0.0208641556
GlobalSensitivity-basedForgetting	0.1	0.0206361797
Sensitivity-basedForgetting	1	0.0202472756
ML1M-GTE5		
SD-based User Factor Fading	1.1	**0.0424313423**
ForgetUnpopular	1.01	0.0402411073
RecentNRetention	3h	0.0310388703
GlobalSensitivity-basedForgetting	0.5	0.0303556332
Sensitivity-basedForgetting	1.001	0.0296813349
LastNRetention	10	0.0294485924
UserFactorFading	0.99999999	0.0290675976
NoForgetting	–	0.0290675976
Music-listen		
SD-based User Factor Fading	1.2	**0.177245787**
UserFactorFading	0.99	0.138297306
ForgetUnpopular	4	0.0695838741
LastNRetention	20	0.0476573054
RecentNRetention	1h	0.0473491079
NoForgetting	–	0.0356019772
Sensitivity-basedForgetting	1.001	0.0354992912
GlobalSensitivity-basedForgetting	0.5	0.0342442817

Table 3: Best results of each method using *positive-only feedback*, sorted w.r.t. avg. incremental recall@10 and grouped by dataset. Higher results are better. Baseline in bold face.

change detectors for recommender systems. To our knowledge, none of change detection algorithms known from the stream mining domain have been successfully adapted to the recommender system domain. Our immediate next step is to investigate the eligibility of our strategies as change detectors.

Acknowledgements

This work is partially supported by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-037281 and by the European Commission through the project MAESTRA (Grant number ICT-2013-612944). The second author's work is supported by the FCT with grant SFRH/BD/77573/2011.

7. REFERENCES

[1] M.W. Berry, S.T. Dumais, and G.W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM review*, pages 573–595, 1995.

[2] O. Celma. *Music Recommendation and Discovery in the Long Tail.* Springer, 2010.

[3] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[4] 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.

[5] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *KDD*, 2009.

[6] Xue Li, Jorge M. Barajas, and Yi Ding. Collaborative filtering on streaming data with interest-drifting. *Intell. Data Anal.*, 11(1):75–87, 2007.

[7] 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.

[8] Paolo Massa and Paolo Avesani. Trust-aware bootstrapping of recommender systems. In *ECAI Workshop on Recommender Systems*, pages 29–33. Citeseer, 2006.

[9] Pawel Matuszyk and Myra Spiliopoulou. Selective Forgetting for Incremental Matrix Factorization in Recommender Systems. In Sašo Džeroski, Panče Panov, Dragi Kocev, and Ljupčo Todorovski, editors, *Discovery Science*, volume 8777 of *Lecture Notes in Computer Science*, pages 204–215. Springer International Publishing, 2014.

[10] Catarina Miranda and Alípio Mário Jorge. Incremental Collaborative Filtering for Binary Ratings. In *Web Intelligence Conference Proceedings*, pages 389–392. IEEE Computer Society, 2008.

[11] Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. Incremental Collaborative Filtering for Highly-Scalable Recommendation Algorithms. In Mohand-Said Hacid, Neil V. Murray, Zbigniew W. Ras, and Shusaku Tsumoto, editors, *ISMIS 2005, Proceedings*, volume 3488 of *Lecture Notes in Computer Science*, pages 553–561. Springer, 2005.

[12] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07.*

[13] 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.

[14] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable Collaborative Filtering Approaches for Large Recommender Systems. *J. Mach. Learn. Res.*, 10, 2009.

[15] João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18(4):271–282, 2012.

[16] João Vinagre, Alípio Mário Jorge, and João Gama. Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback. In Vania Dimitrova, Tsvi Kuflik, David Chin, Francesco Ricci, Peter Dolog, and Geert-Jan Houben, editors, *UMAP*, volume 8538 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.