Selective Forgetting for Incremental Matrix Factorization in Recommender Systems

Pawel Matuszyk and Myra Spiliopoulou

Otto-von-Guericke-University Magdeburg,
Universitätsplatz 2,
D-39106 Magdeburg, Germany
{pawel.matuszyk,myra}@iti.cs.uni-magdeburg.de

Abstract. Recommender Systems are used to build models of users' preferences. Those models should reflect current state of the preferences at any timepoint. The preferences, however, are not static. They are subject to concept drift or even shift, as it is known from e.g. stream mining. They undergo permanent changes as the taste of users and perception of items change over time. Therefore, it is crucial to select the actual data for training models and to forget the outdated ones.
The problem of selective forgetting in recommender systems has not been addressed so far. Therefore, we propose two forgetting techniques for incremental matrix factorization and incorporate them into a stream recommender. We use a stream-based algorithm that adapts continuously to changes, so that forgetting techniques have an immediate effect on recommendations. We introduce a new evaluation protocol for recommender systems in a streaming environment and show that forgetting of outdated data increases the quality of recommendations substantially.

Keywords: Forgetting Techniques, Recommender Systems, Matrix Factorization, Sliding Window, Collaborative Filtering

1 Introduction

Data sparsity in recommender systems is a known and thoroughly investigated problem. A huge number of users and items together with limited capabilities of one user to rate items result in a huge data space that is to a great extent empty. However, the opposite problem to the data sparsity has not been studied extensively yet. In this work we investigate, whether recommender systems suffer from too much information about selected users. Although, the most algorithms for recommender systems try to tackle the problem of an extreme data sparsity, we show that it is beneficial to forget some information and not consider it for training models any more. Seemingly, forgetting information exacerbates the problem of having not enough data. We show, however, that much of the old information does not reflect the current preferences of users and training models upon this information decreases the quality of recommendations.

Reasons for the information about users being outdated are manifold. Users' preferences are not static - they change over time. New items emerge frequently,

depending on the application scenario, e.g. news in the internet. Also the perception of existing items changes due to external factors such as advertisement, marketing campaigns and events related to the items. The environment of a recommender system is dynamic. A recommender system that does not take those changes into account and does not adapt to them deteriorates in quality. Retraining of a model does not help, if the new model is again based on the outdated information. Consequently, the information that a recommender is trained upon has to be selected carefully and the outdated information should be forgotten.

Since a recommender should be able to adapt constantly to changes of the environment, ideally in the real time, in our work we use an incremental, stream-based recommender. It does not learn upon a batch of ratings, but it considers them as a stream, as it is known from e.g. stream mining. Incremental methods have the advantage of learning continuously as new ratings in the stream arrive and, therefore, are always up to date with the current data. The batch-based methods on the other hand use a predefined batch of ratings to train the model and are, after arrival of new ratings, constantly out of date. Our method that uses matrix factorization still requires a retraining of latent item factors. However, the latent user factors are kept up to date constantly between the retraining phases. Also, since the general perception of items changes slower than preferences of a single user, the retraining is not needed as frequently as in the case of batch learners. A further essential advantage of incremental methods is that they can adapt immediately as changes occur. Because an incremental recommender learns upon ratings as soon as they arrive, it can react to changes immediately. Hence, it can capture short term changes, whereas a batch learner has to wait for the next retraining phase to adapt to changes.

Gradual changes in users' preferences and changes in item perception speak in favour of forgetting the outdated information in recommender systems. This type of changes can be related to concept drift in stream mining. It describes slow and gradual changes. There is also a second type of changes called concept shift. These changes are sudden, abrupt and unpredictable. In recommender systems those changes can be related e.g. to situations, where multiple persons share an online account. If we consider an online shop scenario, a recommender would experience a concept shift, when the owner of an account buys items for a different person (e.g. presents). When recommending movies a person can be influenced by preferences of other people, which can be a short-lived, single phenomenon, but it also can be a permanent change. In both cases a successful recommender system should adapt to those changes. This can be achieved by forgetting the old outdated information and learning a model based on information that reflects the current user preferences more accurately. In summary the contribution of our work is threefold: 1) We propose two selective forgetting techniques for incremental matrix factorization. 2) We define a new evaluation protocol for stream-based recommender systems. 3) We show that forgetting selected ratings increases the quality of recommendations.

This paper is structured as follows. In section 2 we discuss related work we used in our method, stressing the differences to existing approaches. Section 3

explains our forgetting mechanisms. The experimental settings and evaluation protocol are described in Section 4. Our results are explained in Section 5. Finally, in section 6, we conclude our work and discuss open issues.

2 Related Work

Recommender systems gained in popularity in recent years. The most widely used category of recommender systems are collaborative filtering (CF) methods. An intuitive, item-based approach in CF has been published in 2001 [6]. Despite its simplicity this method based on neighbourhoods of items has shown to have a strong predictive power. In contrast to content-based recommenders, CF works only with user feedback and without any additional information about users or items. Those advantages as well as the ability to cope with extremely sparse data made CF a highly interesting category of algorithms among practitioners and researchers. Consequently, many extensions of those methods have been developed. A comprehensive survey on those methods can be found in [1].

In 2012 Vinagre and Jorge noticed the need for forgetting mechanisms in recommender systems and proposed forgetting techniques for neighbourhood-based methods [9]. They introduced two forgetting techniques: sliding window and fading factors, which are also often used in stream mining. They also considered a recommender system as a stream-based algorithm and used those two techniques to define which information was used for computing a similarity matrix. According to the sliding window technique only a predefined number of the most recent user sessions was used for calculating the similarity matrix making sure that only the newest user feedback is considered for training a model. Their second technique, fading factors, assigns lower weight to old data than to new ones and, thereby, diminishes the importance of potentially outdated information. In our method we also use the sliding window technique, there are, however, three fundamental differences to Vinagre and Jorge: 1) Our method has been designed for explicit feedback e.g. ratings, whereas the method in [9] was designed for positive-only feedback. 2) We propose forgetting strategies for matrix factorization algorithms as opposed to neighbourhood-based methods in [9]. 3) Vinagre and Jorge apply forgetting on a stream of sessions of *all users*. Our forgetting techniques are *user-specific* i.e. we consider ratings of one user as a stream and apply a sliding window *selectively* on it. Vinagre and Jorge have shown that non-incremental algorithms using forgetting have lower computational requirements without a significant reduction of the predictive power, when compared to the same kind of algorithms without forgetting.

Despite the popularity of the neighbourhood-based methods, the state-of-the-art algorithms for recommenders are matrix factorization algorithms. They became popular partially due to the Netflix competition, where they showed a superior predictive performance, competitive computational complexity and high extensibility. Koren et. al proposed a matrix factorization method based on gradient descent [3], [4], where the decomposition of the original rating matrix is computed iteratively by reducing prediction error on known ratings. In the

method called "TimeSVD++" Koren et al. incorporated time aspects accounting for e.g. changes in user preferences. Their method, however, does not encompass any forgetting strategy i.e. it always uses all available ratings no matter, if they are still representative for users' preferences. Additionally, some of the changes of the environment of a recommender cannot be captured by time factors proposed by Koren et. al. To this category of changes belong the abrupt, non-predictable changes termed before as concept shift. Furthermore, the method by Koren et. al is not incremental, therefore it cannot adapt to changes in real time.

An iterative matrix factorization method has been developed by Takács et. al in [8]. They termed the method biased regularized incremental simultaneous matrix factorization (BRISMF). The basic variant of this method is also batch-based. Takács et. al, however, proposed an incremental variant of the algorithm that also uses stochastic gradient descent. In this variant the model can be adapted incrementally as new ratings arrive. The incremental updates are carried out by fixating the latent item factors and performing further iterations of the gradient descent on the user latent factors. This method still requires an initial training and an eventual retraining of the item factors, but the latent user factors remain always up to date. In our work we use the BRISMF algorithm and extend it by forgetting techniques.

3 Method

Our method encompasses forgetting techniques for incremental matrix factorization. We incorporated forgetting into the algorithm BRISMF by Takács et. al [8]. The method is general and can be applied to any matrix factorization algorithm based on stochastic gradient descent analogously. BRISMF is a batch learning algorithm, the authors, however, proposed an incremental extension for retraining user features (cf. Algorithm 2 in [8]). We adopted this extension to create a forgetting, stream-based recommender. Our recommender system still requires an initial training, which is the first of its two phases.

3.1 Two Phases of Our Method

Phase I - Initial Training creates latent user and items features using the basic BRISMF algorithm in its unchanged form [8]. It is a pre-phase for the actual stream-based training. In that phase the rating matrix R is decomposed into a product of two matrices $R \approx PQ$, where P is a latent matrix containing user features and Q contains latent item vectors. For calculating the decomposition stochastic gradient descent (SGD) is used, which requires setting some parameters that we introduce in the following together with the respective notation.

As an input SGD takes a training rating matrix R and iterates over ratings $r_{u,i}$ for all users u and items i. SGD performs multiple runs called epochs. We estimate the optimal number of epochs in the initial training phase and use it later in the second phase. The results of the initial phase are the matrices P

and Q. As p_u we denote hereafter the latent user vector from the matrix P. Analogously, q_i is a latent item vector from the matrix Q. Those latent matrices serve as input to our next phase. The vectors p_u and q_i are of dimensionality k, which is set exogenously. In each iteration of SGD within one epoch the latent features are adjusted by a value depending on the learning rate η according to the following formulas [8]:

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

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

To avoid overfitting long latent vectors are penalized by a regularization term controlled by the variable λ. As $\overrightarrow{r}_{u*}$ we denote a vector of all ratings provided by the user u. For further information on the initial algorithm we refer to [8].

Despite the incremental nature of SGD, this phase, as also the most of matrix factorization algorithms, is a batch algorithm, since it uses a whole training set at once and the evaluation is performed after the learning on the entire training set has been finished. In our second phase evaluation and learning take place incrementally same as e.g. in stream mining.

Phase II - Stream-based Learning After the initial training our algorithm changes into a streaming mode, which is its main mode. From this time point it adapts incrementally to new users' feedback and to potential concept drift or shift. Also the selective forgetting techniques are applied in this mode, where they can affect the recommendations immediately. Differently from batch learning, evaluation takes place iteratively before the learning of a new data instance, as it is known from stream mining under the name "prequential evaluation" [2]. We explain our evaluation settings more detailed in Section 4.

In Algorithm 1 is pseudo-code of our method, which is an extension and modification of the algorithm presented in [8]. This code is executed at arrival of a new rating, or after a predefined number n of ratings. A high value of n results in a higher performance in terms of computation time, but also in a slower adaptation to changes. A low n means that the model is updated frequently, but the computation time is higher. For our experiments we always use $n = 1$.

The inputs of the algorithm are results of the initial phase and parameters that we also defined in the previous subsection. When a new rating $r_{u,i}$ arrives, the algorithm first makes a prediction $\widehat{r}_{u,i}$ for the rating, using the item and user latent vectors trained so far. The deviation between $\widehat{r}_{u,i}$ and $r_{u,i}$ is then used to update an evaluation measure (cf. line 4 in Algorithm 1). It is crucial to perform an evaluation of the rating prediction first, before the algorithm uses the rating for updating the model. Otherwise the separation of the training and test datasets would be violated. In line 6 the new rating is added to the list of ratings provided by the user u. From this list we *remove* the outdated ratings using one of our forgetting strategies (cf. line 7). The forgetting strategies are described in Section 3.2. In the line 9 SGD starts on the newly arrived rating.

It uses the optimal number of epochs estimated in the initial training. Contrary to the initial phase, here only user latent features are updated. For updating the user features the SGD iterates over all ratings of the corresponding user that *remained* after a forgetting technique has been applied. For the update of each dimension k the formula in line 16 is used.

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: **applyForgetting**$(\overrightarrow{r}_{u*})$ **//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: $predictionError = r_{u,i} - \overrightarrow{p_u} \cdot \overrightarrow{q_i}$
15: **for all** latent dimensions $k \neq 1$ in $\overrightarrow{p_u}$ **do**
16: $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
17: **end for**
18: **end for**
19: **end while**

Our variant of the incremental BRISMF method has the same complexity as the original, incremental BRISMF. In terms of computation time, it performs even better, since the number of ratings that the SGD has to iterate over is lower due to our forgetting technique. The memory consumption of our method is, however, higher, since the forgetting is based on a sliding window (cf. Section 3.2) that has to be kept in the main memory.

3.2 Forgetting Techniques

Our two forgetting techniques are based on a sliding window over data instances i.e. in our case over ratings. Ratings that enter the window are incorporated into a model. Since the window has a fixed size, some data instances have to leave it, when new ones are incorporated. Ratings that leave the window are forgotten and their impact is removed from the model. The idea of sliding window has been used in numerous stream mining algorithms, especially in a stream-based classification e.g. in Hoeffding Trees. In stream mining the sliding window is, however, defined over the entire stream. This approach has also been chosen

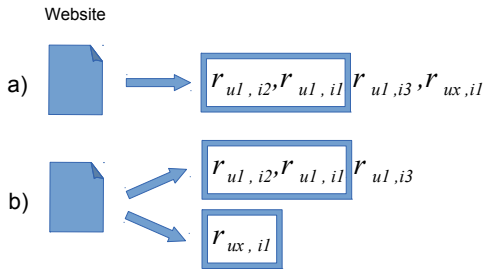


Fig. 1: Conventional definition of a sliding window a) vs. a *user-specific* window b). In case a) information on some users is forgotten entirely and no recommendations are possible (e.g. for user u_x). In case b) only users with too much information are affected (e.g. u_2). Ratings of new users, such as u_x, are retained.

by Vinagre and Jorge in [9]. Our approach is user-specific i.e. a virtual sliding window is defined for each user separately. Figure 1 illustrates this difference.

On the left side of the figure there is a website that generates streams of ratings by different users. The upper part a) of the figure represents a conventional definition of a sliding window (blue frame) over an entire stream. In this case all ratings are considered as one stream. In our example with a window of size 2 this means that in case a) the model contains the ratings $r_{u1,i2}$ and $r_{u1,i1}$. All remaining ratings that left the window have been removed from the model. This also means that all ratings by the user u_x have been forgotten. Consequently, due to the cold start problem, no recommendations for that user can be created. Case b) represents our approach. Here each user has his/her own window. In this case all ratings of the user u_x are retained. Only users, who provided more ratings than the window can fit, are affected by the forgetting (e.g. u_1). User with very little information are retained entirely. Due to the user-specific forgetting the cold start problem is not exacerbated. The size of the window can be defined in multiple ways. We propose two implementations of the *applyForgetting()* function from Algorithm 1, but further definitions are also possible.

Instance-based Forgetting The pseudo code in Algorithm 2 represents a simple forgetting function based on the window size w. In Algorithm 1 new ratings are added into the list of user's ratings $r_{u,*}$. If due to that the window grows above the predefined size, the oldest rating is removed as many times as needed to reduce it back to the size w.

Time-based Forgetting In certain application scenarios it is reasonable to define current preferences with respect to time. For instance, we can assume that after a few years preferences of a user have changed. In very volatile applications a time span of one user session might be reasonable. Algorithm 3 implements a

Algorithm 2 applyForgetting($r_{u,*}$) - Instance-based Forgetting

Input: $r_{u,*}$ a list of ratings by user u sorted w.r.t. time, w - window size
1: **while** $|r_{u,*}| > w$ **do**
2: removeFirstElement($r_{u,*}$)
3: **end while**

forgetting function that considers a threshold a for the age of user's feedback. In this implementation the complexity of forgetting is less than O(w), where w is size of the window, since it does not require a scan over the entire window.

Algorithm 3 applyForgetting($r_{u,*}$) - Time-based Forgetting

Input: $r_{u,*}$ a list of ratings by user u sorted w.r.t. time, a - age threshold
1: forgettingApplied ← true
2: **while** forgettingApplied == true **do**
3: *oldestElement* ← getFirstElement($r_{u,*}$) //the oldest rating
4: **if** $age(oldestElement) > a$ **then**
5: removeFirstElement($r_{u,*}$)
6: forgettingApplied ← true
7: **else**
8: forgettingApplied ← false
9: **end if**
10: **end while**

4 Evaluation Setting

We propose a new evaluation protocol for recommender systems in a streaming environment. Since our method requires an initial training, the environment of our recommender is not entirely a streaming environment. The evaluation protocol should take the change from the batch mode (for initial training) into streaming mode (the actual method) into account.

4.1 Evaluation Protocol

Figure 2 visualizes two modes of our method and how a dataset is split between them. The initial training starts in a batch mode, which corresponds to the part 1) in the Figure (batch train). For this part we use 30% of the dataset. The ratios are example values that we used in our experiments, but they can be adjusted to the idiosyncrasies of different datasets. The gradient descent used in the initial training iterates over instances of this dataset to adjust latent features. The adjustments made in one epoch of SGD are then evaluated on the batch test dataset (part 2). After evaluation of one epoch the algorithm decides, if further

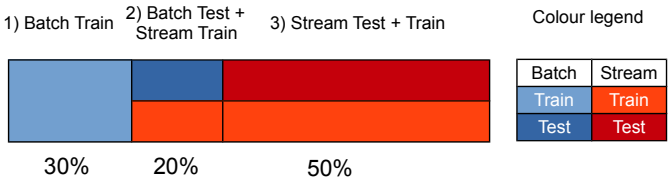


Fig. 2: Visualization of two modes of our method and split between the training and test datasets. The split ratios are example values.

epochs are needed. After the initial phase is finished the latent features serve as input for the streaming mode.

For the stream based evaluation we use the setting proposed by Gama et al. called prequential evaluation [2]. In this setting ratings arrive sequentially in a stream. To keep the separation of a test and training dataset every rating is first predicted and the prediction is evaluated before it is used for training. This setting corresponds to part 3) of our Figure. Two different colours symbolize that this part is used both for training and evaluation. This also applies to part 2) of the figure. Since the latent features have been trained on part 1) and the streaming mode starts in part 3) this would mean a temporal gap in the training set. Since temporal aspects play a big role in forgetting we should avoid it. Therefore, we also train the latent features incrementally on part 2). Since this part has been used for evaluation of the batch mode already, we do not evaluate the incremental model on it. The incremental evaluation starts on part 3).

The incremental setting also poses an additional problem. In a stream new users can occur, for whom no latent features in the batch mode have been trained. In our experiments we excluded those users. The problem of inclusion of new users into a model is subject to our future work.

4.2 Evaluation Measure - *slidingRMSE*

A popular evaluation measure is the root mean squared error (RMSE), which is based on the deviation between a predicted and real rating [7]:

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (r_{u,i} - \widehat{r}_{u,i})^2} \tag{3}$$

where T is a test set. This evaluation measure was developed for batch algorithms. It is a static measure that does not allow to investigate, how the performance of a model changes over time. We propose *slidingRMSE* - a modified version of RMSE that is more appropriate for evaluating stream recommenders. The formula for calculating *slidingRMSE* is the same as for RMSE, but the test set T is different. *slidingRMSE* is not calculated over the entire test set, but

only over a sliding window of the last n instances. Prediction error of ratings that enter the sliding window are added to the squared sum of prediction errors and the ones that leave it are subtracted. The size of the window n is independent from the window size for forgetting techniques. A small n allows to capture short-lived effects, but it also reveals a high variance. A high value of n reduces the variance, but it also makes short-lived phenomena not visible. For our experiments we use $n = 500$. *slidingRMSE* can be calculated at any timepoint in a stream, therefore, is is possible to evaluate how RMSE changes over time.

Since we are interested in measuring how the forgetting techniques affect the prediction accuracy, we measure the performance of an algorithm with and without forgetting, so that the difference can be explained only by application of our forgetting techniques. Forgetting is applied only on a subset of users, who have sufficiently many ratings. Consequently, all other users are treated equally by both variants of the algorithm. Thus, we measure *slidingRMSE* only on those users, who were treated differently by the forgetting and non-forgetting variants.

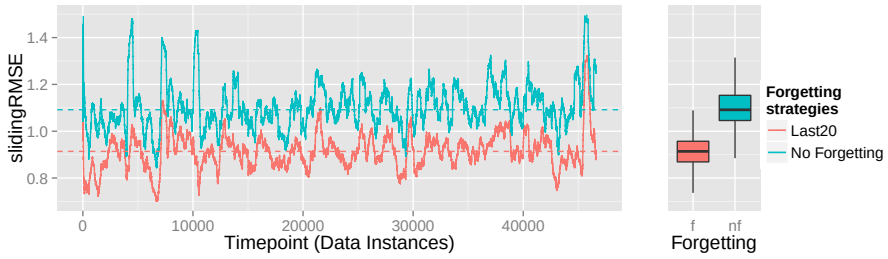
5 Experiments

We performed our experiments on four real datasets: Movielens 1M[1], Movielens 100k, Netflix (a random sample of 1000 users) and Epinions (extended) [5]. The choice of datasets was limited by the requirement of our method to have timestamped data. In all experiments we used our modified version of the BRISMF algorithm [8] with and without forgetting. Since BRISMF requires a few parameters to be set, on each dataset we performed a grid search over the parameter space to find the approximately optimal parameter setting. In Figure 3 we present the results of the best parameter settings found by the grid search on each dataset. As an evaluation measure we used *slidingRMSE* (lower values are better). The left part of the diagrams represents the *slidingRMSE* measured over time. The red curves represent our method with forgetting technique denoted in the legend. "Last20" stands for an instance-based forgetting, when only 20 last ratings of a user are retained. The best results were achieved constantly by the instance-based forgetting. Therefore, the time-based forgetting is not present here. Blue curves represent the method without forgetting. The box plots on the right side are centred around the median of *slidingRMSE*. They visualize the distribution of *slidingRMSE* in a simplified way. Please, consider that box plots

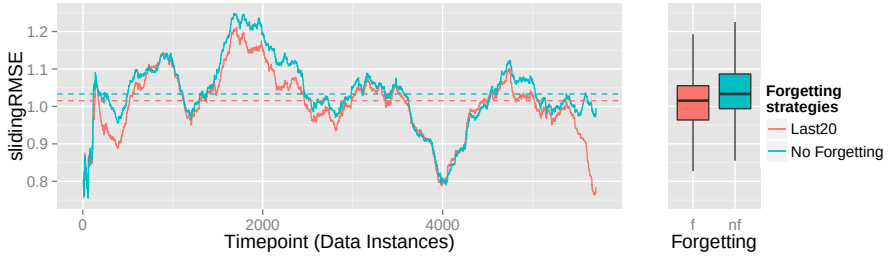
[1] `http://www.movielens.org/`

Dataset	ML1M	ML100k	Epinions	Netflix
avg. slidingRMSE - Forgetting	**0.9151**	**1.0077**	**0.6627**	**0.9138**
avg. slidingRMSE - NO Forgetting	1.1059	1.0364	0.8991	1.0162

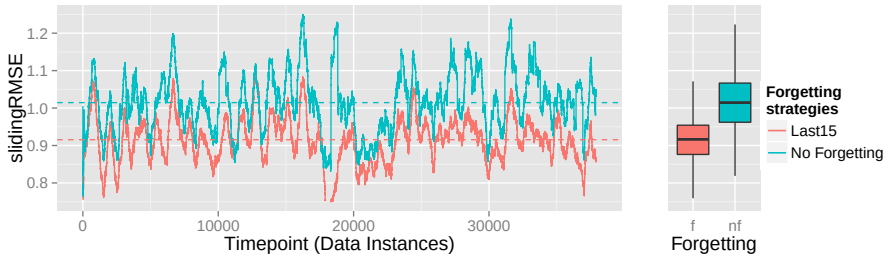
Table 1: Average values of *slidingRMSE* for each dataset (lower values are better). Our forgetting strategy outperforms the non-forgetting strategy on all datasets.



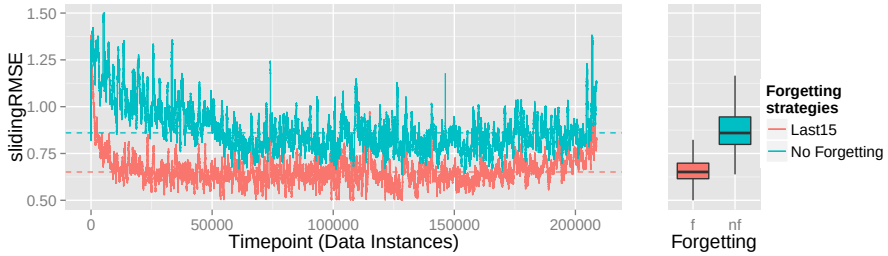
(a) Movielens 1M



(b) Movielens 100k



(c) Netflix (random sample of 1000 users)



(d) Epinions extended

Fig. 3: *SlidingRMSE* on four real datasets with and without forgetting (lower values are better). Application of forgetting techniques yields an improvement on all datasets at nearly all timepoints.

are normally used for visualizing independent observations, this is, however, not the case here. From Figure 3 we see that our method with forgetting dominates the non-forgetting strategy on all datasets at nearly all timepoints. In Table 1 we also present numeric, averaged values of *slidingRMSE* for each dataset.

6 Conclusions

In this work we investigated, whether selective forgetting techniques for matrix factorization improve the quality of recommendations. We proposed two techniques, an instance-based and time-based forgetting, and incorporated them into a modified version of the BRISMF algorithm. In contrast to existing work, our approach is based on a user-specific sliding window and not on a window defined over an entire stream. This has an advantage of selectively forgetting information about users, who provided enough feedback.

We designed a new evaluation protocol for stream-based recommenders that also takes the initial training and temporal aspects into account. We introduced an evaluation measure, *slidingRMSE*, that is more appropriate for evaluating recommender systems over time and capturing also short-lived phenomena. In experiments on real datasets we have shown that a method that uses our forgetting techniques, outperforms the non-forgetting strategy on all datasets at nearly all timepoints. This also proves that user preferences and perception of items change over time. We have shown that it is beneficial to forget the outdated user feedback despite the extreme data sparsity known in recommenders.

In our future work we plan to develop more sophisticated forgetting strategies for recommender systems. Our immediate next step is also a research on a performant inclusion of new users into an existing, incremental model.

References

1. C. Desrosiers and G. Karypis. A Comprehensive Survey of Neighborhood-based Recommendation Methods. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 107–144. Springer US.
2. J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *KDD*, 2009.
3. Y. Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.
4. Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, Aug. 2009.
5. P. Massa and P. Avesani. Trust-aware bootstrapping of recommender systems. In *ECAI Workshop on Recommender Systems*, pages 29–33. Citeseer, 2006.
6. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, WWW '01, 2001.
7. G. Shani and A. Gunawardana. Evaluating Recommendation Systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*.
8. G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable Collaborative Filtering Approaches for Large Recommender Systems. *J. Mach. Learn. Res.*, 10, 2009.
9. J. Vinagre and A. M. Jorge. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18(4):271–282, 2012.