Instance Selection for Online Updating in Dynamic Recommender Environments

Thilina Thanthriwatta[1(✉)][0000−0002−4445−8721] and David S. Rosenblum[1,2][0000−0003−1685−4206]

[1] Department of Computer Science, National University of Singapore, Singapore
e0001932@u.nus.edu
[2] Department of Computer Science, George Mason University, VA, USA
dsr@gmu.edu

Abstract. Online recommender systems continuously learn from user interactions that occur in a streaming manner. A fundamental challenge of online recommendation is to select important instances (i.e., user interactions) for model updates to achieve higher prediction accuracy while omitting noisy instances. In this paper, we study (1) how to select the *best* instances and (2) how to effectively utilize the selected instances in dynamic recommender environments. We present two instance selection strategies based on Self-Paced Learning and rating profiles. We integrate them with Factorization Machines to perform online updates. Moreover, we study the impact of contextual information in online updating. We conducted experiments on a real-world check-in dataset, which contains temporal contextual features. Empirical results demonstrate that our instance selection strategies effectively balance the trade-off between prediction accuracy and efficiency.

Keywords: Instance selection · Context-aware recommender systems · Online recommender systems

1 Introduction

Context-Aware Recommender Systems (CARSs) have gained significant interest with the rise of smart devices such as smartphones that employ physical sensors and applications to capture contextual information (e.g., location and time). Contrary to conventional Recommender Systems (RSs), which utilize information about users and items only, CARSs incorporate contextual information and make the recommendation problem multi-dimensional. In the real world, CARSs should be able to adapt to the dynamic nature of a recommender environment where a stream of user interactions (i.e., instances) happens over time. Note that an event of a user loggin in and searching for items is considered as a user interaction, and CARSs should provide recommendations for each incoming user. We assume that at a given moment of time only one user interaction occurs. Most of the developed CARSs rely on typical batch learning and evaluation (batch setting). These models should be updated from scratch (re-trained) to provide

quality recommendations over time. This introduces a significantly large computational overhead and makes these models impossible to use with a high-velocity stream of interactions.

An online RS should be able to provide recommendations to the incoming users and update the necessary model parameters based on user feedback without re-training the whole model. To satisfy these conditions, the online RS should be equipped with a minimal and efficient online updating mechanism. There are two questions to answer when devising an online updating mechanism: (1) how to select the *best* instances for online updating instead of using all the incoming instances? and (2) what is the *best* strategy that can be employed for online updating while minimizing the information loss that stems from the under-utilization of the incoming instances?

Forgetting obsolete instances has been commonly used to improve efficiency in dynamic recommender environments [11,14,18]. Al-Ghossein et al. proposed to use local models to track the changes in user preference over time [1], instead of using fixed parameters such as the size of a sliding window. Matuszyka et al. [13] presented a set of forgetting methods such as forgetting unpopular items and user factor fading.

However, the existing approaches do not study how the instance selection strategies affect the context-aware recommender task. Moreover, these methods identify the "best" instances either by the changes of components (e.g., latent vectors [13], local models [1], etc.) of users and items or by comparing a set of stored ratings [18]. In contrast to these methods, we use the loss that is incurred by an instance to determine its suitability for online updating. Accordingly, we use *Self-Paced Learning* (SPL) to devise a curriculum that gradually feeds "easy" to "hard" ("complex") instances to the learning process.

Existing studies suggest that biasing towards the users and items with small rating profiles is sufficient to achieve satisfactory performance while speeding up online updating [17]. In line with this observation, we present an instance selection method named as *Profile-based Selection* (`PS`) that selects incoming instances with the users, items, and contextual feature values that have small rating profiles. On the other hand, the practitioners in machine learning advocate not to introduce "complex" instances to a training process at its initial phase since those instances do not fit well with the existing model [10]. Usually, RSs incur relatively large losses when incorporating the users and items with small rating profiles since these RSs have not been sufficiently learned based on those user behaviors and item characteristics. We use the notion of SPL for online updates by changing its pace at regular intervals. Note that the SPL concept has been introduced for strategizing batch learning, which trains the model over a large number of epochs. However, in the online recommender task, a model is updated by each incoming instance over a small number of epochs (usually one or two epochs). Thus, it is not possible to employ SPL directly for online updating. We use it as a scoring function that determines the weight of an incoming instance, and this strategy is named as *SPL-based Selection* (`SPS`). In summary, the contributions of this study are as follows:

– We present two strategies, which are integrated with Factorization Machines (FMs), to select incoming instances for online updates based on the notion of SPL and the rating profiles of users, items, and contexts.
– To the best of our knowledge, this is the first study on how to use SPL with the online recommender task, which is evaluated by the *test-then-learn* method [1,19].
– We perform a comparative analysis of how different instance selection methods behave in the presence of contextual information. Moreover, the use of selected instances for online updating leads to information loss, and our instance selection strategies especially `SPS` balance the trade-off between prediction accuracy and efficiency while minimizing the impact of information loss.

2 Instance Selection in Online Recommendation

As prior work [17] highlighted, it is essential to perform online updating for an instance that relates to a new user and/or new item since the existing model does not have any (or limited) information of new users and new items. This approach makes the online updating of a recommender model biased towards new users (items). However, this leads to suboptimal performance due to the complete removal of the users and the items that have relatively large rating profiles. Furthermore, it is possible that the incorporation of new users (items) significantly changes the existing model. On the other hand, the SPL-based training process initiates training using *easy* instances, which do not lead to large errors, and gradually feeds more complex instances.

2.1 Profile-based Selection

We begin by defining the *Profile-based Selection* (`PS`) which is based on the degree of newness. Assume that an incoming instance ρ consists of user u, item i and M number of contextual feature values $\{c_{m'}\}_1^M$ (e.g., "rainy", "weekend", "morning"). The selection of ρ is defined by the following indicator function:

$$\mathcal{D}_\rho := \mathbb{I}\left[\left(\delta(|u| + |i|) + (1-\delta)\sum_{m'=1}^{M'} |c_{m'}|\right) < \theta\right], \tag{1}$$

where $|u|$ and $|i|$ represent the number of ratings given by user u and received by item i before the arrival of ρ, respectively. $|c_{m'}|$ denotes the number of ratings in which $c_{m'}$ had already appeared. In other words, $|u|$, $|i|$ and $|c_{m'}|$ correspond to the sizes of the rating profiles of u, i and $c_{m'}$, respectively. If the weighted combination of the rating profiles of ρ is less than θ, then it can be used to update the model in the online setting. We use δ to balance the rating profiles of users and items with the rating profiles of contextual values. Because they tend to be drastically different. In a real-world context-aware recommender setting, we can observe that the number of users and items are significantly larger than

the number of contextual values. Due to this imbalance, the rating profile size of a user (an item) grow slowly compared to that of a contextual feature value over time.

2.2 SPL-based Selection

The aim of SPL is to select easy to complex training instances for robust model learning [10]. In contrast to curriculum learning [3], where the model learning is regulated by gradually including easy to complex samples based on different types of heuristics, SPL incorporates parameters into model learning to support the selection process. Zhang et al. [20] used the notion of SPL with the recommender task to improve the optimization by avoiding bad local minima. Their evaluation method is similar to the evaluation in the batch learning setting. Moreover, they used the SPL concept to improve the overall optimization while we use it to select "best" instances for online updates. Hence, the problem that was addressed by them is different from ours.

Let r_{uic} be the actual rating given by user u for item i under context c. Assume that context c corresponds to d number of contextual feature values, and j-th contextual feature value is denoted by c_j. The dataset Ω contains instances (e.g., $< u, i, c_1, ..., c_d >$), for which ratings are known. We represent feature vectors using the one-hot encoding [16]. The SPL-integrated optimization function is defined as follows:

$$\min_{\mathbf{w},\mathbf{V},\mathbf{p}} \sum_{(u,i,c)\in\Omega} p_{uic}\ell_{uic} + \lambda R(.) + \sum_{(u,i,c)\in\Omega} f(p_{uic}, \alpha) \qquad \text{s.t. } \mathbf{p} \in [0,1]^{|\Omega|}. \tag{2}$$

We use the log loss to compute the error ℓ_{uic} between the predicted rating $\hat{r}_{uic}$ and the actual rating r_{uic}. It is defined as $\ell_{uic} = -(r_{uic}\log(\hat{r}_{uic}) + (1 - r_{uic})\log(1 - \hat{r}_{uic}))$. Based on FMs, $\hat{r}_{uic}$ is computed as follows:

$$\hat{r}_{uic} = \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \,, \tag{3}$$

where w_i models the weight of x_i, the i-th variable of the one-hot encoded feature vector that corresponds to instance uic. The weight of the pairwise interaction of i-th and j-th variables is modeled by the dot product of $\mathbf{v}_i$ and $\mathbf{v}_j$. The k-dimensional vector that corresponds to variable i is denoted as $\mathbf{v}_i$. λ is a non-negative regularization parameter, and $\mathbf{p}$ is used to store the weights assigned to the instances in Ω. In order to prevent over-fitting, $\mathbf{L_2}$ regularization function $R(.)$ is used. Self-paced function or self-paced regularizer $f(\mathbf{p}, \alpha)$ is used to select instances based on weights. Kumar et al. [10] proposed a simple self-paced regularizer $f(\mathbf{p}, \alpha) = -\frac{1}{\alpha}\mathbf{p}$. When all model parameters except $\mathbf{p}$ are fixed, the optimal p_{uic} value which is denoted by p^*_{uic}, is calculated as follows:

$$p^*_{uic}(\alpha, \ell_{uic}) = \begin{cases} 1 \text{ if } \ell_{uic} < 1/\alpha \\ 0 \text{ otherwise.} \end{cases} \tag{4}$$

For the sake of simplicity, we denote an instance by uic. Instance uic is considered as *easy* and selected if $p^*_{uic} = 1$. The parameter α regulates the pace at which the model learns from new instances. As $\frac{1}{\alpha}$ increases gradually, more *complex* instances will be used for model learning. SPL regularizer can be seen as a scoring function for training instances. Usually in the use of SPL with batch learning, the model is trained over a large number of epochs, and in each epoch, only a few selected instances are used for training. At the end of an epoch, $\frac{1}{\alpha}$ value is increased, and it allows the training process to use another set of *complex* instances for the training along with the used *easy* instances. Subsequently, all the instances are fed to the model as the number of epochs grows. In other words, during the first few epochs, an immature predictive model will be trained using easy instances, and it is gradually exposed to complex instances as the model becomes mature.

It may not be possible to directly use the concept of SPL for online updating of the recommender task since SPL is defined for batch learning. However, it is possible to conceptually consider performing online updates for all incoming instances as a training process. The main difference is that in the traditional use of SPL each instance is used for training a model in multiple epochs. However, in the online recommender task, we use each incoming instance for model updating with one epoch. We thus use the SPL regularizer as a scoring function to determine whether an incoming instance is suitable for online updating based on its fit to the existing recommender model.

In contrast to the use of rating profiles, the loss that is incurred by an incoming instance is used to define how suitable the instance is. We use two SPL regularizers to devise an *easy-to-hard* curriculum. In this case, more easy instances, which do not incur a large error, are fed to the model and hard (complex) instances will be fed gradually over time. The SPL regularizers are defined as follows [9,10]:

$$f(\mathbf{p}, \alpha) = -\frac{1}{\alpha}\mathbf{p} \tag{5}$$

$$f(\mathbf{p}, \alpha) = \frac{1}{\alpha}(\frac{1}{2}\|\mathbf{p}\|^2 - \mathbf{p}) \tag{6}$$

Equation 6 leads to the following closed-form solution for the optimal p^*_{uic}.

$$p^*_{uic}(\alpha, \ell_{uic}) = \begin{cases} -\alpha\ell_{uic} + 1 \text{ if } \ell_{uic} < 1/\alpha \\ 0 \text{ if } \ell_{uic} \geq 1/\alpha\,. \end{cases} \tag{7}$$

Optimizing p with other variables (w, V) fixed. This can be easily achieved by

$$\min_{\mathbf{p}\in[0,1]^{\Omega}} \sum_{(u,i,c)\in\Omega} p_{uic}\ell_{uic} - \frac{1}{\alpha}p_{uic}\,. \tag{8}$$

Note that here we choose the SPL regularizer defined in Eq. 5 for clarity.

Optimizing (w, V) with p fixed. In this case, the optimization problem in Eq. (2) for solving these variables becomes

$$\min_{\mathbf{w},\mathbf{V}} \sum_{(u,i,c)\in\Omega} p_{uic}\ell_{uic} + \lambda R(.)\,. \tag{9}$$

Based on previous work [10], we use an alternating strategy for the optimization. Once an incoming instance *uic* arrives, the RS determines the weight p_{uic} by minimizing Eq. 8. The closed-form solution for p_{uic} can be computed by either Eq. 4 or Eq. 7. Recall that Eq. 8 is formulated based on the SPL regularizer defined in Eq. 5, and it should be changed accordingly when other SPL regularizers are used. Based on p_{uic}, the optimization Eq. 9 is done using Adagrad optimizer [6]. We monotonically decrease α value by $\mu > 1$ (i.e., $\alpha \leftarrow \alpha/\mu$) in regular time interval (e.g., after observing 10,000 instances in the dynamic setting). Note that μ is set to 1.1. When using `PS`, the model is updated by minimizing the optimization function defined in Eq. 9 by ignoring p_{uic} (i.e., setting $p_{uic} = 1$).

3 Experiments

We conducted experiments with a real-world dataset—*Gowalla* [5]. We extracted check-ins made in the New York City area (40.5 $\leq$ latitude $\leq$ 41.0, -74.5 $\leq$ longitude $\leq$ -73.5) between 2009-04-23 and 2010-04-02 (inclusive). Based on check-in timestamps, three temporal contextual features were extracted as follows: *Times of the day* contains five values-Wee hours (00.00–05.59), Morning (6.00–10.59), Noon (11.00–13.59), Afternoon (14.00–17.59) and Evening (18.00–23.59). The contextual factor *Days of the week* contains seven days as values. *WorkdayEnd* indicates whether check-ins are made during a weekend or not. Since the Gowalla dataset contains only positive implicit ratings (i.e. a user checked-in to a point-of-interest (POI)), a uniform negative sampling method was used to ensure robust model training. We leave the study of the Effects of sampling approaches to the online recommender task as a future work.

For each positive instance with the target value as 1, we randomly created two negative instances with POIs which had not been checked-in by the user and assigned 0 as target values. In total, the Gowalla dataset contains 121,299 positive and negative ratings given by 2,482 users for 10,919 POIs. A one-hot encoded feature vector of the Gowalla dataset contains 13,415 feature values.

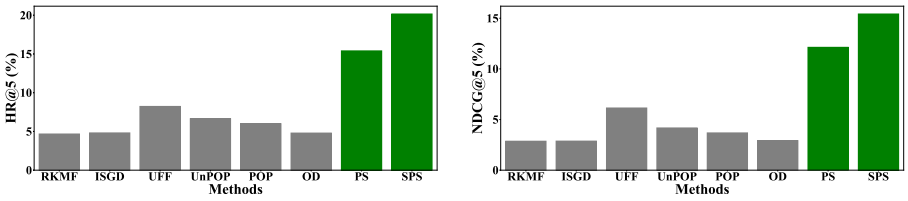
As per the common practices in online recommendations [1], the model is initialized using a small dataset. Note that the Gowalla dataset is chronologically ordered. We first split our dataset into train and test subsets. We used the first 30,000 ratings as the train set, and the rest of the data is used as a stream of incoming instances for testing. A validation set with 5000 ratings is also extracted from the train set for tuning hyperparameters. A large test set ($\approx$ 75% of the dataset) has been used to simulate real-world dynamic recommender environments. For this simulation, we adopt the *test-then-learn* evaluation method.

The recommender models have to first provide recommendations for each incoming instance (i.e., user interaction). Based on user feedback (i.e, test) on the produced recommendations, the models are updated (i.e., learn) to keep parameters refreshed. Note that we produce recommendations only for the incoming positive instances since the negative instances do not represent an actual user interaction. However, the negative instances in the test set are used for the online updating process to avoid the learning bias towards the positive instances.

On the arrival of an incoming instance, we truncate a rank list at 100 items by following the work of He et al. [8]. The rank list includes the item that appeared in the instance and 99 items that had not been rated/checked-in by the user (i.e., the user of the instance) previously. *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG) [7] are used as the evaluation metrics. For both metrics, we report the average result considering all the positive test instances. Both are positive-oriented metrics, meaning that higher values for these metrics indicate better performance.

Baselines. `PS` and `SPS` are compared against the following baselines. Note that, for `SPS`, we use the SPL regularizer defined in Eq. 5: (a) `RKMF`: This is an online update mechanism integrated with Matrix Factorization (MF) [17]. Incoming instances were selected based on the profile sizes of users and items by favoring new users and new items for online updating. We use the linear kernel and set m of `RKMF` to the default value of 50; (b) `ISGD`: `ISGD` is an incremental MF approach that uses only positive feedback (instances) to update the model [19]; (c) `UFF`: User factor fading is one of the latent factor forgetting methods proposed by [13]. A constant has been used to regulate the effect of forgetting. As the name implies, latent vectors of users are penalized over time; (d) `UnPOP`: In this baseline, the latent vectors of unpopular items are penalized [13]; (e) `POP`: This strategy is the contradiction of `UnPOP` [13]. The popularity of an item is determined based on the number of ratings that the item had received; and (f) `OD`: Outlier discarding filters "outlier" of instances based on error values [4]. The authors used this idea to filter out historical ratings in the training, and we adopt this method in the recommender task where each instance is utilized for online updating only if the following condition holds: $|\hat{r}_{uic} - r_{uic}| > \beta.sd_U(u)$, where $\hat{r}_{uic}$ and r_{uic} denote the predicted rating and the ground truth of the instance, and $sd_U(U)$ is the standard deviation of the prediction errors for all previous ratings given by u, and β is the controlling parameter of the forgetting method.

Parameter Setting. For all baselines, we set the learning rate and L_2 regularization parameter to 0.001 and 0.1, respectively. The hyperparameter embedding size k was searched in $\{16, 32, 64, 128\}$, and k was set to 16 and 32 in `PS` and `SPS`, respectively. As presented by the authors, the forgetting factors of `UFF`, `UnPOP` and `POP` were set as 0.5, 2 and 1.00001, respectively. β of `OD` was set to 1.5. We empirically set α to 1.8, δ to 0.98, and θ to 1600. During the initial batch training, all models, except `ISGD`, were trained for 10 epochs, and `PS` and `SPS` used mini-batches (size of 512). Note that, in online updating, we considered the test instances individually and updated all the model using a single epoch.



(a) HR@5 **(b)** NDCG@5

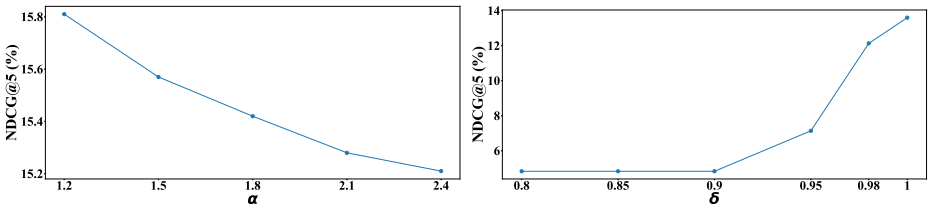
Fig. 1. Prediction accuracy.

3.1 Performance Comparison

Figure 1a and 1b show the comparisons between all methods when generating top-5 recommendations. `SPS` outperforms all methods by achieving 23.6% and 27.1% relative improvements over the best competitor `PS` w.r.t. HR and NDCG values, respectively. The improvements over the best competitor are statistically significant with $p < 0.05$ (two-tailed paired t-test). Compared to the MF-based methods which do not use contextual information, the FM-based methods (`SPS` and `PS`) show superior performance on the sparse Gowalla dataset. We argue that this improvement is attributed to (1) the use of temporal contextual information and (2) the ability to lead to a better model within a limited number of epochs during the initial training.

Out of all MF-based models, `UFF` shows better performance. In `UFF`, during the online update of a user latent vector, the current user vector is computed by multiplying the previous vector by a forgetting factor. Note that `UFF` does not completely ignore user interactions as proposed in `RKMF`. The performance improvement of `UFF` over `RKMF` indicates the adverse effect of neglecting the interactions of users who have higher rating profiles in online updates. We believe that the performance decline of `PS` against `SPS` is also caused by significant information loss that happens especially due to the disregard of short-term preferences of the users who have large rating profiles. On the contrary, `SPS` assigns 0 weights for the interactions that induce higher losses to the already learned model. The decision to incorporate a user interaction does not depend on the rating profile of the user (or the item or the contextual feature values). Irrespective of the size of a user's rating profile, `SPS` utilizes an interaction made by that specific user for online updating if it does not deteriorate the already learned model.

Impact of Testing Data. The performance of `SPS` boosts when the test data size decreases (or the train data size increases). This is an expected observation because the use of more data for initial training improves prediction accuracy. Decreasing the train set drastically leads to suboptimal performance in general, mainly due to the poorly trained models that are highly sensitive to the users, items, and contextual feature values with small rating profiles. We thus highlight the importance of utilizing a sufficient amount of train data, without overly reducing.



(a) Effects of α. **(b)** Effects of δ.

Fig. 2. Sensitivity of parameters.

Efficiency. Ignoring instances in PS and SPS is the main reason for the improvements of efficiency. In FMs, the time complexity for updating an instance is $\mathcal{O}(k\bar{m})$ [16], where k is the dimension of a latent vector which corresponds to a feature value, and $\bar{m}$ is the number of non-zero feature values (i.e., variables) presented in the instance. Let $\mathcal{F}$ be the number of test instances in a recommender environment. Thus, updating all instances takes $\mathcal{O}(k\bar{m}\mathcal{F})$ time.

However, due to the instance selection in online updating, we deliberately reduce $\mathcal{F}$, and it leads to gaining efficiency. For example, $\mathcal{F}$ can be approximately reduced to $\frac{\mathcal{F}}{2}$ by setting δ to 0.98 and θ to 1600 in PS or by setting α to 2.1 in SPS. We thus define the time of complexity of the online updating process integrated with our instance selection strategies as $\mathcal{O}(k\bar{m}\mathcal{F}')$, where $\mathcal{F}' < \mathcal{F}$.

3.2 Sensitivity of Parameters

As can be seen in Figure 2a, the performance of SPS slightly decreases with the increase of α. We conducted a micro-level analysis on this phenomenon. We observe that the increase of α discourages online updating especially at the initial stage of the testing process. For example, when α is set to 2.4, the RS does not update the model during the arrivals of the first 50,000 test instances since the instances are set p^*_{uic} (of Eq. 4) to 0. Moreover, in that setting, the model disregards 61% of instances in total. Omitting a high percentage of testing instances hinders prediction accuracy.

On the contrary, we observe that all the test instances, without selecting, were utilized when α is 1.2; however, this setting is computationally inefficient. Setting α to 1.8 shows a better balance between prediction accuracy and efficiency. If we compare the two settings where α is set to 1.2 and 1.8, when α is 1.8, the online updating declines accuracy only by 2.5% while utilizing 71.8% of testing instances. When α is 2.1, only 52.6% of testing instances are used, and the performance declines against the setting where α is 1.2 just by 3.4%. We thus argue that the use of SPL is important to improve the overall efficiency of the online updating process while achieving comparable prediction accuracy.

Figure 2b plots how NDCG@5 values obtained by PS change with different settings of δ. It is clearly visible that the use of higher values of δ improves the prediction accuracy. In fact, setting δ to 0.8, 0.85, and 0.9 shows similar per-

formance since those settings ignore testing instances similarly. We can observe that setting δ to 1 (i.e., disregard the impact of the profile ratings of contextual feature values) slightly increases the performance. However, the setting where δ to 0.98 declines performance by 12.1% while utilizing only 46.3% of testing instances. Note that the setting where δ is 1 utilizes 93.7% of testing instances.

Based on this effect, we can conclude that the use of contextual information with `PS` filters out a large number of testing instances, which makes the online updating method efficient, with a decline of prediction accuracy. These observations along with the performance comparison illustrated in Figure 1b show the superior performance of `SPS` in the online updating process in terms of balancing the trade-off between prediction accuracy and efficiency compared to `PS`.

3.3 Case Study

Baltrunas et al. released the *Frappe* dataset[1] that consists of 96,203 usage logs as implicit ratings of 957 users on 4,082 mobile applications under different contexts [2]. We use seven contextual factors (e.g., weather) excluding the attribute "cost". If a user has used an app less than 5 times under a context, we considered those events (i.e., app usages) as negative instances and assigned 0 as their target values. The remaining instances are assigned 1. Due to the unavailability of timestamps in the Frappe dataset, the default order of ratings is used for split data, and the split ratio is similar to that of the Gowalla dataset. It is evident that the use of contextual features generally improves prediction accuracy as shown in Figure 1a and 1b.

We further analyse this phenomenon by comparing `SPS`, with `SPS\C`. `SPS\C`, which is a variant of `SPS`, considers users and items to construct feature vectors by ignoring contextual feature values. Both approaches show comparable performance on the Gowalla dataset. However, on the Frappe dataset, `SPS` outperforms `SPS\C` by 1.2% and 3.3% in terms of HR@5 and NDCG@5 values, respectively. This result indicates that the effectiveness of SPL for instance selection in the presence of a large number of contextual feature values.

4 Related Work

Most of the early attempts of handling streams of user interactions and updating user-user similarities are based on neighborhood models [15,18]. Vinagre and Jorge presented a forgetting approach to gradually forget older instances using sliding windows and "fading factors" [18]. In 2014, Matuszyk and Spiliopoulou proposed two forgetting techniques and integrated those with MF [11]. They kept ratings of each user as a list, and once a new rating is provided by a user, the outdated rating was removed from her list of ratings while adding the newly arrived rating. After that, following the *test-then-learn* approach, the latent vector of the incoming user is updated by utilizing the ratings in the incoming user's

[1] http://web.archive.org/web/20180422190150/http://baltrunas.info/research-menu/frappe

list. Matuszyka et al. proposed a novel forgetting method in which they obtained a new latent vector of an incoming user by incremental training and compared it with the previously learned user vector [12,13]. Furthermore, the authors have discussed several other forgetting strategies based on item popularity and change detection.

Rendle and Schmidt-Thieme [17] proposed a filtering mechanism to select incoming user instances based on their profile sizes. If a user (item) of an incoming instance has a small profile, then the model incorporates such instances for online updating. Moreover, they proposed to consider the error between the predicted and true values of a user interaction. Devooght et al. also utilized Rendle's work [17] for online updating. Al-Ghossein et al. used local models to track changes in user preferences over time [1]. They argued that changes in user preferences do not occur uniformly. Their approach is based on the Item-based KNN method, and for each incoming instance, their model compares a set of local models to detect changes.

5 Conclusion

We provide an exploratory study on how instance selection mechanisms affect online updating, which is an essential component of the online recommender task. In addition to defining rating profile-based instance selection with the use of contextual feature values, we show how to utilize *Self-Paced Learning*, which has been traditionally used in the batch-learning setting, with online updating. To the best of our knowledge, this is the first attempt to integrate the notion of SPL with online recommendations. Our results indicate the impact of contexts on balancing the trade-off between prediction accuracy and efficiency. Moreover, we discuss how different usages of SPL affect overall performance. In the future, we believe that it is important to study different types of data manipulation mechanisms with online updating to further minimize the impact of information loss that is caused due to the disregard of instances, while improving efficiency.

References

1. Al-Ghossein, M., Abdessalem, T., Barré, A.: Dynamic local models for online recommendation. In: Companion Proceedings of the The Web Conference 2018. pp. 1419–1423 (2018)
2. Baltrunas, L., Church, K., Karatzoglou, A., Oliver, N.: Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. CoRR **abs/1505.03014** (2015), http://arxiv.org/abs/1505.03014
3. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning. pp. 41–48 (2009)
4. Chen, J., Li, H., Xie, Q., Li, L., Liu, Y.: Streaming recommendation algorithm with user interest drift analysis. In: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data. pp. 121–136. Springer (2019)

5. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: user movement in location-based social networks. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1082–1090 (2011)
6. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(Jul), 2121–2159 (2011)
7. He, X., Chen, T., Kan, M.Y., Chen, X.: Trirank: Review-aware explainable recommendation by modeling aspects. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. pp. 1661–1670. ACM (2015)
8. He, X., Zhang, H., Kan, M.Y., Chua, T.S.: Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. pp. 549–558 (2016)
9. Jiang, L., Meng, D., Mitamura, T., Hauptmann, A.G.: Easy samples first: Self-paced reranking for zero-example multimedia search. In: Proceedings of the 22nd ACM international conference on Multimedia. pp. 547–556 (2014)
10. Kumar, M.P., Packer, B., Koller, D.: Self-paced learning for latent variable models. In: Advances in Neural Information Processing Systems. pp. 1189–1197 (2010)
11. Matuszyk, P., Spiliopoulou, M.: Selective forgetting for incremental matrix factorization in recommender systems. In: International Conference on Discovery Science. pp. 204–215. Springer (2014)
12. Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting methods for incremental matrix factorization in recommender systems. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. pp. 947–953 (2015)
13. Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting techniques for stream-based matrix factorization in recommender systems. Knowledge and Information Systems **55**(2), 275–304 (2018)
14. Nasraoui, O., Cerwinske, J., Rojas, C., Gonzalez, F.: Performance of recommendation systems in dynamic streaming environments. In: Proceedings of the 2007 SIAM International Conference on Data Mining. pp. 569–574. SIAM (2007)
15. Papagelis, M., Rousidis, I., Plexousakis, D., Theoharopoulos, E.: Incremental collaborative filtering for highly-scalable recommendation algorithms. In: International Symposium on Methodologies for Intelligent Systems. pp. 553–561. Springer (2005)
16. Rendle, S.: Factorization machines. In: 2010 IEEE International Conference on Data Mining. pp. 995–1000. IEEE (2010)
17. Rendle, S., Schmidt-Thieme, L.: Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In: Proceedings of the 2008 ACM conference on Recommender systems. pp. 251–258 (2008)
18. Vinagre, J., Jorge, A.M.: Forgetting mechanisms for scalable collaborative filtering. Journal of the Brazilian Computer Society **18**(4), 271–282 (2012)
19. Vinagre, J., Jorge, A.M., Gama, J.: Fast incremental matrix factorization for recommendation with positive-only feedback. In: International Conference on User Modeling, Adaptation, and Personalization. pp. 459–470. Springer (2014)
20. Zhang, Y., Wang, H., Lian, D., Tsang, I.W., Yin, H., Yang, G.: Discrete ranking-based matrix factorization with self-paced learning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2758–2767 (2018)