Dynamic Local Models for Online Recommendation

Marie Al-Ghossein
LTCI, Télécom ParisTech
Université Paris-Saclay
AccorHotels
Paris, France
malghossein@enst.fr

Talel Abdessalem
LTCI, Télécom ParisTech
Université Paris-Saclay
UMI CNRS IPAL NUS
Paris, France
talel.abdessalem@enst.fr

Anthony Barré
AccorHotels
Paris, France
anthony.barre@live.fr

ABSTRACT

With the explosion of the volume of user-generated data, designing online recommender systems that learn from data streams has become essential. These systems rely on incremental learning that continuously update models as new observations arrive and they should be able to adapt to drifts in real-time. User preferences evolve over time and tracking their evolution is not an easy task. In addition to the low number of observations available per user, the preferences change at different moments and in different ways for each individual. In this paper, we propose a novel approach based on local models to address this problem. Local models are known for their ability to capture diverse preferences among user subsets. Our approach automatically detects the drift of preferences that leads a user to adopt a behavior closer to the users of another subset, and adjusts the models accordingly. Our experiments on real world datasets show promising results and prove the effectiveness of using local models to adapt to changes in user preferences.

ACM Reference Format:
Marie Al-Ghossein, Talel Abdessalem, and Anthony Barré. 2018. Dynamic Local Models for Online Recommendation. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3184558.3191586

1 INTRODUCTION

In the last two decades, top-N recommender systems (RS) have been gaining a lot of attention and have been applied in many domains. Their task is to provide users with a personalized ranked list of items to help them make good decisions and increase their satisfaction.

In order to improve the quality of recommendation, contextual information has been exploited in previous work [1]. Time information is in particular very relevant for personalization. For example, in the application of hotel recommendation, user preferences tend to change over time, especially when their standards evolve. They typically move from booking hotels in one segment, e.g., economy segment, to booking hotels in a higher one, e.g., luxury segment. Destination popularity also varies with trends, seasons, and the occurrence of impactful events. Time-aware RS [2] respect the chronological order of observations and try to capture the temporal dynamics existing in user-generated data.

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3191586

Most RS and time-aware RS proposed in the literature build first a model from a large static dataset, and then rebuild it periodically as new chunks of data arrive and are added to the original dataset. Training a model on a continuously growing dataset is computationally expensive and the frequency of model updates usually depends on the model's complexity and scalability. Therefore, RS can not take into account the user feedback generated after a model update before the next one, leading to a lower quality of recommendation and to the inability to adapt to quick changes.

One way to address this issue is to approach the recommendation problem as a data stream problem and design online RS that learn from continuous data streams and adapt to changes in real-time. These systems are based on incremental learning allowing continuous update and retraining of models using recent data [16, 21]. One challenge with learning from data streams is to detect and adapt to concept drifts. Instance selection [15, 17] and instance weighting [6] strategies have been explored in this direction. However, these methods require fixing a set of parameters (e.g., size of a sliding window, decaying factor) and anticipate that the RS should forget old observations and learn from new ones. They assume that we have a prior knowledge of the way the user behavior changes and that the preferences of all users drift at the same rate.

In this paper, we present a novel incremental approach relying on item-based local models to learn user preferences and track their evolution in online RS. Our approach maintains one global model for all users and several local models built separately for each subset of users. Local models have been exploited for batch RS and are able to capture diverse and opposing preferences [3]. We propose to continuously evaluate over time user assignments to subsets. Users are moved from one subset to a more adapted one when a change in preferences is detected, and user profiles are updated accordingly. Experiments on three real datasets show promising results.

The rest of the paper is organized as follows. In Section 2, we discuss related work on online RS. In Section 3, we present our approach performing online recommendation and adapting to changes in user behaviors. Experiments and results are presented and discussed in Section 4. Finally, Section 5 concludes the paper.

2 RELATED WORK

Time-aware RS. Time-aware RS take into account the time information in order to better represent the dynamics of users and items. One way to approach time in RS is to use it as context [1] and model it as a number of discrete categorical variables, e.g., time of day, day of week, seasons. Another way is to consider it as a

continuous variable [10]. Common approaches consist in weighting observations according to their recency [2] or addressing the recommendation problem using time series [12].

Most research in time-aware RS assume that models should be retrained frequently in batch to stay up-to-date. In real world RS, this assumption leads to scalability issues as the data generated by users keeps on growing, and to a loss of accuracy as the RS can not instantly adjust to changes.

Online RS. Recent work [5] has proposed to address the recommendation problem as a data stream problem, which is a more realistic setting. Moreover, Frigó et al. [7] show that simple algorithms updated online can perform better than more complex algorithms updated periodically. Elements of a data stream are considered to arrive in real-time and to be processed sequentially in only a few passes (typically one) using limited memory and processing time per element. Online RS rely on incremental approaches to build models, e.g., incremental neighborhood-based methods [16] and incremental matrix factorization [21].

Dealing with concept drift is one of the core problems in data stream mining [9]. Online RS have to be able to track multiple concepts changing in different ways at different moments, including the preferences of each user. Liu and Aberer [13] propose to model users' short-term and long-term preferences separately using an offline and an online component. Nasraoui et al. [17] develop a user-based neighborhood algorithm using a sliding window that contains a fixed number of observations. Siddiqui et al. [20] also use a sliding window and perform rating prediction by clustering user profiles. Forgetting outdated information is another common approach used to deal with evolving data [15].

Previous work makes assumptions concerning the drift rate of user preferences or the relevance of old observations, and does not take into account the fact that these changes do not occur uniformly over all users. We propose a novel approach to perform online RS that adapts to new observations when change is detected. Our approach is based on local models.

Local models for recommendation. O'Connor and Herlocker [19] propose to estimate multiple local recommendation models in order to perform rating prediction. Users are clustered based on the rating matrix and one local model is built for each cluster. Lee et al. [11] consider that the rating matrix is locally low-rank. Neighborhoods are identified using a distance measure between pairs of users and items and a local low-rank model is estimated for each neighborhood. Christakopoulou and Karypis [3] propose a method using SLIM (Sparse LInear Methods [18]) to perform top-N recommendation. They automatically identify appropriate user subsets and combine global and local SLIM models to improve the recommendation performance.

3 PROPOSED APPROACH

3.1 Motivation

Our goal is to develop an online RS that detects change in user preferences and adapts to it. Our approach extends item-based methods since it has been shown that they outperform user-based methods [4]. A single item-based model may not be enough to capture the preferences of a set of users. In particular, a single model can not detect diverse or opposing preferences existing in user subsets [3]. Local models built separately for each subset of users try, for their part, to represent fine-grained patterns.

We focus on detecting the change of preferences that would push a user to adopt a behavior that is different from the one of those belonging to the same subset and closer to the one of those assigned to another subset. In the hotel recommendation problem, this could be for example the consequence of a change in the user social status. The drift is therefore handled by assigning the user to a more adapted subset and updating the models accordingly.

Our approach is designed to extend any incremental item-based method and we rely in this work on the item K-Nearest Neighbors (KNN) method.

3.2 Incremental item K-Nearest Neighbors

Notation. We denote by U the set of users containing n_U users and by I the set of items containing n_I items. The matrix R of size $n_U \times n_I$ is used to represent the user-item implicit feedback. If user u provided feedback for item i, the entry r_{ui} of R is 1, otherwise it is 0. The set of users that have rated the item i is denoted by U_i. To generate recommendations for a target user u, we compute the recommendation scores (ratings) of every item i unrated by u, denoted by $\widetilde{r_{ui}}$, and select the top-N items for recommendation.

Item-based KNN methods explore similarities between items to provide recommendations. The similarity between two items i and j is computed using the cosine similarity between R_{*i} and R_{*j}, the columns i and j of the matrix R, and is given by:

$$sim(i,j) = \frac{R_{*i}.R_{*j}}{\|R_{*i}\| \times \|R_{*j}\|} = \frac{|U_i \cap U_j|}{\sqrt{|U_i|} \times \sqrt{|U_j|}} \quad (1)$$

We denote by S the item similarity matrix such that $S_{ij} = sim(i,j)$. Recommendations to a user u are computed by aggregating and ranking the K-nearest neighbors of the items already rated by u. Miranda and Jorge [16] propose an incremental version of this method.

3.3 Dynamic local models

In this subsection, we detail our approach also shown in Algorithm 1. We maintain m local item-based models and one global item-based model. Each model is represented by an item-item similarity matrix S_*. We learn m local similarity matrices denoted by S_l, where l is the index of the local model, and one global similarity matrix S_g. At timestamp t, every user u belongs to one subset l where $l \in \{1, ..., m\}$ and the rating $\widetilde{r_{ui}}$ is given by:

$$\widetilde{r_{ui}} = \alpha_g.(\widetilde{r_{ui}})_g + (1 - \alpha_g).(\widetilde{r_{ui}})_l \quad (2)$$

$(\widetilde{r_{ui}})_g$ is computed using the global model, $(\widetilde{r_{ui}})_l$ is computed using the local model l, and α_g is the weight controlling the contribution of the global and local models.

Updating the models. Online RS assume that observations, i.e., user-item pairs (u, i), are continuously generated and handled. Each observation received is used to update the models, i.e., the similarity matrices, and to update the assignments of users to subsets.

The idea is to detect that u is adopting a behavior that is no longer similar to the users of the subset l. We assume that this is the case when we find that there is a local model that performs better than l for u. The change of user preferences also requires forgetting old information that is no longer relevant to the current behavior of u. We compare the performance of l using the full profile of u, i.e., R_{u*}, against the performances of the other local models using only recent observations of R_{u*}.

Comparing the performances of local models. We define the metric measuring the error of a model l when tested for the pair (u, i) as follows:

$$err(l, R_{u*}) = 1 - \frac{1}{rank(i)} \tag{3}$$

where $rank(.)$ is a function returning the ranking of the item i in the item list sorted by decreasing order of $(r_{u*})_l$. A better recommendation model should rank the relevant item higher in the list.

Forgetting irrelevant observations. When a change in the user behavior is detected, the assignment of u to user subsets is modified and his profile, i.e., the row R_{u*} of the matrix R, is updated. The old observations corresponding to the previous adopted behavior have to be forgotten. To this end, we rely on a forgetting strategy [14] where we keep in R_{u*} the last F items observed for u and remove the older observations. We denote by R'_{u*} the result of applying the forgetting strategy to R_{u*}.

Initializing item-based models. Real-world RS usually have access to part of the data before running in an online fashion. We use part of the available data to perform an initial batch training [22]. Users are first separated into clusters either randomly or using a clustering algorithm. The local models and the global model are learned in batch. We then fix the learned models, iterate over users, and verify that the subset they belong to generates the smallest error on the observed user-item pairs. If this is the case, the user remains in the initial cluster. Otherwise, he is moved to the cluster with the smallest error.

Running time. We note that all the models can be trained in parallel and used for recommendation independently. Our approach introduces a very low overhead to the item-based method extended.

Algorithm 1 Overview of our approach DOLORES

for all observations (u, i) **do**
  Compute $err(l, R_{u*})$
  for all $l' \in \{1, ..., m\}$ s.t. $l' \neq l$ **do**
    Compute $err(l', R'_{u*})$
  end for
  set $l_{opt} = l'$ s.t. $err(l_{opt}, R'_{u*}) = \min_{l'} err(l', R'_{u*})$
  if $err(l, R_{u*}) > err(l_{opt}, R'_{u*})$ **then**
    Update the global model matrix S_g with (u, i)
    Assign the user u to the local model l_{opt}
    $R_{u*} \leftarrow R'_{u*}$
  else
    Update the global model matrix S_g with (u, i)
    Update the local model l matrix S_l with (u, i)
  end if
end for

4 EXPERIMENTS

Evaluation protocol. Holdout methods, traditionally used in the evaluation process of RS, are not adapted for the evaluation of online RS [22]. By randomly sampling data for training and testing, these methods ignore the temporal dimension and do not take into account the natural ordering of observations [22].

Since our method requires a short offline initialization phase, we adopt the evaluation protocol proposed in [14]. The dataset is sorted chronologically and then split in three subsets, with the proportions 30%-20%-50%:

(1) **Batch Train** subset, used as a training set for initializing the models in batch.
(2) **Batch Test - Stream Train** subset, used as a test set for the initialization set, and also used for incremental online learning to ensure the transition between (1) and (3).
(3) **Stream Test and Train** subset, used for prequential evaluation.

The prequential evaluation [8] is a *test-then-learn* procedure performed while iterating over the dataset. For each observation, i.e., pair (user u, item i), we use the current model to perform recommendation for u, measure the performance of the model given i, and then use the observation (u, i) to update the model.

Evaluation measures. We use two measures for evaluating the quality of recommendation, recall@N and DCG@N, as defined in [7]. The metrics are computed individually and averaged for all observations.

Datasets. We evaluated the performance of our method on three datasets which characteristics are shown in Table 1. The *AH* dataset is extracted from the hotel industry [1] and gathers actual bookings of European users in the hotels of AccorHotels during a period of 3 consecutive years. The *ml-1M* and *ml-10M* datasets correspond respectively to the MovieLens 1M and MovieLens 10M datasets [2] and represent movie ratings. In order to use the data as implicit feedback (positive-only data), we keep in the dataset the pairs for which the rating is in the top 20% of the rating scale of the dataset, i.e., rating equal to 5 [21].

Parameters. We performed a grid search over the parameter space of the methods in order to find the parameters that give the best performance. Due to space constraints, we only report the performance corresponding to the parameters that lead to the best results. We fix the number of neighbors K to 100 for all the methods.

Methods compared. In order to demonstrate our proposed method, we evaluate the performance of several recommendation models including variants of the proposed method:

[1] http://www.accorhotels.com
[2] http://www.movielens.org

Table 1: Description of the datasets.

Dataset	#Users	#Items	#Transactions
AH	98,130	3,332	704,722
ml-1M	6,012	3,135	157,535
ml-10M	67,297	8,658	1,223,892

AH				
Method	**recall@5**	**DCG@5**	**recall@10**	**DCG@10**
KNNi	0.1621	0.1033	0.2819	0.1415
KNNi_w	0.1683	0.1042	0.2831	0.1421
DOLORES	**0.1852**	**0.1179**	**0.3045**	**0.1561**
DOLORES-G	0.1816	0.1165	0.3012	0.1548
LORES	0.1694	0.1075	0.2902	0.1461
ml-1M				
Method	**recall@5**	**DCG@5**	**recall@10**	**DCG@10**
KNNi	0.0407	0.0253	0.0716	0.0352
KNNi_w	0.0409	0.0258	0.0717	0.0354
DOLORES	**0.0521**	**0.0298**	**0.0798**	**0.0386**
DOLORES-G	0.0505	0.0289	0.0788	0.0381
LORES	0.0412	0.0271	0.0729	0.0364
ml-10M				
Method	**recall@5**	**DCG@5**	**recall@10**	**DCG@10**
KNNi	0.0591	0.0389	0.0943	0.0502
KNNi_w	0.0594	0.0390	0.0944	0.0502
DOLORES	**0.0602**	**0.0394**	**0.0953**	**0.0511**
DOLORES-G	0.0598	0.0392	0.0949	0.0508
LORES	0.0596	0.0390	0.0945	0.0505

Table 2: Metrics measured for each method per dataset.

- **KNNi** denotes the incremental item-based KNN method.
- **KNNi_w** denotes the incremental item-based KNN method that uses a sliding window per user and retains only the last F items rated by each user. We set F=20.
- **DOLORES** stands for Dynamic Local Online RS and denotes the method we propose in this paper. We set m=15, F=20, α_g=0.5 for *AH*, and α_g=0.7 for *ml-1M* and *ml-10M*.
- **DOLORES-G** stands for Dynamic Local Online RS without a Global model. This is equivalent to setting the parameter α_g to 0 in equation 2 and only using local models to perform recommendation. We set m=15 and F=20.
- **LORES** stands for Local Online RS. In LORES, the user assignment to subsets is fixed after the initial optimal assignment of each user. We set m=15, F=20, α_g=0.5 for *AH*, and α_g=0.7 for *ml-1M* and *ml-10M*.

Performance of the methods. The results are shown in Table 2.

- By comparing KNNi with DOLORES and its variants, we show the importance of building local models and considering the change in user preferences.
- By comparing KNNi_w with DOLORES, we highlight the advantage of using our approach instead of sliding windows.
- By comparing DOLORES-G with DOLORES, we show the benefit of using a global model to capture global patterns, instead of only using local models.
- By comparing LORES with DOLORES, we demonstrate the benefit of reevaluating user assignments to subsets as new observations arrive. This also shows that user preferences are shifting over time. Results prove that taking into account this change of preferences has a high impact on the quality of recommendation. We note for example that for the *AH* dataset, user assignments to subsets are modified for 20% of the received observations.

DOLORES outperforms the other methods for the studied datasets and we can see the relative benefit of each of its components.

5 CONCLUSION

Tracking the changes of user preferences in online RS raises unique challenges. On one hand, RS have to be able to process data streams and adapt to drifts in real-time. On the other hand, these drifts happen differently for each individual and we do not have any information about how they occur. We propose a novel approach based on local models to address the problem. Our approach automatically detects change of preferences by continuously reevaluating the assignment of users to subsets and updating the models. In future work, we will be extending this approach to other recommendation models including latent factor models. We will also be investigating methods to distinguish between different types of drifts in preferences in order to enhance the quality of recommendation.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2015. Context-aware recommender systems. In *Recommender systems handbook*. Springer, 191–226.

[2] Pedro G Campos, Fernando Díez, and Iván Cantador. 2014. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction* 24, 1-2 (2014), 67–119.

[3] Evangelia Christakopoulou and George Karypis. 2016. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 67–74.

[4] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 143–177.

[5] M. Benjamin Dias, Dominique Locher, Ming Li, Wael El-Deredy, and Paulo J.G. Lisboa. [n. d.]. The Value of Personalised Recommender Systems to e-Business: A Case Study. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08)*.

[6] Yi Ding and Xue Li. 2005. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 485–492.

[7] Erzsébet Frigó, Róbert Pálovics, Domokos Kelen, Levente Kocsis, and András A. Benczúr. 2017. Online Ranking Prediction in Non-stationary Environments. In *Proceedings of the 1st Workshop on Temporal Reasoning in Recommender Systems co-located with 11th International Conference on Recommender Systems (RecSys 2017), Como, Italy, August 27-31, 2017*. 28–34.

[8] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2009. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 329–338.

[9] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 44.

[10] Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* 53, 4 (2010), 89–97.

[11] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2014. Local collaborative ranking. In *Proceedings of the 23rd international conference on World wide web*. ACM, 85–96.

[12] Xin Liu. 2015. Modeling Users' Dynamic Preference for Personalized Recommendation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[13] Xin Liu and Karl Aberer. 2014. Towards a dynamic top-n recommendation framework. In *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 217–224.

[14] Pawel Matuszyk and Myra Spiliopoulou. 2014. Selective forgetting for incremental matrix factorization in recommender systems. In *International Conference on Discovery Science*. Springer, 204–215.

[15] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. 2015. Forgetting methods for incremental matrix factorization in recommender systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 947–953.

[16] Catarina Miranda and Alípio Mário Jorge. 2009. Item-based and user-based incremental collaborative filtering for web recommendations. In *Portuguese Conference on Artificial Intelligence*. Springer, 673–684.

[17] Olfa Nasraoui, Jeff Cerwinske, Carlos Rojas, and Fabio Gonzalez. 2007. Performance of recommendation systems in dynamic streaming environments. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 569–574.

[18] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 497–506.

[19] Mark OâĂŹConnor and Jon Herlocker. 1999. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR workshop on recommender systems*, Vol. 128. UC Berkeley.

[20] Zaigham Faraz Siddiqui, Eleftherios Tiakas, Panagiotis Symeonidis, Myra Spiliopoulou, and Yannis Manolopoulos. 2014. xstreams: Recommending items to users with time-evolving preferences. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*. ACM, 22.

[21] João Vinagre, Alípio Mário Jorge, and João Gama. 2014. Fast incremental matrix factorization for recommendation with positive-only feedback. In *International Conference on User Modeling, Adaptation, and Personalization*. Springer, 459–470.

[22] João Vinagre, Alípio Mário Jorge, and João Gama. 2015. Evaluation of recommender systems in streaming environments. *arXiv preprint arXiv:1504.08175* (2015).