Local Popularity and Time in top-N Recommendation

Vito Walter Anelli[1], Tommaso Di Noia[1], Eugenio Di Sciascio[1], Azzurra Ragone[2], and Joseph Trotta[1]

[1] Polytechnic University of Bari, Bari, Italy
{firstname.lastname}@poliba.it
[2] Independent Researcher
azzurra.ragone@gmail.com

Abstract. Items popularity is a strong signal in recommendation algorithms. It strongly affects collaborative filtering approaches and it has been proven to be a very good baseline in terms of results accuracy. Even though we miss an actual personalization, global popularity can be effectively used to recommend items to users. In this paper we introduce the idea of a *time-aware personalized popularity* in recommender systems by considering both items popularity among neighbors and how it changes over time. An experimental evaluation shows a highly competitive behavior of the proposed approach, compared to state of the art model-based collaborative approaches, in terms of results accuracy.

1 Introduction

Collaborative-Filtering (CF) [25] algorithms, more than others, have gained a key-role among recommendation approaches and have been effectively implemented in commercial systems to help users in dealing with the information overload problem. Some of them also use additional information (hybrid approaches) to build a more precise user profile in order to serve a much more personalized list of items [3, 9].

However, it is well known [12] that all the algorithms based on a CF approach are affected by the so called "popularity bias" meaning that popular items tend to be recommended more frequently than those in the long tail. Initially considered as a shortcoming of collaborative filtering algorithms and then not useful to produce good recommendations [11], in some works items popularity has been intentionally penalized [17]. Very interestingly, a recommendation algorithm purely based on most popular items, has been proven to be a strong baseline [7] although it does not exploit any actual personalization. More recently, popularity has been also considered as a natural aspect of recommendation that, by measuring the user tendency to diversification, can be exploited to balance the recommender optimization goals [13]. The study of popularity in user tendencies is not completely new in the recommender systems field. Some interesting works explored these criteria for re-ranking purposes [13, 17], and multiple goals optimization [11].

In the approach we present here, we introduce a more fine-grained personalized version of popularity by assuming that it is conditioned by the items that a user u already experienced *in the past*. To this extent, we look at a specific class of neighbors, that we name *Precursors*, defined as the users who already rated the same items of u in the past. This led us to the introduction of a time-aware analysis while computing a recommendation list for u. As time is considered a contextual feature, most of the works dealing with temporal aspects are considered as a sub-class of Context-Aware RS (CARS) [2]: Time-Aware RS (TARS) [24, 1, 14]. In TARS, the freshness of different ratings is often considered as a discriminative factor between candidate items. Usually, a time window [15] is adopted to filter out all the ratings that stand before (and/or after) a certain time relative to the user or the item. Recently, an interesting work that makes use of time windows has been proposed in [5] where the authors focus on the last common interaction between the target user and her neighbors to populate the candidate items list. In [4] social information and time are integrated dealing with the interests of the users as a series of temporal matrices. Probabilistic matrix factorization technique are adopted to learn latent factors. Regarding sequences and recommendation it is worth to mention [22], in which the authors combine an LSTM network with a low-rank matrix factorization algorithm to produce recommendation lists. A pioneer work was proposed more than a decade ago in [8] which used an exponential decay function $e^{-\lambda t}$ to penalize old ratings. An exponential decay function [14] was then used to integrate time in a latent factors model. In the last years, several Item-kNN [16, 8] with a temporal decay function have been deployed. Another interesting work was proposed in [23] where three different kinds of time decay were adopted: exploiting concave, convex and linear functions.

In this paper we present `TimePop`, an algorithm that combines the notion of personalized popularity conditioned to the behavior of users' neighbors while taking into account the temporal dimension. It is worth noticing that `TimePop` works with implicit feedback to compute recommendations. Differently from some of the approaches previously described, in `TimePop` we avoid both the use of a time window and the selection of a fixed number of candidate items. Indeed, while on the one hand, a time window may severely restrict the selection of candidates, on the other hand, a fixed number of candidate items may heavily affect the algorithm results.

2 Time-aware Local Popularity

The leading intuition behind `TimePop` is that the popularity of an item has not to be considered as a global property but it can be personalized if we consider the popularity in a neighborhood of users. We started from this observation to formulate a form of personalized popularity, and then we added the temporal dimension to strengthen this idea.

In `TimePop`, given a user u the first step is then the identification of user's neighbors who rated the same items as u but before u. We name these users *Precursors*. In our intuition, Precursors represent a community of users u relies on to choose the items to enjoy. In a neighborhood of u, the same item is enjoyed by users in different time frames. This leads us to the second ingredient behind `TimePop`: personalized popularity is a function of time. The more the ratings about an item are recent, the more its popularity is relevant for the specific user.

Hence, in order to exploit the temporal aspect of these ratings, the contributions of *Precursors* can be weighted depending on their freshness.

We now introduce some basic notation that will be used in the following. We use $u \in U$ and $i \in I$ to denote users and items respectively. Since we are not just interested in the items a user rated but also at when the rating happened, we have that for a user u the corresponding user profile is $P_u = \{(i_1, t_{ui_1}), \dots, (i_n, t_{ui_n})\}$ with $P_u \subseteq I \times \Re$, being t_{ui} a timestamp representing when u rated i.

Definition 1 (Candidate Precursor and Precursor). *Given* $(i, t_{ui}) \in P_u$ *and* $(i, t_{u'i}) \in P_{u'}$*, we say that* u' *is a* ***Candidate Precursor*** *of* u *if* $t_{u'i} < t_{ui}$*. We use the set* $\hat{¶}^u$ *to denote the set of Candidate Precursors of* u*. Given two users* u' *and* u *such that* u' *is a Candidate Precursor of* u *and a value* $\tau_u \in \Re$ *we say that* u' *is a* ***Precursor*** *of* u *if the following condition holds.*

$$|\{i \mid (i, t_{ui}) \in P_u \wedge (i, t_{u'i}) \in P_{u'} \wedge t_{u'i} < t_{ui})\}| \geq \tau_u$$

We use $¶^u$ *to denote the set of* ***Precursors*** *of* u*.*

A user u' is a *Candidate Precursor* of u if u' rated at least one common item i before u. Although this definition catches the intuition behind the idea of *Precursors*, it is a bit weak as it considers also users u' who have only a few or even just one item in common with u and rated them before she did. Hence, we introduced a threshold taking somehow into account the number of common items in order to enforce the notion of *Precursors*. The threshold parameter τ_u in Defintion 1 can be also computed automatically as:

$$\tau_u = \frac{\sum_{u' \in \hat{¶}^u} |\{i \mid (i, t_{ui}) \in P_u \wedge (i, t_{u'i}) \in P_{u'} \wedge t_{u'i} < t_{ui})\}|}{|\hat{¶}^u|} \tag{1}$$

To give an intuition on the computation of Precursors and of τ_u let us describe the simple example shown in Figure 1.

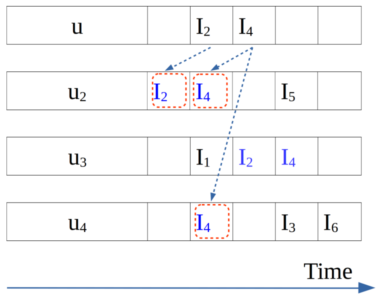


Fig. 1: Example of Precursors computation.

Here, for the sake of simplicity, we suppose that there are only four users and six items and u is the user we want to provide recommendations to. Items that users share with u are highlighted in blue and items with a dashed red square are the ones that have been rated before u. We see that $\hat{¶}^u = \{u_2, u_4\}$. Indeed, although u_3 rated some of the items also rated by u they have been rated after. By Equation (1) we have $\tau_u = \frac{3}{2} = 1.5$. Then, only u_2 results to be in $¶^u$ because she has $2 > 1.5$ shared items rated before those of u. As for u_3, it is more likely that u is a Precursor of u_3 and not vice versa.

Temporal decay. As the definition of Precursor goes through a temporal analysis of user behaviors, we may look at the timestamp of the last rating provided by a Precursor in order to identify how active she is in the system. Intuitively, the contribution to popularity for users who have not contributed recently with a rating is lower than "active" users. On the other side, given an item in the profile of a Precursor we are interested in the freshness of its rating. As a matter of fact, old ratings should affect the popularity of an item less than newer ratings. Summing up, we may classify the two temporal dimensions as **old/recent user** and **old/recent item**. In order to quantify these two dimensions for Precursors we introduce the following timestamps: $\mathbf{t_0}$ this is the reference timestamp. It represents the "now" in our system; $\mathbf{t_{u'i}}$ is the time when u' rated i; $\mathbf{t_{u'l}}$ represents the timestamp associated to the last item l rated by the user u'. Different temporal variables are typically used [8, 14], and they mainly focus on **old/recent items**. ΔT may refer to the timestamp of the items with reference to the last rating of u' [8] with $\Delta T = \mathbf{t_{u'l}} - \mathbf{t_{u'i}}$ or to the reference timestamp [14] with $\Delta T = \mathbf{t_0} - \mathbf{t_{u'i}}$. As we stated before, our approach captures the temporal behavior of both **old/recent users** and **old/recent items** at the same time. We may analyze the desired ideal behavior of ΔT depending on the three timestamps previously introduced as represented in Table 1.

	recent user ($\mathbf{t_0} \approx \mathbf{t_{u'l}}$)	**old user** ($\mathbf{t_0} \gg \mathbf{t_{u'l}}$)
recent item ($\mathbf{t_{u'l}} \approx \mathbf{t_{u'i}}$)	≈ 0	$\mathbf{t_0} - \mathbf{t_{u'l}}$
old item ($\mathbf{t_{u'l}} \gg \mathbf{t_{u'i}}$)	$\mathbf{t_{u'l}} - \mathbf{t_{u'i}}$	$\mathbf{t_0} - \mathbf{t_{u'l}}$

Table 1: *Ideal values of ΔT w.r.t. the Precursor characteristics*

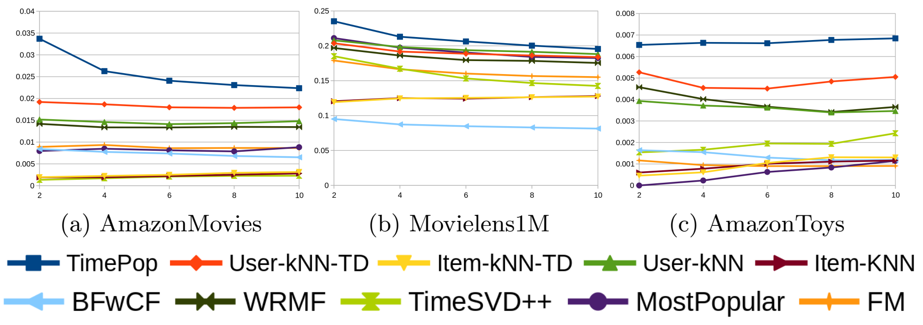
Let us focus on each case. In the upper-left case we want ΔT to be as small as possible because both u' and the rating for i are "recent" and then highly representative for a popularity dimension. In the upper-right case, the rating is recent but the user is old. The last item has been rated very close to i but a large value of ΔT should remain because the age of u' penalizes the contribution. The lower-left case denotes a user that is active on the system but rated i a long time ago. In this case the contribution of this item is almost equal to the age of its rating. The lower-right case is related to a scenario in which both the rating and u' are old. In this scenario, the differences between the reference timestamp minus the last interaction and the reference timestamp minus the rating of i are comparable: $(\mathbf{t_0} - \mathbf{t_{u'l}}) \approx (\mathbf{t_0} - \mathbf{t_{u'i}})$. In this case, we wish the contribution of ΔT to consider the elapsed time from the last interaction (or the rating) until the reference timestamp. All the above observations lead us to define $\Delta T = |\mathbf{t_0} - 2\mathbf{t_{u'l}} + \mathbf{t_{u'i}}|$. In order to avoid different decay coefficients, in our experimental evaluation, all ΔTs are transformed in days (from milliseconds) as a common practice.

The Recommendation Algorithm. We modeled our algorithm `TimePop` to solve a *top-N* recommendation problem. Given a user u, `TimePop` computes the recommendation list by executing the following steps:
1. Compute $\P^u$;
2. For each item i such that there exists $u' \in \P^u$ with $(i, t_{u'i}) \in P_{u'}$ compute a score for i by summing the number of times it appears in $P_{u'}$ multiplied by the corresponding decay function;
3. Sort the list in decreasing order with respect to the score of each i.
For sake of completeness, in case there were no precursors for a certain user, a recommendation list based on global popularity is returned to u. Moreover, if

TimePop is able to compute only m scores, with $m < N$, the remaining items are returned based on their value of global popularity.

3 Experimental Evaluation

In order to evaluate TimePop we tested our approach considering datasets related to different domains. Two of them related to the movie domain —the well-known Movielens1M dataset and Amazon[3] Movies — and a dataset referring to toys and games —Amazon Toys and Games, with 2M ratings and a sparsity of 99.99949%. "All Unrated Items" [21] protocol has been chosen to compare different algorithms where, for each user, all the items that have not yet been rated by the user all over the platform are considered. In order to evaluate time-aware recommender systems in an offline experimental setting, a typical k-folds or hold-out splitting would be ineffective and unrealistic. To be as close as possible to an online real scenario we used the fixed-timestamp splitting method [6, 10], also used in [5] but with a *dataset centered* base set. The basic idea is choosing a single timestamp that represents the moment in which test users are on the platform waiting for recommendations. Their past corresponds to the training set, and the performance is evaluated with data coming from their future. In this work, we select the splitting timestamp that maximizes the number of users involved in the evaluation by setting two constraints: the training set must keep at least 15 ratings, and the test set must contain at least 5 ratings. Training set and test set for the three datasets are publicly available[4] along with the splitting code for research purposes. In order to evaluate the algorithms we measured *normalized Discount Cumulative Gain*@N (*nDCG*@*N*) using *Time-independent rating order condition [6]*. The metric was computed per user and then the overall mean was returned using the RankSys framework and adopting *Threshold-based relevant items* condition [6]. The threshold used to consider a test item as relevant has been set to the value of 4 w.r.t. a 1-5 scale for all the three datasets.



(a) AmazonMovies (b) Movielens1M (c) AmazonToys

Fig. 2: nDCG @N varying N in 2..10

Baselines. We evaluated our approach w.r.t CF and time-aware techniques. **MostPopular** was included as TimePop is a time-aware variant of "Most Popular". From model-based collaborative filtering approaches we selected some of

[3] http://jmcauley.ucsd.edu/data/amazon/

[4] https://github.com/sisinflab/DatasetsSplits

the best performing matrix factorization algorithms **WRMF** trained with a regularization parameter set to 0.015, α set to 1 and 15 iterations, and **FM**[5][25], computed with an ad-hoc implementation of a 2 degree factorization machine considering users and items as features, trained using Bayesian Personalized Ranking Criterion[19]. Moreover, we compared our approach against the most popular memory-based kNN algorithms, **Item-kNN**[5] and **User-kNN**[??] [20], together with their time-aware variants (**Item-kNN-TD**[5], **User-kNN-TD**[5])[8]. We included **TimeSVD++**[5] [14] in our comparison even though this latter has been explicitly designed for the rating prediction task. All model-based algorithms were trained using 10, 50, 100, and 200 factors; only best models are reported in the evaluation: for Movielens1M WRMF 10, FM 10; for Amazon Movies WRMF 100, FM 200; for Amazon Toys and Games WRMF 100, FM 50. Finally **BFwCF** [5] is an algorithm that takes into account interaction sequences between users and it uses the last common interaction to populate the candidate items list. In this evaluation we included the BFwCF variant that takes advantage of similarity weights per user and two time windows, left-sided and right-sided (Backward-Forward). BFwCF was trained using parameters from [5]: 100 neighbors, *indexBackWards* and *indexForwards* set to 5, normalization and combination realized respectively via *DummyNormalizer* and *SumCombiner*. Recommendations were computed with the implementation publicly provided by authors. In order to guarantee a fair evaluation, for all the time-based variants the β coefficient was set to $\frac{1}{200}$ [14]. TimeSVD++ was trained using parameters used in [14].
Results Discussion. Results of experimental evaluation are shown in Figure 2 which illustrate nDCG (2a, 2b, 2c) curves for increasing number of top ranked items returned to the user. Significance tests have been performed for accuracy metrics using Student's t-test and p-values and they result consistently lower than 0.05. By looking at Figure 2a we see that `TimePop` outperforms comparing algorithms in terms of accuracy on AmazonMovies dataset. We also see that algorithms exploiting a Temporal decay function perform well w.r.t. their time-unaware variants (User-kNN and Item-kNN) while matrix factorization algorithms (WRMF ,TimeSVD++ and FM) perform quite bad. The low performance of MF algorithms is very likely due to the temporal splitting that makes them unable to exploit collaborative information. We may assume that the good performance of `TimePop` w.r.t. kNN algorithms are due to the adopted threshold, that emphasizes the popular items, and hence increases accuracy metrics values. Results for Amazon Toys and Games dataset are analogous to those computed for Amazon Movies. Results for Movielens additionally show that the high number of very popular items make neighborhood-based approaches perform similarly.

4 Conclusion

In this paper we presented `TimePop`, a framework that exploits local popularity of items combined with temporal information to compute top-N recommendations. The approach relies on the computation of a set of time-aware neighbors named Precursors that are considered the referring population for a user we want to serve recommendations. We compared `TimePop` against state-of-art algorithms showing its effectiveness in terms of accuracy despite its lower computational cost in computing personalized recommendations.

[5] `https://github.com/sisinflab/recommenders`

References

1. G. Adomavicius and A. Tuzhilin. Multidimensional recommender systems: a data warehousing app.roach. *Electronic commerce*, pp. 180–192, 2001.
2. G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pp. 217–253, 2011.
3. V. Anelli, T. Di Noia, E. Di Sciascio, and P. Lops. Feature factorization for top-n recommendation: From item rating to features relevance. In Proc. of RecSysKTL, pp. 16–21, 2017.
4. H. Bao, Q. Li, S. S. Liao, S. Song, and H. Gao. A new temporal and social pmf-based method to predict users' interests in micro-blogging. *Decision Supp.ort Systems*, 55(3):698–709, 2013.
5. A. Bellogín and P. Sánchez. Revisiting neighbourhood-based recommenders for temporal scenarios. In Proc. of TempRec, pp. 40–44, 2017.
6. P. G. Campos, F. Díez, and I. Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *UMAI*, 24(1-2):67–119, 2014.
7. P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In Proc. of RecSys '10, pp. 39–46, 2010.
8. Y. Ding and X. Li. Time weight collaborative filtering. In Proc. of CIKM '05, pp. 485–492. ACM, 2005.
9. I. Fernández-Tobías, M. Braunhofer, M. Elahi, F. Ricci, and I. Cantador. Alleviating the new user problem in collaborative filtering by exploiting personality information. *UMUAI*, 26(2-3):221–255, 2016.
10. A. Gunawardana and G. Shani. Evaluating recommender systems. In *Recommender Systems Handbook*, pp. 265–308. 2015.
11. T. Jambor and J. Wang. Optimizing multiple objectives in collaborative filtering. In Proc. of RecSys '10, pp. 55–62, 2010.
12. D. Jannach, L. Lerche, F. Gedikli, and G. Bonnin. What recommenders recommend - an analysis of accuracy, popularity, and sales diversity effects. In Proc. of UMAP '13, pp. 25–37, 2013.
13. M. Jugovac, D. Jannach, and L. Lerche. Efficient optimization of multiple recommendation quality factors according to individual user tendencies. *Expert Syst. App.l.*, 81:321–331, 2017.
14. Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.
15. N. Lathia, S. Hailes, and L. Capra. Temporal collaborative filtering with adaptive neighbourhoods. In Proc. of SIGIR '09, pp. 796–797, 2009.
16. N. N. Liu, M. Zhao, E. Xiang, and Q. Yang. Online evolutionary collaborative filtering. In Proc. of RecSys '10, pp. 95–102, 2010.
17. J. Oh, S. Park, H. Yu, M. Song, and S. Park. Novel recommendation based on personal popularity tendency. In Proc. of ICDM '11, pp. 507–516, 2011.
18. S. Rendle. Factorization machines. In Proc. of ICDM '10, pp. 995–1000, 2010.
19. S. Rendle et al. BPR: bayesian personalized ranking from implicit feedback. In Proc. of UAI '09, pp. 452–461, 2009.
20. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In Proc. of EC '00, pp. 158–167, 2000.
21. H. Steck. Evaluation of recommendations: rating-prediction and ranking. In Proc. of RecSys'13, pp. 213–220, 2013.
22. C. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing. Recurrent recommender networks. In Proc. of WSDM 2017, pp. 495–503, 2017.
23. C. Xia, X. Jiang, S. Liu, Z. Luo, and Z. Yu. Dynamic item-based recommendation algorithm with time decay. In Proc. of ICNC '10, pp. 242–247, 2010.
24. A. Zimdars, D. M. Chickering, and C. Meek. Using temporal data for making recommendations. In Proc. of UAI '01, pp. 580–588, 2001.
25. S. Rendle. Using temporal data for making recommendations. In Proc. of ICDM '10, pp. 995–1000, 2001.