Online ranking prediction in non-stationary environments

Erzsébet Frigó Róbert Pálovics Domokos Kelen Levente Kocsis András A. Benczúr
Institute for Computer Science and Control
Hungarian Academy of Sciences (MTA SZTAKI)
{rpalovics, kelen.domokos.miklos, kocsis, benczur}@sztaki.hu

ABSTRACT

Recommender systems have to serve in online environments which can be highly non-stationary.[1]. Traditional recommender algorithms may periodically rebuild their models, but they cannot adjust to quick changes in trends caused by timely information. In our experiments, we observe that even a simple, but online trained recommender model can perform significantly better than its batch version. We investigate online learning based recommender algorithms that can efficiently handle non-stationary data sets. We evaluate our models over seven publicly available data sets. Our experiments are available as an open source project[2].

1 INTRODUCTION

The research of recommender systems became popular since the Netflix prize [4]. As an effect of the competition, *batch rating prediction* is considered the standard recommender evaluation task, with one part of the data used for model training, and the other for evaluation. In contrast to the Netflix Prize task, recommender systems are typically not required to predict ratings, rather they are required to present a ranked top list of relevant items for the user. Also, most users give no explicit ratings and we have to infer their preferences from implicit feedback [17]. Most importantly, users request one or a few items at a time, and may get exposed to new information that can change their needs and taste before they return to the service the next time. In a real application, *top item recommendation by online learning* is therefore more relevant than batch rating prediction usually. There is vast literature on the batch performance of collaborative filtering algorithms, while the more realistic sequential or online evaluation received much less attention. Information on recommendation performance is scarce when collaborative filtering is evaluated online, sequentially.

In this work we intend to consider top recommendation in highly non-stationary environments similarly to the conceptually simpler classification task [22]. Our goal is to promptly update recommender models after user interactions using online learning methods.

There are some indications [19] that the performance gains in batch experiments do not necessary carry over to online learning environments. One result of the Netflix prize is that matrix factorization [18] techniques provide strong baseline results on non-temporal data sets. In our work we compare the online and batch versions of the same matrix factorization algorithm. Due to the highly non-stationary properties of the data set, we apply incremental stochastic gradient descent (SGD) based matrix factorization. We show that the online version strongly outperforms the batch version on non-stationary data sets.

In [23], the online DCG measure is defined as a side result to track the performance of recommender algorithms over time. In our present experiments, we heavily rely on online DCG, introduce another measure, the online MRR, and show their importance for evaluating streaming recommenders. Summarized, our main results are the following:

- We provide some measures suited to non-stationary environments that are predictive of which algorithm would work well on a particular data set.
- We show that simpler algorithms that can be updated online—and therefore use the most recent data as well—often perform better than more complex algorithms that can only be updated periodically. This is in contrast to the case when the testing is batch or the data is stationary.
- We also show that even though initialization by large batch models may be required for optimal performance, online matrix factorization combined with a simple sampling strategy can perform comparably to more computationally intensive models.

1.1 Related Results

The difficulty of evaluating ***streaming recommenders*** was first mentioned in [19], although they evaluated models by offline training and testing split. Ideas for ***online evaluation metrics*** appeared first in [23, 24, 33]. In [33], incremental algorithms are evaluated using recall, which is a batch quality measure. In the other two results, online evaluation appears as a side result of investigating social influence. As the starting point of our results, the last paper notices that some online methods perform better than their batch counterpart.

Since our goal is to recommend different items at different times, our evaluation must be based on the quality of the top list produced by the recommender. This so-called ***top-k*** recommender task is known to be hard [7]. A recent result on evaluating top-k recommenders is found in [6].

In our experiments, we apply widely used methods of ***matrix factorization***. The highly successful gradient descent matrix factorization recommender is described among others by [11, 30]. Note that even though typically performing marginally better than gradient descent, we do not investigate alternating least squares [18, 27], since it is an inherently batch algorithm that cannot be easily adapted to the online setting.

We also evaluate basic ***item-to-item recommenders***. Recent works [26] show that most industry recommendation tasks are item-to-item, since the only information available is the present

[1]
[2] https://github.com/rpalovics/Alpenglow

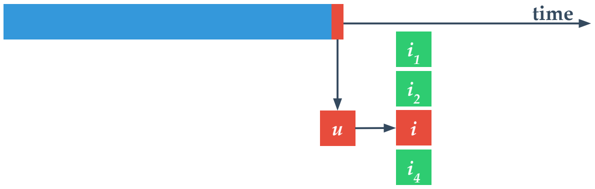

Figure 1: Temporal evaluation of the online ranking prediction problem.

user session. The first item-to-item recommender methods [20, 29] were using similarity information to directly find nearest neighbor [8] transactions. Nearest neighbor methods were criticized for two reasons. First, similarity metrics typically have no mathematical justification. Second, the confidence of the similarity values is often not involved when finding the nearest neighbor, which leads to overfitting in sparse cases. A new method to give session recommendations was described in [28] by Rendle et al. that models the users by factorizing personal Markov chains. Koenigstein and Koren [17] improve the basic transition model by computing latent item factors to represent all items in a low dimensional space. While certainly giving performance gains in batch testing, these algorithms are overly costly to be updated online.

Finally we mention that item-to-item recommendation was also considered as a special context aware recommendation problem. In [14] sequentiality as context is handled using pairwise associations as features in an alternating least squares model by Hidasi et al. They mention that they face the sparsity problem in setting minimum support, confidence and lift of the associations and they use the category of last purchased item as fallback. In a follow-up result [15], they use the same context-aware ALS algorithm, however they only consider seasonality as context in that paper. Our result can be used independently of the ALS based methods and can easily be combined with user personalization. Most recently, context information learning was also performed by recurrent neural networks [13].

2 TEMPORAL EVALUATION

In the implicit top-k recommendation task [6], the goal is not to rate individual items, but to recommend the best candidates. In a time sensitive or online recommender that potentially re-trains the model after each new item, we have to generate a new top list for every recommendation request. The online top-k task is therefore different from the standard recommender evaluation settings, since there is always only a single relevant item. In an online setting, as seen in Figure 1, we

(1) request a top-k recommendation for the active user,
(2) evaluate against the single relevant item,
(3) train the model on the revealed user-item pair.

Next we introduce natural evaluation techniques for the online ranking prediction problem, extending the methods of [23]. In our setting, model training and evaluation happen simultaneously, iterating over the data set only once, in chronological order. Whenever we see a new user-item pair, we assume that the user becomes active and requests a recommendation. The recommendation is online, hence it may depend on all events before the exact time of the interaction. If a user u views item i at time t, our models predict a score $\hat{r}(u, i', t)$ for each item i' that appears in the data so far, and recommend the k items with the largest values from those that u has not seen before.

One possible measure for the quality of a recommended top list could be precision and recall [35, 36]. Note that we evaluate against a single item. Both the number of relevant (1) and the number of retrieved (K) items are fixed. Precision is $1/K$ if we retrieve the single item viewed and 0 otherwise. Recall is 0 if we do not retrieve the single relevant item and 1 otherwise. Hence up to a constant factor K, precision and recall are identical and are binary indicators of whether the item viewed is on the recommended list.

Recently, measures other than precision and recall became preferred for measuring the quality of top-k recommendation [2]. The most common measure is NDCG, the normalized version of the discounted cumulative gain (DCG). Since the decrease of DCG as the function of the rank is smoother than the decrease of precision and recall, it is more advantageous, since we have a large number of items of potential interest to each user. Our choice is in accordance with the observations in [2] as well.

DCG computed individually for each event and averaged in time is an appropriate measure for real-time recommender evaluation. If i is the next consumed item by the user, the ***online DCG@K*** is defined as the following function of the rank of i returned by the recommender system:

$$\mathrm{DCG@K}(i) = \begin{cases} 0 & \text{if } \mathrm{rank}(i) > K; \\ \dfrac{1}{\log_2(\mathrm{rank}(i) + 1)} & \text{otherwise.} \end{cases} \tag{1}$$

The overall evaluation of a model is the average of the DCG@K values over all events of the evaluation period. Note that in our unusual temporal setting of DCG evaluation, there is a single relevant item and hence no normalization is needed as opposed to the original NDCG measure.

Another possible ranking evaluation measure with a natural online extension is Mean Reciprocal Rank (MRR), the average of $1/\mathrm{rank}$ of the first relevant item [34]. In online evaluation, there is a single relevant item i, therefore ***online MRR@K*** is equal to $1/\mathrm{rank}(i)$ if $\mathrm{rank}(i) \leq K$ and is 0 otherwise. Hence the difference between online DCG and MRR the rank discount function, which is reciprocal for MRR and inverse logarithmic for DCG.

3 ALGORITHMS

In this section, we describe the algorithms used in our experiments, both in batch and online setting. The simplest algorithms, for example the count based ones, are online by nature. Most of the algorithms discussed have more or less natural online counterparts. Our goal is to include a variety of count, nearest neighbor, session and matrix factorization based methods to see how these algorithms can adapt to non-stationary environments. Many of the algorithms will have parameters to handle decay in time, i.e. forget older events and emphasize new ones.

Online recommenders seem more restricted, since they may not iterate over the data several times, hence we would expect inferior quality. Online methods however have the advantage of putting much more emphasis on recent events. In an online setting [1], the model needs to be retrained after each new event and hence reiterations over the earlier parts of the data is ruled out. We may implement an online recommender algorithm by allowing a single iteration over the training data only, with this single iteration processing the events in the order of time.

3.1 Popularity (POP)

In this non-personalized method, we recommend the most popular items. The method may be both batch and online depending on the granularity of the item frequency measurement update. For each data set, we selected the best time frame, which had a wide range of variety between 10 minutes and a year.

3.2 Time sensitive nearest neighbor (t-NN)

One of the earliest and most popular collaborative filtering algorithms in practice is the item-based nearest neighbor [29]. For these algorithms similarity scores are computed between item pairs based on the co-occurrence of the pairs in the preference of users. Non-stationarity of the data can be accounted for e.g. with the introduction of a time-decay [9].

Describing the algorithm more formally, let us denote by U_i the set of users that visited item i, by I_u the set of items visited by user u, and by s_{ui} the index of item i in the sequence of interactions of user u. The frequency based time-weighted similarity function is defined by

$$sim(j,i) = \frac{\sum_{u \in U_j \cap U_i} f(s_{ui} - s_{uj})}{\left|U_j\right|}, \tag{2}$$

where $f(\tau) = \gamma^\tau$ is the time decaying function. For non-stationary data we sum only over users that visit item j before item i, setting $f(\tau) = 0$ if $\tau < 0$. For stationary data the absolute value of τ is used. The score assigned to item i for user u is

$$score(u,i) = \sum_{j \in I_u} f\left(|I_u| - s_{uj}\right) sim(j,i). \tag{3}$$

The model is represented by the similarity scores. Since computing the model is time consuming, it is done periodically. Moreover, only the most similar items are stored for each item. When the prediction scores are computed for a particular user, all items visited by the user can be considered, including the most recent ones. Hence, the algorithm can be considered semi-online in that it uses the most recent interactions of the current user, but not of the other users. We note that the time decay function is used here to quantify the strength of connection between pairs of items depending on how closely are located in the sequence of a user, and not as a way to forget old data as in [9].

3.3 Item-to-item transition model (TRANS)

A simple algorithm that focuses on the sequence of items a user has visited is one that records how often users visited item i after visiting another item j. This can be viewed as particular form of the item-to-item nearest neighbor with a time decay function that is non-zero only for the immediately preceding item. While the algorithm is more simplistic, it is fast to update the transition frequencies after each interaction, thus all recent information is taken into account.

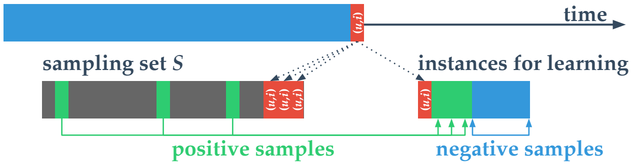

Figure 2: Sampling from the past.

3.4 Batch and online matrix factorization (MF)

One of our methods is the popular gradient descent matrix factorization recommender described among others by [11, 30]. The original algorithm builds a batch model by iterating over a fixed training data set a certain I number of times in random order, performing stochastic gradient descent, until convergence. The algorithm builds F *latent factors*, P_{uf} for user u and Q_{if} for item i, to predict the rating by

$$\hat{r}(u,i) = \sum_{f=1}^{F} P_{uf} Q_{if}. \tag{4}$$

Given an actual rating $r(u,i)$, the factors P_{uf} are updated by regularized gradient descent as

$$P_{uf}^{(t+1)} = P_{uf}^{(t)} + \eta \cdot (r(u,i) - \hat{r}(u,i)) \cdot Q_{if} - \eta \lambda P_{uf}^{(t)}, \tag{5}$$

where η is the *learning rate* and λ is the *regularization coefficient*. Item factors Q_{if} are updated similarly.

In our tasks, all of the ratings are implicit. Whenever we observe a user-item interaction (u,i), we may only infer positive feedback $r(u,i) = 1$. We follow the approach of [27] by treating all unknown elements (u',i') of the matrix as negative feedback, i.e we assume $r(u',i') = 0$. In order to avoid the model fitting to $r \equiv 1$, we generate *negative samples* for each item interaction. In other words, after training on (u,i), we train according to equation (5) for a fixed *negative rate nRate* number of (u,i') pairs. Item i' is drawn uniformly *nRate* times for user u from the set of items that the u has not interacted with.

Online matrix factorization takes the same steps, including negative sample generation, but strictly in the order of the events. In the update equation (5), we may consider the superscript t as time, and process interactions (u,i,t) in the order of t. For negative samples, we generate items i' such that no (u,i',t') interaction exists with $t' \leq t$.

A natural option to combine the advantages of the batch and the online model is to periodically (e.g. weekly) build batch factors P and Q and continue from the same P and Q with the online gradient descent method. We call this method ***batch & online***.

3.5 Sampling from the past for matrix factorization

While batch & online matrix factorization naturally combines the advantages of the two methods, the periodic re-training of the batch model may be computationally inefficient. We propose an online, efficient technique that approximates the batch and online combination. Similar to the online MF model, we only allow a single iteration for the model in a temporal order. However, for each interaction, we generate not only negative but also positive samples. The positive samples are selected randomly from past interactions, i.e. we allow the model to re-learn the past. We generate *pRate* positive samples along with *nRate* negative samples, hence for t events, we only take $(1 + nRate + pRate) \cdot t$ gradient steps.

The samples are not drawn uniformly from the past, but selected randomly from pool S with maximum size s. This avoids oversampling interactions that happened at the beginning of the data set. More specifically, as seen in Fig. 2, for each observed new training instance, we

- update the model by *pRate* random samples from S,
- delete the selected samples from S if $|S| > s$,
- and add the new instance *pRate* times to S.

3.6 Asymmetric Matrix Factorization (AMF)

Asymmetric Matrix Factorization [25] computes item-to-item similarity with the help of latent vectors for items. In contrast to Section 3.4, both latent matrices P and Q correspond to items. Using the notations and time decaying function from Section 3.2, the score assigned to item i for user u is:

$$score(u, i) = \sum_{j \in I_u} f\left(|I_u| - s_{uj}\right) P_j Q_i. \tag{6}$$

Sampling negative instances and updating the latent vectors online using stochastic gradient descent can be done in a similar way to the one described in Section 3.4.

4 DATA

We compare the performance of the batch and the online algorithms of Section 3 over seven data sets described next. For each data set, we discard items that appear less than ten times in interactions. We apply no filtering for the users.

Twitter. We use 400M tweets over four months between February 1 and May 30, 2012 from a crawl introduced in [10]. We recommend new hashtags that the user has not used before.

Last.fm We recommend artists and tracks in the 30Music data set [31]. New tracks appear more frequently than new artists. We expect that user-track events are highly non-stationary, more than user-artists events.

MovieLens data sets [12], first released in 1998, contain timestamped explicit movie ratings. We use the ML10M data set, consisting of 10,000,064 ratings of 69,878 users for 10,681 movies. In our experiments, we consider these records as implicit interactions.

Amazon data sets [21] are browsing and co-purchasing data crawled from Amazon. We use four categories, ***Books***, ***Electronics*** as well as ***Movies and TV***.

4.1 Statistics

Before turning to explaining the findings, we show two statistics that highlight the non-stationary properties of the data sets.

Burstiness. The inter-event time (Fig. 3) is the time elapsed between consecutive events of the same user. In accordance with several results [3, 16, 32], the distribution is highly skewed for all data, which show their burstiness in the approximate order from strongest to weakest as Twitter, Amazon, Last.fm and finally MovieLens (Fig. 3, right to left).

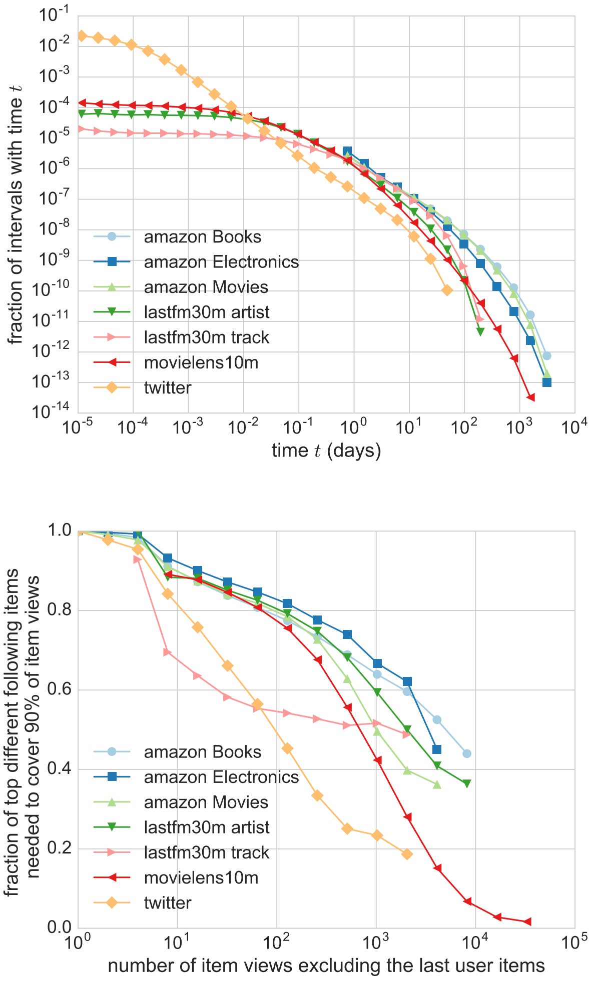

Figure 3: Statistics over the 7 datasets. Top: Inter-event distribution. Bottom: Transition matrix statistics.

Item-to-item transitions. Users of online music services very often follow playlists, which result in artificially very strong performance of the item-to-item methods. When listening to playlists, users do not actually request recommendations and hence evaluating user sequences for recommendation quality is somewhat artificial. Next we examine if the data sets contain trivial item-to-item transitions by computing the item-to-item transition matrix, i.e. the number of times j followed i in the music listening session of the same user. For each item i we count the least number of unique $i \longrightarrow j$ item transitions needed to cover 90% of the item's transitions. In Figure 3 we plot the fraction of these against the total number of unique $i \longrightarrow j$ transitions. We observe that for the Last.fm track data, the MovieLens data set, and for the Twitter data lower fractions are possible, hence these data sets may involve trivial transitions.

Shuffling the temporal order. In order to assess the importance of the time dimension in the streaming recommendation task, we compare our results over the same data sets but with randomly shuffled order of time. In other words, we evaluate over the shuffled stationary variant of each data, where we expect that the temporal effects disappear.

5 RESULTS

In this section, we describe our results, including the performance of the algorithms described in Section 3 and our experiments with matrix factorization training strategies. In Table 1, we summarize the performance of our models by online DCG. Online MRR behaves almost identical and is not shown for space limitations.

Twitter data is very sparse and bursty with the highest fraction of small inter-events (Fig. 3). As a result, factorization methods perform worse and popularity models perform well. The Last.fm track data set is similar, but it incorporates playlist and album listening data, hence transition and similarity models perform better than factorization and popularity. The number of items in the Last.fm artist data are significantly lower than for the track data: as a result factorization gets close to transition and similarity models on this data set . In Fig. 3 we can observe that the Last.fm track data, the MovieLens data set, and the Twitter data contain several predictable item-to-item transitions. This observation is consistent with performance of the transition model, as it provides the strongest DCG values in case of these three data sets.

For batch rating prediction, MF is known in the literature to perform very strong. For ranking prediction, however, our results show the superiority of item-to-item nearest neighbor methods. Furthermore, performance of batch MF models drop considerably when the data is non-stationary. For example, for MovieLens, MF is the best performing algorithm after shuffling, but the worst for the original temporal order. Popularity, the simplest algorithm, performs well for many data sets in non-stationary setting, although the strong performance is mainly due to users with few interactions. AMF performs between MF and NN, slightly better for non-stationary data. The transition model is constantly inferior to NN, but despite its simplicity, it is competitive with more complex factorization models for non-stationary data.

5.1 Online training strategies for MF

The high performance of transition models indicates the existence of trivial item-to-item transitions. In music data, such as the Last.fm data set, the listening of predefined playlists or albums results in predictable item-to-item transitions. These are easy to predict, however they are not valuable for the actual user, therefore they bias recommender evaluation. We filtered the Last.fm artist data before comparing different factor model training strategies to reduce the above effects. We only considered records after at least 30 minutes of user inactivity, i.e. the start of each user session. Statistics of the filtered 30M data are shown in Fig. 4.

We compared multiple different learning strategies of MF models: batch, online, batch & online, and sampling methods. In the experiments we use the following best parameters. We train batch MF weekly with $\eta = 0.05$, nRate $= 69$ and 9 iterations in random order. For online MF we set $\eta = 0.2$, nRate $= 99$. The parameters are the same for batch & online. For the sampling model, we use lower learning rate $\eta = 0.05$ and $nRate = 39$. Best results are achieved by generating $pRate = 3$ positive samples from the past for each record with pool size 3,000.

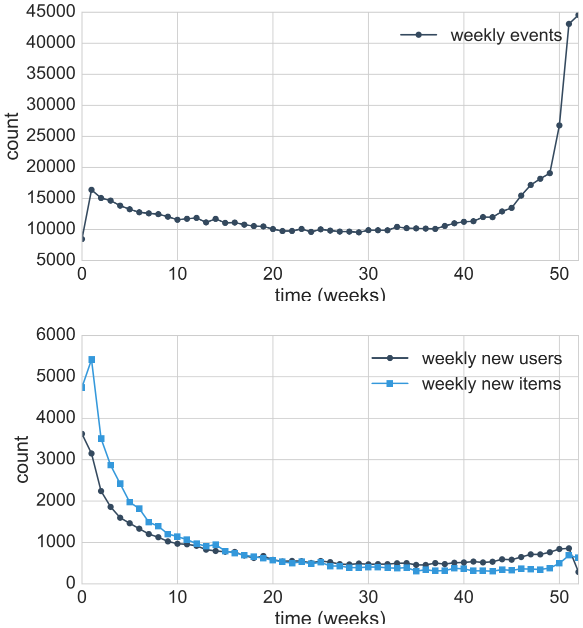

Figure 4: Statistics over the 30M data set. Top: number of records per week. Bottom: number of weekly new users and items.

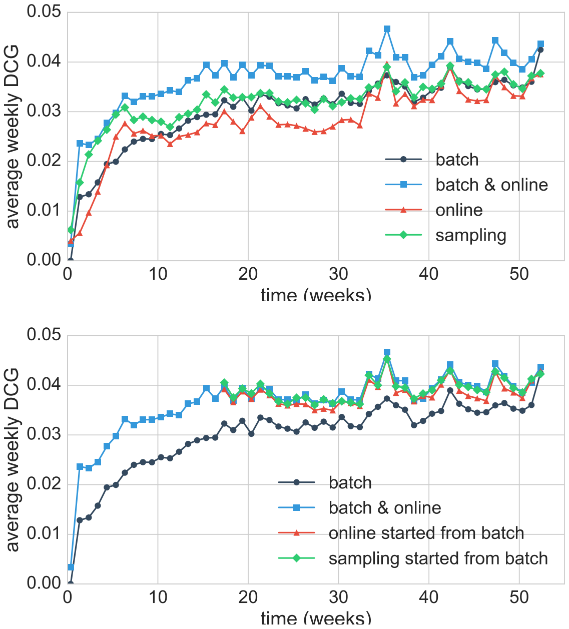

Figure 5: Performance of the sampling method compared to three MF variants: batch, online and batch & online. Top: each model trained individually from the beginning of time. Bottom: the sampling and online models are started from a batch model at week 18.

	original	POP	TRANS	t-NN	batch MF		online MF, F =			batch	AMF	SVD
					full	sample	10	100	1,000	& online		++
	Twitter all	**0.350**	0.149	0.027	0.010	0.006	0.335	0.337	0.340	0.269	0.332	0.348
	Twitter from 10th	0.416	0.267	0.036	0.008	0.005	0.523	**0.527**	0.532	0.457	0.516	0.516
	Last.fm track	0.006	0.276	**0.298**	0.007	0.009	0.022	0.066	0.010	0.021	0.030	0.028
	Last.fm artist	0.024	0.051	**0.064**	0.032	0.031	0.048	0.042	0.004	0.049	0.049	0.051
	MovieLens10M	0.086	0.148	0.148	0.012	0.012	0.140	0.142	0.038	0.133	0.167	**0.172**
Amazon	Books all	0.026	0.033	**0.042**	0.008	0.007	0.025	0.032	0.009	0.016	0.025	0.026
	Books from 10th	0.017	0.046	**0.074**	0.010	0.012	0.042	0.053	0.018	0.021	0.040	0.042
	Electronics	**0.035**	0.015	0.014	0.006	0.006	0.016	0.018	0.019	0.011	0.018	0.019
	Electronics from 10th	0.023	0.024	0.035	0.017	0.018	0.046	0.049	**0.050**	0.023	0.048	0.050
	Movies	**0.066**	0.029	0.034	0.023	0.007	0.035	0.039	0.014	0.024	0.035	0.037
	Movies from 10th	0.048	0.037	0.062	0.040	0.016	0.064	**0.068**	0.037	0.038	0.060	0.064
	shuffled			NN								
	Twitter all	0.020	0.041	**0.055**	0.050	0.045	0.034	0.039	0.045	0.050	0.044	0.044
	Twitter from 10th	0.017	0.047	**0.091**	0.073	0.082	0.060	0.068	0.074	0.073	0.066	0.067
	Last.fm track	0.004	0.002	**0.043**	0.010	0.021	0.008	0.009	0.002	0.010	0.006	0.008
	Last.fm artist	0.023	0.014	0.036	0.043	0.041	0.035	0.025	0.005	**0.043**	0.033	0.037
	MovieLens10M	0.059	0.058	0.065	0.088	0.085	0.080	0.052	0.004	**0.088**	0.072	0.080
Amazon	Books all	0.009	0.016	**0.027**	0.009	0.010	0.007	0.009	0.008	0.011	0.010	0.010
	Books from 10th	0.004	0.010	**0.042**	0.013	0.016	0.012	0.015	0.013	0.014	0.013	0.013
	Electronics	**0.020**	0.010	0.011	0.009	0.007	0.007	0.008	0.009	0.009	0.011	0.011
	Electronics from 10th	0.013	0.008	**0.025**	0.020	0.012	0.019	0.020	0.020	0.020	0.021	0.021
	Movies	0.023	0.018	**0.023**	0.016	0.014	0.011	0.012	0.013	0.016	0.015	0.015
	Movies from 10th	0.007	0.010	**0.034**	0.024	0.024	0.019	0.021	0.021	0.024	0.019	0.020

Table 1: Online DCG@100 of different algorithms. Duplicate data marked "from 10th" are re-evaluation over items past the 10th interaction only, for each user. Top: original data. Bottom: *shuffled* data.

In Fig. 5 we plot the average weekly DCG over a one year period. While sampling appears to keep up with batch & online in the beginning, its performance drops in the second part. If we start each factor model from the batch model of week 18, sampling produces similar results to batch & online. In Fig. 6, as a function of k, we start with the batch model of week k. Then, we only use online updates with or without further sampling. It can be seen that sampling performs roughly the same as batch & online for $k > 10$. Note that the number of weekly incoming new users and items drops after the first 10 weeks as seen in Fig. 4. The single iteration online model produces a comparable, but slightly worse results than the sampling version.

In summary, in a non-stationary environment, multiple passes (i.e. batch models) are required over the data to fully incorporate many new users and items into the system. However, if a large component of the user-item matrix is previously learned by a robust batch model, a simple online sampling algorithm is sufficient to keep up in performance with periodic batch re-training, while requiring only 3 additional positive samples from the past per iteration and thus being much more efficient.

6 CONCLUSIONS

Despite the fact that a practical recommender system processes events and generates top lists in an online sequence, the literature payed relative little attention to designing and evaluating online learning methods in highly non-stationary environments. We proposed and evaluated a variety of algorithms as well as measures that are predictive of which algorithm would work well on a particular data. We showed that simpler algorithms that can use most recent data by updating their models online perform better than more complex algorithms that can be updated only periodically. We also showed that sampling from past events may completely replace

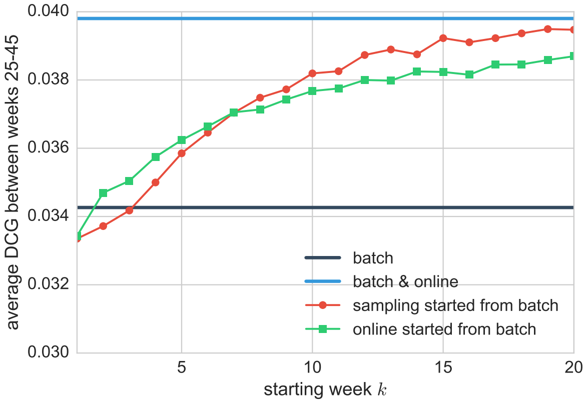

Figure 6: Performance of the online and sampling methods started from a batch model at week k. Averages are taken between week 25 and 45.

batch modeling needs in a real time recommender system, thus reducing latency. We released our code as an open source project[3].

[3] https://github.com/rpalovics/Alpenglow

REFERENCES

[1] J. Abernethy, K. Canini, J. Langford, and A. Simma. Online collaborative filtering. *University of California at Berkeley, Tech. Rep*, 2007.

[2] A. Al-Maskari, M. Sanderson, and P. Clough. The relationship between ir effectiveness measures and user satisfaction. In *Proc. SIGIR*, pp. 773–774. ACM, 2007.

[3] Albert-Laszlo Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005.

[4] J. Bennett and S. Lanning. The netflix prize. In *KDD Cup and Workshop in conjunction with KDD*, 2007.

[5] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games.* Cambridge Univ Pr, 2006.

[6] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. RecSys*, pages 39–46. ACM, 2010.

[7] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM TOIS*, 22(1):143–177, 2004.

[8] C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[9] Y. Ding and X. Li. Time weight collaborative filtering. In *Proc. CIKM*, pages 485–492. ACM, 2005.

[10] L. Dobos, J. Szule, T. Bodnar, T. Hanyecz, T. Sebok, D. Kondor, Z. Kallus, J. Stéger, I. Csabai, and G. Vattay. A multi-terabyte relational database for geo-tagged social network data. In *Proc. CogInfoCom*, pages 289–294. IEEE, 2013.

[11] S. Funk. Netflix update: Try this at home. http://sifter.org/˜simon/journal/20061211.html, 2006.

[12] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM TiiS*, 5(4):19, 2016.

[13] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.

[14] B. Hidasi and D. Tikk. Fast als-based tensor factorization for context-aware recommendation from implicit feedback. In *Machine Learning and Knowledge Discovery in Databases*, pages 67–82. Springer, 2012.

[15] B. Hidasi and D. Tikk. Context-aware item-to-item recommendation within the factorization framework. In *Proc. 3rd Workshop on Context-awareness in Retrieval and Recommendation*, pp. 19–25. 2013.

[16] Mikko Kivelä and Mason A Porter. Estimating inter-event time distributions from finite observation periods in communication networks. *arXiv preprint arXiv:1412.8388*, 2014.

[17] N. Koenigstein and Y. Koren. Towards scalable and accurate item-oriented recommendations. In *Proc RecSys*, pages 419–422. ACM, 2013.

[18] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[19] N. Lathia, S. Hailes, and L. Capra. Temporal collaborative filtering with adaptive neighbourhoods. In *Proc SIGIR*, pages 796–797. ACM, 2009.

[20] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[21] J. McAuley, R. Pandey, and J. Leskovec. Inferring networks of substitutable and complementary products. In *Proc SIGKDD*, pages 785–794. ACM, 2015.

[22] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *Proc SIGKDD*, pages 1222–1230. ACM, 2013.

[23] R. Pálovics and A. A. Benczúr. Temporal influence over the last.fm social network. In *Proc. ASONAM*, pages 486–493. ACM, 2013.

[24] R. Pálovics, A. A. Benczúr, L. Kocsis, T. Kiss, and E. Frigó. Exploiting temporal influence in online recommendation. In *Proc. RecSys*, pages 273–280. ACM, 2014.

[25] A. Paterek Improving regularized singular value decomposition for collaborative filtering In *Proc. SIGKDD*, pages 39–42. ACM, 2007.

[26] I. Pilászy, A. Serény, G. Dózsa, B. Hidasi, A. Sári, and J. Gub. Neighbor methods vs matrix factorization - case studies of real-life recommendations. In *LSRS2015 at RECSYS*, 2015.

[27] I. Pilászy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proc. RecSys*, pages 71–78. ACM, 2010.

[28] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proc. WWW*, pages 811–820. ACM, 2010.

[29] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. WWW*, pages 285–295, 2001.

[30] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proc. 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, pages 1–8. ACM, 2008.

[31] R. Turrin, M. Quadrana, A. Condorelli, R. Pagano, and P. Cremonesi. 30music listening and playlists dataset. In *RecSys Posters*, 2015.

[32] Alexei Vazquez, Balazs Racz, Andras Lukacs, and Albert-Laszlo Barabasi. Impact of non-poissonian activity patterns on spreading processes. *Physical review letters*, 98(15):158702, 2007.

[33] J. Vinagre, A. M. Jorge, and J. Gama. Evaluation of recommender systems in streaming environments. In *Workshop on Recommender Systems Evaluation: Dimensions and Design (REDD), in conjunction with RecSys*, 2014.

[34] E. M. Voorhees et al. The TREC-8 question answering track report. In *TREC*, volume 99, pages 77–82, 1999.

[35] X. Yang, H. Steck, Y. Guo, and Y. Liu. On top-k recommendation using social networks. In *Proc. RecSys*, pages 67–74. ACM, 2012.

[36] Q. Yuan, L. Chen, and S. Zhao. Factorization vs. regularization: fusing heterogeneous social relationships in top-n recommendation. In *Proc. RecSys*, pages 245–252. ACM, 2011.