SCALABLE EXPLORE-EXPLOIT COLLABORATIVE FILTERING

Frédéric Guillou, Inria, Univ. Lille, CNRS, France, frederic.guillou@inria.fr

Romaric Gaudel, Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France, romaric.gaudel@inria.fr

Philippe Preux, Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France, philippe.preux@inria.fr

Abstract

Recommender Systems (RS) aim at suggesting to users one or several items in which they might have interest. These systems have to update themselves as users provide new ratings, but also as new users/items enter the system. While this adaptation makes recommendation an intrinsically sequential task, most researches about RS based on Collaborative Filtering are omitting this fact, as well as the ensuing exploration/exploitation dilemma: should the system recommend items which bring more information about the users (explore), or should it try to get an immediate feedback as high as possible (exploit)? Recently, a few approaches were proposed to solve that dilemma, but they do not meet requirements to scale up to real life applications which is a crucial point as the number of items available on RS and the number of users in these systems explode. In this paper, we present an explore-exploit Collaborative Filtering RS which is both efficient and scales well. Extensive experiments on some of the largest available real-world datasets show that the proposed approach performs accurate personalized recommendations in less than a millisecond per recommendation, which makes it a good candidate for true applications.

Keywords: Recommender Systems, Sequential Recommendation, Matrix Factorization, Multi-Armed Bandit.

1 INTRODUCTION

We consider Collaborative Filtering approaches based on Matrix Completion. Such Recommendation Systems (RS) recommend items to users and adapt the recommendation to user's tastes as inferred from past user behaviour. Depending on the application, items can be ads, news, music, videos, movies, etc. This recommendation setting was popularized by the Netflix challenge (Bennett & Lanning 2007). Most of researches model the taste between users and items as a matrix R^* (Koren et al. 2009).

In that matrix, only a few entries are known: the ones corresponding to feedback gathered in the past. In such a context, the RS recovers unknown values in R^* and the evaluation is done by splitting log data into two parts: the first part (aka. train-set) is used to define the training matrix which is completed by the RS algorithm; the second part (aka. test-set) is used to measure the quality of the matrix returned by the RS. Common measures of that quality are Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) on the test-set.

While such a static batch evaluation makes sense to measure the quality of the matrix-completion step of Collaborative Filtering, it does not evaluate the quality of the final recommendation. Namely, Collaborative Filtering based RS works in reality in a sequential manner and loops through the following steps:

1. Build a model of the users' tastes based on past feedback;
2. Recommend items to users using this model;
3. Gather feedback from users about recommended products. Note that this feedback will be used to build the model at step 1 during next iterations.

Notice that the model built at step 1 heavily depends on the feedback gathered at previous iterations. This feedback only exists for items which were chosen by the model itself. As such, at step 2, one faces the *exploration/exploitation dilemma*: either (i) recommend an item which hopefully leads to a good feedback (aka exploit) or (ii) recommend an item which hopefully brings information on the user's taste (aka explore). This dilemma is the core point of Multi-armed Bandit Theory (Auer et al. 2002; Bubeck & Cesa-Bianchi 2012).

The exploration/exploitation dilemma is already studied in a sub-field of RS which has access to a representation of the context (the user, the item, the webpage …) (Tang et al. 2014). Typical applications are the selection of news or ads to show on a web-page. The corresponding RS builds upon contextual bandits which are supported by strong theoretical results.

In contrast with these studies, we focus on the setting where these representations are unknown and have to be inferred solely from users' feedback. In particular, we want to emphasize that we do not use any side information, neither about users, nor items. That field of research (handling the whole loop without side information) is almost empty and the few attempts therein leave out computational complexity constraints (Kawale et al. 2015).

Our paper introduces the first sequential Collaborative Filtering based RS which (i) makes a good trade-off between exploration and exploitation and (ii) is able to properly scale. Extensive experiments are conducted on real word datasets of millions of ratings and convey three main conclusions:

- First, they highlight the need for a trade-off between exploration and exploitation for a RS to be optimal;
- Second, they demonstrate that the proposed approach brings such optimal trade-off;
- Third, they exhibit that the proposed approach can perform good recommendations in less than a millisecond per recommendation.

After introducing the setting in the next section, Section 3 gives an overview of the state of the art in RS and Section 4 recalls the necessary background in bandit theory. In Section 5, we introduce an algorithm which fully takes into account the sequential aspect of RS. Section 6 provides an experimental study on real datasets. Finally, we draw some future lines of work in Section 7.

2 SEQUENTIAL RECOMMENDATION

Let us focus on a particular recommendation scenario, which illustrates more accurately how typical recommendation systems work. We consider N users, M items, and an unknown matrix R^* of size $N \times M$ such that $r^*_{i,j}$ is the taste of user i with regard to item j. At each time-step t with $1 \leq t \leq T$

1. A user i_t requests a recommendation to the RS,
2. the RS selects an item j_t among the set of available items,
3. user i_t returns a feedback $r_t = r^*_{i_t,j_t} + \varepsilon_t$ for item j_t, where ε_t is a Gaussian noise.

We refer to applications where the feedback r_t corresponds to the quantity that has to be optimized, aka. *the reward*. In such a context, the aim of the RS is to maximize the reward accumulated along time-steps $CumRew_T = \sum_{t=1}^{T} r_t$ or equivalently to minimize the *cumulative regret* defined in Equation (1).

$$\mathcal{R}_T = \sum_{t=1}^{T} \left(\max_j r^*_{i_t,j} - r^*_{i_t,j_t} \right) \tag{1}$$

The cumulative regret measures how much the system loses by recommending a sub-optimal item.

Along the paper, we use following notations. We denote R_t the partially known $N \times M$ matrix such that $r_{i_s,j_s} = r_s$ for any $s \leq t$. We note $\mathcal{S}_t$ the set of known entries of R_t and $I_t(j)$ (respectively $J_t(i)$) the set of users i (resp. items j) for which $(i,j) \in \mathcal{S}_t$. For the sake of readability, the subscript t is omitted in the rest of this paper. Finally, for any matrix A, we denote A_i the vector corresponding to the i-th row of A.

3 STATE OF THE ART

This section presents state of the art approaches for recommendation. We start with standard RS approaches which focus on recovering R^* from previously observed feedbacks, and therefore do not take into account the sequential aspect of recommendation. Thereafter, we list approaches which more or less consider this sequential aspect.

3.1 Matrix Factorization

Since the Netflix challenge (Bennett & Lanning 2007), many works on RS focus on Matrix Factorization (Koren et al. 2009): the unknown matrix R^* is assumed to be of low rank. Namely, there exist U and V such that $R^* = UV^T$, where U is a matrix of size $N \times k$ representing users features, V is a matrix of size $M \times k$ representing items features, k is the rank of R^*, and $k \ll \min(N, M)$. Thereafter, the estimator of R^* is defined as $\hat{R}^* = \hat{U}\hat{V}^T$ where $(\hat{U}, \hat{V})$ is the solution of:

$$\text{argmin}_{U,V} \sum_{\forall (i,j) \in \mathcal{S}} (r_{i,j} - U_i V_j^T)^2 + \lambda \cdot \Omega(U,V), \tag{2}$$

in which $\lambda \in \mathbb{R}^+$, and the usual regularization term is $\Omega(U,V) = \|U\|^2 + \|V\|^2$.

Equation (2) corresponds to a non-convex optimization problem. The minimization is usually performed either by stochastic gradient descent (SGD), or by alternating least squares (ALS).

ALS-WR (Zhou et al. 2008) regularizes users and items according to their respective importance in the matrix of ratings:

$$\Omega(U,V) = \sum_i \#J(i)\,\|U_i\|^2 + \sum_j \#I(j)\,\left\|V_j\right\|^2 \quad (3)$$

This regularization is known to have a good empirical behaviour (limited overfitting, easy tuning of λ and k, low RMSE). This is also the default one in Mahout[1]. To solve Equation (2), ALS-WR uses the alternating least square algorithm described in Figure 1.

Input: $R, \mathcal{S}, \lambda$

Input/Output: $\widehat{U}, \widehat{V}$

while $(\widehat{U}, \widehat{V})$ did not converge **do**

  for $i \in 1, \ldots, N$ **do**

    $\widehat{U}_i \leftarrow \text{argmin}_u \sum_{j \in J_t(i)} (r_{i,j} - u\widehat{V}_j^T)^2 + \lambda \cdot \#J_t(i)\|u\|$

  end for

  for $j \in 1, \ldots, M$ **do**

    $\widehat{V}_j \leftarrow \text{argmin}_v \sum_{i \in I_t(j)} (r_{i,j} - \widehat{U}_i v^T)^2 + \lambda \cdot \#I_t(j)\|v\|$

  end for

end for

Figure 1. ALS-WR: finds a solution to Equation (2) with an alternating least square approach.

Other possible approaches for Matrix Factorization are probabilistic ones, such as Probabilistic Matrix Factorization (PMF) (Salakhutdinov & Mnih 2011) and its Bayesian version Bayesian Probabilistic Matrix Factorization (BPMF) (Salakhutdinov & Mnih 2008). In this case, Gaussian priors are put on user and item feature vectors: for any user i and any item j, $U_i \sim \mathcal{N}(0, \sigma_U^2\, I)$, $V_j \sim \mathcal{N}(0, \sigma_V^2\, I)$ and rating are generated as $r_{i,j} \big| U, V \sim \mathcal{N}(U_i^T V_j, \sigma^2)$.

All these approaches focus on the ability to do **one** relevant recommendation, and thus miss the effect of the feedback arising from that recommendation on future ones. Our approach handles that effect.

3.2 Learning to Rank

Standard RS approaches evaluate themselves using the RMSE measure in a static batch manner. While it is a reasonable evaluation to measure how well the matrix has been completed, there is no promise that the RS will perform high-quality recommendation judging only by the RMSE score. In other words, having a good matrix-completion algorithm does not imply having a good RS: for example, if the RS recommends five items at each recommendation step, this RS only needs to identify these five best items. It would be useless to estimate well the ratings of the ten worst items, or their respective order. Following this direction, Cremonesi et al. (2010) compare state of the art RS algorithms with respect to a rank-based scoring and shows that winning algorithms are not the ones which reach the smallest RMSE score.

Following the same guideline, Rendle et al. (2009), Weston et al. (2013), and Shi et al. (2012) propose RS which directly target a good ranking of the top items instead of a full-completion of the matrix. During the training phase, they replace the L2 loss of Equation (2) with rank-based losses (AUC, MRR, NDCG…).

[1] https://mahout.apache.org/

While targeting a rank-based measure during the training phase could increase the accuracy of the RS, Garcin et al. (2014) and Said & Bellogín (2014) show that the cumulative reward/regret is the unique good metric to evaluate the RS algorithm.

3.3 Exploration/Exploitation Dilemma from Ratings Only

To the best of our knowledge, only few papers consider a RS setting similar to the one presented in the current paper: (Li et al. 2011; Zhao et al. 2013; Xing et al. 2014; Nakamura et al. 2014; Kawale et al. 2015; Mary et al. 2015). Mary et al. (2015) focus on a different recommendation setting where an item can be recommended only one time per user. Their approach builds upon ALS Matrix Factorization framework and extends linear bandits (Li et al. 2010) to handle the exploration/exploitation dilemma. The use of linear bandit framework prevents this approach from scaling.

Other papers propose a solution based on a Bayesian model. The approach PTS introduced in (Kawale et al. 2015) is based on Thompson Sampling and Particle Filtering to tackle the exact same problem as our paper. However, their approach requires a huge computing time which scales with k^3. As a consequence, they provide experiments only on small datasets, with $k = 2$.

Our approach is the first one which both tackles the exploration/exploitation dilemma and scales up well to large datasets and high number of recommendations, while building an accurate representation of users and items ($k \gg 2$). A preliminary work on this approach was presented at a workshop (Guillou et al. 2015).

3.4 Exploration/Exploitation Dilemma Using Contextual Information

Contrary to the non-contextual case, the exploration/exploitation trade-off is already well-studied in the field of RS which has access to a representation of the context (Tang et al. 2014; Vanchinathan et al. 2014). Compared to them, we focus on the setting where these representations are unknown and have to be inferred from users' feedback.

3.5 Cold-Start

The cold-start setting also focuses on the need for exploration (Agarwal et al. 2009; Bhagat et al. 2014): the goal in this case is to deal with new users or new items. While some approaches look at external information on the user (Agarwal et al. 2009), some papers rewrite the problem as an Active Learning problem (Bhagat et al. 2014): *perform recommendation in order to gather information on the user as fast as possible*. Targeted applications would first "explore" the user and then "exploit" him.

Unlike Active Learning strategies, (i) we spread the cost of exploration along time, and (ii) we handle the need for a never-ending exploration to reach optimality (Bubeck & Cesa-Bianchi 2012). Finally, while Rendle and Schmidt-Thieme (2008) and Luo et al. (2012) focus on online Matrix Factorization, they omit the exploration/exploitation dilemma, and they focus on stochastic gradient descent to solve Equation (2).

4 EXPLORATION/EXPLOITATION DILEMMA FOR MULTI-ARMED BANDITS

The RS we focus on works in a sequential context. As a consequence, while the recommendation made at time-step t aims at collecting a good reward at the present time, it affects the information that is collected, and therefore also the future recommendations and rewards. Specifically, in the context of sequential decision under uncertainty problems, an algorithm which focuses only on short term reward loses with regard to expected long term reward. This section recalls standard strategies to handle this short term vs. long term dilemma. This recall is done in a setting way simpler than the one faced in our paper.

We consider the Multi-Armed Bandits (MAB) setting (Auer et al. 2002): we face a bandit machine with M independent arms. At each time-step, we pull an arm j and receive a reward drawn from [0,1] which follows a probability distribution ν_j. Let μ_j denote the mean of ν_j, $j^* = \text{argmax}_j\, \mu_j$ be the best arm and $\mu^* = \max_j \mu_j = \mu_{j^*}$ be the best expected reward. The parameters $\{\nu_j\}, \{\mu_j\}, j^*, \mu^*$ are unknown.

We play T consecutive times and aim at minimizing the *regret* $\mathcal{R}_T = \sum_{t=1}^{T}(\mu^* - \mu_{j_t})$ where j_t denotes the arm pulled at time-step t. As the parameters are unknown, at each time-step (except the last one), we face the dilemma: either (i) focus on short-term reward (aka. *exploit*) by pulling the arm which was the best at previous time-steps, or (ii) focus on long-term reward (aka. *explore*) by pulling an arm to improve the estimation of its parameters. Neither of these strategies is optimal. To be optimal, a strategy has to balance exploration and exploitation.

Auer et al. (2002) propose a strategy based on an upper confidence bound (**UCB1**) to handle this exploration vs. exploitation dilemma. UCB1 balances exploration and exploitation by playing the arm:

$$j_t = \underset{j}{\text{argmax}}\, \hat{\mu}_j(t) + \sqrt{\frac{2 \ln t}{T_j(t)}} \tag{4}$$

where $T_j(t)$ corresponds to the number of pulls of arm j since the first time-step and $\hat{\mu}_j(t)$ denotes the empirical mean reward incurred from arm j up to time t. This equation embodies the exploration/exploitation trade-off: while $\hat{\mu}_j(t)$ promotes exploitation of the arm which looks optimal, the second term of the sum promotes exploration of less played arms. Other flavours of UCB-like algorithms (Audibert et al. 2009; Garivier & Cappé 2011; Li et al. 2010) aim at a strategy closer to the optimal one, or at a strategy which benefits from constraints on the reward distribution.

ε_n-greedy is an even simpler but efficient approach to balance exploration and exploitation (Auer et al. 2002). It consists in playing the greedy strategy ($j_t = \text{argmax}_j\, \hat{\mu}_j(t)$) with probability $1 - \varepsilon_t$ and in pulling an arm at random otherwise. Parameter ε_t is set to α/t, with $\alpha \in \mathbb{R}^+$ a constant to tune, so that there is more exploration at the beginning of the evaluation and then a decreasing chance to fall on an exploration step.

Thompson Sampling is another well-known approach to solve the dilemma in a Bayesian way (Chapelle & Li 2011). With θ denoting the set of parameters of the distribution of the reward, and D being the past observations currently available, the system chooses the arm j with probability:

$$\int \mathbb{I}\left[\mathbb{E}[r|j,\theta] = \max_{j'} \mathbb{E}[r|j',\theta]\right] P(\theta|D) d\theta, \tag{5}$$

Where $\mathbb{I}[a = b]$ stands for 1 when $a = b$ and 0 otherwise.

This is usually implemented simply by sampling θ from the posterior $P(\theta|D)$, and choosing the arm which maximizes the expected reward given the parameters and the action taken, $j = \text{argmax}_{j'} \mathbb{E}[r|j',\theta]$.

5 EXPLORE-EXPLOIT RECOMMENDER SYSTEM

This section introduces a RS which handles the sequential aspect of recommendation. More precisely, the proposed approach works in the context presented in Section 2 and aims at minimizing the cumulative regret $\mathcal{R}_T$. As needed, the proposed approach balances exploration and exploitation.

Named SeALS (for *Sequential ALS-WR*), our approach is described in Figure 2. It builds upon ALS-WR Matrix Completion approach and ε_n-greedy strategy to tackle the exploration/exploitation dilemma. At time-step t, for a given user i_t, ALS-WR associates an expected reward $\hat{r}_{i_t,j}$ to each item j. Then the item to recommend j_t is chosen by an ε_n-greedy strategy.

Input: $T_u, p, \lambda, \alpha, p$

Input/Output: $R, \mathcal{S}$

$(\widehat{U}, \widehat{V}) \leftarrow$ ALS-WR$(\widehat{U}, \widehat{V}, \mathrm{R}, \mathcal{S}, \lambda)$

for $t = 1, 2, \ldots$ **do**

get user i_t and set J_t of allowed items

$$j_t \leftarrow \begin{cases} \mathrm{argmax}_{j \in J_t} \widehat{U}_{i_t} \widehat{V}_j^{\,T}, with\ probability\ 1 - \min(\alpha/t, 1) \\ random(j \in J_t), with\ probability\ \min(\alpha/t, 1) \end{cases}$$

recommend item j_t and receive rating $r_t = r_{i_t, j_t}$

update R and $\mathcal{S}$

if $t \equiv 0\ mod\ T_u$ **then** $(\widehat{U}, \widehat{V}) \leftarrow$ mBALS-WR$(\widehat{U}, \widehat{V}, \mathrm{R}, \mathcal{S}, \lambda, p)$ **end if**

end for

Figure 2. SeALS algorithm: recommend in a sequential context

Obviously, ALS-WR requires too large computation time to be run at each time-step to recalculate user and items features. A solution consists in running ALS-WR from times to times, say every T_u time-steps. While such strategy works well when T_u is small enough, $\mathcal{R}_T$ drastically increases otherwise (see Section 6.3). Taking inspiration from stochastic gradient approaches (Bousquet & Bottou 2008), we solve that problem by designing a mini-batch version of ALS-WR, denoted mBALS-WR. Figure 3 describes that new algorithm.

Input: $R, \mathcal{S}, \lambda, p$

Input/Output: $\widehat{U}, \widehat{V}$

Randomly sample p% of all users in a list l_{users}

Randomly sample p% of all items in a list l_{items}

for $i \in l_{users}$ **do**

$\widehat{U}_i \leftarrow \mathrm{argmin}_u \sum_{j \in J_t(i)} (r_{i,j} - u\widehat{V}_j^T)^2 + \lambda \cdot \#J_t(i) \|u\|$

end for

for $j \in l_{items}$ **do**

$\widehat{V}_j \leftarrow \mathrm{argmin}_v \sum_{i \in I_t(j)} (r_{i,j} - \widehat{U}_i v^T)^2 + \lambda \cdot \#I_t(j) \|v\|$

end for

Figure 3. mBALS-WR: mini-batch version of ALS-WR.

mBALS-WR is designed to work in a sequential context where the matrix decomposition slightly changes between two consecutive calls. As a consequence, there are three main differences from ALS-WR. First, instead of computing $\widehat{U}$ and $\widehat{V}$ from scratch, mBALS-WR updates both matrices. Second, mBALS-WR performs only one pass on the data. And third, mBALS-WR updates only a fixed percentage of the lines of $\widehat{U}$ and $\widehat{V}$. When the parameter p = 100, mBALS-WR is a one-pass ALS-WR.

The main advantage of mBALS-WR is in spreading the computing budget along time-steps which means $\widehat{U}$ and $\widehat{V}$ are more often up to date. On one hand, ALS-WR consumes a huge computing budget every thousands of time-steps; in between two such updates, it selects the items to recommend based

on an outdated decomposition. On the other hand, mBALS-WR makes frequent updates of the decomposition. In the extreme case, updates can be done at each time-step.

6 EXPERIMENTAL INVESTIGATION

In this section, we empirically evaluate the algorithms in the sequential setting on large real-world datasets. Two series of experiments emphasize two aspects of the sequential RS: Section 6.1 shows that exploration helps to improve the quality of the RS model and we compare our model with several baselines, while Section 6.2 focuses on the influence of the method updating the matrix model on the cumulative regret and the running time.

6.1 Experimental Settings and Remarks

We use the exact same setting as used in the paper by Kawale et al. (2015). For each dataset, we start with an empty matrix R to simulate an extreme cold-start scenario where no information is available at all. Then, for a given number of time-steps, we loop on the following procedure:

1. We select a user i_t uniformly at random,
2. The algorithm chooses and item j_t to recommend,
3. We reveal the value of r_{i_t,j_t} and increment the cumulative regret score.

As we have no access to the ground-truth matrix R^*, the regret is computed with respect to the best value in matrix R. To make it specific, the measured cumulative regret is

$$\mathcal{R}_T = \sum_{t=1}^{T} \left(\max_j r_{i_t,j} - r_{i_t,j_t} \right). \tag{6}$$

Note that the replay is allowed: once an item has been recommended, it is not discarded from the set of future possible recommendations. In other words, an item can be recommended several times to the same user.

We consider five real-world datasets for our experiments: Movielens1M and Movielens20M (Harper & Konstan 2015), LibimSeTi (Brozovsky & Petricek 2007), Douban (Ma et al. 2011) and Yahoo! Music user ratings of musical artists[2]. Characteristics of these datasets are reported in Table 1.

	Movielens1M	LibimSeTi	Douban	Movielens20M	Yahoo
Number of users	6040	135,359	129,490	138,493	1,065,258
Number of items	3706	168,791	58,541	26,744	98,209
Number of ratings	1,000,209	17,359,346	16,830,839	20,000,263	109,485,914

Table 1. Dataset characteristics.

Except Movielens1M, all these datasets are among the largest ones available to evaluate RS, and items to recommend are as various as movies (Douban, Movielens), artists (Yahoo), or even people to date (LibimSeTi). The Yahoo dataset used in our experiments is taken from an originally larger dataset from which we extracted only users who have rated at least 20 items.

Some difficulties arise when using real datasets: in most cases, the ground truth is unknown, and only a very small fraction of ratings is known since users gave ratings only to items they have purchased/listened/watched. This makes the evaluation of algorithms uneasy considering we need in

[2] https://webscope.sandbox.yahoo.com/

advance the reward of items we include in the list of possible recommendations. This is the case in our experiments, as we do not have access to the full matrix R in all datasets.

This issue is solved in the case of contextual bandits by using reject sampling (Li et al. 2011): the algorithm chooses an arm (item), and if the arm does not appear in logged data, the choice is discarded, as if the algorithm had not been called at all. For a well collected dataset, this estimator has no bias and has a known bound on the decrease of the error rate (Langford et al. 2008). With our setting, we need no more to rely on reject sampling: we restrict the possible choices for a user at time-step t to the items with a known rating in the dataset.

The SeALS algorithm is compared to the following baselines:

- **Random**: at each iteration, a random item is recommended to the user.
- **Popular**: this approach assumes we know the most popular items in advance based on the ground truth matrix. At each iteration, the most popular item (restricted to the items rated by the user on the dataset) is recommended. This is a strong baseline as it knows beforehand which items have the highest average ratings.
- **UCB1**: this bandit approach considers each reward r_{i_t,j_t} as an independent realization of a distribution ν_{j_t}. In other words, it recommends an item without taking into account the identity of the user requesting the recommendation.
- **PTS**: this approach (Kawale et al. 2015) builds upon a statistical model of the ratings matrix. The recommendations are done after a Thompson Sampling strategy which is implemented with a Particle Filter. We present the results obtained with the non-Bayesian version, as both Bayesian and non-Bayesian versions obtain very similar results.

6.2 Impact of Exploration

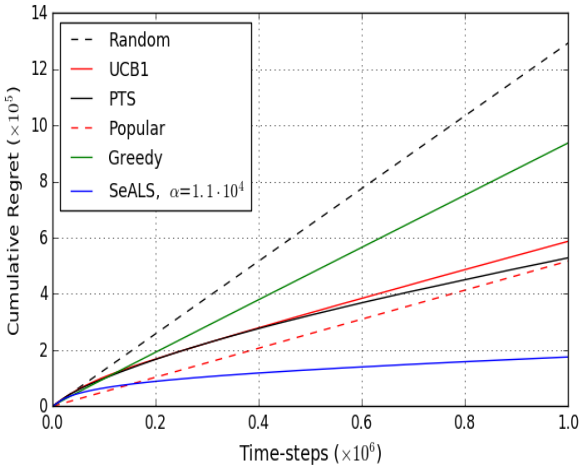
The first set of experiments compares two strategies to recommend an item: SeALS with $\alpha > 0$ and SeALS with $\alpha = 0$ (denoted Greedy). Note that Greedy corresponds to the greedy strategy with no exploration at all. Both strategies use the maximum possible value for p ($p = 100$), which means we update all the users and items every T_u-th time-steps. The value of T_u used is the same for SeALS and Greedy, and is set to 2000 for Movielens1M, 10000 for LibimSeTi, Douban and Movielens20M and 250000 for Yahoo. We set $\lambda = 0.1$ for Greedy and $\lambda = 0.15$ for SeALS, with a parameter k set to 15.

We also compare these two approaches with PTS. We tried to fix the value of the parameters of PTS as in (Kawale et al. 2015): 30 particles and $k = 2$. The experimental results we obtained were less good than the ones presented in their paper. However, increasing k gives closer results to theirs, so we display results of the PTS approach with $k = 10$.

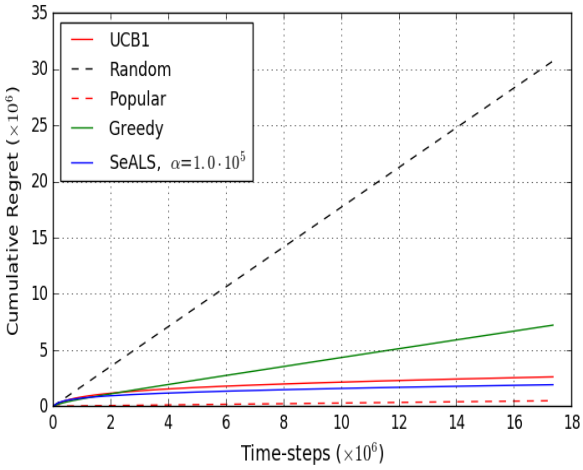
Figure 4 displays the cumulative regret obtained by the Recommender System after a given number of iterations (all results are averaged over 50 runs). Results on all datasets demonstrate the need of exploration during the recommendation process: by properly fixing α, SeALS gets lower cumulative regret value than Greedy. SeALS also obtains the best results on all datasets except LibimSeTi.

Interestingly, on the LibimSeTi dataset, the Popular method obtains a very low cumulative regret all along the iterations, despite not personalizing its recommendations. This means that finding the people who are the most popular among all users is sufficient to provide good dating recommendations to any user. Fortunately, with other datasets, Popular is not the optimal strategy.

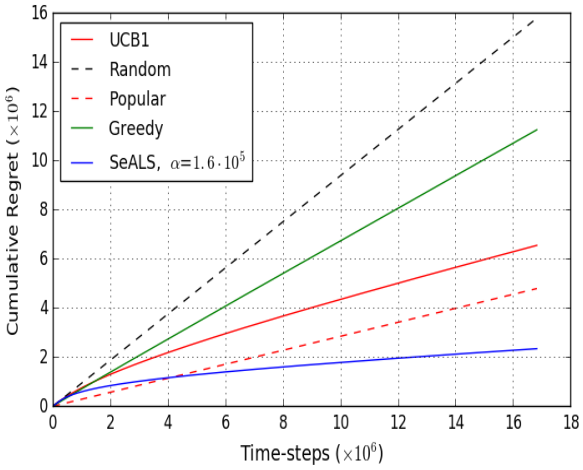
The PTS method which also tackles the exploration/exploitation dilemma appears only on the Movielens1M evaluation as this method does not scale well on large datasets (see Section 6.3 for the running time of PTS on Movielens1M). However, it is important to note that on the original PTS paper, this approach performs only comparably or slightly better than the Popular baseline on the evaluation provided on all small datasets, while our approach performs significantly better than this baseline on all datasets except LibimSeTi. One can reasonably assume that SeALS would perform better than PTS even if the latter one didn't have a scaling issue.



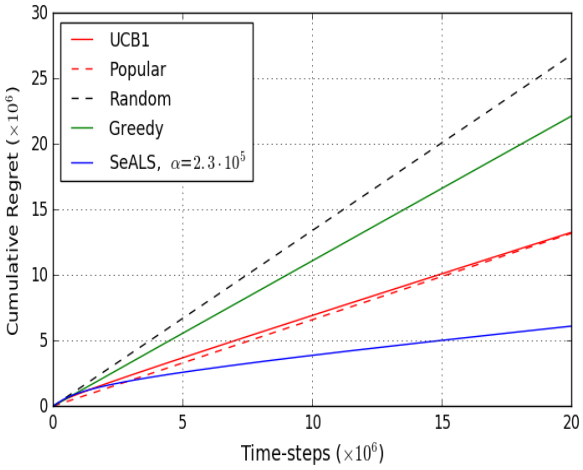
(a) Movielens1M



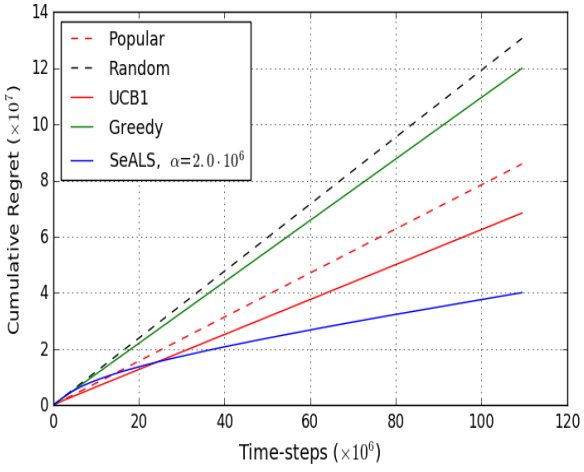
(b) LibimSeTi



(c) Douban



(d) Movielens20M



(e) Yahoo

Figure 4. Impact of exploration on five datasets

6.3 Impact of the Update Strategy

To evaluate the impact of both the method used to update the model, and the period at which updates take place, we set up a different evaluation display for this second experiment. For each RS, we run experiments as presented in Section 6.1, and we store the final running time as well as the cumulative regret at the end of all iterations. Such evaluation methodology allows finding which size of mini-batch (respectively update period) leads to the best trade-off between the final cumulative regret score and the running time. This is an important point as a recommendation should both (i) be accurate and (ii) be quickly provided to the user in a real-word RS.

Figure 5 displays the results on all datasets for this experiment. Each curve corresponds to a fixed size of mini-batch p, and every point of a same curve represents a specific value of the update period T_u. A point located at a high running time results from a small value of T_u (meaning the model was updated very often).

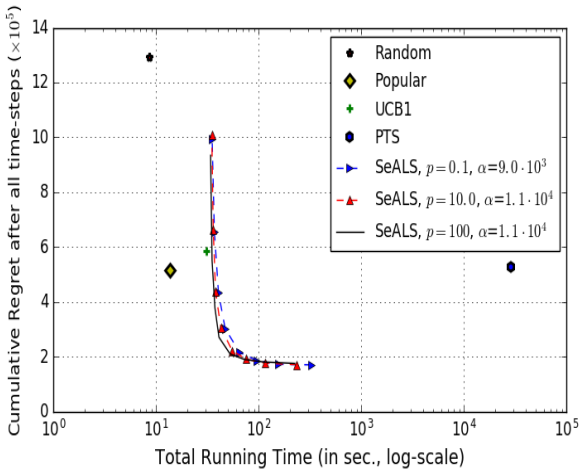
For SeALS with $p = 100$, the period T_u of the updates varies in the range [2000;200,000] for Movielens1M, [10000; 5,000,000] for LibimSeTi, Douban and Movielens20M datasets, and [250000; 10,000,000] for Yahoo dataset. For $p = 10$ and $p = 0.1$, the considered periods are the same ones as for $p = 100$, but respectively divided by 10 and 1000 in order to obtain similar running times. Indeed, since we update a smaller portion of the matrix, it is possible to run this update more often and choose a small period T_u. For each value of p, we display the curve with the value of α (for the exploration) which reaches the lowest cumulative regret.

Three main conclusions have to been drawn from this experiment:

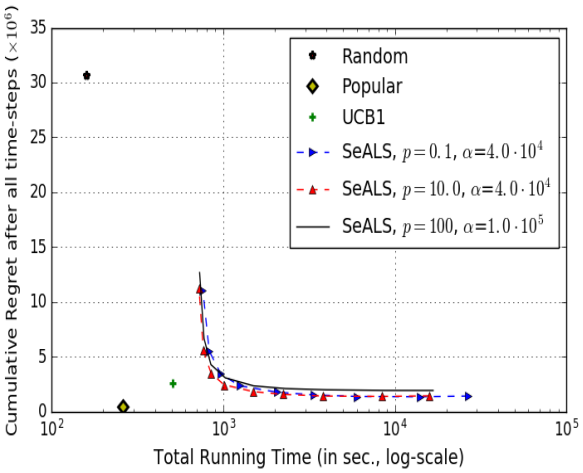
First, the results on Movielens1M highlight the non-scalability of the PTS algorithm, which takes several hours to complete 1million iterations while SeALS only takes few minutes. PTS does not seem to be an appropriate algorithm to provide quick recommendations as it takes too long updating the model.

Second, on each dataset, each curve concerning SeALS quickly decreases: there is a rapid transition from a poor score to a good cumulative regret. This means finding the appropriate period of update is sufficient to obtain a good RS.

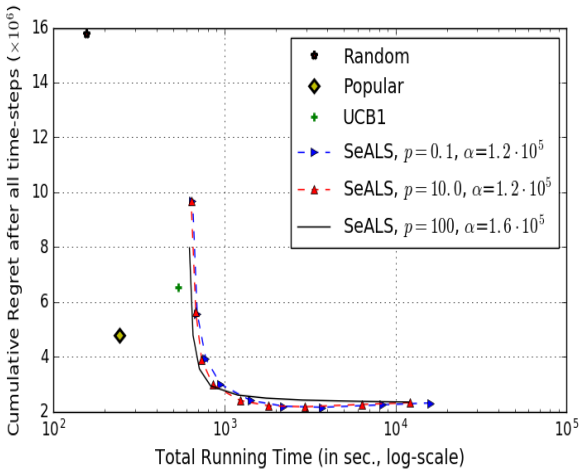
Third, on the large datasets, decreasing the portion of the users and items to update with a smaller p results in a worse score when using large update periods, but leads to a slightly better trade-off between the cumulative regret and the running time at some point, when the updates are happening more often. One has to notice the best results for smaller values of p are obtained with a smaller value of α, which implies less steps of exploration have been done. We guess some sort of exploration is added in the system by not updating each user and each item at each time-step as it happens when $p = 100$ is chosen: while the whole model is not updated, it does not play optimally, which means it explores.



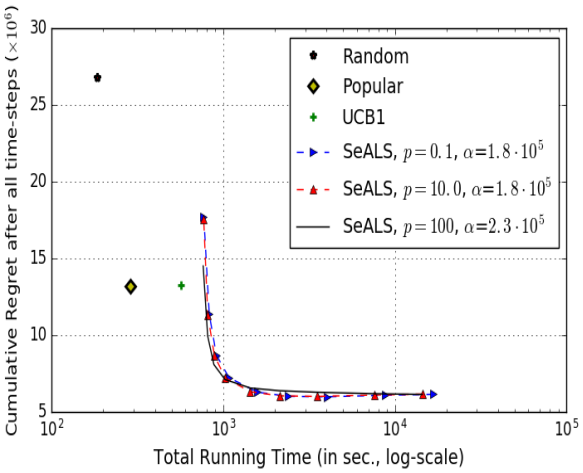
(a) Movielens1M



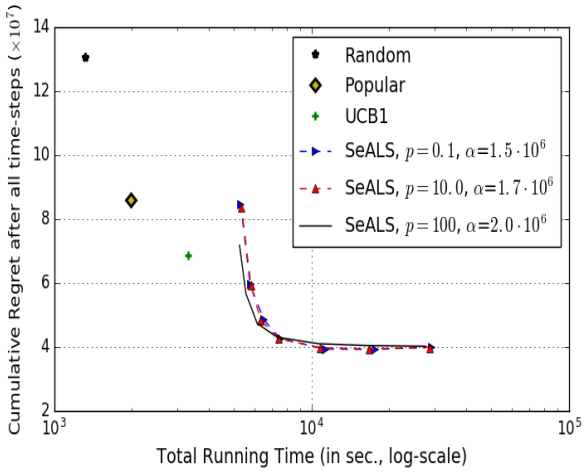
(b) LibimSeTi



(c) Douban



(d) Movielens20M



(e) Yahoo

Figure 5. Impact of the update strategy on five datasets

7 CONCLUSION AND FUTURE WORK

In this paper we handle Recommender Systems based on Matrix Completion in the suitable context: a never-ending alternation between (i) learning of the model and (ii) recommendations given the model.

Our proposed approach, SeALS, meets both challenges which arise in such a context. First, SeALS handles both short-term and long-term reward by balancing exploration and exploitation. Second, SeALS handles constraints on computing budget by adapting mini-batch strategy to alternating least square optimization. Experiments on real-life datasets show that (i) exploration is a necessary evil to acquire information and eventually improve the performance of the RS, and (ii) SeALS runs in a convenient time (less than a millisecond per iteration).

SeALS paves the way to many extensions. SeALS builds upon ALS-WR which is intrinsically parallel; implementations of SeALS in real-life systems should benefit from parallel computing. SeALS could also be extended to mix user feedback and contextual information. All in all, we hope SeALS is revealing a new playground for other bandit algorithms than ε_n-greedy.

Acknowledgements

The authors would like to acknowledge the stimulating environment provided by SequeL research group, Inria and CRIStAL. This work was supported by French Ministry of Higher Education and Research, by CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020, and by FUI Hermès. Experiments were carried out using Grid'5000 testbed, supported by Inria, CNRS, RENATER and several universities as well as other organizations.

References

Agarwal, D., Chen, B. C., Elango, P., Motgi, N., Park, S. T., Ramakrishnan, R., ... and Zachariah, J. (2009). Online models for content optimization. In Advances in Neural Information Processing Systems, 17-24.

Audibert, J. Y., Munos, R., and Szepesvári, C. (2009). Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. Theoretical Computer Science, 410(19), 1876-1902.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2-3), 235-256.

Bennett, J., and Lanning, S. (2007). The netflix prize. In Proceedings of KDD cup and workshop.

Bhagat, S., Weinsberg, U., Ioannidis, S., and Taft, N. (2014, October). Recommending with an agenda: Active learning of private attributes using matrix factorization. In Proceedings of the 8th ACM conference on recommender systems, 65-72. ACM.

Bousquet, O., and Bottou, L. (2008). The tradeoffs of large scale learning. In Advances in neural information processing systems, 161-168.

Brozovsky, L., and Petricek, V. (2007). Recommender system for online dating service. arXiv preprint cs/0703042.

Bubeck, S., and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Foundations and Trends in Machine Learning, 5 (1), 1-122.

Chapelle, O., and Li, L. (2011). An empirical evaluation of Thompson Sampling. In Advances in neural information processing systems, 2249-2257.

Chatterjee, S. (2015). Matrix estimation by universal singular value thresholding. The Annals of Statistics, 43(1), 177-214.

Cremonesi, P., Koren, Y., and Turrin, R. (2010, September). Performance of recommender algorithms on top-n recommendation tasks. In Proceedings of the fourth ACM conference on Recommender systems, 39-46. ACM.

Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., and Huber, A. (2014). Offline and Online Evaluation of News Recommender Systems at Swissinfo.Ch. In Proceedings of the 8th ACM Conference on Recommender Systems, New York, NY, USA, 169-176. ACM.

Garivier, A., and Cappé, O. (2011). The KL-UCB algorithm for bounded stochastic bandits and beyond. In Proceedings of the 24th Annual Conference On Learning Theory (COLT'11), 359-376.
Guillou F., Gaudel R., and Preux P. (2015). Filtering as a multi-armed bandit. In NIPS'15 workshop: Machine Learning for eCommerce.
Harper, F. M., and Konstan, J. A. (2015). The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS), 5(4), 19.
Vanchinathan, H.P., Nikolic, I., De Bona, I., and Krause, A. (2014). Explore-exploit in top-N recommender systems via Gaussian processes. In Proceedings of the 8th ACM Conference on Recommender systems (RecSys '14). ACM, New York, NY, USA, 225-232. ACM.
Kawale, J., Bui, H. H., Kveton, B., Tran-Thanh, L., and Chawla, S. (2015). Efficient Thompson Sampling for Online Matrix-Factorization Recommendation. In Advances in Neural Information Processing Systems, 1297-1305.
Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, (8), 30-37. ACM.
Langford, J., Strehl, A., and Wortman, J. (2008, July). Exploration scavenging. In Proceedings of the 25th international conference on Machine learning, 528-535.
Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010, April). A contextual-bandit approach to personalized news article recommendation. In Proceedings of the 19th international conference on World wide web, 661-670.
Li, L., Chu, W., Langford, J., and Wang, X. (2011, February). Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In Proceedings of the fourth ACM international conference on Web search and data mining, 97-306.
Luo, X., Xia, Y., and Zhu, Q. (2012). Incremental collaborative filtering recommender based on regularized matrix factorization. Knowledge-Based Systems, 27, 271-280.
Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011, February). Recommender systems with social regularization. In Proceedings of the fourth ACM international conference on Web search and data mining, 287-296.
Mary, J., Gaudel, R., and Preux, P. (2015). Bandits and Recommender Systems. In Machine Learning, Optimization, and Big Data (pp. 325-336). Springer International Publishing.
Nakamura, A. (2014). A UCB-Like Strategy of Collaborative Filtering. In ACML.
Rendle, S., and Schmidt-Thieme, L. (2008, October). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In Proceedings of the 2008 ACM conference on Recommender systems, 251-258. ACM.
Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 452–461.
Said, A., and Bellogín, A. (2014). Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks. In Proceedings of the 8th ACM Conference on Recommender Systems, New York, NY, USA, 129–136. ACM.
Salakhutdinov, R., and Mnih, A. (2011). Probabilistic matrix factorization. In Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NIPS).
Salakhutdinov, R., and Mnih, A. (2008). Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In Proceedings of the 25th international conference on Machine learning, 880-887.
Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., and Hanjalic, A. (2012). Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In Proceedings of the sixth ACM conference on Recommender systems, 139–146. ACM.
Weston, J., Yee, H., and Weiss, R. J. (2013). Learning to rank recommendations with the k-order statistic loss. In Proceedings of the 7th ACM conference on Recommender systems, 245–248. ACM.
Tang, L., Jiang, Y., Li, L., and Li, T. (2014, October). Ensemble contextual bandits for personalized recommendation. In Proceedings of the 8th ACM Conference on Recommender Systems (pp. 73-

80). ACM. Xing, Z., Wang, X., and Wang, Y. (2014, October). Enhancing Collaborative Filtering Music Recommendation by Balancing Exploration and Exploitation. In ISMIR, 445-450.
Zhao, X., Zhang, W., and Wang, J. (2013, October). Interactive collaborative filtering. In Proceedings of the 22nd ACM international conference on Conference on information and knowledge management, 1411-1420.
Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. (2008). Large-scale parallel collaborative filtering for the netflix prize. In Algorithmic Aspects in Information and Management, 337-348. Springer Berlin Heidelberg.