What is Happening Right Now ... That Interests *Me*?

Online Topic Discovery and Recommendation in Twitter

Ernesto Diaz-Aviles[1], Lucas Drumond[2], Zeno Gantner[2],
Lars Schmidt-Thieme[2], and Wolfgang Nejdl[1]

[1]L3S Research Center / University of Hannover, Germany
{diaz, nejdl}@L3S.de
[2]Information Systems and Machine Learning Lab / University of Hildesheim, Germany
{ldrumond, gantner, schmidt-thieme}@ISMLL.de

ABSTRACT

Users engaged in the Social Web increasingly rely upon continuous streams of Twitter messages (*tweets*) for real-time access to information and fresh knowledge about current affairs. However, given the deluge of tweets, it is a challenge for individuals to find relevant and appropriately ranked information. We propose to address this knowledge management problem by going beyond the general perspective of information finding in Twitter, that asks: "What is happening right now?", towards an individual user perspective, and ask: "What is interesting to *me* right now?" In this paper, we consider collaborative filtering as an online ranking problem and present `RMFO`, a method that creates, in real-time, user-specific rankings for a set of tweets based on individual preferences that are inferred from the user's past system interactions. Experiments on the *476 million Twitter tweets* dataset show that our online approach largely outperforms recommendations based on Twitter's *global trend* and Weighted Regularized Matrix Factorization (WRMF), a highly competitive state-of-the-art Collaborative Filtering technique, demonstrating the efficacy of our approach.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]—*Information Filtering*
General Terms: Algorithms, Experimentation, Measurement
Keywords: Collaborative Filtering; Online Ranking; Twitter

1. INTRODUCTION

The *Social Web* has been successfully established and is poised for continued growth. Real-time microblogging services, such as Twitter (`twitter.com`), have experienced an explosion in global user adoption over the past years [12].

Despite the recent amount of research dedicated to Twitter, online collaborative filtering and online ranking in Twitter have not yet been extensively addressed.

Given a continuous stream of incoming tweets, we are interested in the task of filtering and recommending topics that meet users' personal information needs. In particular, we use hashtags as surrogates for topics, and learn, online, a personalized ranking model based on low-rank matrix factorization for collaborative prediction.

Collaborative filtering (CF) is a successful approach at the core of recommender systems. CF algorithms analyze past interactions between users and items to produce personalized recommendations that are tailored to users' preferences.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
Copyright 2012 ACM 978-1-4503-1156-4/12/10 ...$10.00.

In the presence of a continuous stream of incoming tweets arriving at a high rate, our objective is to process the incoming data in bounded space and time, and recommend a short list of interesting topics that meet users' individual taste. Furthermore, our online CF algorithm should quickly learn the best Top-N recommendations based on real-time user interactions and prevent repeatedly suggesting highly relevant, but old information. In the absence of high quality *explicit feedback* (e.g., ratings), we infer user preferences about items using *implicit feedback*. For example, in Twitter, if user *Alice* has been tagging her tweets lately with the hashtag *#Olympics2012*; and, so far, she has never used the hashtag *#fashion*, we exploit this information, and use it as a good indicator for her up-to-date preferences. We can infer that currently, *Alice* is more interested in *Olympic Games* than, for instance, in *fashion*. Thus the task can be cast as that of recommending hashtags to users.

The high rate makes it harder to: (i) capture the information transmitted; (ii) compute sophisticated models on large pieces of the input; and (iii) store the amount of input data, which we consider significantly larger than the memory available to the algorithm [8]. In this paper, we present our Online Matrix Factorization approach – `RMFO` – for addressing these research challenges.

To the best of our knowledge this work is the first empirical study demonstrating the viability of online collaborative filtering for Twitter. The main contributions of this work are:

1. We introduce a novel framework for online collaborative filtering based on a pairwise ranking approach for matrix factorization, in the presence of streaming data.
2. We propose `RMFO`, an online learning algorithm for collaborative filtering. We explore different variations of the algorithm and show that it achieves state-of-the-art performance when recommending a short list of interesting and relevant topics to users from a continuous high volume stream of tweets, under the constraints of bounded space and time.
3. Personalized and unpersonalized offline learning to rank have been previously studied in the literature. This paper proposes an innovative perspective to the problem, directed to social media streams and based on online learning and matrix factorization techniques.

2. OUR MODEL: RMFO

In this section, we formally define the problem, introduce our approach `RMFO` and describe three variations of the algorithm, namely, *Single Pass*, *User Buffer*, and *Reservoir Sampling*.

Background and Notation

First we introduce some notation that will be useful in our setting. Let $U = \{u_1, \ldots, u_n\}$ and $I = \{i_1, \ldots, i_m\}$ be the sets of all users and all items, respectively. We reserve special indexing letters to distinguish users from items: for users u, v, and for items i, j. Suppose we have interactions between these two entities, and for some user $u \in U$ and item $i \in I$, we observe a *relational score* x_{ui}. Thus, each instance of the data is a tuple (u, i, x_{ui}). Typical CF algorithms organize these tuples into a sparse matrix $\mathbf{X}$ of size $|U| \times |I|$, using (u, i) as index and x_{ui} as entry value. The task of the recommender system is to estimate the score for the missing entries. We assume a total order between the possible score values. We distinguish predicted scores from the known ones, by using $\hat{x}_{ui}$. The set of all observed scores is $S := \{(u, i, x_{ui}) \mid (u, i, x_{ui}) \in U \times I \times \mathbb{N}\}$. For convenience, we also define for each user the set of all items with an observed score: $B_u^+ := \{i \in I \mid (u, i, x_{ui}) \in S\}$.

Low dimensional linear factor models based on matrix factorization (MF) are popular collaborative filtering approaches [7]. These models consider that only a small number of *latent* factors can influence the preferences. Their prediction is a real number, $\hat{x}_{ui}$, per user item pair (u, i). In its basic form, matrix factorization estimates a matrix $\hat{\mathbf{X}} : U \times I$ by the product of two low-rank matrices $\mathbf{W} : |U| \times k$ and $\mathbf{H} : |I| \times k$ as follows: $\hat{\mathbf{X}} := \mathbf{W}\mathbf{H}^\intercal$, where k is a parameter corresponding to the rank of the approximation. Each row, $\mathbf{w}_u$ in W and $\mathbf{h}_i$ in H can be considered as a feature vector describing a user, u, and an item, i, correspondingly. Thus the final prediction is the linear combination of the factors: $\hat{x}_{ui} = \langle \mathbf{w}_u, \mathbf{h}_i \rangle = \sum_{f=1}^{k} w_{uf} \cdot h_{if}$.

Problem Definition

We focus on learning a matrix factorization model for collaborative filtering in presence of streaming data. To this end, we will follow a pairwise approach to minimize an ordinal loss. Our formalization extends the work of Sculley [11] for unpersonalized learning to rank, to an online collaborative filtering setting.

With slight abuse of notation, we also use S to represent the input stream $s_1, s_2, \ldots$ that arrives sequentially, instance by instance. Let $p_t = ((u, i), (u, j))_t$ denote a pair of training instances sampled at time t, where $(u, i) \in S$ has been observed in the stream and $(u, j) \notin S$ has not.

Formally, we define the set P as the set of tuples $p = ((u, i), (u, j))$ selected from the data stream S, as follows: $P := \{((u, i), (u, j)) \mid i \in B_u^+ \;\wedge\; j \notin B_u^+\}$.

We require pairs that create a *contrast* in the preferences for a given user u over items i and j. Since we are dealing with implicit, positive only feedback data (i.e. the user never explicitly states a negative preference for an item) we follow the rationale from Rendle et al. [9] and assume that user u prefers item i over item j. We will restrict the study to a binary set of preferences $x_{ui} = \{+1, -1\}$, e.g., *observed* and *not-observed*, represented numerically with $+1$ and -1, respectively. For example, if a user u in Twitter posts a message containing hashtag i, then we consider it as a positive feedback and assign a score $x_{ui} = +1$. More formally, $x_{ui} = +1 \iff i \in B_u^+$. In future work we plan to explore how repeated feedback can be exploited to establish a total order for items in B_u^+.

With P defined, we find $\theta = (\mathbf{W}, \mathbf{H})$ that minimizes the pairwise objective function:

$$\underset{\theta=(\mathbf{W},\mathbf{H})}{\operatorname{argmin}} L(P, \mathbf{W}, \mathbf{H}) + \frac{\lambda_W}{2}||\mathbf{W}||_2^2 + \frac{\lambda_H}{2}||\mathbf{H}||_2^2 \ . \quad (1)$$

In this paper, we explore the use of the SVM loss, or *hinge-loss*, used by RankSVM for the learning to rank task [6]. Given the predicted scores $\hat{x}_{ui}$ and $\hat{x}_{uj}$, the ranking task is reduced to a pairwise classification task by checking whether the model is able to correctly rank a pair $p \in P$ or not. Thus, $L(P, \mathbf{W}, \mathbf{H})$ is defined as follows:

$$L(P, \mathbf{W}, \mathbf{H}) = \frac{1}{|P|} \sum_{p \in P} \hbar(y_{uij} \cdot \langle \mathbf{w}_u, \mathbf{h}_i - \mathbf{h}_j \rangle) \ , \quad (2)$$

where $\hbar(z) = max(0, 1-z)$ is the hinge-loss; $y_{uij} = sign(x_{ui} - x_{uj})$ is the $sign(z)$ function, which returns $+1$ if $z > 0$, i.e., $x_{ui} > x_{uj}$, and -1 if $z < 0$. The prediction function $\langle \mathbf{w}_u, \mathbf{h}_i - \mathbf{h}_j \rangle = \langle \mathbf{w}_u, \mathbf{h}_i \rangle - \langle \mathbf{w}_u, \mathbf{h}_j \rangle$ corresponds to the difference of predictor values $\hat{x}_{ui} - \hat{x}_{uj}$. To conclude this section, we compute the gradient of the pairwise loss at instance $p_t \in P$ with non-zero loss, and model parameters $\theta_t = (\mathbf{w}_u, \mathbf{h}_i, \mathbf{h}_j)$, as follows:

$$-\nabla \hbar(p_t, \theta_t) = \begin{cases} y_{uij} \cdot (\mathbf{h}_i - \mathbf{h}_j) & \text{if } \theta_t = \mathbf{w}_u, \\ y_{uij} \cdot \mathbf{w}_u & \text{if } \theta_t = \mathbf{h}_i, \\ y_{uij} \cdot (-\mathbf{w}_u) & \text{if } \theta_t = \mathbf{h}_j, \\ 0 & \text{otherwise.} \end{cases}$$

Online Learning Algorithm for CF

Our goal is to develop an algorithm to efficiently optimize the objective function (1). Based on the stochastic gradient descent concepts [1], we present the framework of our algorithm in Figure 1. The main components of this framework are: (i) a sampling procedure done on the streaming data, and (ii) a model update based on the sample.

The model update procedure performed by `RMFO` is shown in Figure 2, which includes three regularization constants: λ_W, λ_{H^+}, and λ_{H^-}, one for the user factors, the other two for the positive and negative item factors updates. Moreover, we include a learning rate η and a learning rate *schedule* α that adjusts the step size of the updates at each iteration.

In the rest of the section we explore three variations of our online algorithm based on how the sampling is performed.

Sampling Techniques for Twitter Stream

In this work, we explore the following three variations of our approach based on different stream sampling techniques:

(1) Single Pass (`RMFO-SP`) takes a single pair from the stream and performs an update of the model at every iteration. This approach does not "remember" previously seen instances. That is, we sample a pair $p_t \in P$ at iteration t, and execute procedure *updateModel*(p_t, λ_W, λ_{H^+}, λ_{H^-}, η_0, α, $T_\theta = 1$) (Figure 2).

(2) User Buffer (`RMFO-UB`) retains the most recent b instances per user in the system. In this way, we retain certain amount of history so that the algorithm will run in constant space. For each user, we restrict the maximum number of her items to be kept and denote it by b. More precisely, after receiving the training instance $(u, i, x_{ui})_t$ at time t, the user buffer $|B_u^+|$ for u, is updated as follows:

```
if $|B_u^+| < b$ then $B_u^+ \cup \{i\}$
else
    Delete the oldest instance from $B_u^+$
    $B_u^+ \cup \{i\}$
end if
```

We update the model selecting pairs, $p_t \in P$, from the candidate pairs implied by the collection of all user buffers B, which is defined by the function $B := u \rightarrow B_u^+$.

(3) Reservoir Sampling (RMFO-RSV) involves retaining a fixed size of observed instances in a *reservoir.* The reservoir should capture an accurate "sketch" of history under the constraint of fixed space. The technique of random sampling with a reservoir [13] is widely used in data streaming, and recently has been also proposed for online AUC maximization in the context of binary classification [15]. We represent the reservoir as a list $R := [s_1, s_2 \ldots, s_{|R|}]$ that "remembers" $|R|$ random instances from stream S. Instances can occur more than once in the reservoir, reflecting the distribution of the observed data. We note that this approach also bounds the space available for the algorithm, but in contrast to the user buffer technique, we do not restrict the space per user, but instead randomly choose $|R|$ samples from the stream and update the model using this history.

```
RMFO Framework
Input: Stream representative sample at time $t$: $S_t$; Regularization parameters $\lambda_W$, $\lambda_{H^+}$, and $\lambda_{H^-}$; Learning rate $\eta_0$; Learning rate $\eta_0$; Learning rate schedule $\alpha$; Number of iterations $T_S$, and $T_\theta$; Parameter $c$ to control how often to perform the model updates
Output: $\theta = (\mathbf{W}, \mathbf{H})$
 1: initialize $\mathbf{W}_0$ and $\mathbf{H}_0$
 2: initialize sample stream $S' \leftarrow \emptyset$
 3: counter $\leftarrow 0$
 4: for $t = 1$ to $T_S$ do
 5:     $S' \leftarrow$ updateSample($S_t$)
 6:     counter $\leftarrow$ counter + 1
 7:     if $c =$ counter then
 8:         $\theta \leftarrow$ updateModel($S_t, \lambda_W, \lambda_{H^+}, \lambda_{H^-}, \eta, \alpha, T_\theta$)
 9:         counter $\leftarrow 0$
10:     end if
11: end for
12: return $\theta_T = (\mathbf{W}_T, \mathbf{H}_T)$
```

Figure 1: **RMFO Framework for Online CF.**

```
RMFO Model Update based on SGD for MF
Input: Stream representative sample at time $t$: $S_t$; Regularization parameters $\lambda_W$, $\lambda_{H^+}$, and $\lambda_{H^-}$; Learning rate $\eta_0$; Learning rate schedule $\alpha$; Number of iterations $T_\theta$
Output: $\theta = (\mathbf{W}, \mathbf{H})$
 1: procedure UPDATEMODEL($S_t, \lambda_W, \lambda_{H^+}, \lambda_{H^-}, \eta_0, \alpha, T_\theta$)
 2:     for $t = 1$ to $T_\theta$ do
 3:         $((u,i),(u,j)) \leftarrow \text{randomPair}(S_t) \in P$
 4:         $y_{uij} \leftarrow sign(x_{ui} - x_{uj})$
 5:         $\mathbf{w}_u \leftarrow \mathbf{w}_u + \eta\, y_{uij}\, (\mathbf{h}_i - \mathbf{h}_j) - \eta\, \lambda_W\, \mathbf{w}_u$
 6:         $\mathbf{h}_i \leftarrow \mathbf{h}_i + \eta\, y_{uij}\, \mathbf{w}_u - \eta\, \lambda_{H^+}\, \mathbf{h}_i$
 7:         $\mathbf{h}_j \leftarrow \mathbf{h}_j + \eta\, y_{uij}\, (-\mathbf{w}_u) - \eta\, \lambda_{H^-}\, \mathbf{h}_j$
 8:         $\eta = \alpha \cdot \eta$
 9:     end for
10:     return $\theta = (\mathbf{W}_{T_\theta}, \mathbf{H}_{T_\theta})$
11: end procedure
```

Figure 2: **RMFO Model Update**

3. EXPERIMENTAL STUDY

In this section, we demonstrate our approach by analyzing real-world data consisting of millions of tweets.

476 million Twitter tweets Dataset

The dataset corresponds to the *476 million Twitter tweets* [14]. For our evaluation we computed a 5-core of the dataset, i.e., every user has used at least 5 different hashtags, and every hashtag has been used by least by 5 different users. The 5-core consists of 35,350,508 tweets (i.e., user-item interactions), 413,987 users and 37,297 hashtags.

Evaluation Methodology

Evaluation of a recommender in the presence of stream data requires a time sensitive split. We split the dataset S into training S_{train} and a testing set S_{test} according to a timestamp t_{split}: the individual training examples (tweets) with timestamps less that t_{split} are put into S_{train}, whereas the others go into S_{test}. Note that given the dynamics in Twitter, there might be users in S_{train} not present in S_{test}.

To evaluate the recommenders we followed the *leave-one-out* protocol. In particular, a similar schema as the one described in [2].

For each user $u \in |U_{test}|$ we rank her items in the test set, S_{test}, according to their frequencies and choose one item i at random from the top-10. The goal of a recommender system is to help users to discover new items of interest, therefore we impose the additional restriction that the hidden item has to be *novel* for the user, and therefore we remove from the training set all occurrences of the pair (u, i). In total, we have $|U_{test}| = 260,246$ hidden items. Then, for each hidden item i, we randomly select 1000 additional items from the test set S_{test}. Notice that most of those items selected are probably not interesting to user u. We predict the scores for the hidden item i and for the additional 1000 items, forming a ranking by ordering the 1001 items according to their scores. The best expected result is that the interesting item i_u to user u will precede the rest 1000 random items.

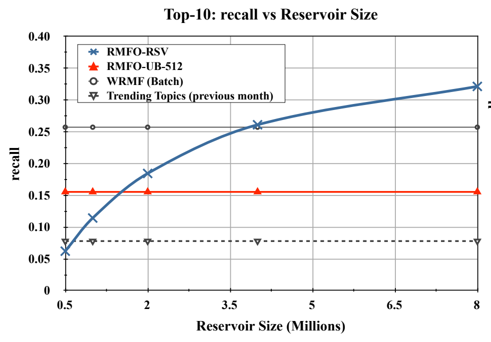
Finally, for each user, we generate a Top-N_u recommendation list by selecting the N items with the highest score. If the test item i_u is in the Top-N_u, then we have a *hit*, otherwise we have a *miss*.

Evaluation Metric: Recall

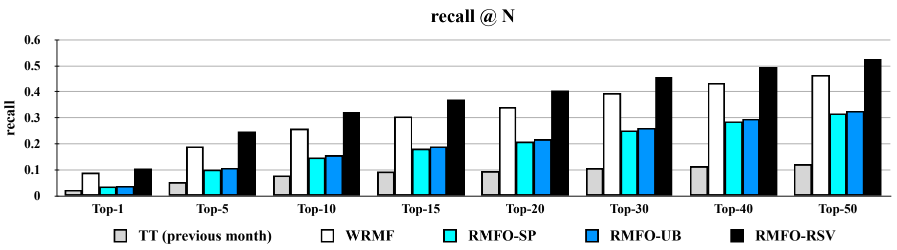
We measure Top-N recommendation performance by looking at the *recall* metric, also known as *hit rate*, which is widely used for evaluating Top-N recommender systems (e.g., [2]). In our recommender systems setting, recall at top-N lists is defined as follows:

$$\textbf{recall@N} := \frac{\sum_{u \in U_{test}} \mathbb{1}_{[i_u \in \text{Top-N}_u]}}{|U_{test}|}\ , \qquad (3)$$

where $\mathbb{1}_{[z]}$ is the indicator function that returns 1 if condition z holds, and 0 otherwise. A recall value of 1.0 indicates that the system was able to always recommend the hidden item, whereas a recall of 0.0 indicates that the system was not able to recommend any of the hidden items. Since the precision is forced by taking into account only a restricted number N of recommendations, there is no need to evaluate *precision* or *F1* measures, i.e., for this kind of scenario, precision is just the same as recall up to a multiplicative constant.



(a)



(b)

Figure 3: **Recommendation performance for (a) different sizes of the reservoir and (b) different Top-N recommendation lists.**

Experimental Setting

We implemented the three variations of our model `RMFO-SP`, `RMFO-UB` and `RMFO-SRV`, and evaluated them against two other competing models:

(1) Trending Topics (TT). This model sorts all hashtags based on their popularity, so that the top recommended hashtags are the most popular ones, which represent the trending topics overall. This naive baseline is surprisingly powerful, as crowds tend to heavily concentrate on few of the many thousands available topics in a given time frame. We evaluate the TT from the whole training set and the ones from the last four weeks before the evaluation.

(2) Weighted Regularized Matrix Factorization (WRMF). This is a state-of-the-art matrix factorization model for item prediction introduced by Hu et al. [5]. WRMF is formulated as a regularized Least-Squares problem, in which a weighting matrix is used to differentiate the contributions from observed interactions (i.e., positive feedback) and unobserved ones. WRMF outperforms neighborhood based (item-item) models in the task of item prediction for implicit feedback datasets, and therefore is considered as a more robust contender. Please note that this reference model is computed in *batch mode*, i.e., assuming that the *whole stream* is stored and available for training. WRMF setup is as follows: $\lambda_{\text{WRMF}} = 0.015$, $C = 1$, $epochs = 15$, which corresponds to a regularization parameter, a confidence weight that is put on positive observations, and to the number of passes over *all* observed data, respectively[1] [5].

For all variations of `RMFO` we simulate the stream receiving one instance at the time based on the tweets' publication dates. Tweets without hashtags were ignored.

For `RMFO-UB`, we want to explore the effect of the user's buffer size b on the recommendation performance, we vary $b \in \{2^m \mid m \in \mathbb{N}, 1 \leq m \leq 9\}$, i.e., from 2 to 512.

For `RMFO-SRV`, we vary the reservoir size $|R| \in \{0.5, 1, 2, 4, 8\}$ million, and compute the model using 15 epochs over the reservoir only. We set regularization constants $\lambda_W = \lambda_{H^+} = \lambda_{H^-} = 0.1$, learning rate $\eta_0 = 0.1$, and a learning rate schedule $\alpha = 1$, and find that the setting gives good performance. We are currently investigating how to efficiently perform a grid search on stream data to tune-up the hyperparameters dynamically.

We divide the seven-month Twitter activity of our dataset by choosing the first six months for training. We use the remaining month, i.e., December, to build 10 independent test sets following the evaluation protocol described previously in this section. We compute the recall metric for Top-N recommendations, where $N \in \{1, 5, 10, 15, 20, 30, 40, 50\}$. The performance is evaluated on the test set only, and the reported results are the average over 10 runs.

[1] We have observed that WRMF is not so sensitive to changes in the hyperparameters, the most important aspect is the number of iterations before early stopping, i.e., epochs=15

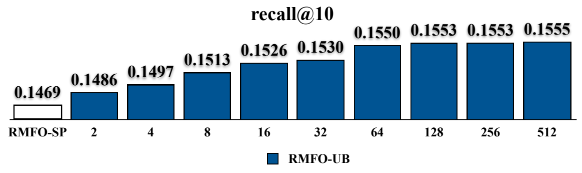


Figure 4: **`RMFO-SP` and `RMFO-UB` Top-*10* performance for different sizes of user buffer.**

Results and Discussion

We found that recent topics are more valuable for recommendations: trending topics from the previous four weeks achieve a recall@10 of 7.8%, compared to 6.77% from the ones corresponding to the whole training period (6 months). The performance exhibited by this recommender, based on the crowd behavior in Twitter, largely outperforms a random model, whose recall@10 is under 1%. In the rest of the discussion we focus only on the recent trending topics.

Figure 4 shows the recommendation quality in terms of recall@10 for `RMFO-SP`, and `RMFO-UB` with varied user buffer sizes. We can see that recall@10 for `RMFO-SP` is 14.69%, 88.3% better than the overall trend.

We also observed that having a per-user buffer improves the performance. However if the buffer is small (e.g., 2 or 4), `RMFO-UB` achieves low recall. Although increasing the buffer size boosts the recommendation quality, we found that as the quality reaches a plateau (see Figure 4), the buffer size provides limited improvements.

Figure 3a shows that `RMFO-SRV` achieves the best performance over all methods evaluated when the reservoir size is greater than 4 million, which corresponds to 11.32% of the entire number of transactions in the dataset. We summarize in Figure 3b the best performance achieved by the methods evaluated for different Top-N recommendations.

With a fixed reservoir size of 8M, we also explored the impact of model dimensionality over the recommendation quality for `RMFO-RSV`. The results are presented in Figure 5. From the figure, we see that the 16-factor low-rank approximation given by `RMFO-RSV` exhibits a better recall@10 than WRMF computed in batch mode using 128 factors.

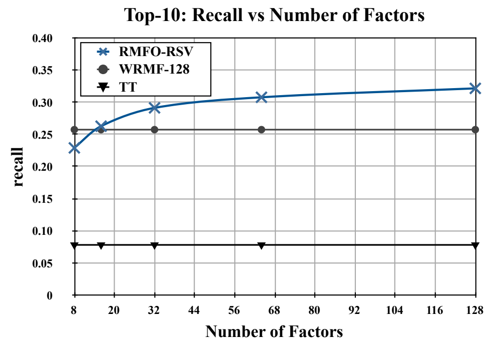


Figure 5: **Performance for different number of factors.**

Time, Space and Performance Gains

We report in this section the CPU training times and space required for the best performing variation of our online approach: RMFO-RSV, and the ones for the strongest baseline: WRMF. Please remember that running times heavily depend on platform and implementation, so they should be only taken as relative indicators.

All variations of RMFO were implemented in Python. RMFO ran on a Intel Xeon 1.87GHz machine. For WRMF, we used the C# implementation provided by *MyMediaLite* library [4]. The baseline WRMF was run on a machine with a slightly faster CPU (Intel Xeon 2.27GHz). None of the methods was parallelized and therefore used a single CPU for computations. GNU/Linux 64-bit was used as OS.

In Table 1, we can observe the gains in speed of our approach over the baseline for all the evaluated reservoir sizes. For reservoir sizes of 4M and 8M, RMFO-RSV is not only faster and space efficient, but also exhibits a better recommendation performance with respect to WRMF, for example, RMFO-RSV with a reservoir size 8M is over 36 times faster and uses 77% less space than WRMF, and yet it delivers a recommendation performance almost 25% better than the state-of-the-art baseline. As a reference, we also include the performance of RMFO-RSV INF, which uses an *infinite* reservoir, e.g., one that is able to remember all observed transactions.

Method (128 factors)	Time (seconds)	recall@10	Space	Gain in speed	Gain in recall
WRMF (Baseline)	23127.34	0.2573	100.00%	–	–
RMFO-RSV 0.5 M	47.97	0.0621	1.41%	482.16	-75.85%
RMFO-RSV 1 M	89.15	0.1143	2.83%	259.42	-55.56%
RMFO-RSV 2 M	171.18	0.1845	5.66%	135.11	-28.30%
RMFO-RSV 4 M	329.60	0.2611	11.32%	70.17	+1.49%
RMFO-RSV 8 M	633.85	0.3215	22.63%	36.49	+24.95%
RMFO-RSV INF	1654.52	0.3521	100.00%	13.98	+36.84%

Table 1: **Time, Space and Performance Gains.**

4. RELATED WORK

Online learning of matrix factorization methods for rating prediction have been investigated by Rendle and Schmidt-Thieme in [10]. They propose online update rules on a stochastic gradient descent style based on the *last* example observed. However, the best performing variant of our approach, RMFO-RSV, maintains a *reservoir* with a representative set of previously seen data points from the stream, which provides a significant boost in performance compared to the one obtained when only the last example is considered (e.g., RMFO-SP). The technique of random sampling with a reservoir is widely used in data streaming [13], and recently has also been exploited by Zhao et al. in the context of binary classification [15].

5. CONCLUSIONS AND FUTURE WORK

This paper provides an example of integrating large-scale collaborative filtering with the real-time nature of Twitter.

We proposed RMFO, an approach for recommending topics to users in presence of streaming data. Our online setting for collaborative filtering captures "What is interesting to *me* right now?" in the social media stream.

RMFO receives instances from a microblog stream, and updates a matrix factorization model following a pairwise learning to rank approach for dyadic data. At the core of RMFO is stochastic gradient descent which makes our algorithm easy to implement and efficiently scalable to large-scale datasets. From the RMFO's variants explored in this work, we found that the one using reservoir sampling technique performed the best.

Our empirical study used Twitter as test bed and showed that our approach worked well relative to matrix factorization models computed in batch mode, in terms of recommendation quality, speed and space efficiency.

Currently, we are investigating alternative sampling techniques, for example, based on active learning principles that select the instances based on their gradients, thus keeping the most informative ones in the reservoir. Initial promising results towards this direction can be found in [3].

Acknowledgments This work was funded, in part, by the European Commission FP7/2007-2013 under grant agreement No.247829 for the M-Eco Project and the DFG project Multi-relational Factorization Models. Lucas Drumond is sponsored by a scholarship from CNPq, a Brazilian government institution for scientific development.

6. REFERENCES

[1] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT'2010*, 2010.
[2] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *ACM RecSys Conference*, 2010.
[3] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-Time Top-N Recommendation in Social Streams. In *ACM RecSys Conference*, 2012.
[4] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. MyMediaLite: A free recommender system library. In *ACM RecSys Conference*, 2011.
[5] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proc. of the 8th IEEE International Conference on Data Mining*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.
[6] T. Joachims. "optimizing search engines using clickthrough data". In *ACM Conference KDD*, 2002.
[7] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, August 2009.
[8] S. Muthukrishnan. *Data streams: algorithms and applications.* Now Publishers, 2005.
[9] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI Conference*, 2009.
[10] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *ACM RecSys Conference*, 2008.
[11] D. Sculley. Combined regression and ranking. In *ACM Conference KDD*, 2010.
[12] Semiocast. Countries on Twitter. http://goo.gl/RfxZw, 2012.
[13] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11:37–57, March 1985.
[14] J. Yang and J. Leskovec. "patterns of temporal variation in online media". In *ACM Conference WSDM*, 2011.
[15] P. Zhao, S. Hoi, R. Jin, and T. Yang. "online auc maximization". In *ICML*, 2011.