Real-Time Top-N Recommendation in Social Streams

Ernesto Diaz-Aviles[1], Lucas Drumond[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, schmidt-thieme}@ISMLL.de

ABSTRACT

The Social Web is successfully established, and steadily growing in terms of users, content and services. People generate and consume data in real-time within social networking services, such as Twitter, and increasingly rely upon continuous streams of messages for real-time access to fresh knowledge about current affairs. In this paper, we focus on analyzing social streams in real-time for personalized topic recommendation and discovery. We consider collaborative filtering as an online ranking problem and present *Stream Ranking Matrix Factorization* – `RMFX` –, which uses a pairwise approach to matrix factorization in order to optimize the personalized ranking of topics. Our novel approach follows a selective sampling strategy to perform online model updates based on active learning principles, that closely simulates the task of identifying relevant items from a pool of mostly uninteresting ones. `RMFX` is particularly suitable for large scale applications and experiments on the *476 million Twitter tweets* dataset show that our online approach largely outperforms recommendations based on Twitter's global trend, and it is also able to deliver highly competitive Top-N recommendations faster while using less space than Weighted Regularized Matrix Factorization (WRMF), a state-of-the-art matrix factorization technique for Collaborative Filtering, demonstrating the efficacy of our approach.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]—*Information Filtering*

General Terms: Algorithms, Experimentation, Measurement, Performance

Keywords: Collaborative Filtering; Matrix Factorization; Online Learning; Ranking; Selective Sampling; Twitter

1. INTRODUCTION

The amount of user generated content in social media applications and the rate at which such content is made available poses a challenge to state-of-the-art recommender system algorithms. For instance, the number of users of the popular micro-blog service *Twitter*[1] is estimated to have surpassed 300 million generating more than 200 million *tweets* (micro-blog posts) per day [18].

When dealing with user generated content in social media applications, it is crucial that finding good patterns and making inference about them is done in a reasonable time. One scenario where this point becomes particularly critical is that of filtering a continuous stream of incoming tweets in order to recommend topics that match user interests at a specific moment, given the huge scale of this kind of data. The challenge here is to account for dynamic short term information needs of the users. As a surrogate for a user's topic interests one could use *hashtags*. For example, in Twitter, if user *Alice* often tags her tweets with the hashtag *#TechCrunch* and never uses the hashtag *#fashion*, we can exploit this information and use it as a good indicator for her preferences. We can infer that, currently, *Alice* is more interested in *technology news* than, for instance, in *fashion*. Thus the task can be cast as that of recommending hashtags to users.

Collaborative filtering (CF) has been shown to be an effective approach to recommender systems. The essence of CF lies in analyzing past user and item interactions to generate personalized recommendations based on the preferences of other users with similar behavior. One of CF's most successful techniques are low dimensional linear factor models, that assume user preferences can be modeled by only a small number of *latent* factors [11].

Although latent factor models are able to generate high quality recommendations, coping with fast changing trends in the presence of large scale data might be a challenge, since retraining such models is costly. One alternative is to learn the parameters online, updating the decision function for each new observation [2]. Unfortunately, the gain in processing time achieved by online learning algorithms comes at the cost of reduced prediction quality and work has been done on closing this gap. The main issue with online approaches is their short-term "memory", i.e., since the updates based only on the most recent data point do not take into account past observations, the model quickly "forgets" them. Recently Zhao et al. [23] proposed an online learning algorithm for maximizing Area Under the ROC Curve (AUC), a metric that is widely used for measuring the classification performance for imbalanced data distributions, that addresses this problem by keeping a representative sample of the data set in a reservoir and using only this sample plus the new observation for retraining.

[1]**Twitter**: `http://twitter.com`

One issue that arises with the reservoir approach is how to perform the model updates. In this paper we propose to selectively sample the most informative instances from a reservoir using personalized small buffers and perform stochastic gradient descent updates based on *active learning* principles [6, 22]. We elaborate on the notion of "informative elements" and illustrate the application of our approach to learning factorization models. We demonstrate its usefulness on the task of recommending hashtags to Twitter users based on real world data.

To summarize, the main contributions of this work are as follows:

- We introduce a novel framework for online collaborative filtering. The novelty of our approach lies in a selective sampling strategy to update the model based on personalized small buffers.
- We propose Stream Ranking Matrix Factorization (`RMFX`), an online learning algorithm based on a pairwise ranking approach for matrix factorization that is intended for streaming data, and is well founded in stochastic gradient descent.
- For unpersonalized learning to rank, many studies have been made in the field of information retrieval. This paper presents an innovative personalized ranking perspective to matrix factorization for social media streams, which has not been reported before in the literature.
- Finally, this paper provides an example of integrating large-scale collaborative filtering with the real-time nature of Twitter.

The reminder of the paper is organized as follows: In Section 2, we present background material and notation. In Section 3, we present our Stream Ranking Matrix Factorization approach. Section 4 discusses related work. In Section 5, we show the results of our approach by analyzing real-world data consisting of millions of tweets. Finally, in Section 6, we conclude and present directions for future work.

2. BACKGROUND

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}). For example in the movie recommendation case, the tuple might correspond to an explicit "rating" given by user u to movie i or, in the case of hashtag/topic recommendation, to a "weight" that is implicitly derived from user u's interaction patterns, e.g., how many times the user u has used hashtag i. Typical CFs 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. The relational scores themselves are ordinal and need not be numbers. Thus, 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 S of all observed scores is defined as follows:

$$S := \{(u, i, x_{ui}) \mid (u, i, x_{ui}) \in U \times I \times \mathbb{N}\} \ .$$

For convenience, we also define for each user the set of all items with an observed score, denoted by B_u^+:

$$B_u^+ := \{i \in I \mid (u, i, x_{ui}) \in S\} \ .$$

Low dimensional linear factor modeling are popular collaborative filtering approaches [11]. 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). Some of the most successful realizations of latent factor models are based on matrix factorization (MF). In its basic form, matrix factorization estimates a matrix $\mathbf{X} : U \times I$ by the product of two low-rank matrices $\mathbf{W} : |U| \times k$ and $\mathbf{H} : |I| \times k$:

$$\hat{\mathbf{X}} := \mathbf{W}\mathbf{H}^\intercal \ , \tag{1}$$

where k is a parameter corresponding to the rank of the approximation.

The factorization process is performed by minimizing a loss function that measures the quality of the reconstruction $\hat{\mathbf{X}}$. One alternative to learn the optimal parameters of the model is to use a Stochastic Gradient Descent (SGD) approach [2].

Even though the *squared loss* has been successfully used for MF in the context of rating prediction (e.g., [11]) and item prediction [8], we are interested in a *ranking* approach to MF, and therefore require an *ordinal loss* to guide the factorization process. In particular we are interested in a *pairwise* approach, similar to the one used by RankSVM [9], a popular ranking method in the field of learning to rank. We present the details of our approach in the next section and discuss related work in Section 4.

3. STREAM RANKING MATRIX FACTORIZATION

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.

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 input data, which can be significantly larger than the algorithm's available memory.

This problem setting fits a streaming model of computation by Muthukrishnan [12], which establishes that, by imposing a space restriction on algorithms that process streaming data, we may not be able to store all the data we see. The impact is that the data generated in real-time carries high-dimensional information which is difficult to extract and process. Any time lag in modeling the data could render the outcome of the modeling obsolete and useless.

We assume that topics of interest are captured by the hash-tagging behavior in Twitter. Hashtags are words or phrases prefixed with the symbol #, e.g., *#recsys*, a form of metadata tag used to mark keywords or topics in a tweet. Hashtags evolve over time, reflecting the dynamics of user preferences in the social stream. Our approach seeks to incorporate these dynamics to produce a short list of interesting recommendations based on a matrix factorization model for CF, which is learned online.

In this section, we formally define the problem and introduce our approach Stream Ranking Matrix Factorization or RMFX, and develop our model in steps discussing the rationale behind them. Such steps are illustrated in Figure 1.

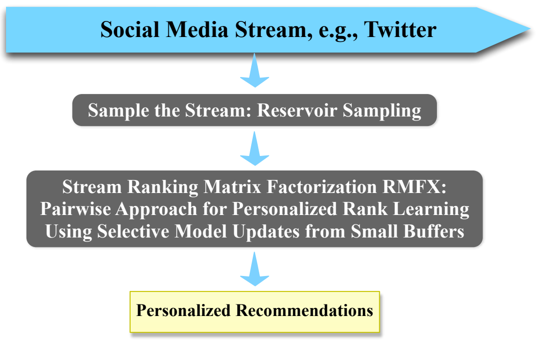
Social Media Stream, e.g., Twitter
Sample the Stream: Reservoir Sampling
Stream Ranking Matrix Factorization RMFX: Pairwise Approach for Personalized Rank Learning Using Selective Model Updates from Small Buffers
Personalized Recommendations

Figure 1: **Main steps of our approach Stream Ranking Matrix Factorization (RMFX).**

3.1 Pairwise Approach for Personalized Rank Learning

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 Scully [15] 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 $(u,j) \notin S$ 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. [13] 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^+.

It is obvious that, in the case of streaming data, we do not compute P explicitly, but instead select pairs from the stream, at each time step, that meet P's membership requirements.

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

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

In this paper, we explore the use of the SVM loss, or *hinge-loss*, used by RankSVM for the learning to rank task [9].

```
RMFX Framework
Input:
    Reservoir representing a sample of the stream at time
    $t$: $R$; Regularization parameters $\lambda_W$, $\lambda_{H^+}$, and $\lambda_{H^-}$;
    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:     $R \leftarrow$ updateReservoir($R$)
 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 2: **RMFX Framework for Real-Time CF.**

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:

$$L(P, \mathbf{W}, \mathbf{H}) = \frac{1}{|P|} \sum_{p \in P} \hbar(y_{uij} \cdot \langle \mathbf{w}_u, \mathbf{h}_i - \mathbf{h}_j \rangle) \ , \quad (3)$$

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 $\hat{x}_{ui} - \hat{x}_{uj}$.

Please note that in this special case of binary rank values of *observed* and *not-observed*, the optimization problem defined by Eq. (3) is equivalent to the problem of optimizing area under the ROC curve (AUC) for binary-class data [15].

Other convex loss functions can also be applied, e.g., *squared* or *logistic* loss [13, 15], as well as any prediction function besides the dot product $\langle \cdot, \cdot \rangle$ [14].

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:

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

Our goal is to develop an algorithm to efficiently optimize the objective function (2).

Based on stochastic gradient descent concepts [2], we present the framework of our algorithm in Figure 2. The main components of this framework are: (i) a sampling procedure done on the streaming data and (ii) a selective model update based on small buffers created per each user.

3.2 Random Sampling the Social Stream with a Reservoir

When processing streams of data, one usually wants to avoid the cost of retraining a model every time new data points arrive; thus online updates are usually used. Unfortunately, the gain in processing time and bounded space achieved by this online learning approach comes at the cost of reduced prediction quality, compared to more accurate models that the large training set could allow. The main issue with online approaches is their short-term "memory", i.e., since the updates based only on the most recent data point do not take into account past observations, the model quickly "forgets" them. In the presence of an abundant source of training examples, a way to reduce complexity of a learning algorithm consists of picking a random subset of training examples and building a model on this subset. In this phase of our model, we employ the technique of random sampling with a reservoir [20], which is widely used in data streaming, and recently has been proposed for online AUC maximization in the context of binary classification [23].

A reservoir sampling algorithm incrementally maintains a random sample of fixed size of the incoming stream of tweets. 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, thus the reservoir captures an accurate "sketch" of history under the constraint of fixed space.

Let be t the index reflecting the order of arrival of data in the stream, note that until $t = |R|$ all data points enter the reservoir. When $t = |R|$ we have a random sample of size $|R|$ of the stream; indeed the entire dataset so far is in the reservoir. For subsequent t we need to decide whether the newly arrived data should be put in the reservoir and, if so, which data already in the reservoir it should replace.

Vitter shows in [20] that if one includes the t^{th} data instance with probability $|R|/t$ and replaces uniformly at random an instance from the reservoir, the reservoir is a random sample of the current dataset. This reservoir sampling mechanism is implemented by the procedure **updateReservoir**(R) in Figure 2.

3.3 Selective Model Update from Small Buffers

The random sampling with a reservoir allows us to retain a fixed size of observed instances, bounding the space available for the algorithm to a set of $|R|$ randomly chosen samples from the stream and update the model using this history. Although simply updates of the model based on the reservoir may yield better results than single online updates, it is still far from the accuracy achieved by the offline cases. On top of that, in the reservoir we store only user and item pairs observed in the stream, and the question of how to sample the pairs needed for creating the contrasts P still remains.

In order to address this drawback, we need to exploit as much information as possible from the sampled tweets in the reservoir. In particular we propose to perform model updates and retraining on the most informative examples present in the reservoir, then, the question is how to select such examples from this sketch of the stream. This scenario is similar to the one of *active learning*, where the system asks the user to evaluate a minimum set of items which will contribute the most to learning his/her preferences (e.g., [10]).

Consider the case of binary classification using Support Vector Machines (SVM). SVM attempt to find a hyperplane that divides the two classes with the largest margin. From the theoretical foundations of SVM we know that only the support vectors have an effect on the solution. The support vectors are the points that lie closest to the hyperplane, therefore the *most informative* training points, and the goal of training is to discover them [19].

Usually, the training set is chosen to be a random sampling of instances, for example the tweets in our reservoir. However, in many cases principled criteria can be used to sample the training data with the goal to reduce its need for large quantities of labeled data.

Our scenario of dyadic data, i.e., user-item interactions, differs from the one of SVM in two fundamental ways: (i) since we are learning personalized rankings, there are as many hyperplanes as users, unlike an SVM, (ii) we are not just learning a hyperplane per user, but simultaneously also the item feature vectors, in contrast to SVM where the values of the features vectors, defining the training points, are known and given in advance.

Moreover, remember that we are concerned with learning personalized rankings from pairwise comparisons, hence the most informative instances are the ones that have opposite labels but are close to each other in the ranking induced by the user's hyperplane, intuitively they are more difficult to order than the ones away from each other in the ranking [22]. Figure 3 illustrates how user u's feature vector $\mathbf{w}_u$ induces a particular (personalized) ranking at a given iteration in a two dimensional example[2]. $\mathbf{w}_u$ determines the ordering of four item points. For any user *weight* vector $\mathbf{w}_u$, the items are ordered by the projection onto $\mathbf{w}_u$, or equivalently, by their signed distance to a hyperplane with normal vector $\mathbf{w}_u$. The items in the figure are ordered (h_1, h_3, h_2, h_4). We denote as δ the distance between two projections of data points with different labels on the induced ranking, the smaller the δ, the more informative the instances are for training the model.

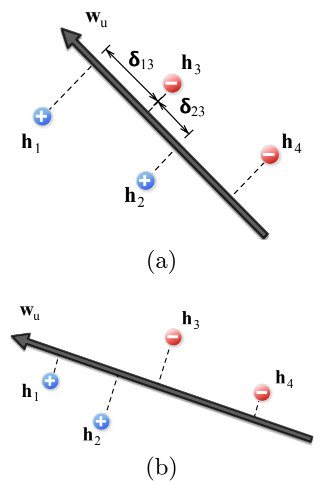

Figure 3: **Example of how a user *weight* vector $\mathbf{w}_u$ ranks four item points. (a) The vector $\mathbf{w}_u$ ranks the points as ($\mathbf{h}_1$, $\mathbf{h}_3$, $\mathbf{h}_2$, $\mathbf{h}_4$), erroneously ranking h_3 higher than h_2. (b) The model updates vector $\mathbf{w}_u$ (user features) and the item features iteratively based on pairwise differences and learns the correct ordering ($\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{h}_3$, $\mathbf{h}_4$). In this example, the pair ($\mathbf{h}_2$, $\mathbf{h}_3$) with δ_{23} is considered more informative than ($\mathbf{h}_1$, $\mathbf{h}_2$) with δ_{13}, since $|\delta_{23}| < |\delta_{13}|$, i.e., the smaller the δ, the more informative the instances are for training the model.**

[2]Observe that Figure 3 can be regarded as a *personalized* adaptation of Figure 2 in [9].

Finally, to answer the question of how to select such examples from the reservoir, we will use an active learning inspired approach. In classical active learning [17], the search for the most informative instance is performed over the entire dataset, which involves the recomputation of each training example's distance to the new hyperplane. This process is prohibitively expensive for large datasets or unbounded data streams. Therefore, we propose a selection method based on the "59 trick" [16, 6], that establishes that randomly sampling only 59 instances, regardless the training set size, is enough to guarantee with 95% probability, that one of them is among the top 5% closest instances to the hyperplane. This approach also simulates the real world scenario of given a pool of items, ranking the positive ones higher than the negatives, modeled into the recommender system evaluation protocol proposed in [3].

At each iteration, we select at random a user-item (u, i) interaction from the reservoir, which represents a positive feedback observation. Next, we construct a small buffer for user u by sampling 59 negative items j's, creating the required *contrast* in the preferences for user u over items i and j's. The user buffer contains exactly 59 pairs of the form $p_b = ((u, i), (u, j_b))$, $b = 1 \dots 59$. Then, we compute the values δ_{uijb} between the projections on $\mathbf{w}_u$ of each instance in the pair p_b. Finally, we sample a pair p^* with probability proportional to its *informativeness*, which is given by $1/\delta_{uijb}$, and use p^* to perform the matrix factorization model updates. This procedure is shown in Figure 4, 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, and a learning rate *schedule* α that adjusts the step size of the updates at each iteration.

```
RMFX Model Update based on SGD for MF using active learning with small buffers
Input:
    Reservoir representing a sample of the stream at time $t$: $R$; 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:         Select a user-item pair $(u, i)$ from $R$ uniformly at random
 4:         Construct a small buffer for user $u$ by sampling 59 negative items $j$'s from $R$ ("59 trick" [16, 6])
 5:         Compute the distances $\delta_{uijb}$ for each pair $p_b = ((u, i), (u, j_b)) \in P$, $b = 1 \dots 59$ in the small buffer
 6:         Sample a pair $p^* = ((u, i), (u, j))$ from the buffer with probability proportional to its informativeness: $1/\delta_{uijb}$
            // Perform the model updates as follows:
 7:         $y_{uij} \leftarrow sign(x_{ui} - x_{uj})$
 8:         $\mathbf{w}_u \leftarrow \mathbf{w}_u + \eta\, y_{uij}\, (\mathbf{h}_i - \mathbf{h}_j) - \eta\, \lambda_W\, \mathbf{w}_u$
 9:         $\mathbf{h}_i \leftarrow \mathbf{h}_i + \eta\, y_{uij}\, \mathbf{w}_u - \eta\, \lambda_{H^+}\, \mathbf{h}_i$
10:         $\mathbf{h}_j \leftarrow \mathbf{h}_j + \eta\, y_{uij}\, (-\mathbf{w}_u) - \eta\, \lambda_{H^-}\, \mathbf{h}_j$
11:         $\eta = \alpha \cdot \eta$
12:     end for
13:     return $\theta = (\mathbf{W}_{T_\theta}, \mathbf{H}_{T_\theta})$
14: end procedure
```

Figure 4: **Matrix factorization model update based on SGD using personalized active learning with small buffers. The procedure minimizes the SVM loss, or hinge-loss, following a pairwise learning to rank approach for dyadic data.**

4. RELATED WORK

Learning of large-scale recommender systems for dealing with dynamic and fast changing content has been addressed before, for instance in the context of the *Google News* system [4]. However the problem setting in [4] is different from the one addressed here, since their work does not deal with a continuous stream of user generated data, but instead provides recommendations to users based on offline models.

The Fast Online Bilinear Factor Model (FOBFM) [1] addresses the related task of click through rate prediction. They combine offline training with online updates in a principled framework. While FOBFM addresses a regression task, we are concerned here with a learn to rank problem. Also our approach does not need an explicit dimensionality reduction step for the offline learned features.

Online matrix factorization learning methods have also been investigated by Rendle and Schmidt-Thieme for rating prediction [14]. They propose online update rules on a stochastic gradient descent style based on the last example observed. While those update rules take into account only the last observed data point, RMFX uses a *reservoir* with a representative set of previously seen data points from the stream. This idea has been previously explored by Zhao et al. [23] in the context of binary classification, in contrast, a novel idea introduced in this work is the selective sampling based on *personalized* buffers according to the distance of points to the decision boundary which, as shown in our experiments, delivers better results than using exclusively the random sampling technique used by Zhao et al. [23].

Yu proposed a selective sampling technique for learning ranking functions in the context of data retrieval applications [22]. Our method, on the other hand, is a learning to rank approach for personalized item prediction.

Since we deal with pairwise classification from positive-only data, negative examples must be sampled. The sampling of the 59 negative examples for each positive one has been proposed, discussed and proved in [6]. Whereas they do it for active learning, we adapt it for our online learning to rank scenario.

5. EXPERIMENTAL STUDY

In this section, we demonstrate our approach by analyzing real-world data consisting of millions of tweets. We present the evaluation protocol and experimental setting, as well as the results of the empirical study.

476 million Twitter tweets Dataset

The dataset corresponds to the *476 million Twitter tweets*[3] [21], which includes over 476 million Twitter posts from 20 million users, covering a 7 month period from June 1, 2009 to December 31, 2009. The number of hashtags present in the dataset is 49,293,684. It is estimated that this is about 20-30% of all public tweets published on Twitter during the particular time frame. 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

[3]http://snap.stanford.edu/data

at least by 5 different users. The 5-core consists of 35,350,508 tweets, 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 evaluated by splitting the dataset S into two sets: a training set S_{train} and a testing set S_{test}. Consider we make the split at time t_{split}, then we put into S_{train} the individual training examples (tweets) with timestamps less that t_{split}. Into S_{test}, we put the user rankings with timestamps greater than t_{split}. The recommenders are trained on S_{train} and then their performance is measured on 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 used a variant of the *all-but-1* protocol, also known as the leave-one-out holdout method. In particular, we follow a similar schema as the one described in [3].

Our goal is to evaluate the system performance when it suggests *Top-N* topics to a user. For example, recommending the user a few specific hashtags which are supposed to be the most attractive to him. That is, to find the relative position of these interesting items within the total order of items ranked for a specific user.

To this end, for each user $u \in |U_{test}|$ we aggregate his rankings in the test set S_{test} by accumulating the item frequencies across those rankings in order to produce a single total ranking. The items are again sorted in descending order of their accumulated frequencies.

We take one item i at random from the top-10 of the aggregated ranking and hide it. 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, we generate a *Top-N* recommendation list by selecting the N items with the highest score. If the test item i_u is in the *Top-N*, then we have a *hit*, otherwise we have a *miss*. We measure the quality by looking at the *recall* metric.

Evaluation Metric: Recall

Traditionally, collaborative filtering algorithms are evaluated by the accuracy of their predicted ratings. One commonly used performance metric for rating accuracy is the Mean Absolute Error (MAE).

However, we are interested in measuring Top-N recommendation performance and not in rating prediction. Therefore, we measure the quality by looking at the *recall* metric, also known as *hit rate*, which is widely used for evaluating Top-N recommender systems (e.g., [5, 3]).

In our recommender systems setting, the *recall* metric is defined as follows:

$$\mathbf{recall} := \frac{\sum_{u \in U_{test}} \mathbb{1}_{[i_u \in \text{Top-N}_u]}}{|U_{test}|} \quad , \tag{4}$$

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.

Experimental Setting

We implemented our `RMFX`, and evaluated them against the following competing models:

1. RMF-RSV: Reservoir Sampling involves retaining a fixed size of observed instances in a *reservoir*. The reservoir captures an accurate "sketch" of history under the constraint of fixed space. We randomly choose $|R|$ samples from the stream and update the model using this history, i.e., without performing any selective sampling.

2. RMF-SP: Single Pass takes a single pair from the stream and performs an update of the model every iteration. This approach does not "remember" previously seen instances. That is, we sample a pair $p_t \in P$ at iteration t, and directly execute the model updates described in lines 7 to 11 in Figure 4.

3. Trending Topics (TT). This model sorts all hashtags based on their popularity, so that the top recommended hashtags are the most popular ones, which represents 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.

4. Weighted Regularized Matrix Factorization (WRMF). This is a state-of-the-art matrix factorization model for item prediction introduced by Hu et al. [8]. Their method outperforms neighborhood based (item-item) models in the task of item prediction for implicit feedback datasets. The model is computed in *batch mode*, assuming that the whole stream is stored and available for training. It is expected that the performance of this offline method, with full access to the user-item interactions, will set an upper bound for the online approaches.

We simulate the stream receiving one instance at the time based on the tweets' publication dates. Tweets without hashtags were ignored.

For `RMFX`, RMF-RSV, and RMF-SP 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. Moreover, the number of iterations is set to the size of the reservoir for both `RMFX` and RMF-RSV.

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 observed data, respectively[4] [8].

We divided the six-month Twitter activity of our dataset, by choosing the first five months (from first of June, 2009 to end of November, 2009) for training. We use the remaining

[4]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

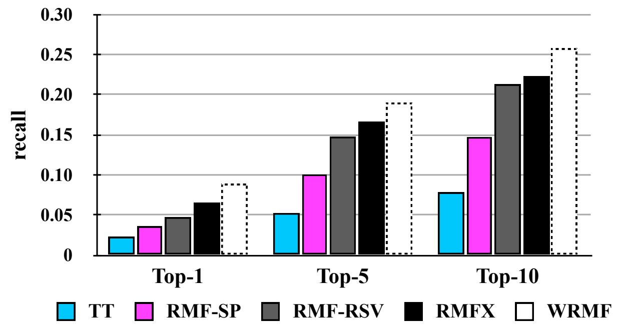

Figure 5: **Recommendation performance in terms of recall for Top–{1, 5, 10}. The reservoir size for RMF-RSV and our method `RMFX` is set to 8M. The number of factors for the matrix factorization models is set to 128. The batch mode WRMF provides an upper bound reference for the the online methods' performance.**

month, i.e., December, to build 10 independent test sets following the evaluation protocol described previously in this section. The models `RMFX` and RMF-RSV are built on the sketch of the stream available just before the evaluation period, i.e., end of November, 2009. For TT, we use as predictors the most popular hashtags from the last four weeks before the evaluation, i.e., TT of November, 2009.

We restricted the analysis to short list of recommendations and computed the recall metric for Top-N recommendations, where $N \in \{1, 5, 10\}$. The value of the metric for a particular Top-N is denoted as *recall@N*. The performance is evaluated on the test set only and the reported results are the average over 5 runs. All the differences reported are statistically significant (two-sample t-test, $p < 0.015$).

Reproducibility of Experiments. We will provide an anonymized dataset of the 5-core dataset used in our experiments and a reference implementation of `RMFX` upon request by email. We used the WRMF implementation provided by *MyMediaLite*, a free software recommender system library[5] [7].

Results

Figure 5 summarizes the recommendation performance for `RMFX`, the baselines and the upper bound given by WRMF. We can observe that `RMFX` is superior with respect to the online methods RMF-SP and RMF-RSV, and largely outperforms the trending topics (TT). Please note that the trending topics from the previous four weeks achieve a recall@10=7.8%, this performance based on the crowd behavior in Twitter is much better than a random model, whose recall@10 is under 1%.

Table 1 shows that `RMFX` achieves the best performance over all online methods evaluated with reservoir sizes 2, 4 and 8 million. As expected, the offline method WRMF sets an upper bound for the online approaches achieving a recall@5 of 18.96%, but `RMFX` is still competitive with a recall@5=16.58% and the advantage of real-time updates.

Finally, with a fixed reservoir size of 8M, we also explored the impact of model dimensionality over the recommendation quality for `RMFX`. The results are presented in Figure 6 and Table 2. From the figure, we see that `RMFX` consistently outperforms the baseline TT and the online competitors for 32, 64 and 128 dimensions.

[5]**MyMediaLite:**
`http://www.ismll.uni-hildesheim.de/mymedialite`

Method	Reservoir Size	Recall@1	Recall@5	Recall@10
`RMFX`	8M	6.50%	16.58%	22.25%
RMF-RSV	8M	4.70%	14.72%	21.25%
`RMFX`	4M	4.16%	11.24%	15.84%
RMF-RSV	4M	2.82%	9.02%	13.70%
`RMFX`	2M	1.95%	5.59%	8.41%
RMF-RSV	2M	1.68%	4.89%	7.36%
RMF-SP	–	3.57%	10.03%	14.69%
TT	–	2.26%	5.22 %	7.80%
WRMF	(Batch)	8.85%	18.96%	25.73%

Table 1: **Recommendation performance for different sizes of the reservoir. The number of factors is set to 128 for the online methods `RMFX`, RMF-RSV, as well as for the offline approach WRMF.**

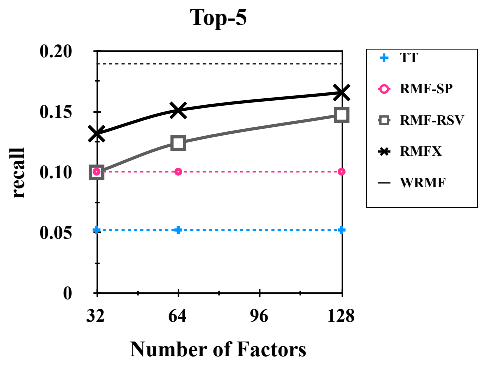

Figure 6: **Recall@5 for 32, 64 and 128 factors.**

Method	Factors	Recall@1	Recall@5	Recall@10
`RMFX`	64	5.57%	15.10%	20.79%
RMF-RSV	64	3.79%	12.40%	18.89%
`RMFX`	32	4.32%	13.18%	18.96%
RMF-RSV	32	2.69%	9.98%	16.07%

Table 2: **Recommendation performance for 32 and 64 factors, with a fixed reservoir of size 8M (the information for 128 factors is shown in Table 1).**

Time and Space Savings vs. Recommendation Quality

We report in this section the CPU training times and space required for the best performing variation of our online approach: `RMFX`, and the ones for the strongest baseline: WRMF. We also discuss the trade-off between time and space savings against the recommendation performance.

`RMFX` is implemented in the Python programming language using SciPy[6]. We ran `RMFX` on a Intel Xeon 1.87GHz machine. For WRMF, we used the implementation provided by *MyMediaLite* [7], which is implemented in C#. The baseline WRMF was run on a machine with a slightly faster CPU (Intel Xeon 2.27GHz). The experiments were conducted using GNU/Linux 64-bit as operating system. None of the methods was parallelized and therefore used one single CPU for computations. Please remember that running times heavily depend on platform and implementation, so they should be only taken as relative indicators.

In Table 3, we can observe the gains in speed of our approach over the baseline for all the evaluated reservoir sizes. We can observe that for all reservoir sizes `RMFX` is faster and more space efficient than WRMF. For example, `RMFX` with a

[6]**SciPy :** `http://scipy.org`

Method (128 factors)	**Time** (hh:mm:ss)	**recall@10**	**Space**	**x times faster than WRMF**	**Recommendation Quality w.r.t WRMF**
WRMF (Batch) [Baseline]	96:21:50.093	0.2573	100.00%	–	100%
`RMFX` 2M	0:33:12.898	0.0841	5.66%	174.07	32.69%
`RMFX` 4M	1:07:10.570	0.1584	11.32%	86.07	61.52%
`RMFX` 8M	2:14:32.355	0.2225	22.63%	42.98	86.47%

Table 3: **Time, Space and Recommendation Quality Comparison. `RMFX` is compared against WRMF for different reservoir sizes: 2, 4 and 8 million. The dimensionality of all methods is 128 factors. Time is measured in seconds and corresponds to the training time of 15 epochs in case of WRMF and one single epoch in case of `RMFX`, the gain is computed w.r.t. WRMF's duration. Space is measured as the percentage of all transactions in the dataset, which is 100%=35.35M. Recommendation performance quality is computed with respect to WRMF's recall, which is set as an upper-bound.**

reservoir size 8M is approximately 43 times faster and uses 77% less space than WRMF, and yet it delivers a highly competitive recommendation performance corresponding to 86.47% of the state-of-the-art baseline computed offline.

6. CONCLUSIONS AND FUTURE WORK

We proposed `RMFX`, 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 within the social media stream". `RMFX` represents a novel principled approach for online learning from streams, that selects a subsample of the observed data based on the objective function gradients, and uses it to guide the matrix factorization.

`RMFX` 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 `RMFX` is stochastic gradient descent which makes our algorithm easy to implement and efficiently scalable to large-scale datasets. Our empirical study used Twitter as a test bed and showed that models updated using the selective sampling approach proposed here, significantly outperform online methods that use random samples of the data.

In future work, we plan to investigate how the frequency of repeated events (i.e. users using the same hashtag or listening to the same song) can be incorporated to the model to generate more accurate predictions.

Acknowledgments We would like to thank Zeno Gantner and Stefan Siersdorfer for fruitful discussions during the genesis of this paper. This work was funded, in part, by the European Commission FP7/2007-2013 under grant agreement No.247829 for the M-Eco Project, the DFG project Multi-relational Factorization Models, and the NTH School (Niedersächsische Technische Hochschule) for IT Ecosystems. Lucas Drumond is sponsored by a scholarship from CNPq, a Brazilian government institution for scientific development.

7. REFERENCES

[1] D. Agarwal, B.-C. Chen, and P. Elango. Fast online learning through offline initialization for time-sensitive recommendation. In *Proceedings of the ACM KDD conference*, 2010.

[2] L. Bottou. Online Algorithms and Stochastic Approximations. In *Online Learning and Neural Networks*. 1998.

[3] P. Cremonesi, Y. Koren, and R. Turrin. Performance of Recommender Algorithms on Top-N recommendation Tasks. In *Proceedings of the ACM RecSys conference*, 2010.

[4] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the World Wide Web*, 2007.

[5] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22:143–177, January 2004.

[6] S. Ertekin, J. Huang, L. Bottou, and L. Giles. Learning on the Border: Active Learning in Imbalanced Data Classification. In *Proceedings of the ACM CIKM conference*, 2007.

[7] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. MyMediaLite: A free recommender system library. In *Proceedings of the ACM RecSys conference*, 2011.

[8] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE ICDM conference*, 2008.

[9] T. Joachims. Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, 2002.

[10] R. Karimi, C. Freudenthaler, A. Nanopoulos, and L. Schmidt-Thieme. Towards optimal active learning for matrix factorization in recommender systems. In *IEEE ICTAI conference*, 2011.

[11] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, August 2009.

[12] S. Muthukrishnan. *Data streams: algorithms and applications*. Now Publishers, 2005.

[13] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of UAI conference*, 2009.

[14] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the ACM RecSys conference*, 2008.

[15] D. Sculley. Combined regression and ranking. In *Proceedings of the ACM KDD conference*, 2010.

[16] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the International Conference on Machine Learning*, 2000.

[17] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, Mar. 2002.

[18] twittereng. 200 million tweets per day. Twitter Blog. `http://goo.gl/eybp0`, June 2011.

[19] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[20] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11:37–57, March 1985.

[21] J. Yang and J. Leskovec. Patters of temporal variation in online media. In *Proceedings of the ACM WSDM conference*, 2011.

[22] H. Yu. SVM Selective Sampling for Ranking with Application to Data Retrieval. In *Proceedings of the eleventh ACM KDD conference*, 2005.

[23] P. Zhao, S. Hoi, R. Jin, and T. Yang. Online AUC Maximization. In *Proceedings of the International Conference on Machine Learning*, 2011.