Streaming Ranking Based Recommender Systems

Weiqing Wang
The University of Queensland
weiqingwang@uq.edu.au

Hongzhi Yin*
The University of Queensland
h.yin1@uq.edu.au

Zi Huang
The University of Queensland
huang@itee.uq.edu.au

Qinyong Wang
The University of Queensland
qinyong.wang@uq.edu.au

Xingzhong Du
The University of Queensland
x.du@uq.edu.au

Quoc Viet Hung Nguyen
Griffith University
quocviethung1@gmail.com

ABSTRACT

Studying recommender systems under streaming scenarios has become increasingly important because real-world applications produce data continuously and rapidly. However, most existing recommender systems today are designed in the context of an offline setting. Compared with the traditional recommender systems, large-volume and high-velocity are posing severe challenges for streaming recommender systems. In this paper, we investigate the problem of streaming recommendations being subject to higher input rates than they can immediately process with their available system resources (i.e., CPU and memory). In particular, we provide a principled framework called as SPMF (Stream-centered Probabilistic Matrix Factorization model), based on BPR (Bayesian Personalized Ranking) optimization framework, for performing efficient ranking based recommendations in stream settings. Experiments on three real-world datasets illustrate the superiority of SPMF in online recommendations.

CCS CONCEPTS

• **Information systems → Data mining**; **Retrieval models and ranking**;

KEYWORDS

recommender systems; information retrieval; streaming data; online applications; user behaviour modeling

ACM Reference Format:
Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. 2018. Streaming Ranking Based Recommender Systems. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3210016

*This author is the corresponding author.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210016

1 INTRODUCTION

The success of many web-enabled services has largely been attributed to recommender systems, which are the key to attracting users and promoting items. Most real-world systems generate massive data at an unprecedented rate. For example, more than 10 million transactions are made per day in eBay [1] and about half a billion tweets are generated every day [11]. These data are temporally ordered, large-volume and high-velocity, which necessitate real-time streaming recommendation algorithms [33].

Most existing recommendation methods are static, e.g. nearest neighbors and correlation coefficients usually are precomputed for collaborative filtering or factor matrices in matrix factorization. This means that the new coming data in the streams cannot be integrated into the trained model efficiently for classic recommender systems. There are three major challenges to be tackled in designing a streaming recommender system.

Capturing users' long-term interests. One alternative of developing a streaming recommender system is to learn the parameters of some classic recommender systems (e.g., latent factor models) online with stochastic gradient descent method, updating users' interests based on each new observation [20]. The main issue with this online approach is that they cannot maintain users' long-term interests because of their short-term "memory". Specifically, since the updates of users' interests are only based on the most recent data points, the model quickly "forgets" users' past behaviors [4].

Users' drifted interests and modeling new users or items. The data in streams are temporarily ordered. Users' preference may drift over time [30, 32, 35]. For example, a mother tends to be interested in different goods for children as her child grows up. How to capture users' latest interests to avoid being overwhelmed by the large amount of data in the past is also important in streaming recommender systems. On the other hand, new users and new items arrive continuously in data streams. For example, from 2007 to 2015, the number of registered users on Amazon saw a dramatic increase from 76 millions to 304 millions[1]. How to identify and model new users or items from the large-volume and high velocity data streams is another major challenge in streaming recommender systems.

Overload. Stream-oriented systems usually confront higher input rates than they can immediately process with their available computing resources [10, 26], including recommender systems [20]. When input rates exceed the computing capacity, the system becomes overloaded. Figure 1 is a simplified overload scenario. Assume that, for the streaming recommender systems in [1, 4], the process time

[1]https://www.statista.com/statistics/237810/number-of-active-amazon-customer-accounts-worldwide/

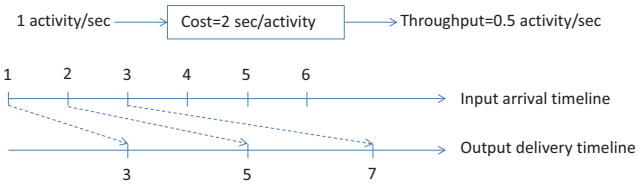


Figure 1: A Simple Overload Scenario

cost by each new input data is 2 seconds but the streaming data are coming at 1 activity per second. The input arrival timeline shows the time points when a new activity arrives and the output delivery timeline shows the points where the processing of corresponding input activities are delivered. Since the system resources are able to process only 1 activity every 2 seconds, a queue starts to build up and response time starts increasing due to the queue waiting time.

Recently, several methods have been developed to make recommendations dynamically for streaming data [1, 2, 4, 20, 23]. As a matter of fact, these approaches either do not address all the three challenges, or address the challenges with ineffective strategies.

In [4], Diaz-Aviles et. al maintain a random sample from the coming data and update the model only based on the sampled instances. Their aim is to stop the system from forgetting users' long-term interests. However, its sampling strategy tends to overlook users' drifted interests as it has smaller probability of sampling to more recent data. Specifically, the coming data are sampled based on the rule including the t^{th} data instance in the stream with the probability proportional to $\frac{1}{t}$. With this rule, as time goes by (i.e., t increases), the newly generated data have decreasing probabilities to be sampled. In this way, this model has low capability in capturing users' drifted preferences and modeling new generated users or items.

Some other works designed for online recommendation, such as [1, 23], are able to capture users' interest drifts by designing schemes to update the models only based on the new generated data. However, they all tend to forget users' long-term interests as they only update their model based on the most recent data points [4]. Another major limitation is that they both overlook the overload problem by assuming that they are able to learn sufficiently from all the newly generated data. To deal with the possible overload problem, Rendle et al. present approaches that determine whether online updates are performed or not on a data instance based on two metrics in [20]. First metric aims at selecting the instances related to users with fewer history activities. The other metric is based on the error between the true rating and the predicted rating for a data instance. This work is rating oriented and has been proven by Diaz-Aviles et al. tending to forget users' long-term interest in [4].

In this paper, we propose SPMF, a stream-centered probabilistic matrix factorization model, for streaming recommendation. SPMF is based on PMF (probabilistic matrix factorization model) [22], which has been widely used in recommendation because of its scalability in large data sets [22, 31]. The goal of most traditional PMF models is to predict a personalized score r_{ui} which represents the preference of user u for item i. However, the task of personalized recommendation in many real applications is to provide a user with a ranked list of items. In this paper, under the guide of feasible extension to streaming setting, we design a novel and scalable offline pairwise PMF based on BPR [19], which treats personalized recommendation problem as personalized ranking and thus integrates the goal of ranking into the optimization process. Stochastic gradient decent (SGD) method, which is well known suitable for online updating, is adopted to optimize the proposed model.

To capture users' long-term interests, SPMF employs the technique of reservoir, which has been widely used in stream-centered data management systems [4, 5, 27], to maintain a sample of history data. SPMF samples the data following the principle of capturing an accurate "sketch" of history data instances under the constraint of fixed memory space. To capture both long-term interests and drifted interests, SPMF uses the samples in the reservoir plus the new observations in data streams as the input for updating the model.

For the overload problem in streaming recommendation, we design a wise positive sampling strategy to sample informative subsets from the designed input to conduct model updating. There are many different sampling strategies [18, 19, 25, 31], and all these sampling strategies reduce the load at the expense of increasing sampling time. Under the high-velocity streaming setting, we have limited time to do both sampling and updating. Thus, a major challenge in designing such a positive sampling strategy under streaming setting is to trade off between sampling time and updating accuracy. In this paper, we design a novel Gaussian classification model, to sample the data wisely and efficiently from the reservoir and the new observations in data streams. The basic idea is that the activities at a lower rank should have a higher probability to be sampled, as this kind of activities are more informative and helpful in correcting the model.

With this Gaussian classification model, SPMF is able to deal with new users, new items and interest drift problems. The rank of an instance (u_i, v_j) is related to the current system's predicting ability on the fact that user u_i is interested in item v_j. This predicting ability is learned from the history activities containing the similar patterns. A higher rank represents that there are more history activities indicating that u_i is interested in v_j. For new users, it is obvious that the system has no prior knowledge at all about their preferences and the instances related to them will get a lower rank. Similarly, for the new items, there is also no history activities about them and the system cannot predict users' preference over them. Thus, the data containing new items will also get a lower rank. Interest drift problem about existing users refers to the phenomenon that the users' interest change over time. It's obvious that this scheme also prefers to sample the data instances containing drifted interest for existing users as these instances contain very different behavior patterns compared with the history patterns captured by the obtained model parameters.

This paper is organized as follows: Section 2 provides some mathematical notations and preliminary concepts about the traditional recommendation model. Section 3 formulates our streaming recommendation problem and describes the SPMF model. Section 4 discusses the experimental evaluation of SPMF model. Existing research related to our work is surveyed in Section 5. Section 6 closes this paper with some conclusive remarks.

2 PRELIMINARIES

In this section, we will provide some preliminary concepts about the traditional recommendation model. Before that, let us define

some mathematical notations that will be useful in both this section and the following parts.

Let $\mathcal{U} = \{u_1, \ldots, u_I\}$ be the set of users, $\mathcal{V} = \{v_1, \ldots, v_J\}$ be the set of all items. I and J are used to denote the numbers of users and items, respectively. We reserve special indexing letters to distinguish users from items: for users i and for items j. Suppose we have interaction between u_i and v_j, we observe a score x_{ij}.

Following the notation, each instance of the data is a tuple (u_i, v_j, x_{ij}). In most existing recommender systems, each rating x_{ij} corresponds to either an explicit "rating" given by the user u_i to the item v_j or to a "weight" derived from u_i's interaction patterns with v_j, e.g., how many times u_i has checked in at a location v_j. The task of traditional recommender systems is to estimate the score x_{ij} for the missing entries.

Matrix factorization (MF) based models are popular collaborative filtering approaches [24]. They assume that users' preferences are influenced by a small number of latent factors. Specifically, MF models estimate each rating x_{ij} based on the latent factors. The goal of MF is to minimize the differences between the estimated ratings and the real ratings. The squared loss has been widely used for measuring these differences.

However, in many real applications, personalized recommendation is to provide users with a ranked list of items instead of predicting a rating [4, 19]. For example, recommender systems on e-commerce are to provide users with a list of products that they may be interested in. For most traditional MF models, they provide the ranked unrated items based on their estimated scores. In these models, ranking is separated from optimization process. In this paper, we are interested in a pairwise approach, similar to the one used by BPR [19], a popular method integrating ranking into optimization. We present the details of our approach in the next section.

3 STREAMING PROBABILISTIC MATRIX FACTORIZATION MODEL

The goal of our model is to learn a probabilistic matrix factorization model in the presence of streaming data. To adapt to the ranking problem in recommendation, the proposed model will follow a pairwise approach similar to BPR [19]. In this section, we first propose a novel probabilistic matrix factorization model under the off-line setting with the pairwise optimization framework. Then, we extend this model to the streaming setting.

3.1 Pairwise Probabilistic Matrix Factorization

When designing this offline probabilistic matrix factorization model, we pay special attention to its extendability to streaming settings. Following the settings in BPR, we restrict the ratings as binary, with $x_{ij} = 1$ indicating u_i has a positive attitude to item v_j and $x_{ij} = 0$ otherwise. Note that, this binary setting has been used widely in both the implicit feedback systems and explicit feedback systems, e.g., in terms of ratings. In implicit feedback systems, if the interaction (such as, click, browse, buy and so on) between u_i and v_j has been observed, $x_{ij} = 1$. For the systems with explicit feedbacks (i.e., the systems with ratings), there are also many existing work which transform the ratings into binary ratings representing the users' positive or negative feedback [17, 36]. For a rating in the scale of $1-5$, assigned by user u_i to item v_j, if this rating is larger than 3, these existing work assume that $x_{ij} = 1$, otherwise $x_{ij} = 0$. In this paper, we assume that all the positive ratings constitute $\mathcal{D}^+$ and the others constitute $\mathcal{D}^-$ (i.e., we assume that $\mathcal{D}^-$ contain both the observed data with negative feedback and unobserved data). The whole data set is defined as $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$.

Based on the obtained binary ratings, we propose a novel probabilistic matrix factorization model in which each user and item are assigned with a z-dimensional latent factor vector, denoted as $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ respectively. Θ is used to denote the parameter set in our model, i.e., $\Theta = \{\boldsymbol{u}_i, \boldsymbol{v}_j | u_i \in \mathcal{U}, v_j \in \mathcal{V}\}$. Inspired by BPR in [19], we assume that all x_{ij} are conditionally independent with each other given latent vectors of users and items. $f(i, j; \Theta)$ in our model is implemented as follows:

$$f(i, j; \Theta) = \sum_{z'=1}^{z'=z} u_{iz'} \times v_{jz'} \tag{1}$$

The goal of our model is to maximize the predicting ability of the existence of the positive feedback in $\mathcal{D}^+$. For one rating x_{ij} in $\mathcal{D}^+$, our model predicts its existence via a score function $f(i, j; \Theta)$ which represents the model's confidence that user u_i likes v_j given the parameters Θ. The conditional independence assumptions of our model allow the probability model to be written as follows:

$$P(\mathcal{D}|\Theta) = \prod_{i=1}^{I} \prod_{j=1}^{J} Ber(x_{ij} | \sigma(f(i, j; \Theta))) \tag{2}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, and $Ber(x|p)$ follows the Bernoulli distribution, defined as follows:

$$Ber(x|p) = \begin{cases} p, & x = 1 \\ 1 - p, & x = 0 \end{cases} \tag{3}$$

Inspired by the success of probabilistic matrix factorization [22], we introduce the Gaussian priors over the latent parameters Θ. Thus, the generative process of our model is as follows:

- For each $i \in (1, \ldots, I)$, generate $\boldsymbol{u}_i \sim \mathcal{N}(\boldsymbol{0}, \rho_U^2 \boldsymbol{I})$;
- For each $j \in (1, \ldots, J)$, generate $\boldsymbol{v}_j \sim \mathcal{N}(\boldsymbol{0}, \rho_V^2 \boldsymbol{I})$;
- For each user u_i and each item v_j, generate $x_{ij} \sim Ber(x_{ij}| < \boldsymbol{u}_i, \boldsymbol{v}_j >)$.

Based on the above probabilistic generative process, the posterior distribution of the latent vectors of users and items is formulated as in Equation 4, through a simple Bayesian inference. In this equation, $\boldsymbol{U}, \boldsymbol{V}$ are two matrices that are formed by $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ as their column vectors, respectively. Our goal of this model is to maximize the likelihood in Equation 4. Inspired by BPR optimization framework in [19], maximizing the likelihood in Equation 4 is transformed into minimizing the negative log likelihood in Equation 5, where the regularization parameters $\lambda_U = \rho_U^{-2}$ and $\lambda_V = \rho_V^{-2}$.

In the off-line settings, directly minimizing Equation 5 is computationally expensive as many data sets in recommender systems are sparse and thus the number of negative samples is square to the number of users or items. Many existing work [18, 19, 21] simplify the computation of Equation 5 by selecting several negative examples for each positive sample and transform the goal function into Equation 6. Specifically, they fix user u_i and sample some negative items v_{j_n} according to a sampling strategy, and treat u_i, v_{j_n} as the negative samples. There are many different sampling strategies, such as the naive idea that samples the items uniformly for a given positive sample, popularity-biased item sampling in [25], and the latest Generative Adversarial Nets (GAN) based item sampling in [28]. GAN based negative sampling has proven its superiority in [28].

$$P(\boldsymbol{U}, \boldsymbol{V} \mid \mathcal{D}, \rho_U^2 \boldsymbol{I}, \rho_V^2 \boldsymbol{I}) \propto P(\boldsymbol{X} \mid \boldsymbol{U}, \boldsymbol{V}) P(\boldsymbol{U} \mid \boldsymbol{0}, \rho_U^2 \boldsymbol{I}) P(\boldsymbol{V} \mid \boldsymbol{0}, \rho_V^2 \boldsymbol{I}) = \prod_{i=1}^{I} \prod_{j=1}^{J} Ber(x_{ij} \mid \sigma(f(i, j; \Theta))) \prod_{i=1}^{I} \mathcal{N}(\boldsymbol{u}_i \mid \boldsymbol{0}, \rho_U^2 \boldsymbol{I}) \prod_{j=1}^{J} \mathcal{N}(\boldsymbol{v}_j \mid \boldsymbol{0}, \rho_V^2 \boldsymbol{I}) \tag{4}$$

$$\mathcal{L} = -\sum_{x_{ij} \in \mathcal{D}^+} log\sigma(f(i, j; \Theta)) - \sum_{x_{ij} \in \mathcal{D}^-} log(1 - \sigma(f(i, j; \Theta))) + \frac{\lambda_U}{2} \sum_{i=1}^{I} \| \boldsymbol{u}_i \|^2 + \frac{\lambda_V}{2} \sum_{j=1}^{J} \| \boldsymbol{v}_j \|^2 \tag{5}$$

$$O = -\sum_{x_{ij} \in \mathcal{D}^+} \Big(log\sigma(f(i, j; \Theta)) + \sum_{n=1}^{N} log(1 - \sigma(f(x_{ij_n}; \Theta))) \Big) + \frac{\lambda_U}{2} \sum_{i=1}^{I} \| \boldsymbol{u}_i \|^2 + \frac{\lambda_V}{2} \sum_{j=1}^{J} \| \boldsymbol{v}_j \|^2 \tag{6}$$

However, it's very expensive to maintain a generator when applied to the streaming data with high velocity. Moreover, in streaming setting, the data available are far away from being complete. Thus, the statistics-based negative sampling strategies, like [25], widely used in traditional recommender systems with offline setting, are not reliable. Thus, to adapt to the streaming settings, we adopt the most simple but most efficient uniform sampling strategy.

We minimize Equation 6 with well known Stochastic Gradient Descent (SGD). Given a rating x_{ij} which can be either 1 or 0, the value of the loss function at this sample is defined as follows:

$$O(x_{ij}) = -\Big(x_{ij} log\sigma(f(i, j; \Theta)) + (1 - x_{ij}) log(1 - \sigma(f(i, j; \Theta))) \Big) \tag{7}$$

where we omit the regularization terms. Then, the gradients of $O(x_{ij})$ w.r.t. $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ are computed as follows:

$$\frac{\partial O(x_{ij})}{\partial \boldsymbol{u}_i} = (\sigma(f(i, j; \Theta)) - x_{ij}) \boldsymbol{v}_j \tag{8}$$

$$\frac{\partial O(x_{ij})}{\partial \boldsymbol{v}_j} = (\sigma(f(i, j; \Theta)) - x_{ij}) \boldsymbol{u}_i \tag{9}$$

Assume that the set of samples (including both positive and negative ones) associated with user u_i in a batch is defined as $\mathcal{D}_{u_i}$. Similarly, let $\mathcal{D}_{v_j}$ denote the set of examples associated with item v_j. The gradients of the objective function in Equation 6 w.r.t. $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ for a batch are updated as follows:

$$\frac{\partial O}{\partial \boldsymbol{u}_i} = \sum_{x_{ij} \in \mathcal{D}_{u_i}} \frac{\partial O(x_{ij})}{\partial \boldsymbol{u}_i} + \lambda_U \boldsymbol{u}_i \tag{10}$$

$$\frac{\partial O}{\partial \boldsymbol{v}_j} = \sum_{x_{ij} \in \mathcal{D}_{v_j}} \frac{\partial O(x_{ij})}{\partial \boldsymbol{v}_j} + \lambda_V \boldsymbol{v}_j \tag{11}$$

Input: Training set $\mathcal{D}$,
the dimension of embeddings z;
Output: The latent vectors Θ for users and items;

```
iter = 0;
while iter < m do
    Sample a positive example x_ij from D+;
    Sample N negative examples x_in from D- by fixing the user u_i;
    Update the gradients of the latent variables associated with users
      and items in the batch based on Equation 10 and 11;
    Update the latent variables by batch SGD;
    iter = iter + 1;
end
```

Algorithm 1: Offline Model Training

The algorithm of model training in the off-line setting is illustrated in Algorithm 1. In this algorithm, a positive sample and N sampled negative examples are treated as a batch.

3.2 Streaming Model

In this part, we extend our proposed offline model to the streaming settings. When dealing with streams of data arriving consistently, one usually wants to avoid the cost of retraining a model every time new data instances arrive. Thus, online updates are usually used [20]. However, the gain in processing time by this online learning approach has been proven coming at the cost of reduced prediction quality [4]. The major issue with this online updating approach in recommender systems is that they cannot maintain users' long-term interests, since the updates are based only on the most recent data instances and the model quickly "forgets" past observations. To keep the system aware of users' long-term interests, our proposed streaming model, SPMF, employs the reservoir, which has been widely used in solving the limited memory problem in streaming database management systems [4, 5, 15, 27], to keep a long term "memory" of history data instances. The goal of this reservoir is to keep an accurate "sketch" of history data instances, thus SPMF employs the technique of random sampling proposed in [27] to select the data maintained in the reservoir. It includes the t^{th} data instance in the stream with probability $|\mathcal{R}|/t$, where $\mathcal{R}$ is the set of data instances stored in the reservoir, and replaces uniformly at random an instance from the reservoir $\mathcal{R}$. The resulted reservoir has been shown as a random sample of the current dataset [27] and also been proven to be able to maintain users' long-term interests in [4].

Input: The current set of parameters $\Theta^t = \{\boldsymbol{U}, \boldsymbol{V}\}$,
the current reservoir $\mathcal{R} = \{s_1, s_2, \ldots, s_{|\mathcal{R}|}\}$,
a window of new data streams
$\mathcal{W} = \{e_{t+1}, e_{t+2}, \ldots, e_{t+|\mathcal{W}|}\}$;
Output: The updated latent vectors $\Theta^{t+|\mathcal{W}|}$ for users and items;

```
while next window of data have not arrived do
    Sample a data instance x_ij from R ∪ W;
    Sample N negative examples x_in from D_t^- by fixing the user u_i;
    Update the gradients of the latent variables associated with users
      and items in the batch based on Equation 10 and 11;
    Update the latent variables by batch SGD;
end
```

Algorithm 2: Online Model Training

As we argued before, capturing users' drifted interests over time and modeling new users or items are also very important for recommender systems [30, 34, 35]. All these patterns are contained in the latest generated data. However, the proposed sampling strategy of reservoir tends to overlook the recent data. To solve this problem, SPMF updates the model parameters Θ_t based on both the

new input data $\mathcal{W}$ and the maintained data in the reservoir $\mathcal{R}$. In the on-line setting, the goal of model updating is to maximize the model's predicting accuracy for both the newly generated positive data and the positive instances stored in the reservoir, inspired by Equation 2.

Note that, as we do not distinguish the observed negative data from the unobserved data, we assume that the data streams contain only the observed positive data and treat all the other data as negative data instances. Another thing we need to pay attention to in on-line setting is that we have limited time to update the model. We need to finish model updating before next window of data come. Under this assumption, the problem of updating the model in stream setting is presented in Problem 1.

Problem 1. ***(Model Updating in On-line Setting)*** *Given the set of parameters Θ^t until time t, the maintained reservoir $\mathcal{R} = \{s_1, s_2, \ldots, s_{|\mathcal{R}|}\}$, and a window of new data streams $\mathcal{W} = \{e_{t+1}, e_{t+2}, \ldots, e_{t+|\mathcal{W}|}\}$, the problem is updating the parameters Θ_t based on $\mathcal{W}$ and $\mathcal{R}$ before next window of new data streams arrive, with the goal to maximize the Equation 12.*

$$P(\mathcal{W}, \mathcal{R}|\Theta^{t+|\mathcal{W}|}_{\mathcal{W}\cup\mathcal{R}}) = \prod_{w=1}^{|\mathcal{W}|} Ber(e_{t+w}|\sigma(f(e_{t+w}; \Theta^{t+|\mathcal{W}|}_{\mathcal{W}\cup\mathcal{R}})))\times \prod_{r=1}^{|\mathcal{R}|} Ber(s_r|\sigma(f(s_r; \Theta^{t+|\mathcal{W}|}_{\mathcal{W}\cup\mathcal{R}}))) \tag{12}$$

where $\Theta^{t+|\mathcal{W}|}_{\mathcal{W}\cup\mathcal{R}}$ is the model parameters obtained at time $t+|\mathcal{W}|$ after updating based on $\mathcal{W}\cup\mathcal{R}$, and e_{t+w} and s_r are data instances in new stream window $\mathcal{W}$ and the maintained reservoir $\mathcal{R}$ respectively.

In Problem 1, both e_{t+w} and s_r are denoted as $\{u_i, v_j\}$, indicating u_i has a positive attitude to v_j. In this problem, we have a slight notation abuse for both e_{t+w} and s_r considering the convenience of representation. Let us take e_{t+w} for an example. We use $f(e_{t+w}; \Theta^{t+|\mathcal{W}|}_{\mathcal{W}\cup\mathcal{R}})$ to represent the score of the rating x_{ij} contained in the data tuple e_{t+w} and it is equal to $f(i, j; \Theta^{t+|\mathcal{W}|}_{\mathcal{W}\cup\mathcal{R}})$ denoted in Equation 1. Similarly, $Ber(e_{t+w}|\sigma(f(e_{t+w}; \Theta^{t+|\mathcal{W}|}_{\mathcal{W}})))$ has the same meaning of Equation 3.

A naive solution for Problem 1 is presented in Algorithm 2 which applies the batch based SGD on $\mathcal{W}\cup\mathcal{R}$ directly. Line 4 in Algorithm 2 aims at sampling a negative data instances for user u in on-line setting. $\mathcal{D}^-_t$ denotes the negative sample set on time t. Negative sampling over streaming data is an extreme challenging task because of its own nature [12]. The essential reason why online negative sampling is very challenging is that it is hard to tell which instance is truly negative as we can only store part of the observed positive data in the streaming setting. The current existing work [4, 12] about online negative sampling are all based on the assumption that the data instances outside the current data stream are negative. For example, in [12], May et. al draw the negative samples based on the empirical distribution over words in the stream instead of over a smoothed empirical overall distribution. In this paper, we also design our negative sampling method based on this assumption. Specifically, we treat all the examples outside $\mathcal{W}$ and $\mathcal{R}$ as candidate negative samples. Specifically, $\mathcal{D}^-_t = \mathcal{D} - \mathcal{W} - \mathcal{R}$ in Algorithm 2. For a user u and his/her activity set S_u observed in $\mathcal{W}\cup\mathcal{R}$, we adopt the efficient uniform random sampling method to generate negative items not contained in S_u, similar to the off-line setting. In recommender systems, it is well known that the data is very sparse with the sparsity generally higher than 99%. Thus, this uniform random sampling method has a very small chance to sample a very positive example in recommender systems.

The major challenge of Algorithm 2 is that the high-velocity of streaming data and limited CPU resources of recommender systems may lead to very limited iterations in updating the model. Limited iterations on all the training data tend to lead to poor performance. This problem is well known as system overload in many stream processing systems [13, 14, 16]. This overload can be caused by many factors, for example, the temporary bursts of shopping online on weekends.

In the presence of an abundant source of training examples arriving at a high speed, a natural way to reduce complexity of learning the recommendation model consists of wisely selecting a subset of training examples. What's worth noting is that for the online setting, we have limited time for sampling and optimization. A wise sampling strategy usually costs more time. Thus, we need to take the trade-off between the time cost and sampling quality into consideration in designing the sampling strategy.

In [20], Rendle et al. propose one possible method to deal with the overload problem in Algorithm 2 by retraining the feature vector for the new users only and keeping the vectors for all the other users fixed. The motivation for this algorithm is the assumption that the model Θ^t built from the history activities and the model Θ^t_S updated after the new data coming are mostly the same from a global perspective. But if a user is a new user, his/her features need to be learned from the new data. This method overlooks the potential interest drift for the existing users. It has been widely proven that users' preferences may drift over time [30, 34, 35]. A very obvious example is that the preferences of a female who just became a mother will change a lot compared with her previous interests. These changes for the existing users are also very important for recommender systems.

To deal with the overload problem in Algorithm 2, we propose an efficient Gaussian classification model. This classification model focuses on efficiently sampling instances from $\mathcal{W}\cup\mathcal{R}$ based on which SPMF conducts the model updating. The basic idea of this classification model is selecting data instances that are expected to change the existing model most. We rank the input instances according to the scores computed by $f(i, j; \Theta^t)$ with the current set of parameters Θ^t, and for each data instance x_{ij}, we get a rank $rank_{ij}$ in the ranked list for it. To reduce the time cost in sampling and leave more time for model updating, we assume that the ranks for all the data instances in the ranked list do not need to be updated during the model optimization process, inspired by the observation that the model Θ^t built from the history activities and the model $\Theta^{t+|\mathcal{W}|}$ updated after the new data coming are mostly the same from a global perspective [20]. Our intuition is that when sampling a data instance x_{ij}, the smaller the score $f(i, j; \Theta^t)$ is, the more informative and helpful the data instance is to correct the model. Thus, we propose to compute the weight of a data instance x_{ij} based on its rank, as follows:

$$w_{ij} = exp(\frac{rank_{ij}}{|\mathcal{W}\cup\mathcal{R}|}) \tag{13}$$

where w_{ij} is the weight of x_{ij}. We adopt the exponential function to compute the weight of a data instance according to its normalized

Input: same with the input in Algorithm 2;
Output: The updated latent vectors $\Theta^{t+|\mathcal{W}|}$ for users and items;

$\mathcal{R}' = \varnothing$;
foreach *data instance* $\{u_i, v_j\} \in \{\mathcal{R} \cup \mathcal{W}\}$ **do**
  if u_i *or* v_j *is a new user or a new item* **then**
    Generate $\boldsymbol{u}_i$ or $\boldsymbol{v}_j$ based on Gaussian prior distribution;
  end
  else
    Get $\boldsymbol{u}_i$ or $\boldsymbol{v}_j$ from Θ^t;
  end
  Compute the score $f(i, j; \Theta^t) = \boldsymbol{u}_i \cdot \boldsymbol{v}_j$;
  if *this is an instance in* $\mathcal{W}$ **then**
    Sample this instance with probability $|\mathcal{R}|/(t + i')$ where $1 \le i' \le |\mathcal{W}|$ to a temporary set $\mathcal{R}'$;
  end
end
Rank data instances in $\{\mathcal{R} \cup \mathcal{W}\}$ decreasingly according to $f(i, j; \Theta^t)$ to get a rank $rank(x_{ij})$ for each x_{ij};
Compute w_{ij} for each x_{ij} in $\{\mathcal{R} \cup \mathcal{W}\}$ using Equation 13;
Compute $P(x_{ij})$ for each x_{ij} in $\{\mathcal{R} \cup \mathcal{W}\}$ using Equation 14;
while *next window of data have not arrived* **do**
  Sample a data instance x_{ij} from $\mathcal{R} \cup \mathcal{W}$ according to $P(x_{ij})$;
  Sample N negative examples x_{in} from $\mathcal{D}_t^- = \mathcal{D} - \mathcal{R} - \mathcal{W}$ by fixing the user u_i;
  Update the gradients of the latent variables associated with users and items in the batch based on Equation 10 and 11;
  Update the latent variables by batch SGD;
end
Replace the data in $\mathcal{R}$ randomly with the data in $\mathcal{R}'$;

Algorithm 3: Improved Online Model Training

rank following [18]. This ranking-based weighting scheme favors the data instances at a lower rank much more than the ones at the top. Finally, the sampling probability for a data instance is defined as follows:

$$P(x_{ij}) = \frac{w_{ij}}{\sum_{x_{ij} \in \mathcal{W} \cup \mathcal{R}} w_{ij}} \tag{14}$$

The details of the online algorithm with the proposed Gaussian classification model are demonstrated in Algorithm 3.

With this Gaussian classification model, the proposed streaming model is able to deal with new users, new items and interest drift problems.

For a data instance (u_i, v_j), $f(i, j; \Theta^t)$ represents the current system's predicting ability on the fact that user u_i has preference over item v_j. This predicting ability is learned from the history activities containing the similar patterns. For example, for a user u on a movie platform, if the system knows that u has a preference over scientific movies from his/her history activities, then the system tends to give a large score for a new instance in the stream that u likes movie "Harry Potter". Intuitively, the system will gain less knowledge from an instance with a larger score as the system has been trained over many instances with such similar patterns. Thus, in Algorithm 3, for an instance x_{ij}, a smaller score $f(i, j; \Theta^t)$ will lead to larger probability that x_{ij} is sampled as the system will gain more knowledge from this kind of instances. A small score $f(i, j; \Theta^t)$ indicates that there is limited history activities indicating that user u_i is fond of item v_j. For new users (Line 3 and 4), it is obvious that the system has no prior knowledge at all about their preferences and the instances related to them will get higher probabilities being sampled. Similarly, for the items (Line 3 and 4), the system also has no prior knowledge about their properties and cannot predict the users' preference over them. Interest drift problem about existing users refer to the phenomenon that the users' interest drift with time (e.g., a female who bought a lot of female clothes and makeups, prefers purchasing baby products recently as she just became a mother). It's obvious that Algorithm 3 also prefers to sample the data instances containing drifted interests for existing users as these instances contain very different behavior patterns compared with the history patterns captured by the obtained model parameters.

4 EVALUATION

In this section, we first describe the settings of experiments and then demonstrate the experimental results.

4.1 Experimental Settings

4.1.1 Data Sets. To assess the effectiveness of the proposed model, we evaluate the methods on three different real-world datasets and their detailed descriptions are as follows:

MovieLens: This is a well known recommendation data set [6] which contains 1,000,209 ratings of approximately 3,900 movies made by 6,040 users who joined MovieLens in 2000.

Netflix: This dataset was constructed to support participants in the Netflix Prize[2] and it contains over 100 million ratings from 480 thousand customers over 17 thousand movies. The data were collected between October, 1998 and December, 2005.

The above two datasets both are very classic movie data sets with explicit feedbacks, which have been widely used in both classic recommender systems and streaming recommender systems [1, 20]. To further evaluate our methods in data sets with implicit feedbacks, we also conducted our experiments on Taobao.

Taobao: Taobao is one of the most famous Chinese websites for online shopping, which is similar to eBay and Amazon. This data set[3] contains user purchasing history on clothing from June, 2014 to June, 2015. It contains 13,731,675 records on 462,379 clothes made by 1,103,702 customers. Other than the interaction information, the original dataset also contains other information (e.g., texts and videos). But in this paper, we only concentrate on the interaction information.

4.1.2 Comparative Methods. We compare SPMF with several representative recommender systems including:

sRec: sRec is a latest probabilistic streaming recommendation model [1]. It updates parameters depending on all the incoming data and the posterior probabilities at the previous moment.

RMFX: Diaz-Aviles et. al consider collaborative filtering as an online ranking problem and present *Stream Ranking Matrix Factorization*, called as RMFX, which uses a pairwise approach to matrix factorization in the streaming setting [4]. RMFX uses a fixed size reservoir to store the data instances used to update the model.

[2] http://www.netflixprize.com
[3] https://tianchi.aliyun.com/datalab/dataSet.htm?id=13

RKMF: Regularized kernel matrix factorization (RKMF) is designed by Rendle et. al in [20]. A flexible online-update algorithm is developed for RKMF model which updates the model on selected data instances. The selection criteria in this model assigns larger probabilities to new users and new items.

WRMF: Weighted Regularized Matrix Factorization (WRMF) is state-of-the-art matrix factorization model for item prediction introduced by Hu et al. [8]. Their method is designed for implicit feedback datasets. We implement this model in pairwise way [19]. This model is not designed for streaming data and during the evaluation, we assume that the whole stream is stored and available for the training of this model. It is expected that the performance of this offline method, with full access to the history data and unlimited time to update the model, will set up an upper bound for the on-line approaches.

In summary, WRMF is batch based and it has full access to all the history activities. Both RMFX and SPMF adopt a fixed-size reservoir with same size to store the long-term data. The other two methods: sRec and RKMF, completely learn from the data streams without storing any history records.

To further validate the benefits brought by exploiting the proposed Gaussian classification model, maintaining the reservoir and updating the model over the new input data respectively, we implement three variant versions of our SPMF model.

SPMF-S1 is the simplified version of SPMF without the proposed Gaussian classification model and it samples positive data instances from $\mathcal{W} \cup \mathcal{R}$ randomly.

SPMF-S2 is another simplified version without maintaining the reservoir $\mathcal{R}$. It updates the SPMF based on the new input $\mathcal{W}$ only.

SPMF-S3 is the other simplified version which does not exploit the data in $\mathcal{W}$. It updates the SPMF only based on $\mathcal{R}$, similar to RMFX.

4.1.3 Evaluation Method. We simulate the streaming recommendation by following [1]. More specifically, we first order data instances in the whole data set $\mathcal{D}$ based on their temporary information and then divide all data into halves. The first half is called as "base training set" , denoted as $\mathcal{D}^{train}$, and is used for parameter estimations; and the remaining half is the "candidate set". Here, the base training set is considered as the historical data pool while the candidate set mimics the on-line streaming inputs. All the comparison methods are trained over the base training set to determine the best hyper parameters related to the model by cross validation. Specifically, we use the grid search algorithm to obtain the optimal hyper parameter setup. For the candidate set, we further divide it into five slices sequentially. Each slice is treated as a test set $\mathcal{D}^{test}$. Our task is to predict user-item interactions in each test set. All the data prior to the target test set in the candidate set are used for on-line learning and these data are called as "online training set". Specifically, if we are going to predict user-item interactions in the i^{th} test set $\mathcal{D}_i^{test}$, all the sets $\mathcal{D}_j^{test}$, where $j < i$, are used for on-line learning and each one of them is treated as an input window $\mathcal{W}$. It is worth mentioning that, for users who have no activities before $\mathcal{D}_i^{test}$, we recommend the most popular items for all comparison methods.

To evaluate the performance of all streaming methods in same overload settings, we assume that online input data instances all arrive in a same rate for all the streaming methods. In other words, given a specific window in our evaluation, all the streaming methods have same length of time to update their models based on current window before next window of data come. For all these methods, once the next window of data arrive, we stop their model updating over current window of data instances.

The evaluation methodology and measurement $Hits@n$, which has been widely applied in [29, 30], is adopted. Specifically, for each case (u_i, v_j) in the test set: 1) we compute the score for item v_j with the score function $f(i, j; \Theta)$ and the scores for items that u_i has not shown preference to previously; 2) we form a ranked list by ordering all of these items according to their ranking scores. Let $rank$ denote the position of the item v_j within this list. The best result corresponds to the case where v_j precedes all the other items (that is, $rank = 1$); and 3) we form a top-n recommendation list by picking the n top ranked items from the list. If $rank \leq n$, we have a hit (i.e., the ground truth item v_j is recommended to the user). Otherwise, we have a miss. The probability of a hit increases with the increasing value of n and we always have a hit when $n = J$. The computation of $Hits@n$ proceeds as follows:

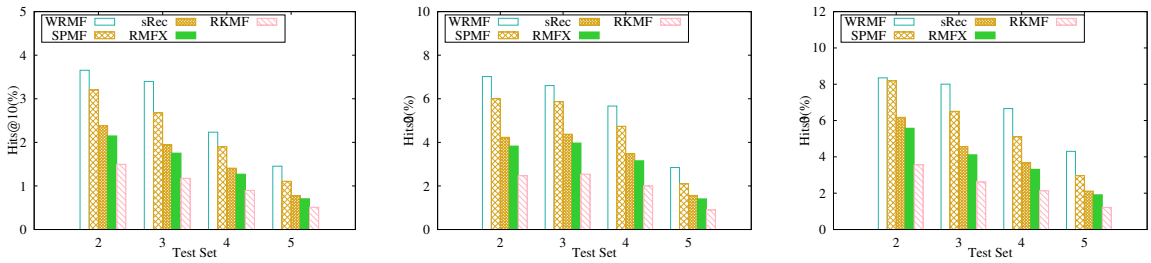
$$Hits@n = \frac{\#hit@n}{|\mathcal{D}^{test}|} \tag{15}$$

where $\#hit@n$ is the number of hits in $\mathcal{D}^{test}$.

4.2 Recommendation Effectiveness

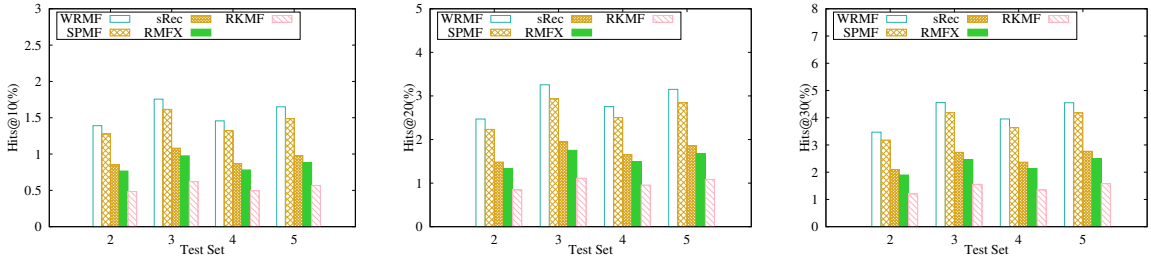
The size of reservoir is set to $|\mathcal{D}|/20$ for both RMFX and SPMF. The time for updating models based on current window of data are same for all the streaming methods, which are set to the time used to update RMFX for $\frac{|\mathcal{W}|}{2}$ iterations on each $\mathcal{R} \cup \mathcal{W}$. Note that, we did not use a fixed time (e.g., 5 seconds) as the updating time as the hardware condition has a big influence on the time cost and we cannot make too much sense about the exact overload situation from a fixed time. To avoid the possible bias to our own model, we use the updating iterations of a comparison method instead of our own method as an indicator of overload. We tuned the parameters in all comparison methods and picked up the best results as the parameter settings. Under these well-tuned parameter settings, we ran all the methods 30 times and the average results are presented in this part. Figures 2, 3 and 4 report the comparisons with the other state-of-art methods in terms of $Hits@10$, $Hits@20$ and $Hits@30$ on MovieLens, Netflix and Taobao, respectively. Figure 5 demonstrate the comparisons with other three variant versions respectively on MovieLens data.

Figures 2, 3 and 4 show that the recommendation methods have significant performance disparity in terms of the top-n hit ratios. WRMF performs best among all the comparison methods on both Movielens and Netflix, which is consistent with the expectation that the offline method WRMF sets an upper bound for the online approaches. On all three datasets, compared with WRMF, our proposed SPMF model achieves very competitive results even under the limitation of memory and training time. Let us take $Hits@10$ on $\mathcal{D}_5^{test}$ for example. Compared with WRMF, SPMF achieves $1.200/1.447 = 82.93\%$, $1.482/1.640 = 90.37\%$ and $0.315/0.359 = 87.74\%$ hit ratios on MovieLens, Netflix and Taobao respectively. For $\mathcal{D}_i^{test}$, WRMF has access to the data $\mathcal{D}^{train} \cup \mathcal{D}_1^{test} \cup \ldots \cup \mathcal{D}_{i-1}^{test}$ while SPMF only has the access to $\mathcal{R}$, with size less than $|\mathcal{D}^{train}|/10$. Moreover, WRMF has unlimited time to train the model (i.e., about



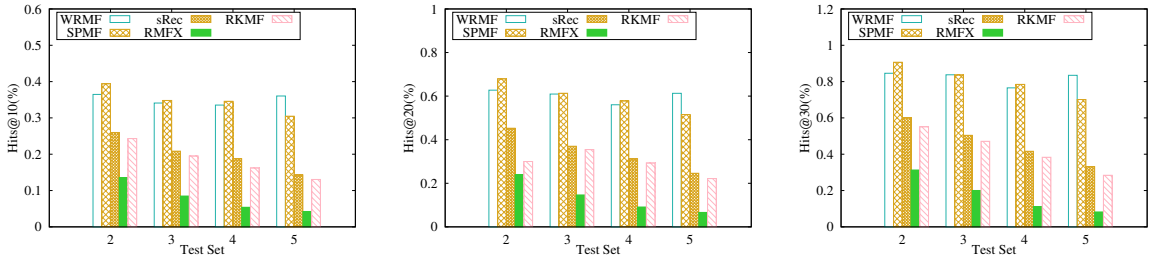
(a) $Hits@10$ (b) $Hits@20$ (c) $Hits@30$

Figure 2: Performance on Movielens



(a) $Hits@10$ (b) $Hits@20$ (c) $Hits@30$

Figure 3: Performance on Netflix



(a) $Hits@10$ (b) $Hits@20$ (c) $Hits@30$

Figure 4: Performance on Taobao

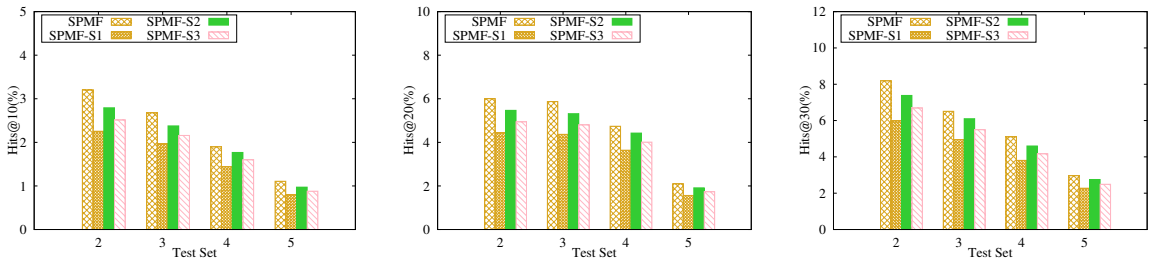
$25 \times |\mathcal{D}^{train} \cup \mathcal{D}_1^{test} \cup \ldots \cup \mathcal{D}_{i-1}^{test}|$ iterations) while SPMF has very limited updating to the model (i.e., $\frac{|\mathcal{D}_i^{test}|}{2}$ iterations).

Several additional observations are made from the results in Figures 2, 3 and 4: 1) Clearly, our proposed SPMF model outperforms other competitor streaming models significantly, demonstrating the advantages of SPMF over other streaming methods; 2) All the comparison methods present a best performance on MovieLens. This is because that Movielens has the densest data and smallest number of items. For a test case (u_i, v_j), the methods have larger chance to put v_j on the top-n recommendations if there are less candidate items; 3) All the methods demonstrate a significant drop of performance on Taobao. One obvious reason is that Taobao has a much sparser data compared with the other two datasets. Another possible reason is that Taobao has much larger numbers of test cases containing new items. For example, there are about 45.86% test cases in $\mathcal{D}_2^{test}$ on Taobao containing items that have not appeared in $\mathcal{D}^{train}$ while there are less than 1% such test cases in other two datasets. 4) SPMF demonstrates a better performance compared with WRMF in many cases on Taobao. As we know, WRMF is popularity based and is biased to the popular users or items. However, as we analyzed before, the test cases in Taobao contain many new items. This observation demonstrates the advantage of SPMF in dealing with new items compared with WRMF.

To validate the benefits brought by exploiting the proposed Gaussian classification model, maintaining the reservoir and updating the model over the new input data respectively, we also compare SPMF with its three variant versions. Due to the space limitation, we only present the comparison result on MovieLens data set. Figure 5 present the comparison results with the proposed three variant versions. From the results, we observe that SPMF consistently outperforms the three variant versions, indicating the benefits brought by each factor, respectively. For instance, the performance gap between SPMF and SPMF-S1 validates the effectiveness of adopting the Gaussian classification model in sampling data instances to alleviate the overload problem in streaming settings. We also see that SPMF-S2 performs second best on this dataset. This demonstrates that the advantage of wisely sampling data instances and updating model based on new input data outweigh the advantage of using reservoir to keep the long term memory on this dataset.

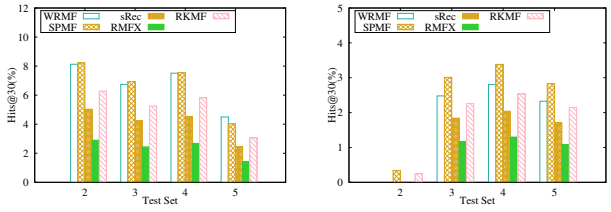
4.3 Related Studies

4.3.1 Test for New Users and New Items. As most test cases in Taobao contain new items (e.g., 75.08% test cases in $\mathcal{D}_5^{test}$ on



(a) $Hits@10$ (b) $Hits@20$ (c) $Hits@30$

Figure 5: Evaluation of Factors on Movielens dataset



(a) newUsers (b) newItems

Figure 6: Performance for New Users and New Items

Taobao containing items), the results in Figure 4 are already very close to the results of recommendation for new items and they already show that our method are effective in recommendation for new items in Taobao. Thus, in this part, we choose a dataset (i.e., Movielens) from the other two datasets as a further demonstration of recommendation for new users and new items. We tested the performance of the methods in test cases containing new users who have no history records in $\mathcal{D}^{train}$ and test cases containing new items which have no related records in $\mathcal{D}^{train}$, respectively. Due to the space constraint, we only present $Hits@30$ for all the methods in Figure 6.

Figure 6 shows that SPMF performs best in recommendation for new users and new items even compared with the batch based WRMF, which is consistent with the results on Taobao. The reason is that WRMF is popularity based and is biased to the popular users or items, while Gaussian classification model in SPMF prefers data instances containing new users or new items. Another observation is that RKMF performs best among the other streaming methods. In overload setting, similar to SPMF, RKMF also prefers to train its model based on the data related to less popular users or less popular items. What's worth noting is that RMFX performs worst in Figure 6, which is consistent with our previous argument that RMFX tends to overlook new users or new items.

4.3.2 Test for Overload Problem. We proposed a Gaussian classification model to reduce the number of iterations needed in updating the model, so that the model can converge in a short time. In this way, SPMF is able to deal with the overload problem by reducing the update time in online setting. In this section, we tested the performance of SPMF with varying number of iterations allowed. A smaller number of iterations indicates a heavier work load in stream setting. The results on $\mathcal{D}_2^{test}$ of Movielens are in Table 1.

From the results, we observe that as the number of iterations increases, the $Hits@n$ values of SPMF first increase rapidly, and then the values become steady. The most significant increases occur between $\frac{|\mathcal{W}|}{16}$ and $\frac{|\mathcal{W}|}{8}$. The increases of iterations after $\frac{|\mathcal{W}|}{2}$ almost do not improve the recommendation performance any more. These observations indicate that SPMF is able to converge very fast in online setting. In other words, SPMF is able to complete model updating in a very limited time and thus can deal with the overload problem in streaming recommender systems.

n	Number of Iterations					
	$\frac{\lvert\mathcal{W}\rvert}{16}$	$\frac{\lvert\mathcal{W}\rvert}{8}$	$\frac{\lvert\mathcal{W}\rvert}{4}$	$\frac{\lvert\mathcal{W}\rvert}{2}$	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{W}\rvert \times 2$
5	0.0123	0.0150	0.0163	0.0171	0.0173	0.0173
10	0.0237	0.0286	0.0306	0.0321	0.0324	0.0325
15	0.0338	0.0417	0.0437	0.0456	0.0460	0.0460
20	0.0435	0.0535	0.0573	0.0601	0.0609	0.0608
25	0.0519	0.0640	0.0683	0.0708	0.0717	0.0718
30	0.0601	0.0739	0.0793	0.0818	0.0828	0.0828

Table 1: $Hits@n$ of SPMF under Varying Loads

5 RELATED WORK

There are two major types of streaming recommender systems. One type aims at designing an online learning scheme for model based systems while the other type focuses on supplying real-time recommendation for memory based systems.

Memory-based algorithms in recommender systems make recommendations for users based on their history activities. Typically, the prediction is calculated as a weighted average of the ratings or implicit feedback given by other users where the weight is proportional to the "similarity" between users. There are two major problems in applying memory-based algorithms to streaming data. The first problem is that it requires an offline phase, in which the similarity between every pair of items or users is computed, and the update of a single rating may require the re-computation of the similarities between users or items [9, 23]. The other problem is that one cannot assume that the whole history activities can be stored in main memory under streaming settings [9, 23]. Min-hash technique has been proposed to perform online recommendation in [23]. The main idea of this application is that the similarity between users can be approximately computed by tracking the relevant users for each item in a min-hash index. Min-hash index is implemented with the use of hash functions. In this way, they reduce the size of data in memory by storing the hash functions instead of use-item interaction histories. Huang et al. analyze the online recommendation problem from a wide range of applications using the considerable computation ability of Storm, together with a data access component and a specially developed data storage component in [9].

In contrast to the memory-based algorithms, model-based algorithms try to model the users based on their history activities

and then use these well-trained models to make recommendations on unseen items. There are a variety of research work studying the online learning of model-based recommendation algorithms [1, 3, 4, 7, 20, 30]. The common problem focused on in these work is how to update the model efficiently in streaming setting. Beyond the efficient updating problem, Chang et al. focus on solving the user drift problem by designing a continuous Markov process for each time-varying user/item topic in [1]. He and Devooght et al. [3, 7] focus on improving the online recommendation with implicit feedback and they pay more attention to the negative sampling in streaming settings. Rendle et al. study how to update the matrix factorization model to adapt to the new users and items in the streams in [20]. Diaz-Aviles et al. assume that the system cannot deal with all the input data in streaming setting and they update the model only based on the sampled data maintained in a well-designed reservoir in [4].

Our proposed streaming model is built on model based recommender systems. To our knowledge, we are the first one to solve user interest drift, new users/items and system overload problem in one single framework.

6 CONCLUSION

In this paper, we proposed a stream-centered probabilistic matrix factorization model called as SPMF to effectively overcome the challenges arising from streaming recommendations. Specifically, we first design a novel probabilistic matrix factorization model under the guide of flexible extendability to streaming setting. Then, we design effective schemes to extend this model to the online setting. To maintain users' long-term interests, SPMF adopted a fixed-size reservoir to keep an accurate sketch of the whole data set. Other than maintained samples in the reservoir, SPMF also used the new observations, which contain users' drifted interests and data for new users and items, in data streams to update the model. To deal with the overload problem, a novel and efficient Gaussian classification model is designed to sample a subset of data instances wisely from the input data. The basic idea of this Gaussian classification model is that the data instances at a lower rank should have a higher probability to be sampled, as this kind of data are more informative and helpful to correct the model. We conducted extensive experiments on three real datasets and the experimental results revealed the advantages of the SPMF model in streaming settings.

7 ACKNOWLEDGMENT

This work was supported by ARC Discovery Early Career Researcher Award (DE160100308), ARC Discovery Project (DP170103954) and New Staff Research Grant of The University of Queensland (613134). It was also partially supported by National Natural Science Foundation of China (61572335).

REFERENCES

[1] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A. Hasegawa-Johnson, and Thomas S. Huang. 2017. Streaming Recommender Systems. In *WWW*. 381–389.
[2] Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. 2013. TeRec: A Temporal Recommender System over Tweet Stream. *PVLDB* 6, 12 (2013), 1254–1257.
[3] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic Matrix Factorization with Priors on Unknown Values. In *SIGKDD*. 189–198.
[4] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. Real-ime Top-n Recommendation in Social Streams. In *Recsys*. 59–66.
[5] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted Random Sampling with A Reservoir. *Inf. Process. Lett.* 97, 5 (2006), 181–185.
[6] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *TiiS* 5, 4 (2016), 19:1–19:19.
[7] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. In *SIGIR*. 549–558.
[8] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *ICDM*. 263–272.
[9] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. 2015. TencentRec: Real-time Stream Recommendation in Practice. In *SIGMOD*. 227–238.
[10] Ling Liu and M. Tamer Özsu (Eds.). 2009. *Encyclopedia of Database Systems*. Springer US.
[11] Michael Mathioudakis and Nick Koudas. 2010. TwitterMonitor: Trend Detection Over the Twitter Stream. In *SIGMOD*. 1155–1158.
[12] Chandler May, Kevin Duh, Benjamin Van Durme, and Ashwin Lall. 2017. Streaming Word Embeddings with the Space-Saving Algorithm. *CoRR* abs/1704.07463 (2017).
[13] Alexandru Moga, Irina Botan, and Nesime Tatbul. 2011. UpStream: Storage-centric Load Management for Streaming Applications with Update Semantics. *VLDB J.* 20, 6 (2011), 867–892.
[14] Barzan Mozafari and Carlo Zaniolo. 2010. Optimal Load Shedding with Aggregates and Mining Queries. In *ICDE*. 76–88.
[15] Byung-Hoon Park, George Ostrouchov, Nagiza F. Samatova, and Al Geist. 2004. Reservoir-Based Random Sampling with Replacement from Data Stream. In *SIAM*. 492–496.
[16] Thao N. Pham, Panos K. Chrysanthis, and Alexandros Labrinidis. 2016. Avoiding Class Warfare: Managing Continuous Queries with Differentiated Classes of Service. *VLDB J.* 25, 2 (2016), 197–221.
[17] Al Mamunur Rashid, George Karypis, and John Riedl. 2008. Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach. *SIGKDD* 10, 2 (2008), 90–100.
[18] Steffen Rendle and Christoph Freudenthaler. 2014. Improving Pairwise Learning for Item Recommendation From Implicit Feedback. In *WSDM*. 273–282.
[19] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.
[20] Steffen Rendle and Lars Schmidt-Thieme. 2008. Online-updating Regularized Kernel Matrix Factorization Models for Large-scale Recommender Systems. In *RecSys*. 251–258.
[21] Steffen Rendle and Lars Schmidt-Thieme. 2010. Pairwise Interaction Tensor Factorization for Personalized Tag Recommendation. In *WSDM*. 81–90.
[22] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *NIPS*. 1257–1264.
[23] Karthik Subbian, Charu C. Aggarwal, and Kshiteesh Hegde. 2016. Recommendations for Streaming Data. In *CIKM*. 2185–2190.
[24] Panagiotis Symeonidis and Andreas Zioupos. 2016. *Matrix and Tensor Factorization Techniques for Recommender Systems*. Springer.
[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077.
[26] Nesime Tatbul, Ugur Çetintemel, and Stanley B. Zdonik. 2007. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. In *VLDB*. 159–170.
[27] Jeffrey Scott Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (1985), 37–57.
[28] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *SIGIR*. 515–524.
[29] Weiqing Wang, Hongzhi Yin, Ling Chen, Yizhou Sun, Shazia Wasim Sadiq, and Xiaofang Zhou. 2015. Geo-SAGE: A Geographical Sparse Additive Generative Model for Spatial Item Recommendation. In *SIGKDD*. 1255–1264.
[30] Weiqing Wang, Hongzhi Yin, Ling Chen, Yizhou Sun, Shazia Wasim Sadiq, and Xiaofang Zhou. 2017. ST-SAGE: A Spatial-Temporal Sparse Additive Generative Model for Spatial Item Recommendation. *TIST* 8, 3 (2017), 48:1–48:25.
[31] Hongzhi Yin, Hongxu Chen, Xiaoshuai Sun, Hao Wang, Yang Wang, and Quoc Viet Hung Nguyen. 2017. SPTF: A Scalable Probabilistic Tensor Factorization Model for Semantic-Aware Behavior Prediction. In *ICDM*. 585–594.
[32] Hongzhi Yin, Bin Cui, Ling Chen, Zhiting Hu, and Xiaofang Zhou. 2015. Dynamic User Modeling in Social Media Systems. *TOIS* 33, 3 (2015), 10:1–10:44.
[33] Hongzhi Yin, Bin Cui, Xiaofang Zhou, Weiqing Wang, Zi Huang, and Shazia W. Sadiq. 2016. Joint Modeling of User Check-in Behaviors for Real-time Point-of-Interest Recommendation. *TOIS* 35, 2 (2016), 11:1–11:44.
[34] Hongzhi Yin, Weiqing Wang, Hao Wang, Ling Chen, and Xiaofang Zhou. 2017. Spatial-Aware Hierarchical Collaborative Deep Learning for POI Recommendation. *TKDE* 29, 11 (2017), 2537–2551.
[35] Hongzhi Yin, Xiaofang Zhou, Bin Cui, Hao Wang, Kai Zheng, and Nguyen Quoc Viet Hung. 2016. Adapting to User Interest Drift for POI Recommendation. *TKDE* 28, 10 (2016), 2566–2581.
[36] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. 2011. Functional Matrix Factorizations for Cold-start Recommendation. In *SIGIR*. 315–324.