Training Streaming Factorization Machines with Alternating Least Squares

Xueyu Mao*
xmao@cs.utexas.edu
The University of Texas at Austin
Austin, TX

Saayan Mitra
smitra@adobe.com
Adobe Research
San Jose, CA

Sheng Li*
sheng.li@uga.edu
University of Georgia
Athens, GA

ABSTRACT

Factorization Machines (FM) have been widely applied in industrial applications for recommendations. Traditionally FM models are trained in batch mode, which entails training the model with large datasets every few hours or days. Such training procedure cannot capture the trends evolving in real time with large volume of streaming data. In this paper, we propose an online training scheme for FM with the alternating least squares (ALS) technique, which has comparable performance with existing batch training algorithms. We incorporate an online update mechanism to the model parameters at the cost of storing a small cache. The mechanism also stabilizes the training error more than a traditional online training technique like stochastic gradient descent (SGD) as data points come in, which is crucial for real-time applications. Experiments on large scale datasets validate the efficiency and robustness of our method.

CCS CONCEPTS

• **Computing methodologies** → **Factorization methods**.

KEYWORDS

Factorization Machines, Online Learning, Alternating Least Squares

ACM Reference Format:
Xueyu Mao*, Saayan Mitra, and Sheng Li*. 2019. Training Streaming Factorization Machines with Alternating Least Squares. In *Proceedings of 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Context aware recommender systems [13] are increasingly employed to provide personalized recommendations to users or to predict the reactions of users to a given subject or situation. Many such recommender systems employ factorization machines (FM) [10] which combines the advantages of both a regression model and matrix factorization. In recent years, many variants of FMs have been proposed. Field-aware FM leverages the field information associated with each feature, and it usually outperforms the original FM in many real-world evaluations [6]. Robust FM utilizes the robust optimization framework, and achieves stable performance under interval uncertainty [9]. FMs are typically trained in batch mode, where the prediction model is trained on large datasets every few hours or days. In these systems, the models can become stale and fail to capture trends as they are evolving in real-time. This can be a particular problem in use cases such as sports or news, where it is important to react quickly to shifting trends to make relevant recommendations based on the current viewing patterns observed over large number of users. Improving the system performance of real-time stream recommendation has become an active research topic from the database perspective, but mainly focuses on the item-based collaborative filtering [5]. Another disadvantage of batch mode training is the significant processing time and memory storage required to deal with large scale datasets. To this end, it would be desirable to have a system that is capable of training and updating the FM in an efficient, incremental, and continuous fashion, based on each newly received data point from a stream of data, without the need to perform extensive re-calculations based on large volumes of previously received historical data. Such a system can enable the FM to capture dynamic information from streaming data sources in a real-time fashion by performing a single processing pass on each data point as soon as it arrives.

Existing FM training methods include: 1) stochastic gradient descent [11], which can be easily implemented in an online setting, however, the learning rate needs to be tuned and may need multiple epochs to converge; 2) tuning-free coordinate descent, which includes (a) alternating least squares (ALS) [12], (b) efficient training for FM using ANOVA kernel [1] which needs cache scaling with number of data points in the dataset; 3) MCMC [11], which is usually time consuming and it is usually hard to pick a consistently good model as it uses ensembles for prediction. [7] proposed a provable one-pass algorithm for generalized FMs and rank-one matrix sensing using alternating minimization, however, it has strong assumptions that the data are sampled from random Gaussian vectors and is computationally expensive. In [2], the authors propose a fast learning method when FM is used for binary classification, and the objective function is modified to make it multi-block convex. Recently, some methods that combine the merits of deep learning and FM have been proposed, such as DeepFM [3], neural factorization machines [4] and attentional factorization machines [15]. Like many other deep learning approaches, these FM based deep learning methods mainly use gradient descent for model training. In this work, we propose an algorithm based on ALS for fast online update of FM model with streaming data. We give the detailed

* This work was done while the authors were at Adobe Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGIR'19, July 21-25, 2019, Paris, France
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

derivation of the algorithm and validate on large scale simulated and real industrial data.

2 PRELIMINARIES

2.1 Model

We focus on degree-2 FM in this work, which was proposed by Rendle [10]. The model equation is given by:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j,$$

where, $\mathbf{x} \in \mathbb{R}^d$ is a feature vector representation of a data point. $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^d$, $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_d]^T \in \mathbb{R}^{d \times k}$ are model parameters, d is the dimension of the input feature vector, and $k \in \mathbb{Z}^+$ is the rank of the factorization. $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ is the inner product of two latent factors, which models the interaction between the i-th and j-th variable.

2.2 ALS update for FM

As pointed in [12], the objective function of the FM model is linear in each parameter (w_0, w_i, V_{ij}) given the others are fixed. Let $\theta \in \Theta$ be a parameter, then the FM model can be written as: $\hat{y}(\mathbf{x}|\theta) = g_{(\theta)}(\mathbf{x}) + \theta h_{(\theta)}(\mathbf{x})$. For example, if $\theta = w_0$, we have $h_{(w_0)}(\mathbf{x}) = 1$ and $g_{(w_0)}(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$. When we use square loss $L = \sum_{(\mathbf{x},y)\in S} (\hat{y}(\mathbf{x}) - y)^2 + \sum_{\theta} \lambda_{(\theta)} \theta^2$, this linear function has a closed form optimal solution:

$$\theta = -\frac{\sum_{(\mathbf{x},y)\in S} (g_{(\theta)}(\mathbf{x}) - y) h_{(\theta)}(\mathbf{x})}{\sum_{(\mathbf{x},y)\in S} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}}, \tag{1}$$

where S is the set of all the data points and $\lambda_{(\theta)}$ is the regularization parameter for θ. [12] proposed an alternating least squares based algorithm for FM training using Equation (1), which forms the basis for our method.

3 ONLINE ALS UPDATE FOR FM

We consider a streaming setting where data points come one by one: $(\mathbf{x}|_{t_1}, y|_{t_1}), (\mathbf{x}|_{t_2}, y|_{t_2}), \cdots$. Let the model parameters at time t_i be $w_0|_{t_i}$, $\mathbf{w}|_{t_i}$, $\mathbf{V}|_{t_i}$. Our premise is that the data points can only be used once for training, and then thrown away. We use $S|_{t_i}$ to denote all the data points that has been seen at time t_i.

In an online setting, for normal ALS training, in each epoch, at step i (time t_i), it is ideal to update the model using all the data points that have been seen:

$$\begin{aligned}\theta|_{t_i} &= -\frac{\sum_{(\mathbf{x},y)\in S|_{t_i}} (g_{(\theta)}(\mathbf{x}) - y) h_{(\theta)}(\mathbf{x})}{\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}} \\ &= -\frac{\sum_{(\mathbf{x},y)\in S|_{t_i}} (e(\mathbf{x},y) - \theta|_{t_i} h_{(\theta)}(\mathbf{x})) h_{(\theta)}(\mathbf{x})}{\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}},\end{aligned} \tag{2}$$

where $e(\mathbf{x}, y) = \hat{y}(\mathbf{x}) - y$, which is the error term for the prediction of data point $\mathbf{x}$. Note that at step t_{i-1}, we also have:

$$\theta|_{t_{i-1}} = -\frac{\sum_{(\mathbf{x},y)\in S|_{t_{i-1}}} (e(\mathbf{x},y) - \theta|_{t_{i-1}} h_{(\theta)}(\mathbf{x})) h_{(\theta)}(\mathbf{x})}{\sum_{(\mathbf{x},y)\in S|_{t_{i-1}}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}}.$$

Also, it is easy to show that

$$\begin{aligned}&\sum_{(\mathbf{x},y)\in S|_{t_i}} (e(\mathbf{x},y) - \theta|_{t_i} h_{(\theta)}(\mathbf{x})) h_{(\theta)}(\mathbf{x}) \\ &= -\theta|_{t_{i-1}} \left(\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)} \right) + e(\mathbf{x}|_{t_i}, y|_{t_i}) h_{(\theta)}(\mathbf{x}|_{t_i}) \\ &\quad + \sum_{(\mathbf{x},y)\in S|_{t_i}} (\theta|_{t_{i-1}} h_{(\theta)}(\mathbf{x}) - \theta|_{t_i} h_{(\theta)}(\mathbf{x})).\end{aligned} \tag{3}$$

Now, combining Equations (2) and (3), we have,

$$\begin{aligned}\theta|_{t_i} =& \theta|_{t_{i-1}} - \frac{e(\mathbf{x}|_{t_i}, y|_{t_i}) h_{(\theta)}(\mathbf{x}|_{t_i})}{\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}} \\ &- \frac{\sum_{(\mathbf{x},y)\in S|_{t_i}} (\theta|_{t_{i-1}} h_{(\theta)}(\mathbf{x}) - \theta|_{t_i} h_{(\theta)}(\mathbf{x}))}{\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}} \\ \approx& \theta|_{t_{i-1}} - \frac{e(\mathbf{x}|_{t_i}, y|_{t_i}) h_{(\theta)}(\mathbf{x}|_{t_i})}{\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}}.\end{aligned} \tag{4}$$

Equation (4) is the update rule for any parameter θ at time t_i given a new data point $(\mathbf{x}|_{t_i}, y|_{t_i})$. The updates rules for specific parameters of FM models can then be derived as following:

$$w_0|_{t_i} = w_0|_{t_{i-1}} - \frac{e(\mathbf{x}|_{t_i}, y|_{t_i})}{|S|_{t_i}| + \lambda_{(\theta)}}, \tag{5}$$

$$w_l|_{t_i} = w_l|_{t_{i-1}} - \frac{e(\mathbf{x}|_{t_i}, y|_{t_i}) x_l|_{t_i}}{\sum_{(\mathbf{x},y)\in S|_{t_i}} x_l^2 + \lambda_{(w_l)}}, \tag{6}$$

$$V_{lf}|_{t_i} = V_{lf}|_{t_{i-1}} - \frac{e(\mathbf{x}|_{t_i}, y|_{t_i}) h_{(V_{lf})}(\mathbf{x}|_{t_i})}{\sum_{(\mathbf{x},y)\in S|_{t_i}} h^2_{(V_{lf})}(\mathbf{x}) + \lambda_{(V_{lf})}}, \tag{7}$$

where $|S|_{t_i}|$ denotes the number of data points received at time t_i, x_l is the l-th element of vector $\mathbf{x}$, and

$$h_{(V_{lf})}(\mathbf{x}) = x_l \sum_{l=1}^{n} V_{lf} x_l - x_l^2 V_{lf} = x_l q(\mathbf{x}, f) - x_l^2 V_{lf}, \tag{8}$$

with

$$q(\mathbf{x}, f) = \sum_{i=l}^{n} V_{lf} x_l \tag{9}$$

as shown in [12]. We can see that, in order to update efficiently, we need to store some cache values. For linear term updates, $\forall l \in [n]$, we need $a_l = \sum_{(\mathbf{x},y)\in S} x_l^2$; for interaction terms, $\forall l \in [n]$, $f \in [k]$, we need $B_{lf} = \sum_{(\mathbf{x},y)\in S} h^2_{(V_{lf})}(\mathbf{x})$. With these update rules, our online FM training algorithm is summarized in Algorithm 1.

Algorithm 1 Online-ALS for FM

Require: Data stream $(\mathbf{x}|_{t_i}, y|_{t_i})$, $i = 1, 2, \cdots$
Ensure: FM model parameters w_0, $\mathbf{w}$, $\mathbf{V}$; cache $\mathbf{a}$, $\mathbf{B}$
1: Initialize $w_0|_{t_0}$, $\mathbf{w}|_{t_0}$, $\mathbf{V}|_{t_0}$ from Gaussian samples;
2: Initialize cache a_l and B_{lf} to 0, $\forall l \in [n]$, $f \in [k]$
3: **while** data point $(\mathbf{x}|_{t_i}, y|_{t_i})$ arrives **do**
4: $e(\mathbf{x}|_{t_i}, y|_{t_i}) = \hat{y}(\mathbf{x}|_{t_i}) - y|_{t_i}$
5: Update w_0 using Equation (5)
6: $e(\mathbf{x}|_{t_i}, y|_{t_i}) = e(\mathbf{x}|_{t_i}, y|_{t_i}) + (w_0|_{t_i} - w_0|_{t_{i-1}})$
7: **for** $l \in [n]$ **do**
8: $a_l = a_l + x_l^2|_{t_i}$
9: Update w_l using Equation (6)
10: $e(\mathbf{x}|_{t_i}, y|_{t_i}) = e(\mathbf{x}|_{t_i}, y|_{t_i}) + (w_l|_{t_i} - w_l|_{t_{i-1}})x_l|_{t_i}$
11: **end for**
12: **for** $f \in [k]$ **do**
13: Calculate $q(\mathbf{x}|_{t_i}, f)$ using Equation (9)
14: **for** $i \in [n]$ **do**
15: Calculate $h_{(V_{lf})}(\mathbf{x}|_{\mathbf{t_i}})$ using Equation (8)
16: $B_{lf} = B_{lf} + h^2_{(V_{lf})}(\mathbf{x}|_{\mathbf{t_i}})$
17: Update V_{lf} using Equation (7)
18: $e(\mathbf{x}|_{t_i}, y|_{t_i}) = e(\mathbf{x}|_{t_i}, y|_{t_i}) + (V_{lf}|_{t_i} - V_{lf}|_{t_{i-1}})x_l|_{t_i}$
19: $q(\mathbf{x}|_{t_i}, f) = q(\mathbf{x}|_{t_i}, f) + (V_{lf}|_{t_i} - V_{lf}|_{t_{i-1}})x_l|_{t_i}$
20: **end for**
21: **end for**
22: **end while**

It is easy to see that the memory cost for the cache is $(k + 1)n$, which is independent of the number of data points. Note that as $\mathbf{x}$ is typically sparse in real data, we can always check if x_l is 0 and skip the iteration if it is so. Hence, the computation cost for each while loop is $O(km)$, where m is the number of non-zero elements in $\mathbf{x}$. So our algorithm yields fast implementation.

It is notable that although Algorithm 1 starts with random initialization, if we have a pre-trained FM model, we can skip step 1 in the algorithm (or even step 2 if the cache values have been saved). This is useful in a real world setting because one can always use batch training on a small set of historical data to get a reasonably good model, and then update the model in a streaming setting to capture the real-time trends, and get a good accuracy without re-training using the large dataset. Similarly, a rolling bootstrap dataset from recent past can be used to initialize from time to time.

4 EXPERIMENTS

In this section, we will present our experimental evaluation of Algorithm 1. Besides Online-ALS, we also consider Online-ALS-pretrained, which uses 10% of data to pre-train the model in batch fashion, and then do online updating as in Algorithm 1. By default we use ALS for pre-training. We compare Online-ALS, Online-ALS-pretrained and SGD.

Our evaluation metric is root-mean-square error (RMSE), suppose there are N data points for evaluation, then it is defined as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}(\mathbf{x}|_{t_i}) - y|_{t_i}))^2}$$

We also use two kind of training error to evaluate the model:

- True-training Error: RMSE of training error without reloading the model, i.e., prediction for data point $\mathbf{x}|_{t_i}$ will not be adjusted after t_i.
- Training Error: RMSE for the model at time t_N on the entire training set, i.e., use the final model to recalculate the prediction for each data point

Note that True-training Error is more important for a streaming setting, as after updating the model we will dispose of the data point and there is no way to reuse it. It also measures the quality of recommendations in a real-time system.

4.1 Simulations

We first evaluate our method with simulation data. We generate the ground-truth FM model with random samples from Gaussian, and then simulate data from this FM, with perfect label. The size of the simulated dataset is 1.5M for training, and 171K for testing. The minimum target value is -0.986732, and the maximum is 1.25055. The result is shown in Fig. 1, we can see that SGD has faster convergence on test error, but Online-ALS-pretrained is much better because of the pre-trained procedure. In terms of training errors, Online-ALS converges faster than SGD. And we can see, pre-trained is very useful for both training and true training error.

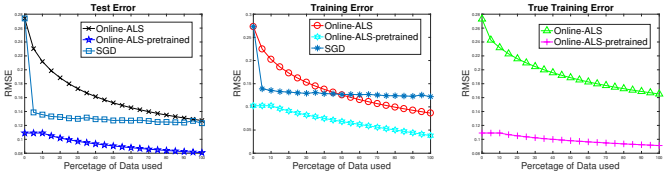

Figure 1: Simulated Data

4.2 Real data

In this section, we present the results on large scale industrial datasets. We use the feature selection framework in [8] to select about 10 representative features from hundreds of them to accelerate the training procedure. The target (y) of the dataset is session progress introduced in [14], which is a real valued number in $[0, 1]$. Session progress is an implicit measure of engagement of the user with the video, which measures how much of the video was watched. We also use RMSE to measure the accuracy of the prediction.

Table 1: Statistics of datasets.

Dataset	Training size	Test size	Feature Selection
1	61K	6.8K	Yes
2	1.65M	61K	Yes
3	154K	17K	Yes
2-subset	36K	4K	No

We use real industrial dataset and reorder the data points by the time it appears for training and test. Unlike the simulation experiments, the data points can be dependent with each other and the underlying model may not necessarily be a perfect FM model.

The statistics of datasets we are using are shown in Table 1 but we omit the names of the datasets to preserve confidentiality. 2-subset is a subset of dataset 2, but without applying any feature selection technique.

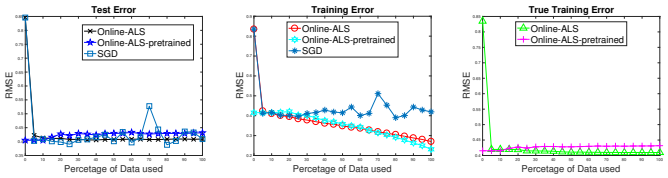

Figure 2: Dataset 1

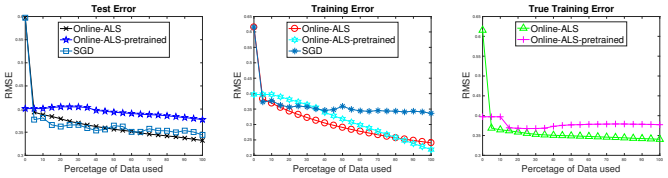

Figure 3: Dataset 2

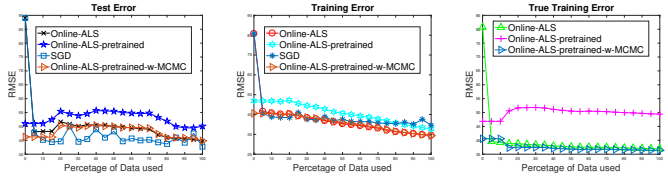

Figure 4: Dataset 3

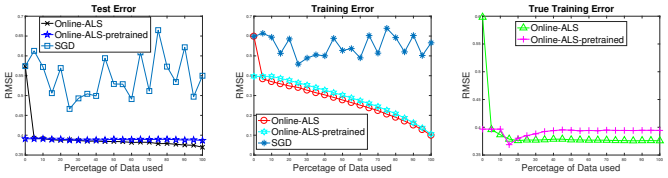

Figure 5: Dataset 2-subset

Figs 2, 3, 4 and 5 show the results. We can see that for real datasets, online-ALS is much more stable than SGD, both for training and test errors. Pre-training is not as advantageous as it is on the simulation data, maybe because the first 10% of training data is not representative and gives a bad initialization, such as Dataset 3 in Fig 4. However, pre-training with MCMC [11] gives a better result. In a real application, we can evaluate which batch training is better for a specific dataset and pre-train to get a better initialization. Also in Fig 5, it is very interesting to see that Online-ALS is more stable when using more context features, where SGD fluctuates a lot. It is also notable to mention that for batch training (ALS, MCMC, SGD) with real industrial data, training with multiple passes of the data does not perform much better than training with one pass of the data (batch or online). Hence, online training in a streaming setting can provide faster results without compromising accuracy in real world applications. We omit the detailed results here for space constraints.

5 CONCLUSIONS

In this paper, we introduced an algorithm for online FM model training with streaming data. It has comparable results with batch training and has much more stable training error than SGD. We validated the efficiency and robustness of our algorithm with extensive experiments on simulated and industrial datasets.

REFERENCES

[1] Mathieu Blondel, Masakazu Ishihata, Akinori Fujino, and Naonori Ueda. 2016. Polynomial networks and factorization machines: New insights and efficient training algorithms. In *33rd International Conference on International Conference on Machine Learning (ICML)*, Vol. 48. 850–858.

[2] Wei-Sheng Chin, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin. 2018. An efficient alternating newton method for learning factorization machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 9, 6 (2018), 72.

[3] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press, 1725–1731.

[4] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 355–364.

[5] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. 2015. Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 227–238.

[6] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 43–50.

[7] Ming Lin and Jieping Ye. 2016. A non-convex one-pass framework for generalized factorization machine and rank-one matrix sensing. In *Advances in Neural Information Processing Systems*. 1633–1641.

[8] Xueyu Mao, Saayan Mitra, and Viswanathan Swaminathan. 2017. Feature Selection for FM-Based Context-Aware Recommendation Systems. In *2017 IEEE International Symposium on Multimedia (ISM)*. 252–255. https://doi.org/10.1109/ISM.2017.42

[9] Surabhi Punjabi and Priyanka Bhatt. 2018. Robust Factorization Machines for User Response Prediction. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. 669–678.

[10] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[11] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 57.

[12] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 635–644.

[13] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.

[14] Gang Wu, Viswanathan Swaminathan, Saayan Mitra, and Ratnesh Kumar. 2014. Online video session progress prediction using low-rank matrix completion. In *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on*. IEEE, 1–6.

[15] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press.