Context Aware Recommender System for Large Scaled Flash Sale Sites

Wanying Ding*, Ran Xu†, Ying Ding‡, and Yue Zhang§, and Chuanjiang Luo ¶ and Zhendong Yu‖
VIPSHOP(US) Inc., US R & D Center
San Jose, USA
*Email: *alice.ding@vipshop.com,wanying.alice@gmail.com, †ran.xu01@vipshop.com, ‡ian.ding@vipshop.com,*
§yue.zhang07@vipshop.com, ¶larry.luo@vipshop.com,
‖zhendong01.yu@vipshop.com

***Abstract*—Flash Sale Sites popularize because they save great money for users. Good recommender systems can further save users' time to improve their online shopping experiences. Although there exsit a lot of studies on recommender system, very few focus on flash sale sites. *Big Data*, *Context Sensitivity*, and *Feature Engineering* are three key challenges for one to build a good recommender system. This paper proposes two deep learning oriented models: Tensor-AutoRec and Hybrid-AutoRec to cope with the problems within an industrial context. First, these two models can handle storage and speed problem caused by big data. Second, both models incorporate context information, so they can generate more relevant recommendations by adapting to specific contextual situations. Third, our deep learning-based models can be trained end-to-end without tedious feature engineerings. Extensive experiments with a half year real transcation data demonstrate that our models can outperform classifcal ones in terms of different evaluation metrices. Finally, online A/B testing results showed that our model can improve our old recommendation system over various online performance indicators.**

***Keywords*-Recommender System;Deep Learning; Big Data; Representation Learning; Distributed System**

I. Introduction

Flash Sale Site is a type of e-commerce websites featuring "flash sales" on certain products with heavy discounts for a short period of time (as shown in Figure 1). Vipshop[1], Groupon[2] and Zulily[3] are three representations of such websites. Since they provide users with a way to save money, flash sale sites have thrown a hit over the internet. Like many other e-commerce businesses, flash sale sites also need recommender system to cope with information overload problem. Although recommender system for e-commerce has been extensively studied [1]–[3], most exsiting systems cannot be applied directly to flash sale sites as there are some new challenges.

Context Sensitivity. Context information, such as *Time*, *Discount*, *Aavailable Size* etc. (as shown in Figure 1), heavily impacts consumers' purchase decisions. Many existing recommender systems do not consider it and are not suitable for recommendations in flash sale sites. For example, it is not very meaningful to recommend a LV bag with 10% discount to a user who normally purchase luxury bags with 90% discount.

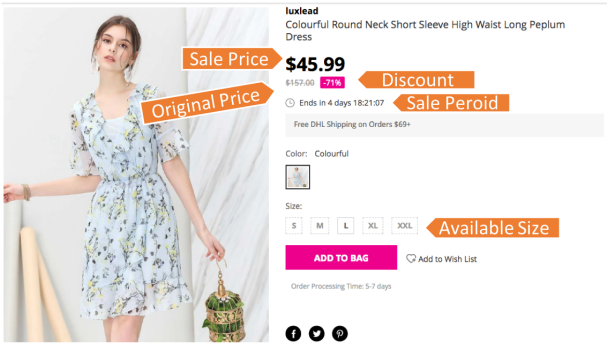


Figure 1. Demo of Flash Sale Sites

Feature Engineering and Data Sparsity. Feature Engineering always plays a critical role in machine learning. Wrong or missing features will dramatically harm a model's performance. Feature selection and interactions are also critical but require human labors. In addition, data sparsity is often quite serious when training recommender systems. It is challenging to find a computation and memory-efficient way to build an effective recommender system in this situation.

Big Data. Big Data makes many theoretically successful recommender systems fail in practice. We tested the most widely used Collaborative Filtering [4] on our transaction datasets. With millions of users × millions of items, it failed to load the whole dataset into memory. Large scale dataset can make the training process dramatically long as it cannot converge quickly. We tested a two-layer neural network and found it needs at least three days to converge when trained on one day's purchase data on a standalone machine. It is unpractical to use such a long time for training.

To alleviate the problems above, we proposed two deep learning oriented models, Tensor-AutoRec and Hybrid-AutoRec. Our main contributions are:

- We designed two context-aware recommender systems, Tensor-AutoRec and Hybrid-AutoRec.
- We proposed a distributed computation workflow to

[1]https://us.vip.com/
[2]https://www.groupon.com/
[3]https://www.zulily.com/

deal with large scale input data.
- We conducted extensive experiments using large scale real transaction data. We compared our models with several classifcal recommendation models offline and also tested their performances online with A/B testing.

II. Related Work

Our work is closely related to hybird recommender system, context-aware recommender system and deep learning-based recommendation systems. in this section, we will give an overview of the related work along directions.

A. Hybrid Recommender System

Technically, recommender models can be classified into three categories: *Collaborative Filtering* [5], [6], *Content Based Model* [7], [8] and *Hybrid Model* [9], [10].

Collaborative Filtering tries to find users in a community that share similar interests via their historical transactions. Collaborative Filtering has achieved great success but also suffers from cold start [11] and sparsity problem [12]. Content-based models, contrast to collaborative filtering, can deal with cold start and sparsity problem better. However, it always faces missing or wrong data problems.

Hybrid models aggregate collaborative filtering and content-based model to improve recommendation accuracy. Recent research has proved that a hybrid approach could be more effective in some cases. Therefore, we design two hybrid models: Tensor-AutoRec and Hybrid-AutoRec, to guarantee reliable recommender results.

B. Context Sensitive Recommendation System

Context-aware recommender systems (CARS) can generate more relevant recommendations by adapting recommendations to specific contextual situations. A context-aware recommender system deals with data records of the form $< user, item, context, rating >$, where each specific record includes not only how much a user like an item, but also the contextual information. Formally, CARS problem can be represented as Equation (1) [1]:

$$f_R : User \times Item \times \prod_{i}^{c} Context \rightarrow Relevance \qquad (1)$$

Context information can be either concatenated with other features [13] or decomposed from a tensor factorization [14]. Our proposed Tensor-AutoRec and Hybrid-AutoRec try these two methods respectively. Considering most existing models usually consider just one type of context factors, or model different factors independently, our models take multiple contextual factors as well as their correlations into consideration.

C. Deep Learning Oriented Recommendation System

. Recent studies demonstrate deep learning's effectiveness in recommender systems. CNN [15], AutoEncoder [16], [17], RNN [18], DSSM [19] and GAN [20] have been applied to recommender systems. Among these models, CNN requires the input to be a matrix, RNN usually works on sequential data, and DSSM or GAN usually deal with search system. To deal with classical transaction data, we apply *AutoEncoder* as our framework.

There exist several studies using AutoEncoder for recommender system. Collaborative Deep Learning (CDL) [16] integrates Stack Denoising Autoencoder (SDAE) with collaborative filtering for content based recommendation. AutoRec [17] applies a purchase matrix to control the construction of user partial vectors $\mathbf{r^{(u)}}$ and item partial vectors $\mathbf{r^{(i)}}$. Collaborative Filtering Neural Network (CFN) [5] extends AutoRec by incorporating side information, such as user profiles and item description, to mitigate the sparsity and cold start influence.

However, all the models mentioned about are not context-aware. Our Tensor-AutoRec and Hybrid-AutoRec model can incorporate context information into deep learning oriented recommender systems and achieve more reliable context sensitive recommendations.

III. Methogology

The main goal of our model is to predict the unknown user-item relationship $R^c_{u,i}$ from user u to item i in a certain context c. $R^c_{u,i}$ represents users' click or purchase information (also referred as *Implicit Feedback*). One major problem associated with implicit feedback is that all our observed $R^c_{u,i}$ are 1, but we don't have any negative samples with $R^c_{u,i} = 0$. This is defined as One Class Classification (OCC) problem by previous work [21]. Although there are many methods [22] to generate negative samples, most of them are too complicated for large scale data. Thus, we simply corrupt some features to generate negative samples. From a record like "one user purchases a floral dress in August with 40% discount", we generate one negative sample as "the user doesn't purchase the floral dress in December with 20% discount" by corrupting the "Time" and "Discount" feature. We also set a negative loss weight wl to lower the risk that it may generate a sample like "the user doesn't purchase the floral dress in July when it is 60% discounted", which is more likely to be a positive sample.

Note our proposed deep learning models aim at learning suitable representations of entities including users, items and context. These representations will then be used in the recommder system of a falsh sales website to make recommendations. Towards this end, we will mainly rely on models that can naturally learn entities representations and seamlessly encode users' peferences over items under variaous contexts. In the next subsection, we will first introduce denoising stacked autoencoder as our models are

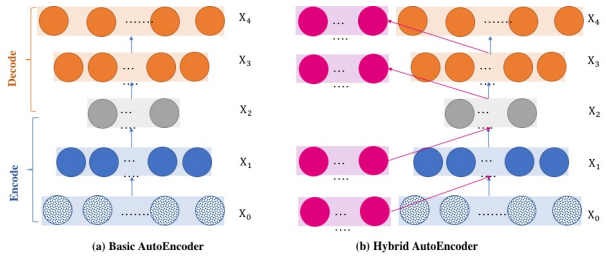


Figure 2. Demonstraction of AutoEncoders

designed on top of it . We will then explain our proposed models in detail.

A. Denoising Stacked AutoEncoder

Our proposed models are based on the AutoEncoder framework. An *AutoEncoder* is a neural network that is trained to reconsturct the input data through a decoder-encoder framework. Figure 2(a) demonstrates the most basic AutoEncoder. In our models, we only use two-layer decoder and encoder for the sake of model complexity and training and inference time. As shown in Figure 2, the input $\mathbf{X_0}$ is first transformed to a hidden state $\mathbf{X_1}$ through an activation function F_1:

$$\mathbf{X_1} = F_1(\mathbf{W_0}\mathbf{X_0} + \mathbf{b_0}) \tag{2}$$

Similarly, we further transform $\mathbf{X_1}$ into a hidden vector $\mathbf{X_2}$ through an activation function F_2:

$$\mathbf{X_2} = F_2(\mathbf{W_1}\mathbf{X_1} + \mathbf{b_1}) \tag{3}$$

For the decoding step, the vector $\mathbf{X_2}$ is first transformer to $\mathbf{X_3}$ thtough the first reconstruction layer with activation function F_3, and $\mathbf{X_3}$ is further mapped into $\mathbf{X_4}$ with another activaiton function F_4.

$$\mathbf{X_3} = F_3(\mathbf{W_2}\mathbf{X_2} + \mathbf{b_2}) \tag{4}$$

$$\mathbf{X_4} = F_4(\mathbf{W_3}\mathbf{X_3} + \mathbf{b_3}) \tag{5}$$

Here, F_1, F_2, F_3 and F_4 are all non-linear activiation functions, such as sigmoid, tanh or relu. $\mathbf{W_0}$, $\mathbf{W_1}$, $\mathbf{W_2}$ and $\mathbf{W_3}$ are weight matrices and $\mathbf{b_0}$, $\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{b_3}$ are bias vectors. The final reconstruction weight matrix $\mathbf{X_4}$ should have the same shape as input vecotr $\mathbf{X_1}$. By minimizing the reconstruction error (shown as Equation 6, in which n is the input dimension), we can learn a vector $\mathbf{X_2}$, which has a much lower dimension than X_0, as the representation vector of $\mathbf{X_0}$.

$$L_{a\_rec}(\mathbf{X_0}, \mathbf{X_4}) = \frac{1}{n}\sum_{i}^{n} ||X_0 - X_4|| \tag{6}$$

To learn more robust representations, we usually train the autoencoder to reconstruct the input from a corrupted version of it. There are usaually two ways to do corruption: (1)mask some dimension to zero or (2)add Gaussian noise. We choose the second method since we already have many zero values in our input data.

Although many successful models combine autoencoder with Collaborative Filtering, how to apply it to a context-aware recommendation problem was rarely discussed. To answer this question, we proposed two models. *Tensor-AutoRec* (as shown in Figure 3(a)) and *Hybrid-AutoRec* (as shown in Figure 3(b)) and explain them in the next two subsections.

B. Tensor-AutoRec

The most straightforward method to incorporate context into a recommender system is to use another AutoEncoder to embed context information, and use a tensor decomposition model to combine user, item and context information together. Thus, we first propose a Tensor-AutoRec model (as shown in Figure 3(a)).

We create three independent denoising autoencoders to encode User x_i, Item y_j, and Context z_k. In this way, user input x_i can be encoded into u_i, Item y_j can be encoded into i_j, and context z_k can be encoded into c_k. Then, we combine these three vectors together through a *Tucker Decomposition* [23] model.

Tucker Decomposition, which is also known as Higher-Order SVD (HOSVD), can be treated as a stable way of extending SVD to a higher order case [24]. It decomposes an N-th order tensor into a single core tensor multiplied by N component matrices (i.e. a matrix for each dimension of the tensor). The core tensor acts as a scaling factor, which depicts the relationships between the component matrices, just like the singular values in SVD. For a tensor $\mathfrak{R} \in \mathbb{R}^{I\times J\times K}$ the prediction of a rating $\hat{r}_{ijk}$ is calculated as Equation 7:

$$\hat{r}_{ijk} = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}u_{ip}i_{jq}c_{kr} \tag{7}$$

Here, $\mathfrak{g} \in \mathbb{R}^{P\times Q\times R}$ is the core tensor and parameters P,Q, and R represent vector length of $\mathbf{u}$, $\mathbf{i}$ and $\mathbf{c}$ resprectively.

In this way,the loss fnction of Tensor-AutoRec model consists two parts:(1) *Reconstruction Loss* coming from entity embedding (shown as Equation 6) and (2) *Prediction Loss* coming from tensor decomposition. As we have mentioned before, our negative samples are artifically created and we need to take advantage of wl to lower the effect coming from fake negative samples. Thus, the prediction loss can be formulated as :

$$L_{pre} = \frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r_i}) + wl \cdot \frac{1}{m}\sum_{j=1}^{m}(r_j - \hat{r_j}) \tag{8}$$

where n is the number of positive samples, and m is the number of negative samples. wl is the negative weight ranging in $[0, 1]$. In our experiments, we set it to 0.5.

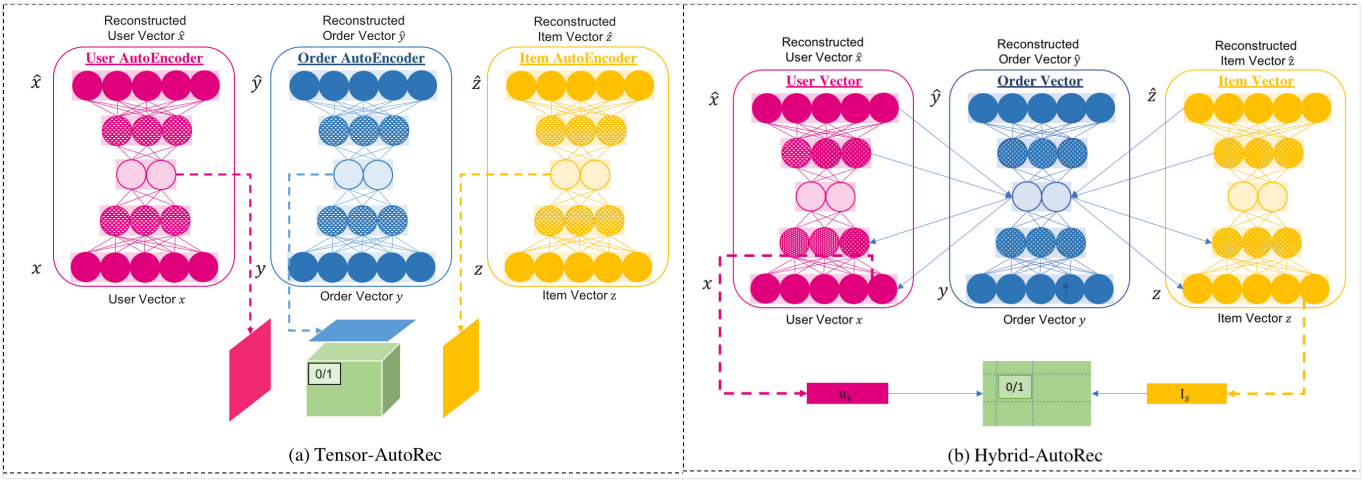


Figure 3. Brief View of Two Models

Thus the whole loss function for Tennsor-AutoRec can be formulated as Equation 9

$$Loss = \lambda_u L_{a\_rec}(\mathbf{x}, \hat{\mathbf{x}}) + \lambda_i L_{a\_rec}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda_c L_{a\_rec}(\mathbf{z}, \hat{\mathbf{z}}) + \lambda_r L_{pre}(\mathbf{r}, \hat{\mathbf{r}}) \tag{9}$$

where the λs are the weights of different loss compositions.

C. Hybrid-AutoRec

Inspired by Amiri and Resnik's work [13], we create another model Hybrid-AutoRec (show in Figure 3(b)) to incorporate context information into our recommender system. First, we learn a context embedding $\mathbf{h_c}$ with basic AutoEncoder from context information. Then we combine $\mathbf{h_c}$ with raw input $\mathbf{X_0}$ to generate $\mathbf{X_1}$. This process can be formulized as

$$\mathbf{h_c} = F_c(\mathbf{W_c c} + \mathbf{b_c}) \tag{10}$$

$$\mathbf{X_1} = F_1(\mathbf{W_{x1} X_0} + \mathbf{V_{c1} h_c} + \mathbf{b_{x1}}) \tag{11}$$

In a similar way, we can embed $\mathbf{X_1}$ and $\mathbf{h_c}$ into $\mathbf{X_2}$, which we think contains both input information X_0 and context information C. The we decode $\mathbf{X_2}$ into $\mathbf{X_3}$ and $\mathbf{h_c}$, and further decode $\mathbf{X_3}$ to $\mathbf{X_4}$ and $\mathbf{h_c}$. $\mathbf{X_4}$ should have the sample shape with $\mathbf{X_1}$. In this model, the reconstruction loss can be written as:

$$L_{h\_rec}(\mathbf{X_0}, \mathbf{X_4}) = \frac{1}{n}\sum_{i}^{n} ||X_0 - X_4|| + \lambda_c \frac{1}{L/2}\frac{1}{S}\sum_{l=1}^{L/2}\sum_{k=1}^{S} ||h_c - \hat{h_c}|| \tag{12}$$

where L is the number of layers, and $L/2$ indicate the decode layers. S is the h_c dimension length. n is nput dimension length. λ is the loss weight.

Our Hybrid-AutoRec can combine two Hybrid AutoEncoders: one is for user encoding and the other one is for item encoding. In this model, we first embed context information as c_k with basic autoencoder via Equation 10, and then combine c_k with each layer of user input and item input. Since the user embeding u_i and item embeding i_j already encod context information, we can use the basic idea of matrix factorization to model ratings.

$$\hat{r_{ij}} = g_{ij} u_i i_k \tag{13}$$

We combine all the loss function together, and get the overall objective function for our Hybrid-AutoRec Model as Equation 14. L_{pre} in Equation 14 is the same as Equation 8

$$Loss = \lambda_u L_{h\_rec}(\mathbf{x}, \hat{\mathbf{x}}) + \lambda_i L_{h\_rec}(\mathbf{y}, \hat{\mathbf{y}}) + \lambda_r L_{pre}(\mathbf{r}, \hat{\mathbf{r}}) \tag{14}$$

IV. Experiments and Evaluation

A. Distributed System Overview

To deal with large scale dataset, we developed a distributed computation platform (as shown in Figure 4). Our system consists fof two main components: Data ETL (extract, transform and load) and model training/ testing.

Data Extract, Transform and Load (ETL): Users' click and purchase data are collected and stored in Hive data warehouse. We implement SparkSQL to transform the data into TFRecords, which is a standard data format for Tensorflow[4], an open-source deep learning library provided by Google.

Distributed Model Training: We have also built a distributed Tensorflow cluster for model training and testing. Within a Tensorflow sever cluster, machines are divided into two parts: *Workers* and *Parameter Severs*. Within the cluster, all model parameters are shared across workers

[4]https://www.tensorflow.org/

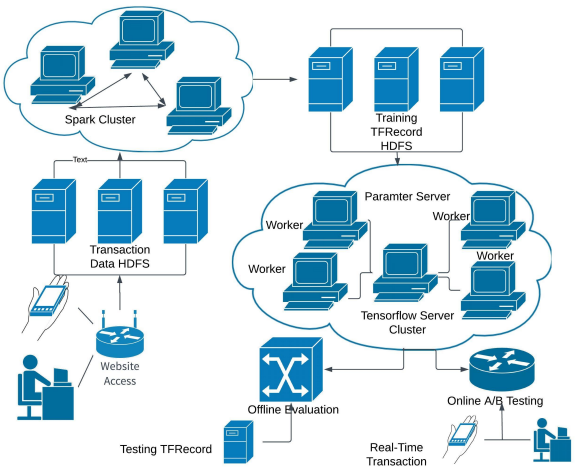


Figure 4. Distributed System Overview

while parallelizing data and gradient updates. With multiple workers, several batches are processed at the same time. Once all the workers are done, all gradients will be averaged, and a single update will be sent to the parameter server.

After one model has been well trained, we test the model's performance with our testing data.

B. Offline Evaluation

1) Datasets: Since Click Through Rate (CTR) and Conversion Rate(CVR) are two essential concepts for e-commerce websites, we collect these two types of datasets for our experiments: *Brand Click* and *Item Order* as our datasets for evaluaton.

(1) *Brand Click Dataset*. Brand Click Data is collected from January 2018 to July 2018. For each month, we randomly sample of 50 million entries. 10% of the dataset is randomly sampled as the testing dataset and the left is used for training. Each brand click record includes user information, brand information, and the context, e.g., time, discount, in which the user clicks the brand.

(2) *Item Purchase Dataset*. Item purchase data is also collected from Jan. 2018 to July 2018. We also ramdomly sample 50 milllion entries in each month and use 10% of the whole dataset for testing. Each item order record includes user information, item information, and the context, e.g., time, discount, in which the user buy the item.

2) Measurement Methods: We apply four metrics over our datasets to compare the performances among different methods.

1) Root Mean Square Error(RMSE). RMSE measures the deviation between the predicted value $\hat{y}$ and the truth value y.It can be calculated as Equation 15

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y_i} - y_i)^2)}{n}} \tag{15}$$

2) Average Cross Entropy(ACE). Cross entropy measures the difference among two probability distribution: truth and prediction. It is also a common metrics to evaluate whether a classification is good enough. It can be calculated as Equation 16

$$ACE = \frac{-\sum_i^n y_i log\hat{y_i} - \sum_i^n (1-y_i) log(1-\hat{y_i})}{n} \tag{16}$$

3) Area Under ROC Curve (AUC). The ROC curve is created by plotting the true positive rate(TPR) against false positive rate(FPR) at various threshold settings. A larger AUC indicates the model can distinguish positive samples and negative samples better.

4) Average Precision Score(APS). APS summerizes a precision-recall curve as the weighted mean of precision achieived at each threshold. with the increased in recall from the previous threshold used as the weight. It can be calculated via Equation17

$$APS = \sum_n (R_n - R_{n-1}) P_n \tag{17}$$

where P_n and R_n are the precision and recall at the n^{th} thrshold.

3) Comparison with Baselines: Since the most widely used models in the industry for recommender systems are Logistic Regression and Factorization Machine. We choose these two models as our baselines for comparison.

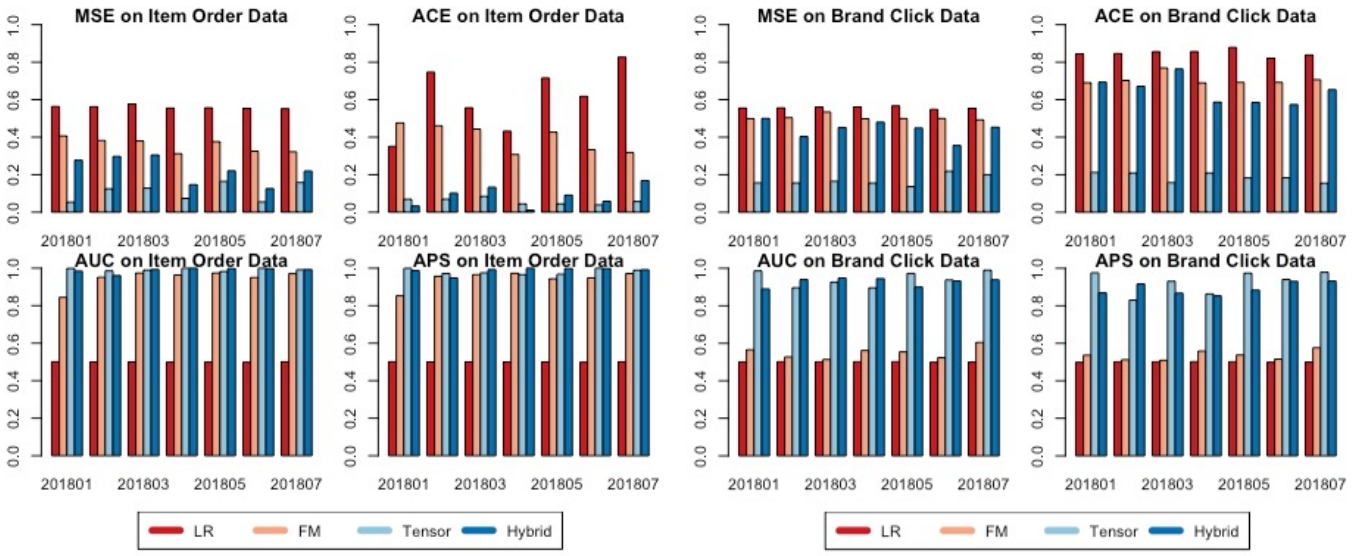
Logistic Regression: Logistic Regression(LR) is the most popular model for the online recommendation system. Given feature list $\mathbf{X}$, the probability $P(r = 1)$ can be calculated as Equation 18

$$P(r = 1|\theta) = \frac{1}{1 + e^{-(WX+b)}} \tag{18}$$

Factorization Machine: Factorization Machine(FM) models are a combination of linear regression and matrix factorization taht models feature intreactions but in linear time. It takes inspiration from matrix factorization, and models the feature interactions like using latent vectors. As a result, every feature f_i has a corresponding latent vector v_i. And two features's interactions are modeled as $v_i \cdot v_j$. FM can be calculated as Equation 19.

$$\hat{r} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \mathbf{v_i}, \mathbf{v_j} > x_i x_j \tag{19}$$

The evaluation result is shown as Figure 5. Figure 5(a) shows the evaluation results over item order dataset. It clearly shows that LR has the worst performance since it has higher error and entropy but lower AUC and precision. In comparison, FM and two autoencoder models have much



(a) Evaluation on Item Order Data

(b) Evaluation on Brand Click Data

Figure 5. Evaluation Results

better performance. All of these three models have very high AUC and APS, indicating they all give positive samples with high scores and negative samples with lower scores. However, with respect to MSE and ACE, FM performs a little worse since it has higher MSE and ACE values. We can take a brief overview of the predict results from FM, Tensor-AutoRec, and Hybrid-AutoRec.

> **Truth**: [1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
> **FM Prediction**: [0.513, 0.257, 0.385, 0.583, 0.238, 0.379, 0.167, 0.01, 0.01, 0.641]
> **Tensor Prediction**: [0.948,1.12e-28, 0.992,0.934,2.81e-5,0.902,5.92e-15,2.55e-10,2.36e-14,0.996]
> **Hybrid Prediction**:[0.997,8.085e-4,0.988,0.999,6.89e-6, 0.821,0.0302,0.0113,0.00144,0.865]

It is easy to note that predictions of positive samples by FM are close to 0.5. We suspect this is why FM can achieve high AUC and APS score but performs poorly on MSE and ACE.

We get similar results from the brand click dataset; all FM, Hybrid-AutoRec, and Hybrid-AutoRec have outperformed the LR model. However, FM has much worse performance compared to it does on item order dataset. The only reason we can come is that brand dataset has much fewer features, and FM can infer robust results well from limited features.

4) The Importance of Incorporating Context: As we mentioned in the very beginning, one key feature that distinguishes flash sale sites from traditional e-commerce websites is that flash sale sites are more context sensitive.To confirm this hypothesis, we compare our proposed Hybrid-AutoRec model and vanilla auto-encoder, which does not incorporate context informaiton. Considering the Tensor model has achieved better performance over the Hybrid model (according to Figure 5) if the Hybrid model can outperform Raw-AutoRec, Tensor-Model will definite outperform it.

The comparison results are shown as table I and II. Obviously, without context embedded, the model's performance has significantly dropped in every metrics. Thus, we can conclude that context information indeed matters for flash sale sites, and they should be considered seriously.

C. Online Evaluation (A/B Testing)

In industry, hundred times of offline evaluation is no match for one time online testing. Considering the stringent response time requirement, we could not put our model online as an end-to-end recommender system in the current stage. Thus, we submit our embedded vectors as features to support the current online recommender system to investigate whether our embedded features can improve the current models' performance. We submitted our user vector and tested our result from Jan. 2nd 2018 to Jan. 28th 2018. We use the following four online preformance indicators to measure the performance and show the results in Figure 6.

1) Per Customer Transaction.It indicates the average ammount of money per transaction in the correponding day.
2) Per Item Transaction. It indicates the amount of money every item has been spent on each of transaction.
3) Per Person Consumption. It indicates the amount of money that a user spends on his/her every visiting.
4) Conversion Rate(CVR). Percentage of purchase against click.

Table I
CONTEXT IMPORTANCE TEST OVER ITEM ORDER DATA

Date	RMS		ACE		AUC		APS	
	With Context	Without Context	With Context	Without Context	With Context	Without Context	With Context	Without Context
201801	0.277	0.634	0.0327	1.712	0.984	0.528	0.986	0.560
201802	0.297	0.677	0.101	3.798	0.960	0.508	0.948	0.569
201803	0.305	0.651	0.132	1.874	0.993	0.523	0.992	0.565
201804	0.147	0.618	0.110	1.802	0.999	0.570	0.999	0.658
201805	0.220	0.670	0.0907	2.892	0.997	0.515	0.998	0.596
201806	0.125	0.637	0.0581	1.592	0.998	0.534	0.997	0.616
201807	0.219	0.617	0.0684	1.651	0.992	0.553	0.991	0.639

Table II
CONTEXT IMPORTANCE TEST OVER BRAND CLICK DATA

Date	RMS		ACE		AUC		APS	
	With Context	Without Context	With Context	Without Context	With Context	Without Context	With Context	Without Context
201801	0.499	0.705	0.693	6.283	0.890	0.500	0.869	0.501
201802	0.403	0.700	0.671	6.111	0.939	0.503	0.916	0.505
201803	0.451	0.586	0.763	5.208	0.947	0.634	0.867	0.645
201804	0.480	0.700	0.587	5.759	0.945	0.528	0.853	0.543
201805	0.449	0.683	0.585	5.156	0.899	0.520	0.883	0.524
201806	0.355	0.643	0.573	4.789	0.932	0.608	0.930	0.654
201807	0.453	0.703	0.653	6.776	0.938	0.502	0.931	0.501

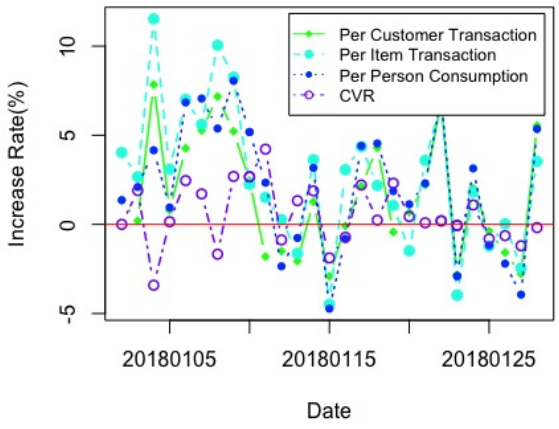


Figure 6. Online A/B Testing Result

Figure 6 shows that recommendations incorporating embeddings learnt by our models get improvement on all four online performance indicators most of the time. This is because the embeddins learnt by our models can embedd users' profile information and pereferences. They can provide more meaningful input to our recommendation system.

D. Serve as Entity Embedding

To evaluate our results qualitatively, we visualize our embedding vectors. Intuitively, a good embedding should be able to group similar entities. We use a K-Means model to label similar users and visualize them. Taking user data

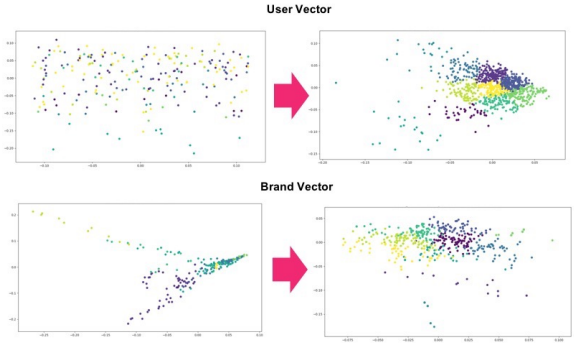


Figure 7. Visualization of Embeddings

as an example, before we embed them, every user record is of high dimension. It is hard to measure their similarity, and all data is scattered in the space(shown as the left part in Figure 7). After embedding, we find that similar users are easier to be grouped (shown as the right part in Figure 7). Thus our vectors can be used in many other applications instead of the original long user records.

V. CONCLUSION

In this work, we proposed two context-aware recommender systems, **Tensor-AutoRec** and **Hybrid-AutoRec**, for flash sale sites. Compared with existing work on recommender systems, our main contributions are: (1) Our proposed models can take context information into consideration, which makes it particually suitble for online flash sales. (2) Our deep learning framework does not require tedious feature engineering that are necessary to tranditional recommendation algorithms. But they can still achieve better

performances. (3) We have designed a distributed pipeline for data ETL and model training/ testing, to cope with big data problem. (4) We conducted A/B testings and proved that our proposed models can improve the current system in real industrial applications.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," in *Recommender systems handbook*. Springer, 2015, pp. 191–226.

[2] Y. Xia, G. Di Fabbrizio, S. Vaibhav, and A. Datta, "A content-based recommender system for e-commerce o ers and coupons," 2017.

[3] M. Azizi and H. Do, "A collaborative filtering recommender system for test case prioritization in web applications," *arXiv preprint arXiv:1801.06605*, 2018.

[4] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender systems handbook*. Springer, 2015, pp. 77–118.

[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 173–182.

[6] T. Chen, Y. Sun, Y. Shi, and L. Hong, "On sampling strategies for neural network-based collaborative filtering," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 767–776.

[7] J. N. Gross, "Topic based recommender system and method," May 9 2017, uS Patent 9,646,109.

[8] M. Scholz, V. Dorner, G. Schryen, and A. Benlian, "A configuration-based recommender system for supporting e-commerce decisions," *European Journal of Operational Research*, vol. 259, no. 1, pp. 205–215, 2017.

[9] D. Benz, N. M. Ullmann, G. Wetzel, A. Felic, and S. Alexakis, "A weighted hybrid recommender system approach for product configuration," *CERC2017*, p. 95, 2017.

[10] J. He, H. H. Zhuo, and J. Law, "Distributed-representation based hybrid recommender system with short item descriptions," *arXiv preprint arXiv:1703.04854*, 2017.

[11] C. C. Aggarwal, "Content-based recommender systems," in *Recommender Systems*. Springer, 2016, pp. 139–166.

[12] Z. Huang, H. Chen, and D. Zeng, "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 116–142, 2004.

[13] H. Amiri, P. Resnik, J. Boyd-Graber, and H. Daumé III, "Learning text pair similarity with context-sensitive autoencoders," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 1882–1892.

[14] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering," in *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010, pp. 79–86.

[15] H. T. Nguyen, M. Wistuba, J. Grabocka, L. R. Drumond, and L. Schmidt-Thieme, "Personalized for tag recommendation," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2017, pp. 186–197.

[16] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1235–1244.

[17] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 111–112.

[18] S. Wu, W. Ren, C. Yu, G. Chen, D. Zhang, and J. Zhu, "Personal recommendation using deep recurrent neural networks in netease," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 1218–1229.

[19] C. Chen, X. Meng, Z. Xu, and T. Lukasiewicz, "Location-aware personalized news recommendation with deep semantic analysis," *IEEE Access*, vol. 5, pp. 1624–1638, 2017.

[20] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang, "Irgan: A minimax game for unifying generative and discriminative information retrieval models," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2017, pp. 515–524.

[21] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Irish conference on artificial intelligence and cognitive science*. Springer, 2009, pp. 188–197.

[22] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 502–511.

[23] Y.-D. Kim and S. Choi, "Nonnegative tucker decomposition," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[24] E. Frolov and I. Oseledets, "Tensor methods and recommender systems," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 3, 2017.