Joint Optimization of Tree-based Index and Deep Model for Recommender Systems

Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, Kun Gai
Alibaba Group
{zhuhan.zh,daqing.cdq,ziru.xzr,pengye.zpy,yushi.lx,jay.hj,lihan.lh,xiyu.xj,jingshi.gk}@alibaba-inc.com

ABSTRACT

Large-scale industrial recommender systems are usually confronted with computational problems due to the enormous corpus size. To retrieve and recommend the most relevant items to users under response time limits, resorting to an efficient index structure is an effective and practical solution. Tree-based Deep Model (TDM) for recommendation [39] greatly improves recommendation accuracy using tree index. By indexing items in a tree hierarchy and training a user-node preference prediction model satisfying a max-heap like property in the tree, TDM provides logarithmic computational complexity w.r.t. the corpus size, enabling the use of arbitrary advanced models in candidate retrieval and recommendation.

In tree-based recommendation methods, the quality of both the tree index and the trained user preference prediction model determines the recommendation accuracy for the most part. We argue that the learning of tree index and user preference model has interdependence. Our purpose, in this paper, is to develop a method to jointly learn the index structure and user preference prediction model. In our proposed joint optimization framework, the learning of index and user preference prediction model are carried out under a unified performance measure. Besides, we come up with a novel hierarchical user preference representation utilizing the tree index hierarchy. Experimental evaluations with two large-scale real-world datasets show that the proposed method improves recommendation accuracy significantly. Online A/B test results at Taobao display advertising also demonstrate the effectiveness of the proposed method in production environments.

CCS CONCEPTS

• **Computing methodologies** → **Classification and regression trees**; **Neural networks**; • **Information systems** → **Recommender systems**;

KEYWORDS

Joint Optimization, Hierarchical Model, Recommender Systems

1 INTRODUCTION

Recommendation has become an increasingly popular means to help users acquire information from content providers. Personalized recommendation methods have been extensively studied and adopted in various kinds of applications like video streaming [7, 9], news recommendation [25] and e-commerce [39].

Recommendation problem is basically to retrieve a set of most relevant or preferred items for each user request from the entire corpus. In the practice of large-scale recommendation, the algorithm design should strike a balance between accuracy and efficiency. In corpus with tens or hundreds of millions of items, methods that need to linearly scan each item's preference score for each single user request are not computationally tractable. To solve the computational problem, index structure is commonly used to accelerate the retrieval process. In early recommender systems, item-based collaborative filtering (item-CF) along with the inverted index is a popular solution to overcome the calculation barrier [22]. In item-CF based systems, the pre-calculated item similarity is used to build an inverted index, in which items that most similar to user's historical behaviors could be retrieved quickly and precisely. However, the scope of candidate set is limited, because only those items similar to user's historical behaviors can be ultimately recommended.

In recent days, vector representation based methods like matrix factorization [19, 29], factorization machine [28] and deep learning models [2, 7, 25] have been actively researched. This kind of methods can learn user and item's vector representations, the inner-product of which represents user-item preference. For systems that use vector representation based methods, the recommendation set generation is equivalent to the k-nearest neighbor (kNN) search problem. Quantization-based index [17, 23] for approximate kNN search is widely adopted to accelerate the retrieval process. However, in the above solution, the vector representation learning and the kNN search index construction are optimized towards different objectives individually. The vector representation learning aims to minimize the estimation error of user-item preference, while the index construction usually minimizes the quantization error. The divergence between these two objectives leads to suboptimal vector representations and index structure [4]. An even more important problem is that the dependence on vector kNN search index requires an inner-product form of user preference modeling, which limits the model capability [13]. For example, models like Deep Interest Network [37], Deep Interest Evolution Network [36] and xDeepFM [21], which have been proven to be effective in user preference prediction, could not be used to generate candidates in recommendation.

In order to break the inner-product form limitation and make arbitrary advanced user preference models computationally tractable to retrieve candidates from the entire corpus, our previous work Tree-based Deep Model (TDM) [39] creatively uses tree structure as index and greatly improves the recommendation accuracy. TDM uses a tree hierarchy to organize items, and each leaf node in the tree corresponds to an item. Like a max-heap, TDM assumes that each user-node preference is the largest one among the node's all children's preferences. In the training stage, a user-node preference prediction model is trained to fit the max-heap like preference

distribution. Unlike vector kNN search based methods where the index structure requires an inner-product form of user preference modeling, there is no restriction on the form of preference model in TDM. And in prediction, preference scores given by the trained model are used to perform layer-wise beam search in the tree index to retrieve the candidate items. The time complexity of beam search in tree index is logarithmic w.r.t. the corpus size without restriction on model structures, which is a prerequisite to make advanced user preference models feasible to retrieve candidates in recommendation.

The index structure plays different roles in kNN search based methods and tree-based methods. In kNN search based methods, the user and item's vector representations are learnt first, and the vector search index is built then. While in tree-based methods, the tree index's hierarchy also affects the retrieval model training. Therefore, how to learn the tree index and user preference model jointly is an important problem. Tree-based method is also an active research topic in literature of extreme classification [1, 6, 8, 11, 26, 27, 32], which is sometimes considered as the same with recommendation [15, 26]. In the existing tree-based methods, the tree structure is learnt for a better hierarchy in the sample or label space. However, the objective of sample or label partitioning task in the tree learning stage is not fully consistent with the ultimate target, i.e., accurate recommendation. The inconsistency between objectives of index learning and prediction model training leads the overall system to a suboptimal status. To address this challenge and facilitate better cooperation of tree index and user preference prediction model, we focus on developing a way to simultaneously learn the tree hierarchy and user preference prediction model by optimizing a unified performance measure. The main contributions of this paper are summarized as follows:

- We propose a joint optimization framework to learn the tree structure and user preference prediction model in tree-based recommendation, where a unified performance measure, i.e., the accuracy of user preference prediction is optimized.
- We demonstrate that the proposed tree structure learning algorithm is equivalent to the weighted maximum matching problem of bipartite graph, and give an approximate algorithm to learn the tree structure.
- We propose a novel method that makes better use of tree index to generate hierarchical user representation, which can help learn more accurate user preference prediction model.
- We show that both the tree structure learning and hierarchical user representation can improve recommendation accuracy. These two modules can even mutually improve each other to achieve more significant performance promotion.

The remainder of this paper is organized as follows: in Section 2, we will compare some large-scale recommendation methods to show their differences. In Section 3, we firstly give a brief introduction to our previous work TDM to make this paper self-contained, and then describe the proposed joint learning method in detail. In Section 4, experimental results of both offline comparison and online A/B test are given to show the effectiveness of proposed methods. At last, we give a conclusion of our work in Section 5.

2 RELATED WORK

In real-world applications, the recommendation process usually has two stages: candidate generation and ranking [9, 38, 39]. Model-based large-scale recommendation methods are usually confronted with computational restrictions in the candidate generation stage. To overcome the calculation barrier, there are mainly three kinds of approaches: 1) Pre-calculate item or user similarities and use inverted index to accelerate the retrieval [22]; 2) Convert user preference to distance of embedding vectors, and use approximate kNN search in retrieval [7]; 3) Use tree or ensemble of trees to perform efficient retrieval [39].

Industrial recommender systems typically adopt vector kNN search to achieve fast retrieval, e.g., YouTube video recommendation [2, 7], Yahoo news recommendation [25] and extensions that use recurrent neural network to model user behavior sequence [14, 31, 33]. Such approaches use either traditional deep neural network (DNN) or recurrent neural network (RNN) to learn user and item's embedding representations based on various user behavioral and contextual data. However, due to the dependence of approximate kNN search index structures in retrieval, user preference models that use attention network or cross features [5, 36, 37] are challenging to be applied.

Tree-based methods are also studied and adopted in real-world applications. Label Partitioning for Sublinear Ranking (LPSR) [32] uses k-means clustering with data points' features to learn the tree hierarchy and then assign labels to leaf nodes. In the prediction stage, the test sample is passed down along the tree to a leaf node according to its distance to each node's cluster center, and the 1-vs-All base classifier is used to rank all labels belonged to the retrieved leaf node. Partitioned Label Trees (Parabel) [26] also use recursive clustering to build tree hierarchy, but the tree is built to partition the labels according to label similarities. Multi-label Random Forest (MLRF) [1] and FastXML [27] learn an ensemble of sample partitioning trees (a forest), and a ranked list of the most frequent labels in all the leaf nodes retrieved from the forest is returned in prediction. MLRF optimizes the Gini index when splitting nodes, and FastXML optimizes a combined loss function including a binary classification loss and a label ranking loss. In all the above methods, the tree structure keeps unchanged in training and prediction once built, which is hard to completely adapt the retrieval model dynamically.

Our previous work TDM [39] introduces a tree-based model for large-scale recommendation differentiated from existing tree-based methods with a max-heap like user-node preference formulation. In TDM, tree is used as a hierarchical index [20], and an attention model [37] is trained to predict user-node preference. Different from most tree-based methods where non-leaf nodes are used to route decision-making to leaves, TDM explicitly formulates user-node preference for all the nodes to facilitate hierarchical beam search in the tree index. Despite achieving remarkable progress, the joint optimization problem of index and model is not well solved yet as that the proposed alternatively learning method of model and tree has different objectives.

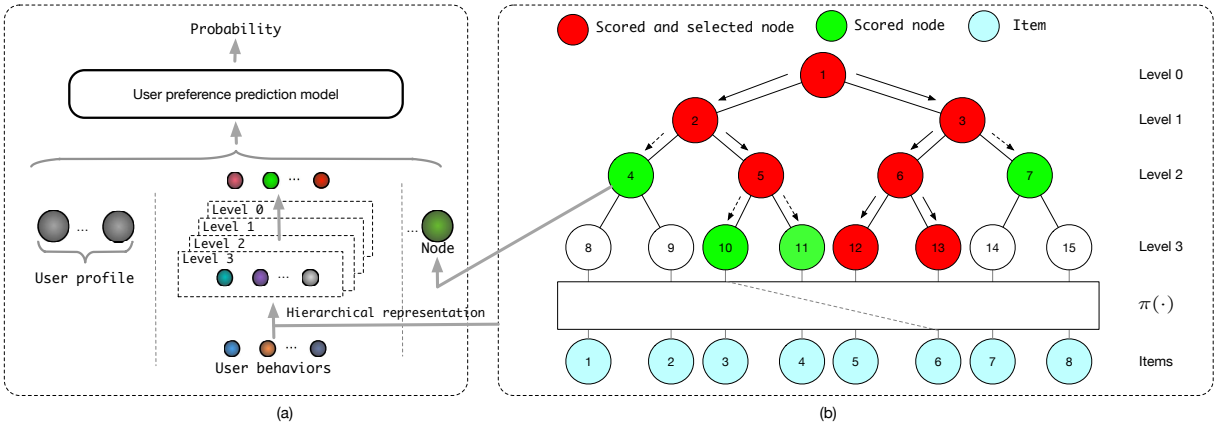


Figure 1: Tree-based deep recommendation model. (a) User preference prediction model. We firstly hierarchically abstract the user behaviors with nodes in corresponding layers. Then the abstract user behaviors and the target node together with the other feature such as the user profile are used as the input of the model. (b) Tree hierarchy. Each item is firstly assigned to a different leaf node with a projection function $\pi(\cdot)$. Red nodes (items) in the leaf level are selected as the candidate set.

3 JOINT OPTIMIZATION OF TREE-BASED INDEX AND DEEP MODEL

In this section, we firstly give a brief review of TDM [39]. TDM uses a tree hierarchy as index and allows arbitrary advanced deep model as user preference prediction model in recommendation. Then we propose the joint learning framework of the tree-based index and deep model. It alternatively optimizes the index and prediction model under a global loss function. A greedy-based tree learning algorithm is proposed to optimize the index. In the last subsection, we specify the hierarchical user preference representation used in model training.

3.1 Tree-based Deep Recommendation Model

A recommender system needs to return a candidate set containing items that a user has interests in from the item corpus. In practice, how to make effective and efficient retrieval from a large item corpus is a challenging problem. TDM uses a tree as index and creatively proposes a max-heap like probability formulation on the tree, where the user preference for each non-leaf node n in level l is derived as:

$$p^{(l)}(n|u) = \frac{\max_{n_c \in \{n's \text{ children in level } l+1\}} p^{(l+1)}(n_c|u)}{\alpha^{(l)}} \quad (1)$$

where $p^{(l)}(n|u)$ is the ground truth probability that the user u prefers the node n. $\alpha^{(l)}$ is a layer normalization term. The above formulation means that the ground truth user-node probability on a node equals to the maximum user-node probability of its children divided by a normalization term. Therefore, the top-k nodes in level l must be contained in the children of top-k nodes in level $l-1$ and the retrieval for top-k leaf items can be restricted to top-k nodes in each layer without losing the accuracy. Based on this, TDM turns the recommendation task into a hierarchical retrieval problem. By a top-down retrieval process, the candidate items are selected gradually from coarse to detailed. The candidate generating process of TDM is shown in Fig 1.

Each item in the corpus is firstly assigned to a leaf node of a tree hierarchy $\mathcal{T}$. The non-leaf nodes can be seen as a coarser abstraction of their children. In retrieval, the user information combined with the node to score is firstly vectorized to a user preference representation as the input of a deep neural network $\mathcal{M}$ (e.g. fully connected networks). Then the probability that the user is interested in the node is returned by $\mathcal{M}$, as shown in Fig 1(a). While retrieving for the top-k items (leaf nodes), a top-down beam search strategy is carried out level by level, as shown in Fig1(b). In level l, only the children of nodes with top-k probabilities in level $l-1$ are scored and sorted to pick k candidate nodes. This process continues until k leaf items are reached.

With tree index, the overall retrieval complexity for a user request is reduced from linear to logarithmic w.r.t. the capacity of item corpus without restrictions on the preference model structure. These make TDM break the inner-product form of user preference modeling restriction brought by vector kNN search index and enable arbitrary advanced deep models to retrieve candidates from the entire corpus, which greatly raises the recommendation accuracy.

3.2 Joint Optimization Framework

According to the retrieval process, the recommendation accuracy of TDM is determined by the quality of the user preference model $\mathcal{M}$ and tree index $\mathcal{T}$. Given n pairs of positive training data (u_i, c_i), which means the user u_i is interested in the target item c_i, $\mathcal{T}$ determines which non-leaf nodes $\mathcal{M}$ should select to achieve c_i for u_i. Instead of separately learning $\mathcal{M}$ and $\mathcal{T}$ as previous and related works, we propose to jointly learn $\mathcal{M}$ and $\mathcal{T}$ with a global loss function. As we will see in experiments, jointly optimizing $\mathcal{M}$ and $\mathcal{T}$ could improve the ultimate recommendation accuracy.

Algorithm 1: Joint learning framework of the tree index and deep model

Input: Loss function $\mathcal{L}(\theta, \pi)$, initial deep model $\mathcal{M}$ and initial tree $\mathcal{T}$

1: **for** $t = 0, 1, 2 \ldots$ **do**
2: Solve $\min_\theta \mathcal{L}(\theta, \pi)$ by optimizing the model $\mathcal{M}$.
3: Solve $\min_\pi \mathcal{L}(\theta, \pi)$ by rebuilding the tree hierarchy with Algorithm 2
4: **end for**

Output: Learned model $\mathcal{M}$ and tree $\mathcal{T}$

Denote $p(\pi(c_i))|u_i; \pi)$ as user u's preference probability over leaf node $\pi(c_i)$ given a user-item pair (u_i, c_i), where $\pi(\cdot)$ is a projection function that projects an item to a leaf node in $\mathcal{T}$. Note that the projection function $\pi(\cdot)$ actually determines the item hierarchy in the tree, as shown in Fig 1(b). The model $\mathcal{M}$ is used to estimate and output the user-node preference $\hat{p}(\pi(c_i))|u_i; \theta, \pi)$, given θ as model parameters. If the pair (u_i, c_i) is a positive sample, we have the ground truth preference $p(\pi(c_i))|u_i; \pi) = 1$ following the multi-class setting [2, 7]. According to the max-heap property, the user preference probability of all $\pi(c_i)$'s ancestor nodes, i.e., $\{p(b_j(\pi(c_i))|u_i; \pi)\}_{j=0}^{l_{max}}$ should also be 1, in which $b_j(\cdot)$ is the projection from a node to its ancestor node in level j and l_{max} is the max level in $\mathcal{T}$. To fit such a user-node preference distribution, the global loss function is formulated as

$$\mathcal{L}(\theta, \pi) = -\sum_{i=1}^{n} \sum_{j=0}^{l_{max}} \log \hat{p}\left(b_j(\pi(c_i))|u_i; \theta, \pi\right). \tag{2}$$

where we sum up the negative logarithm of predicted user-node preference probability on all the positive training samples and their ancestor user-node pairs as the global empirical loss.

Since optimizing the projection function $\pi(\cdot)$ is a combinational optimization, it can hardly be simultaneously optimized with θ using gradient-based algorithms. To conquer this, we propose a joint learning framework as shown in Algorithm 1. It alternatively optimizes the loss function (2) with respect to the user preference model and the tree hierarchy. The consistency of the training loss in model training and tree learning promotes the convergence of the framework. Actually, Algorithm 1 surely converges if both the model training and tree learning decrease the value of (2) since $\{\mathcal{L}(\theta_t, \pi_t)\}$ is a decreasing sequence and lower bounded by 0. In model training, $\min_\theta \mathcal{L}(\theta, \pi)$ is to learn a user-node preference model for each layer. Benefiting from the tree hierarchy, $\min_\theta \mathcal{L}(\theta, \pi)$ is converted to learn the user-node preference distribution and therefore arbitrary advanced deep model can be used, which can be solved by popular optimization algorithms for neural networks such as SGD[3], Adam[18]. In the normalized user preference setting [2, 7], since the number of nodes increases exponentially with the node level, Noise-contrastive estimation[10] is used to estimate $\hat{p}\left(b_j(\pi(c_i))|u_i; \theta, \pi\right)$ to avoid calculating the normalization term by sampling strategy. The task of tree learning is to solve $\min_\pi \mathcal{L}(\theta, \pi)$ given θ, which is a combinational optimization problem. Actually, given the tree structure, $\min_\pi \mathcal{L}(\theta, \pi)$ equals to find the optimal

Algorithm 2: Tree learning algorithm

Input: Gap d, max tree level $l_{\max}$, original projection $\pi_{old}(\cdot)$
Output: Optimized projection $\pi_{new}(\cdot)$

1: Set current level $l = d$, initialize $\pi_{new}(\cdot) = \pi_{old}(\cdot)$
2: **while** True **do**
3: **for** each node n_i in level $l - d$ **do**
4: Denote C_{n_i} as the item set, $\forall c \in C_{n_i}, b_{l-d}(\pi_{new}(c)) = n_i$
5: Find a projection $\pi'(\cdot)$ that maximize $\sum_{c \in C_{n_i}} \mathcal{L}^{l-d+1,l}_{c,\pi'(c_i)}$, s.t. $b_{l-d}(\pi'(c)) = n_i, \forall c \in C_{n_i}$
6: Update $\pi_{new}(\cdot)$. Set $\pi_{new}(c) = \pi'(c), \forall c \in C_{n_i}$
7: **end for**
8: **if** $l == l_{max}$ **then**
9: Break the loop
10: **end if**
11: **if** $l + d > l_{max}$ **then**
12: $d = l_{max} - l$
13: **end if**
14: $l = l + d$
15: **end while**

matching between items in the corpus C and the leaf nodes of $\mathcal{T}$. Furthermore, we have [1]

Remark 1. *$\min_\pi \mathcal{L}(\theta, \pi)$ is essentially an assignment problem to find a maximum weighted matching on a weighted bipartite graph.*

Proof. Suppose the k-th item c_k is assigned to the m-th leaf node n_m, i.e. $\pi(c_k) = n_m$, the following weight value can be computed:

$$\mathcal{L}_{c_k, n_m} = \sum_{(u,c) \in \mathcal{A}_k} \sum_{j=0}^{l_{max}} \log \hat{p}\left(b_j(\pi(c))|u; \theta, \pi\right) \tag{3}$$

where $\mathcal{A}_k$ contains all positive sample pairs (u, c) that the target item c is c_k.

If we take leaf nodes in $\mathcal{T}$ and items in corpus C as vertices and the full connection between leaf nodes and items as edges, we can construct a weighted bipartite graph V with $\mathcal{L}_{c_k, n_m}$ as the weight of edge between c_k and n_m. Furthermore, we can see that each assignment $\pi(\cdot)$ between items and leaf nodes equals a matching of V. Given an assignment $\pi(\cdot)$, the total loss (2) can be computed by

$$\mathcal{L}(\theta, \pi) = -\sum_{i=1}^{|C|} \mathcal{L}_{c_i, \pi(c_i)},$$

where $|C|$ is the corpus size. Therefore, $\min_\pi \mathcal{L}(\theta, \pi)$ equals to find the maximum weighted matching of V. □

Traditional algorithms for assignment problems such as the classic Hungarian algorithm are hard to apply for large corpus because of their high complexity. Even for the simplest greedy algorithm that greedily chooses the unassigned pair (c_k, n_m) with the largest weight $\mathcal{L}_{c_k, n_m}$, a big weight matrix needs to be computed and

[1] For convenience, we assume $\mathcal{T}$ is a given complete binary tree. It is worth to mention that the proposed algorithm can be naturally extended to multi-way trees.

stored in advance, which is not acceptable. To conquer this issue, we propose a segmented tree learning algorithm.

Instead of assigning items directly to leaf nodes, we assign the items every d levels from top to bottom. Denote the partial weight of $\mathcal{L}_{c_k,\pi(c_k)}$ from level s to level d given projection function $\pi(\cdot)$ as

$$\mathcal{L}^{s,d}_{c_k,\pi(c_k)} = \sum_{(u,c)\in\mathcal{A}_k} \sum_{j=s}^{d} \log \hat{p}\left(b_j(\pi(c_k))|u;\theta,\pi\right).$$

We firstly find an assignment to maximize $\sum_{i=1}^{|C|} \mathcal{L}^{1,d}_{c_i,\pi(c_i)}$ w.r.t. the projection function $\pi(\cdot)$, which is equivalent to assign all the items to nodes in level d. For a complete binary tree $\mathcal{T}$ with max level l_{max}, each level d node is assigned with no more than $2^{l_{max}-d}$ items. This is also a maximum matching problem which can be efficiently solved by a greedy algorithm, since the number of possible locations for each item is largely decreased if d is well chosen (e.g. for $d = 7$, the number is $2^d = 128$). Keeping each item c's corresponding ancestor node in level d (which is $b_d(\pi(c))$) unchanged, we then successively maximize the next d levels. The recursion stops until each item is assigned to a leaf node. The proposed algorithm is detailed in Algorithm 2.

In line 5 of Algorithm 2, we use a greedy algorithm with rebalance strategy to solve the sub-problem. Each item $c \in C_{n_i}$ is firstly assigned to the child of n_i in level l with largest weight $\mathcal{L}^{l-d+1,l}_{c,\cdot}$. Then, to guarantee that each child is assigned with no more than $2^{l_{max}-l}$ items, a rebalance process is applied. To promote the stability of tree learning and facilitate the convergence of the whole framework, for nodes that have more than $2^{l_{max}-l}$ items, we keep those items that have the same assignment in level l with the former iteration (i.e., $b_l(\pi'(c)) == b_l(\pi_{old}(c))$) in priority. The other items assigned to the node are sorted in descending order of weight $\mathcal{L}^{l-d+1,l}_{\cdot,n}$, and the exceeded part of items are moved to other nodes that still have redundant space, according to the descending order of each item's weight $\mathcal{L}^{l-d+1,l}_{c,\cdot}$. Algorithm 2 helps us avoid storing a single big weight matrix. Furthermore, each sub-task can run in parallel to further improve the efficiency.

3.3 Hierarchical User Preference Representation

As shown in Section 3.1, TDM is a hierarchical retrieval model to generate the candidate items hierarchically from coarse to detailed. In retrieval, a top-down beam search is carried out levelly through the tree index by the user preference prediction model $\mathcal{M}$. Therefore, $\mathcal{M}'s$ task in each level are heterogeneous. Based on this, a layer-specific input of $\mathcal{M}$ is necessary to raise the recommendation accuracy.

A series of related work [9, 19, 22, 35, 37–39] has shown that the user's historical behaviors play a key role in predicting the user's interests. Besides, since each item in user behaviors is a one hot ID feature, a common way in the generation of the deep model's input is firstly embedding each item into a continuous feature space. Based on the fact that a non-leaf node is an abstraction of its children in the tree hierarchy, given a user behavior sequence $\boldsymbol{c} = \{c_1, c_2, \cdots, c_m\}$ where c_i is the i-th item the user interacts, we propose to use $\boldsymbol{c}^l = \{b_l(\pi(c_1)), b_l(\pi(c_2)), \cdots, b_l(\pi(c_m))\}$ together with the target node and other possible features such as the user profile to generate the input of $\mathcal{M}$ in layer l to predict the user-node preference, as shown in Fig 1(a). In this way, the ancestor nodes of items the user interacts are used as the abstract user's behaviors, with which we replace the original user behavior sequence in training $\mathcal{M}$ for the corresponding layer. Generally, the hierarchical user preference representation brings two main benefits:

1 **Layer independence**. As a common way, shared item embeddings between layers will bring noises in training $\mathcal{M}$ as the user preference prediction model for different layers because the targets differ for different layers. An explicit way to solve this is to attach an item with an independent embedding for each layer to generate the input of $\mathcal{M}$. However, this will greatly increase the number of parameters and make the system hard to optimize and apply. The proposed abstract user behaviors generate the input of $\mathcal{M}$ with node embeddings in the corresponding layer and achieve layer independence in training without increasing the number of parameters.

2 **Precise description**. $\mathcal{M}$ generates the candidate items hierarchically through the tree index. With the increase of retrieval level, the candidate nodes in each level describe the ultimate recommended items from coarsely to precisely until the leaf level is reached. The proposed hierarchical user preference representation grasps the nature of the retrieval process and gives a precise description of user behaviors with nodes in corresponding layer, which promotes the predictability of user preference by reducing the confusion brought by too detailed or coarse description. For example, $\mathcal{M}$'s task in upper layers is to coarsely select a candidate set and the user behaviors are also coarsely described with homogeneous node embeddings in the same upper layers in training and prediction.

4 EXPERIMENTAL STUDY

We study the performance of our proposed method in this section both offline and online. In offline experiments, we use two large-scale real-world datasets to evaluate different methods: Amazon Books dataset [12, 24] and UserBehavior dataset [39]. We firstly compare the overall performance of the proposed method with other existing recommendation models to show the effectiveness of the joint learning framework. And then, ablation study results are given to help comprehend how each part of the framework works in detail. At last, we evaluate the proposed framework in Taobao display advertising platform with real online traffic.

4.1 Datasets

The offline experiments are conducted in two large-scale real-world datasets: 1) user-book review dataset from Amazon; 2) user-item behavior dataset from Taobao called UserBehavior. The details are as follows:

Amazon Books[2]**:** This dataset is made up of product reviews from Amazon. Here we use its largest subset Books and only keep users who have reviewed no less than 10 books. Each review record forms a user-book pair, with the format of user ID, book ID, rating and timestamp.

[2]http://jmcauley.ucsd.edu/data/amazon/

UserBehavior[3]**:** It's a subset of Taobao user behavior data, containing about 1 million randomly sampled users who have behaviors from November 25 to December 03, 2017. Similar to Amazon Books, only users with at least 10 behaviors are kept. Each user-item behavior consists of user ID, item ID, item's unique category ID, behavior type and timestamp. All behavior types are treated equal in our experiments.

Table 1 summarizes the details of the above two datasets after preprocessing.

	Amazon Books	UserBehavior
# of users	294,739	969,529
# of items	1,477,922	4,162,024
# of categories	3,700	9,439
# of records	8,654,619	100,020,395

Table 1: Details of the two datasets after preprocessing. One record is a user-item pair that represents user feedback.

4.2 Compared Methods and Experiment Settings

To evaluate the performance of the proposed framework, we compare the following methods:

- **Item-CF** [30] is a basic collaborative filtering method and is widely used for personalized recommendation especially for large-scale corpus [22].
- **YouTube product-DNN** [7] is a practical method used in YouTube video recommendation. It's the representative work of vector kNN search based methods. The inner-product of the learnt user and item's vector representation reflects the preference. And we use the exact kNN search to retrieve candidates in our experiments.
- **TDM** [39] is the tree-based deep model for recommendation. It enables arbitrary advanced models to retrieve user interests using the tree index. We use the proposed DNN version of TDM without tree learning.
- **TDM-A** is a variant of TDM without tree index. The only difference is that it directly learns a user-item preference model and linearly scan all items in prediction to retrieve the top-k candidates. TDM-A is not computationally tractable in online system but a strong baseline in offline comparison.
- **JTM** is the proposed joint learning framework of the tree index and user preference prediction model.

We follow the settings of TDM [39] to split the dataset. Considering the user amount of two datasets, we randomly sample $5,000$ disjoint users to create Amazon Books' validation set and testing set, while $10,000$ disjoint users are selected as UserBehavior's validation and testing set each. Other users in two datasets compose training set accordingly. For each user in validation and testing set, we take the first half of behaviors along the time line as known features and the latter half as ground truth.

We implement YouTube product-DNN in Alibaba's deep learning platform X-DeepLearning (XDL) and the source code is given [4]. TDM, TDM-A and JTM are also implemented in XDL [5], and we use the same user preference prediction model for them. The user preference model is a three-layer plain-DNN, each layer of which has 128, 64 and 24 hidden units respectively with PReLU [34] activation function. For all compared methods except Item-CF, we use the same user behavior feature as input. Each user behavior sequence has at most 69 user-item pairs. To utilize the sequential information, input user behaviors are divided into 10 time windows in time order. The ultimate user feature is the concatenation of each time window's average item embedding vector. YouTube product-DNN uses the inner-product of learnt user and candidate item's vector representations to reflect user preference, while other methods compute user-item preference with DNN using the concatenation of user feature and candidate item's embedding as input. We deploy negative sampling for all methods except Item-CF and use the same negative sampling ratio. One implicit feedback has 100 negative samples in Amazon Books and 200 in UserBehavior.

TDM and JTM requires an initial tree in advance of training process. Amazon Books uses a random tree, in which items are randomly arranged in the leaf layer, since there's no finer categories under books for about 80.7% of books. By taking advantage of item-category relation of UserBehavior, a category tree can be created, where items from the same category aggregate in the leaf layer. The tree learning layer gap d is set to 7 in all experiments that have joint optimization.

Precision, Recall and F-Measure are three general metrics and we use them to evaluate the performance of different methods. For a user u, suppose $\mathcal{P}_u$ ($|\mathcal{P}_u| = M$) is the recalled set and $\mathcal{G}_u$ is the ground truth set. The equations of three metrics are

$$\text{Precision@}M(u) = \frac{|\mathcal{P}_u \cap \mathcal{G}_u|}{|\mathcal{P}_u|}, \ \text{Recall@}M(u) = \frac{|\mathcal{P}_u \cap \mathcal{G}_u|}{|\mathcal{G}_u|}$$

$$\text{F-Measure@}M(u) = \frac{2 * \text{Precision@}M(u) * \text{Recall@}M(u)}{\text{Precision@}M(u) + \text{Recall@}M(u)}$$

4.3 Comparison Results

Table 2 shows the quantitative results of all methods in two datasets. It clearly shows that our proposed JTM outperforms other baselines in all metrics. Compared with the previous best model TDM-A in two datasets, JTM achieves 45.3% and 8.1% recall lift in Amazon Books and UserBehavior respectively.

As mentioned in Section 4.2, though computationally intractable in online system, TDM-A is a significantly strong baseline for offline comparison and the theoretical upper-bound for YouTube product-DNN and tree-based models (TDM and JTM) on the condition of similar DNN user preference model. Comparison results of TDM-A and other methods give insights in many aspects.

Firstly, results of YouTube product-DNN and TDM-A indicate the limitation of inner-product form. Evidently, these two methods adopt the same input. The difference is that YouTube product-DNN give the rank list with inner-product of learnt user and item's vector, while TDM-A computes the score with DNN using user and item's vector concatenation as input. Such a slight change brings apparent improvement, which verifies the effectiveness of the neural network over inner-product form.

[3] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1

[4] https://github.com/alibaba/x-deeplearning/tree/master/xdl-algorithm-solution/TDM/script/tdm_u

[5] https://github.com/alibaba/x-deeplearning/tree/master/xdl-algorithm-solution/TDM/script/tdm_u

Method	Amazon Books			UserBehavior		
	Precision@200	Recall@200	F-Measure@200	Precision@200	Recall@200	F-Measure@200
Item-CF	0.52%	8.18%	0.92%	1.56%	6.75%	2.30%
YouTube product-DNN	0.53%	8.26%	0.93%	2.25%	10.15%	3.36%
TDM	0.51%	7.58%	0.89%	2.23%	10.84%	3.40%
TDM-A	0.56%	8.57%	0.98%	2.81%	13.45%	4.23%
JTM	**0.79%**	**12.45%**	**1.38%**	**3.06%**	**14.54%**	**4.61%**

Table 2: Comparison results of different methods in Amazon Books and UserBehavior.

Next, TDM performs worse than TDM-A as a result of tree hierarchy. Tree hierarchy takes effect in both training and prediction process. User-node samples are generated along the tree to fit max-heap like preference distribution, and layer-wise beam search is deployed in the tree index when prediction. Without a well-defined tree hierarchy, user preference prediction model may converge to a suboptimal version with confused generated samples, and it's possible to lose targets in the non-leaf layers so that inaccurate candidate sets are returned. Especially in sparse data like Amazon Books, learnt embedding of each node in tree hierarchy is not distinguishable enough so that TDM with a random tree doesn't perform well than other baselines. This phenomenon illustrates the influence of tree and necessity to learn more advanced tree.

By joint learning of tree index and user preference prediction model, JTM outperforms TDM-A on all metrics in two datasets. More precise user preference prediction model and more advanced tree hierarchy are obtained, as well as better item set selection. Hierarchical user preference representation alleviates the data sparsity problem in upper layers, because the feature space of user behavior feature is much smaller while having the same number of samples. And it helps model training in a layer-wise way to reduce the propagation of noises between layers. Besides, tree hierarchy learning makes similar items aggregate in the leaf layer, so that the internal layer models can get training samples with more consistent and unambiguous distribution. Benefited from the above two reasons, a unified optimization of hierarchical user preference model and tree index makes it possible for JTM to provide better item set selection than TDM-A. More specific analysis of each part can be seen in Section 4.4.

4.4 Ablation Study

Hierarchical User Preference Representation. To explore why the proposed hierarchical user preference representation works, we perform additional experiments on three variants of user preference representation in tree-based model in two datasets. Tree-based model samples target nodes from all layers of the tree and uses the concatenation embedding of user behaviors and target node as input. The difference of three variants lies in user behavior features and they utilize a fixed initial tree as explained in Section 4.2 with no tree learning algorithms adopted. More details are as follows:

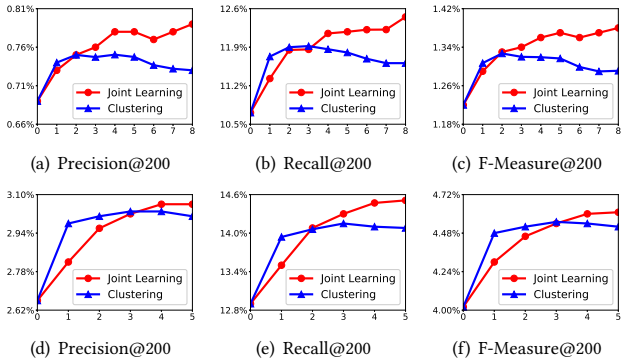
- **TDM** is the basic tree-based model introduced in section 4.2. Each node has only one embedding. When dealing with samples of all layers, user behavior feature for one user is totally the same.
- **JTM-HI** is an advanced version of TDM which uses **layer-independent feature space**. More specifically, the user behavior features are directly mapped to different embedding spaces when training different layers' models. Compared to TDM, the parameter size increases multiple times according to the height of the tree.
- **JTM-H** is TDM with **hierarchical user preference representation**. It simplifies JTM-HI by taking advantage of tree hierarchy. The same as TDM, each node in tree has only one embedding. User behaviors in the leaf layer map to corresponding layers naturally by embeddings of their ancestors in the tree. No more parameters is needed.

Dataset	Method	Metric@200		
		Precision	Recall	F-Measure
Amazon Books	TDM	0.51%	7.58%	0.89%
	JTM-HI	0.59%	8.53%	1.02%
	JTM-H	**0.69%**	**10.71%**	**1.22%**
UserBehavior	TDM	2.23%	10.84%	3.40%
	JTM-HI	2.40%	11.44%	3.62%
	JTM-H	**2.66%**	**12.93%**	**4.02%**

Table 3: Evaluations of hierarchical representation for user preference model in tree-based models in Amazon Books and UserBehavior.

From Table 3, we have several observations. JTM-HI outperforms TDM in both datasets, which proves that the layer-independent feature space indeed reduces the noises brought by sharing embedding space of user behavior feature in all layers of the tree. JTM-H gets higher performance than JTM-HI with less parameters, which demonstrates that hierarchical user preference representation works well. On the one hand, tree hierarchy provides a natural hierarchical representation. Node embedding in the same layer of tree are homogeneous, thus it's easier to capture latent feature cross in the same layer than between leaf and non-leaf layers. On the other hand, with hierarchical user preference representation, the parameter space of user behavior feature shrinks a lot in upper layers, which partially solves the data sparsity problem.

In UserBehavior, the recall metric raises from 10.84% to 11.44% with layer-independent feature, as a result of feature confusion alleviation between layers. Another recall improvement from 11.44% to 12.93% comes from homogeneity and appropriate granularity features inside each layer. The relative improvements are more significant in Amazon Books, because the data sparsity problem is more serious and the random initialized tree introduces more inter-layer noise, which can be well solved by the proposed method.



(a) Precision@200 (b) Recall@200 (c) F-Measure@200

(d) Precision@200 (e) Recall@200 (f) F-Measure@200

Figure 2: Results of iterative joint learning in two datasets. 2(a), 2(b), 2(c) are results in Amazon Books and 2(d), 2(e), 2(f) shows the performance in UserBehavior. The horizontal axis of each figure represents the number of iterations, and the vertical axis denotes the value of corresponding metrics. *Joint Learning* represents results learnt with the proposed tree learning algorithm and *Clustering* indicates the results of using k-means clustering to learn the tree. Two models here both adopt hierarchical user preference representation.

Iterative Joint Learning. Tree hierarchy decides sample generation and search path. A suitable tree would benefit model training and inference a great deal. Fig 2 gives the comparison results of iterative joint learning between clustering-based tree learning algorithm proposed in TDM [39] and the proposed tree learning algorithm.

Obviously, the proposed tree learning algorithm has two merits: 1) It can converge to an optimal tree stably; 2) Tree learning and user preference prediction model training share the same goal, guaranteeing the accuracy of recommendation. From Fig 2, we can see that results increase iteratively on all three metrics. Besides, though clustering method has better results at early iterations, tree learning algorithm makes the model stably converge to a better result through joint learning in both datasets. The above results demonstrate the effectiveness of iterative joint learning. It helps optimize the information maintained in tree hierarchy, thus facilitating training and inference.

Joint Performance of User Preference Prediction Model and Tree Learning. To further study the contribution of each part and their joint performance, we perform some contrast experiments on each part of JTM. Detailed descriptions are as follows:

- **TDM** is the basic tree-based model. It uses a plain-DNN for user preference model and applies no tree learning.
- **JTM-J learns tree hierarchy and user preference prediction model jointly** and iteratively. It adopts the same user preference prediction model as TDM.
- **JTM-H** deploys **hierarchical representation** in user preference prediction model with the fixed initial tree hierarchy.
- **JTM** optimizes user preference prediction model with **hierarchical representations** and tree hierarchy alternatively **in a joint framework**.

The corresponding results[6] of the above variants are in Table 4. Take recall metric as an example. In Amazon Books, the recall increases from 7.58% to 10.71% with hierarchical representation. Limited by the confusion and difficulty from initial feature constitution and the random tree hierarchy, only a slight increase occurs after joint learning of tree hierarchy and the former user preference prediction model. However, it lifts to 12.45% by adopting the joint learning framework of more expressive features and tree learning. A more obvious comparison result can be seen in UserBehavior. Tree learning and hierarchical representation of user preference brings 0.88% (TDM 10.84% → JTM-J 11.72%) and 2.09% (TDM 10.84% → JTM-H 12.93%) absolute gain separately on recall metric. Furthermore, more improvement up to 3.70% TDM 10.84% → JTM 14.54%) absolute recall is achieved by simultaneous optimization of both, which is more than the sum of 0.88% and 2.09%.

Dataset	Method	Metric@200		
		Precision	Recall	F-Measure
Amazon Books	TDM	0.51%	7.58%	0.89%
	JTM-J	0.51%	7.60%	0.89%
	JTM-H	0.69%	10.71%	1.22%
	JTM	**0.79%**	**12.45%**	**1.38%**
UserBehavior	TDM	2.23%	10.84%	3.40%
	JTM-J	2.48%	11.72%	3.73%
	JTM-H	2.66%	12.93%	4.02%
	JTM	**3.06%**	**14.54%**	**4.61%**

Table 4: Performances of joint learning in Amazon Books and UserBehavior.

The above results in Table 4 clearly show the effectiveness of hierarchical representation and tree learning, as well as the joint learning framework. **Evidently, the joint learning of user preference model with hierarchical representation and learnt tree brings more promotion than the arithmetic sum of each single one in all metrics. Thus it's beneficial to optimize user preference model with hierarchical representation and learn tree hierarchy in a unified framework.**

4.5 Online Results

The proposed JTM is also evaluated in production environments. We conduct the experiments in display advertising scenario of *Guess What You Like* column of Taobao App Homepage. We use click-through rate (CTR) and revenue per mille (RPM) to measure the performance, which are the key performance indicators for online display advertising. The equations of online metrics are:

$$\text{CTR} = \frac{\#\text{ of clicks}}{\#\text{ of impressions}}, \quad \text{RPM} = \frac{\text{Ad revenue}}{\#\text{ of impressions}} * 1000$$

In our advertising systems, advertisers bid on plenty of granularities, such as ad clusters, items, shops, etc. Several simultaneously running recommendation approaches in all granularities produce candidate sets and the combination of them are passed to subsequent stages, like CTR prediction [36, 37], ranking [16, 38], etc.

[6]Note that JTM-J and JTM jointly optimize user preference model and tree hierarchy iteratively. Here we only list the converged result of these two methods.

Our baseline is such a combination of all running recommendation methods. To assess the effectiveness of JTM, we deploy JTM to replace Item-CF, which is one of the major candidate-generation approaches in granularity of items in our systems. TDM is evaluated in the same way as JTM. The corpus to deal with contains tens of millions of items. Each comparison bucket has 2% of all online traffic. Under our efforts, we have accomplished the first version of JTM and evaluated its performance online. The results are presented in Table 5.

Method	CTR	RPM
Baseline	0.0%	0.0%
TDM	+5.4%	+7.6%
JTM	+11.3%	+12.9%

Table 5: Online results from Jan 21 to Jan 27, 2019 in *Guess What You Like* column of Taobao App Homepage.

Table 5 reveals the lift on two online metrics. 11.3% growth on CTR exhibits that more precise items have been recommended with JTM. As for RPM, it has a 12.9% improvement, indicating JTM can bring more income for Taobao advertising platform. The experimented JTM only works with basic fully connected network to capture user preference. Under joint optimization framework, more advanced user preference model can achieve more significant performance improvements. Note that TDM is a strong baseline with significant improvement, however JTM still has 5.9% and 5.3% gain in CTR and RPM respectively compared with TDM in exactly the same scenario.

5 CONCLUSION

Recommender system plays a key role in various kinds of applications such as video streaming and e-commerce. In this paper, we propose a joint learning framework of the tree index and user preference prediction model used in tree-based deep recommendation model. The tree index and deep model are alternatively optimized under a global loss function. An efficient greedy algorithm is proposed in tree learning. Besides, a novel hierarchical user preference representation is proposed to make a precise description of user behaviors utilizing the tree hierarchy. Both online and offline experimental results show the advantages of the proposed framework over other related large-scale recommendation models.

REFERENCES

[1] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. 2013. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 13–24.

[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 46–54.

[3] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[4] Yue Cao, Mingsheng Long, Jianmin Wang, Han Zhu, and Qingfu Wen. 2016. Deep Quantization Network for Efficient Image Retrieval.. In *AAAI*. 3457–3463.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[6] Anna E Choromanska and John Langford. 2015. Logarithmic time online multiclass prediction. In *Advances in Neural Information Processing Systems*. 55–63.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *ACM Conference on Recommender Systems*. 191–198.

[8] Hal Daumé III, Nikos Karampatziakis, John Langford, and Paul Mineiro. 2017. Logarithmic time one-against-some. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 923–932.

[9] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 293–296.

[10] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. 297–304.

[11] Lei Han, Yiheng Huang, and Tong Zhang. 2018. Candidates vs. Noises Estimation for Large Multi-Class Classification Problem. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 1890–1899.

[12] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 507–517.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. 173–182.

[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[15] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 935–944.

[16] Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. 2018. Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. *arXiv preprint arXiv:1802.09756* (2018).

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[18] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[20] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 489–504.

[21] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *arXiv preprint arXiv:1803.05170* (2018).

[22] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[23] Ting Liu, Andrew W Moore, Ke Yang, and Alexander G Gray. 2005. An investigation of practical approximate nearest neighbor algorithms. In *Advances in neural information processing systems*. 825–832.

[24] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 43–52.

[25] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1933–1942.

[26] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 993–1002.

[27] Yashoteja Prabhu and Manik Varma. 2014. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 263–272.

[28] Steffen Rendle. 2010. Factorization Machines. In *IEEE International Conference on Data Mining*. 995–1000.

[29] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *International Conference on Neural Information Processing Systems*. 1257–1264.

[30] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*. 285–295.

[31] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 17–22.

[32] Jason Weston, Ameesh Makadia, and Hector Yee. 2013. Label partitioning for sublinear ranking. In *International Conference on Machine Learning*. 181–189.

[33] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. ACM, 495–503.

[34] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853* (2015).

[35] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep Learning based Recommender System: A Survey and New Perspectives. (2017).

[36] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Evolution Network for Click-Through Rate Prediction. *arXiv preprint arXiv:1809.03672* (2018).

[37] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1059–1068.

[38] Han Zhu, Junqi Jin, Chang Tan, Fei Pan, Yifan Zeng, Han Li, and Kun Gai. 2017. Optimized Cost Per Click in Taobao Display Advertising. In *Proceedings of the 23rd ACM SIGKDD Conference*. ACM, 2191–2200.

[39] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-based Deep Model for Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1079–1088.