Forest-based Deep Recommender

Chao Feng
chaofeng@mail.ustc.edu.cn
School of Computer Science and Technology, University of Science and Technology of China
Hefei, China

Defu Lian*
liandefu@ustc.edu.cn
School of Computer Science and Technology, University of Science and Technology of China
Hefei, China

Zheng Liu
zhengliu@microsoft.com
Microsoft Research Asia
Beijing, China

Xing Xie
xing.xie@microsoft.com
Microsoft Research Asia
Beijing, China

Le Wu
lewu.ustc@gmail.com
School of Computer Science and Information Engineering, Hefei University of Technology
Hefei, China

Enhong Chen
cheneh@ustc.edu.cn
School of Computer Science and Technology, University of Science and Technology of China
Hefei, China

ABSTRACT

With the development of deep learning techniques, deep recommendation models also achieve remarkable improvements in terms of recommendation accuracy. However, due to the large number of candidate items in practice and the high cost of preference computation, these methods also suffer from low efficiency of recommendation. The recently proposed tree-based deep recommendation models alleviate the problem by directly learning tree structure and representations under the guidance of recommendation objectives. However, such models have two shortcomings. First, the max-heap assumption in the hierarchical tree, in which the preference for a parent node should be the maximum between the preferences for its children, is difficult to satisfy in their binary classification objectives. Second, the learned index only includes a single tree, which is different from the widely-used multiple trees index, providing an opportunity to improve the accuracy of recommendation.

To this end, we propose a Deep Forest-based Recommender (DeFoRec for short) for an efficient recommendation. In DeFoRec, all the trees generated during training process are retained to form the forest. When learning node representation of each tree, we have to satisfy the max-heap assumption as much as possible and mimic beam search behavior over the tree in the training stage. This is achieved by DeFoRec to regard the training task as multi-classification over tree nodes at the same level. However, the number of tree nodes grows exponentially with levels, making us to train the preference model by the guidance of sampled-softmax technique. The experiments are conducted on real-world datasets, validating the effectiveness of the proposed preference model learning method and tree learning method.

*Corresponding author

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3531980

CCS CONCEPTS

• **Information systems** → **Top-k retrieval in databases**; **Similarity measures**; **Learning to rank**.

KEYWORDS

Recommender system; Forest-based Index; Multi-classification; Sampled Softmax; Efficient Recommendation

ACM Reference Format:
Chao Feng, Defu Lian, Zheng Liu, Xing Xie, Le Wu, and Enhong Chen. 2022. Forest-based Deep Recommender. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3477495.3531980

1 INTRODUCTION

Due to the excessive quantity of daily information, people severely suffer from information overload in the information era. Recommendation is an important means to address information overload by providing a personalized ranking list on a set of information. With the development of deep learning techniques, recommendation techniques also achieve remarkable development and improvements in ranking performance. The widespread application of these techniques has created great economic benefits for various kinds of content providers in industrial companies.

Through the use of deep learning, we not only learn better representations for users, items, and contexts but also provides a more generalized expression for users' preference scores via neural networks than the widely-used inner product in matrix factorization. Both lead to stronger recommendation performance, as shown in models like DIN [50], DIEN [49], NCF [14], CDL [43], and CKE [46]. However, the use of the neural preference function brings online-serving challenges for recommender systems due to the high cost of preference computation. Generally speaking, immediate responses to adaptive recommendations are prerequisites for excellent customer experiences and custom retention. Existing popular and well-performed graph-based indexes, e.g. HNSW [27], quantization-based indexes, e.g. PQ[17], AQ [1] and SCANN [12], and Hash-based indexes, e.g. SGDH [21] are usually built based on inner product or Euclidean distance, such that they are not suitable

for speeding up the recommendation of deep models. In particular, neural networks based preference functions is not usually a valid metric, and difficult to be compatible with Euclidean distance and inner product. Therefore, two items being close in Euclidean or inner-product space may have divergent neural preference scores.

To guarantee the compatibility between search indexes with the neural preference functions, it should be a good solution to learn the search indexes together with the recommendation model under the guidance of recommendation objectives. The representative work is the tree-based deep model (i.e. TDM [53] and its improved version JTM [52]). These models use the balanced tree index, which is constructed by hierarchically clustering item representations from top to down. Given the tree, the top-k ranked items are obtained by layer-wise beam search, which selects the k-largest tree-nodes based on neural preference scores in each level from top to down. In this way, beam search achieves logarithmic computation complexity w.r.t. the number of items. To guarantee the accuracy of beam search, these tree-based deep models rely on the following max-heap assumption: the preference scores of query for a parent node should be the maximum between the preference scores of its children node. For the sake of satisfying the assumption, these tree-based models cast the overall problem as a binary classification problem by treating nodes in the path from the root to positive samples as positive and randomly sampled nodes as negative. However, these tree models suffer from the following two drawbacks. First, the max-heap assumption is not well satisfied by the used binary classification objectives due to the lack of horizontal competition among all tree nodes at the same level. Second, the learned index only includes a single tree, which is different from the widely-used multiple tree index in industrial libraries, such as Annoy. A single tree may easily misuse data around the boundary, which can be corrected by other trees of different structures. Therefore, this drawback also provides an opportunity to improve the accuracy of efficient recommendations and strike a better balance between efficiency and accuracy.

We proposed Deep Forest-based Recommender (DeFoRec for short) for compositing multiple trees. To guarantee that a set of trees can bring improvements over a single tree, these trees should be diversified. In DeFoRec, we keep all the trees and the corresponding preference models generated during the training process goes on to form the forest model. To satisfy the max-heap assumption as much as possible, DeFoRec regards the training task as multi-classification over tree nodes at the same level, which enables horizontal competition among them and mimics beam search behavior in the training stage. However, the number of tree nodes in each level grows exponentially with levels, making the softmax loss suffer from computational challenges. To promote the training efficiency, we resort to sampled softmax for approximation, which only requires a small number of sampled nodes to use. Additionally, we propose a tree learning method that allow us to learn the preference model and tree alternately.

To summarize, we have delivered the following contributions:

- We propose a Deep Forest-based Recommender to composite multiple diversified trees for improving recommendation accuracy while guaranteeing logarithmic time retrieval.
- We regard tree training as a layer-wise multi-classification problem to satisfy the max-heap assumption as much as possible and resort to the sampled softmax loss for efficient optimization.
- we propose a tree learning method that allows us to learn the preference model and tree index alternately.
- We evaluate the proposed methods with several real-world datasets and validate the superiority of DeFoRec to the baselines and the effectiveness of tree learning method.

2 RELATED WORK

We first review recent advances of search efficient recommendations and negative sampling in recommender systems, and then survey recent important techniques for speeding up softmax computation.

2.1 Search-Efficient Recommendation

Search-efficient recommendation relies on building a search index, including LSH [8], inverted index [2, 32], tree index [33, 34][1] and graph index [27] for all items. The recommender system usually uses the inner product for computing preference scores, and the top-k recommendation can be cast into the maximum inner product search (MIPS) problem. The search index is usually constructed based on Euclidean distance and has been extended to the inner product. This extension can be achieved by establishing the relationship between nearest neighbor search and MIPS [3, 31, 40], or learning from either item representations [11, 19, 30, 51] or the raw data directly [23, 26, 28, 47]. With the introduction of deep learning into the recommender system, the preference score function becomes complicated, such that it is challenging to transform from neural ranking to NNS or MIPS . Existing work either directly used metric-based index[42], or learns search index from raw data directly together with recommendation models [24, 52, 53].

2.2 Negative Sampling in RecSys

Negative sampling is an important method to address the negative missing and exposure bias problems and to speed up the convergence of recommender training [18, 22, 25]. It includes static sampling, such as uniform sampling [37] and popularity sampling, and adaptive sampling. The adaptive sampling is context-dependent, whose representation work includes adaptive oversampling [37], rejection sampling [16, 45], clustering-based sampling [22], and dynamic negative sampling [48]. The core idea of these adaptive samplers is that items with larger preference scores should be sampled with higher probability.

2.3 Techniques of Speedup Softmax Computation

In natural language applications, it is very computationally expensive to represent an output distribution over the choice of a word based on the softmax function. To address the efficiency, many approximate algorithms were proposed. For example, hierarchical softmax [29] and lightRNN [20] decomposed the probabilities, and Contrastive Divergence [15] approximated the gradient-based on MCMC. As an alternative, negative sampling is also widely used

[1] It is used in an industrial library ANNOY (https://github.com/spotify/annoy)

Algorithm 1: Beam Search: Layer-wise Retrieval

Input: User u, tree, beam size B and solution size K, the preference model p

1: Result set $A = \Phi$, candidate set $Q = \{root\ node\ n_1\}$;
2: **while** Q is not empty **do**
3: Remove all the leaf nodes from Q and add these leaf nodes into result set A if Q contains leaf nodes;
4: Compute preference $p(n|u)$ for each left node in set Q;
5: Parents={the top-B nodes of Q according to $p(n|u)$};
6: Q={children of node n $|n \in$ Parents};
7: **end while**
8: **return** The top-K items w.r.t. the top-K leaf nodes according to $p(n|u)$, $n \in A$.

in reducing the computational cost of training the models. The representative work includes Noise-Contrastive Estimation [13] with the unigram distribution as a sampler, Generative Adversarial Networks [9, 44] with a neural-network empowered sampler, Self-Contrast Estimator [10] by the model in the immediately preceding epoch as a sampler, self-adversarial negative sampling [41] and Kernel-based sampling [6] with the tree index.

3 TREE-BASED AND FOREST-BASED RECOMMENDERS

In this section, we firstly recall the tree-based recommendation models, i.e. TDM and JTM. Then we elaborate the preference model's training process in TDM and JTM. We introduce how to form the forest model lastly.

3.1 Tree-based index and the deep model

To tackle the top-k retrieval of the most preferred items, Zhu et al. [52, 53] develop a max-heap like tree index which is compatible with any advanced preference models. We reproduce their deep model and the tree-based index in **Figure 1**. In the bottom right of **Figure 1**, the example is a binary tree but any multi-way tree such that each item of corpus is mapped to a leaf node is compatible in fact. The rigorous max-heap tree satisfies the formulation that

$$p^j(n|u) = \frac{\max_{n_c \in node\ n's\ children} p^{(j+1)}(n_c|u)}{\alpha^j} \tag{1}$$

where $p^j(n|u)$ denotes user u's real preference to node n at level j and α^j is a layer-specific normalization term to make the sum of preference scores in level j equals to one. **E.q. (1)** illustrates that the user's preference to a node is proportional to the largest one of the its children's preference scores.

Top-k retrieval in recommendation is to find the k leaf nodes with largest preference scores in the tree. If the real preference $p^j(n|u)$ can be available for any node at each level j for user u, we can find the top-k leaf nodes by only retrieving all the top-k nodes at each layer from top layer to bottom layer. This retrieval process guarantees to find the top-k leaf nodes because that the top-k nodes with largest preference scores at level $j+1$ must belong to the children of top-k nodes with largest preference scores at layer j. The max-heap tree index makes users can obtain the top-k recommendation from coarse clusters to fine clusters until each cluster only contains on leaf node. Furthermore, the time complexity of the retrieval process is logarithmic w.r.t. the corpus size. The layer-wise retrieval process is presented in **Algorithm 1**, where $p(n|u)$ is exactly user u's preference to node n and we omit the superscript (i.e. layer).

However, the real preference $p^j(n|u)$ is unavailable and various kinds of prediction models can be used to estimate the preference. The prediction model used in TDM and JTM can be seen in the left part of **Figure 1**. The inputs of the prediction model are the history of user behaviours (i.e. the items that the user interacted with sorted by interaction time) and the node. The user profile (e.g. age, gender, height and so on) can be used to enhance model if these information is available. The user behaviours are partitioned into several time windows and zero padding is used to make up the time windows if some items are missing. Each time window outputs the weighted average embedding where the wight is calculated by an activation unit. Seeing the activation unit in the top right part of **Figure 1**, the concatenation (of the item embedding, point-wise product between item embedding and node embedding, node embedding) is fed into the MLP and the output is the activation weight. The concatenation (embedding of user profile, each output of time windows, node embedding) is fed into MLP to generate the like and dislike probabilities in preference model. PReLU is used as the activation function and batch normalization is applied to the MLP in preference model.

3.2 Learn the preference model and the tree

In tree-based recommendation systems, we need to learn the preference model (i.e. the neural network model to measure the scores between users and nodes) and the tree (i.e. the bijective mapping relationships between items and the leaf nodes of the tree). We use $\pi(\cdot)$ to denote the bijective mapping between items and leaf nodes in latter content, i.e. $\pi(item_i) = n_{leaf}$ means that the leaf node n_{leaf} represents the item $item_i$. The preference model and the tree can be learned alternately. Concretely, fix the bijective mapping $\pi(\cdot)$ when learning the preference model and fix the preference model when learning the bijective mapping $\pi(\cdot)$. Repeat the alternate learning process until both the two parts converge.

To learn the preference model, TDM and JTM regard the training task as a binary classification problem essentially. Supposing user u has an interaction with an item which is corresponding to a leaf node is n_{leaf}, n_{leaf} and its ancestors are positive nodes with label 1 but all the left nodes are negative nodes with label 0. Then the training loss for user u is

$$\begin{aligned}\mathcal{L}(u, Y_u^+, Y_u^-) = &- \sum_{n \in Y_u^+} y_u(n) \log p(\hat{y}_u(n) = 1|u) \\ &- \sum_{n \in Y_u^-} (1 - y_u(n)) \log p(\hat{y}_u(n) = 0|u).\end{aligned} \tag{2}$$

Y_u^+ and Y_u^- denote the positive node set and negative node set for user u respectively. $y_u(n)$ denotes the label of node n for user u. $p(\hat{y}_u(n) = 1|u)$ and $p(\hat{y}_u(n) = 0|u)$ denote the like and dislike probabilities for user u to node n, which are exactly corresponding to the two outputs of preference model. Using all negative nodes to train the model is unacceptable both in time consuming and memory consumption. Both TDM and JTM draw a negative node

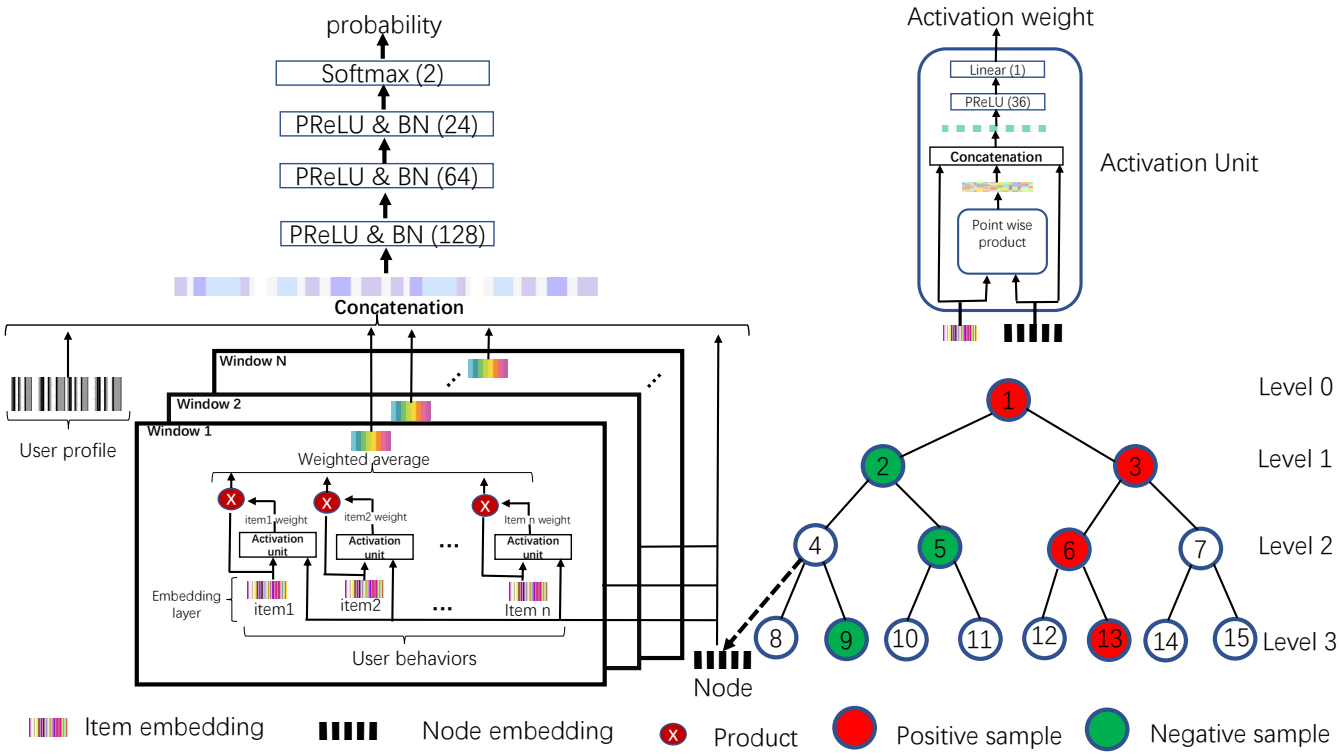


Figure 1: The tree-based deep model in TDM and JTM.

set (denoted as S_u^-) uniformly from all the negative nodes. Then $\mathcal{L}(u, Y_u^+, S_u^-)$ is used as training loss.

Regrading the training task as a binary classification problem leads to three shortcomings. 1).The max-heap assumption is not well satisfied due to the lack of horizontal competition among all tree nodes at the same level. 2). The retrieval is conducted by a layer-wise stream (e.g. **Algorithm 1**) but the training process doesn't include layer-wise mode which can lead to a gap between training and prediction. 3). There are much more negative nodes than positive nodes which can make the negative nodes dominate the training loss. At the same time, the imbalance between negative and positive nodes leads to extra computation burden to find the proper size of sampled negative nodes. To overcome these shortcomings, we regard the preference model learning task as a multi-class classification problem which will be elaborated in latter sections.

Both TDM and JTM utilize the hierarchical clustering to update the tree structure. TDM clusters all the items recursively by k-means using the item embedding until each cluster only contains one item. The recursive clustering process can form a tree structure and the item in the each final cluster is exactly mapped to a leaf node. JTM assigns all the items to each node from top layer to bottom layer such that the sum of the log likelihoods for all user on each layer can be maximized. Interested readers can refer to the content of TDM [53] and JTM [52] for more details about how to learn the trees respectively. In our work, we propose a new tree learning method which is compatible with the our proposed preference model learning mode.

3.3 Forest-based index

The retrieval is only conducted on the last generated tree by the last updated preference model both in TDM and JTM. However, the previous generated trees and preference models may also provide some good candidates. We keep all the generated trees and the corresponding preference models to form the Deep Forest-based Recommender. In prediction, each tree is retrieved by using the corresponding preference model to generate some candidates. After retrieving the whole forest, we rank all the candidates by a user-specified discriminator. The discriminator outputs the user's preference to certain item. For example, the discriminator can be the inner product between user embedding and item embedding or it can be a pre-trained neural collaborative filtering [14], e.g. DIN [50], DIEN [49] and so on. In our experiments, we choose the last updated preference model as the discriminator to rank the all the candidates mapping to the leaf nodes on the last updated tree. Lastly, top-k items are recommended among the ranked items.

4 MULTI-CLASS CLASSIFICATION TRAINING MODE

To address the shortcomings of training mode in TDM and JTM, we develop a layer-wise training mode in this section. Firstly, we regard the training task as a multi-classification problem at each layer. Secondly, we develop a tree-based sampling method which makes the training process efficient.

4.1 Multi-class cross entropy loss

TDM and JTM regard the training task as a binary classification problem and binary cross entropy is used as the training loss. In this way, each node contributes to the training loss individually. However, the layer-wise retrieval is conducted from top layer to bottom layer and we need to compare the candidates with each other node at the same layer. There is a gap between training and prediction in TDM and JTM. We propose the layer-wise training mode which regards the training task as a multi-class classification problem. Concretely speaking, at layer j who consists of $c(j)$ nodes, each node is corresponding to a class. For user u, the positive node with label 1 can be obtained by looking back upon from leaf node to root node like the authors do in TMD and JTM. The remaining $c(j)-1$ nodes are the negative nodes with label 0 w.r.t. to the positive node. We adjust the preference mode in **Figure 1** slightly to fit the multi-class classification problem. We remove the softmax module and make the last layer of the preference model only contains one neuron. After the adjustment, the preference model only outputs one value. For user u, o_i^j represents user u's preference to the i-th node at layer j and p_i^j denotes the probability that the i-th node is positive node at layer j, where

$$p_i^j = \frac{\exp o_i^j}{\sum_{k=1}^{c(j)} \exp o_k^j}. \tag{3}$$

The multi-class cross entropy is used as the training loss at layer j, i.e.

$$\mathcal{L}_j(u) = -\sum_{i=1}^{c(j)} y_i^j \log p_i^j = (\log \sum_{i=1}^{c(j)} \exp(o_i^j)) - (\sum_{i=1}^{c(j)} y_i^j o_i^j). \tag{4}$$

where y_i^j is the label of i-th node at layer j. From **E.q. (4)** and **E.q. (3)**, we can know that all the nodes contribute to the training loss simultaneously at layer j which may relieve the gap between training and prediction. For user u, the training loss w.r.t. the whole tree can be written as

$$\mathcal{L}(u) = \sum_{j \in all\ layers} \mathcal{L}_j(u). \tag{5}$$

It is obvious that we no longer suffer from the imbalance of positive node size and negative node size that happens in binary-class classification problem.

4.2 Sampled softmax

To compute the training loss (i.e. (**E.q. (5)**) for user u, we need to calculate the partition function $\sum_{i=1}^{c(j)} \exp o_i^j$ at each layer j. However, the calculation of partition function is unacceptable both in time complexity and memory consumption which both increase linearly w.r.t. the corpus size.

Sampled softmax is proposed to approximate full softmax during model training [4, 5]. Instead of calculating the training loss **E.q. (4)** over all classes, only the positive class and m negative classes are considered where the m negative classes are sampled from all the negative classes according to certain probability distribution $\boldsymbol{q}$ with replacement. In the rest of the paper, we use s^j to denote the training samples at layer j and use $q_{s_i^j}$ $(1 \le i \le c(j))$ to denote the probability that the i-th term of s^j can be sampled from the negative nodes of layer j. We suppose s_1^j is always the positive class and the rest terms of s^j are negative classes of layer j without loss of generality. For example, $m = 6$ and the sample set is $s^j = \{3, 5, 7, 8, 2, 7, 4\}$. The s_j indicates that the 3-rd node of layer j is the positive node and the 7-th node is sampled twice while other nodes (i.e. indexed at 5,8,2,4) are sampled once each.

However, we don't use the outputs w.r.t. sampled nodes of s^j to approximate the loss directly. The slight adjustment is conducted for each output by

$$\hat{o}_{s_i^j}^j = \begin{cases} o_{s_i^j}^j - \ln(m q_{s_i^j}^j) & if\ y_{s_i^j} = 0 \\ o_{s_i^j}^j - \ln(1) & if\ y_{s_i^j} = 1 \end{cases} \tag{6}$$

This adjustment can guarantee the sampled softmax is unbiased when m is infinite (i.e. $m \to \infty$) [5]. The training loss is calculated over the adjusted outputs and the original training loss **E.q. (4)** at layer j can be adjusted to

$$\hat{\mathcal{L}}_j(u) = -\sum_{i=1}^{m+1} y_{s_i^j}^j \log \hat{p}_i^j = (\log \sum_{i=1}^{m+1} \exp(\hat{o}_{s_i^j}^j)) - (\sum_{i=1}^{m+1} y_{s_i^j}^j \hat{o}_{s_i^j}^j). \tag{7}$$

where

$$\hat{p}_i^j = \frac{\exp \hat{o}_{s_i^j}^j}{\sum_{k=1}^{m+1} \exp \hat{o}_{s_k^j}^j}. \tag{8}$$

The training loss w.r.t. the whole tree for user u becomes

$$\hat{\mathcal{L}}(u) = \sum_{j \in all\ layers} \hat{\mathcal{L}}_j(u). \tag{9}$$

Up to now, we haven't specified how to sample m negative nodes at each layer. The previous literature [5, 6] has proved that proper specified sampling distribution $\boldsymbol{q}$ used in sampled softmax can lead to unbiased estimator of gradient for the original loss (i.e. **E.q. (4)** and **E.q. (5)**). We state their theorem in a different way so that the theorem can be compatible with our tree index.

Theorem 1 (**Theorem 2.1** of [6]). *The gradient of loss **E.q. (7)** (or **E.q. (9)**) w.r.t. sampled softmax is an unbiased estimator of the gradient of the loss **E.q. (4)** (or **E.q. (5)**) w.r.t. full softmax if and only if $q_i^j \propto \exp o_i^j$ holds where $1 \le i \le c(j)$ & the i-th node isn't a positive node at layer j.*

This theorem indicates that if the sampling probability is proportional to the exponential of preference w.r.t. the negative node then we can get the training loss's unbiased estimator of gradient. However, it still leads to linear time complexity w.r.t. the corpus size to make $q_i^j \propto \exp o_i^j$ as calculating the full softmax. To address this problem, the previous literature [6, 35] develops the kernel-based methods to approximate the full softmax for inner product models. However, their kernel-based methods aren't suitable to our model as our preference model is a complicated neural network. Uniform sampling is easy to understand and can be implemented quickly. Concretely, at layer j, each negative node is sampled uniformly with replacement until m negative nodes are sampled. The i-th node of layer j is sampled with probability $q_i^j = \frac{1}{c(j)-1}$ probability at each sampling operation. Besides simpleness, uniform sampling deserves another import advantage that it can lead to good exploration ability. As every negative node can be chosen with equal chance, the

sampled nodes deserve good diversity among each other. Furthermore, uniform sampling doesn't introduce any extra computational burden (e.g kernel-based sampling methods require to compute the kernel to get the distribution q.) so that the training process can be quick. In our experiments, we choose uniform sampling to get the negative nodes at each layer, i.e. $q_i^j = \frac{1}{c(j)-1}$ at **E.q. (6)**.

5 TREE LEARNING

Here we introduce how to update the tree (i.e. update the bijective mapping $\pi(\cdot)$) when the preference model is fixed. We regard the level of root node as level 0 and the level of leaf nodes as level l_{max}. $\mathcal{A}_i = \{(u', i')|i' = item_i\}$ means $\mathcal{A}_i$ consists of all the user-item pairs where the item is exactly the i-th item. $b_j(\pi(item_i))$ denotes leaf node $\pi(item_i)$'s ancestor node at level j. At first, we use the way of TDM and JTM to construct the initial tree. The tree is used to represent the user interests' hierarchical information so that similar items should organized in close positions on the level l_{max} of the tree. It's natural to rely on the category information of items to build the initial tree. Firstly, we shuffle all the categories and put the items together if they share the same category id. If the item belongs to more than one category, a random one is assigned to the item for uniqueness. Secondly, all the items locating in one category are partitioned into two equal parts recursively until each part only contains one item. In this way, we can construct a near-complete binary tree from root node to leaf nodes and some simple balancing strategy can be used to adjust the tree. We use the binary tree in our experiments. In fact, any multi-way tree can be constructed by such way if we partition each cluster into more nearly equal parts at each iteration.

Given the old mapping π and each interaction pair set $\mathcal{A}_i$, once we fix the preference model, we assign each item to each node of tree step by step from root node to leaf nodes to get the new mapping π. Firstly, all items are assigned to the root node, i.e. current level $l = 0$. Then, we try to assign all the items to the nodes of level $min(l_{max}, l + d)$ (where d is a small number and we use $d = 7$ in our experiments). The number of assigned items for node n at level $min(l_{max}, l + d)$ should be the number of the leaf nodes of the sub-tree who regards the node n as root node. The assigned item set w.r.t. each node at level $min(l_{max}, l + d)$ has no intersection and the union is exactly the whole item set (i.e. all the items will be assigned to the nodes at level $min(l_{max}, l + d)$ disjointly.). For the i-th item $item_i$ which is assigned to root node, it need to be assigned to one of the node at level $min(l_{max}, l + d)$ and it has $c \leq 2^d$ candidate nodes as we use the binary tree. Without loss of generality, we denote these candidate nodes as $n_1, n_2, \ldots, n_c$ and the user u's ($u \in \mathcal{A}_i$) preference to node n_j ($1 \leq j \leq c$) is denoted as o_u^j computed by the fixed preference model. We define $item_i$'s matching score to node j ($1 \leq j \leq c$) as follows

$$Score(item_i, n_j) = \sum_{(u, item_i) \in \mathcal{A}_i} \log \hat{p}(n_j|u) \tag{10}$$

where $\hat{p}(n_j|u) = \frac{\exp o_u^j}{\sum_{k=1}^c \exp o_u^k}$. We rank the c scores and assign $item_i$ to the node with the maximum matching score with $item_i$. If the number of assigned items breaks the limitation (i.e. the number of assigned items to each node is exactly the number of leaf nodes who regard the current node as root), we assigned $item_i$ to the node with second largest matching score with $item_i$ or even successive nodes if the number of assigned items still breaks the limitation. By this way, we can assign each item belonging to the root to a node at level $min(l_{max}, l + d)$. Now, we regard $n_1, \ldots, n_c$ as root nodes of c sub-trees and repeat the process recursively to assign the items to higher layers until each leaf node is assigned a item. Finally, we can get the new mapping π'. To compute the matching score (i.e. **E.q. (10)**), we need to compute the denominator $\sum_{k=1}^c \exp o_u^k$, so the number d need to be small to fit the time complexity and computational resource.

In fact, our tree learning strategy also carries on the hierarchical clustering from coarse grain to fine grain and d controls the grain size. The main difference between ours and the one in JTM is the calculation of matching scores. The matching score in JTM can be calculated by sum up the log-likelihood probability as the preference model output the probability directly but our strategy relies on the softmax probability when calculating the matching scores.

6 EXPERIMENTS

In this section, we study the performance of our proposed method. Firstly, We verify that regarding the preference model training task as the multi-class classification problem is superior over regarding it as binary-class classification problem. Secondly, we study the effectiveness of our proposed method compared with some classical baseline. Thirdly, we study the effectiveness of the tree learning method. At last, we sense the influence of varying the number of sampled nodes and the beam size when we conduct the beam search on the tree model. The used preference model in our proposed method is adjusted from the preference model of **Figure 1** by setting the last layer only contains one neuron and removing the softmax function. Following the settings of preference model in TDM and JTM, the preference model takes at most M user-item interactions as input user behavior features and split them into N time windows by time order. M is set to be 69 and N is set to be 10. Each of the 10 time windows contains $[1, 1, 1, 2, 2, 2, 10, 10, 20, 20]$ (sum up to 69; if the length of behavior history is less than 69, pad the absence by zero) interactions.

The paper JTM argues that hierarchical user preference representation brings to two benefits, i.e. level independence and precision description. Given a user behavior sequence $\boldsymbol{c} = \{c_1, c_2 \cdots, c_N\}$ where c_i is the i-th item the user interacts with, then the user behavior sequence is represented as $\boldsymbol{c}^j = \{b_j(c_1), b_j(c_2), \cdots, b_j(c_N)\}$ at layer j, where $b_j(c_i)$ denotes the corresponding ancesotor node of item c_i . We resort to the category information to initialize the tree like previous section mentioned. The experiments are conducted on two real-world datasets. **MIND Small Dev**[2] (abbreviated as **MIND**) consists of the interactions from 50,000 users on 5,369 news. We filter the dataset by removing the users who read no more that

[2]https://drive.google.com/drive/folders/1MucMieAUkjbAZVka3mxTaGuGvXbA1SYT

Table 1: Dataset

Dataset	#User	#Item	#Interaction
MIND	26,712	5,367	2,427,016
Movie	57,534	10,675	9,704,223

Table 2: Verify the superiority of multi-class classification over binary-class classification for tree model.

Dataset	Algorithm	Topk=20			Topk=40			Topk=60		
		Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
MIND	JTM	0.437	0.273	0.310	0.353	0.419	0.351	0.304	0.518	0.351
	DeFoRec	**0.531**	**0.333**	**0.379**	**0.408**	**0.477**	**0.404**	**0.339**	**0.569**	**0.390**
Movie	JTM	0.158	0.079	0.092	0.145	0.139	0.120	0.135	0.187	0.133
	DeFoRec	**0.173**	**0.088**	**0.103**	**0.155**	**0.150**	**0.130**	**0.143**	**0.199**	**0141**

Table 3: Compare with classical baselines.

Dataset	Algorithm	Topk=20			Topk=40			Topk=60		
		Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
MIND	Item-CF	0.364	0.243	0.269	0.337	0.419	0.342	0.304	0.537	0.357
	FM	0.373	0.243	0.271	0.348	0.439	0.356	0.317	0.567	0.374
	BPRMF	0.436	0.276	0.312	0.369	0.445	0.369	0.324	0.559	0.376
	Youtube	0.470	0.305	0.342	0.403	0.493	0.407	0.353	0.613	0.412
	TDM	0.448	0.281	0.319	0.362	0.428	0.359	0.310	0.528	0.358
	JTM	0.554	0.352	0.399	0.429	0.508	0.427	0.356	0.604	0.412
	DeFoRec	**0.593**	**0.375**	**0.426**	**0.452**	**0.530**	**0.449**	**0.370**	**0.621**	**0.427**
Movie	Item-CF	0.056	0.033	0.037	0.094	0.109	0.088	0.110	0.182	0.119
	FM	0.084	0.044	0.051	0.107	0.122	0.099	0.117	0.196	0.128
	BPRMF	0.089	0.043	0.050	0.105	0.113	0.093	0.113	0.180	0.119
	Youtube	0.190	0.098	0.113	0.175	0.179	0.150	0.162	0.235	0.164
	TDM	0.141	0.070	0.081	0.131	0.126	0.108	0.124	0.171	0.121
	JTM	0.185	0.097	0.112	0.167	0.167	0.143	0.155	0.223	0.156
	DeFoRec	**0.215**	**0.111**	**0.129**	**0.189**	**0.186**	**0.161**	**0.171**	**0.242**	**0.171**

15 news. 26,712 users, 5,367 movies and 27,270,160 interactions are left. 3,000 users are randomly selected as validation data and 3,000 users are randomly selected as test data and the left users are regarded as training data. **MovieLens 10M**[3](abbreviated as **Movie**) is a movie rating dataset which contains 1,000,054 rating records from 69,878 users to 10,677 movies. The users who rate at least 15 movies can be kept. Then 57,534 users, 10,675 movies and 2,427,016 rating records are left. 6,000 users are randomly selected as validation data and 6,000 users are randomly selected as test data and the left users are regarded as training data. The information of datasets is summarized in **Table 1**. The dimension of item embedding and node embedding is set to be 24. All the experiments are repeated 10 times and the average value is reported. For validation users and test users, the first half behaviors are regarded as the interaction history and the second half behaviors are regarded as ground truths. To evaluate the performance of each method, we use $Precision@K$, $Recall@K$ and $F\text{-}measure@K$ as the metrics. Suppose $\mathcal{P}(u)$ ($|\mathcal{P}(u)| = K$) denotes the recalled set w.r.t. user u and $\mathcal{G}(u)$ denote the ground truth set. Then

$$Precision@K(u) = \frac{|\mathcal{P}(u) \cap \mathcal{G}(u)|}{K}, \ Recall@K(u) = \frac{|\mathcal{P}(u) \cap \mathcal{G}(u)|}{|\mathcal{G}(u)|}, \tag{11}$$

and

$$F - measure@K(u) = \frac{2 \cdot Precision@K(u) \cdot Recall@K(u)}{Precision@K(u) + Recall@K(u)}. \tag{12}$$

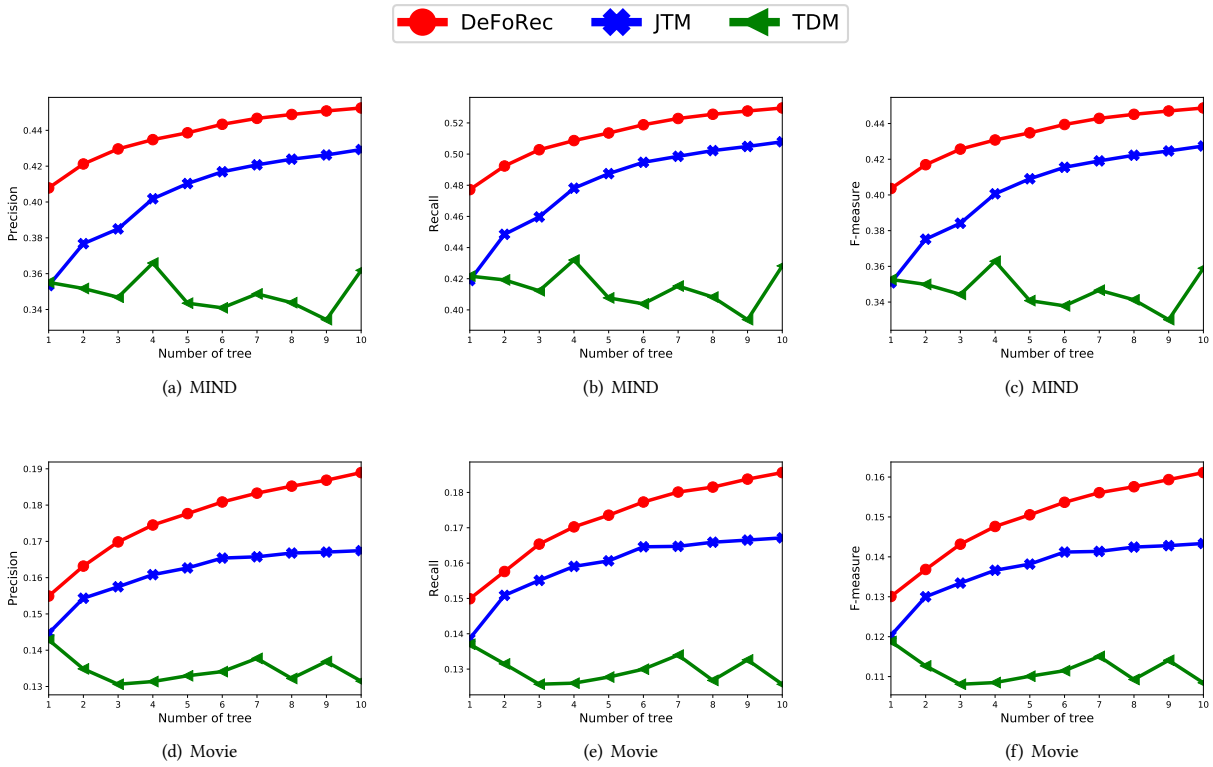
[3]https://grouplens.org/datasets/movielens/10m/

6.1 Superiority of Multi-class classification loss

TDM and JTM use the binary cross entropy i.e. (**E.q. (2)**) to train the preference model. Our proposed method regard the training task as a mulit-class classification problem. To verify multi-class classification is superior to train the preference model over binary-class classification, We train the preference model on the initial trees of the two datasets. As the only difference between TDM and JTM is the how to update the tree but we don't update the tree here, so TDM and JTM share the results. As we test the influence of parameters in the last subsection, so we just present the used parameter settings in this subsection. For **MIND**, we sample 40 times negative nodes over positive nodes (i.e. $m = 40$ in **E.q. (6)** for our method and 9 times negative nodes over positive nodes for JTM. For **Movie**, we sample 60 times negative nodes and 8 times negative nodes over positive nodes for ours method and JTM respectively. The beam size is set to be 150, i.e. we select at most 150 items to expand when we conduct beam search on the tree. The results are presented in **Table 2**. We can know that the results of our method significantly outperforms JTM for all cases on both datasets. The results indicate that regarding preference model training task as a multi-class classification problem is better than regarding it as binary classification problem.

6.2 Comparisons with other baselines

We also compare our methods with the following methods.
Item-CF [39] : It is a basic collaborative filtering algorithm. We implement it by our-self and the Pearson correlation coefficient is



(a) MIND (b) MIND (c) MIND

(d) Movie (e) Movie (f) Movie

Figure 2: Results when we increase the number of tree.

used to measure the similarity between items.

FM [36]: It is designed for factorization task. We use the implementation provided by DeepMatch[4].

BRPMF [38] : The matrix factorization is used for Bayesian personalized ranking. We use the open-source code [5].

Youtube [7]: It is the recommender system used in Youtube which uses the inner products as the preference scores between users and items. After all the user embedding and item embedding are learnt by the DNN model, Exact KNN search is used to find the top-K items for users in prediction. We use the implementation provided by DeepMatch with the same source as **FM**.

For our method, JTM and TDM we use the same setting as **subsection 6.1** and 10 trees are learnt to form the forests for our algorithm, TDM and JTM respectively. The preference model trained on the last updated tree is regarded as the discriminator to rank all the candidates generated from each tree for our method, JTM and TDM respectively. For Item-CF, FM and BPRMF, we tune several most important hyper-parameters based on validation set. Concretely, the number of factors and BPRMF, the number of neighbors for Item-CF. For Youtube, the dimension of user embedding and item embedding is set to be 24 and the hidden unit number of three fully connected layers are 128,64,24 respectively. Additionally, The negative number in Youtube is set to be the same as our method on each dataset.

[4]https://github.com/shenweichen/DeepMatch
[5]https://github.com/SpringtoString/BPR-Torch

The results are exhibited in **Table 3**. We can see that our method significantly outperforms the classical baseline like Item-CF, FM, BPRMF under different metric measure and different Topk on both datasets. The Youtube also performs quite well especially on **Movie**. This phenomenon may be caused by two reasons. Firstly, Youtube is desigened for video recommendation and **Movie** is exactly video rating dataset. Secondly, after the user embedding and item embedding are learnt by Youtube, we use exact KNN search to find the item embedding with maximum inner product with query user. Exactly KNN search scans all items with linear time complexity. All the results in **Table 3** indicate the superiorities of our proposed method.

6.3 Effectiveness of updating the tree

As stated in previous sections, We train the preference model and update the tree alternately for ours method, JTM and TDM. In prediction, each tree is retrieved by the corresponding preference model to generate a candidate set. All the candidates generated by each tree are mapped to the leaf nodes of the last updated tree so that we can choose the last updated preference model as the discriminator to rank all the candidates. Top-K items can be returned from the ranked items. The experiments are conducted on both two datasets and the numbers of sampled negative nodes are the same as ones in the above subsections. Beam size is 150 and Topk=40. The results are presented in **Figure 2** w.r.t. the three metrics respectively. We can see that our method outperforms JTM

Table 4: The influence of beam size.

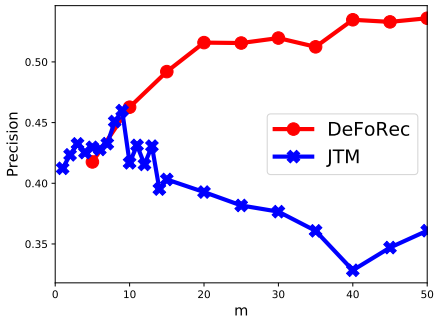
Dataset		MIND			Movie		
Beam size		100	150	200	100	150	200
P	JTM	0.356	0.353	0.352	0.145	0.145	0.143
	DeFoRec	**0.408**	**0.408**	**0.407**	**0.156**	**0.155**	**0.154**
R	JTM	0.421	0.419	0.418	0.140	0.139	0.137
	DeFoRec	**0.477**	**0.477**	**0.476**	**0.151**	**0.150**	**0.149**
F	JTM	0.353	0.351	0.350	0.120	0.120	0.119
	DeFoRec	**0.404**	**0.404**	**0.403**	**0.131**	**0.130**	**0.129**

and TDM steadily as the number of tree increases. The experiment results of JTM and TDM in paper [52] have indicated that the tree learning method of TDM doesn't work well and it can even learn a worse tree. Our results also verify the fact. Overall, all these results indicate that the proposed tree learning method is effective.

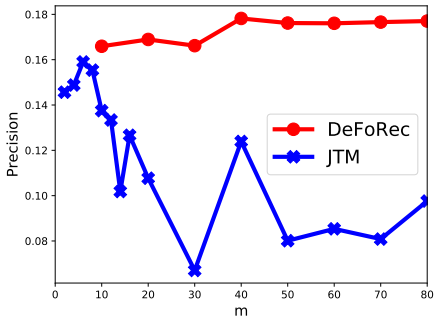
6.4 Tune parameters

At first, we investigate the influence of beam size when we conduct beam search on the tree. The number of sampled negative nodes are the same as ones in previous subsections for our method and JTM respectively. Topk is set to be 40 and all the results are obtained on the initial trees of the two datasets. The results are presented in **Table 4** where P means precision, R means recall and F means F-measure. We can see that beam size has no significant influence when we conduct beam search on the trees with trained preference model of ours method or JTM. Larger beam size can even damage the performance of preference model slightly. This phenomenon may be because that we can't learn a exact preference model and larger beam size lead to more candidates to rank. But the inexact preference model makes more mistakes when it needs to rank more candidates.

Too small number of sampled negative nodes may lead to weak exploration and too many sampled negative nodes can exacerbate the imbalance between number of positive nodes and number of negatives for TDM and JTM. We vary the number (i.e. m, which means the number of negative nodes is m times larger than number of positive nodes) of sampled negative nodes for our method and JTM. The experiments are conducted on the initial trees w.r.t. **MIND** and **Movie** respectively. Beam size is set to be 100 and TopK is set to be 20. The precision is reported in **Figure 3**. We can know that the performance increases very slightly for our methods as m increasing. However, the performance of JTM first increases and then decreases in general. This phenomenon meets our expectation that too many sampled negative nodes can aggravate the imbalance between negative nodes and positive nodes for binary-class entropy loss so that degrading the performance.



(a) MIND



(b) Movie

Figure 3: Vary the number of negative nodes.

7 CONCLUSION

The existed tree-base recommendation systems (e.g. TDM and JTM) regard the preference model training task as a binary-class classification problem. Their training mode can suffers from the imbalance between the number of positive nodes and the number of negative nodes. Further, there is a gap between training and prediction as the lacking of competition among nodes at the same layer. To address these problem, We develop a layer-wise training mode that regrading the training task as a multi-class classification problem and we provide the tree learning strategy to update the tree. To implement our training mode efficiently, we sample negative nodes at each layer by the guidance of sampled softmax theory. The retrieval is conducted only on the last updated tree in both TDM and JTM. We utilize all the trees to form the Deep Forest-based recommendation system. Each tree is retrieved and all the generated candidates on each tree are ranked by a user-specified discriminator. The experimental results validate the effectiveness of our proposed training mode and the Deep Forest-based recommender.

ACKNOWLEDGMENTS

The work was supported by grants from the National Key R&D Program of China under Grant No. 2020AAA0103800, the National Natural Science Foundation of China (No. 61976198 and 62022077).

REFERENCES

[1] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of CVPR'14*. 931–938.
[2] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.
[3] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of RecSys'14*. ACM, 257–264.

[4] Yoshua Bengio and Jean-Sébastien Senécal. 2003. Quick training of probabilistic neural nets by importance sampling. In *International Workshop on Artificial Intelligence and Statistics*. PMLR, 17–24.

[5] Yoshua Bengio and Jean-Sébastien Senécal. 2008. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks* 19, 4 (2008), 713–722.

[6] Guy Blanc and Steffen Rendle. 2018. Adaptive sampled softmax with kernel based sampling. In *International Conference on Machine Learning*. PMLR, 590–599.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[10] Ian J Goodfellow. 2014. On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515* (2014).

[11] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. 2016. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*. 482–490.

[12] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.

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

[14] 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.

[15] Geoffrey E Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation* 14, 8 (2002), 1771–1800.

[16] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of WWW'17*. International World Wide Web Conferences Steering Committee, 193–201.

[17] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[18] Binbin Jin, Defu Lian, Zheng Liu, Qi Liu, Jianhui Ma, Xing Xie, and Enhong Chen. 2020. Sampling-decomposable generative adversarial recommender. *Advances in Neural Information Processing Systems* 33 (2020), 22629–22639.

[19] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of CIKM'12*. ACM, 535–544.

[20] Xiang Li, Tao Qin, Jian Yang, and Tie-Yan Liu. 2016. LightRNN: Memory and computation-efficient recurrent neural networks. In *Advances in Neural Information Processing Systems*. 4385–4393.

[21] Zechao Li, Jinhui Tang, Liyan Zhang, and Jian Yang. 2020. Weakly-supervised semantic guided hashing for social image retrieval. *International Journal of Computer Vision* 128, 8 (2020), 2265–2278.

[22] Defu Lian, Qi Liu, and Enhong Chen. 2020. Personalized ranking with importance sampling. In *Proceedings of The Web Conference 2020*. 1093–1103.

[23] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *Proceedings of KDD'17*. 325–334.

[24] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. LightRec: A Memory and Search-Efficient Recommender System. In *Proceedings of The Web Conference 2020*. 695–705.

[25] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2009–2019.

[26] Defu Lian, Xing Xie, Enhong Chen, and Hui Xiong. 2021. Product Quantized Collaborative Filtering. *IEEE Transactions on Knowledge and Data Engineering* 33, 9 (2021), 3284–3296. https://doi.org/10.1109/TKDE.2020.2964232

[27] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[28] Denis Mazur, Vage Egiazarian, Stanislav Morozov, and Artem Babenko. 2019. Beyond vector spaces: compact data representation as differentiable weighted graphs. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 6906–6916.

[29] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model.. In *Aistats*, Vol. 5. Citeseer, 246–252.

[30] Stanislav Morozov and Artem Babenko. 2018. Non-metric similarity graphs for maximum inner product search. *Advances in Neural Information Processing Systems* 31 (2018), 4721–4730.

[31] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *Proceedings of ICML'15*. 1926–1934.

[32] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *Proceedings of CVPR'12*. IEEE, 3108–3115.

[33] Franco P Preparata and Michael I Shamos. 2012. *Computational geometry: an introduction*. Springer Science & Business Media.

[34] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 931–939.

[35] Ankit Singh Rawat, Jiecao Chen, X Yu Felix, Ananda Theertha Suresh, and Sanjiv Kumar. 2019. Sampled Softmax with Random Fourier Features.. In *NeurIPS*.

[36] Steffen Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. 995–1000. https://doi.org/10.1109/ICDM.2010.127

[37] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of WSDM'14*. ACM, 273–282.

[38] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback *(UAI '09)*. AUAI Press, Arlington, Virginia, USA, 452–461.

[39] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (Hong Kong, Hong Kong) *(WWW '01)*. Association for Computing Machinery, New York, NY, USA, 285–295. https://doi.org/10.1145/371920.372071

[40] Anshumali Shrivastava and Ping Li. 2014. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). *arXiv preprint arXiv:1410.5410* (2014).

[41] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197* (2019).

[42] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast item ranking under neural network based measures. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 591–599.

[43] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1235–1244.

[44] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 515–524.

[45] Jason Weston, Samy Bengio, and Nicolas Usunier. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning* 81, 1 (2010), 21–35.

[46] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 353–362.

[47] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of SIGIR'16*. ACM, 325–334.

[48] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 785–788.

[49] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5941–5948.

[50] 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*. 1059–1068.

[51] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *Proceedings of KDD'12*. ACM, 498–506.

[52] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. Joint Optimization of Tree-based Index and Deep Model for Recommender Systems. In *NeurIPS*.

[53] 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* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1079–1088.