SR-PredictAO: Session-based Recommendation with High-Capability Predictor Add-On

Ruida WANG, Raymond Chi-Wing WONG, Weile TAN
The Hong Kong University of Science and Technology
Kowloon, Hong Kong
rwangbr@connect.ust.hk, raywong@cse.ust.hk, wtanae@connect.ust.hk

***Abstract*—Session-based recommendation, aiming at making the prediction of the user's next item click based on the information in a single session only, even in the presence of some random user's behavior, is a complex problem. This complex problem requires a high-capability model of predicting the user's next action. Most (if not all) existing models follow the encoder-predictor paradigm where all studies focus on how to optimize the encoder module extensively in the paradigm, but they overlook how to optimize the predictor module. In this paper, we discover the critical issue of the low-capability predictor module among existing models. Motivated by this, we propose a novel framework called *Session-based Recommendation with Predictor Add-On* (SR-PredictAO). In this framework, we propose a high-capability predictor module which could alleviate the effect of random user's behavior for prediction. It is worth mentioning that this framework could be applied to any existing models, which could give opportunities for further optimizing the framework. Extensive experiments on two real-world benchmark datasets for three state-of-the-art models show that *SR-PredictAO* out-performs the current state-of-the-art model by up to 2.9% in HR@20 and 2.3% in MRR@20. More importantly, the improvement is consistent across almost all the existing models on all datasets, and is statistically significant, which could be regarded as a significant contribution in the field.**

***Index Terms*—session-based recommendation, recommender system, neural decision forest, tree-based method**

I. Introduction

Next-item recommender systems show their importance in the current age of e-commerce by accurately predicting the user's subsequent behavior. *Session-based recommendation* is one recent hot topic of the next-item recommender. It is different from the *general next-item recommendation systems*, which put great attention on a specific group of existing users with a large number of historical behavior records to perform the next-item prediction. The *session-based recommendation*, as its name indicates, groups all the activities in the basic unit of the session and is based only on the information within a single session. The idea of session-based recommendation systems comes from [1]. It shows that intra-session-dependencies have a more significant impact than inter-session dependencies on the user's final decision to view the next item. In particular, the user's next-item behavior is usually related to behaviors in the current session. For example, a user's behavior in buying phone accessories in one session has a relatively low connection to his/her action of buying clothes two days ago but has a strong relationship with his/her visit to a phone charger in the same session.

Due to the highly practical value in the field of modern commerce, the session-based recommendation attracts researchers' interest. In recent years, most (if not all) proposed models followed the *encoder-predictor paradigm* [2]–[7], involving 2 components. The first component is the *session encoder module*, and the second component is the *predictor module*. The session encoder module transforms the input session (represented in the form of a sequence of items) into an n'-dimensional vector called the *latent variable*, where n' is a positive integer denoting a model parameter. The predictor module generates a probability distribution over all items that represents how likely each item is to be the next item. The paradigm is shown in Fig. 1 (a). Different existing models have different implementations of the encoder modules. For example, in [8], the encoder module is a Gated GNN that captures complex transitions of items to obtain the latent variable, and in [9], the encoder module is a Star GNN that uses a star node, representing the whole session, and a Highway Network, handling the overfit problem. The predictor modules of most (if not all) existing models are all *linear* models.

Although existing models following the current encoder-predictor paradigm perform well, there are still some issues for further enhancement. The first issue is that most (if not all) existing models have a *low-capability* predictor module, which affects the prediction accuracy. Specifically, under the encoder-predictor paradigm, even though there is an advanced model in the encoder module constructing the latent variable, which could represent the latent intent of a user's purchase; it is the predictor component that makes the recommendation, which could somehow simulate the complicated decision process of a user's purchase. Unfortunately, most (if not all) existing models use linear predictors, which are low-capability models, limiting the prediction performance.

The second issue is that designing a *high-capability* model is challenging by considering the overfit problem [10]. Specifically, one straightforward solution for the first issue is to design a high-capability model. It is well-known that an *extremely* high-capability model suffers from the overfit problem. How to design an *appropriate* high-capability model is needed for detailed investigation.

The third issue is that there is *random user's behavior* in the input session, which may affect the prediction performance. It includes multi-intention problems where the user is distracted

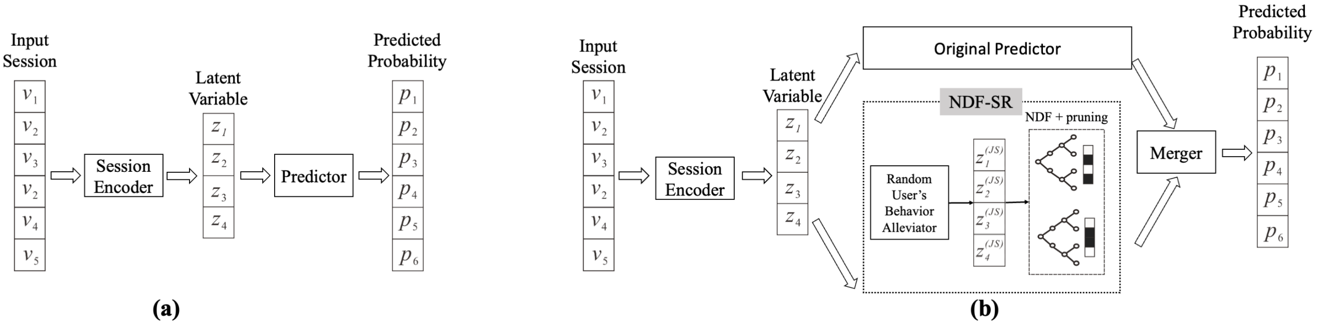

Fig. 1: (a) The overview of the base model, (b) Framework *SR-PredictAO*; Given an input session S, the encoder module generates the latent variable $\boldsymbol{z}$. In (a), $\boldsymbol{z}$ is passed to the base model predictor module to obtain the predicted probability distribution over all items. In (b), $\boldsymbol{z}$ is passed to both the base model predictor module and the new predictor module (called *NDF-SR*) to obtain two predicted probability distributions over all items. Then, module *Merger* combines the two distributions to output the final distribution.

from her/his original intention of the current session. But, more generally, it can represent any random behavior of user, which could create a challenge for prediction in existing models. Previous studies [3], [9] have tried to solve that in the GNN encoder but not completely.

In this paper, we propose a novel framework called *Session-based Recommendation with Predictor Add-On* (*SR-PredictAO*). Under *SR-PredictAO*, given an existing model called *base model* in this paper, we keep all existing modules of this base model but we augment the base model with two additional modules. The first additional module is the high-capability predictor module, which takes the latent variable as input and outputs the predicted probability distribution over all items being the next item in the session. Maintaining the original (low-capability) predictor module, with our new high-capability predictor module, we can capture different sides of user's decision process. The second additional module is module *Merger*, which takes the probability distributions over all items predicted by both the original predictor module and the new predictor module and outputs the final probability distribution over all items. This framework provides a lot of opportunities to researchers for optimization on how to specify these 2 modules, which is quite promising. The SR-PredictAO framework could be found in Fig. 1 (b) where the first augmented module is named as *NDF-SR* (which will be described next). It is worth mentioning that our framework *SR-PredictAO* could be applied to all existing models following the encoder-predictor paradigm (with the two additional modules), which could further improve the prediction performance of all existing models. Due to the nature limitation that tree-based methods hardly models linear decision boundaries, we combine the tree-based model with the linear model to complement each other.

In this paper, we propose a model called *Neural Decision Forest for Session-based Recommendation* (NDF-SR) for the first high-capability predictor module. Specifically, NDF-SR involves two components. The first component is called the *random user's behavior alleviator*, which could minimize the effect of random user's behaviors for the prediction process (addressing the third issue). The second component is called the *Neural Decision Forest* (NDF) model, which is a high-capability model (addressing the first issue). It could be regarded as a *forest* involving a number of *decision trees* each constructed with the use of *neural* network models. We also propose a pruning method in the NDF model to avoid the overfit problem (addressing the second issue). Furthermore, in this paper, for the second *Merger* module, we adopt a simple linear combination which combines the predicted distributions from the original predictor and the new predictor to obtain the final predicted probability distribution. In the following, for clarify, when we describe *SR-PredictAO*, we mean the framework adopting the above modules.

In summary, our contributions are shown as follows.

1) To the best of our knowledge, we are the first to find the important low-capability issue in the predictor module of most (if not all) existing models, lowering down their prediction accuracy.
2) To address this important issue, we propose a framework called *SR-PredictAO* including the high-capability predictor module where this module involves two components, namely the *random user's behavior alleviator* (addressing the random user's behavior issue) and the *Neural Decision Forest* (NDF) model (addressing the low-capability predictor issue). Moreover, we propose some pruning methods in the NDF model to address the overfit problem.
3) We conduct extensive experiments on two public benchmark datasets, namely *Yoochoose* and *Diginetica*, for three state-of-the-art models. Experimental results show that *SR-PredictAO* improves almost all state-of-the-art models on all datasets up to 2.9% on HR@20 (one accuracy measurement) and up to 2.1% on MRR@20 (another accuracy measurement), which could set a new state-of-the-art in the literature. This improvement is *consistent* on all datasets. By considering the consistency of improvement and the ease of applicability

of our framework, we regard our contribution as a major improvement to the field of the session-based recommendation system.

II. Related Work

In this section, we introduce the related work about session-based recommendation (Section II-A) and neural decision forest (Section II-B).

A. Session-based recommendation

We categories existing studies about session-based recommendation into three categories: (1) conventional recommendation methods, (2) neural-network-based methods and (3) graph neural-network-based methods.

Due to the similarity between the *session-based recommendation* (SR) problem and the traditional recommendation problem, conventional methods like Collaborative Filtering (CF) approaches [11], [12], nearest-neighbor approaches [13], [14] and Markov's chain approaches [15] are applied to the SR problem. However, due to the limited information in the session, they all performed poorly in the SR problem.

With the improvement of computation power and knowledge in *Neural Network* (NN), many NN-based models, including RNN approaches [16], the transformer-based approach [17] and the CNN-based approach [18], [19], have been proposed. However, most of them do not perform well due to the traditional NN's encoding methods does not fit the session data well.

In recent years, graph neural networks (GNNs) have become popular and have been shown to have state-of-the-art performance in many domains. Many recommendation systems [3], [4], [8], [9] also utilize GNNs due to its ability of modeling complex relationships among objects. In [8], Wu et al. apply gated graph neural networks (GGNNs) to capture the complex transitions of items, which result in accurate session representations. In [3], to solve information loss problems in GNN-based approaches for session-based recommendation, Chen et al. proposed a lossless encoding scheme, involving a dedicatedly designed aggregation layer and a shortcut graph attention layer. In [9], Pan et al. proposed Star Graph Neural Networks with Highway Networks (SGNN-HN) for session-based recommendation. In particular, the highway networks (HN) can select embeddings from item representations adaptively to order to prevent from overfitting. However, all aforementioned studies [3], [4], [8], [9] use the (low-capability) linear model as the predictor (described in Section I).

B. Tree-based method

The traditional tree-based method was proposed by Breiman in [20], [21]. Its outstanding performance in simulating the human decision process is studied by Quinlan et al. in [22] The high capability of the tree-based methods was shown by Mentch et al. [23]. With the rapid development of computation power and neural networks, a lot of effort has been made to combine classical tree-based methods with neural networks. In [24], Richmond et al. introduced *Convolutional Neural Networks* (CNNs) as representation learners on a traditional random forest. Jancsary et al. in [25] introduced *regression tree fields* for image restoration. To solve the problem that the traditional tree-based method cannot do backward propagation with other NN-based parts in the model, in [26], Kontschieder et al. constructed uniform and end-to-end differentiable Deep Neural Decision Forest and applied it to some computer vision models. To the best of our knowledge, no existing studies about session-based recommendation system utilizes the the tree-based models incorporated with the backward propagation with the NN-based parts in the models. We are the first one to propose this in the field of session-based recommendation system.

III. Preliminaries

In this section, we introduce (1) problem definition (Section III-A), (2) some preliminary knowledge about a *base model*, an existing model, following the encoder-predictor paradigm (Section III-B) and (3) the traditional version of the tree-based method (Section III-C).

A. Problem Definition

The session-based recommendation is a sub-field of the next-item recommendation only with the input from a specific session. Its goal is to predict the next item that a user will browse based on the current active session involving all previus items browsed. We denote $I = \{v_1, v_2, \cdots, v_N\}$ by the universal set of items in the whole dataset, where N is the total number of items. A session, denoted by $\boldsymbol{s}_i = [s_{i,1}, s_{i,2}, \cdots, s_{i,l_i}]$, is a time-ordered sequence of items, where i is a temporary index of the session, l_i denotes the length of $\boldsymbol{s}_i$ and, for each $t \in [1, l_i]$, $s_{i,t} \in I$ is the item at time step t in the session. The goal of the session-based recommendation is to predict what the next item s_{i,l_i+1} is. A typical session-based recommendation system generates a probability distribution over all items predicted being the next item, i.e., $\mathbb{P}(s_{i,l_i+1}|\boldsymbol{s}_i)$.

Additionally, we formally define the random-user behavior and low-capability problem that *SR-PredAO* tries to solve as: (1) Low-capability of the predictor can be defined as low Degrees-of-Freedom (DoF) problems in the predictor because DoF usually means the max ability of the model [23]. (2) Random-user behavior is the mean-square difference between the real value of model encoded result and its true value [27], the rigorous definition is in SectionIV-A. This can generally be caused by multiple comprehensive reasons including multiple intent, distractions, etc.

B. Base Model

The base model (following the encoder-predictor paradigm) is formulated as follows.

$$\boldsymbol{z} = f_{encode}(\boldsymbol{s}|\Theta_{encoder}) \tag{1}$$

$$\boldsymbol{y}_{base} = g_{predict}(\boldsymbol{z}|\Theta_{predictor}) \tag{2}$$

where (1) $\boldsymbol{s}$ is the input session (represented in the form of a sequence of items), (2) $\boldsymbol{z}$ is the latent variable generated by the encoder module of the model, (3) $\boldsymbol{y}_{base}$ denotes the probability distribution over all items predicted being the next item, (4) f_{encode} is the encoder which takes the input session as input and outputs a latent variable (a vector in $\mathbb{R}^{n'}$) (5) g_{predict} is the predictor module which takes the latent variable as input and outputs the probability distribution, and (6) Θ_{encode} ($\Theta_{\mathit{predict}}$) is the parameter configuration of the encoder (predictor) module.

As described in Section I, different existing models have different implementations of the encoder modules. In the following, we describe the encoder module and the predictor module of a base model of some state-of-the-art models.

1) Encoder Module: This section focuses on the most popular base model's session encoding method, the GNNs encoder. But our methods can work on all kinds of session encoders as long as it generates a latent variable. GNNs are *Neural Networks* (NN) that directly operate on the graph of data, given a graph $G = (V, E)$, where each node $v_i \in V$ represents an item in $\boldsymbol{s}$ (the session). Typically, v_i is associated with a node feature vector $\boldsymbol{x}_i$, which is the input to the first layer of GNNs. $\boldsymbol{x}_i \in \mathbb{R}^n$ is obtained by multiplying the embedding matrix (we define embedding matrix as $\boldsymbol{A} \in \mathbb{R}^{N\times n}$ with the item ID), where n is the embedding dimensionality. And $\boldsymbol{A}$ is a trainable matrix. Assume we totally have L layers of GNN. The formula of l-th ($l \leqslant L$) layer of GNN can be represented as follows:

$$\boldsymbol{x}_i^{(l+1)} = f^{(l)}(\boldsymbol{x}_i^{(l)}, \boldsymbol{a}_i^{(l)}) \tag{3}$$

$$\boldsymbol{a}_i^{(l)} = agg^{(l)}(\{msg^{(l)}(\boldsymbol{x}_i^{(l)}, \boldsymbol{x}_j^{(l)})|(j,i) \in E_{in}(i)\}) \tag{4}$$

where $\boldsymbol{x}_i^{(l)}$ is the embedding vector of node i in the l-th layer of the GNN, and $E_{in}(i)$ is the set of incoming edges for node $v_i \in V$. The message processing function at the l-th layer $f^{(l)}$ generates the updated embedding of the target node based on its neighborhood. $agg^{(l)}$ is the aggregate function that connects the information of different edges together, and $msg^{(l)}$ is the message-extracting function that obtains information from the edge between $(x_i^{(l)}, x_j^{(l)})$. Let L be the total number of layers in the GNN. After L steps of message passing, the final representation for the latent variable is:

$$\boldsymbol{h}_G = f_{out}(\{\boldsymbol{x}_i^{(L)}|v_i \in V\}) \tag{5}$$

$\boldsymbol{h}_G$ is the graph-level representation that we regard as the graph latent variable generated by the readout function f_{out}.

After the graph level latent variable $\boldsymbol{h}_G$ is obtained, most models adds some additional information to obtain a better result. For example, [3] adds all results of the Embedding layer, EOPA Layer, and SGAT Layer's (two special kinds of GNN mentioned in [3]) information to the graph representation, and [9] formulates the final result by concatenating $\boldsymbol{z}_g$ and $\boldsymbol{z}_r$, which are the last item's representation and the combination of all the graphs' result representation come from different levels respectively. After considering all the required information of the base model, we define this vector as the latent variable $\boldsymbol{z} \in \mathbb{R}^{n'}$, where n' is the dimensionality of the latent variable. This approach is used in almost all well-known session-recommendation models [3], [4], [8], [9] .

2) Predictor Module: After the encoder module outputs the latent variable, the predictor module takes this as input and performs the following steps.

1) The first step is to perform a prediction function (normally a linear model), which takes the latent variable as input and outputs an embedding called the *session embedding* $\boldsymbol{s}_h \in \mathbb{R}^n$ where n is the dimensionality of the session embedding, same as the embedding dimension of $\boldsymbol{A}$
$$\boldsymbol{s}_h = \mathrm{Linear}(\boldsymbol{z}) \tag{6}$$
2) The second step is to obtain the *score vector* $\boldsymbol{c} \in \mathbb{R}^N$ over all items predicted being the next item.
$$\boldsymbol{c} = [c_1, c_2, \cdots, c_N]^T = \boldsymbol{A}\boldsymbol{s}_h \tag{7}$$
where $c_i \in \mathbb{R}$ is a score of item v_i predicted being the next item for each $i \in [1, N]$ and $\boldsymbol{A} \in \mathbb{R}^{N\times n}$ is the item embedding matrix we used before.
3) The third step is to obtain the *probability vector* $\hat{\boldsymbol{y}}_{base} \in \mathbb{R}^N$ over all items predicted being the next item by using the softmax function based on the score vector $\boldsymbol{c}$.
$$\hat{\boldsymbol{y}}_{base} = softmax(\boldsymbol{c}) = \frac{\exp(\boldsymbol{c})}{\sum_{i\in[1,N]} \exp(c_i)} \tag{8}$$

C. Tree-based method

From the mathematical point of view, the tree-based method is a way of generating a locally constant function, represented by function $tree : \mathbb{R}^{n'} \rightarrow \mathbb{R}^N$ that divides the input space $\mathbb{R}^{n'}$ into many regions, and give each subspace a constant value in $\mathbb{R}^N$. And we can define the tree recursively by first defining the *tree-split* function φ:

$$\varphi(\boldsymbol{x}) = \chi(\boldsymbol{x} \in S)\boldsymbol{c}_l + \chi(\boldsymbol{x} \notin S)\boldsymbol{c}_r \tag{9}$$

where $S \subseteq \mathbb{R}^{n'}$ is a subregion of the input space, and $\chi(\boldsymbol{x} \in S)$ is judging function that returns 1 when $x \in S$, and 0 otherwise. The $\boldsymbol{c}_l, \boldsymbol{c}_r$ are defined as the left and right nodes of the tree-split. If $\boldsymbol{c}_l$ or $\boldsymbol{c}_r$ have its value in $\mathbb{R}^N$, where N is the dimension of the predicted result, then we say it is a *leaf node*; if not, it is an *internal node* that is associated with another tree split $\varphi_{l/r}$. And the tree function can be represented as $tree(\boldsymbol{x}) = \varphi_{root}(\boldsymbol{x})$ where φ_{root} is the tree-split function associated with the *root node*, the beginning node of the tree. The max number of tree-split need to have from the root to the leaf node is defined as *depth*.

For example, in Fig. 2, each node $d_i (i \in [1, 7])$ is associated with a tree split function φ_i with corresponding region S_i. The node of d_1 is the root node (i,e., $tree = \varphi_1$), the nodes of $d_{i\neq 1}$ are internal nodes. And node of $\boldsymbol{\pi}_j (j \in [1, 8])$ is leaf node that have its value $\boldsymbol{\pi}_j \in \mathbb{R}^N$.

IV. Framework SR-PredictAO

Framework *SR-PredictAO* involves two modules, namely the high-capability predictor module (Section IV-A) and the Merger module (Section IV-B). The training process of *SR-PredictAO* is presented in Section IV-C.

A. High-Capability Predictor Module

We propose a model called *Neural Decision Forest for Session-based Recommendation* (NDF-SR) for the high-capability predictor module. Specifically, NDF-SR involves two components. The first component is called the *random user's behavior alleviator* (Section IV-A1) and the second component is called the *Neural Decision Forest* (NDF) model (Section IV-A2). As described in Section I, we also propose a pruning method in the NDF model to avoid the overfit problem. This pruning method could be found in the description for the second component.

1) Random User's Behavior Alleviator: The base-model encoded latent variable for the previous session view of items is normally heavily affected by random user's behavior. To solve this problem, we could take the Empirical Bayes' point of view [27]. For Empirical Bayes', the observed data is not the underlying true value but a sample under a certain distribution around the truth. We would design our Alleviator under this cognition.

Formally, if a batch $\boldsymbol{Z} \in \mathbb{R}^{m \times n'}$ of m latent variables each with dimensionality of n' we observe from the base model's encoder is:

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1^T \\ \boldsymbol{z}_2^T \\ \vdots \\ \boldsymbol{z}_m^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{n'} \end{bmatrix}^T = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1n'} \\ z_{21} & z_{22} & \cdots & z_{2n'} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m1} & z_{m2} & \cdots & z_{mn'} \end{bmatrix}$$

We denote $\boldsymbol{z}_j$ to be the j-th row of $\boldsymbol{Z}$ and also the latent variable of the j-th session in the batch for each $j \in [1, m]$. We denote $\boldsymbol{\xi}_i$ to be the i-th column of $\boldsymbol{Z}$ for each $i \in [1, n']$.

$\boldsymbol{Z}$ is not the underlying truth value for the latent variable but a sample from a distribution with the underlying truth value as its expected value. Suppose that $\boldsymbol{\mu} \in \mathbb{R}^{m \times n'}$ denotes the correspondence truth values as follows.

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1n'} \\ \mu_{21} & \mu_{22} & \cdots & \mu_{2n'} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{m1} & \mu_{m2} & \cdots & \mu_{mn'} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_1^T \\ \boldsymbol{\mu}_2^T \\ \vdots \\ \boldsymbol{\mu}_m^T \end{bmatrix}$$

The Empirical Bayes' assumption is that $\forall i, j; z_{ij}|\mu_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_j^2)$, which is a normal distribution with mean μ_{ij} and variance σ_j^2, with an additional assumption that $\sigma_j^2 \geqslant 1$. This assumption also means that the variance is the same across different columns. We aim to obtain an estimator for $\boldsymbol{\mu}$ given the observation $\boldsymbol{Z}$. The *Maximum Likelihood Estimator* (MLE) that is commonly used in the field suggests that we should just take the $\boldsymbol{Z}$ itself. That is, for each $i \in [1, m]$ and each $j = [1, n']$,

$$\hat{\mu}_{ij}^{(MLE)} = z_{ij} \tag{10}$$

But, our alleviator uses the *James-Stein Estimator for Session-based Recommendation* (JSE-SR) that applies indirect evidence from other values of the same entry in the batch. The JSE-SR is defined as follows:

$$\hat{\mu}_{ij}^{(JS)} = (1 - \frac{m-2}{\|\boldsymbol{\xi}_j\|^2}) z_{ij} \tag{11}$$

For each of the two estimators $\hat{\mu}_{ij}$ (i.e.,$\hat{\mu}_{ij}^{(MLE)}$ and $\hat{\mu}_{ij}^{(JS)}$), the effect of random user's behavior on the latent variable can be quantified as follows. For each $j \in [1, n']$, $\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij})^2]$.

We can show the following lemma. In this lemma, we know that the estimator $\hat{\mu}_{ij}^{(JS)}$ gives a smaller error compared with the estimator $\hat{\mu}_{ij}^{(MLE)}$.

Lemma 4.1:

$$\mathbb{E}[\sum_{i=1}^{m} (\mu_{ij} - \hat{\mu}_{ij}^{(JS)})^2] \leqslant \mathbb{E}[\sum_{i=1}^{m} (\mu_{ij} - \hat{\mu}_{ij}^{(MLE)})^2] \tag{12}$$

Proof Sketch: Firstly, for all predictor $\hat{\mu}_{ij} := \hat{\mu}_{ij}(z_{ij})$ of μ_{ij}, we can decompose $\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu_{ij}})^2] = \sum_{i=1}^m \mathbb{E}[(z_{ij} - \hat{\mu}_{ij})^2] + 2 \sum_{i=1}^m \mathbb{E}[(\hat{\mu}_{ij} - \mu_{ij})(z_{ij} - \mu_{ij})]$. Secondly, we perform integration by parts, we have: $\mathbb{E}[(z_{ij} - \mu_{ij})(\hat{\mu}_{ij} - \mu_{ij})] = \sigma_j^2 \mathbb{E}[\frac{\partial \hat{\mu}_{ij}}{\partial z_{ij}}]$. Thirdly, we plug the $\hat{\mu}_{ij}^{(JS)}$ and $\hat{\mu}_{ij}^{(MLE)}$ into the equation, we have Equation 11. A complete proof can be found in the supplementary material in https://github.com/RickySkywalker/SR-PredictAO-official/blob/main/Supplementary%20Material.pdf. □

Therefore, applying JSE-SR to all entries in $\boldsymbol{Z}$, we have:

$$\hat{\boldsymbol{Z}}^{(JS)} = [\hat{\mu}_{ij}^{(JS)}]_{i \in [1,m], j \in [1,n']} \tag{13}$$

2) Neural Decision Forest (NDF): As described in Section I, the Neural Decision Forest (NDF) model could be regarded as a *forest* involving a number of *decision trees* each constructed with the use of *Neural Network* (NN) models. Each decision tree in this model is formally named as a *Neural Decision Tree* (NDT).

In the following, we first define NDT and then NDF.

NDT: Our proposed NDT method is the part that provides (more than) enough capability to solve the lack of capability problem of the linear predictor. Considering the representation learning in the session-based recommendation, our proposed NDT differs from the traditional trees that greedily find the split that may reduce the loss function in the given variable space and entries proposed by [20], which requires a fixed encoder, but our proposed NDT uses NN to do the split and are optimized by backward propagation together with the encoder. In our case, this encoder is normally a GNN-based encoder. The NDT that has depth d, and it takes values from alleviator-processed latent variable $\boldsymbol{z}^{(JS)} \in \mathbb{R}^{n'}$ as input. It consists of the following.

- A decision function (normally a deep neural network): $f : \mathbb{R}^{n'} \to \mathbb{R}^{2^d - 1}$ (because a tree with depth d requires $2^d - 1$ number of the split, resulting in 2^d leaf nodes)
- A probability score matrix $\boldsymbol{\pi} \in \mathbb{R}^{2^d \times N}$ (which is trainable) for all leaf nodes:

$$\boldsymbol{\pi} = [\pi_{ij}] = [\boldsymbol{\pi}_1, \cdots, \boldsymbol{\pi}_{2^d}]^T \tag{14}$$

We mark the leaf nodes of a tree from left to right with index $1, 2, \cdots, 2^d$, where the i-th leaf node means the leaf node with index i. Note that under our definition, the NDT is always a balanced tree. π_{ij} means the probability score of the j-th item in the i-th leaf node. $\boldsymbol{\pi}_i$ means a vector containing the probability scores of all items in I of the i-th leaf node.

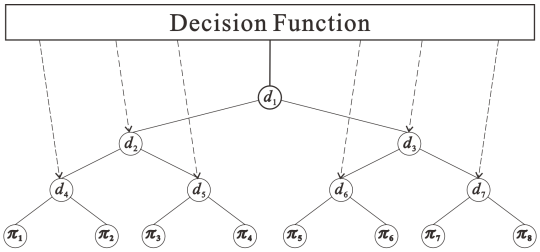

Fig. 2: The overview of the NDT, decision function gives the split score for root and internal nodes, and the leaves nodes' result is the probability of the session reaching the node

The NDT works as follows. The decision function generates a decision score for each split. Then, applying a sigmoid function to the decision score to obtain the right and left decision probability. A binary split is associated with the probability of arriving at the root of this split as p_{root}, which is generated by previous splits. Let $s = \sigma(f(z^{(JS)}))$. The split here means the process of giving an item in the root of the subtree what is the probability that this item goes to the right and left of the root. The probability is calculated as follows.

$$\begin{cases} p_{left} = p_{root} \cdot s \\ p_{right} = p_{root} \cdot (1 - s) \end{cases} \tag{15}$$

For example, in Fig. 2, p_{root} for node d_1 is 1, and p_{root} for node d_2 is set to p_{left} computed within node d_1.

We recursively apply this split method from the tree's root to the leaf nodes. We obtain the leaf-reaching probability $\boldsymbol{p}_{leaf} = [p_1^{(leaf)}, p_2^{(leaf)}, \cdots, p_{2^d}^{(leaf)}]^T \in \mathbb{R}^{2^d}$ to represent what is the probability that this session may fall into each leaf node. Then, multiply $softmax(\boldsymbol{\pi})$ matrix by $\boldsymbol{p}_{leaf}$ to obtain the probability distribution $\hat{\boldsymbol{p}} \in \mathbb{R}^N$ over all items that this session may represent.

$$\hat{\boldsymbol{p}} = \boldsymbol{p}_{leaf}^T softmax(\boldsymbol{\pi}) = \sum_{k=1}^{2^d} p_k^{(leaf)} softmax(\boldsymbol{\pi}_k) \tag{16}$$

where $\hat{\boldsymbol{p}}$ is the predicted probability for each item for this tree. To make $\boldsymbol{\pi}$ normalized, we apply the softmax function before we use it.

Pruning: Because that all tree-based methods, including NDT, suffer from serious overfitting because they normally have excessive capability. The problem is more severe in our case since our NDT is trained simultaneously with the encoder. To solve that problem, we propose *NDT-pruning* that can control the excessive capability to control overfitting.

Traditional pruning uses the judgment of loss function to see which leaves should drop, but for an NDT, it is hard to do a similar thing. Thus, to prune the NDT, we apply a random mask to the outcomes of NDT. So we do the following:

$$\boldsymbol{p}'_{leaf} = softmax(RandomMask(\boldsymbol{p}_{leaf}, r)) \tag{17}$$

where $\boldsymbol{p}_{leaf} \in \mathbb{R}^{2^d}$ is the leaf-reaching probability, and each leaf node has a probability r (we call it pruning rate) to be 0, and $r \in [0, 1]$. After the random mask, we use $\boldsymbol{p}'_{leaf}$ to replace $\boldsymbol{p}_{leaf}$ to obtain $\hat{\boldsymbol{p}}$, the predicted next-item distribution of this tree.

Since NDT typically has excessive capability than needed, which may fit into unrelated information in data, this makes the model easy to overfit. Our proposed NDT-pruning controls the overfitting by removing the excessive capability of the NDT. By choosing a good pruning rate, we can control the capability of our model in a reasonable range that can compensate for the lack of capability in linear predictors and not be too high to overfit. More details of the relation between the model's capability and NDT-pruning can be found in Section V-C

NDF: We construct the NDF by the basic building block NDT and NDT-pruning in this section. Breiman proved that combining trees into a forest model generally makes the model's outcome more stable [21]. Non-neural trees that formulate Random Forest should have a different mask of entries for every split, but that is not possible if we use a uniform decision function for each tree. So, we independently drop some entries for each NDT.

For example, if an input Alleviator-processed latent variable for the NDT is $\boldsymbol{z}^{(JS)} = [z_1^{(JS)} \cdots, z_{n'}^{(JS)}]^T \in \mathbb{R}^{n'}$, for the i-th NDT after the variable mask-off, a fixed subset of $\boldsymbol{z}'_i = [z_{1_i}, z_{2_i}, \cdots, z_{\gamma_i}]$ where $|\boldsymbol{z}'_i| = \gamma_i \leqslant n'$, and $\boldsymbol{z'} \subseteq \boldsymbol{z}^{(JS)}$. For each NDT, the list of entries to drop is randomly selected when building the model, but this list is fixed during training. If there are T number of NDTs in the NDF-SR, and their predicted next-item probability is $\boldsymbol{P} = [\hat{\boldsymbol{p}}_1, \hat{\boldsymbol{p}}_2, \cdots, \hat{\boldsymbol{p}}_T]$, where $\hat{\boldsymbol{p}}_i \in \mathbb{R}^N$ for all $i = 1, 2, \cdots, T$. The NDF's predicted result is:

$$\hat{\boldsymbol{y}}_{NDF-SR} = \frac{1}{T} \cdot (\sum_{i=1}^{T} \hat{\boldsymbol{p}}_i) \tag{18}$$

which is also the predicted result of the NDF-SR, our proposed high-capability predictor.

Time complexity analysis: Under tree-parallel setting, the NDF-SR module's time complexity is $\mathcal{O}(m \cdot n' \cdot k + m \cdot k \cdot N)$, where k is the number of leaves in the tree (typically 32 to 64). This is only slightly higher than the $\mathcal{O}(m \cdot n' \cdot N)$ for traditional linear predictors.

B. Merger Module

In this paper, for the second *Merger* module, we adopt a simple linear combination which combines the predicted distributions from the original predictor and the new predictor to obtain the final predicted probability distribution by using a user parameter $q \in [0,1]$ as follows.

$$\hat{\boldsymbol{y}} = q \cdot \hat{\boldsymbol{y}}_{base} + (1-q) \cdot \hat{\boldsymbol{y}}_{NDF-SR} \tag{19}$$

Here, $\hat{\boldsymbol{y}} \in \mathbb{R}^N$ is the probability distribution over all items predicted being the next item (which is the combined result from the original predictor module and the new predictor module). $\hat{\boldsymbol{y}}$ is the output of framework *SR-PredAO*.

C. Training

Note that $\hat{\boldsymbol{y}}$ obtained in module Merger is the output of framework *SR-PredAO*. Let $\boldsymbol{y}$ be the real probability distribution over all items being the next item, which is a one-hot vector. The loss function of framework *SR-PredAO* $\mathcal{L}(\cdot,\cdot)$ is the same as the one used in the base model, which is the cross-entropy loss.

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\boldsymbol{y}^T \log(\hat{\boldsymbol{y}}) \tag{20}$$

For initialization, all trainable parameters in both the base model and the additional modules in framework *SR-PredAO* are initialized randomly, and they are jointly updated in an end-to-end back propagation manner.

V. Experiment

We give the experiment setup in Section V-A and the results in Section V-B. Implementation of this paper can be found in https://github.com/RickySkywalker/SR-PredictAO-official

A. Experimental Setup

1) Datasets: We evaluated the performance of state-of-the-art models and our proposed framework on the following two benchmark real-world datasets:

- *Yoochoose*[1] is a dataset obtained from the RecSys Challange 2015, which contains user sessions of click events from an online retailer.
- *Diginetica*[2] is a dataset released by the CIKM Cup 2016, which includes user sessions extracted from e-commerce search engine logs.

Our dataset preprocess directly following [3], [9], [17]. The statistics of the datasets after pre-processing are provided in Table I.

Statistic	**Yoochoose 1/64**	**Diginetica**
# of Clicks	565,332	982,961
# of Training Sessions	375,625	647,523
# of Test Sessions	55,896	71,947
# of Items	17,792	43,097
Average length	6.14	5.12

TABLE I: Statistics of datasets

2) Evaluation Metrics: Following previous studies [2]–[4], [8], [9], [28]–[30], we adopt the commonly used HR@20 (Hit Rate)[3] and MRR@20 (Mean Reciprocal Rank) as our evaluation metrics.

3) Base Model & Baselines: Framework *SR-PredAO* involves a base model (together with our proposed high-capability predictor module and the Merger module). In our experiments, we choose the following three base models, namely LESSR [3], SGNN-HN [9] and DIDN [4], since they are representative in the literature. Roughly speaking, LESSR has a clear encoder-predictor paradigm for the ease of illustration and conducting subsequent experiments. SGNN-HN and DIDN have the best performance on datasets Yoochoose 1/64 and Diginetica, respectively.

We also considered using some newer proposed models like [5]–[7] as base models, but their performance is less satisfactory in our benchmarks and thus not demonstrated in the paper. In the following, when we describe framework *SR-PredAO* using the base model M, we write *SR-PredAO(M)*.

To more effectively illustrate the superiority of our framework's capability in enhancing models. In addition to three base models without enhancement as baselines, we have also included other baselines in the comparison. These encompass traditional recommendation methods such as **Item-KNN** [31], the GRU-based method **GRU4REC** [16], two transformer-like methods, **STAMP** [32] and **SR-IEM** [33], and a basic GNN-based method, **SR-GNN** [8].

4) Implementation Details: In the *SR-PredAO* framework, hyper-parameters (e.g., batch size and learning rate) for the base models are kept as the best experimental configurations reported in their papers [3], [4], [9]. This allows us to observe the improvements made by the *SR-PredAO* framework, which includes the new predictor module and the merger module, over the base models. The additional hyper-parameters are determined through binary search. Additionally, every reported result is the best outcome for both the baselines and the *SR-PredAO*-enhanced models, and these configurations may vary. We also use the accuracy of the training data as the validation set for model selection. The training averagely costs 3 RTX-4090 GPU days.

B. Experimental Results

1) Performance Comparison: Table II shows the experimental results for all models, we can see that *SR-PredAO* has a relatively significant improvement on HR@20 for all models and on MRR@20 for almost all models. Specifically, framework *SR-PredAO*, when applied to existing state-of-the-art models,could have up to 2.9% improvement on HR@20 and 2.3% of improvement on MRR@20. Furthermore, to test the significance of our *SR-PredAO*, we conducted the two-proportion z-tests and reported p-value of such test in the table. We can spot that when the base model has a good

[1] http://2015.recsyschallenge.com/challenge.html

[2] http://cikm2016.cs.iupui.edu/cikm-cup

[3] Note that [2]–[4], [8], [9], [28]–[30] used different metric names for HR@20 (e,g, P@20 and Recall@20). But, they used the same formula to obtain this measurement (i.e., the proportion of cases when the target item is in the top-20 items in all test cases).

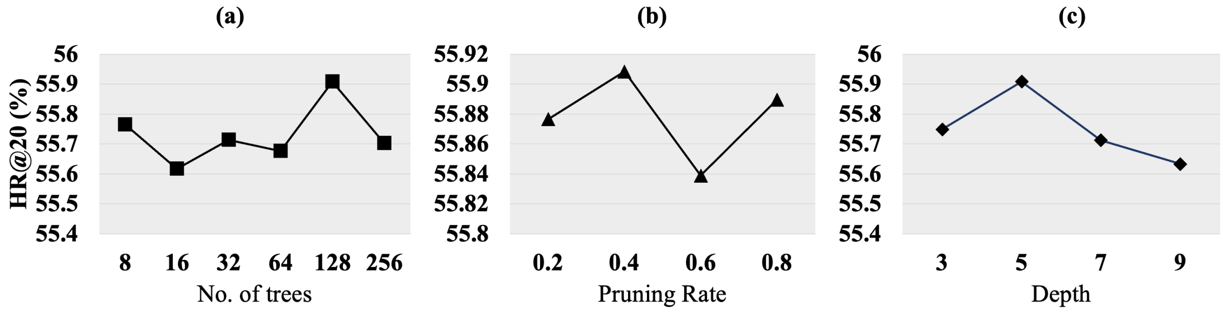

Fig. 3: The hyper-parameter study results of SR-PredAO(SGNN-HN)

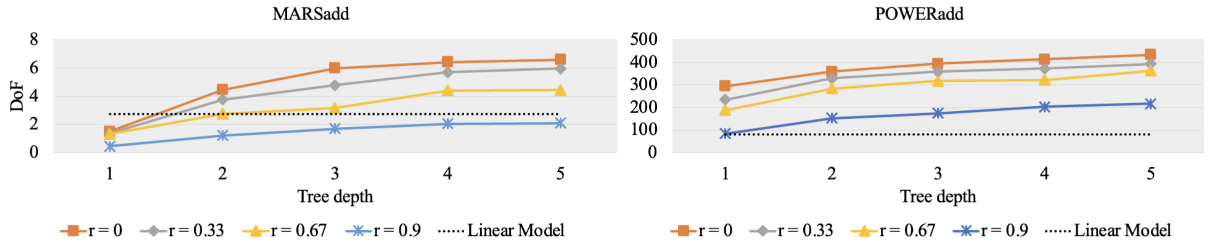

Fig. 4: DoF of NDF-SR with different depth and pruning rates (r), dotted lines represent DoF of linear model

Method	Diginetica		Yoochoose 1/64	
	HR@20	MRR@20	HR@20	MRR@20
Item-KNN	35.75	11.57	51.60	21.81
GRU4REC	29.45	8.33	60.64	22.89
STAMP	45.64	14.32	68.74	29.67
SR-IEM	52.35	17.64	71.15	31.71
SR-GNN	50.73	17.59	70.57	30.94
LESSR	51.71	18.15	70.94	31.16
SR-PredAO(LESSR)	**53.10**	**18.38**	**71.73**	**31.70**
Improvement (%)	**2.7**	**1.3**	**1.1**	**1.7**
p-value	$< \mathbf{10^{-5}}$	-	$\mathbf{1.8 \times 10^{-3}}$	-
SGNN-HN	55.67	**19.12**	72.06	**32.61**
SR-PredAO(SGNN-HN)	**55.91**	19.06	**72.62**	32.47
Improvement (%)	**0.4**	-0.3	**0.8**	-0.4
p-value	0.179	-	$\mathbf{1.8 \times 10^{-2}}$	-
DIDN	56.22	20.03	68.95	31.27
SR-PredAO(DIDN)	**57.86**	**20.49**	**69.50**	**31.44**
Improvement (%)	**2.9**	**2.3**	**0.8**	**0.5**
p-value	$< \mathbf{10^{-5}}$	-	$\mathbf{2.3 \times 10^{-2}}$	-

TABLE II: Experimental result (%) on three enhanced models and baselines on two datasets

encoder-predictor split (i.e., DIDN and LESSR). The *SR-PredAO* enhancement statistically significantly outperforms the base models. According to Paper-with-code, *SR-PredAO* achieves state-of-the-art on all experimented benchmarks on HR@20 and almost all on MRR@20.

It is worth mentioning that using framework *SR-PredAO* on any existing model could automatically improve the prediction accuracy, which is a great advantage. Compared with recent papers [2]–[7], [9] showing that 1.4% of improvement is considered as a major contribution, framework *SR-PredAO* has a significant improvement in the field.

2) Ablation Studies: This section presents the ablation studies results for two important components in *SR-PredAO*, namely Random User Behavior Alliviator and NDT-pruning

Dataset	type	LESSR	SGNNHN	DIDN
Diginetica	full model	**53.15**	**55.91**	**57.86**
	w/o Alleviator	52.99	55.79	57.33
	w/o Pruning	53.06	55.78	57.26
YC 1/64	full model	**71.73**	**72.62**	**69.50**
	w/o Alleviator	71.67	72.58	69.26
	w/o Pruning	71.66	72.58	69.20

TABLE III: Ablation test results (%) on random user's behavior alleviator (Alleviator) and NDT-pruning (Pruning)

Table III shows that if we drop the random user's behavior alleviator or NDT-Pruning in framework *SR-PredAO*, the improvement of *SR-PredAO* over the base model drops to a great extent in the Diginetica dataset but not that much in *YooChoose* (YC) 1/64 dataset. This is because the YC dataset is simpler compared with Diginetica. And in YC, the random user behavior and the overfitting problem are not that obvious to a certain extent.

3) Hyper-parameter Study: In this section, we study how the number of trees, the depth of the tree, and the pruning rate affect the performance of *SR-PredAO*. All the results are shown in Fig. 3. When the number of trees reaches 128, HR@20 of *SR-PredAO* is the highest. When the number of trees is larger 128, HR@20 decreases because more trees affects the model's learning capacity. For the pruning rate, as long as we do not remove the pruning feature, we can see that varying the rate does not affect the performance too much. For the depth of the tree, we can see that if the tree goes too deep (i.e., the depth is greater than 5), it may have a serious overfit problem due to the excessive capability, and if the tree is too shallow (i.e., the depth is smaller than 5), it cannot provide enough capability enhancement for prediction.

4) Model Size Comparison: In order to perform a fair comparison between the base model (without using our framework) and our framework, we conduct experiments so that they have the same model complexities. Specifically, after we obtain SR-PredAO(SGNN-HN), we enlarge the base model (i.e., SGNN-HN) by increasing the embedding dimensionality and this base model (without using our framework), after parameter-tuning, is regarded as a baseline. The experimental result on Diginetica is shown in Table IV. The enlarged base model cannot outperform SR-PredAO(SGNN-HN) due to the inappropriate training capacity increment of the base model.

Model	**Size**	**HR@20**	**MRR@20**
Enlarged SGNN-HN	1605MB	55.24	18.64
SR-PredAO(SGNN-HN)	1118MB	**55.91**	**18.78**

TABLE IV: Size comparison (%) result on Diginetica

C. Degrees-of-Freedom Study

This section gives a comprehensive and quantitative study of the capability of the NDF-SR module to show the wide bandwidth of capability our model can provide. As stated in Section III-A, we use *Degrees-of-Freedom* (DoF) to formally define the capability of a model in this paper. Following [23], the DoF is a good illustration of model's capability. It is defined as follows: Assuming we have a dataset $D_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M}$, where $\boldsymbol{x}_i \in \mathbb{R}^{n'}$ and $\varepsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$. The relationship between $\boldsymbol{x}$ and y is in the form $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$, where f is the true relation between $\boldsymbol{x}$ and y. The model we fit to estimate f is denoted as $\hat{f}$, in our case, and prediction is $\hat{y}_i = \hat{f}(\boldsymbol{x}_i)$, it can be either the base model or *SR-PredAO* enhanced model. Then, the DoF is defined as $DoF(\hat{f}) = \frac{1}{\sigma^2}\sum_{i=1}^{N}\text{Cov}(\hat{y}_i, y_i)$.

()SMince in the session-based recommendation problem, we do not know the true function f, we perform DoF analysis on two simulated underlying functions, namely the "MARSadd" [23]: $y_i = 0.1e^{4x_{i1}} + \frac{4}{e^{-20(x_{i2}-0.5)}} + 3x_{i3} + 2x_{i4} + x_{i5} + \varepsilon_i$, where $\epsilon \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ and $\boldsymbol{x}_i = [x_{i1}, \cdots, x_{i5}]^\top$ are randomly sampled from uniform distribution between 0 and 1. Another underlying function is we proposed "POWERadd": $y_i = \sum_{j=1}^{5} x_{ij} + \sum_{j=6}^{10} x_{ij} + \varepsilon_i$, this function in order to test the model's DoF behavior under high-order curvature and extra dimension. Data and error terms are sampled as "MARSadd".

The experimental results are demonstrated in Fig. 4. We can observe that except for extreme cases (like $r = 0.9$ or $depth = 1$), NDF-SR shows significantly higher DoF than the linear model. Additionally, by controlling depth and pruning rate (r), we can achieve a very flexible change in DoF in both experiments. This is further evidence of the effectiveness of pruning. From the results of simple simulated functions, we can easily extrapolate that the linear model suffers from a low-capability problem, while the NDF-SR (or DoF) we proposed can provide a higher and more controllable capability.

D. Summary

In summary, framework *SR-PredAO*, when applied to existing state-of-the-art models, could have up to 2.9% improvement on HR@20 and 2.3% of improvement on MRR@20. We can observe this improvement in almost all base models on all datasets. By considering the consistency of improvement and the ease of applicability of our framework, we regard our contribution as a major improvement to the field of the session-based recommendation system.

VI. Conclusion

In this paper, we are the first to discover the important low-capability issue in the predictor module of most (if not all) existing models, lowering down their prediction accuracy. To address this important issue, we propose a framework called *SR-PredictAO* which could be applied to any existing models following the common encoder-predictor paradigm. Extensive experimental results on two public benchmark datasets show that when framework *SR-PredictAO* is applied to 3 existing state-of-the-art models, their performance are consistently improved up to 2.9% on HR@20 and up to 2.1% on MRR@20. Due to the consistent improvement on all datasets, we regard our contribution as a major improvement to the field of the session-based recommendation system.

VII. Future Work

Although SR-PredAO sets an effective and general enhancement that can be applied for all models that lack capability in modeling complex underlying behaviors. There are many future possible studies that can be developed based on SR-PredAO. Firstly, the cost of this framework is relatively high when the number of leaf nodes is large. Thus, how to develop an efficient tree-based method is a potential direction. Secondly, this paper only discusses the task in session-based recommendation, and the tree-based enhancement can be applied to a wider field such as language and vision modeling. Thirdly, the theoretical foundation for neural-based trees also needs to be built up by future studies.

VIII. Acknowledgement

We are grateful to the anonymous reviewers for their constructive comments. The research is supported in part by WEB24EG01-H.

References

[1] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. V. Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," in *RecSys*, 2010, pp. 293–296.

[2] Y. Zheng, S. Liu, Z. Li, and S. Wu, "Dgtn: Dual-channel graph transition network for session-based recommendation," in *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2020, pp. 236–242.

[3] T. Chen and R. C.-W. Wong, "Handling information loss of graph neural networks for session-based recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1172–1180.

[4] X. Zhang, H. Lin, B. Xu, C. Li, Y. Lin, H. Liu, and F. Ma, "Dynamic intent-aware iterative denoising network for session-based recommendation," *Information Processing & Management*, vol. 59, no. 3, p. 102936, 2022.

[5] R. Yeganegi and S. Haratizadeh, "Star: A session-based time-aware recommender system," *arXiv preprint arXiv:2211.06394*, 2022.

[6] P. Zhang, J. Guo, C. Li, Y. Xie, J. B. Kim, Y. Zhang, X. Xie, H. Wang, and S. Kim, "Efficiently leveraging multi-level user intent for session-based recommendation via atten-mixer network," in *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, 2023, pp. 168–176.

[7] Z. Pan, F. Cai, W. Chen, C. Chen, and H. Chen, "Collaborative graph learning for session-based recommendation," *ACM Transactions on Information Systems (TOIS)*, vol. 40, no. 4, pp. 1–26, 2022.

[8] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural network," in *AAAI*, 2019, pp. 346–353.

[9] Z. Pan, F. Cai, W. Chen, H. Chen, and M. de Rijke, "Star graph neural networks for session-based recommendation," in *Proceedings of the 29th ACM international conference on information & knowledge management*, 2020, pp. 1195–1204.

[10] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[11] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," *Advances in neural information processing systems*, vol. 20, 2007.

[12] Y. Koren and R. Bell, "Advances in collaborative filtering. recommender systems handbook, francesco ricci, lior rokach, bracha shapira, paul b. kantor editors, chapter 5," 2011.

[13] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston *et al.*, "The youtube video recommendation system," in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 293–296.

[14] S. E. Park, S. Lee, and S.-g. Lee, "Session-based collaborative filtering for predicting the next song," in *2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering*. IEEE, 2011, pp. 353–358.

[15] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 811–820.

[16] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[17] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1419–1428.

[18] T. X. Tuan and T. M. Phuong, "3d convolutional networks for session-based recommendation with content features," in *Proceedings of the eleventh ACM conference on recommender systems*, 2017, pp. 138–146.

[19] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He, "A simple convolutional generative network for next item recommendation," in *Proceedings of the twelfth ACM international conference on web search and data mining*, 2019, pp. 582–590.

[20] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[22] J. R. Quinlan, "Decision trees and decision-making," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 339–346, 1990.

[23] L. Mentch and S. Zhou, "Randomization as regularization: A degrees of freedom explanation for random forest success," 2020.

[24] D. Richmond, D. Kainmueller, M. Y. Yang, E. Myers, and C. Rother, "Relating cascaded random forests to deep convolutional neural networks for semantic segmentation," 07 2015.

[25] J. Jancsary, S. Nowozin, and C. Rother, "Loss-specific training of non-parametric image restoration models: A new state of the art," in *European Conference on Computer Vision*. Springer, 2012, pp. 112–125.

[26] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò, "Deep neural decision forests," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1467–1475.

[27] C. Stein, "Variate normal distribution," in *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability: Contributions to the Theory of Statistics*, vol. 1. University of California Press, 1956, p. 197.

[28] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *CIKM*, 2017, pp. 1419–1428.

[29] P. Ren, Z. Chen, J. Li, Z. Ren, J. Ma, and M. de Rijke, "RepeatNet: A repeat aware neural recommendation machine for session-based recommendation," in *AAAI*, 2019, pp. 4806–4813.

[30] R. Qiu, J. Li, Z. Huang, and H. Yin, "Rethinking the item order in session-based recommendation with graph neural networks," in *CIKM*, 2019, pp. 579–588.

[31] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

[32] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang, "Stamp: short-term attention/memory priority model for session-based recommendation," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1831–1839.

[33] Z. Pan, F. Cai, Y. Ling, and M. de Rijke, "Rethinking item importance in session-based recommendation," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020, pp. 1837–1840.

APPENDIX

A. Proof of Alleviator

1) Assumption 1: All session data are i.i.d. (i.e., independent and identically distributed) samples affected by random user's behavior under some uniform distribution $\mathbb{P}_{\mathbf{s}}$.

i.e. For a set of collected sampled sessions: $\{\boldsymbol{s}_i\}_{i=1}^m$, where $\boldsymbol{s}_i = [s_{i,1}, s_{i,2}, \cdots, s_{i,l_i}]$ as we defined in Section III-A. For all i, we have: $\boldsymbol{s}_i - \epsilon_i \sim \mathbb{P}_s$ where ϵ_i is the random user's behavior, which can be intuitively understood as the user's random behavior

2) Assumption 2 (Empirical Bayes Assumption I): The encoded value of a session (i.e., z_{ij} in Section IV-A1) is not the real value, just an observation affected by random user's behavior:

i.e., for a not affected session $\boldsymbol{s}_i - \epsilon_i$; the real encoded latent variable is $\boldsymbol{\mu}_i = [\mu_{i1}, \cdots, \mu_{in'}]^T$; For a fixed $j = 1, 2, \cdots, n'$ (where n' is the dim for latent variable), the $\{\mu_{ij}\}_{i=1}^m$ and follows distribution $\mathbb{P}_s^{(j)}$.

That means the real value for $j-th$ propriety of the $i-th$ session should be μ_{ij}; But due to the effect of random user's behavior, the encoded result we observe from the session-encoder is z_{ij}

3) Assumption 3 (Empirical Bayes Assumption II): The observed value of the encoded session z_{ij} follows the distribution of $\mathcal{N}(\mu_{ij}, \sigma_j^2)$ and $\sigma_j \geqslant 1$ (if this assumption is not met, we can always do batch normalization to make the σ_j not far from 1).

The Normal distribution assumption comes from the statistic common that if a distribution is affected by extremely complex factors, like the random user's behavior. The safest way is to assume that they are normally distributed. Since all numbers in $\{z_{ij}\}_{i=1}^m$ represent the same factor of the session (the $j-th$ encoded factor), it is reasonable to assume they have the same and relatively large variance.

4) Target: In high-level understanding, what we observed in the real data is not the full fact but noisy data that have information of the underlying true value. Our goal is to obtain the underlying true value (i.e., μ_{ij} in our case) through observed values (the z_{ij} in our case).

The rigorous definition of the target is: given a batched, observed encoded result: $\boldsymbol{Z} \in \mathbb{R}^{m \times n'}$ and its corresponding underlying true value $\boldsymbol{\mu} \in \mathbb{R}^{m \times n'}$

For a fixed $j \in [1, n']$, get an estimator $\hat{\mu}_{ij} | \boldsymbol{\xi}_j$ for μ_{ij} s.t. $\mathbb{E}[\sum_{i=1}^m (\hat{\mu}_{ij} - \mu_{ij})^2]$ is small.

Consider the Max likelihood estimator $\hat{\mu}_{ij}^{(MLE)} = z_{ij}$, and $\hat{\mu}_{(JS)}^{ij} = (1 - \frac{m-2}{\|\boldsymbol{\xi}_j\|^2}) z_{ij}$.

Claim that: $\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij}^{(JS)})^2] \leqslant \mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij}^{(MLE)})^2]$

5) Proof of claim: $\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij})^2] = \sum_{i=1}^m \mathbb{E}[(z_{ij} - \hat{\mu}_{ij})^2 - (z_{ij} - \mu_{ij})^2 + 2(\hat{\mu}_{ij} - \mu_{ij})(z_{ij} - \mu_{ij})] = \sum_{i=1}^m \mathbb{E}[(z_{ij} - \hat{\mu}_{ij})^2] - m \cdot \sigma_j + 2 \sum_{i=1}^m \mathbb{E}[(\hat{\mu}_{ij} - \mu_{ij})(z_{ij} - \mu_{ij})]$

Consider distribution function for z_{ij} as: $\varphi(z_{ij} | \mu_{ij}, \sigma_j) = \frac{1}{\sqrt{2\pi} \cdot \sigma_j} \exp(-\frac{(z_{ij} - \mu_{ij})^2}{2\sigma_j^2})$. Therefore, $(z_{ij} - \mu_{ij})(\hat{\mu}_{ij} - \mu_{ij}) = -\sigma_j^2 \frac{\partial}{\partial z_{ij}} \varphi(z_{ij} | \mu_{ij}, \sigma_j)$

Therefore, for any continuous, differentiable, and $|f(z)| < \infty$, function $f : \mathbb{R} \to \mathbb{R}$. For simplicity, denote $\varphi(z_{ij} | \mu_{ij}, \sigma_j)$ as $\varphi(z_{ij})$ we have:

$$
\begin{aligned}
&\mathbb{E}[(z_{ij} - \mu_{ij}) f(z_{ij})] \\
=& \int_{-\infty}^{+\infty} (z_{ij} - \mu_{ij}) f(z_{ij}) \varphi(z_{ij}) dz_{ij} \\
=& (-\sigma_j^2) \cdot (\int_{-\infty}^{+\infty} (\frac{\partial}{\partial z_{ij}} \varphi(z_{ij})) f(z_{ij}) dz_{ij}) \\
=& (-\sigma_j^2) \cdot \varphi(z_{ij}) \cdot f(z_{ij}) |_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} f'(z_{ij}) \varphi(z_{ij}) dz_{ij} \\
=& \sigma_j^2 \cdot (\int_{-\infty}^{+\infty} f'(z_{ij}) \varphi(z_{ij}) dz_{ij}) \\
=& \sigma_j^2 \mathbb{E}[\frac{\partial}{\partial z_{ij}} f(z_{ij})]
\end{aligned}
$$

Therefore, we have: $\mathbb{E}[(z_{ij} - \mu_{ij})(\hat{\mu}_{ij} - \mu_{ij})] = \mathbb{E}[\frac{\partial \hat{\mu}_{ij}}{\partial z_{ij}}] \cdot \sigma_j^2$. Therefore, when $\hat{\mu}_{ij}$ is MLE: $\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij}^{(MLE)})^2] = 0 - m \cdot \sigma_j^2 + 2m \cdot \sigma_j^2 = m \cdot \sigma_j^2$ When $\hat{\mu}_{ij}$ is $\hat{\mu}_{ij}^{(JS)}$. We have: $\mathbb{E}[\frac{\partial \hat{\mu}_{ij}^{(JS)}}{\partial z_{ij}}] = 1 - \frac{m-2}{\|\boldsymbol{\xi}_j\|^2} + \frac{2(m-2) z_{ij}^2}{\|\boldsymbol{\xi}_j\|^4}$

Therefore, $\sum_{i=1}^m \mathbb{E}[(\hat{\mu}_{ij}^{(JS)} - \mu_{ij})(z_{ij} - \mu_{ij})] = m - \mathbb{E}[\frac{(m-2)^2}{\|\boldsymbol{\xi}_j\|^2}]$

With $\mathbb{E}[(z_{ij} - \hat{\mu}_{ij}^{(JS)})^2] = \mathbb{E}[\frac{(m-2)^2}{\|\boldsymbol{\xi}_j\|^2} z_{ij}^2]$, we have: $\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij}^{(JS)})^2] = m \cdot \sigma_j^2 + m(1 - 2\sigma_j^2) \mathbb{E}[\frac{(m-2)^2}{\|\boldsymbol{\xi}_j\|^2}]$

Since in our assumption $\sigma_j^2 \geqslant 1$, we have:

$$
\mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij}^{(JS)})^2] \leqslant \mathbb{E}[\sum_{i=1}^m (\mu_{ij} - \hat{\mu}_{ij}^{(MLE)})^2] \tag{21}
$$