MetaSelector: Meta-Learning for Recommendation with User-Level Adaptive Model Selection

Mi Luo *
Huawei Noah's Ark Lab
rosemaryluo@outlook.com

Fei Chen
Huawei Noah's Ark Lab
chen.f@huawei.com

Pengxiang Cheng
Huawei Noah's Ark Lab
chengpengxiang1@huawei.com

Zhenhua Dong
Huawei Noah's Ark Lab
dongzhenhua@huawei.com

Xiuqiang He
Huawei Noah's Ark Lab
hexiuqiang1@huawei.com

Jiashi Feng
National University of Singapore
elefjia@nus.edu.sg

Zhenguo Li
Huawei Noah's Ark Lab
li.zhenguo@huawei.com

1 INTRODUCTION

In recommender systems, deep learning has played an increasingly important role in discovering useful behavior patterns from huge amount of user data and providing precise and personalized recommendation in various scenarios [39, 6, 19, 40]. Data from one user may be sparse and insufficient to support effective model training. In practice, deep neural networks are trained collaboratively on a large number of users, while it is important to distinguish the specific users to make personalized recommendation. Certain user identification processes are therefore often performed in alignment with the model training procedure, such as encoding a unique ID or user history information for each user[41], or fine-tuning the recommender on user local data before making recommendations [4]. On the other hand, various deep models have been proposed and experimentally verified effective in terms of average performance on the whole datasets.

Although certain recommendation models could achieve better overall performance than other models, it is unlikely that there is a single model that performs better than other models for every user[13, 11]. In other words, the best performance on different users may be achieved by different recommendation models. We observed this phenomenon on both private production and public datasets. For instance, in an online advertising system, multiple CTR prediction models are deployed simultaneously. We found that no single model performs best on all users. Moreover, in terms of averaged evaluation, no single model achieves the all-time best performance. This implies that the performance of recommendation models is sensitive to user-specific data. Consequently, user-level model design in deep recommender systems is of both research interests and practical values.

In this work, we address the problem of user-level model selection to improve personalized recommendation quality. Given a collection of deep models, the goal is to select the best model from them for each individual user or to combine these models to maximize their strengths. We introduce a model selector on top of specific recommendation models that outputs a decision of which model to use for an user. In particular, the model selector that we focus on adopts the recently revived meta-learning methodology [33, 37, 1, 38, 30, 14], thus called MetaSelector.

Meta-learning algorithms learn to efficiently solve new tasks by extracting prior information from a number of related tasks. Of particular interest are optimization-based approaches, such as the popular Model-Agnostic Meta-Learning (MAML) algorithm [14], that apply to a wide range of models whose

*This work was done when Mi Luo was intern at Huawei Noah's Ark Lab.

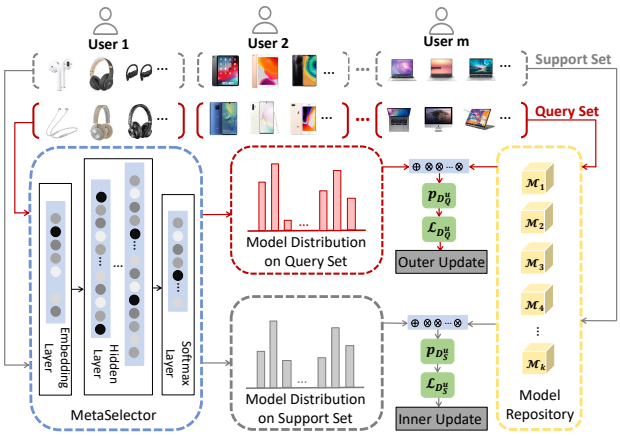


Figure 1: The MetaSelector framework.

parameters are updated by stochastic gradient descent (SGD), with little requirement on the model structure. MAML involves a bi-level meta-learning process. The outer loop is on task level, where the algorithm maintains an *initialization* for the parameters. The objective is to optimize the initialization such that when applied to a new task, the initialization leads to optimal performance on the test set after one or a few gradient updates on the training set. The inner loop is on sample level and executed within tasks. Receiving the initialization maintained in the outer loop, the algorithm adapts parameters on the support (training) set and evaluates the model on the query (test) set. The evaluation result on test set returns a loss signal to the outer loop. After meta-training, in the meta-testing or deployment phase the learned initialization enables fast adaptation on new tasks.

As shown in Figure 1, in our method, we use optimization-based meta-learning methods to construct MetaSelector that learns to make model selection from a number of tasks, where a task consists of data from one user. Given a data point as input, MetaSelector outputs a probability distribution over the recommendation models. We consider MAML as a concrete example, although the framework extends to other optimization-based methods. In the meta-training phase, MAML optimizes an initialization for MetaSelector through episodic learning. In each episode, a batch of users are sampled, each of which contains a support set and a query set. On the support set of each task, a soft model selection is made based on the output of MetaSelector. The parameters of MetaSelector are updated using the training loss obtained by comparing the final prediction with ground truth. Then MetaSelector is evaluated on the query set, and test loss is similarly computed to update the initialization in the outer loop. The recommendation models are updated together in the outer loop, which can be optionally pre-trained before the meta-training process. In the deployment phase, with the learned initialization, MetaSelector adapts to individual users using personalized historical data (support sets), and aggregates results of recommendation models for new queries.

We experimentally demonstrate the effectiveness of our proposed method on two public datasets and a production dataset. In all experiments, MetaSelector significantly improves over baseline models in terms of AUC and LogLoss, indicating that MetaSelector can effectively weigh towards better models at the user level. We also observe that pre-training the recommendation models is crucial to express the power of MetaSelector.

Contributions. To summarize, our contributions are three-fold. Firstly, we address the problem of model selection for recommender systems, motivated by the observation of varying performance of different models among users on public and production datasets. Secondly, we propose a novel framework MetaSelector which introduces meta-learning to formulate a user-level model selection module in hybrid recommender system. This framework can be trained end-to-end and requires no manual definition of meta-features. To the best of our knowledge, this is the first work to study recommendation model selection problem from the optimization-based meta-learning perspective. Thirdly, we run extensive experiments on both public and private production datasets to provide the insight into which level to optimize in model selection. The results indicate that MetaSelector can improve the performance over single model baseline and sample-level selector, showing the potential of MetaSelector in real-world recommender systems.

2 RELATED WORK

Since we study how to apply meta-learning for model selection in a hybrid recommender system, we first survey relevant work on meta-learning and model selection. Besides, our initial observations about the varying performances of recommendation models occurred in a real-world industrial CTR prediction problem. Hence we also review some classic CTR prediction models and mainly focus on CTR prediction task when considering the experimental design.

2.1 Optimization-Based Meta-Learning

In meta-learning, or "learning to learn", the goal is to learn a model on a collection of tasks, such that it can achieve fast adaptation to new tasks [5]. One research direction is metric-based meta-learning, aiming to learn the similarity between new tasks and previous tasks. Representative works include Matching Network [38] and Prototypical Networks [36]. Another promising direction is optimization-based meta-learning which has recently demonstrated effectiveness on few-shot classification problems by "learning to fine-tune". Among the various methods, some focus on learning an optimizer such as the LSTM-based meta-learner [30] and the Meta Networks with an external memory [27]. Another research branch aims to learn a good model initialization [14, 24, 28], such that the model has optimal performance on a new task with limited samples after a small number of gradient updates. In our work, we consider MAML [14] and Meta-SGD [24] which are model- and task-agnostic. These optimization-based meta-learning algorithms promise to extract and propagate transferable representations of prior tasks. As a result, if we regard each task as learning to predict user preference for selecting recommendation models, each user will not only receive personalized model selection suggestions but also benefit from the choices of other users who have similar latent features.

2.2 Model Selection for Recommender Systems

In recommender systems, there is no single-best model that gives the optimal results for each user due to the heterogeneous data distributions among users. This means that the recommendation quality largely varies between different users [12] and some users may receive unsatisfactory recommendations. One way to solve this problem is to give users the right to choose or switch the recommenders. As a result, explicit feedback can be collected from a subset of users to generate initial states for new users [13, 10, 32]. Another solution is a hybrid recommender system [2], which combines multiple models to form a complete recommender. This type of recommender has been proven to be useful for trimming error because it can blend the strengths of different recommendation models. There are two types of methods to hybridize recommenders. One is to make a soft selection choice, that is, to compute a linear combination of individual scoring functions of different recommenders. A well-known work is feature-weighted-linear-stacking (FWLS) [35] which learns the coefficients of model predictions with linear regression. The other line of research is to make a hard decision to select the best individual model for the entire dataset [8, 9], for each user [11] or for each sample [7]. However, most of the works mentioned above are limited to collaborative filtering algorithms and require manually defined meta-features which is very time-consuming. Besides, despite the considerable performance improvement, methods like FWLS mainly focus on sample-level optimization which lacks interpretability about why some models work well for particular users, but not for others. In contrast, our proposed MetaSelector can be trained end-to-end without extra meta-features. To our knowledge, our proposed framework is the first to explore the model selection problem for CTR Prediction, rather than collaborative filtering. We also provide an insight into which level to optimize in model selection by conducting extensive experiments for sample-level and user-level model selection.

2.3 CTR Prediction

Click-through rate (CTR) prediction is an important task in cost-per-click (CPC) advertising system. Model architectures for CTR prediction have evolved from shallow to deep. As a simple but effective model, Logistic Regression has been widely used in the advertising industry [3, 26]. Considering feature conjunction, [31] presented Factorization Machines (FMs) which learns the weight of feature conjunction by factorizing it into a product of two latent vectors. As a variant of FM, Field-aware Factorization Machines (FFM) has been proven to be effective in some CTR prediction

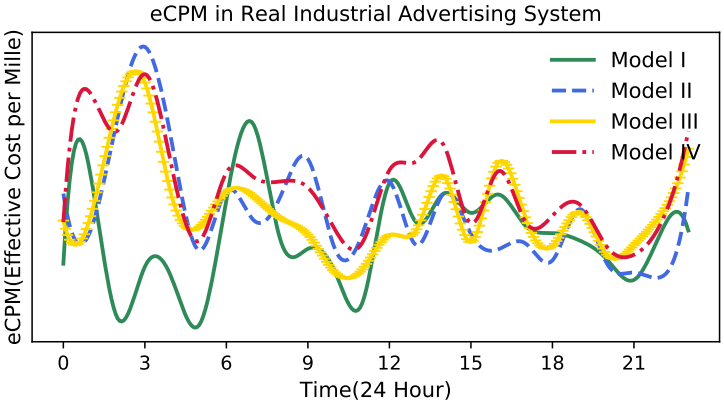


Figure 2: The performances of four models in one day.

competitions [20, 21]. To capture higher-order feature interactions, model architectures based on deep networks have been subsequently developed. Examples include Deep Crossing [34], Wide & Deep [6], PNN [29], DeepFM [15] and DIN [41].

Different from the above studies which focus on the design of model architectures, our work provides novel insight on how to select the most suitable model for individual users and how to combine these existing models to maximize their strengths.

3 PERFORMANCE ANALYSIS

In this section, we firstly present our observations about the varying online performance of recommendation models in a real industrial advertising system. Next, we conduct some pilot experiments to quantify this phenomenon with two public datasets.

3.1 Model Performance in Online Test

In order to compare the performances of different models, we implement four state-of-the-art CTR prediction models, including shallow models and deep models. Then we deploy these models in a large-scale advertising system to verify the varying performances of them through online A/B test.

Experimental Setting. Users have been split into four groups, each of which contains at least one million users. Each user group receives recommendations from one of the four models. Our advertising system uses first price ranking approach, which means the candidate ads are ranked by bid*pCTR and displayed with the descending order. The bid is offered by the advertisers and the pCTR is generated by our CTR prediction model. The effective cost per mille (eCPM) is used as the evaluation metric:

$$eCPM = \frac{TotalAdsIncome}{TotalAdsImpressions} \times 1000. \tag{1}$$

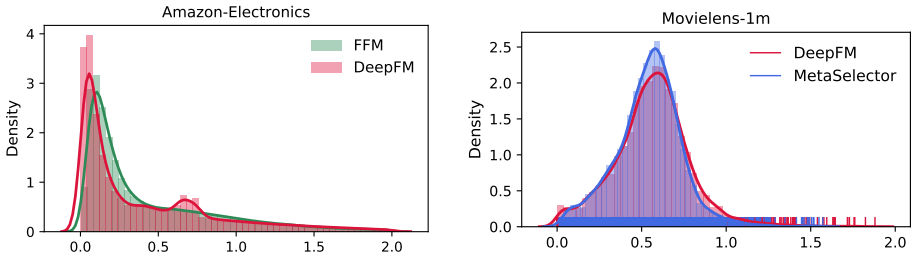
Observations in Online Experiments. We present the trends of eCPM values for four models within 24 hours in Figure 2. Because of the commercial confidential, the absolute values of eCPM are hidden. We see that during the online A/B test, there is no single model which can achieve all-time best performance. For example, in general, Model I and Model III perform poorly during the day. However, Model I and Model III achieve leading performances from 7 a.m. to 8 a.m. and from 5 p.m. to 6 p.m. respectively. We also notice that although Model IV performs best on average, its eCPM is lower than that of some other models in particular time periods. In other words, the first place model is refreshed over time.

3.2 Model Performance on Public Datasets

We conducted some pilot experiments on MovieLens [16] and Amazon Review [18] datasets to quantify the varying performance of models over different users. We consider four models (LR,

Table 1: User proportion of different models.

Dataset	LR	FM	FFM	DeepFM
Movielens-1m	21.37%	18.49%	20.11%	40.03%
Amazon-Electronics	13.73%	13.61%	20.08%	52.58%



(a) KDE for Amazon .

(b) KDE for Movielens.

Figure 3: Kernel density estimation of user loss.

FM [31], FFM [20] and DeepFM [15]).We select the best model for each user by comparing the LogLoss, that is, the model with the lowest LogLoss is considered to be the best.

As shown in Table 1, in general, DeepFM performs better than other models: It is the best model for nearly 40% users in MovieLens, and the best for more than 52% users in Amazon. Although FM is the least popular model for both datasets, there are still 18.49% users in MoviesLens and 13.61% users in Amazon choosing FM.

Interestingly, we find that FFM performs slightly better than DeepFM on Amazon in terms of AUC and LogLoss, but more than half of users select DeepFM rather than FFM. To analyze this phenomenon, we visualize the Kernel Density Estimation (KDE) of the testing loss for each user. As shown in Figure 3a, on the area with low loss(0-0.5), the loss distribution of DeepFM is denser and the mean is lower compared with FFM, which means that DeepFM performs better than FFM in this case. However, there exists an unusual curve protrusion for DeepFM where the loss is around 0.7. This unusual distribution indicates that DeepFM performs poorly under this condition and some users are badly modeled. This also explains the reason why the overall performance of DeepFM is worse than that of FFM.

4 METHODOLOGY

In this section, we elaborate technical details for our proposed model selection framework MetaSelector. Suppose there is a set U of users, where each user $u \in U$ has a dataset D^u available for model training. A data point $(x, y) \in D^u$ consists of feature x and label y. Note that our proposed framework provides a general training protocol for recommendation models, and is independent of specific model structure and data format, as long as they are consistent across users.

4.1 The MetaSelector Framework

The framework MetaSelector consists of two major modules: the base models module and the model selection module. Next we describe the details of the workflow.

Base models module. A base model $\mathcal{M}$ refers to a parameterized recommendation model, such as LR or DeepFM. A model $\mathcal{M}$ with parameter θ is denoted by $\mathcal{M}(\cdot;\theta)$, such that given feature x, the model outputs $\mathcal{M}(x;\theta)$ as the prediction for the ground truth label y. Suppose in the base models module there are K models $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_K$, where $\mathcal{M}_k$ is parameterized by θ_k. Note that the $\mathcal{M}_k$'s could have different structures, and hence contain distinct parameters θ_k's. In general the module allows different input features for different base models, while in what follows we assume all models have the same input form for ease of exposition.

Model selection module. This module contains a model selector $\mathcal{S}$ that operates on top of the base models module. The model selector $\mathcal{S}$ takes as input the data feature x and outputs of base

Algorithm 1: MetaSelector

Data: Training set D^u for user $u \in U$

1 Initialize θ_k for $\mathcal{M}_k$ with $k \in \{1, \ldots, K\}$, and φ for $\mathcal{S}$;
2 Denote $\theta = (\theta_1, \ldots, \theta_K)$ and $\lambda = (\lambda_1, \ldots, \lambda_K)$;
3 (Optional) Pretrain θ using $\bigcup_{u \in U} D^u$;
4 **foreach** episode $t = 1, 2, ...$ **do**
5 Sample a set U_t of m users from U ;
6 **foreach** user $u \in U_t$ **do**
7 Sample D_S^u and D_Q^u from D^u;
8 **foreach** $(x, y) \in D_S^u$ **do**
9 $\lambda \leftarrow \mathcal{S}(x; \varphi)$;
10 $p(x; \theta, \varphi) \leftarrow \sum_{k=1}^{K} \lambda_k \mathcal{M}_k(x; \theta_k)$;
11 **end**
12 $\mathcal{L}_{D_S^u}(\theta, \varphi) \leftarrow \frac{1}{|D_S^u|} \sum_{(x,y) \in D_S^u} \ell(p(x; \theta, \varphi), y)$;
13 $(\theta^u, \varphi^u) \leftarrow (\theta, \varphi) - \alpha \nabla_{\theta, \varphi} \mathcal{L}_{D_S^u}(\theta, \varphi)$;
14 **foreach** $(x, y) \in D_Q^u$ **do**
15 $\lambda \leftarrow \mathcal{S}(x; \varphi^u)$;
16 $p(x; \theta^u, \varphi^u) \leftarrow \sum_{k=1}^{K} \lambda_k \mathcal{M}_k(x; \theta_k^u)$;
17 **end**
18 $\mathcal{L}_{D_Q^u}(\theta^u, \varphi^u) \leftarrow \frac{1}{|D_Q^u|} \sum_{(x,y) \in D_Q^u} \ell(p(x; \theta^u, \varphi^u), y)$;
19 **end**
20 $(\theta, \varphi) \leftarrow (\theta, \varphi) - \beta \cdot \frac{1}{m} \sum_{u \in U_t} \nabla_{\theta, \varphi} \mathcal{L}_{D_Q^u}(\theta^u, \varphi^u)$;
21 **end**

models $\mathcal{M}(x; \theta) := (\mathcal{M}_1(x; \theta_1), \mathcal{M}_2(x; \theta_2), \ldots, \mathcal{M}_K(x; \theta_K))$ where $\theta := (\theta_1, \theta_2, \ldots, \theta_K)$, and outputs a distribution on base models. Suppose $\mathcal{S}$ is parameterized by φ, the selection result is thus $\mathcal{S}(x, \mathcal{M}(x; \theta); \varphi)$. In practice, $\mathcal{S}$ can be a multilayer perceptron (MLP) that takes x only as input (without $\mathcal{M}(x; \theta)$) and generates a distribution $\lambda = \mathcal{S}(x; \varphi)$ over the base models, and the final prediction is the corresponding weighted average $\langle \lambda, \mathcal{M}(x; \theta) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes inner product.

4.2 Meta-training MetaSelector

The key ingredient that differentiates MetaSelector with previous model selection approaches is that we use meta-learning to learn the model selector $\mathcal{S}$, as shown in Algorithm 1. Our algorithm extends MAML into the MetaSelector framework. The original MAML is applied to a single prediction model (such as CNN for image classification and MLP for regression), while in our case MAML is used to jointly learn the model selector and base models.

Episodic Meta-training. The meta-training process proceeds in an episodic manner. In each episode, a batch of users are sampled as tasks from a large training population (line 5). For each user u, a *support* set D_S^u and a *query* set D_Q^u are sampled from D^u, which are considered as "training" and "test" sets in the task corresponding to user u, respectively (line 7). We adopt the common practice in meta-learning literature that guarantees no intersection between D_S^u and D_Q^u to improve generalization capacity. After an in-task adaptation procedure is performed for each task (lines 8–18), at the end of an episode, the *initialization* φ for the model selector and θ for base models are updated according to the loss signal received from in-task adaptation (line 20). Here for both the base models and model selector, the initialization is maintained as they will be adapted to new user when deployed. Next we describe the in-task adaptation procedure.

In-task Adaptation. Given the currently maintained parameters θ and φ, the MetaSelector first iterates the support set D_S^u to generate a per-item distribution λ on base models (line 9), and then get a final prediction $p(x; \theta, \varphi)$ which is a convex combination of outputs $\mathcal{M}(x; \theta)$ (line 10). The training loss $\mathcal{L}_{D_S^u}(\theta^u, \varphi)$ is computed by averaging $\ell(p(x; \theta, \varphi), y)$ over data points in D_S^u (line 12), where ℓ is a pre-defined loss function. In this work we focus on CTR prediction problems and use

LogLoss as the loss function:

$$\ell(p(x;\theta,\varphi),y) = -y\log p(x;\theta,\varphi) - (1-y)\log(1-p(x;\theta,\varphi)), \tag{2}$$

where $y \in \{0,1\}$ indicates if the data point is a positive sample. Then a gradient update step is performed to parameters of the base models and model selector, leading to a new set of parameters θ^u and φ^u adapted to the specific task (line 13). The test loss $\mathcal{L}_{D_Q^u}(\theta^u,\varphi^u)$ is then computed on the query set in a similar way as computing training loss, using the updated parameters of base models and model selector instead (lines 14–18). Note that by keeping the path of in-task adaptation (from (θ,φ) to (θ^u,φ^u)), the test loss $\mathcal{L}_{D_Q^u}(\theta^u,\varphi^u)$ can be expressed as a function of θ and φ, which is passed to the outer loop for updating θ and φ using gradient descent methods such as SGD or Adam.

Jointly Meta-training θ and φ. We further note that θ and φ are update together in the outer loop (line 20) that serve as initialization for the base models and model selector, respectively. The parameters are updated to adapt to each user as shown in line 13 of Algorithm 1. This step is crucial for MetaSelector to operate at the user level, i.e., to execute user-level model selection via base models and model selector modules adaptive to specific users. The episodic meta-learning procedure plays an important role to obtain learnable initialization for MetaSelector to enable fast adaptation on users. The objective of meta-training can be formulated as follows:

$$\min_{\theta,\varphi} \mathbb{E}_{u\in U}\left[\mathcal{L}_{D_Q^u}\left((\theta,\varphi) - \alpha\nabla_{\theta,\varphi}\mathcal{L}_{D_S^u}(\theta,\varphi)\right)\right]. \tag{3}$$

Learning Inner Learning Rate α. The inner learning rate α, which is often a hyper-parameter in normal model training protocols, can also be learned in meta-learning approaches by considering the test loss $\mathcal{L}_{D_Q^u}(\theta^u,\varphi^u)$ as a function of α as well. Li et al. [24] showed that learning per-parameter inner learning rate α (a vector of same length as θ) achieves consistent improvement over MAML for regression and image classification. Algorithm 1 can be slightly modified accordingly: in line 13, the inner update step becomes:

$$(\theta^u,\varphi^u) \leftarrow (\theta,\varphi) - \alpha \circ \nabla_{\theta,\varphi}\mathcal{L}_{D_S^u}(\theta,\varphi), \tag{4}$$

where $\circ$ denotes Hadamard product. Considering θ^u, φ^u as a function of α, the outer update step in line 20 becomes:

$$(\theta,\varphi,\alpha) \leftarrow (\theta,\varphi,\alpha) - \beta\cdot\tfrac{1}{m}\textstyle\sum_{u\in U_t}\nabla_{\theta,\varphi,\alpha}\mathcal{L}_{D_Q^u}(\theta^u,\varphi^u), \tag{5}$$

where gradients flow to α through θ^u and φ^u. The objective function can be accordingly written as:

$$\min_{\theta,\varphi,\alpha} \mathbb{E}_{u\in U}\left[\mathcal{L}_{D_Q^u}\left((\theta,\varphi) - \alpha\circ\nabla_{\theta,\varphi}\mathcal{L}_{D_S^u}(\theta,\varphi)\right)\right]. \tag{6}$$

In practice we find that learning a vector α could significantly boost the performance of MetaSelector for recommendation tasks.

Meta-testing/Deployment. Meta-testing MetaSelector on new tasks follows the same in-task adaptation procedure as in meta-training (lines 7–17), after which evaluation metrics are computed such as AUC and LogLoss. A separate group of meta-testing users (with no intersection with meta-training users) may be considered to justify the generalization capacity of meta-learning on new tasks.

Simplifying MetaSelector. We propose a simplified version of meta-training for MetaSelector, where no in-task adaptation for base models is required. The base models are pre-trained before meta-training and then fixed. The model selector is trained episodically. We note that this procedure is also in the meta-learning paradigm since φ is updated using user-wise mini-batches, where for each user u the distribution λ is generated using a support set D_S^u, and evaluated by computing test loss on a separate query set D_Q^u. This enables MetaSelector to learn at user level and generalize to new users efficiently. At meta-testing phase, base models as well as the model selector are fixed, and the training set is simply used for the model selector to generate a distribution over base models. The simplified MetaSelector may be of particular interest in practical recommender systems where in-task adaptation is restricted due to computation and time costs, such as news recommendation for mobile users using on-device models.

5 EXPERIMENT

In this section, we evaluate the empirical performance of the proposed method, and mainly focus on CTR Prediction tasks where the prediction quality plays a very important role and has a direct

impact on the business revenue. We experiment with two public datasets and a real-world production dataset. The statistics of the selected datasets are summarized in Table 2. We raise and try to address two major research questions: **(Q1)**: Can model selection help CTR Prediction? **(Q2)**: What benefits could MetaSelector bring to personalized model selection?

5.1 Datasets

Movielens-1m. Movielens-1m [16] contains 1 million movie ratings from 6040 users and each user has at least 20 ratings. We regard both 5-star and 4-star ratings as positive feedbacks and label them with 1, and treat the rest ratings as negative feedback and label them with 0. We select the following features: user_id, age, gender, occupation, user_history_genre, user_history_movie, movie_id, movie_genre, day of week and season.

Amazon-Electronics. Amazon Review Dataset [18] contains user reviews and metadata from Amazon and has been widely used for product recommendation. We select a subset called Amazon-Electronics from the collection and shape it into a binary classification problem like Movielens-1m. Following [17], we use the 5-core setting to retain users with at least 5 ratings. The selected features include user_id, item_id,item_category, season, user_history_item (including 5 products recently rated), user_history_categories.

Production Dataset. To demonstrate the effectiveness of our proposed methods on real-world application with natural data distribution over users, we also evaluate our methods on a large production dataset from an industrial recommendation task. Our goal is to predict the probability that a user will click on the recommended mobile services based on his or her history behavior. In this dataset, each user has at least 203 history records.

Table 2: Statistics of selected datasets.

Dataset	Users	Items	Samples	Features
Movielens-1m	6,040	3,952	1,000,209	14,025
Amazon-Electronics	192,403	63,001	1,689,188	319,687
Production Dataset	7,684	2,420	3,333,246	11,860

5.2 Baselines

We compare the proposed methods with two different kinds of competitors: single recommendation models including LR, FM, FFM and DeepFM, and hybrid recommender with sample-level selector.

Single Models. We consider three types of model architectures, including linear (LR), low rank (FM [31] and FFM [20]) and deep models (DeepFM [15]). The latent dimension of FM and FFM is set to 10. The field numbers of FFM for Movielens, Amazon and Production are 22, 18 and 8 respectively. For DeepFM, the dropout setting is 0.9. The network structures for Movielens, Amazon and production datasets are 256-256-256, 400-400-400 and 400-400-400 respectively. We use ReLU as the activation function.

Sample-level Selector. This method is used as a model selection competitor and is designed to predict the model probability distribution for each sample. For each user, 80% local data is used for training and the rest for testing. Then the local data of all users is collected to generate the whole training and testing data. While training, 75% training data is firstly used to train four CTR prediction models in a mini-batch way [23]. The batch size is set to 1000. Then the pretrained recommenders predict the CTR values and Logloss for the remaining training data. We give these samples labels from 0-3 by comparing the LogLoss of each recommender. Afterward, we add the CTR predictions of the four recommenders as meta-features to train a 400-400-400 MLP classifier. While testing, the final prediction for each testing sample is the weighted average of the predicted values of the individual models.

5.3 Settings and Evaluation Metrics

For MetaSelector, the division of user local data is the same as the division for sample-level MLP selector. During meta-training process, the training data of each user is further divided into 75% support set and 25% query set. During meta-testing phase, the model selector and base models

are firstly fine-tuned before evaluating on the testing data. The performance metrics used in our experiments are AUC (Area under ROC), LogLoss and RelaImpr. RelaImpr is calculated as follows:

$$RelaImpr = \left(\frac{AUC(to\ be\ compared) - 0.5}{AUC(single\ best\ model) - 0.5} - 1 \right) \times 100\%. \quad (7)$$

For pre-training of CTR models, we use FTRL optimizer[25] for LR and Adam optimizer [22]for FM, FFM and DeepFM. The mini-batch size is 1000. For MetaSelector, we use Meta-SGD [24] to adaptively learn the inner learning rate α. The initial value of inner learning rate α for Movielens, Amazon and Production dataset is 0.001, 0.0001, 0.001. The outer learning rate β is set to $1/10$ of α. In each episode of meta-training, the numbers of active users are 10. We use a 200-200-200 MLP as the model selector.

5.4 Performance of Model Selection

Q1: Overall Performance Comparison. To investigate RQ1, we study the performance of baselines and MetaSelector on three datasets, the results are summarized in Table 3. To explore the potential and limit of model selection approaches, we compute the upper bound through two perfect model selectors: (1) perfect sample-level selector which chooses the best model for each sample; (2) perfect user-level selector which chooses the best model for each user. First, comparing single model baselines with hybrid recommender with model selection, we see that all model selection methods achieve a considerable improvement in terms of AUC and Logloss. This result is highly encouraging, indicating the effectiveness of model selection methods. Second, comparing the sample-level selectors with the user-level selectors, we find that perfect sample-level model selector is expected to achieve greater improvements than perfect user-level selector. The expected RelaImpr can reach up to 61.35% for Amazon. However, the reality is not as good as expected. In the last three rows of Table 3, we show the performance of actual selectors and observe that user-level selectors including MetaSelector and MetaSelector-Simplified achieve higher AUC and lower Logloss, rather than the sample-level selector. This discovery implies that the differences between samples may be too subtle for the selector to be well fitted. In contrast, the latent characteristics of different users vary widely, which makes the MetaSelector work well. Finally, we compare MetaSelector and MetaSelector-simplified, finding that the performance of the simplified version dropped slightly. This verifies our argument in section 4 that the in-task adaptation could make model selection more user-specific.

Table 3: AUC and LogLoss Results.

Model	Movielens.			Amazon.			Production.		
	AUC	LogLoss	RelaImpr	AUC	LogLoss	RelaImpr	AUC	LogLoss	RelaImpr
LR	0.7914	0.55112	-1.45%	0.6981	0.46374	-6.29%	0.7813	0.54011	-29.83%
FM	0.7928	0.54917	-0.98%	0.6953	0.46242	-7.62%	0.8821	0.42618	-4.69%
FFM	0.7936	0.54826	-0.71%	0.7114	0.45216	0.00%	0.8850	0.42469	-3.97%
DeepFM	0.7957	0.54672	0.00%	0.7101	0.45696	-0.61%	0.9009	0.39215	0.00%
Perfect Sample-level Selector	0.9008	0.41079	35.54%	0.8411	0.37088	61.35%	0.9710	0.26043	17.49%
Perfect User-level Selector	0.8187	0.51829	7.78%	0.8135	0.38999	48.30%	0.9051	0.37835	1.05%
Sample-level Selector	0.7963	0.54482	0.20%	0.7121	0.45152	0.33%	0.9011	0.39137	0.05%
MetaSelector	**0.8047**	**0.53531**	**3.04%**	**0.7141**	**0.44996**	**1.28%**	**0.9023**	**0.39036**	**0.35%**
MetaSelector-Simplified	0.8036	0.53550	2.67%	0.7134	0.45044	0.95%	0.9022	0.39095	0.32%

Q2: Performance Distribution Analysis. Despite the overall improvement, it is also worth studying RQ2: In what ways does MetaSelector help model selection? To this end, we further investigate the testing loss distribution on all users with MovieLens-1m dataset. Figure 3b shows the kernel density estimation of MetaSelector and DeepFM which is a strong single model baseline. We observe that MetaSelector not only leads to lower mean LogLoss but also achieves more concentrated loss distribution with lower variance. This shows that MetaSelector encourages a more fair loss distribution across users and is powerful to model heterogeneous users. The above observations verify the effectiveness of our proposed methods in terms of personalized model selection.

6 CONCLUSION

In this work, we addressed the problem of model selection for recommender systems, motivated by the observation of varying performance of different models among users on public and private

production datasets. We initiated the study of user-level model selection problems in recommendation from the meta-learning perspective, and proposed a new framework MetaSelector that introduces meta-learning methods to formulate a user-level model selection module. We also ran extensive experiments on both public and private production datasets, showing that MetaSelector can improve the performance over single model baseline and sample-level selector. This shows the potential of MetaSelector in real-world recommender systems.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, 2016.

[2] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-adapted Interaction*, 12(4):331–370, 2002.

[3] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4): 61, 2015.

[4] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. *arXiv preprint arXiv:1802.07876*, 2018.

[5] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

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

[7] Andrew Collins, Joran Beel, and Dominika Tkaczyk. One-at-a-time: A meta-learning recommender-system for recommendation-algorithm selection on micro level. *arXiv: Information Retrieval*, 2018.

[8] Tiago Cunha, Carlos Soares, and Andre C P L F De Carvalho. Selecting collaborative filtering algorithms using metalearning. pages 393–409, 2016.

[9] Tiago Cunha, Carlos Soares, and Acplf De Carvalho. Metalearning and recommender systems: A literature review and empirical study on the algorithm selection problem for collaborative filtering. *Information Sciences*, 423:128–144, 2018.

[10] Simon Dooms. Dynamic generation of personalized hybrid recommender systems. pages 443–446, 2013.

[11] Michael D Ekstrand and John Riedl. When recommenders fail: predicting recommender failure for algorithm selection and combination. pages 233–236, 2012.

[12] Michael D Ekstrand, F Maxwell Harper, Martijn C Willemsen, and Joseph A Konstan. User perception of differences in recommender algorithms. pages 161–168, 2014.

[13] Michael D Ekstrand, Daniel Kluver, F Maxwell Harper, and Joseph A Konstan. Letting users choose recommender algorithms: An experimental study. pages 11–18, 2015.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[16] F. Maxwell Harper and Joseph A. Konstan. *The MovieLens Datasets: History and Context*. 2015.

[17] Ruining He and Julian Mcauley. Vbpr: Visual bayesian personalized ranking from implicit feedback. *arXiv: Information Retrieval*, 2015.

[18] Ruining He and Julian Mcauley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *the web conference*, pages 507–517, 2016.

[19] Xiangnan He and Tatseng Chua. Neural factorization machines for sparse predictive analytics. pages 355–364, 2017.

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

[21] Yuchin Juan, Damien Lefortier, and Olivier Chapelle. Field-aware factorization machines in a real-world online advertising system. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 680–688. International World Wide Web Conferences Steering Committee, 2017.

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

[23] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. pages 661–670, 2014.

[24] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[25] H Brendan Mcmahan and Matthew J Streeter. Adaptive bound optimization for online convex optimization. *arXiv: Learning*, 2010.

[26] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.

[27] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2554–2563. JMLR. org, 2017.

[28] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

[29] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. pages 1149–1154, 2016.

[30] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

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

[32] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. *conference on computer supported cooperative work*, pages 175–186, 1994.

[33] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1987.

[34] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and Jianchang Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. pages 255–262, 2016.

[35] Joseph Sill, Gabor Takacs, Lester Mackey, and David Lin. Feature-weighted linear stacking. *arXiv: Learning*, 2009.

[36] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv: Learning*, 2017.

[37] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[38] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. pages 3637–3645, 2016.

[39] Hao Wang, Naiyan Wang, and Dityan Yeung. Collaborative deep learning for recommender systems. pages 1235–1244, 2015.

[40] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tatseng Chua. Neural graph collaborative filtering. *international acm sigir conference on research and development in information retrieval*, pages 165–174, 2019.

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