Federated Meta-Learning for Recommendation

Fei Chen **Zhenhua Dong** **Zhenguo Li** **Xiuqiang He**
Huawei Noah's Ark Lab
{chenfei100,dongzhenhua,li.zhenguo,hexiuqiang}@huawei.com

Abstract

Recommender systems have been widely studied from the machine learning perspective, where it is crucial to share information among users while preserving user privacy. In this work, we present a federated meta-learning framework for recommendation in which user information is shared at the level of *algorithm*, instead of *model* or *data* adopted in previous approaches. In this framework, user-specific recommendation models are locally trained by a shared *parameterized* algorithm, which preserves user privacy and at the same time utilizes information from other users to help model training. Interestingly, the model thus trained exhibits a high capacity at a small scale, which is energy- and communication-efficient. Experimental results show that recommendation models trained by meta-learning algorithms in the proposed framework outperform the state-of-the-art in accuracy and scale. For example, on a production dataset, a shared model under Google Federated Learning (McMahan et al., 2017) with 900,000 parameters has prediction accuracy 76.72%, while a shared algorithm under federated meta-learning with less than 30,000 parameters achieves accuracy of 86.23%.

1. Introduction

Recommendation has become an important component of functionality on current terminal user devices. Building personalized recommender systems with high performance is an interesting topic in the machine learning research area. Stand-alone recommendation models serve as a simple and popular choice since they are built locally on user devices and no interaction among users needs to be considered. It is natural to assert that a personalized model that is separately trained with local data should work well for each user. However, information derived from a large group of users, such as common behavior patterns among smart phone users, could improve the performance of recommendation models.

Collaborative filtering methods such as matrix factorization (Koren, 2008) provide an approach to building recommendation models with user data all together. The main drawback of these methods is that they aggregate user information at the *data* level, and the models need to be trained on centralized servers. User privacy has become a crucial concern in recent days, and consequent restrictions on collection and usage of user data have become a crucial issue in practice. Federated learning (McMahan et al., 2017) provides an alternative approach which makes use of user information at the *model* level while preserving user privacy. A unified model is trained that could be large in size since it necessarily incorporates input from all users and provides personalized recommendation for all users. This restricts implementation of federated learning techniques in practice as user devices often have limited network bandwidth and computation resource to operate recommendation models.

In this paper, we propose a federated meta-learning framework for recommendation that shares user information at the *algorithm* level. In this framework there is a parameterized algorithm that trains parameterized recommendation models. In other words, both the algorithm and models are parameterized and hence need to be optimized. The algorithm is trained with *tasks* via a two-level *meta-training* procedure. In an iteration of meta-training, a batch of tasks is sampled. At the *model level* operated on user devices, in each task a model is first trained on a *support set* by the current algorithm, and then evaluated on a separate *query set* to provide test feedback, e.g., testing loss gradient. The test feedback is used to improve the algorithm's ability to train models. At the *algorithm level* operated on server, the algorithm is updated with test feedback from the tasks. After meta-training, for each demanding user, a recommendation model is again first trained by the algorithm using local data before making predictions for unknown queries.

The major advantage of the federated meta-learning framework is that at the same time it enables information sharing (as in collaborative filtering and federated learning, but at a higher algorithm level) and local model training without privacy issue and significant expansion in model size (as in stand-alone approaches).

Experiments show that recommendation models trained in the meta-learning approach achieve highest prediction accuracies compared to baselines. Moreover, the models can be adapted fast to new users in terms of update steps (which

is gradient steps in our experiments). In particular, we compare a unified neural network model NN-unified (with pre-training in the federated learning approach), a separately trained small neural network NN-self (the stand-alone approach), and the same small network NN-meta trained in the meta-learning approach, as shown in Table 1. See Section 3.2 for detailed discussion.

Table 1. Experimental results on a production dataset.

Model	# params	# steps	Accuracy
NN-unified	918452	100	76.72%
NN-self	9256	100	57.20%
		10000	83.79%
NN-meta	model: 9256 algorithm: 18512	100	86.23%

Our contribution can be summarized as follows.

- We propose a federated meta-learning framework to share user information at the model-training algorithm level while preserving user privacy.
- In this framework the recommendation model is user-specific and can be kept at a small scale. It is practical to train and deploy the model on user devices with moderate resource consumption.
- Experiments show that the proposed framework achieves higher prediction accuracies than previous approaches, even with small recommendation model and fast adaptation to new users.

1.1. Related Work

Deep learning for recommender systems has been studied in the literature, from the perspective of pure deep neural networks (Cheng et al., 2016; Wang et al., 2017) or combination with other methods (He et al., 2017; Guo et al., 2017). Mobile APPs prediction can employ contextual information from corresponding users (Yan et al., 2012; Huang et al., 2012). In this work we propose a federated meta-learning framework, aiming to facilitate practical user information sharing while preserving privacy, and discuss its application in recommendation.

Meta-learning has recently demonstrated effectiveness in few-shot learning for regression (Finn et al., 2017; Li et al., 2017), image classification (Vinyals et al., 2016; Ravi & Larochelle, 2017; Finn et al., 2017; Li et al., 2017; Snell et al., 2017; Zhou et al., 2018) and reinforcement learning (Duan et al., 2016; Wang et al., 2016; Finn et al., 2017; Li et al., 2017). Among the various meta-learning algorithms some are focused on the optimization of models and work across different task domains, such as Meta-LSTM (Ravi & Larochelle, 2017), MAML (Finn et al., 2017) and Meta-SGD (Li et al., 2017), which can be readily used to train recommendation models in our framework. Another type of meta-learning algorithms has a closer connection with domain knowledge, such as Matching Networks (Vinyals et al., 2016) and Prototypical Networks (Snell et al., 2017) for image classification, and RNNs (Duan et al., 2016; Wang et al., 2016) for reinforcement learning.

Vartak et al. (Vartak et al., 2017) studied cold-start recommendation for items from a meta-learning perspective. We note that our work is different from (Vartak et al., 2017) in two major aspects. Firstly, Vartak et al. considered a specific binary classification problem for item recommendation, and proposed two neural architectures for the problem. In this work, we address the user privacy issue in current recommender systems, and propose a general federated meta-learning framework that can cope with a wide range of meta-learning algorithms as well as recommendation models. Secondly, the architectures in (Vartak et al., 2017) share user information at the model level with part of the network encoding meta-information. In our framework, it is natural to apply two-level meta-learning methods that capture meta information at the algorithm level and user-specific information at the model level.

2. Federated Meta-Learning

2.1. The Problem

As in (Vartak et al., 2017), we consider making recommendations for a user as one task. We define the recommendation model in a general form. A data point (X, Y) consists of a feature vector $X \in \mathcal{X}$ and a *value* $Y \in \mathcal{Y}$ (or *label* for discrete $\mathcal{Y}$). For a user u, a *support set* D_S^u contains history data points that can be used to train a recommendation model θ. With slight abuse of notation, we also denote by θ the model's parameter. Given an input from the feature space $\mathcal{X}$, the goal is to use θ to output a value (or a collection of values) from the value space $\mathcal{Y}$.

2.2. Meta-Learning for Recommendation

In meta-learning for recommendation tasks, given support set D_S^u from a user u, an algorithm $\mathcal{A}$ parameterized by φ (which appears as *meta-learner* in recent meta-learning literature) generates a recommendation model θ_u for user u, where $\theta_u = \mathcal{A}_{D_S^u}(\varphi)$. In particular, the generation process involves optimizing the user-specific parameter θ_u.

In the *meta-training* procedure, algorithm $\mathcal{A}$ is trained with a number of *training tasks*. For user u we sample a support set D_S^u and a *query set* D_Q^u from the user's history data. Note that D_S^u and D_Q^u are separate and both contain labeled data. We use D_S^u to generate a model θ_u for u, and then make predictions on D_Q^u using θ_u and compute the test loss $\mathcal{L}_{D_Q^u}(\theta_u)$. The algorithm's parameter φ is optimized by using the test loss collected from sampled tasks, and the

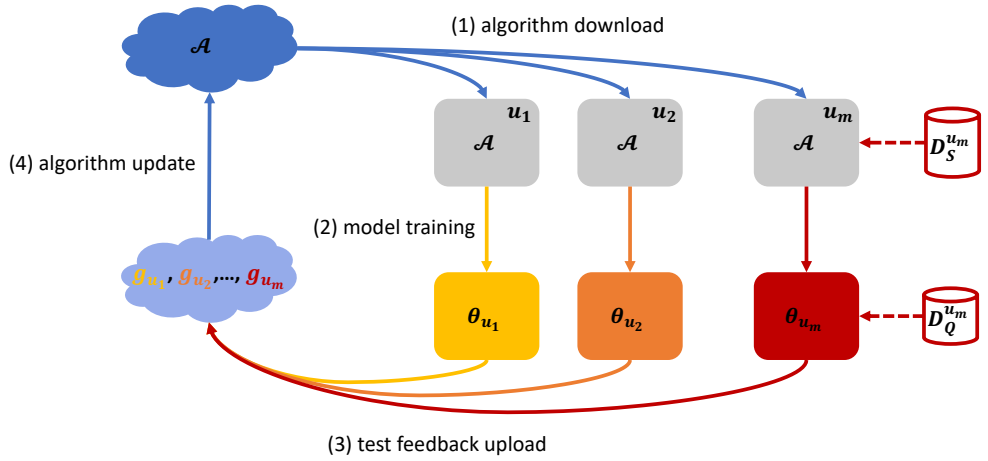

Figure 1. Architecture of federated meta-learning.

objective function is defined as follows:

$$\min_{\varphi} \mathbb{E}_u[\mathcal{L}_{D_Q^u}(\theta_u)] = \min_{\varphi} \mathbb{E}_u[\mathcal{L}_{D_Q^u}(\mathcal{A}_{D_S^u}(\varphi))]. \quad (1)$$

When applying algorithm $\mathcal{A}$ to a new user u, we first use a support set D_S^u (sampled from history data) to generate a model $\theta_u = \mathcal{A}_{D_S^u}(\varphi)$, and then use θ_u to make predictions on new queries. To evaluate the performance of algorithm $\mathcal{A}$, we carry out a *meta-testing* procedure where we apply $\mathcal{A}$ to a number of *testing tasks*. For each testing task u, the model θ_u generated by $\mathcal{A}$ is evaluated on a query set D_Q^u. The test loss function for meta-training and evaluation method for meta-testing depend on specific types of recommendation tasks.

2.3. The Federated Meta-Learning Framework

We incorporate meta-learning into the decentralized training process as in federated learning. In this framework, meta-training proceeds naturally in a distributed manner, where each user has a specific model that is trained using local data. The model level training is performed on user devices, and the server only maintains and updates $\mathcal{A}$ at the algorithm level. As shown in Algorithm 1, $\mathcal{A}$ can be trained by using SGD. The server first sends $\mathcal{A}$ with parameter φ to a set of sampled users (training tasks). Then each sampled user trains a user-specific model using current $\mathcal{A}$ and evaluates the model on the corresponding device. Finally the server collects from these users the test loss gradients to update the parameter φ of algorithm $\mathcal{A}$. It is not necessary to upload any user data to the server in the process, instead only the algorithm $\mathcal{A}$ and test loss gradients are transfered in each episode. The framework is shown in Figure 1.

2.4. Comparison with Previous Work

We note that the federated meta-learning framework is different from federated learning and collaborative filtering in

Algorithm 1 Federated Meta-Learning

AlgorithmUpdate: // Run on the server
  Initialize φ for algorithm $\mathcal{A}$
  For *each episode* $t = 1, 2, ...$
    Sample m users U_t
    For *each user* $u \in U_t$ *in parallel*
      $g_u \leftarrow \text{ModelTraining}(\mathcal{A}, \varphi)$
    $\varphi \leftarrow \varphi - \beta \sum_{u \in C_t} \frac{g_u}{m}$

ModelTraining$(\mathcal{A}, \varphi)$**:** // Run on user u
  Sample support set D_S^u and query set D_Q^u
  $\theta_u \leftarrow \mathcal{A}_{D_S^u}(\varphi)$ // Generate recommendation model
  $g_u \leftarrow \nabla_\varphi \mathcal{L}_{D_Q^u}(\theta_u)$ // Compute loss gradient
  Return g_u to server

Table 2. Comparison among collaborative filtering, federated learning and federated meta-learning.

Approaches	Sharing	Privacy	Small
collaborative filtering	data	×	×
federated learning	model	√	×
federated meta-learning	algorithm	√	√

significant ways, as summarized in Table 2.

Federated learning. Conceptually, federated learning provides an approach to sharing user information at the model level. Although trained in a distributed manner, the unified model needs to take all user input into consideration and to provide personalized recommendation for all users, which suffers from the large size necessary in many practical circumstances. Federated meta-learning, on the other hand, provides an approach to sharing user information at the higher algorithm level, making it possible to train small user-specific models. Technically, in federated learning the transmission between the server and user devices involves current models, while in federated meta-learning

Table 3. Performance on MovieLens 100k

		RMSE	Precision@1	Precision@3	Precision@10
80% Support	SVD++ (SELF)	1.0124	–		
	SVD++ (MIXED)	1.0919			
	NN (SELF)	1.1968	44.94%	40.26%	33.26%
	NN (MIXED)	0.9507	70.25%	58.44%	43.72%
	MAML	**0.9295**	**71.52%**	**59.31%**	**43.95%**
	Improvement	2.23%	1.81%	1.49%	0.53%
50% Support	SVD++ (SELF)	1.0217	–		
	SVD++ (MIXED)	1.0807			
	NN (SELF)	1.2662	46.89%	41.05%	41.14%
	NN (MIXED)	0.9679	73.45%	65.35%	54.56%
	MAML	**0.9441**	**74.58%**	**66.48%**	**55.77%**
	Improvement	2.46%	1.54%	1.73%	2.22%
20% Support	SVD++ (SELF)	1.0517	–		
	SVD++ (MIXED)	1.0753			
	NN (SELF)	1.3542	35.39%	37.45%	38.32%
	NN (MIXED)	0.9831	73.60%	65.73%	56.30%
	MAML	**0.9593**	**74.16%**	**67.98%**	**57.46%**
	Improvement	2.42%	0.76%	3.42%	2.06%

the transmission involves the algorithm from server and test feedback from (training) users to improve the algorithm's model-training capacity.

Collaborative filtering. Collaborative filtering provides an approach to sharing user information at the data level. Privacy issue arises when the server requires to collect user data to centrally train the model. In federated meta-learning the recommendation model is locally trained for each user, and no user data are transferred to server or to other user devices.

3. Experiments

We study the performance of two-level meta-learning algorithms by performing experiments on a public dataset, MovieLens, and a production dataset. We investigate two major questions:

(1) What benefit could the meta-learning approach bring to training recommendation models?

(2) How efficient could the meta-learning approach adapt to new users, with limited data and update steps?

For the first question, we compare the performance of meta-learning algorithms with baselines for recommendation. We consider two meta-learning algorithms MAML (Finn et al., 2017) and Meta-SGD (Li et al., 2017), both of which are simple optimization oriented algorithms and have demonstrated powerfulness for few-shot image classification tasks. Both MAML and Meta-SGD adopt variants of gradient descent at the algorithm level. In MAML the algorithm's parameter is the initialization (of model parameter) in gradient descent, while in Meta-SGD the algorithm's parameters include the initialization and vectorized learning rate.

For the second question, we vary the fraction p of data that are used as support set during the *meta-testing* procedure, and denote the setting by "p Support" when presenting experimental results. For instance, "80% Support" corresponds to the setting where for each testing user, 80% of the data are used as the support set. We also consider different update steps in the experiment on the production dataset.

3.1. MovieLens

We first evaluate the meta-learning approach on a public recommendation dataset MovieLens 100k (Harper & Konstan, 2015), which contains $100,000$ ratings from 943 users for 1682 movies. We randomly select 80% of the users as training users, and the remaining as testing users. We choose MAML as the meta-learning algorithm and a shallow neural network with one hidden layer as the recommendation model. The hidden layer of the neural network contains 32 neurons. The input is a feature vector encoding movie info including ID and genre, and user info including gender, age, occupation and rating time. After the hidden layer we add a Sigmoid activation, the output of which is scaled between 0 and 5 as the predicted rating score. The network is optimized to minimize the squared error between ground-truth and predicted scores. In this experiment MAML updates the model with 10 gradient steps within each task.

We consider two types of baselines. (1) Collaborative filtering methods: we consider SVD++ (Koren, 2008) implemented in the Python Scikit Surprise (Hug, 2017). We evaluate SVD++ on the query set of testing users, while for

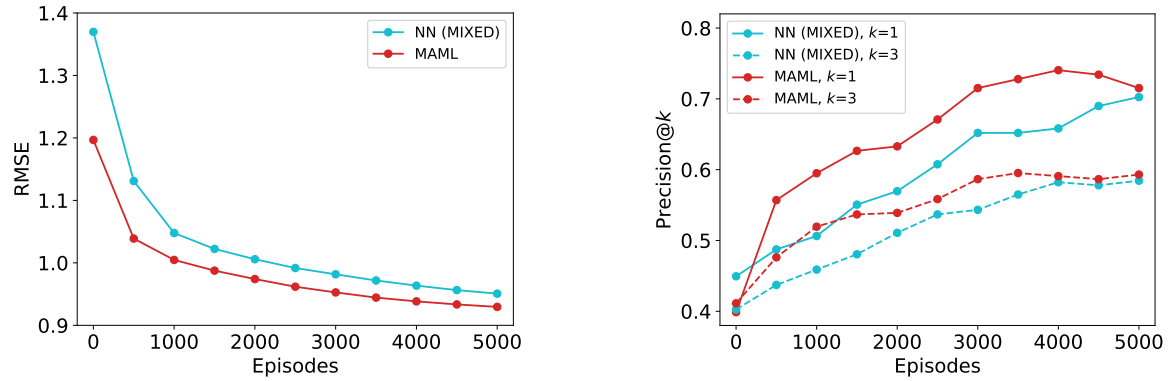

Figure 2. Performance of NN (MIXED) and MAML in the pre-training (for MIXED) and meta-training (for MAML) process, with RMSE on the left figure, and Precision@1 and Precision@3 on the right figure.

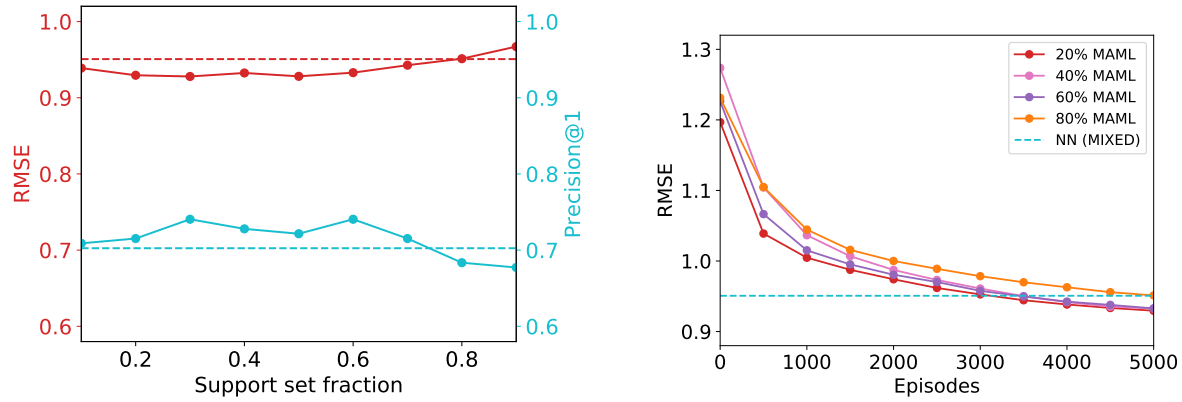

Figure 3. Varying the fraction of data as support set in training tasks ("80% Support" setting for testing users). Solid lines represent the results for MAML, while dashed lines represent the results for NN (MIXED) after 5000 episodes of pre-training. Left: RMSE and Precision@1 after 5000 episodes of meta-training in terms the fraction. Right: RMSE in terms of episodes, where the percentage is the fraction of data as support set in training tasks.

the training set there are two settings. In the SELF setting, the training set is the support set of the corresponding testing user. In the MIXED setting, the training set consists of all the data of *training users* (same as those in the meta-training procedure for meta-learning methods). Note that the performance might be improved if the model for a testing user is trained with data from both the support set of that testing user and training users, but this requires training a different large model for each user and we omit evaluation for this strategy. (2) Neural network methods (NN): we consider the same architecture as in the meta-learning approach, while the model is trained in two (non meta-learning) settings. In the SELF setting, a specific model is separately trained for each testing user using local data. No data from training users are used. In the MIXED setting, a unified model is pre-trained by using history data from all training users (same as those in the meta-training procedure for meta-learning methods) and then fine-tuned to testing users.

We use two evaluation metrics: RMSE and Precision@k, where Precision@k is the average fraction of movies with ground-truth score 5 among the top k movies with highest predicted scores.

Table 3 shows the RMSE and Precision@k (for $k = 1, 3, 10$) results. In the "p Support" case, a p fraction of data are used as support set for each testing user, and the remaining as query set. For NN (MIXED) and MAML, the model is trained for 5000 episodes on training tasks, where each episode contains one task (i.e., batch size is 1). The two-level meta-learning approach MAML achieves lowest RMSE and highest precisions. Comparing NN (SELF), NN (MIXED) and MAML for training the same neural network as the recommendation model, NN (SELF) performs significantly worse than the other two methods, indicating that pre-training and meta-training with other users could improve the performance by a large gap. MAML performs consistently better than NN (MIXED). The performance of SVD++ falls between NN (SELF) and NN (MIXED). In particular, SVD++ (SELF) performs better than SVD++ (MIXED), showing that the user's own data is more effective than the (much more) data from other users for training

collaborative filtering models.

For varying p value in "p Support", the performance of MAML drops in the cold-start situation (20% Support case) under the RMSE metric. However, the Precision@k tends to increase as p decreases, which is observed in the NN (MIXED) method as well. This might imply that the performance of algorithms for recommendation largely relies on optimization strategies (e.g., minimizing squared error) and evaluation metrics.

We also study the performance of NN (MIXED) and MAML as the number of training episodes increases in pre-training for NN (MIXED) and meta-training for MAML, as shown in Figure 2. For both RMSE and Precision@k, MAML performs better and converges faster than NN (MIXED) in the process. We note that RMSE improves in a more stable process than Precision@k. This may be because that the model is optimized to minimize the squared error, and Precision@k involves structural operation (ranking the rating scores) not considered during training.

During meta-training for MAML, in each training task we use all data points of the corresponding user. It remains to determine the fraction of data points used as support set to train the model, and the rest as query set to evaluate the model. (Note that the p value in "p Support" represents the fraction for *testing* users.) We study the effect of this fraction on the performance. As shown in Figure 3, the model achieves a lower RMSE and higher precision when the fraction for support set in training tasks is between 20% and 60%. When this fraction goes up to 80%, the META method performs worse than MIXED. Moreover, when this fraction is 20%, 40% or 60%, the RMSE for META after 3500 episodes is already lower than that for MIXED (at 5000 episodes). This suggests that the meta-training strategy, particularly usage of support and query sets in training tasks, has a considerable impact on the performance.

3.2. Production Data

In order to verify the performance of the two-level meta-learning method in real industrial recommendation task, we conduct experiment on production dataset of mobile service usage records. In each record the value (label) is a service that the user used, and the feature contains service features (e.g., service ID and etc), user features (e.g. last used service and etc) and the context features (e.g. battery level, time, and etc). The goal is to recommend mobile services which the user will open based on given information. The accurate recommendation can help pre-load the right services to the memory for efficient execution or help float the desired services to the right place of the mobile phone (e.g. home screen) for quicker launch. We cast recommendation as a classification problem. The recommendation model is therefore a classifier, the output of which is a probability distribution over all possible services.

Table 4. Statistics of the Production Dataset

# users	9093	7000 (training)
		2093 (testing)
# records	6.4 million	
# records per user	100 – 5243	
# services (classes)	2400	
# services per user	2 – 36	

Table 5. # Parameters of Architectures

methods	# parameters	
LR	4160	
NN	9256	
NN-unified	918452	
MAML + LR	algorithm: 4160	model: 4160
Meta-SGD + LR	algorithm: 8320	model: 4160
MAML + NN	algorithm: 9256	model: 9256
Meta-SGD + NN	algorithm: 18512	model: 9256

The dataset consists of over 6 million service usage records with user consent in 30 consecutive days. There are 2400 distinct services and 9093 subjects, where each subject has 100 to over 5000 records and 2 to 36 services. In the following experiments, we randomly sample 7000 users as training users, which might be used in pre-training (for baselines) or meta-training (for meta-learning methods), and the remaining as testing users, which are used to (train and) evaluate recommendation models. The statistics of the dataset is summarized in Table 4.

We consider two meta-learning algorithms MAML and Meta-SGD. In federated meta-learning the recommendation model is locally trained and used, and hence a classifier for 40 classes would suffice. This is in contrast with the federated learning approach where the unified classifier needs to cope with 2400 classes among all users. We consider two architectures for the classifier: logistic regression and neural network, denoted by LR and NN respectively in the rest of this section. The neural network contains one hidden layer of 64 neurons followed by ReLU activations. We avoid using deep neural networks, since we focus on studying the advantage that meta-learning algorithms could bring to training recommendation models, instead of searching for an optimal model. Moreover, in practice the model would be trained on user devices that have limited computation resources where simple models are preferable.

We compare the meta-learning approach (META) with two types of baselines. (1) The MIXED type represents the federated learning approach, where a unified classifier is first trained on training users and then fine-tuned to each testing user with the corresponding support set. The classifier is a neural network with one hidden layer of 64 neurons, and the output layer consists of 2420 neurons. The network is

Table 6. Accuracies on the Production Dataset

			80% Support		5% Support	
			Top 1	Top 4	Top 1	Top 4
MIXED	NN-unified		76.72%	89.13%	66.47%	79,88%
SELF	MFU		42.92%	81.49%	42.18%	72.87%
	MRU		70.44%	81.43%	70.44%	**81.43%**
	NB		78.18%	92.57%	59.16%	72.83%
	LR	100 steps	58.30%	86.52%	52.53%	75.25%
		10000 steps	78.31%	93.70%	65.35%	77.11%
	NN	100 steps	57.20%	88.37%	49.89%	75.26%
		10000 steps	83.79%	94.56%	68.87%	77.66%
META	MAML + LR		47.69%	71.60%	46.75%	66.26%
	Meta-SGD + LR		81.70%	93.56%	72.32%	77.94%
	MAML + NN		83.87%	94.88%	**73.08%**	78.02%
	Meta-SGD + NN		**86.23%**	**96.46%**	72.98%	78.17%

denoted by NN-unified. (2) The SELF type represents the stand-alone approach where a distinct model is applied to each user, and there is no interaction among users. This type of models are directly trained on the support set of testing users, without using any data from training users. We choose the following baselines: most frequently used (MFU), most recently used (MRU), naive Bayes (NB) and classifier with the two architectures LR and NN used in the meta-learning approach.

The input feature vectors are constructed differently for different approaches. For both META and SELF, user-specific models are trained, and the feature vector encodes information in the services usage records with dimension 103. For MIXED, a unified model is trained across all users. To improve the prediction accuracy, the feature vector further encodes user ID and service ID, which has dimension 11892. The number of parameters in different architectures are summarized in Table 5.

Table 6 shows the Top 1 and Top 4 accuracies for the meta-learning approach and baselines. For each testing user, the last 20% records in chronological order are used as query set, and in "p Support" case the p fraction of records right before the query set are used as support set. In the MIXED approach, the unified model is trained with 80000 (gradient) steps during pre-training. In each step a batch of 500 data points (or half of the user records, whichever is smaller) are sampled from a single user. In the SELF approach, both LR and NN are trained on each testing user with 100 to 10000 steps. The batch size is 500 or half size of the query set. In the META approach, the algorithms are trained with 20000 episodes, each consisting of one task from a single training user during meta-training. In each meta-training task 500 data points (or half of the user records) are sampled, among which the first 80% are used as support set and the remaining as query set.

From the experimental results we observe the following.

(1) Comparing baselines in the SELF setting, naive Bayes serves as a simple yet strong baseline. With 10000 training steps, LR has an accuracy comparable with NB, and NN outperforms both LR and NB. However, both LR and NN are insufficiently trained in 100 steps.

(2) Comparing meta-learning algorithms in the META setting, Meta-SGD + NN achieves the highest accuracy. In particular, with other modules and settings being the same, Meta-SGD outperforms MAML and NN outperforms LR. The simplest combination MAML + LR performs significantly worse than other methods, implying that either the algorithm or the model should have certain complexity to guarantee performance of the meta-learning framework.

(3) Comparing MIXED, SELF and META, the meta-learning methods MAML + NN and Meta-SGD + NN generally outperform all baselines, while Meta-SGD + LR also has competitive performance. The MIXED approach has lower accuracies than strong baselines in SELF, among which NN performs best. However, we note that NN are trained up to 10000 gradient steps. On the other hand, all methods in META trains the model for each testing user with only 100 gradient steps, and Meta-SGD + NN still performs significantly better than the baseline NN. Although the meta-learning approach admits a meta-training procedure for updating algorithms (where only 100 gradient steps are executed in each training task as well), it demonstrates strong ability to fast adapt to new users with good performance, which is an important advantage for deployment in practice. Another interesting observation is that MRU has highest Top 4 accuracy in the "5% Support" case. This may be because in the dataset users use a small number of services in a short period of time, and MRU is not affected by the fraction of support set. However, as the support set expands, which is often the case in practice, MRU would be outperformed by meta-learning methods.

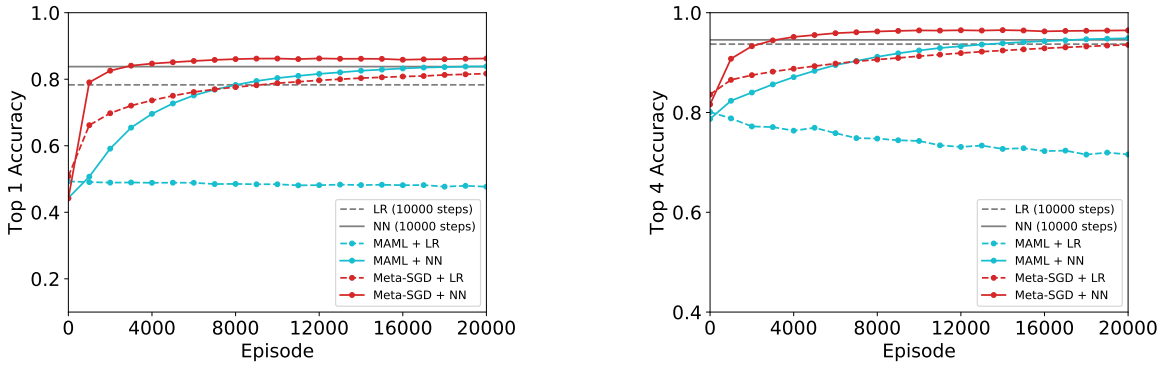

Figure 4. Convergence of Top 1 (left) and Top 4 (right) accuracies of meta-learning methods ("80% Support" case). Solid lines correspond to the NN model and dashed lines correspond to the LR model. Grey lines represent accuracies of SELF baselines LR and NN.

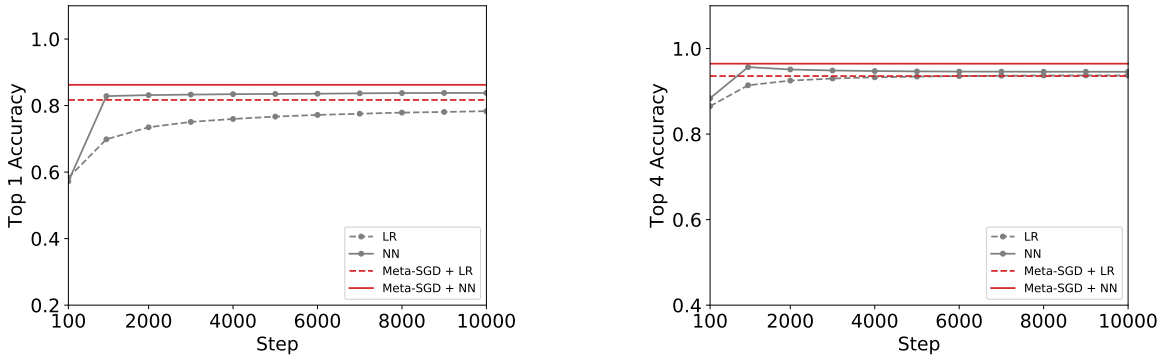

Figure 5. Convergence of Top 1 (left) and Top 4 (right) accuracies of SELF baselines LR and NN ("80% Support" case). Solid lines correspond to the NN model and dashed lines correspond to the LR model. Red lines represent accuracies of meta-learning algorithm Meta-SGD.

(4) Overall meta-learning algorithms help improve the performance of recommendation models. Meta-SGD + NN achieves the best performance. Taking into consideration network bandwidth, computation power and other resources for user devices to operate in the federated meta-learning framework, Meta-SGD + LR and MAML + NN provide a good balance between size (Table 5) and performance.

We further study the convergence of meta-learning methods in terms of the number of episodes during meta-training. as shown in Figure 4. The performance of MAML + LR surprisingly becomes worse as meta-training proceeds, especially in the Top 4 case. The other three methods converge within 20000 episodes, while Meta-SGD converges faster than MAML. In particular, Meta-SGD + NN already outperforms the best baseline NN after 4000 episodes.

In Table 6 the SELF baselines LR and NN perform significantly better when the number of training gradient steps increases from 100 to 10000. It remains to see how many steps are sufficient to train the models. Figure 5 shows that LR and NN indeed converge in 10000 steps, where the accuracy of LR is below or marginally above Meta-SGD + LR, while that of NN is below Meta-SGD + NN. We stress that Meta-SGD trains the models with only 100 steps, which is much more efficient than training the models from scratch by using (non-parametric) optimization algorithms.

4. Conclusion

In this work we introduced the federated meta-learning framework for training and deploying recommender systems that preserves user privacy while utilizing massive data from other users to build user-specific recommendation models. Experiments show that the two-level meta-learning approach performs better than both stand-alone models without pre-training and unified models with pre-training. The proposed framework can keep the algorithm and model at a small scale while maintaining high capacities, compared to unified models that are possibly trained in the federated learning approach. This may lead to reduced network bandwidth and computation resource consumption for model training and prediction, making it promising for deployment on user devices in practice. An interesting future direction is to investigate resource consumption of the federated meta-learning framework, and impacts of batch size and other factors on the performance as well.

References

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

Duan, Yan, Schulman, John, Chen, Xi, Bartlett, Peter L, Sutskever, Ilya, and Abbeel, Pieter. RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv preprint arXiv:1611.02779*, 2016.

Finn, Chelsea, Abbeel, Pieter, and Levine, Sergey. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pp. 1126–1135, 2017.

Guo, Huifeng, Tang, Ruiming, Ye, Yunming, Li, Zhenguo, and He, Xiuqiang. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI*, pp. 1725–1731, 2017.

Harper, F. Maxwell and Konstan, Joseph A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2015.

He, Xiangnan, Liao, Lizi, Zhang, Hanwang, Nie, Liqiang, Hu, Xia, and Chua, Tat-Seng. Neural collaborative filtering. In *WWW*, pp. 173–182, 2017.

Huang, Ke, Zhang, Chunhui, Ma, Xiaoxiao, and Chen, Guanling. Predicting mobile application usage using contextual information. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pp. 1059–1065. ACM, 2012.

Hug, Nicolas. Surprise, a Python library for recommender systems. http://surpriselib.com, 2017.

Koren, Yehuda. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pp. 426–434, 2008.

Li, Zhenguo, Zhou, Fengwei, Chen, Fei, and Li, Hang. Meta-SGD: Learning to Learn Quickly for Few Shot Learning. *arXiv preprint arXiv:1707.09835*, 2017.

McMahan, Brendan, Moore, Eider, Ramage, Daniel, Hampson, Seth, and y Arcas, Blaise Aguera. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, pp. 1273–1282, 2017.

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

Snell, Jake, Swersky, Kevin, and Zemel, Richard. Prototypical networks for few-shot learning. In *NIPS*, pp. 4080–4090. 2017.

Vartak, Manasi, Thiagarajan, Arvind, Miranda, Conrado, Bratman, Jeshua, and Larochelle, Hugo. A meta-learning perspective on cold-start recommendations for items. In *NIPS*, pp. 6907–6917. 2017.

Vinyals, Oriol, Blundell, Charles, Lillicrap, Tim, and Wierstra, Daan. Matching networks for one shot learning. In *NIPS*, 2016.

Wang, Jane X, Kurth-Nelson, Zeb, Tirumala, Dhruva, Soyer, Hubert, Leibo, Joel Z, Munos, Remi, Blundell, Charles, Kumaran, Dharshan, and Botvinick, Matt. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Wang, Ruoxi, Fu, Bin, Fu, Gang, and Wang, Mingliang. Deep & cross network for ad click predictions. *arXiv preprint arXiv:1708.05123*, 2017.

Yan, Tingxin, Chu, David, Ganesan, Deepak, Kansal, Aman, and Liu, Jie. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 113–126. ACM, 2012.

Zhou, Fengwei, Wu, Bin, and Li, Zhenguo. Deep meta-learning: Learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*, 2018.