Probabilistic Meta-learner for Few-Shot Collaborative Filtering

Anonymous Author(s)

Abstract

In this work, we consider *few-shot learning* problem in recommender systems. Deriving accurate recommendations for users with few known observations, i.e. *few-shot users*, is a challenging task for classic collaborative filtering (CF) algorithms. To address this inherent deficiency in CF, we develop a new recommendation framework namely *probabilistic meta-learning collaborative filtering* (PMLCF), which synergizes information sharing between related users through feature distillation and aggregation, then employs a flexible amortization network that takes the data of few-shot users as inputs, and outputs a distribution over parameters of a user-specific downstream estimator in a forward pass. As such, PMLCF amortizes the cost of inference, and enables fast prediction at test-time. Experimental results on a series of real-world benchmark tasks show that under strong and weak few-shot scenarios, the proposed method is superior to state-of-the-art algorithms.

Introduction

Given the constant arrival of large number of new users, how to quickly and effectively update recommender models to provide satisfactory recommendation for users with limited observations, i.e. few-shot users, poses a practical as well as theoretical challenge to existing recommendation techniques.

This work looks into the few-shot learning problem in the recommender system domain. The difficulty behind this problem is twofold: few-shot users might be unseen through the model training process, similar to cold-start problem (Schein et al. 2002; Vartak et al. 2017); and secondly, the observations for (un)known users we can use for inference are limited, as well as user profile which is arduous for service providers to collect. Therefore, we limit ourselves to implicit data, though PMLCF can also enjoy the benefits of content information.

The conventional model-based collaborative filtering approaches, including factor models (Salakhutdinov and Mnih 2008; Rendle et al. 2009) and neural models (Wu et al. 2016; He et al. 2017), typically require significant amount of parameter updates using stochastic gradient descent (Vinyals et al. 2016) when a new user is present. Learning the embedding vectors for new users is often slow and involves large datasets. As a result, these approaches usually are infeasible for dealing with fresh users in real-time. A much more desirable approach would be allowing quick adaptations for few-shot users *without changing the underlying model*.

Memory-based approaches (Herlocker et al. 1999; Sarwar et al. 2001) do not require any extensive training, and guarantee efficiency in most cases. For instance, nearest neighbors can be approximated in nearly constant time, thanks to hashing-based techniques (Gionis, Indyk, and Motwani 1999; Indyk and Motwani 1998). However, approaches in this family suffer from the curse of dimensionality, therefore often underperform model-based CF approaches (Rendle 2010).

Recently proposed graph-based approaches (Ying et al. 2018; Wang et al. 2019) have shown promising performance for many recommendation tasks. This kind of approach is conceptually superior to previous works, because intuitively incorporating connectivity features on user-item graph is beneficial to obtain better embeddings, analogous to (Koren 2008). However, we argue that existing models, like NGCF (Wang et al. 2019), are not well-suited for few-shot collaborative filtering. This is because dissimilar users (items) are likely to gather together when the data is very sparse, and the lack of explicit mechanisms to separate nuisance factors will degrade the performance.

In this paper, we formulate the few-shot recommendation problem within the well-studied framework of meta learning (Snell, Swersky, and Zemel 2017). The basic idea is to view the recommendation for a user as one task, then to derive a learning algorithm that takes as input the observations of a given user and outputs a posterior predictive distribution which describes user future behaviors. We parameterize the learning algorithm with a flexible amortization network. Limited amount of available data for new users will likely lead to overfitting. To address this issue, we propose feature distillation and feature aggregation operations that encode information shared across users into an essential representation, which is subsequently used in the downstream model for making personalized recommendation.

The contributions of our work are threefold: (1) formulation of the few-shot learning problem in the recommender system domain; (2) development of PMLCF, a new recommendation framework which meta-learns fast and accurate

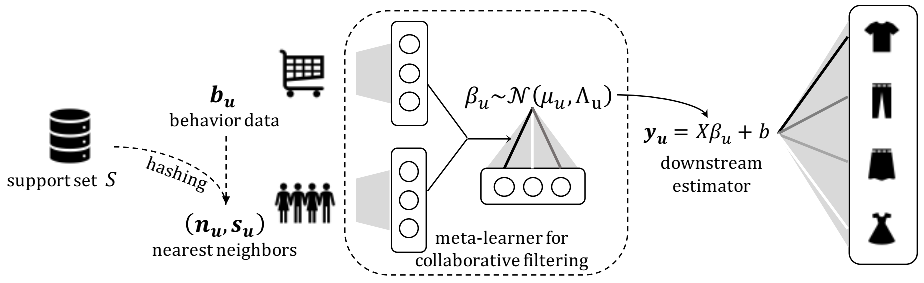

Figure 1: PMLCF framework. Given a user's behavior data $\mathbf{b}_u$ as a query, we search the support set S for nearest neighbors $\mathbf{n}_u$, together with cosine similarities $\mathbf{s}_u$. Then a meta-learner maps the data $(\mathbf{b}_u, \mathbf{n}_u, \mathbf{s}_u)$ to a user(task)-specific distribution $\mathcal{N}(\mu_u, \Lambda_u)$ over parameter β_u, which specifies a downstream estimator. And X, b are parameters shared across all users(tasks).

approximations to the posterior predictive distribution; and (3) evaluation using real-world datasets which demonstrates that PMLCF outperforms state-of-the-art techniques, including memory-based, model-based and graph-based algorithms.

Related Work

Meta Learning on Sparse Data. Meta-learning has shown promising results on rapid adaptation to new tasks which are small but related. Despite the success in computer vision (Vinyals et al. 2016; Oreshkin, López, and Lacoste 2018), it appears less studied in *rapid learning from sparse data*. A notable exception is MeLU (Lee et al. 2019), which builds upon MAML (Finn, Abbeel, and Levine 2017) to meta-learn a good weight initialization and use gradient decent update to fine-tune neural networks on sparse data. In contrast, our approach solves the problem in an entirely feed-forward manner with no model update required, thus makes it more convenient for real-time recommender systems.

There is also a flurry of research e.g. (Vartak et al. 2017), which predicts essential embeddings based on the content information, aligned with many cold-start solutions (Zhou, Yang, and Zha 2011; Zhang et al. 2014). Indeed, our PMLCF approach can also enjoy the benefits of content information (see Eq. (7)). Due to limited space, we left it for future work.

The most related methodologies to ours are (Qiao et al. 2018) and (Gordon et al. 2019). These approaches employ flexible amortization networks that take few-shot learning datasets, and output a distribution over task-specific parameters in a single forward pass. The sparse data will likely lead to poor learning. Therefore, we design two operations to explicitly distill and aggregate information shared across related tasks (users). This makes our approach more data-efficient, regardless of the sparsity of the data.

Shallow Matrix Factorization Models. Matrix factorization (MF) is one of the most popular approaches in collaborative filtering (Rendle et al. 2009; He et al. 2016). Among matrix factorization CF solutions, probabilistic matrix factorization (Mnih and Salakhutdinov 2007) and its Bayesian version (Salakhutdinov and Mnih 2008) are closely related to our approach. Their main deficiency is that when new data is encountered, the models must inefficiently relearn their parameters to adequately incorporate new information, suboptimal for the large datasets in real-world. In this paper, the authors investigate amortized inference model, powered by neural networks, that maps from observation space to latent factor (i.e., variational parameter) space. By doing so, the latent factor for unseen users can be obtained directly by feeding their observed data to neural networks.

Graph-based Collaborative Filtering Methods. Another line of research leverages graph techniques to model the user preference. Early efforts, such as SVD++ (Koren 2008), apply one embedding layer on the neighborhood to exploit connections between the users. Hence, the signals in high-order connectivities cannot be revealed. To alleviate the problem, the recently proposed method PinSage (Ying et al. 2018) applies multiple graph convolution layers on item-item graph to collect high-order item relations. Likewise, NGCF (Wang et al. 2019) adopts the idea of message passing to propagate the information between users and items so that collective user and item behaviors can be both achieved. Despite their effectiveness, we argue that in few-shot settings, these methods are inferior to shallow CF models due to the lack of explicit mechanisms to tackle nuisance variation among user/item relations (see empirical section for more details).

Neural Networks for Collaborative Filtering. With the evolution of neural networks, pioneering works (Van, Dieleman, and Schrauwen 2013; Wang, Wang, and Yeung 2015) focus on applying it to model the auxiliary information e.g., audio and text; collaborative deep learning methods (Cheng et al. 2016; He et al. 2017) endow the MF models with nonlinear neural networks for better modeling user-item interactions. However, neural models generally require large amounts of data, otherwise they would suffer from severe overfitting, for example in few-shot settings (see empirical section).

Probably most closely related to our approach is *Multi-VAE* (Liang et al. 2018), one of the few neural generative models for recommendation (Li and She 2017). Aside from the process of feature distillation and aggregation, the other difference lies in the choice of the distribution when modeling the implicit data. In *Multi-VAE*, each user-item pair is sampled i.i.d from multinomial distributions rather than normal distributions that is widely used (He et al. 2017). By contrast, our approach permits the items to be correlated so that we are conceptually superior to previous works in fitting the data. Empirical section highlights the comparison with *Multi-VAE* (Liang et al. 2018) and *NeuMF* (He et al. 2017).

The Model

Let us define a task of user u sampled from the task space $\mathcal{U}$, and we have observations $\mathbf{r}_u$ where $\mathbf{r}_{ui}$ equals to 1 if item i is purchased/viewed and 0 otherwise. To match inference at test time, observations $\mathbf{r}_u$ that are used to train the meta-learner can be divided into meta-training data $\mathbf{b}_u$ with a few examples (e.g., up to 3) and meta-test data $\mathbf{y}_u$. Tasks of all observed users compose the support set $S = \{\mathbf{b}_u, \mathbf{y}_u\}_{u=1}^{m}$, then k nearest-neighbors $\mathbf{n}_u$ and cosine similarities $\mathbf{s}_u$ can be computed with respect to the support set S. The performance of the meta learner can be evaluated on either already-seen tasks $\{\mathbf{r}_u, \tilde{\mathbf{y}}_u\}_{u=1}^{m}$ (i.e., weak few-shot setting) or never-before-seen tasks $\{\tilde{\mathbf{b}}_v, \tilde{\mathbf{y}}_v\}_{v=1}^{m'}$ (i.e., strong few-shot setting).

As mentioned before, the traditional model-based CF approaches e.g., BPMF (Salakhutdinov and Mnih 2008), would need to re-run the optimization procedure, if the system has new test observations $\tilde{\mathbf{b}}_v$ that we would like to use to infer $\tilde{\mathbf{y}}_v$. And also the number of parameters we need to optimize grows linearly with the number of users and items, not optimal in large datasets. To remedy these problems, we propose substituting optimization procedures at test time with forward passes through an amortization network.

To be specific, our probabilistic meta learning approach for few-shot collaborative filtering, illustrated in Fig. 1, relies on two key components: (1) a parameterized function f_Θ which is broadly applicable to different user tasks, maps from the observations $(\mathbf{b}_u, S)$ to a distribution $\mathcal{N}(\mu_u, \Lambda_u)$ over the downstream model's parameter β_u; and (2) a linear model composed of β_u is then used to make predictions:

$$\mu_u, \Lambda_u = f_\Theta(\mathbf{b}_u, S), \beta_u \sim \mathcal{N}(\mu_u, \Lambda_u) \tag{1}$$

$$\mathbf{y}_u = X\beta_u + b \tag{2}$$

where μ_u, Λ_u are the mean vector and diagonal covariance matrix of the gaussian distribution respectively, and the matrix X and bias vector b denote internal representations that are learned from the data. The parameters X, b can be shared across all users such that the data for similar users can help to estimate the unknown user-item interactions.

Conceptually, the approximate posterior predictive distribution $q_\Theta(\mathbf{y}_u|\mathbf{b}_u, S)$ as in Eq. (2) assumes that user preferences $\mathbf{y}_u$ are distributed normally, analogous to (Mnih and Salakhutdinov 2007; Salakhutdinov and Mnih 2008):

$$q_\Theta(\mathbf{y}_u|\mathbf{b}_u, S) = \mathcal{N}(\mathbf{y}_u|X\mu_u + b, (X^\top \Lambda_u^\top \Lambda_u X)^{1/2}) \tag{3}$$

where the mean $X\mu_u + b$ describes how much the user would like the items and the covariance matrix $X^\top \Lambda_u^\top \Lambda_u X$ accounts for variations in decision making or the confidence we have in the predictions. Besides the superiority in system scalability and flexibility, another advantage over previous works lies in the structure of covariance matrix i.e., our approach assumes $\mathrm{var}(\mathbf{y}_u)$ is of low rank rather than isomorphism. This enables our estimator to fit the correlations among all items (e.g., opinions regarding the movies of a particular actor) and thus makes potential for the accuracy improvement.

Probabilistic Meta Learner for CF

In practice, we might employ deep neural networks to approximate the function f_Θ, but the limitation of such a scheme is that neural networks usually require large amounts of data, which becomes a bottleneck for commercial recommender systems. Intuitively, one of the possible solutions is to synergize the information sharing across related tasks (users), like SVD++ (Koren 2008) which enriches user representations by propagating embeddings among users' neighbors $\mathbf{n}_u$.

However, we argue that the conventional routine, which computes the sum of the embeddings of all nearest neighbors as ancillary part of the user representation, is sub-optimal in few-shot settings. The reason is twofold: (1) one user can be surrounded by dissimilar nearest neighbors under few-shot settings, and such nuisance neighbors might degrade the model performance; and (2) high-level neighbor-neighbor interactions have not been well utilized in such routine, and a notable exception is the neural graph CF model (Wang et al. 2019). Nevertheless, empirical results showed that it suffered from overfitting in few-shot settings, due to the lack of an explicit mechanism to separate informative features that are useful for collaborative filtering from nuisance noise.

To tackle above problems, we build upon neural networks to learn the high-level collaborative features across the tasks of users' neighbors, and formulate this procedure with two major operations: *feature distillation* and *feature aggregation*.

Feature Distillation. For one user's k nearest neighbors $\mathbf{n}_u \in \mathbb{N}_+^k$, we start by embedding each neighbor into a continuous vector with a length of d_n, such that the whole set of k nearest neighbors constitutes a feature map $N_u \in \mathbb{R}^{k \times d_n}$.

To filter nuisance neighbors, we propose learning an adaptive weighting function which allows the most similar users to contribute the most at the final representations. The neighbor-level filtering function in terms of related similarities $\mathbf{s}_u$ is defined as follows

$$N_u^{(0)} = \mathrm{ReLU}(W_s \mathbf{s}_u + b_s) \odot N_u \tag{4}$$

where $\odot$ represents element-wise product, and W_s, b_s denote transformation matrix and bias respectively. The ReLU (Nair and Hinton 2010) activation determines how each neighbor impacts the output representations – larger (smaller) values of the output encourage (suppress) the contribution; when the output equals to zero, the contribution is ignored. By doing so, we are able to focus on a subset of users within the neighborhood, without specifying the number of users.

To distill neighbor-neighbor features, we resort to the 1D dilated convolution (Yu and Koltun 2015) rather than 2D regular convolution (Van et al. 2016). This is because the component-level (column-wise) interactions are complicated, requiring more sophisticated networks (Lian et al. 2018). The advantage of using dilated convolution is that it can exponentially increase the effective receptive field size without increasing computational cost. Mathematically, 1D convolution with the dilation d is defined as

$$(N_u^{(l)} *_d k)_p = \sum_{s+dt=p} N_{u,s}^{(l)} k_t$$

where $N_u^{(l)}$ is the input feature map at *l-th* layer and k is the convolution kernel. It is worth mentioning that the *p-th* row vector of the output feature map is a linear combination of a subset of rows of the input feature map. As such, non-linear and high-level correlations among neighbors can be obtained by stacking dilated convolutional neural networks

$$N_u^{(1)} = \psi(W_0 *_{d_0} N_u^{(0)}), \ldots, N_u^{(L+1)} = \psi(W_L *_{d_L} N_u^{(L)})$$

where all Ws represent weight matrices used for transformation and all bs denote the bias parameters, and d_l ($l \in [0, L]$) controls the dilation of the l-th convolutional layer. Here we use ReLU as the activation ψ to help detect nonlinearities in resulting features and avoid vanishing gradients.

After propagating with L convolutional layers, we obtain the neighboring representations for user u, namely $N_u^{(L+1)}$. Since the representations obtained in different positions emphasize different associations with the user, they have different contributions in reflecting user preference. As such, we concatenate them into the final representation g for the user:

$$g = \mathrm{concat}(N_u^{(L+1)}). \tag{5}$$

We note that besides concatenation, other techniques such as LSTM and pooling can also be applied. The reason why we use concatenation is that it involves no additional parameters to learn, and experimental results have shown its efficacy in learning embeddings that meet desired properties. To summarize, this feature filtering operation not only increases the model representation ability, but also boosts the performance for recommendation (see our empirical section).

Feature Aggregation. Compared to user nearest neighbor data $(\mathbf{n}_u, \mathbf{s}_u)$ which provides the implicit evidence on a user's preference, the user's historical behavior data $\mathbf{b}_u$ exhibits more fine-grained user profiles. For instance, we can deduce from the purchase of luxury-brand high heels that the consumer might probably be a young lady with good salary. As a result, we aggregate the embeddings of these two kinds of the data by regarding learned features from the behavior data $\mathbf{b}_u$ as the basis, then using the neighbor data $(\mathbf{n}_u, \mathbf{s}_u)$ to complement the basis in a gradient update manner

$$h(\mathbf{b}_u) \leftarrow h(\mathbf{b}_u) + \alpha(\mathbf{n}_u, \mathbf{s}_u) \odot \nabla(\mathbf{n}_u, \mathbf{s}_u) \tag{6}$$

where $h(\mathbf{b}_u)$ is the behavior-dependent feature, $\alpha(\mathbf{n}_u, \mathbf{s}_u)$ and $\nabla(\mathbf{n}_u, \mathbf{s}_u)$ work as neighbor-dependent learning rate and gradients with respect to the parameters $h(\mathbf{b}_u)$ respectively. In effect, this establishes an explicit mechanism to reduce the component-level noise impact, where learning rate $\alpha(\mathbf{n}_u, \mathbf{s}_u)$ controls which component of the gradients $\nabla(\mathbf{n}_u, \mathbf{s}_u)$ can be passed to the feature $h(\mathbf{b}_u)$ – larger (smaller) values encourage (suppress) the feature refinement and noticeably the gradient would be filtered if the learning rate equals to zero.

To put everything together, the function f_Θ used to amortize the approximate posterior $q_\Theta(\beta|\mathbf{b}_u, S)$ is defined as

$$h(\mathbf{b}_u) = F\mathbf{b}_u + b_0 \tag{7}$$
$$\alpha(\mathbf{n}_u, \mathbf{s}_u) = \mathrm{sigmoid}(Gg + b_1) \tag{8}$$
$$\nabla(\mathbf{n}_u, \mathbf{s}_u) = \tanh(Hg + b_2) \tag{9}$$
$$h_\mu, h_\Lambda = \mathrm{split}(h(\mathbf{b}_u) + \alpha(\mathbf{n}_u, \mathbf{s}_u) \odot \nabla(\mathbf{n}_u, \mathbf{s}_u)) \tag{10}$$
$$\mu_u = \mathrm{sigmoid}(h_\mu) \tag{11}$$
$$\Lambda_u = \mathrm{diag}(\exp(h_\Lambda)) \tag{12}$$

where F, G, H are transformation matrices and b_0, b_1, b_2 are biases, and g is the latent representation for the user nearest neighbor data $(\mathbf{n}_u, \mathbf{s}_u)$ as defined in Eq. (5).

The key characteristics of the proposed feature aggregator can be summarized in three aspects: (1) Eq. (7) applies two-layer linear perceptron to learn behavior-dependent features, and the non-linear activation function is performed in Eq. (11)-(12). It is worth pointing out that Eq. (7) can also enjoy recent advances in neural recommender systems (Guo et al. 2017; Lian et al. 2018), if the content information is available; and (2) Eq. (8) and (9) employ neural networks to approximate the learning rate $\alpha(\mathbf{n}_u, \mathbf{s}_u)$ which determines how much of each component of the gradients should be let through, and the gradients $\nabla(\mathbf{n}_u, \mathbf{s}_u)$ which describe the shifting direction to approach a local extremum of the optimization function, respectively; and (3) Eq. (10) makes the output representations h_μ, h_Λ dependent on both a user's behavior data $\mathbf{b}_u$ and neighbor data $(\mathbf{n}_u, \mathbf{s}_u)$, i.e., passing different levels of the learned features to the final representations. This not only enhances the expressivity of the model, but also makes the potential for the improved performance.

Meta-Training with Kernel MMD

To learn the parameters, we minimize the expected value of the KL divergence $D_{\mathrm{KL}}[p(\mathbf{y}_u|\mathbf{b}_u, S) \| q_\Theta(\mathbf{y}_u|\mathbf{b}_u, S)]$, which is equivalent to maximizing the following objective

$$\begin{aligned} &\mathrm{E}_{u\sim\mathcal{U}}\mathrm{E}_{(\mathbf{b}_u,\mathbf{y}_u)\sim\mathbf{r}_u} \ln \int p(\mathbf{y}_u|X, b, \beta) q_\Theta(\beta|\mathbf{b}_u, S)\,\mathrm{d}\beta \\ &= \mathrm{E}_{u\sim\mathcal{U}}\mathrm{E}_{(\mathbf{b}_u,\mathbf{y}_u)\sim\mathbf{r}_u} \Big\{ -\frac{n}{2}\ln 2\pi - \frac{1}{2}\ln\det(\Sigma_u^{1/2}) \\ &- \frac{1}{2}(\mathbf{y}_u - X\mu_u - b)^\top \Sigma_u^{-1}(\mathbf{y}_u - X\mu_u - b)\Big\} \end{aligned} \tag{13}$$

where Σ_u is the covariance matrix $X^\top \Lambda_u^\top \Lambda_u X$. The main difficulty when dealing with Eq. (13) is that in most cases, the covariance matrix Σ_u is a degenerate matrix which is not invertible, such that standard gradient descent methods cannot be applied, even though μ_u, Λ_u are deterministic functions with parameters Θ that can be learned from the data.

To solve the problem, we resort to maximum mean discrepancy (MMD) (Gretton et al. 2007), which measures the difference between the mean function values (e.g., the moments) on the samples drawn from the data distribution $p(\mathbf{y}_u|\mathbf{b}_u, S)$ and the model distribution $q_\Theta(\mathbf{y}'_u|\mathbf{b}_u, S)$ separately. When the difference is large, the samples are likely from different distributions. In general, MMD can be computed efficiently via the kernel embedding trick:

$$\begin{aligned} &\mathrm{MMD}\left(p(\mathbf{y}_u|\mathbf{b}_u, S) \| q_\Theta(\mathbf{y}'_u|\mathbf{b}_u, S)\right) \\ &= \sup_{\|f\|_{\mathcal{H}}\le 1} \left(\mathrm{E}_{y_u\sim p}[f(y_u)] - \mathrm{E}_{y'_u\sim q}[f(y'_u)]\right) \\ &= \mathrm{E}_{p,p}k(\mathbf{y}_u, \mathbf{y}'_u) - 2\mathrm{E}_{p,q}k(\mathbf{y}_u, \mathbf{y}'_u) + \mathrm{E}_{q,q}k(\mathbf{y}_u, \mathbf{y}'_u) \end{aligned} \tag{14}$$

where $k(\mathbf{y}_u, \mathbf{y}'_u)$ is any universal kernel in a RKHS $\mathcal{H}$. The advantage of using MMD lies in its efficiency, since the inverse of the matrix Σ_u is no longer required to compute. And the overall learning process can be done by first expressing the samples from the model distribution $q_\Theta(\mathbf{y}'_u|\mathbf{b}_u, S)$ as a *deterministic* function of its parameters Θ and a fixed source of randomness, then back-propagating the gradients with respect to Θ through the samples. Note that by doing so, the computational cost is reduced from $\mathcal{O}(n^3)$ of the matrix inverse, to *mainly* $\mathcal{O}(n)$ linear to the number of items n.

Reparameterized Sampler. To back-propagate the error, we make up a reparameterized sampler by following (Kingma and Welling 2013): given the mean and diagonal covariance μ_u, Λ_u, we can sample from $\mathcal{N}(X\mu_u + b, (X^\top \Lambda_u^\top \Lambda_u X)^{1/2})$

Table 1: Statistics of the evaluation datasets. Similarity@5 denotes the mean similarity averaged over the top-5 nearest users.

Dataset	#Users	#Items	#Interactions	Density (%)	Similarity@5
Automotive	22,030	6,385	92,388	0.065	0.545
Baby	9,182	1,712	68,642	0.436	0.392
MovieLens 10M	69,781	10,614	9,746,146	1.315	0.408

by first sampling $\epsilon \sim \mathcal{N}(0, I)$, then reparameterizing $\beta'_u = \Lambda_u \epsilon + \mu_u$, and finally calculating $\mathbf{y}'_u = X\beta'_u + b$. By doing so, given a fixed $(\mathbf{b}_u, \mathbf{n}_u, \mathbf{s}_u, \epsilon)$, the sample $\mathbf{y}'_u$ is a deterministic function and continuous in the parameters of the model, indicating that the gradients with respect to Θ can be computed and back-propagated through this sampling process.

Kernelized Objective. The choice of the kernel function used in Eq. (14) would largely affect the performance of the downstream estimator. In the literature, the radial basis function (RBF) kernel is widely used, which is defined as

$$k(\mathbf{y}_u, \mathbf{y}'_u) = \exp(-(\mathbf{y}_u - \mathbf{y}'_u)^\top \bar{\Lambda}^{-1}(\mathbf{y}_u - \mathbf{y}'_u)). \tag{15}$$

where the diagonal matrix $\bar{\Lambda}$ is a free parameter. Distinct from other kernels like polynomial kernel, the RBF kernel provides a natural way to express the uncertainty in the observations. To be specific, the label $\mathbf{y}_u$ is very sparse and only contains positive examples, perhaps more importantly the missing data is a mixture of the unknown and negative feedback. As such, it is straightforward to customize $\bar{\Lambda}$ to specify the confidence in the missing entries to be negative.

In coincidence with (He et al. 2016) which utilizes item popularity to express the confidence, we refine the strategy by taking user activation into account. Formally, we define the confidence $\bar{\Lambda}_{ii}^{-1}$ for the user u to the item i as follows

$$\bar{\Lambda}_{ii}^{-1} = \begin{cases} 1 & \text{if } \mathbf{y}_{u,i} = 1 \\ c_0 \frac{c_{\text{item}}^{\alpha}(i)}{\sum_{j=1}^{n} c_{\text{item}}^{\alpha}(j)} \min(\frac{c_{\text{user}}^{\alpha}(u)}{\sum_{j=1}^{m} c_{\text{user}}^{\alpha}(j)/m}, c_1) & \text{else} \end{cases}$$

where the functions $c_{\text{user}}(u)$ ($c_{\text{item}}(i)$) return the times that user u (item i) presents in the data, c_0 determines the overall weight of missing data, and α controls the significance of the discrepancy in frequency, and c_1 limits the maximum impact and has a smoothing effect.

Ranking Prediction

We recall that in this work the user opinions $\mathbf{y}_u$ is uncertain and distributed normally. To produce personalized recommendations, one simple practice is to rank the expected scores

$$\mathrm{E}\,\mathbf{y}_{ui} = X_i^\top \mu_u + b \tag{16}$$

in the descending order and recommend the top-*N* items to the user. Alternatively, we can rank the items in term of the probability that current item *i* is in the recommended list:

$$\begin{aligned} &\Pr(\mathbf{y}_{ui} \geq c_u | \mathbf{b}_u, S, \Theta) \\ &= \iint_D \text{p.d.f}(\mathbf{y}_u)\, \mathrm{d}\mathbf{y}_u \approx \frac{1}{J}\sum_{j=1}^{J} \mathbf{1}|\hat{\mathbf{y}}_{ui}^{(j)} \geq c_u| \end{aligned} \tag{17}$$

where c_u is the *N-th* highest score in $\mathrm{E}\,\mathbf{y}_u$ and all $\hat{\mathbf{y}}_{ui}^{(j)}$ are random samples of $\mathbf{y}_{ui}$. This sampling method with $J = 30$ empirically showed slight improvement, but its computational overhead is $J = 30$ times larger than that of the simple method based solely on expected scores. Thus, we still adopt the simple prediction method in the empirical section.

PMLCF Generalizes Matrix Factorization

In this section, we show how our proposed PMLCF generalizes NMF (Lee and Seung 2001), and in what follows, we emphasize the difference to conventional matrix factorization based CF models.

NMF can be viewed as a special case of PMLCF with neither the capacity of amortized inference nor the mechanism to synergize information sharing across related tasks. In particular, to maximize the objective function defined in Eq. (13), it is equivalent to minimize the sum of squared errors loss function with regularization terms. And for the covariance matrix Σ_u, we disable the correlations between the items, namely $\Sigma_u = \gamma^{-1}I$. Thereafter, the optimization objective can be formulated as:

$$\min_{\Theta, X, b} \gamma\|\mathbf{y}_u - \mathbf{y}'_u\|^2 + \lambda\|X\|^2 \tag{18}$$

where we set $\mathbf{y}'_u = X\mu_u + b$, and λ determines the strength of the regularization.

By dividing μ_u as a product of a positive constant C and a positive vector p, the estimate $\mathbf{y}'_u$ can be written as

$$\mu_u = Cp_\Theta,\ C = \sum_i \mu_{ui} \tag{19}$$

$$p_\Theta(i) = p_\Theta(X_i|\mathbf{b}_u, S) = \mu_{ui}/C \geq 0 \tag{20}$$

$$\mathbf{y}'_u = CXp_\Theta + b \tag{21}$$

where Eq. (19) holds since the output of the *sigmoid* function is non-negative, Eq. (20) amortizes the cost of inferring the posterior distribution and offers fast test-time performance, and Eq. (21) resembles NMF where the non-negative coefficients $p_\Theta(i)$ pick up related elements from the dictionary X to make recommendations.

To summarize, PMLCF enjoys the merits of traditional matrix factorization CF models, but more importantly it acquires the power to fast adapt to new users even in few-shot settings, without any changes to the model.

Experiments and Results

This section evaluates our proposed model, and we refer the readers to supplementary materials for baseline analysis.

Experimental Setup

We conduct the experiments on three real-world datasets: Automotive and Baby product categories from the Amazon dataset [1], and the MovieLens (ML) 10M dataset [2]. Table 1 summarizes statistics of the data. In the experiments, we adopt evaluation metrics of Precision (P), Recall (R), HitRate(H) and Normalized Discounted Cumulative Gain (N), and all reported results are averaged over five train-test splits.

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

[2]https://grouplens.org/datasets/movielens/

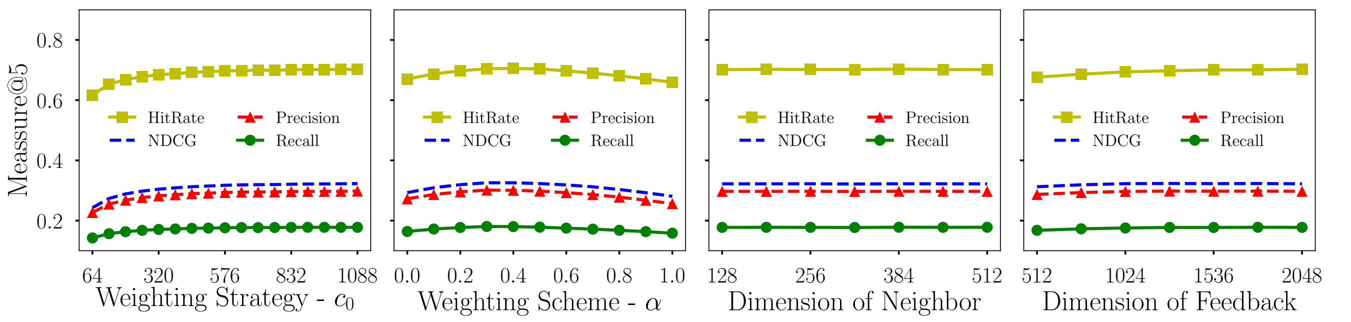

Figure 2: Accuracies for top-5 ranking tasks of PMLCF with varying c_0 (left) and α (left middle) used in the weighting strategy, the embedding size of neighbors (right middle) and the embedding size of implicit feedback (right) on the ML10M dataset.

Table 2: Performance in weak few-shot scenario. All numbers are percentage numbers with '%' omitted. Boldfaces mean that the method performs statistically significantly better under t-tests, at the level of 95% confidence level.

Dataset	Automotive				Baby				MovieLens 10M			
Measure@5(%)	P	R	H	N	P	R	H	N	P	R	H	N
MostPopular	0.48	2.10	2.40	0.53	0.59	1.64	2.91	0.62	7.77	3.90	25.1	8.10
User-based	1.76	7.86	8.10	2.14	0.65	1.96	3.20	0.73	17.0	10.9	53.4	19.3
BPR	2.03	9.17	9.98	2.43	0.88	2.66	4.27	0.96	26.3	15.5	65.9	28.4
eALS	2.06	9.23	10.0	2.45	0.88	2.77	4.25	0.98	26.6	16.9	67.3	28.5
NeuMF	0.79	3.37	3.89	0.91	0.63	1.70	3.10	0.64	27.6	16.4	67.7	29.7
NGCF	1.17	5.26	5.83	3.93	0.59	1.86	2.93	0.62	25.4	14.8	64.9	27.3
Mult-VAE	1.70	7.72	8.41	2.02	0.94	2.98	4.58	1.01	27.4	17.5	69.0	28.8
Ours-S, k=0	1.66	7.30	8.20	1.82	1.01	3.15	4.93	1.06	29.6	17.4	69.1	32.3
Ours-S, k=5	1.83	8.22	9.07	2.12	1.01	3.17	4.93	1.07	29.9	17.7	69.8	32.5
Ours-S, k=50	2.06	9.23	10.2	2.43	1.01	3.17	4.94	1.07	30.6	18.1	70.7	33.0
Ours-M, k=50	2.06	9.22	10.2	2.42	1.01	3.18	4.98	1.07	30.6	18.1	70.7	33.0
Ours-L, k=50	2.06	9.23	10.2	2.43	1.01	**3.22**	**4.99**	**1.11**	**30.6**	**18.1**	**70.7**	**33.0**

We use Adam (Kingma and Ba 2015) to optimize our model and half learning rate every 5 epochs to accelerate the training process. We perform grid search of learning rate and ℓ_2 regularizer over {5e-3, 2e-4, 1e-4} and {2e-3, 1e-4, 5e-5} to select the best parameters. We use batch size 1024 and 64 on the MovieLens dataset and Amazon datasets respectively.

Sensitivity Analysis

This study uses ML10M dataset with the ratio of training and test data 7:3. We set as default parameters c_0=512 and α=0.5 in the weighting strategy, the embedding size of neighbors d_n=128 and implicit feedback d_h=1024. We recommend 5 items (i.e., N=5) for each user.

Figure 2 (left) and Figure 2 (left middle) present the effects of the weighting strategy with varying c_0 and α. As we can see, all four ranking measures increase along with the increase of the parameter c_0, though the performance growth slows down. And we observe that the recommendation accuracies first increase as the scaling parameter α increases, then decrease after the optimal accuracies are achieved. We note that when α = 0.0, the proposed weighting strategy will be reduced to uniform sampling. All above results demonstrate the superiority of our kernel MMD objective.

Figure 2 (right middle) analyzes the effects of the embedding size d_n of neighbors as in Eq. (4). We can see that all ranking measures keep stabilized, as the model capacity for fitting the data increases with the growing embedding size. Thus we adopt d_n=128 for all following experiments.

Figure 2 (right) shows the effects of the embedding size d_h of implicit feedback as in Eq. (7). As can be seen, the accuracies get better when the embedding size gets bigger. With the larger embedding size, however, overfitting seems not to be a problem.

Accuracy Comparisons

This section compares the performance of our PMLCF with other methods. Specifically, we compare with memory-based approaches (MostPopular and User-based (Herlocker et al. 1999)), with shallow point-wise and pairwise matrix factorization (eALS (He et al. 2016) and BPR (Rendle et al. 2009)), with best neural networks models (NeuMF (He et al. 2017) and Multi-VAE (Liang et al. 2018)), and with state-of-the-art neural graph model (NGCF (Wang et al. 2019)).

We consider two real-world few-shot evaluation settings: (1) *weak few-shot scenario*, where we split the data of each user into training/test sets with the ratio 7:3 to simulate user few-shot scenario (for example on Automotive dataset, each user only has three records in average.); and (2) *strong few-shot scenario*, where we split all users into training/test users with the ratio 9:1. We train the model on the entire data of training users. Each test user who is unseen in the training procedure is provided three history records to make inference,

Table 3: Performance in strong few-shot scenario. All numbers are percentage numbers with '%' omitted. Boldfaces mean that the method performs statistically significantly better under t-tests, at the level of 95% confidence level.

	Automotive				Baby				MovieLens			
Measure@5(%)	P	R	H	N	P	R	H	N	P	R	H	N
MostPopular	0.96	1.77	4.42	1.07	2.08	3.13	12.5	2.93	46.0	3.11	75.9	46.6
User-based	3.33	8.63	14.9	3.86	3.36	4.03	14.9	3.77	59.4	4.74	87.4	60.6
Mult-VAE	3.29	7.96	14.3	3.65	3.55	4.19	15.9	4.01	59.3	4.88	89.1	60.3
Ours-S, k=0	1.94	4.02	8.46	2.02	2.72	3.01	12.0	2.80	53.3	4.15	86.5	54.8
Ours-S, k=5	3.09	7.60	14.3	3.48	4.30	5.08	18.4	4.64	59.7	5.08	89.3	61.0
Ours-S, k=50	3.26	7.68	14.9	3.62	4.37	5.12	18.8	4.70	59.9	5.10	89.5	61.4
Ours-M, k=50	3.51	8.52	16.1	4.08	4.43	5.17	19.3	4.85	60.3	5.10	89.8	61.5
FSCF-L, k=50	**3.69**	**8.88**	**16.8**	**4.15**	**4.52**	**5.40**	**19.0**	**4.82**	**60.6**	**5.12**	**90.1**	**61.7**

and the remaining data is used for evaluation.

And we also present the impact of the convolution layer in feature distillation operation by varying the depth and dilation from [1,2,4], [1,2,4, 8] to [1,2,4, 8, 16]. They represent small, medium, and large models and we simply term them as Ours-S, Ours-M and Ours-L.

Weak Few-shot Scenario. Table 2 shows the accuracies of all baselines on all three datasets, where we recall that k is the number of nearest neighbors.

For the Automotive product dataset which is extremely sparse (i.e., density 0.065 %), shallow factor models BRP and eALS outperform neural models NeuMF, NGCF and Mult-VAE. This discloses the main deficiency of neural models – requiring large labeled data. It is interesting to see that our PMLCF method first underperforms factor models but then achieves comparable results as the number of nearest neighbors k increases.

The experiment on the Baby product dataset studies how our model performs with less similar neighbors. Interestingly, the accuracies of all baselines degrade, even though the Baby dataset is much denser than the Automotive dataset. This can be explained by the nature of collaborative filtering which is to exploit the history of *similar* users to collaboratively infer the future. Another interesting fact is that the local neighborhood contributes nothing to the accuracy improvement, in contrast to prior results. This is due to the insufficient features among neighbors.

We also present the results on the MovieLens 10M dataset which has comparable Similarity@5 with the Baby product dataset but is much denser. We can see that neural models dominate the best results in all metrics, and all PMLCF models present consistent improvement over all seven baselines. And the accuracies of our models increase as the number of neighbors increases, even if users are dissimilar to each other. This might be caused by the increasing data density which enriches the diversified features between related users.

Strong Few-Shot Scenario. Table 3 presents the results on all three datasets, where we omit the comparisons with BPR, eALS, NGCF and NeuMF because they need to re-run optimization process to learn the embedding for unseen users. We can see from the results that the user-based method performs comparably with Mult-VAE which achieves best baseline results in weak few-shot scenario. This phenomenon highlights the extreme difficulty of the strong few-shot problem. Notably, our large model with $k = 50$ significantly outperforms all baselines. Similar to prior study, all metrics of PMLCF models are improved when we increase the number of nearest neighbors k from 0 to 50 and neural networks from small (i.e., Ours-S) to large(i.e., Ours-L).

Table 4: Runtime (seconds) per epoch on ML10M dataset.

BPR	eALS	NeuMF	NGCF	Multi-VAE	Ours
38.56	22.83	724.19	18069.20	17.83	13.80

Overall Discussion. To summarize, the reasons why our PMLCF method can further improve the recommendation accuracies are: (1) the new meta-learning framework which takes the data uncertainty into account and shares information between tasks through feature distillation and aggregation, offers the ability to rapid learning from sparse data; (2) the kernel objective function provides a convenient way to meta-train the underlying amortization network regardless of the scarcity of the data;

Efficiency Comparisons

Table 4 shows the execution time on the ML10M dataset under weak few-shot scenario, and we present the performance of PMLCF which achieves the best accuracies (i.e., ours-S with k=50 in Table 2). All results were run on a machine with E5-2678 CPU, GTX 1080Ti GPU and 188G memory. It is worth mentioning that NGCF considers the connectivity to all related users (items) rather than a few nearest neighbors. This makes the training time-consuming. We can see that PMLCF is even more efficient than the shallow models BPR and eALS. This study emphasizes that PMLCF beats state-of-the-art methods in both efficiency and accuracy.

Conclusion

We identify the importance of user few-shot problem, then propose a probabilistic meta learning framework to address the problem. Specifically, we learn a learning algorithm that takes as input the implicit data of a user, and outputs a linear model that predicts user behaviors in the future. Empirical results on multiple datasets demonstrate that the proposed method outperforms all state-of-the-art models including best factor models and best neural models, and achieves state-of-the-art performance for the user few-shot problem.

References

Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; et al. 2016. Wide & deep learning for recommender systems. *arXiv preprint arXiv:1606.07792*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 1126–1135.

Gionis, A.; Indyk, P.; and Motwani, R. 1999. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, 518–529.

Gordon, J.; Bronskill, J.; Bauer, M.; Nowozin, S.; and Turner, R. 2019. Meta-learning probabilistic inference for prediction. In *ICLR*.

Gretton, A.; Borgwardt, K.; Rasch, M.; Schölkopf, B.; and Smola, A. J. 2007. A kernel method for the two-sample-problem. In *NIPS*, 513–520.

Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. Deepfm: A factorization-machine based neural network for ctr prediction. In *IJCAI*, 1725–1731.

He, X.; Zhang, H.; Kan, M.-Y.; and Chua, T.-S. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 549–558.

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *WWW*, 173–182.

Herlocker, J. L.; Konstan, J. A.; Borchers, A.; and Riedl, J. 1999. An algorithmic framework for performing collaborative filtering. In *SIGIR*, 230–237.

Indyk, P., and Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 604–613. ACM.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Koren, Y. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, 426–434.

Lee, D. D., and Seung, H. S. 2001. Algorithms for non-negative matrix factorization. In *NIPS*, 556–562.

Lee, H.; Im, J.; Jang, S.; Cho, H.; and Chung, S. 2019. Melu: Meta-learned user preference estimator for cold-start recommendation. In *KDD*, 1073–1082. ACM.

Li, X., and She, J. 2017. Collaborative variational autoencoder for recommender systems. In *KDD*, 305–314. ACM.

Lian, J.; Zhou, X.; Zhang, F.; Chen, Z.; Xie, X.; and Sun, G. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *KDD*, 1754–1763. ACM.

Liang, D.; Krishnan, R. G.; Hoffman, M. D.; and Jebara, T. 2018. Variational autoencoders for collaborative filtering. In *WWW*, 689–698.

Mnih, A., and Salakhutdinov, R. 2007. Probabilistic matrix factorization. In *NIPS*, 1257–1264.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*, 807–814.

Oreshkin, B.; López, P. R.; and Lacoste, A. 2018. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NIPS*, 721–731.

Qiao, S.; Liu, C.; Shen, W.; and Yuille, A. L. 2018. Few-shot image recognition by predicting parameters from activations. In *CVPR*, 7229–7238.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 452–461.

Rendle, S. 2010. Factorization machines. In *ICDM*, 995–1000.

Salakhutdinov, R., and Mnih, A. 2008. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, 880–887.

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*, 285–295.

Schein, A. I.; Popescul, A.; Ungar, L. H.; and Pennock, D. M. 2002. Methods and metrics for cold-start recommendations. In *SIGIR*, 253–260. ACM.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *NIPS*, 4077–4087.

Van, Oord, A.; Kalchbrenner, N.; Espeholt, L.; Vinyals, O.; Graves, A.; et al. 2016. Conditional image generation with pixelcnn decoders. In *NIPS*, 4790–4798.

Van, Oord, A.; Dieleman, S.; and Schrauwen, B. 2013. Deep content-based music recommendation. In *NIPS*, 2643–2651.

Vartak, M.; Thiagarajan, A.; Miranda, C.; Bratman, J.; and Larochelle, H. 2017. A meta-learning perspective on cold-start recommendations for items. In *NIPS*, 6904–6914.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *NIPS*, 3630–3638.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T.-S. 2019. Neural graph collaborative filtering. In *SIGIR*, 165–174.

Wang, H.; Wang, N.; and Yeung, D.-Y. 2015. Collaborative deep learning for recommender systems. In *KDD*, 1235–1244.

Wu, Y.; DuBois, C.; Zheng, A. X.; and Ester, M. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*, 153–162. ACM.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 974–983. ACM.

Yu, F., and Koltun, V. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.

Zhang, M.; Tang, J.; Zhang, X.; and Xue, X. 2014. Addressing cold start in recommender systems: A semi-supervised co-training algorithm. In *SIGIR*, 73–82. ACM.

Zhou, K.; Yang, S.-H.; and Zha, H. 2011. Functional matrix factorizations for cold-start recommendation. In *SIGIR*, 315–324. ACM.