Meta-learning Task-specific Regularization Weights for Few-shot Linear Regression

Tomoharu Iwata **Atsutoshi Kumagai** **Yasutoshi Ida**
NTT Corporation

Abstract

We propose a few-shot learning method for linear regression, which learns how to choose regularization weights from multiple tasks with different feature spaces, and uses the knowledge for unseen tasks. Linear regression is ubiquitous in a wide variety of fields. Although regularization weight tuning is crucial to performance, it is difficult when only a small amount of training data are available. In the proposed method, task-specific regularization weights are generated using a neural network-based model by taking a task-specific training dataset as input, where our model is shared across all tasks. For each task, linear coefficients are optimized by minimizing the squared loss with an L2 regularizer using the generated regularization weights and the training dataset. Our model is meta-learned by minimizing the expected test error of linear regression with the task-specific coefficients using various training datasets. In our experiments using synthetic and real-world datasets, we demonstrate the effectiveness of the proposed method on few-shot regression tasks compared with existing methods.

1 INTRODUCTION

Linear regression is used in many fields (Bzovsky et al., 2022; Jarantow et al., 2023; Yuan et al., 2020; Mu et al., 2020) since it is interpretable, it often performs well on small tabular data, and it is computationally efficient. In linear regression, the coefficients directly represent the relationship between feature and target variables, and the influence of each feature on each prediction is straightforwardly calculated. For linear regression, regularization is important to achieve a high generalization performance especially when the number of features is large compared with the number of training instances. For example, ridge regression (Hoerl and Kennard, 1970) uses a linear model with an L2 regularizer to shrink the coefficients towards zero, where a regularization weight is used to control the amount of regularization. The weight needs to be tuned appropriately for each task. For tuning the weight, cross-validation (Arlot and Celisse, 2010; Zhang and Yang, 2015) has been widely used. However, it is difficult in few-shot settings, i.e., only a small number of training instances are given. In many applications, enough training data can be unavailable in tasks of interest since collecting data requires high costs and is time-consuming.

In this paper, we propose a neural network-based method that meta-learns how to choose task-specific regularization weights using various datasets with different feature spaces for improving the generalization performance of few-shot linear regression. Since similar datasets would be likely to have similar appropriate regularization weights, we learn knowledge on their tuning that can be shared across datasets via meta-learning, and use it for unseen datasets. Our model, which is shared across all tasks, generates task-specific regularization weights given a small number of training data, called a support set, in each task. Then, task-specific coefficients are optimized by minimizing the squared error with an L2 regularizer using the generated regularization weights and support set. The task-shared model parameters are optimized by minimizing the expected test error with an episodic training framework (Snell et al., 2017; Finn et al., 2017). Figure 1 illustrates our meta-learning framework.

Our model generates a task-specific regularization weight for each feature, which enables us to regularize coefficients more flexibly than a scalar regularization weight shared for all features. To generate feature-specific regularization weights, we design our model

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

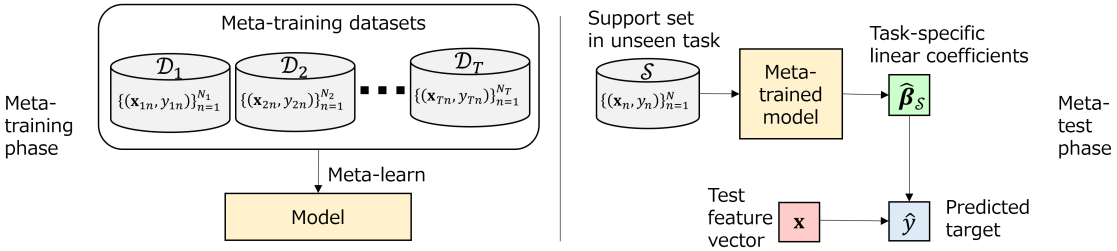


Figure 1: Our framework of meta-learning. In the meta-training phase, we meta-learn our model using meta-training datasets $\{\mathcal{D}_t\}_{t=1}^T$. In the meta-test phase, we generate task-specific linear coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ from support set $\mathcal{S}$ with a few labeled data in an unseen task using the meta-trained model. The coefficients are used for predicting target values. Feature spaces are different across datasets.

to have the permutation equivariant property (Zaheer et al., 2017); when input features are permuted, output regularization weights are also permuted accordingly. In particular, we use attention mechanisms (Vaswani et al., 2017; Iwata and Kumagai, 2023), where attention between instances and attention between feature and target values are alternately iterated. By the model, we can encode the information of the given support set considering relationship between feature and target values of all the instances, which is useful for finding appropriate task-specific regularization weights. Our model can handle datasets with different feature spaces, different numbers of features, and different numbers of instances, which enables us to meta-learn knowledge from a wide variety of datasets and apply it to tasks with new feature spaces that are unseen in the meta-training phase. Our meta-learning framework is formulated as a bilevel optimization, where task-shared model parameters are optimized by minimizing the expected test error in the outer optimization, while coefficients are adapted to a support set for each task in the inner optimization. Since our inner optimization problem can be solved in a closed form, we can perform the bilevel optimization efficiently by backpropagating the outer loss through the inner optimization to update the model parameters by stochastic gradient methods.

Our main contributions are summarized as follows: 1) We propose a method for generating task-specific regularization weights in linear regression by meta-learning from various datasets for the first time. 2) We develop a neural network-based model that generates task-specific linear coefficients given training data, which can handle data with different feature spaces. 3) Using synthetic and real-world datasets, we demonstrate that the proposed method achieves better generalization performance than existing meta-learning and cross-validation methods.

2 RELATED WORK

Regularization weights are usually tuned by minimizing the validation error (Franceschi et al., 2018; Ji et al., 2021; Bischl et al., 2023; Stephenson et al., 2021). For ridge regression, efficient algorithms for leave-one-out cross-validation have been developed (Patil et al., 2021). Although optimizing the cross-validation error consistently estimates the optimal regularization weight as the number of instances grows proportionally with the number of features (Patil et al., 2021; Hastie et al., 2022), it does not guarantee the optimality in the overparameterized setting, i.e., the number of features is larger than the number of instances. Asymptotic theoretical analysis for overparameterized ridge regression has been studied (Wu and Xu, 2020; Kobak et al., 2020; Hastie et al., 2022; Bartlett et al., 2020). However, it is inapplicable to the few-shot setting. Bayesian methods have also been proposed for tuning the weights (MacKay, 1994; Tipping, 2001; Tew et al., 2023). Although a unimodal posterior of a Bayesian formulation of ridge regression is proved for a large enough number of training data (Tew et al., 2023), regularization weight tuning is difficult with a small number of training data. Multi-task learning methods have been proposed that share regularizers across tasks for linear regression (Zhou et al., 2010; Hallac et al., 2015; Yamada et al., 2017; De Oliveira et al., 2019). However, they require datasets in target tasks for training and are inapplicable to unseen tasks. In addition, they require that the feature space is identical for all tasks. On the other hand, the proposed method does not need target tasks for meta-learning and can share knowledge among tasks with different feature spaces.

Many meta-learning, or few-shot learning, methods have been proposed (Schmidhuber, 1987; Bengio et al., 1991; Ravi and Larochelle, 2017; Andrychowicz et al., 2016; Vinyals et al., 2016; Snell et al., 2017; Finn et al., 2017; Oreshkin et al., 2018; Guo and Che-

ung, 2020), which include methods that can handle datasets with different feature spaces (Iwata and Kumagai, 2020; Zhu et al., 2023; Iwata and Kumagai, 2023). However, they use black-box models for prediction, and therefore, it is hard to interpret them (Rudin, 2019). On the other hand, the prediction with the proposed method is interpretable since it is based on linear models although neural networks are used for generating regularization weights. Several meta-learning methods for linear regression have been proposed (Rusu et al., 2019; Kong et al., 2020; Zhang et al., 2024; Thekumparampil et al., 2021b,a). Model-agnostic meta-learning (Finn et al., 2017) is applicable to linear regression. Adaptive learning of hyperparameters for fast adaptation (Baik et al., 2020) estimates L2 regularization weights in meta-learning. However, these methods assume an identical feature space across different tasks, and cannot meta-learn from tasks with heterogeneous feature spaces. Our model is related to encoder-decoder-based meta-learning (Xu et al., 2020), such as neural processes (Garnelo et al., 2018; Kim et al., 2019), where a support set is encoded and predictions are decoded by neural networks. Our model encodes a support set and decodes regularization weights by neural networks. The proposed method can perform the inner optimization in a closed form, unlike model-agnostic meta-learning (Finn et al., 2017), which requires iterative optimization procedures. Although ridge regression differentiable discriminator (Bertinetto et al., 2018) analytically obtains task-adapted coefficients for meta-learning, it uses ridge regression on encoded features by neural networks and is inapplicable to linear regression on original features.

3 PROPOSED METHOD

3.1 Problem formulation

In the meta-training phase, we are given meta-training datasets $\{\mathcal{D}_t\}_{t=1}^T$ from T tasks, where $\mathcal{D}_t = \{(\mathbf{x}_{tn}, y_{tn})\}_{n=1}^{N_t}$ is the labeled data of the tth task, $\mathbf{x}_{tn} \in \mathbb{R}_t^{M_t}$ is the feature vector of the nth instance in the tth task, $y_{tn} \in \mathbb{R}$ is its target value, and N_t is the number of instances. In the meta-test phase, we are given a small number of labeled data $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, which is the support set, from an unseen meta-test task that is different from but related to the meta-training tasks. Here, $\mathbf{x}_n \in \mathbb{R}^M$ and $y_n \in \mathbb{R}$. We are also given unlabeled data, which are called the query set, from the meta-test task. Our aim is to obtain a linear regression model that achieves high prediction performance on the query set in meta-test tasks. The feature space and number of features can be different across tasks.

3.2 Model

Our model takes support set $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ of a task as input, where $\mathbf{x}_n \in \mathbb{R}^M$ and $y_n \in \mathbb{R}$. The output of our model is task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}} \in \mathbb{R}^{M+1}$, which are used to predict the target value by $\hat{y} = \hat{\boldsymbol{\beta}}_{\mathcal{S}}^\top \mathbf{x}'$, where $'$ represents the concatenation of one for the bias term, i.e., $\mathbf{x}' = [\mathbf{x}^\top, 1]^\top \in \mathbb{R}^{M+1}$. The proposed model can handle support sets with different feature spaces, different numbers of features M and different numbers of instances N.

We consider desired properties of models for generating coefficients from a support set. First, the model needs to be permutation equivariant on features and permutation invariant on instances. Namely, when the input features are permuted, the output coefficients are also permuted accordingly; when the instances in the support set are permuted, the output coefficients are unchanged. This is because the order of the features and instances should not affect coefficients. Second, the model needs to consider all feature and target values in the support set since the characteristics of a coefficient depend on not only all values of the feature but also values of the other features and target.

We design our model to have these desired properties as follows. First, feature and target values are encoded into embedding vectors for each feature and for each target (Section 3.2.1). Second, the embedding vectors are transformed by alternately performing attention between instances and between feature and target variables and by position-wise neural networks (Section 3.2.2). Third, task-specific regularization weights $\boldsymbol{\lambda}$ are obtained from the transformed embedding vectors using permutation invariant neural networks (Section 3.2.3). Finally, task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ are analytically obtained by minimizing the squared error with the L2 regularization (Section 3.2.4). Algorithm 1 shows the forwarding procedure of our model, and Figure 2 illustrates it. Extensions of our model to multiple target variables, classification tasks, linear basis function models, and coefficient prior generation are described in Appendix A.

3.2.1 Initial encoding of feature and target values

First, we encode feature and target values for each instance, feature, and target using the following initial encoder,

$$\mathbf{z}_{0nm} = \begin{cases} f_{\mathrm{x}}(x_{nm}), & m = 1, \dots, M, \\ f_{\mathrm{y}}(y_n), & m = M+1, \end{cases} \tag{1}$$

where $f_{\mathrm{x}} : \mathbb{R} \to \mathbb{R}^{H_0}$ and $f_{\mathrm{y}} : \mathbb{R} \to \mathbb{R}^{H_0}$ are the feed-forward neural networks, $x_{nm} \in \mathbb{R}$ is the mth feature

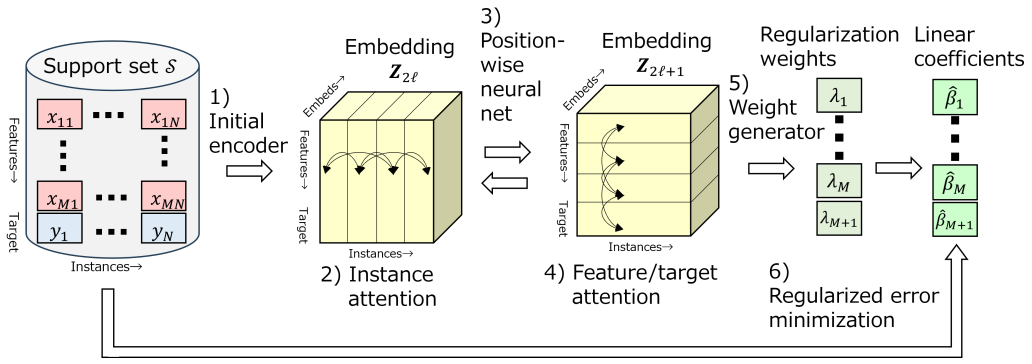


Figure 2: Our model for estimating task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ from support set $\mathcal{S}$.

Algorithm 1 Forwarding procedure of our model.

Input: Support set $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$.
Output: Task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ adapted to $\mathcal{S}$.

Calculate initial embedding vectors $\{\{\mathbf{z}_{0nm}\}_{n=1}^N\}_{m=1}^{M+1}$ from $\mathcal{S}$ by initial encoders in Eq. (1).
for $\ell = 1, \ldots, L$ **do**
  Calculate query $\{\{\mathbf{q}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$, key $\{\{\mathbf{k}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$, and value $\{\{\mathbf{v}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$ vectors by Eq. (2).
  if ℓ is an odd number **then**
    Calculate output tensor $\{\{\mathbf{o}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$ by performing attention across instances by Eqs. (3,5).
  else
    Calculate output tensor $\{\{\mathbf{o}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$ by performing attention across feature and target variables by Eqs. (4,5).
  end if
  Calculate embedding vectors $\{\{\mathbf{z}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$ using position-wise neural networks in Eq. (6).
end for
Calculate regularization weights $\boldsymbol{\lambda}$ using $\{\{\mathbf{z}_{Lnm}\}_{n=1}^N\}_{m=1}^{M+1}$ by Eqs. (7,8).
Calculate task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ using $\mathcal{S}$ and $\boldsymbol{\lambda}$ by regularized error minimization in Eq. (10) or Eq. (11).

of the nth instance, and $\mathbf{z}_{0nm} \in \mathbb{R}^{H_0}$ is an initial embedding vector. All features are encoded using the same neural network f_{x}. We use different neural networks for the features and the target since they have different effects on appropriate regularization weights.

3.2.2 Attention between instances and between feature and target variables

Let $\{\{\mathbf{z}_{\ell nm}\}_{n=1}^N\}_{m=1}^{M+1}$ be a set of embedding vectors at the ℓth layer. We transform embedding vectors by alternately performing attention across instances and attention across feature and target variables using multi-head variable-feature self-attention (MVSA) (Iwata and Kumagai, 2023). With MVSA, we can flexibly obtain embedding vectors that contain information about other instances and other feature and target variables while keeping the permutation equivariance.

In particular, in the ℓth layer, query $\mathbf{q}_\ell \in \mathbb{R}^{H_\ell^{\mathrm{K}}}$, key $\mathbf{k}_\ell \in \mathbb{R}^{H_\ell^{\mathrm{K}}}$, and value $\mathbf{v}_\ell \in \mathbb{R}^{H_\ell^{\mathrm{V}}}$ vectors are calculated for each instance $n = 1, \ldots, N$ and for each variable $m = 1, \ldots, M+1$ using embedding vectors at the previous layer by

$$\mathbf{q}_{\ell nm} = \mathbf{W}_\ell^{\mathrm{Q}} \mathbf{z}_{\ell-1,n,m}, \quad \mathbf{k}_{\ell nm} = \mathbf{W}_\ell^{\mathrm{K}} \mathbf{z}_{\ell-1,n,m}, \quad \mathbf{v}_{\ell nm} = \mathbf{W}_\ell^{\mathrm{V}} \mathbf{z}_{\ell-1,n,m}, \tag{2}$$

where $\mathbf{W}_\ell^{\mathrm{Q}} \in \mathbb{R}^{H_\ell^{\mathrm{K}} \times H_{\ell-1}}$, $\mathbf{W}_\ell^{\mathrm{K}} \in \mathbb{R}^{H_\ell^{\mathrm{K}} \times H_{\ell-1}}$, and $\mathbf{W}_\ell^{\mathrm{V}} \in \mathbb{R}^{H_\ell^{\mathrm{V}} \times H_{\ell-1}}$ are linear projection matrices.

Variable-feature self-attention (VSA) calculates attention weights by the dot-product between the query and keys, and updates the embedding tensor by aggregating values using the attention weights (Iwata and Kumagai, 2023). In the odd-numbered layers, the attention is performed across instances,

$$\mathrm{VSA}_{\ell nm} = \sum_{n'=1}^{N} \mathrm{softmax}\left(\frac{\mathbf{q}_{\ell n}^\top \mathbf{k}_{\ell n'}}{\sqrt{(M+1)H_\ell^{\mathrm{K}}}} \right) \mathbf{v}_{\ell n' m}, \tag{3}$$

where $\mathbf{q}_{\ell n} = [\mathbf{q}_{\ell n1}; \ldots; \mathbf{q}_{\ell,n,M+1}] \in \mathbb{R}^{(M+1)H_\ell^{\mathrm{K}}}$ and $\mathbf{k}_{\ell n'} = [\mathbf{k}_{\ell n'1}; \ldots; \mathbf{k}_{\ell,n',M+1}] \in \mathbb{R}^{(M+1)H_\ell^{\mathrm{K}}}$ are query and key vectors concatenated for all feature and target variables for the nth and n'th instances, softmax is the softmax function normalized over instances n', and $\mathrm{VSA}_{\ell nm} \in \mathbb{R}^{H_\ell^{\mathrm{V}}}$. By Eq. (3), embedding vectors are updated using embedding vectors of other instances. In the even-numbered layers, the attention is

performed across variables,

$$\mathrm{VSA}_{\ell nm} = \sum_{m'=1}^{M+1} \mathrm{softmax}\left(\frac{\mathbf{q}_{\ell m}^\top \mathbf{k}_{\ell m'}}{\sqrt{N H_\ell^{\mathrm{K}}}}\right) \mathbf{v}_{\ell nm'}, \quad (4)$$

where $\mathbf{q}_{\ell m} = [\mathbf{q}_{\ell 1m}; \ldots; \mathbf{q}_{\ell Nm}] \in \mathbb{R}^{NH_\ell^{\mathrm{K}}}$ and $\mathbf{k}_{\ell m'} = [\mathbf{k}_{\ell 1m'}; \ldots; \mathbf{k}_{\ell Nm'}] \in \mathbb{R}^{NH_\ell^{\mathrm{K}}}$ are query and key vectors concatenated for all instances for the mth and m'th variables, softmax is the softmax function normalized over variables m', and $\mathrm{VSA}_{\ell nm} \in \mathbb{R}^{H_\ell^{\mathrm{V}}}$. By Eq. (4), embedding vectors are updated using embedding vectors of other variables.

Multi-head variable-feature self-attention (MVSA) concatenates a set of VSAs by

$$\mathbf{o}_{\ell nm} = \mathbf{W}_\ell^{\mathrm{O}} [\mathrm{VSA}_{\ell nm1}; \ldots; \mathrm{VSA}_{\ell nmR}] \in \mathbb{R}^{H_\ell^{\mathrm{O}}}, \quad (5)$$

where R is the number of heads, $\mathrm{VSA}_{\ell nmr} \in \mathbb{R}^{H_\ell^{\mathrm{V}}}$ is the rth VSA, linear projection matrices are different across heads, $[\mathrm{VSA}_{\ell nm1}; \ldots; \mathrm{VSA}_{\ell nmR}] \in \mathbb{R}^{RH_\ell^{\mathrm{V}}}$ is their concatenation, and $\mathbf{W}_\ell^{\mathrm{O}} \in \mathbb{R}^{H_\ell^{\mathrm{O}} \times RH_\ell^{\mathrm{V}}}$ is a linear projection matrix.

As in Transformers (Vaswani et al., 2017), a position-wise feed-forward neural network is applied for each instance $n = 1, \ldots, N$ and for each variable $m = 1, \ldots, M+1$ after each of the MVSA layers to increase expressiveness,

$$\mathbf{z}_{\ell nm} = \mathbf{W}_\ell^{\mathrm{R}} \mathbf{o}_{\ell nm} + f_\ell(\mathrm{LN}_\ell(\mathbf{o}_{\ell nm})), \quad (6)$$

where $\mathbf{W}_\ell^{\mathrm{R}} \in \mathbb{R}^{H_\ell \times H_\ell^{\mathrm{O}}}$ is a linear projection matrix, $f_\ell : \mathbb{R}^{H_\ell^{\mathrm{O}}} \to \mathbb{R}^{H_\ell}$ is a feed-forward neural network, and LN_ℓ is a layer normalization.

3.2.3 Regularization weight generators

After L layers of MVSA and position-wise neural networks, final embedding vectors $\{\{\mathbf{z}_{Lnm}\}_{n=1}^N\}_{m=1}^{M+1}$ are obtained, where $\mathbf{z}_{Lnm} \in \mathbb{R}^{H_L}$. Using the final embedding vectors, we generate regularization weights for each feature, $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_{M+1}]^\top \in \mathbb{R}_{>0}^{M+1}$. With the feature-specific regularization weights, we can regularize coefficients flexibly (Hoerl and Kennard, 1970; Maruyama and Strawderman, 2005; Mori and Suzuki, 2018; Wu and Xu, 2020). Regularization weight $\lambda_m \in \mathbb{R}_{>0}$ for the mth feature is calculated from final embedding vectors $\{\mathbf{z}_{Lnm}\}_{n=1}^N$ of the feature by aggregating them using average pooling,

$$\lambda_m = f_{\mathrm{M}}\left(\frac{1}{N}\sum_{n=1}^N \mathbf{z}_{Lnm}\right), \quad m = 1, \ldots, M, \quad (7)$$

where $f_{\mathrm{M}} : \mathbb{R}^{H_L} \to \mathbb{R}_{>0}$ is a feed-forward neural network. Eq. (7) is a permutation invariant neural network over instances, which does not change the output with the permutation of instances, due to the permutation invariant operation of average pooling (Zaheer et al., 2017). Regularization weight λ_{M+1} for the bias term is calculated using a set of all final embedding vectors $\{\{\mathbf{z}_{Lnm}\}_{n=1}^N\}_{m=1}^{M+1}$ since the bias term depends on all of the feature and target variables,

$$\lambda_{M+1} = f_{\mathrm{B2}}\left(\frac{1}{M+1}\sum_{m=1}^{M+1} f_{\mathrm{B1}}\left(\frac{1}{N}\sum_{n=1}^N \mathbf{z}_{Lnm}\right)\right), \quad (8)$$

where $f_{\mathrm{B1}} : \mathbb{R}^{H_L} \to \mathbb{R}^H$ and $f_{\mathrm{B2}} : \mathbb{R}^H \to \mathbb{R}_{>0}$ are feed-forward neural networks. Eq. (8) is a permutation invariant neural network over instances and variables. To make λ_m be always positive, we use the exponential function at the last layer in f_{M} and f_{B2}.

3.2.4 Task-specific coefficient estimation by regularized error minimization

Using regularization weights $\boldsymbol{\lambda}$ obtained by Eqs. (7,8), task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ are estimated by minimizing the squared error on support set $\mathcal{S}$ with a weighted L2 regularizer,

$$\hat{\boldsymbol{\beta}}_{\mathcal{S}} = \underset{\boldsymbol{\beta}}{\mathrm{argmin}} \frac{1}{N} \parallel \mathbf{y} - \mathbf{X}'\boldsymbol{\beta} \parallel^2 + \boldsymbol{\beta}^\top \mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{\beta}, \quad (9)$$

where $\mathrm{diag}(\boldsymbol{\lambda}) \in \mathbb{R}^{(M+1)\times(M+1)}$ is the diagonal matrix with the elements of $\boldsymbol{\lambda}$ as the diagonal, $\mathbf{y} = [y_1, \ldots, y_N]^\top \in \mathbb{R}^N$ is the vector of the support target values, and $\mathbf{X}' = [\mathbf{x}'_1; \ldots; \mathbf{x}'_N]^\top \in \mathbb{R}^{N\times(M+1)}$ is the matrix of the support feature vectors.

Eq. (9) can be solved in a closed form by

$$\hat{\boldsymbol{\beta}}_{\mathcal{S}} = \left(\mathrm{diag}(\boldsymbol{\lambda}) + \mathbf{X}'^\top \mathbf{X}'\right)^{-1} \mathbf{X}'^\top \mathbf{y}. \quad (10)$$

Since it requires the inverse of a matrix of size $(M+1)\times(M+1)$, its computational complexity is $O(M^3)$. When number of features M is larger than number of instances N, we can reduce the complexity by rewriting Eq. (10) by the Woodbury formula,

$$\hat{\boldsymbol{\beta}}_{\mathcal{S}} = \mathrm{diag}(\boldsymbol{\lambda}^{-1})\mathbf{X}'^\top \left(\mathbf{I} + \mathbf{X}'\mathrm{diag}(\boldsymbol{\lambda}^{-1})\mathbf{X}'^\top\right)^{-1} \mathbf{y}, \quad (11)$$

where its complexity is $O(N^3)$. Since we consider few-shot settings, we use Eq. (11). All of the operations in our model in Eqs. (1–11) are permutation equivariant on features, and permutation invariant on instances.

3.3 Meta-learning

Let $\boldsymbol{\theta}$ be our model parameters, which consist of parameters in neural networks f_{x}, f_{y}, f_ℓ for each layer $\ell = 1, \ldots, L$, f_{M}, f_{B1}, and f_{B2}, linear projection matrices $\mathbf{W}_\ell^{\mathrm{Q}}$, $\mathbf{W}_\ell^{\mathrm{K}}$, $\mathbf{W}_\ell^{\mathrm{V}}$, $\mathbf{W}_\ell^{\mathrm{O}}$, and $\mathbf{W}_\ell^{\mathrm{R}}$ for each layer, and

parameters in layer normalization LN_ℓ for each layer. Since model parameters $\boldsymbol{\theta}$ are shared across different datasets, we can store common knowledge for generating regularization weights that are useful in various datasets. To make our model generate appropriate tasks-specific regularization weights that achieve high generalization performance in linear regression, we train model parameters $\boldsymbol{\theta}$ by minimizing the following expected test squared error

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}_t \left[\mathbb{E}_{(\mathcal{S},\mathcal{Q})\sim\mathcal{D}_t} \left[\frac{1}{|\mathcal{Q}|} \sum_{(\mathbf{x},y)\in\mathcal{Q}} \| \mathbf{y} - \hat{\boldsymbol{\beta}}_{\mathcal{S}}^\top \mathbf{x}' \|^2 \right] \right], \tag{12}$$

where $\mathbb{E}_t$ is the expectation over meta-training tasks, $\mathcal{Q} \subset \mathcal{D}_t$ is a query set that is a subset of meta-training dataset $\mathcal{D}_t$ and does not overlap with support set $\mathcal{Q} \cap \mathcal{S} = \emptyset$, $\mathbb{E}_{(\mathcal{S},\mathcal{Q})\sim\mathcal{D}_t}$ is the expectation over support and query sets generated from $\mathcal{D}_t$, and $|\mathcal{Q}|$ is the query set size. With the squared errors of the query set that are different from the support set used for estimating the coefficients, we can evaluate the generalization error. Eq. (12) can improve the generalization performance in the meta-test phase under the assumption that the task distributions are identical between the meta-training and meta-test phases.

Algorithm 2 shows the meta-training procedure of the proposed model. The expectation in Eq. (12) is approximated with the Monte Carlo method, where tasks, support sets, and query sets are randomly sampled from the given meta-training datasets at Lines 3 and 4. By generating various tasks from the meta-training datasets, we can expect that our model can estimate coefficients properly for unseen tasks. Line 5 corresponds to the inner optimization of estimating task-specific coefficients with regularized training error minimization. Line 7 corresponds to the outer optimization.

Algorithm 2 Meta-learning procedure of our model.

Input: Meta-training data $\{\mathcal{D}_t\}_{t=1}^T$, support set size N^{S}, query set size N^{Q}.
Output: Trained model parameters $\boldsymbol{\theta}$.
1: Initialize model parameters $\boldsymbol{\theta}$.
2: **while** End condition is satisfied **do**
3: Randomly select task t from $\{1, \cdots, T\}$.
4: Randomly sample support set $\mathcal{S}$ of size N^{S} and query set $\mathcal{Q}$ of size N^{Q} from meta-training dataset $\mathcal{D}_t$, where $\mathcal{S}, \mathcal{Q} \subset \mathcal{D}_t$, and $\mathcal{S} \cap \mathcal{Q} = \emptyset$.
5: Optimize task-specific coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ using Algorithm 1.
6: Compute the mean squared error on the query set using the estimated task-specific coefficients, $\frac{1}{|\mathcal{Q}|} \sum_{(\mathbf{x},y)\in\mathcal{Q}} \| \mathbf{y} - \hat{\boldsymbol{\beta}}_{\mathcal{S}}^\top \mathbf{x}' \|^2$.
7: Update model parameters $\boldsymbol{\theta}$ using a stochastic gradient method.
8: **end while**

4 EXPERIMENTS

4.1 Synthetic data

4.1.1 Data

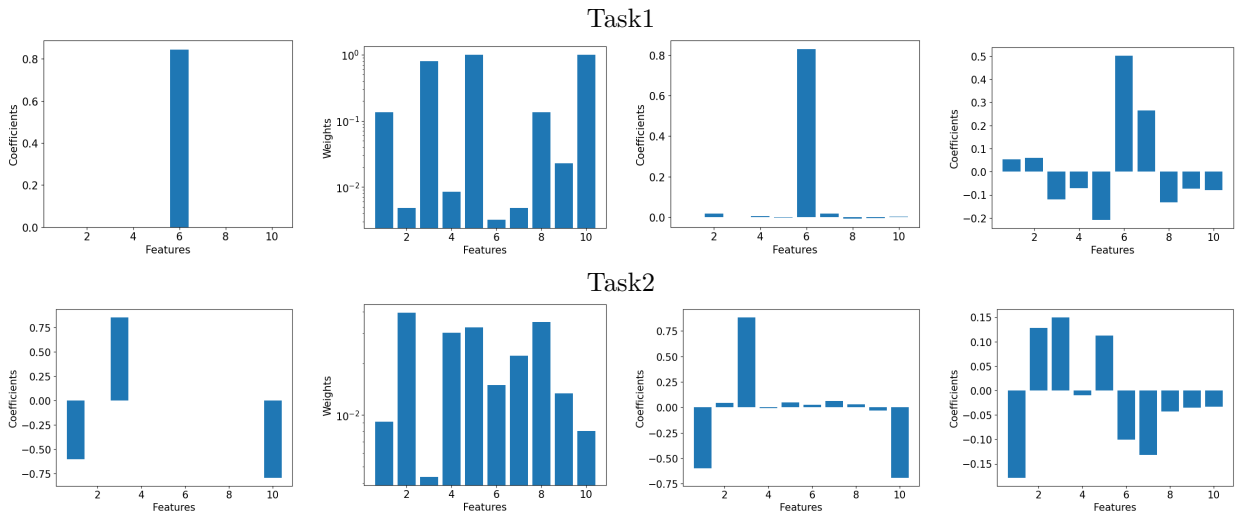
We first evaluated the proposed method on synthetic tasks. For each task, data were generated from one of three classes, $p(\mathcal{D}_t) = \sum_{k=1}^K p(k) \prod_{n=1}^{N_t} \mathcal{N}(\mathbf{x}_{tn}|\boldsymbol{\mu}_k, \mathbf{I}) \mathcal{N}(y_{tn}|\boldsymbol{\beta}_k^\top \mathbf{x}, 10^{-3})$, where $K = 3$, $p(k) = \frac{1}{K}$, $\mathcal{N}(\cdot|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is Gaussian with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}_k$ is the mean of the kth class, and $\boldsymbol{\beta}_k$ is coefficients of the kth class. A class was randomly assigned to each task. The number of features was ten for all tasks. Coefficients $\boldsymbol{\beta}_k$ were sparse, where only $k \in \{1, 2, 3\}$ elements had non-zero values for a task with the kth class. The non-zero values in coefficients and mean $\boldsymbol{\mu}_k$ were generated uniform randomly in $[-1, 1]$. Linear coefficients $\boldsymbol{\beta}_k$ and feature vector $\mathbf{x}$ were randomly permuted over features for each task to have different feature spaces aross tasks. The meta-training, meta-validation, and meta-test tasks were 70, 10, and 20, respectively. The number of instances per meta-training task was 100. The support set size was $N^{\mathrm{S}} = 5$, and the query set size was $N^{\mathrm{Q}} = 10$.

4.1.2 Settings

For all neural networks f_{x}, f_{y}, f_ℓ for $\ell = 1, \ldots, L$, f_{M}, f_{B1}, and f_{B2} in our model, we used three-layered feed-forward neural networks with 128 hidden units and leaky rectifier linear unit activation functions. A part of initial encoders f_{x} and f_{y} were shared, where the difference was modeled by inputting a binary value as follows, $f_{\mathrm{x}} = f_0(\cdot, 0)$, $f_{\mathrm{y}} = f_0(\cdot, 1)$, and $f_0 : \mathbb{R}^2 \to \mathbb{R}^{H_0}$. All of the embedding dimensions were set to $H_0 = H_\ell = H_\ell^{\mathrm{K}} = H_\ell^{\mathrm{V}} = H_\ell^{\mathrm{O}} = H = 128$. We used MVSA layers with $R = 4$ heads and $L = 6$ layers. For meta-learning, we used Adam (Kingma and Ba, 2015) with learning rate 10^{-4}, and a batch size of 32 tasks. The maximum number of meta-training epochs was 100 for synthetic data and 5,000 for OpenML data, and the meta-validation datasets were used for early stopping. The code of the proposed method is available at `https://www.kecl.ntt.co.jp/as/members/iwata/meta_linear.html`.

4.1.3 Results

Figure 3 shows examples of the estimated regularization weights and coefficients by the proposed method,



(a) True $\boldsymbol{\beta}$ (b) $\boldsymbol{\lambda}$ by ours (c) $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ by ours (d) $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ by ridge

Figure 3: Examples of (a) true coefficients $\boldsymbol{\beta}$ in synthetic data, (b) estimated regularization weights $\boldsymbol{\lambda}$ by the proposed method, (c) estimated coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ by the proposed method, and (d) estimated coefficients $\hat{\boldsymbol{\beta}}_{\mathcal{S}}$ by ridge regression in two meta-test tasks, where the Task1's class is $k = 1$, and the Task2's class is $k = 3$. The horizontal axis is the feature index.

and the estimated coefficients by ridge regression with a regularization weight tuned with leave-one-out cross-validation for each task. The estimated regularization weights by the proposed method for non-zero true coefficients were relatively small, e.g., the sixth feature in Task1. The average of the estimated regularization weights for non-zero true coefficients was 0.018, and that for zero true coefficients was 0.264. This result suggests that the proposed method can properly estimate regularization weights using meta-learned knowledge. The estimated regularization weights were different across tasks, which demonstrates that the proposed method can flexibly determine the weights depending on a few observed data. Ridge regression uses a single regularization weight shared for all features, and the weight is estimated using only the support set based on cross-validation. On the other hand, the proposed method uses different weights across features, and the task-specific weights are estimated using the support set and meta-learned knowledge. Therefore, the estimated coefficients by the proposed method were closer to the true coefficients than those by ridge regression. From the estimated coefficients as shown in Figure 3(c), we can directly understand the influence of each feature on prediction, which is unavailable for nonlinear regression models. The performance was evaluated by the test mean squared error on the meta-test datasets averaged over the ten experiments with different dataset splits. The test mean squared error by the proposed method was 0.355 with standard error 0.014, and that by ridge regression was 0.922 with standard error 0.022. The proposed method achieved better prediction performance than ridge regression.

4.2 OpenML Data

4.2.1 Data

We next evaluated the proposed method using 35 datasets for regression in OpenML-CTR23 (Fischer et al., 2023), which is a curated tabular regression benchmarking suite. The number of instances ranges from 517 to 72,000, and the number of features ranges from five to 190 after onehot encoding. The statistics of the datasets are shown in Table 3 in Appendix B.1. Missing values were imputed using the mean for numerical features, and using the most frequent value for categorical features. Numerical features were normalized with zero mean and one standard deviation, and categorical features were represented by binary vectors with onehot encoding. For each experiment, we randomly selected 21 datasets for meta-training, seven datasets for meta-validation, and the remaining seven datasets for meta-test. In the proposed method, a single model was meta-trained using 21 datasets. There was no overlap of data between meta-training, meta-validation, and meta-test datasets. Therefore, the distribution of the meta-training tasks differed from that of the meta-test tasks, corresponding to an out-of-distribution setting. We performed 20 experiments with different dataset splits. For each of the meta-validation and meta-test datasets, we randomly generated ten tasks with different support and query sets. The support set size was $N^{\mathrm{S}} = \{3, 5, 10\}$, and the

query set size was $N^{\mathrm{Q}} = 10$.

4.2.2 Compared methods

We compared the proposed method (Ours) with the ridge regression differentiable discriminator (R2D2) (Bertinetto et al., 2018), meta-learning Gaussian processes with deep kernels (MGP) (Iwata and Tanaka, 2022), model-agnostic meta-learning (MAML) (Finn et al., 2017), meta-learn from heterogeneous feature spaces (MLH) (Iwata and Kumagai, 2020), hypernetworks (Hyper) (Ha et al., 2016), meta-learning the shared regularization weights (Shared), ridge regression with a regularization weight tuned with leave-one-out cross-validation (Ridge), Lasso linear regression with cross-validation (Lasso), elastic net linear regression with cross-validation (Elastic), linear models with automatic relevance determination (ARD) (MacKay, 1994), Bayesian linear models (BLM) (Tipping, 2001), ridge regression that jointly learns coefficients and regularization weights within an iterative expectation maximization (REM) (Tew et al., 2023), decision trees (DT), gradient boosting (GB) (Ke et al., 2017), and multi-layer perceptron regressor (MLP).

The proposed method, R2D2, MGP, MAML, MLH, Hyper, and Shared are meta-learning methods, which were trained using meta-training datasets by minimizing the expected test error. The other methods, i.e., Ridge, Lasso, Elastic, ARD, BLM, REM, DT, GB, and MLP, are standard machine learning methods, which were trained using the support set for each task. Ridge, Lasso, and Elastic are cross-validation-based methods. ARD, BLM, and Ridge EM are Bayesian methods. The proposed method, Hyper, Shared, Rdige, Lasso, Elastic, ARD, BLM, and REM consider a linear regression model for each task. The details of the implementation of the compared methods are described in Appendix B.2. The settings of the proposed method are described in Section 4.1.2.

4.2.3 Results

The test mean squared errors on the meta-test datasets are shown in Table 1. The values in bold are not statistically different at 5% level from the best-performing method in each dataset by a Wilcoxon signed-rank test. The proposed method achieved the lowest test errors. With most of the methods, the error reasonably decreased as the support set size increased. Meta-learning methods (our method, MGP, MAML, MLH, Hyper, and Shared) outperformed single-task methods (Ridge, Lasso, Elastic, ARD, BLM, REM, DT, GB, and MLP). This result indicates that sharing knowledge across tasks is important, and it is difficult to perform well with only a few training instances. The errors of existing meta-learning methods, i.e., R2D2, MGP, MAML, and MLH, were higher than those of the proposed method. This result implies that considering linear models is effective for few-shot regression. Hyper was worse than the proposed method because it is difficult for Hyper to directly generate coefficients using a forwarding pass of neural networks. On the other hand, the proposed method generates regularization weights by neural networks, and obtains coefficients by minimizing the regularized error using support sets. Therefore, the proposed method can perform well in at least areas close to the support instances. Since Shared used the same regularization weight for all tasks, its performance was worse than that of the proposed method. This result indicates the effectiveness of generating task-specific regularization weights with the proposed method. ARD is ridge regression with feature-specific regularization weights, which are optimized based on evidence maximization. Since obtaining appropriate feature-specific weights is difficult with a few training instances, ARD did not perform well.

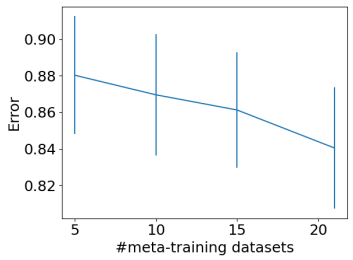


Figure 4: Average test mean squared error on OpenML data with support set size $N^{\mathrm{S}} = 5$ by our method with different numbers of meta-training datasets. The bars show the standard error.

Even when distributions between meta-training and meta-test tasks are different, if there exists a function that can output appropriate regularization weight given support set in both the meta-training and meta-test tasks, the proposed method can improve the meta-test performance as demonstrated by its better performance. The proposed method avoids the risk of losing important information during the encoding process by conducting linear regression using raw feature vectors instead of encoded vectors. This property alleviates the difficulty of meta-learning few-shot regression for tasks with different feature spaces.

Figure 4 shows that the test mean squared errors of the proposed method decreased as the number of meta-training datasets increased. With more meta-training datasets, tasks that resemble meta-test datasets are more likely to be included in meta-learning. The

Table 1: Average test mean squared errors on OpenML data and their standard errors with different support set sizes N^{S}.

N_{S}	3	5	10
Ours	**0.888 ± 0.033**	**0.840 ± 0.033**	**0.778 ± 0.036**
R2D2	1.025 ± 0.040	1.035 ± 0.048	1.050 ± 0.066
MGP	0.960 ± 0.029	0.930 ± 0.030	0.852 ± 0.032
MAML	1.066 ± 0.039	1.029 ± 0.036	0.954 ± 0.031
MLH	0.932 ± 0.032	0.897 ± 0.032	0.826 ± 0.033
Hyper	0.978 ± 0.038	0.913 ± 0.033	0.869 ± 0.039
Shared	0.922 ± 0.032	0.879 ± 0.032	0.861 ± 0.072
Ridge	1.277 ± 0.066	1.106 ± 0.040	0.913 ± 0.039
Lasso	1.535 ± 0.148	1.227 ± 0.044	1.064 ± 0.057
Elastic	1.375 ± 0.104	1.189 ± 0.043	0.998 ± 0.050
ARD	6.757 ± 1.009	1.911 ± 0.101	4.940 ± 3.233
BLM	1.404 ± 0.080	1.221 ± 0.040	0.993 ± 0.044
REM	1.278 ± 0.037	1.142 ± 0.034	0.960 ± 0.042
DT	1.710 ± 0.050	1.581 ± 0.047	1.385 ± 0.045
GB	1.326 ± 0.038	1.195 ± 0.032	1.091 ± 0.028
MLP	1.216 ± 0.043	1.199 ± 0.050	1.040 ± 0.045

Table 2: Ablation study. Average test mean squared errors on OpenML data with different support set sizes N^{S}. NoMVSA is our model that uses Transformer encoders instead of MVSA layers, where Transformer performs attention only across features. Singleλ is our model that generates a scalar feature-shared regularization weight instead of feature-specific regularization weights for each task, where Eq. (8) is used for the generation. NoTarget is our model that does not use target values $\{y_n\}_{n=1}^N$ in support set $\mathcal{S}$ for generating regularization weights. Prior is our model that generates Gaussian priors for coefficients instead of regularization weights as described in Appendix A.4.

N_{S}	3	5	10
Ours	**0.888 ± 0.033**	**0.840 ± 0.033**	**0.778 ± 0.036**
NoMVSA	0.922 ± 0.035	0.878 ± 0.033	0.793 ± 0.038
Singleλ	0.923 ± 0.031	0.879 ± 0.031	0.795 ± 0.035
NoTarget	0.920 ± 0.035	0.885 ± 0.031	0.796 ± 0.035
Prior	0.927 ± 0.034	0.885 ± 0.035	0.805 ± 0.036

proposed method learned how to set regularization weights from a wide variety of datasets, and adequately used the knowledge for improving the performance on meta-test datasets.

Table 2 shows the test mean squared errors in ablation study. The increases of the error by removing MVSAs indicates the effectiveness of attention across instances, features, and targets using MVSAs. The table also shows the effectiveness of generating regularization weights for each feature (Singleλ), and the effectiveness of the use of target values for encoding (NoTarget). When both the mean and precision of Gaussian priors for coefficients were generated (Prior), their errors were higher than the proposed method, which generates the prior precision (or regularization weights) and uses fixed zero mean. This is because generating an appropriate prior mean is difficult, and the zero mean is an effective inductive bias for linear regression as ridge regression has been successfully used in various domains. Additional experimental results are shown in Appendix B.3.

5 CONCLUSION

We proposed a few-shot learning method for linear regression. With the proposed method, regularization weights are generated by neural networks, which are shared across datasets with different feature spaces. The neural networks are meta-learned by minimizing the expected test error using various datasets. Although we believe that our approach is an important step for learning linear models with few data, we must extend our method in several directions. First, we will evaluate the extensions of the proposed method to classification tasks, multiple target variables, and linear basis function models. Second, we would like to apply our method to other regularizations than ridge, such as Lasso, elastic net, and group Lasso.

References

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

S. Baik, M. Choi, J. Choi, H. Kim, and K. M. Lee. Meta-learning with adaptive hyperparameters. *Advances in Neural Information Processing Systems*, 33:20755–20765, 2020.

P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48): 30063–30070, 2020.

Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *International Joint Conference on Neural Networks*, 1991.

L. Bertinetto, J. F. Henriques, P. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2018.

B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.

S. Bzovsky, M. R. Phillips, R. H. Guymer, C. C. Wykoff, L. Thabane, M. Bhandari, and V. Chaudhary. The clinicians guide to interpreting a regression analysis. *Eye*, 36(9):1715, 2022.

S. De Oliveira, A. R. Gonçalves, and F. Von Zuben. Group lasso with asymmetric structure estimation for multi-task learning. In *International Joint Conference on Artificial Intelligence*, pages 3202–3208, 2019.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135, 2017.

S. F. Fischer, M. Feurer, and B. Bischl. OpenML-CTR23–a curated tabular regression benchmarking suite. In *AutoML Conference*, 2023.

L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577, 2018.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1690–1699, 2018.

Y. Guo and N.-M. Cheung. Attentive weights generation for few shot learning via information maximization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13499–13508, 2020.

D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2016.

D. Hallac, J. Leskovec, and S. Boyd. Network Lasso: Clustering and optimization in large graphs. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 387–396, 2015.

T. Hastie, A. Montanari, S. Rosset, and R. J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *Annals of Statistics*, 50(2): 949, 2022.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

T. Iwata and A. Kumagai. Meta-learning from tasks with heterogeneous attribute spaces. *Advances in Neural Information Processing Systems*, 33:6053–6063, 2020.

T. Iwata and A. Kumagai. Meta-learning of semi-supervised learning from tasks with heterogeneous attribute spaces. *arXiv preprint arXiv:2311.05088*, 2023.

T. Iwata and Y. Tanaka. Few-shot learning for spatial regression via neural embedding-based Gaussian processes. *Machine Learning*, 111:1239–1257, 2022.

S. W. Jarantow, E. D. Pisors, and M. L. Chiu. Introduction to the use of linear and nonlinear regression analysis in quantitative biological assays. *Current Protocols*, 3(6):e801, 2023.

K. Ji, J. Yang, and Y. Liang. Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning*, pages 4882–4892, 2021.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

D. Kobak, J. Lomond, and B. Sanchez. The optimal ridge penalty for real-world high-dimensional data can be zero or negative due to the implicit ridge regularization. *Journal of Machine Learning Research*, 21(169):1–16, 2020.

W. Kong, R. Somani, Z. Song, S. Kakade, and S. Oh. Meta-learning for mixed linear regression. In *International Conference on Machine Learning*, pages 5394–5404, 2020.

D. J. MacKay. Bayesian nonlinear modeling for the prediction competition. *ASHRAE Transactions*, 100 (2):1053–1062, 1994.

Y. Maruyama and W. E. Strawderman. A new class of generalized Bayes minimax ridge regression estimators. *Annals of Statistics*, 33(4):1753–1770, 2005.

Y. Mori and T. Suzuki. Generalized ridge estimator and model selection criteria in multivariate linear regression. *Journal of Multivariate Analysis*, 165: 243–261, 2018.

K. Mu, Q. Shi, Y. Ma, and J. Tan. Exploration of entrepreneurship education by linear regression and psychological factor analysis. *Frontiers in Psychology*, 11:2045, 2020.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

B. Oreshkin, P. Rodríguez López, and A. Lacoste. TADAM: Task dependent adaptive metric for improved few-shot learning. *Advances in Neural Information Processing Systems*, 31, 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

P. Patil, Y. Wei, A. Rinaldo, and R. Tibshirani. Uniform consistency of cross-validation estimators for high-dimensional ridge regression. In *International Conference on Artificial Intelligence and Statistics*, pages 3178–3186, 2021.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5: 101–141, 2004.

C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

J. Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Master's thesis, Technische Universitat Munchen, Germany, 1987.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

W. Stephenson, Z. Frangella, M. Udell, and T. Broderick. Can we globally optimize cross-validation loss? quasiconvexity in ridge regression. *Advances in Neural Information Processing Systems*, 34:24352–24364, 2021.

S. Y. Tew, M. Boley, and D. Schmidt. Bayes beats cross validation: Efficient and accurate ridge regression via expectation maximization. *Advances in Neural Information Processing Systems*, 36, 2023.

K. K. Thekumparampil, P. Jain, P. Netrapalli, and S. Oh. Sample efficient linear meta-learning by alternating minimization. *arXiv preprint arXiv:2105.08306*, 2021a.

K. K. Thekumparampil, P. Jain, P. Netrapalli, and S. Oh. Statistically and computationally efficient linear meta-representation learning. *Advances in Neural Information Processing Systems*, 34:18487–18500, 2021b.

M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1(Jun):211–244, 2001.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

D. Wu and J. Xu. On the optimal weighted ℓ_2 regularization in overparameterized linear regression. *Advances in Neural Information Processing Systems*, 33:10112–10123, 2020.

J. Xu, J.-F. Ton, H. Kim, A. Kosiorek, and Y. W. Teh. Metafun: Meta-learning with iterative functional updates. In *International Conference on Machine Learning*, pages 10617–10627, 2020.

M. Yamada, T. Koh, T. Iwata, J. Shawe-Taylor, and S. Kaski. Localized lasso for high-dimensional regression. In *Artificial Intelligence and Statistics*, pages 325–333, 2017.

Y. Yuan, K. Dehghanpour, F. Bu, and Z. Wang. A data-driven customer segmentation strategy based on contribution to system peak demand. *IEEE Transactions on Power Systems*, 35(5):4026–4035, 2020.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

T. T. Zhang, L. F. Toso, J. Anderson, and N. Matni. Sample-efficient linear representation learning from non-iid non-isotropic data. In *International Conference on Learning Representations*, 2024.

Y. Zhang and Y. Yang. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112, 2015.

Y. Zhou, R. Jin, and S. C.-H. Hoi. Exclusive lasso for multi-task feature selection. In *International Conference on Artificial Intelligence and Statistics*, pages 988–995, 2010.

M. Zhu, K. Kobalczyk, A. Petrovic, M. Nikolic, M. van der Schaar, B. Delibasic, and P. Lio. Tabular few-shot generalization across heterogeneous feature spaces. *arXiv preprint arXiv:2311.10051*, 2023.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] It is described in Section 3.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] The analysis of the properties is described in Section 4, and the complexity is described in Section 3.2.4.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] The code of the proposed method is available at `https://www.kecl.ntt.co.jp/as/members/iwata/meta_linear.html`. The details of our experiments were described in Section 4 and Appendix B.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] The training details were described in Appendix B.2.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes] The definition, statistical test results, and error bars are described in Section 4 and Appendix B.3.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] It is described in Section 4.2.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A EXTENSIONS

A.1 Multiple target variables

In Section 3.2, we consider a single target variable. Our model can be extended to multiple target variables $\mathbf{y} \in \mathbb{R}^C$. Initial encoding, attention, and position-wise transformation are performed in the same way with the single target variable setting. We generate a regularization weight for the mth feature and the cth target using final embedding vectors of the feature $\{\mathbf{z}_{Lnm}\}_{n=1}^N$ and the target $\{\mathbf{z}_{L,n,M+1+c}\}_{n=1}^N$ by

$$\lambda_{mc} = f_{\mathrm{M2}}\left(\frac{1}{N}\sum_{n=1}^{N} f_{\mathrm{M1}}(\mathbf{z}_{Lnm}, \mathbf{z}_{L,n,M+1+c})\right), \tag{13}$$

where $f_{\mathrm{M1}} : \mathbb{R}^{2H_L} \to \mathbb{R}^H$ and $f_{\mathrm{M2}} : \mathbb{R}^H \to \mathbb{R}$ are feed-forward neural networks. A regularization weight for the bias term is calculated using a set of all final embedding vectors of all features and all targets,

$$\lambda_{M+1,c} = f_{\mathrm{B3}}\left(\frac{1}{M}\sum_{m=1}^{M} f_{\mathrm{B2}}\left(\frac{1}{N}\sum_{n=1}^{N} f_{\mathrm{B1}}(\mathbf{z}_{Lnm}, \mathbf{z}_{L,n,M+1+c})\right)\right), \tag{14}$$

where $f_{\mathrm{B1}} : \mathbb{R}^{H_L} \to \mathbb{R}^H$ $f_{\mathrm{B2}} : \mathbb{R}^H \to \mathbb{R}^H$, $f_{\mathrm{B3}} : \mathbb{R}^H \to \mathbb{R}$ are feed-forward neural networks.

A.2 Classification

The proposed method is also applicable to classification tasks by using least-squares classification (LSC) (Rifkin and Klautau, 2004; Rasmussen and Williams, 2005) for task-specific coefficient estimation at the inner optimization, where the classification task is treated as a regression one by setting target values $+1$ or -1 in binary classification. With LSC, no iterative procedures are needed. Another way is to use iterative solvers for the inner optimization. Our model is applicable as long as the operations are differentiable. When the iteratively reweighted least squares method (Murphy, 2012; Bertinetto et al., 2018) is used, the estimated coefficient at the ith iteration for binary target variable $y \in \{-1, 1\}$ is given by

$$\hat{\boldsymbol{\beta}}_{\mathcal{S}i} = \mathrm{diag}(\boldsymbol{\lambda}^{-1})\mathbf{X}'^{\top}\left(\mathrm{diag}(\mathbf{s}_i^{-1}) + \mathbf{X}'\mathrm{diag}(\boldsymbol{\lambda}^{-1})\mathbf{X}'^{\top}\right)^{-1}\mathbf{u}_i, \tag{15}$$

where $\mathbf{s}_i = \mathbf{m}_i(1-\mathbf{m}_i)$, $\mathbf{u}_i = \mathbf{X}'\hat{\boldsymbol{\beta}}_{\mathcal{S},i-1} + (\mathbf{y}-\mathbf{m}_i)/\mathbf{s}_i$, and $\mathbf{m}_i = \mathrm{sigmoid}(\mathbf{X}'\hat{\boldsymbol{\beta}}_{\mathcal{S},i-1})$. For updating model parameters in Eq. (12), a classification loss such as the cross-entropy loss can be used.

A.3 Linear basis function models

Instead of linear models with respect to original feature values, we can also consider linear models with respect to basis functions. Let $\bar{\mathbf{x}} \in \mathbb{R}^{\bar{M}}$ be an original feature vector, and $\{\phi_m\}_{m=1}^M$ be a set of basis functions, where $\phi_m : \mathbb{R}^{\bar{M}} \to \mathbb{R}$. By setting $x_m = \phi_m(\bar{\mathbf{x}})$ for $m = 1, \ldots, M$, the proposed method is directly applicable to linear basis function models. We can use arbitrary basis functions. For example, polynomial basis functions can enhance the expression power while keeping the interpretability. Parameters in basis functions can be trained by minimizing the expected test error in Eq. (12) by including the basis function parameters in model parameters $\boldsymbol{\theta}$.

A.4 Generating coefficient priors

The regularization weights can be seen as precision parameters of a Gaussian prior $\mathcal{N}(\boldsymbol{\beta}|\mathbf{0}, \mathrm{diag}(\boldsymbol{\lambda}^{-1}))$ for coefficients with the maximum a posteriori (MAP) estimation in a Bayesian framework, where the mean of the Gaussian prior is considered as zero. We can consider to generate a task-specific mean vector for the prior based on the proposed method. Let $\mathcal{N}(\boldsymbol{\beta}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\lambda}^{-1}))$ be the Gaussian prior with mean $\boldsymbol{\mu}$ and covariance $\mathrm{diag}(\boldsymbol{\lambda}^{-1})$. Mean vector $\boldsymbol{\mu}$ is generated in the same way with the regularization weights in Eqs. (7,8). The MAP estimation can be performed in a closed form by

$$\hat{\boldsymbol{\beta}}_{\mathcal{S}} = \boldsymbol{\mu} + \mathrm{diag}(\boldsymbol{\lambda}^{-1})\mathbf{X}'^{\top}\left(\mathbf{I} + \mathbf{X}'\mathrm{diag}(\boldsymbol{\lambda}^{-1})\mathbf{X}'^{\top}\right)^{-1}(\mathbf{y} - \mathbf{X}'\boldsymbol{\mu}). \tag{16}$$

Table 3: The number of instances and the number of features after onehot encoding for each dataset in OpenML data.

Name	#instances	#features
Moneyball	1232	72
QSAR-fish-toxicity	908	6
abalone	4177	10
airfoil-self-noise	1503	5
auction-verification	2043	16
brazilian-houses	10692	49
california-housing	20640	8
cars	804	17
concrete-compressive	1030	8
cps88wages	28155	12
cpu-activity	8192	21
diamonds	53940	26
energy-efficiency	768	8
fifa	19178	190
forest-fires	517	12
fps-benchmark	24624	125
geographical-origin	1059	116
grid-stability	10000	12
health-insurance	22272	25
kin8nm	8192	8
kings-county	21613	132
miami-housing	13932	15
naval-propulsion-plant	11934	14
physiochemical-protein	45730	9
pumadyn32nh	8192	32
red-wine	1599	11
sarcos	48933	21
socmob	1156	39
solar-flare	1066	29
space-ga	3107	6
student-performance-por	649	56
superconductivity	21263	81
video-transcoding	68784	24
wave-energy	72000	48
white-wine	4898	11

B EXPERIMENTS

B.1 Data

Table 3 shows the statistics of OpenML datasets used in our experiments.

B.2 Settings

In R2D2, MGP, and MAML, we used three-layered feed-forward neural networks with 128 hidden units for embedding each input feature. Then, the embedding vectors were encoded using Transformer (Vaswani et al., 2017) encoders with three layers, 128 hidden units, and four heads, which are permutation equivariant on features and can handle features with different sizes. In R2D2, target values were predicted by linear projection of the encoded vectors, where the linear projection was adapted to the support set for each task. In MGP, the encoded vectors were used for the input of a Gaussian process (Rasmussen and Williams, 2005) with RBF kernels, where the GP was adapted to the support set. In MAML, target values were predicted using a three-layered feed-forward neural network with 128 hidden units from the encoded vectors, where model parameters were adapted to the support set by gradient descent with learning rate 10^{-2} and five epochs. In MLH, feature and target embedding vectors were obtained by an inference network with three layers of deep sets (Zaheer et al., 2017), and then target values were predicted by a prediction network using the embedding vectors and an input feature vector. For each block of MLH, we used three-layered feed-forward neural networks with 128 hidden units. Hyper generated coefficients using the same neural networks as our model instead of generating regularization weights.

Table 4: Average test mean squared errors on synthetic data.

Ours	R2D2	MGP	MAML	MLH	Hyper	Shared	Ridge	Lasso	Elastic	ARD	BLM	REM	DT	GB	MLP
0.355	0.783	1.101	1.153	0.422	0.638	0.575	0.922	0.928	0.972	0.606	0.828	0.875	1.882	1.426	0.984

Table 5: Accuracy of selecting best regularization weights by cross-validation with ridge regression on OpenML data.

N_{S}	Accuracy
3	0.474 ± 0.016
5	0.445 ± 0.018
10	0.458 ± 0.022

Table 6: Average test mean squared errors on OpenML data by ridge regression with hyperparameter tuning based on leave-one-out cross-validation. Ridge selected a regularization weight from $\{10^{-1}, 1, 10\}$, which is the default setting in scikit-learn, and Ridge+ selected it from $10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3$.

N^{S}	Ridge	Ridge+
3	$\mathbf{1.277 \pm 0.066}$	1.342 ± 0.091
5	$\mathbf{1.106 \pm 0.040}$	1.162 ± 0.042
10	$\mathbf{0.913 \pm 0.039}$	0.994 ± 0.043

Table 7: Meta-training time in seconds on OpenML data with support set size $N^{\mathrm{S}} = 5$.

Ours	R2D2	MGP	MAML	MLH	Hyper	Shared
10120	11384	13238	86132	9970	11199	3111

Table 8: Meta-test time in seconds on OpenML data with support set size $N^{\mathrm{S}} = 5$.

Ours	R2D2	MGP	MAML	MLH	Hyper	Shared	Ridge	Lasso	Elastic	ARD	BLM	REM	DT	GB	MLP
0.715	0.597	0.973	6.125	0.671	0.789	0.259	0.337	4.264	3.902	1.024	0.495	1.529	0.185	0.467	2.155

Note that we newly developed this Hyper as a baseline since there have been no existing hypernetworks for linear regression with datasets with different feature spaces. Shared meta-learned scalar regularization weight $\lambda \in \mathbb{R}_{>0}$, which is shared across all tasks. We implemented the meta-learning methods with PyTorch (Paszke et al., 2019). For neural network-based compared methods, i.e., R2D2, MGP, MAML, MLH, and Hyper, we used the same meta-learning procedures as our method. We used scikit-learn (Pedregosa et al., 2011) for Ridge, Lasso, Elastic, ARD, BLM, DT, and MLP, used LightGBM (Ke et al., 2017) for GB with their default parameter settings, and used the authors' implementation for REM.

B.3 Results

Table 4 shows the test mean squared errors on synthetic data.

Table 5 shows the accuracy of selecting the best regularization weights by leave-one-out cross-validation (LOOCV) with ridge regression. Here, regularization weights were selected from $\lambda = \{10^{-1}, 1, 10\}$, and the accuracy is the probability that the regularization weight that minimized the squared error with LOOCV matched with the regularization weight that minimized squared error with the query set. All of the accuracies were low, and this result indicates that selecting appropriate regularization weights using LOOCV is difficult with few-shot settings. Table 6 shows the test mean squared error with different settings for regularization weight candidates. Even when the number of candidates increased, the performance did not increase in ridge regression. This result also indicates that tuning weights with a small number of data with LOOCV is difficult.

Tables 7 and 8 show the computational time for meta-training and meta-test on computers with NVIDIA GeForce GTX 1080 Ti GPU, Intel Xeon Gold 6126 CPU, and 128GB memory. The meta-training time of the proposed method was comparable to R2D2, MGP, and Hyper. Since MAML needs iterative optimization for each task, it took longer than the other meta-learning methods. Once our method is meta-trained, it can obtain coefficients efficiently as demonstrated in its short meta-test time.

Table 9: Average test mean squared errors on OpenML data with support set size $N^{\mathrm{S}} = 3$.

Dataset	Ours	R2D2	MGP	MAML	MLH	Hyper	Shared	Ridge	Lasso	Elastic	ARD	BLM	REM	DT	GB	MLP
Moneyball	0.843	**0.545**	0.756	0.875	0.665	0.820	0.909	0.936	1.011	1.032	9.183	1.011	**0.541**	1.432	1.222	0.958
QSAR-fish-toxicity	0.942	**0.744**	0.825	0.938	**0.708**	1.051	0.989	1.102	1.130	1.239	9.035	1.410	1.012	1.604	1.407	1.188
abalone	**0.987**	1.140	**0.978**	**1.025**	**1.020**	**0.993**	**0.973**	**1.667**	**2.413**	**2.695**	8.056	**2.452**	1.530	1.829	**1.170**	**1.055**
airfoil-self-noise	**0.946**	1.064	**0.952**	1.035	1.175	**0.906**	**0.983**	1.279	1.352	1.451	2.729	1.540	1.282	1.641	1.225	1.226
auction-verification	1.087	1.104	**0.969**	1.015	**0.927**	1.110	**1.037**	1.549	1.478	1.489	3.514	1.602	1.422	1.991	1.484	1.402
brazilian-houses	0.736	**0.061**	**0.062**	0.067	0.461	0.804	0.829	1.171	1.193	1.188	2.404	1.292	0.089	1.301	1.230	1.176
california-housing	1.023	1.143	1.075	1.119	1.000	1.083	1.097	**0.781**	**0.877**	**0.863**	2.882	**0.856**	1.907	1.454	1.044	**0.748**
cars	**0.659**	0.956	**0.886**	**1.065**	**0.871**	**0.699**	**0.658**	0.793	0.778	0.766	0.809	0.785	1.001	1.046	0.804	**0.735**
concrete-compressive	**0.659**	0.896	0.944	1.131	0.893	**0.680**	**0.652**	0.995	1.049	1.130	7.197	1.013	1.330	1.559	1.077	0.903
cps88wages	**0.902**	**0.919**	**0.753**	**0.752**	**0.758**	**0.942**	**0.930**	1.030	1.057	1.050	2.947	1.173	0.937	1.295	1.107	0.997
cpu-activity	0.934	1.237	1.351	1.406	1.535	1.032	0.976	**0.751**	0.770	0.918	2.603	**0.718**	1.423	0.779	0.841	**0.624**
diamonds	0.936	0.776	0.976	1.316	**0.465**	1.102	1.094	0.851	1.007	1.057	1.879	0.773	0.789	0.989	1.427	0.617
energy-efficiency	0.723	**0.375**	0.808	1.106	0.527	0.703	0.731	1.049	1.107	1.082	8.468	1.146	0.952	1.306	1.133	1.225
fifa	0.717	0.606	0.567	0.595	**0.460**	0.735	0.747	1.941	1.963	4.656	8.504	1.995	0.950	2.332	2.122	2.049
forest-fires	**1.736**	**2.060**	**2.076**	**2.041**	**2.160**	**1.797**	**1.771**	**0.969**	**1.017**	**1.010**	2.764	**1.015**	**2.309**	1.502	**0.963**	**1.010**
fps-benchmark	**0.698**	1.306	0.991	1.025	0.946	0.847	0.753	1.571	1.724	2.016	2.911	1.670	1.380	2.214	1.636	1.296
geographical-origin	0.897	1.098	**0.885**	1.214	**0.959**	0.869	**0.822**	2.882	3.421	3.783	10.483	2.485	1.651	1.968	1.391	2.388
grid-stability	**0.578**	0.776	0.942	0.744	0.803	**0.606**	**0.574**	1.156	1.233	1.393	1.805	1.252	1.396	1.699	1.433	0.971
health-insurance	**0.926**	1.242	1.028	1.063	1.067	1.048	**0.934**	1.451	1.611	1.519	2.570	1.624	1.479	2.192	1.549	1.368
kin8nm	1.137	**0.974**	**0.886**	1.045	1.171	1.277	**1.018**	1.776	2.024	2.375	2.837	1.979	1.323	2.259	2.053	1.663
kings-county	**0.894**	1.135	1.077	1.266	**0.871**	**1.215**	**0.924**	1.401	1.531	1.507	4.058	1.630	1.188	1.741	1.234	1.299
miami-housing	1.039	**0.721**	**0.723**	**0.753**	0.743	1.048	1.005	1.272	1.262	1.265	9.307	1.326	0.834	1.733	1.350	1.307
naval-propulsion-plant	**0.990**	2.446	1.118	1.151	**1.095**	1.048	1.014	1.351	1.400	1.357	18.257	1.372	1.587	2.109	1.285	1.677
physiochemical-protein	**1.419**	1.300	**1.152**	1.318	**1.119**	**1.598**	**1.705**	2.214	2.145	2.124	3.503	2.409	1.799	1.798	1.700	**1.414**
pumadyn32nh	0.748	1.057	0.947	1.323	0.980	0.915	**0.687**	1.246	1.241	1.393	2.364	1.346	1.393	1.623	1.551	1.166
red-wine	**0.745**	1.148	1.124	1.284	1.099	0.851	0.789	1.489	1.514	1.540	12.296	1.544	1.634	2.146	1.663	1.383
sarcos	**0.783**	1.079	1.052	1.163	1.014	**0.775**	**0.784**	1.127	1.155	1.269	9.232	1.171	1.113	1.686	1.141	1.151
socmob	0.774	0.784	0.867	1.177	**0.691**	0.888	0.924	1.488	1.665	1.761	3.817	1.975	0.873	1.973	1.482	1.419
solar-flare	**0.889**	**1.550**	**1.337**	**1.343**	**1.297**	**1.918**	0.944	**1.101**	1.200	1.259	27.373	1.239	1.737	1.378	1.223	1.163
space-ga	**0.991**	**1.032**	**0.967**	1.214	**1.024**	**1.014**	**1.043**	**1.240**	1.396	1.478	2.770	1.398	1.741	1.729	**1.294**	**0.994**
student-performance-por	**0.888**	1.294	1.049	1.052	1.047	0.997	1.083	1.343	1.342	1.344	4.230	1.417	1.623	2.089	1.379	1.280
superconductivity	**1.142**	**0.592**	0.803	0.978	**0.613**	**1.156**	**1.203**	**0.686**	**0.783**	**0.818**	**0.846**	**0.585**	1.226	1.089	1.407	**0.753**
video-transcoding	0.849	**0.759**	**0.760**	0.789	**0.738**	0.888	0.900	0.952	1.023	1.030	3.462	0.986	1.000	1.635	1.086	1.092
wave-energy	**0.692**	0.978	1.118	1.272	0.945	**0.719**	0.748	1.002	1.069	1.084	7.989	1.023	1.415	1.454	1.108	1.120
white-wine	**0.941**	1.144	**0.939**	**0.941**	**1.008**	**0.976**	**0.960**	**1.063**	1.069	1.085	21.390	**1.463**	**1.462**	**1.479**	1.095	**1.088**

Table 10: Average test mean squared errors on OpenML data with support set size $N^{\mathrm{S}} = 5$.

Dataset	Ours	R2D2	MGP	MAML	MLH	Hyper	Shared	Ridge	Lasso	Elastic	ARD	BLM	REM	DT	GB	MLP
Moneyball	0.758	0.517	0.641	0.751	0.574	0.782	0.815	0.788	0.863	0.892	1.218	0.785	**0.340**	1.195	1.128	0.808
QSAR-fish-toxicity	**0.844**	**0.753**	0.765	0.846	**0.657**	0.952	0.887	1.024	1.090	1.118	1.608	1.090	1.016	1.368	1.229	1.238
abalone	**0.940**	1.200	**0.978**	**0.994**	**0.945**	1.000	**0.967**	**1.052**	**1.181**	**1.291**	**1.619**	**1.065**	1.313	2.083	**1.148**	**0.993**
airfoil-self-noise	**0.855**	**1.140**	**0.872**	1.084	1.099	**0.912**	0.919	1.212	1.206	1.324	2.197	1.488	1.239	1.497	1.130	1.038
auction-verification	1.004	1.110	0.940	1.092	**0.870**	1.014	**0.970**	1.437	1.497	1.156	1.894	1.579	1.345	1.709	1.384	1.396
brazilian-houses	0.737	**0.054**	**0.056**	0.062	0.168	0.785	0.709	0.988	1.079	1.096	1.104	1.009	0.072	1.074	1.182	1.173
california-housing	0.921	1.276	1.064	1.108	0.956	1.042	1.002	**0.747**	**0.753**	**0.930**	**0.748**	**0.780**	1.513	1.076	0.901	**0.691**
cars	0.644	**0.759**	**0.791**	**0.886**	**0.757**	**0.614**	**0.644**	0.706	0.738	**0.716**	0.863	0.775	**0.694**	1.275	0.725	0.734
concrete-compressive	**0.613**	0.849	0.913	1.118	0.851	**0.616**	0.628	0.979	1.055	1.082	2.738	1.238	1.387	1.306	0.991	0.862
cps88wages	**0.831**	1.297	**0.759**	**0.767**	**0.774**	**0.929**	**0.878**	1.301	1.403	1.438	2.956	1.424	1.220	1.785	1.036	1.128
cpu-activity	0.850	1.321	1.371	1.539	1.532	0.975	0.946	**0.544**	0.721	0.713	1.595	0.614	1.424	0.775	0.842	**0.810**
diamonds	0.865	0.635	0.803	0.859	**0.359**	1.101	1.046	0.725	0.817	0.828	0.775	0.739	0.735	0.998	1.324	0.616
energy-efficiency	0.704	**0.264**	0.694	1.030	0.564	0.648	0.694	1.011	0.984	0.975	2.488	1.171	0.563	1.237	1.042	1.214
fifa	0.743	0.538	0.538	**0.693**	**0.470**	0.736	0.760	1.713	1.834	1.916	2.327	1.770	0.858	2.265	1.965	2.058
forest-fires	**1.754**	**2.091**	**2.083**	**2.090**	**2.112**	**1.778**	**1.810**	**0.894**	**0.872**	**0.910**	1.933	1.116	**2.199**	1.492	**0.867**	1.164
fps-benchmark	**0.760**	1.261	1.004	1.034	0.974	**0.782**	**0.753**	**1.276**	**1.305**	**1.351**	1.532	**1.388**	1.362	1.807	1.409	1.474
geographical-origin	0.841	**1.023**	**0.896**	1.218	0.948	0.866	**0.795**	1.577	1.623	1.534	2.160	1.426	1.480	1.800	1.263	2.409
grid-stability	**0.514**	0.789	0.895	0.704	0.898	**0.600**	0.538	0.737	0.885	0.911	1.581	0.718	1.175	1.541	1.275	0.820
health-insurance	**0.840**	1.189	1.013	1.111	1.076	1.037	0.890	1.222	1.319	1.349	3.777	1.340	1.351	1.993	1.365	1.220
kin8nm	1.043	1.039	**0.848**	1.007	1.035	1.138	**0.912**	1.360	1.399	1.172	1.603	1.649	1.327	1.712	**1.503**	1.478
kings-county	**0.879**	1.071	1.072	1.191	0.880	**0.857**	**0.877**	1.040	1.127	1.219	1.848	1.090	0.995	1.772	1.146	1.219
miami-housing	0.941	**0.724**	**0.687**	0.786	**0.675**	1.117	0.947	1.327	1.372	1.531	2.100	1.549	**0.789**	1.969	1.285	1.303
naval-propulsion-plant	**0.971**	2.514	1.188	1.198	**1.096**	1.089	1.020	1.362	1.638	1.754	2.225	1.266	1.250	1.643	1.144	2.082
physiochemical-protein	**1.197**	**1.897**	**1.257**	**1.241**	**1.356**	1.311	1.602	**1.361**	**2.133**	**2.089**	2.351	**2.005**	**1.953**	**1.488**	**1.442**	**1.579**
pumadyn32nh	**0.644**	1.108	0.956	1.110	1.008	0.778	**0.599**	0.986	1.026	1.045	1.600	1.161	1.244	1.389	1.253	1.153
red-wine	**0.766**	1.202	1.094	1.224	1.059	0.856	**0.750**	1.235	1.338	1.359	2.088	1.364	1.446	1.848	1.431	1.478
sarcos	**0.658**	1.080	1.048	1.131	0.914	**0.684**	0.714	1.044	1.101	1.158	1.443	1.125	1.040	1.780	1.033	1.120
socmob	0.814	**0.695**	0.759	0.837	**0.722**	0.936	0.937	1.324	1.597	1.738	2.282	1.670	**0.783**	1.699	1.330	1.197
solar-flare	**0.905**	1.564	**1.279**	**1.393**	**1.250**	**0.954**	**0.901**	**1.244**	1.260	1.326	1.648	1.491	**1.483**	1.651	1.121	**0.994**
space-ga	**0.998**	**1.068**	**0.931**	1.199	**0.964**	**1.067**	**1.038**	**1.158**	**1.101**	**1.166**	**1.382**	**1.086**	1.232	1.565	**1.131**	**1.104**
student-performance-por	**0.867**	1.206	1.028	1.046	1.020	0.998	1.051	1.066	1.157	1.253	2.039	1.161	1.344	1.504	1.085	1.013
superconductivity	1.173	0.805	0.698	0.893	**0.473**	1.126	1.177	**0.384**	**0.347**	0.374	**0.420**	**0.386**	0.830	0.701	1.294	0.566
video-transcoding	**0.716**	**0.780**	**0.720**	**0.779**	**0.724**	0.830	0.845	**0.828**	**0.939**	**0.951**	1.334	**0.999**	**0.947**	1.162	**1.020**	**0.989**
wave-energy	**0.677**	0.849	1.095	1.022	0.932	0.722	0.726	0.951	1.028	1.063	1.742	1.025	1.187	1.533	1.026	1.110
white-wine	**0.922**	1.380	**0.964**	**1.021**	**1.049**	**0.953**	**1.012**	1.234	1.179	1.263	4.552	2.325	1.121	2.040	**1.089**	**1.295**

Table 11: Average test mean squared errors on OpenML data with support set size $N^{\mathrm{S}} = 10$.

Dataset	Ours	R2D2	MGP	MAML	MLH	Hyper	Shared	Ridge	Lasso	Elastic	ARD	BLM	REM	DT	GB	MLP
Moneyball	0.657	0.279	0.468	0.549	0.363	0.728	1.510	0.536	0.508	0.523	0.583	0.537	**0.184**	1.103	1.071	0.628
QSAR-fish-toxicity	0.727	0.747	0.639	0.790	**0.561**	0.839	0.979	0.763	0.825	0.859	1.073	0.764	0.801	1.046	1.094	0.930
abalone	0.947	1.151	0.862	0.901	1.039	0.979	0.905	**0.687**	0.767	0.780	0.983	0.825	0.967	1.182	1.008	0.937
airfoil-self-noise	**0.842**	2.345	**0.779**	0.977	0.997	0.879	**0.861**	**1.005**	1.160	1.332	1.344	1.112	0.872	1.325	1.014	**0.827**
auction-verification	**0.868**	**0.858**	**0.833**	0.981	**0.870**	0.964	**0.878**	**1.170**	1.313	1.325	1.487	1.298	0.940	1.579	1.228	1.132
brazilian-houses	0.660	**0.055**	**0.053**	**0.055**	0.215	0.710	0.649	0.801	0.772	0.772	0.743	0.891	0.061	1.156	1.057	0.746
california-housing	**0.889**	1.165	0.978	1.113	**0.854**	0.926	0.939	**0.853**	**1.024**	**1.227**	**0.900**	**0.908**	1.059	**0.725**	0.823	**0.772**
cars	**0.639**	**0.573**	**0.653**	0.826	**0.568**	**0.620**	**0.677**	**0.695**	**1.638**	1.862	2.492	0.900	**0.546**	2.493	**0.712**	**0.968**
concrete-compressive	**0.541**	0.656	0.805	0.974	0.782	0.559	0.568	0.707	0.811	0.944	0.975	0.819	1.026	0.905	0.863	0.742
cps88wages	**0.740**	2.099	0.771	0.752	0.781	0.870	**0.772**	0.869	1.006	0.909	1.456	0.930	0.948	1.154	0.973	0.905
cpu-activity	0.822	1.168	1.271	1.327	1.298	0.810	0.799	**0.399**	**0.431**	**0.411**	117.236	**0.632**	1.351	**0.551**	0.776	0.559
diamonds	0.728	0.518	0.643	0.590	**0.386**	0.873	0.828	**0.474**	**0.457**	**0.484**	**0.455**	0.469	**0.435**	0.685	1.252	**0.448**
energy-efficiency	0.525	**0.186**	0.431	1.019	0.249	0.500	0.502	0.883	0.923	0.936	2.037	0.983	**0.220**	1.165	0.973	1.255
fifa	0.744	0.543	0.478	**0.426**	0.465	0.754	0.782	1.645	1.709	1.673	1.859	1.681	0.817	2.077	1.930	1.759
forest-fires	**1.736**	**2.177**	**2.120**	**2.102**	**2.194**	**1.736**	**1.767**	1.022	1.082	1.340	1.654	1.187	**2.123**	1.453	**0.798**	**0.897**
fps-benchmark	**0.701**	1.177	0.955	1.005	0.947	0.726	0.776	**1.174**	**1.220**	1.266	1.871	1.249	1.330	1.790	**1.283**	1.306
geographical-origin	**0.716**	1.147	0.953	1.113	0.981	0.805	**0.684**	1.333	1.717	2.362	1.949	1.339	1.715	1.999	1.031	1.543
grid-stability	**0.487**	0.650	0.743	0.572	0.744	0.533	0.728	**0.539**	**0.554**	**0.561**	**0.668**	**0.595**	1.078	1.129	1.165	**0.601**
health-insurance	**0.795**	1.445	0.921	1.040	0.980	0.922	**0.826**	**0.959**	1.042	1.320	2.567	1.104	1.072	1.417	1.230	1.082
kin8nm	0.978	0.973	**0.806**	1.077	0.972	1.140	**0.895**	**1.304**	**1.533**	**1.515**	2.188	1.560	1.166	1.746	**1.402**	**1.495**
kings-county	0.763	**0.803**	0.915	0.912	**0.779**	**0.721**	**0.687**	1.006	1.121	1.194	4.696	0.982	0.794	1.337	1.054	1.103
miami-housing	0.904	**0.617**	0.621	0.751	**0.557**	1.119	0.932	1.107	1.207	1.234	1.650	1.156	0.711	1.754	1.182	1.203
naval-propulsion-plant	**0.913**	1.923	1.110	1.054	1.100	1.056	**0.931**	**1.116**	**1.038**	**1.099**	2.280	1.338	1.079	1.904	**1.044**	1.798
physiochemical-protein	**1.232**	2.828	**1.171**	1.436	**1.170**	**1.141**	**1.300**	1.027	**1.124**	**1.200**	**1.019**	**0.947**	**1.383**	1.660	1.341	1.453
pumadyn32nh	**0.671**	1.246	1.007	1.037	1.059	0.971	**0.696**	**0.829**	**0.910**	0.969	1.259	**0.899**	1.419	1.278	1.099	**0.878**
red-wine	**0.660**	1.277	1.079	1.301	0.972	0.705	**0.665**	1.122	1.160	1.180	4.277	1.264	1.269	1.671	1.260	1.428
sarcos	**0.551**	1.066	1.037	1.203	0.781	0.622	0.595	0.741	0.711	0.728	0.966	0.772	0.800	1.400	0.944	0.987
socmob	0.719	**0.626**	0.672	0.744	**0.559**	0.854	0.811	0.982	1.155	1.167	1.360	1.091	**0.688**	1.345	1.134	0.980
solar-flare	**0.885**	1.910	**1.229**	**1.322**	**1.238**	1.650	**1.023**	**0.949**	1.191	1.231	1.401	1.049	**1.412**	1.328	1.029	**0.850**
space-ga	**0.928**	1.234	0.812	1.139	0.784	1.031	**0.959**	**0.727**	**0.776**	**0.777**	**0.819**	**0.720**	1.006	1.129	1.053	1.251
student-performance-por	**0.868**	1.224	1.033	1.144	1.042	0.904	**0.945**	1.007	1.116	1.230	1.537	1.024	1.264	2.205	0.949	1.068
superconductivity	1.210	0.601	0.540	1.004	0.494	1.094	2.257	**0.192**	**0.178**	0.198	0.241	**0.209**	0.759	0.479	1.243	0.369
video-transcoding	**0.655**	**0.770**	**0.686**	**0.710**	**0.762**	0.727	**0.670**	**0.833**	**0.834**	**0.913**	1.536	**0.874**	**0.757**	1.667	**0.986**	**0.957**
wave-energy	**0.649**	**0.739**	0.996	0.901	0.868	**0.666**	**0.683**	0.891	0.961	0.982	1.259	0.957	1.040	1.480	1.018	1.013
white-wine	**0.938**	1.482	**1.008**	1.154	**0.923**	**1.130**	**0.980**	**1.369**	**1.403**	**1.371**	3.502	**1.706**	**1.093**	1.747	**1.090**	1.263

Tables 9, 10, and 11 show the test mean squared errors for each dataset in OpenML data. Our method did not achieve the best performance in some datasets although our method achieved significantly better performance on average as shown in Table 1. When the meta-training datasets are not related to the test dataset, our method cannot improve the performance. When the relationship between feature vectors and target values is nonlinear, our method does not perform well.

C Limitations

Compared with the existing cross-validation-based and Bayesian methods that tune regularization weights for each task, our method requires meta-learning using multiple datasets that are computationally more expensive. When a sufficient number of training data are available, the existing cross-validation, Bayesian or deep learning-based methods perform well. Although our method achieved better performance than the compared methods in terms of the test mean squared error averaged over meta-test datasets as shown in Table 1, our method was outperformed for some datasets as shown in Tables 9, 10, and 11. We evaluated our method only for linear regression in our experiments.