5-1-2023

A Research on Automatic Hyperparameter Recommendation via Meta-Learning

Liping Deng
Southern Illinois University Carbondale, liping.deng@siu.edu

A RESEARCH ON AUTOMATIC HYPERPARAMETER RECOMMENDATION VIA META-LEARNING

by

Liping Deng

B.S., Anshan Normal University, 2016
M.S., Shenzhen University, 2019

A Dissertation
Submitted in Partial Fulfillment of the Requirements for the
Doctor of Philosophy Degree

School of Mathematical and Statistical Sciences
in the Graduate School
Southern Illinois University Carbondale
May 2023

DISSERTATION APPROVAL

A RESEARCH ON AUTOMATIC HYPERPARAMETER RECOMMENDATION VIA META-LEARNING

by

Liping Deng

A Dissertation Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in the field of Mathematics

Approved by:

Dashun Xu, Chair

David Olive

Jianghong Xu

Yaser, Samadi

Lingguo, Bu

Graduate School
Southern Illinois University Carbondale
March 30, 2023

AN ABSTRACT OF THE DISSERTATION OF

Liping Deng, for the Doctor of Philosophy degree in Mathematics, presented on March 30, 2023, at Southern Illinois University Carbondale.

TITLE: A RESEARCH ON AUTOMATIC HYPERPARAMETER RECOMMENDATION VIA META-LEARNING

MAJOR PROFESSOR: Dr. MingQing, Xiao

The performance of classification algorithms is mainly governed by the hyperparameter configurations deployed. Traditional search-based algorithms tend to require extensive hyperparameter evaluations to select the desirable configurations during the process, and they are often very inefficient for implementations on large-scale tasks. In this dissertation, we resort to solving the problem of hyperparameter selection via meta-learning which provides a mechanism that automatically recommends the promising ones without any inefficient evaluations. In its approach, a meta-learner is constructed on the metadata extracted from historical classification problems which directly determines the success of recommendations. Designing fine meta-learners to recommend effective hyperparameter configurations efficiently is of practical importance.

This dissertation divides into six chapters: the first chapter presents the research background and related work, the second to the fifth chapters detail our main work and contributions, and the sixth chapter concludes the dissertation and pictures our possible future work.

In the second and third chapters, we propose two (kernel) multivariate sparse-group Lasso (SGLasso) approaches for automatic meta-feature selection. Previously, meta-features were usually picked by researchers manually based on their preferences and experience or by wrapper method, which is either less effective or time-consuming. SGLasso, as an embedded feature selection model, can select the most effective meta-features during the meta-learner training and thus guarantee the optimality of both meta-features and meta-learner which are essential for successful recommendations.

In the fourth chapter, we formulate the problem of hyperparameter recommendation as a problem of low-rank tensor completion. The hyperparameter search space was often stretched to a one-dimensional vector, which removes the spatial structure of the search space and ignores the correlations that existed between the adjacent hyperparameters and these characteristics are crucial in meta-learning. Our contributions are to instantiate the search space of hyperparameters as a multi-dimensional tensor and develop a novel kernel tensor completion algorithm that is applied to estimate the performance of hyperparameter configurations.

In the fifth chapter, we propose to learn the latent features of performance space via denoising autoencoders. Although the search space is usually high-dimensional, the performance of hyperparameter configurations is usually correlated to each other to a certain degree and its main structure lies in a much lower-dimensional manifold that describes the performance distribution of the search space. Denoising autoencoders are applied to extract the latent features on which two effective recommendation strategies are built.

Extensive experiments are conducted to verify the effectiveness of our proposed approaches, and various empirical outcomes have shown that our approaches can recommend promising hyperparameters for real problems and significantly outperform the state-of-the-art meta-learning-based methods as well as search algorithms such as random search, Bayesian optimization, and Hyperband.

Keywords: Machine learning, Classification, Automatic hyperparameter recommendation, Meta-learning

ACKNOWLEDGMENTS

I would like to thank Prof. MingQing Xiao for his invaluable assistance and insights leading to the writing of this dissertation and the guidance of research over the past four years. It is impossible for me to have papers published without his assistance. Prof. Xiao is a very responsible and diligent professor who always has a high standard for his students and gives feedback quickly even during the hard time he had to go through, and I have benefited a lot from the training in my Ph.D. study. I really appreciate his constant enthusiasm in correcting my academic writing and the encouragement given to me when papers were rejected in a row.

I would like also to thank Prof. Dashun Xu for his help and time in my schoolwork when my advisor was on leave so my Ph.D. study could go on smoothly. My sincere thanks also go to the four members of my graduate committee for their patience, guidance, time, and suggestions given to my preliminary exam and dissertation. They are Prof. David Olive, Prof. Jianhong Xu, Prof. Lingguo Bu, and Prof. Yaser Samadi, respectively.

To complete this dissertation, many codes, data, and computation resources were obtained free from the Github, Scikit-learn, UCI machine learning repository, OpenML, Kaggle, Colab platform from Google, and BigDwag high-performance cluster provided by SIUC. I sincerely thank those institutions and authors who made these resources publicly available.

Finally, I want to say thanks to my parents, sister, relatives, and all my friends who gave me tremendous support and care over the past four years so I can finish my Ph.D. Studying overseas is not so easy, especially given the pandemic started at the beginning of 2020. However, these unforgettable experiences will be important parts of my entire life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Classification is one of the most critical tasks in the study of machine learning, and many classification algorithms, such as support vector machines [1], decision trees [2], random forests [3], and neural networks [4], have been developed for solving various classification problems. The performance of classification algorithms is mainly governed by the hyperparameter configurations deployed, mainly under the mathematical framework [5]. To obtain the desirable outcomes, classification algorithms bear the required costs such as time and computational resources to determine the well-suited hyperparameter configurations for each single classification task, which presents high demand for efficient classification in real scenarios. Therefore, hyperparameter optimization, whose goal is to seek optimal hyperparameter configurations, has been becoming one of the most important research topics in the machine learning community for recent years [6, 7, 8, 9].

Search-based methods associated with hyperparameter selection are widely studied in the literature. In this approach, an appropriate search space of the target algorithm is predefined in which the optimal hyperparameter configuration resides. By exploring various well-known search strategies, many search algorithms have been designed and applied to hyperparameter tuning such as grid search, random search [7], Bayesian optimization [6], Hyperband [8], estimation of distribution algorithms [10], particle swarm optimization [11], etc. Nevertheless, since search algorithms often require extensive configuration evaluations from scratch for every single new task, the low efficiency on large-size problems appears to be a critical issue. Quite often, practitioners have to limit the search rounds due to the burden of high computational costs. As the number of hyperparameters increases, the size of the search space can grow exponentially, and thus looking for effective configurations within limited searches deems to be necessary.

Meta-learning [12], or learning to learn, provides a mechanism that learns the experience from historical classification tasks and then uses it to solve new problems in a much more effective and efficient way. Meta-learning has been widely investigated for automatic hyperparameter configuration recommendations where time-consuming configuration evaluations are not needed during the recommendation phase [13, 14]. In its process, the *metadata*, i.e., dataset characteristics (called meta-features) and classification performance of the candidate hyperparameter configurations, is extracted from a group of historical classification problems. Accordingly, a *meta-learner* is built on the metadata to identify the intrinsic relationship between dataset characteristics and historical performance, and therefore, the prospective configuration of a new problem can be inferred by its dataset characteristics. Because of their critical roles, both meta-learners and meta-features are the essential components for configuration recommendation tasks under the framework of meta-learning.

One of the classical and typical recommendation algorithms is KNN [15, 16], which determines the *similarity* between the new problems and the historical problems by measuring the suitable defined distance among meta-features, and the problems that are close to the new problem (measured by "distance") are assumed to be similar ones, thus the shared optimal hyperparameter configurations may be identified. Another type of hyperparameter recommendation strategy in meta-learning is model-based collaborative filtering (CF) [17, 18]. In the context of CF, configurations are treated as items and problems as users. The performance can be evaluated by the score for which the users rate the items. Hence, when partial evaluations of a new problem are available, the performance of the entire configuration space can be predicted via CF, and the one with the highest predictive performance is retrieved. The notable advantage of CF is that there is no need to evaluate all configurations in the search space over the historical problems, leading to the efficiency of the offline performance evaluation being further improved. Other meta-based models also include decision trees [19], linear regression [20, 21], neural networks [22], and self-organizing maps [3], etc, where hyperparameter configuration selection is considered as either a meta-classification or meta-regression problem.

Meta-features are a group of measures that characterize datasets from various aspects, capturing the characteristics of an individual dataset uniquely [23]. There are mainly eight types of meta-features for classification problems in literature, namely, statistics and information theory (SIT) [21, 20], model structure (MS) [24, 25], (relative) landmarking (LM, RLM) [26, 27, 28], structural information (SI) [15], concept (Con) [29, 30], clustering (Clu) [31], and problem complexity (PC) [32]. Each of them characterizes datasets following different assumptions, e.g., SIT collects simple statistical or information-theory-based measures to describe datasets such as the number of instances (classes, attributes), class probability (entropy), and mutual information; LM and RLM adopt the performance of base learners with distinct learning strategies, called landmakers, evaluated on the datasets as meta-features; PC tries to grasp the decision boundaries of classification datasets and measure the complexity of making a correct classification.

Although the existing approaches in meta-learning have achieved significant progress recently, there are many issues that remained to be addressed. For instance, because of the incapability of selecting the most suitable meta-features automatically for existing meta-learners, meta-features must be manually prescribed based on researchers' preferences and experience for various recommendation tasks for which the optimal effectiveness often cannot be guaranteed. In current studies, some important information such as data spatial structure and the latent space associated with its hyperparameter search space, which are critical characteristics in hyperparameter recommendation tasks, has not been applied to the algorithm learning process yet. In this dissertation, we aim to address these challenges by developing new effective hyperparameter recommendation approaches under the framework of meta-learning. Respectively, we 1) propose two multivariate group-Lasso models for automatic meta-feature selection; 2) model the search space as multi-dimensional tensors to sustain its spatial structure and convert the recommendation tasks as a tensor completion problem; 3) discover the latent features of search space via denoising autoencoders to identify the performance distribution of candidate configurations. Extensive experiments on real-world classification problems are provided and the obtained results demonstrate the superiority of our proposed approaches for hyperparameter recommendation, compared to the state-of-the-art meta-learning baselines as well as various search algorithms.

1.2 NOTATIONS

We summarize the common, standard mathematical notations as well as matrix (vector) norms used throughout this dissertation in Table 1.1 and Table 1.2, respectively, and the other specific ones will be explained when necessary.

Table 1.1: The mathematical notations defined throughout the dissertation.

Name	Description	Example
Matrices	Bold uppercase letters	$\mathbf{X}$
Vectors	Bold lowercase letters	$\mathbf{x}$
Tensors	Bold calligraphic uppercase letters	$\boldsymbol{\mathcal{X}}$
Identity Matrices	Bold uppercase letter “I”	$\mathbf{I}$
Zero Matrices (Vectors)	Bold number “0”	$\mathbf{0}$
Scalars	Lowercase letters	x
Entries of Matrices	Uppercase letters with subscripts	X_{ij}
Entries of Vectors	Lowercase letters with subscripts	x_i
Entries of Tensors	Calligraphic letters with subscripts	$\mathcal{X}_{i_1,i_2,\cdots,i_d}$
Columns of Matrices	Bold lowercase letters with subscripts	$\mathbf{x}_i$
Rows of Matrices	Bold lowercase letters with subscripts	$\mathbf{x}_{i\star}$
Matrix Transposes	Bold uppercase letters with superscript $\top$	$\mathbf{X}^\top$
Vector Transposes	Bold lowercase letters with superscript $\top$	$\mathbf{x}^\top$
Trace	Trace of square matrices	$tr(\mathbf{X})$
Real Number Field	Blackboard bold uppercase letter “R”	$\mathbb{R}$
Sets	Capital omega	Ω
Classification Algorithms	Ralph Smith’s formal script “A”	$\mathscr{A}$
Datasets	Ralph Smith’s formal script “D”	$\mathscr{D}$

1.3 HYPERPARAMETER OPTIMIZATION

In literature [33], for a given classifier $\mathscr{A}$, e.g., support vector machines (SVM), the hyperparameter search space can be described by the Cartesian product

$$\boldsymbol{\Omega}_{\mathscr{A}} = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_q, \tag{1.1}$$

where q is the number of hyperparameters of classifier $\mathscr{A}$, and $\Omega_i, i = 1, 2, \cdots, q$, represent the search space of each hyperparameter. These search spaces can be real-valued (e.g., the regularization coefficient and RBF kernel width for SVM), integer-valued (e.g., the number of nearest neighbors for KNN), and binary (e.g., split criteria “`entropy`” or “`gini`” for

Table 1.2: The matrix and vector norms defined throughout the dissertation.

Name	Object	Definition
L_1-norm	$\mathbf{x} \in \mathbb{R}^n$	$\|\mathbf{x}\|_1 = \sum_{i=1}^n \|x_i\|$
L_2-norm	$\mathbf{x} \in \mathbb{R}^n$	$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$
Frobenius-norm	$\mathbf{X} \in \mathbb{R}^{m\times n}$	$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n X_{ij}^2}$
Kernel-norm	$\mathbf{X} \in \mathbb{R}^{m\times n}$	$\|\mathbf{X}\|_\star = \sum_{i=1}^{\min\{m,n\}} \sigma_i$, σ_i's are the singular values of $\mathbf{X}$
$L_{2,1}$-norm	$\mathbf{X} \in \mathbb{R}^{m\times n}$	$\|\mathbf{X}\|_{2,1} = \sum_{j=1}^n \sqrt{\sum_{i=1}^m X_{ij}^2}$

decision trees). For a given dataset $\mathscr{D}$, we are required to select the optimal hyperparameter configuration (in short, we call it configuration hereafter)

$$\boldsymbol{\omega}^\star = [\omega_1, \omega_2, \cdots, \omega_q],$$

where $\omega_j \in \Omega_j, j = 1, 2, \cdots, q$, to optimize the classification performance of $\mathscr{A}$ in the form of

$$\boldsymbol{\omega}^\star = \arg\operatorname{opt}_{\boldsymbol{\omega}\in\boldsymbol{\Omega}_{\mathscr{A}}} V(\mathcal{P}, \mathscr{A}_{\boldsymbol{\omega}}, \mathscr{D}),$$

where $V(\cdot,\cdot,\cdot)$ represents a validation strategy, and the most common one is the k-fold cross-validation. The notation $\mathcal{P}$ stands for a predefined classification performance metric, e.g., classification accuracy (error) rate or statistical indices.

1.3.1 Overview of Search algorithms

Grid search, known as a model-free blackbox strategy, was first proposed for hyperparameter selection in the 1990s [9]. Based on its mechanism, each point in the hyperparameter search space must be evaluated, so grid search can find the optimal one among the candidate configurations. However, its computational complexity grows exponentially as the number of hyperparameters increases[1]. In contrast, random search approach takes samples from the configuration space randomly, which then are evaluated with a preset stopping criterion [7]. It works better than grid search when part of the configurations is more effective than the others. Because of model-free, the advantage of both grid search and random search lies in that the configuration evaluations can be deployed in a parallel way.

Bayesian optimization [6] is a model-based blackbox global hyperparameter selection method, different from grid search and random search, it iteratively seeks the optimal configuration using the experience learned from historical configuration searches. Bayesian optimization comprises a probabilistic surrogate model, e.g., Gaussian process, random forest, tree Parzen estimator, etc., that is fitted to all historical evaluations explored so far, and an acquisition function, e.g., expected improvement, which decides the most prospective configuration that should be evaluated next. Bayesian optimization has been widely applied for hyperparameter tuning in machine learning and deep learning, and many variants, e.g., heteroscedastic evolutionary Bayesian optimization [34] and gradient Bayesian optimization [35], are proposed for effective hyperparameters tuning in applications.

Li et al. [8] proposed a novel random search-based algorithm, called Hyperband, by formulating hyperparameter optimization as a non-stochastic infinite-armed bandit problem. For a given number of configurations (‘`n`’) and total finite budget (‘`B`’), e.g., time, and ‘`B/n`’ stands for the maximal budget that can be assigned for each configuration, ‘`n`’ configurations are randomly picked from the search space and then the `SuccessiveHalving` algorithm [36] is leveraged to determine the best one among the selected. In the process

[1]For example, there are 175 billion parameters in OpenAI’s GPT-3. Details can be seen at `https://www.sciencefocus.com/future-technology/gpt-3/`

of `SuccessiveHalving`, we purport to drop some mediocre configurations and thus allocate more budget for the remaining configurations in the next iteration. To balance the number of tested configurations '`n`' and the budget '`B/n`' allocated for each configuration, the authors embedded `SuccessiveHalving` into the simple grid search to select the best '`n`'. Despite Hyperband sometimes outperforming Bayesian optimization, a significant amount of computational resources must be consumed and it does not utilize any historical evaluations as Bayesian optimization does, which often impacts the effectiveness.

Uniquely, the Estimation of Distribution Algorithms (EDAs) [10] attempt to explore an explicit probabilistic model that is built on the *promising* configurations found so far to suggest the prospective configuration for the next iteration. EDAs start with a population of candidate configurations of the algorithm which is generated according to the uniform distribution over the configuration search space. This population of configurations is then evaluated and scored so we can rank the configurations based on their performance to select the partial most promising configurations among them, e.g., 50% of the population is kept. Next, an explicit probabilistic model is constructed on the selected configurations, which is used to generate new configurations by sampling the distribution encoded by this model. Thus, we can incorporate them back into the old population (or simply replace them) and repeat the process until a stopping criterion is attained. EDAs have a wide range of applications [37], such as military antenna design, groundwater remediation design, and economic dispatch, and so on.

Additionally, other approaches contain gradient search [38], population-based search, e.g., particle swarm optimization [11], Tabu search [39], and racing algorithm [40], etc. The detailed surveys of search algorithms can be found in [41].

1.4 META-LEARNING

Suppose that $\{\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_n\}$ are a collection of historical classification datasets, and $\mathscr{A}$ is the investigated classification algorithm instantiated with search space $\Omega_{\mathscr{A}} = \{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m\}$ containing m points, then the metadata $\{\mathbf{x}_i, \mathbf{y}_i\}$ is extracted from each of $\mathscr{D}_i$, where $\mathbf{x}_i$ is meta-features and $\mathbf{y}_i = [y^{(i)}_{\boldsymbol{\omega}_1}, y^{(i)}_{\boldsymbol{\omega}_2}, \cdots, y^{(i)}_{\boldsymbol{\omega}_m}]^\top$ is the classification performance of the candidate hyperparameter configurations, respectively. A meta-learner $\mathcal{M}_{\boldsymbol{\theta}}$ is next built on the metadata to grasp the underlying relationship between meta-features and classification performance such that

$$\boldsymbol{\omega}^\star_i = \mathcal{M}_{\boldsymbol{\theta}}(\mathbf{x}_i), i = 1, 2, \cdots, n \tag{1.2}$$

where $\boldsymbol{\theta}$ is the model parameters determined via *meta-learning* and $\boldsymbol{\omega}^\star_i$ stands for the optimal configuration on dataset $\mathscr{D}_i$, i.e., $\boldsymbol{\omega}^\star_i = \arg\max_{\boldsymbol{\omega}}([y^{(i)}_{\boldsymbol{\omega}_1}, y^{(i)}_{\boldsymbol{\omega}_2}, \cdots, y^{(i)}_{\boldsymbol{\omega}_m}])$, and our goal is that for a new dataset $\mathscr{D}$ with meta-feature $\mathbf{x}$, we can recommend the one suggested by the trained meta-learner to $\mathscr{D}$, i.e., $\boldsymbol{\omega}^{recom} = \mathcal{M}_{\boldsymbol{\theta}}(\mathbf{x})$, which satisfies

$$\boldsymbol{\omega}^{recom} = \arg\max_{\boldsymbol{\omega}}([y_{\boldsymbol{\omega}_1}, y_{\boldsymbol{\omega}_2}, \cdots, y_{\boldsymbol{\omega}_m}]), \tag{1.3}$$

where $\mathbf{y} = [y_{\boldsymbol{\omega}_1}, y_{\boldsymbol{\omega}_2}, \cdots, y_{\boldsymbol{\omega}_m}]$ are the true performance of $\mathscr{A}$ evaluated on $\mathscr{D}$, here $\mathbf{y}$ is unknown and is to be determined.

We summarize the framework of meta-learning-based recommendation, similar to [42], in Figure 1.1. As one can see, during the recommendation phase, computation resources are needed to extract meta-features instead of evaluating any configurations as search algorithms

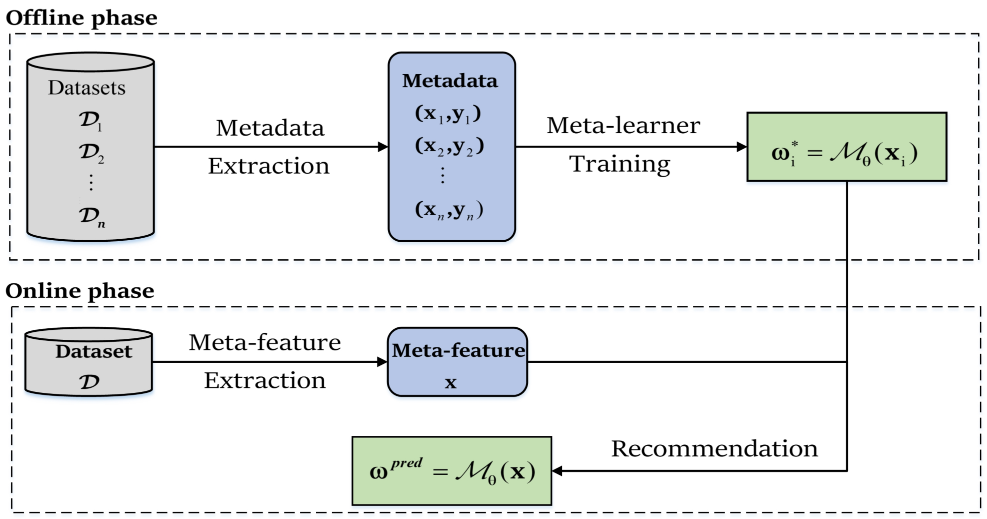


Figure 1.1: The general framework of meta-learning-based hyperparameter recommendation.

do.

1.4.1 Meta-learning-based recommendation

Essentially, the meta-learning-based hyperparameter selection methods only differ from the recommendation algorithms (or meta-learners) that are employed. Currently, there are four common categories in literature, i.e., similarity-based recommendation, collaborative filtering, meta-classification, and meta-regression, respectively, as illustrated next in detail.

Similarity-based recommendation (SR): For a completely new classification task at hand, the similarity with the historical datasets can be characterized by defining appropriate metric distances, e.g., Euclidean distance, Cosine, and L1 distance, between their meta-features to determine the most suitable configurations for the recommendation. KNN is usually adopted to select the most K similar datasets so that we can sort the configurations on each dataset and recommend the one that has the minimal average rank over the K datasets to the new dataset [43, 44, 45, 16]. Soares et al. [43] first applied KNN to the kernel width selection for SVM in regression tasks, in which the metadata on 42 datasets and 14 statistical meta-features are generated. Similarly, Gomes et al. [44] built a recommendation architecture on 40 regression problems to select both kernel width and regularization coefficient for SVM, the suggested configurations were further fed to search algorithms, such as particle swarm optimization and Tabu search, to obtain more effective ones. In general, the number of neighbors K itself will affect the recommendation results. Smith et al. [46] suggested an interval to choose K, whose range is from 10% to 25% of the total number of historical datasets. This method only utilizes several closest datasets, and it does not make use of any information from other remaining historical datasets that may contain useful information for the data overall characteristics.

Collaborative filtering (CF): Stern et al. [17] pioneered a new approach by utilizing the ideal of model-based collaborative filtering (CF). The performance prediction of a configuration is treated as the prediction of the score that a user (dataset) may give to an item (configuration). Probabilistic matrix factorization (PMF) techniques were employed to mine the latent variables of both items (configurations) and users (datasets). One issue of this approach is that we must evaluate some configurations in advance for a new given classification task. Yang et al. [47] employed a D-optimal experiment design to select the configuration evaluation subset that requires an extra computational effort. To deal with this issue, Misir and Sebag [18] trained a model, e.g., random forest, that maps meta-features to the extracted variables to estimate the preference of the new dataset. However, this estimation often appears to be not accurate enough. Fusi and Elibol [48] leveraged PMF and Gaussian process to predict the performance of configurations, where Bayesian optimization can be applied efficiently to search for the best configuration associated with a low number of dimensions. The initialization of Bayesian optimization is done by KNN but the cold-starting issue remains.

Meta-Classification (MC): Mantovani et al. [49] considered the configuration recommendation as a binary classification problem. They first evaluated the SVM on 124 datasets using default settings from some libraries, e.g., LibSVM and Weka, and the configurations that were acquired through searching algorithms, then the datasets were divided into two classes, each of them in which the default setting performs better was labeled as 1 and otherwise was labeled as 0. Six popular classifiers were therefore selected, e.g., decision trees, naive Bayes, multilayer perceptron, etc., to fit the metadata appropriately. However, under this setting, it could only recommend two fixed configurations for new problems. Later, the same authors conducted a further study by focusing on the tuning hyperparameters of SVM within the same framework [50]. In other related literature, the decision tree (DT) classifier was widely adopted as a meta-learner because of its ability in capturing the importance of meta-feature variables [51, 52, 53]. The notable drawback of meta-classification is that some configurations may be biased by the historical datasets, namely, the configurations that are not included in labels will never be selected for new datasets.

Meta-Regression (MR): Gama and Brazdil [54] first used linear regression models to solve the problem of classifier selection, where the classification errors were normalized in three manners by transforming them into other units so that they are comparable across different datasets. Sohn [20] utilized a logistic transformation to normalize the classification error rates that lead to a logistic regression model built on 19 datasets. To eliminate the influence of the small size of the data pool, the Bootstrap re-sampling approach was adopted to select a subset of meta-features based on the statistical information. However, these regression models consider neither the multicollinearity existing in meta-features nor

Table 1.3: The comparisons between the characteristics of the representative meta-learners from SR, CF, MC, and MR.

Meta-learner (Category)	1	2	3	4	5	6
KNN (SR)		✓		✓		
PMF (CF)	✓	✓	✓			
DT (MC)	✓		✓	✓		
KM (MR)	✓	✓		✓		

the nonlinear relationship between meta-features and classification performance, and hence they are not suitable for configuration selection. Furthermore, Bensusan and Kalousis [55] constructed the regression models using Cubist and kernel methods (KM), where a larger pool of datasets, as well as more types of meta-features, were considered, but reasons for choosing these two regression models were not well justified and the applicable scope was unclear.

In summary, we list the characteristics of the representative meta-learners from each category in Table 1.3, where the numbers 1, 2, 3, 4, 5, and 6 represent the meta-learner: utilizes all the datasets, utilizes all configurations, has the model interpretability, does not have the cold-starting problem, performs meta-feature selection automatically, and utilizes the spatial structure of search spaces, respectively.

1.5 CONCLUDING REMARKS

This chapter overviews two hyperparameter tuning strategies in the literature, namely, search and meta-learning-based recommendation. Search-based algorithms require configuration evaluations in their processes which are inefficient in practice. Instead of searching, meta-learning resorts recommend promising ones based on the experience learned from historical classification problems, where the meta-learner is supposed to grasp the intrinsic connection between dataset features and the best underlying configurations. Nevertheless, existing approaches have their own disadvantages which cannot produce the desired recommendation performance. The next four chapters will present our work that solves various demerits that existed in previous approaches.

CHAPTER 2

AUTOMATIC META-FEATURE SELECTION

2.1 INTRODUCTION

Currently, there are mainly eight types of meta-features in literature, which are statistics and information theory, model structure, (relative) landmarking, structural information, concept, clustering, and problem complexity [56, 57]. Since meta-features are the main source for the meta-learner to generate the configuration recommendation, the study on the automatic selection of meta-features appears to be practically attractive. Depending on the datasets, some measures from different types of meta-features are either redundant or correlated, and thus meta-feature selection often becomes necessary in order to achieve effective and efficient recommendations. However, to the best of our knowledge, the current study in dealing with meta-feature selection under the framework of meta-learning is quite limited. For example, Todorovski [58] and Kalousis [59] employed the wrapper method to conduct meta-feature selection, where meta-learner training is required for every round of meta-feature subset search, which leads to a computationally expensive process. Similarly, Cruz and Sabourin et al. [60] adopted the binary particle swarm optimization to search the optimal meta-feature subset, which is designed only for dynamic ensemble selection tasks, and the computational issues that appeared in the wrapper method remain. Bilalli and Abello et al. [19] combined the principal component analysis with a partial correlation graph to discover the important latent features of meta-features but little time budget can be saved during the meta-feature extraction phase.

In this chapter, we propose a novel multivariate sparse-group Lasso (SGLasso) method, aiming at automatic meta-feature selection, for the classification hyperparameter configuration recommendations under the framework of meta-learning. Different from the existing approaches, as an embedded feature selection model, SGLasso can inlay meta-feature selection into the meta-learner training process so that the desired meta-features can be identified and the optimal performance of the meta-learner can be attained. More specifically, we first set up a multivariate sparse-group Lasso meta-regression problem on a collection of historical datasets where meta-features and historical classification performance of configurations are applied as predictors and responses respectively. Then, an efficient algorithm is developed to learn the model coefficients used for meta-feature selection. For a new dataset, the classification performance of candidate configurations can be predicted by the meta-features of the new dataset selected by the trained SGLasso model. As a result, the configuration with the highest predictive performance can be recommended for the new dataset. The main contributions of this chapter are summarized as follows:

- Different from current approaches in the literature, we establish a new model via multivariate sparse-group Lasso learning, focusing on the meta-feature selection through meta-learning. Technically, we adopt the $L_{2,1}$-norm (see, e.g., [61, 62, 63]) in the regularization to reduce the number of model parameters and promote the sparsity between and within groups. An efficient learning algorithm via majorization minimization is developed for SGLasso training in the selection of principle meta-futures.

- The developed SGLasso model is embedded into the configuration recommendation under the framework of meta-learning. An automatic hyperparameter configuration architecture is fully developed and tested on 136 UCI datasets for applications.

- Extensive experiments are conducted to demonstrate the effectiveness of our proposed approach. The empirical results have shown that SGLasso is capable of recommending highly suitable configurations via feature selection and outperforming the existing meta-learning approaches as well as classic search algorithms such as random search, Bayesian optimization, and Hyperband.

2.2 RELATED WORK

2.2.1 Meta-feature selection

In literature, how to obtain dataset features refers to meta-feature selection. Todorovski and Brazdil et al. [58] first experimentally studied the potential of automatic meta-feature selection based on recommendation performance, and they adopted the wrapper method to perform the meta-feature selection with two types of meta-features: statistics & information theory and landmarking. A clear performance improvement was observed based on their approach. Later, Kalousis and Hilario [59] conducted a similar study with an increasing number of features, and found that the wrapper method was not efficient due to the computation intensity.

Bilalli and Abello et al. [19] proposed to use principal component analysis together with partial correlation graphs to select the latent features of meta-features. In their approach, principal component analysis was first applied to the extracted meta-features, and the orthogonal rotation was next employed on the retained principal factors to obtain latent features from the meta-features. Then, the latent features were further augmented by the performance of a candidate algorithm of interest, on which a partial correlation graph was generated to visualize the subtle relationship between the latent features and the responses so that the most relevant latent features can be picked. They also empirically showed that the automatic meta-feature selection can improve the predictive power of the deployed meta-learner (random forest). However, their work only considered the simple 61 statistical or information theory-based meta-features and the methodology was designed for the performance prediction of a single algorithm therefore it is not applicable to the hyperparameter recommendation task.

At the same time, Cruz and Sabourin et al. [60] employed the so-called binary particle swarm optimization (BPSO) to conduct the automatic meta-feature selection, and they adopted both S-shaped and V-shaped transfer functions that delivered the best overall performance. The objective function of BPSO is the difference between the performance of the configuration recommended by the meta-learner fitted on the currently selected meta-features and that of the ideal optimal configuration, which implies that meta-learner training is required in each round of search. Their approach bears a heavy burden of computations and hinders the efficiency of meta-learning. Further, in their experiments, their studies focused on the problem of dynamic ensemble selection of classifiers. The effectiveness of their proposed BPSO technique for the automatic meta-feature selection on 15 sets of meta-features and 30 datasets confirmed that meta-features are required to be well-selected for each individual recommendation task. Also, their proposed approach for meta-feature selection was designated to measure the level of *competence* for the basis classifiers on input samples and was limited to the dynamic ensemble selection problem, rather than a general meta-feature selection.

2.2.2 Group Lasso

Group Lasso was initially developed by Yuan and Lin [64] for the univariate regression problems, the proposed objective function is given by

$$\min_{\boldsymbol{\beta}_1,\boldsymbol{\beta}_2,\cdots,\boldsymbol{\beta}_p} \|\mathbf{y} - \sum_{l=1}^{p} \mathbf{X}_l^\top \boldsymbol{\beta}_l\|_2^2 + \lambda \sum_{l=1}^{p} \sqrt{g_l}\|\boldsymbol{\beta}_l\|_2,$$

where $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{X}_l \in \mathbb{R}^{r_l \times n}$, $\sqrt{g_l}$ depends on the group sizes, and $\boldsymbol{\beta}_l \in \mathbb{R}^{r_l}, l = 1, 2, \cdots, p$. This model is allowed to drop some groups of variables, i.e., impose the between-group sparsity by setting the corresponding parameter vector $\boldsymbol{\beta}_l$ to be zeros via tuning parameter λ. To add sparsity within the groups, Simon et al. [65] proposed a sparse-group Lasso, which is formulated as

$$\min_{\boldsymbol{\beta}_1,\boldsymbol{\beta}_2,\cdots,\boldsymbol{\beta}_p} \|\mathbf{y} - \sum_{l=1}^{p} \mathbf{X}_l^\top \boldsymbol{\beta}_l\|_2^2 + \lambda_1 \sum_{l=1}^{p} \|\boldsymbol{\beta}_l\|_2 + \lambda_2 \sum_{l=1}^{p} \|\boldsymbol{\beta}_l\|_1.$$

This model can enforce both within and between group sparsity via a traditional L_1-norm constraint.

Multivariate sparse-group Lasso [66], which deals with the cases of high-dimensional response variables, is given as follows:

$$\min_{\mathbf{W} \in \mathbb{R}^{m \times r}} \|\mathbf{Y} - \sum_{l=1}^{p} \mathbf{W}_l \mathbf{X}_l\|_F^2 + \lambda \sum_{l=1}^{p} \|\mathbf{W}_l\|_F + \lambda_G \|\mathbf{W}\|_1,$$

where $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p]$, $\|\mathbf{W}\|_1 = \sum_{ij} |W_{ij}|$, and $\mathbf{Y} \in \mathbb{R}^{m \times n}$. Obviously, this model considers the element-wise sparsity of $\mathbf{W}$, and thus the predicted response $\mathbf{y}$ for any given predictor $\mathbf{x}$ must be sparse too, which can present undesirable restrictions for linear regression tasks. Moreover, such an approach is not suitable for the meta-feature selection addressed in this chapter since sparsity or small value of features may lead to numerical instability during the optimization process. We will address this issue under our approach setting.

2.3 MULTIVARIATE SPARSE-GROUP LASSO

Suppose that $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n] \in \mathbb{R}^{m\times n}$ is the response matrix and each column $\mathbf{y}_i \in \mathbb{R}^m$ is an instance with continuous entries, and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{r\times n}$ is the design matrix and each column $\mathbf{x}_i \in \mathbb{R}^r$ is an instance. Then a multivariate multiple linear regression model can be constructed as

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{E}, \tag{2.1}$$

where $\mathbf{E} = [\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \cdots, \boldsymbol{\epsilon}_n] \in \mathbb{R}^{m\times n}$ is the error term and $\boldsymbol{\epsilon}_i \sim \mathcal{N}_m(\mathbf{0}, \sigma_i^2\mathbf{I}), i = 1, 2, \cdots, n$, where $\mathcal{N}_m(\cdot,\cdot)$ represents the multivariate Gaussian distributions. Aiming at being able to conduct feature selection and capture the intrinsic connection between the predictors and responses, a multivariate sparse-group Lasso (SGLasso) model is constructed as

$$F(\mathbf{W}) = \frac{1}{2}\|\mathbf{Y} - \sum_{l=1}^{p} \mathbf{W}_l\mathbf{X}_l\|_F^2 + \lambda\sum_{l=1}^{p}\|\mathbf{W}_l\|_{2,1}, \tag{2.2}$$

where $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p] \in \mathbb{R}^{m\times r}$ is the coefficient matrix in which $\mathbf{W}_l \in \mathbb{R}^{m\times r_l}, l = 1, 2, \cdots, p$, and $\mathbf{X}_l \in \mathbb{R}^{r_l\times n}$ $(l = 1, 2, \cdots, p)$ with $\sum_{l=1}^{p} r_l = r$ is acquired by partitioning $\mathbf{X}$ row-wisely into p disjoint groups. $\lambda > 0$ is a tuning parameter that controls the sparsity of $\mathbf{W}$ in (2.2). Notice that model (2.2) degrades to multivariate Lasso [67] when $p = 1$.

To solve for $\mathbf{W}$ that minimizes (2.2), we adopt block-coordinate descent by fixing $\mathbf{W}_l, l \neq k$, and solve for $\mathbf{W}_k$ each time. Hence, we can rewrite the loss function of (2.2) by ignoring the terms that do not involve $\mathbf{W}_k$ as

$$F(\mathbf{W}_k) = \frac{1}{2}\|\mathbf{S}_{-k} - \mathbf{W}_k\mathbf{X}_k\|_F^2 + \lambda\|\mathbf{W}_k\|_{2,1} + C_{-k}, \tag{2.3}$$

where $\mathbf{S}_{-k} = \mathbf{Y} - \sum_{l\neq k}^{p} \mathbf{W}_l\mathbf{X}_l$, and $C_{-k} = \lambda \sum_{l\neq k}^{p}\|\mathbf{W}_l\|_{2,1}$ is a constant. To simplify notations, we rewrite (2.3) as

$$F(\mathbf{B}) = \frac{1}{2}\|\mathbf{S} - \mathbf{B}\mathbf{H}\|_F^2 + \lambda\|\mathbf{B}\|_{2,1} + C, \tag{2.4}$$

where $\mathbf{S} = \mathbf{S}_{-k} \in \mathbb{R}^{m\times n}$, $\mathbf{B} = \mathbf{W}_k \in \mathbb{R}^{m\times r_k}$, $\mathbf{H} = \mathbf{X}_k \in \mathbb{R}^{r_k\times n}$, and $C = C_{-k}$.

With some matrix algebra, the gradient of (2.4) with respect to $\mathbf{B}$ is given by

$$\nabla F(\mathbf{B}) = -\mathbf{S}\mathbf{H}^\top + \mathbf{B}\mathbf{H}\mathbf{H}^\top + \lambda\mathbf{U}, \tag{2.5}$$

where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{r_k}]$ and

$$\mathbf{u}_i = \begin{cases} \frac{\boldsymbol{\beta}_i}{\|\boldsymbol{\beta}_i\|_2}, & \text{if } \|\boldsymbol{\beta}_i\|_2 \neq 0, \\ \mathbf{z}_i, & \text{if } \|\boldsymbol{\beta}_i\|_2 = 0, \end{cases} \tag{2.6}$$

where $\boldsymbol{\beta}_i$ is the i^{th} columns of $\mathbf{B}$ and $\|\mathbf{z}_i\|_2 \leq 1$. Therefore, if $\|\boldsymbol{\beta}_i\|_2 \neq 0$, then $\boldsymbol{\beta}_i$ can be updated by letting $\nabla F(\mathbf{B}) = \mathbf{0}$ via

$$\boldsymbol{\beta}_i = \left(\mathbf{S}\mathbf{h}_{i\star}^\top - \sum_{j\neq i}^{r_k} \boldsymbol{\beta}_j \|\mathbf{h}_{j\star}\|_2^2\right)\left(\|\mathbf{h}_{i\star}\|_2^2 + \frac{\lambda}{\|\boldsymbol{\beta}_i\|_2}\right)^{-1}; \tag{2.7}$$

otherwise, if $\|\boldsymbol{\beta}_i\|_2 = 0$, it satisfies

$$\left\|\mathbf{S}\mathbf{h}_{i\star}^\top - \sum_{j\neq i}^{r_k} \boldsymbol{\beta}_j \|\mathbf{h}_{j\star}\|_2^2\right\|_2 \leq \lambda. \tag{2.8}$$

In fact, updating rule (2.7) is unstable in the scenarios where $\|\mathbf{h}_{i\star}\|_2 \to 0$, e.g., the i^{th} feature vector of $\mathbf{H}$ contains very small values or it is extremely sparse. The analysis of the instability of (2.7) can be found in Appendix IV. Next, we tackle this issue via the majorization minimization scheme.

2.3.1 Majorization minimization

Denote

$$l(\mathbf{S}, \mathbf{B}) = \frac{1}{2}\|\mathbf{S} - \mathbf{B}\mathbf{H}\|_F^2, \tag{2.9}$$

and assume that $\mathbf{H}$ does not have zero rows, namely, $\frac{1}{\|\mathbf{h}_{j\star}\|_2^2}, j = 1, 2, \cdots, r_k$, are well-defined, then we have the following theorem and the detailed proof is given in Appendix IV for the sake of completeness.

Theorem 1. Centered at $\mathbf{B}^t$, a conformable constant matrix, (2.9) can be majorized as

$$l(\mathbf{S}, \mathbf{B}) \leq l(\mathbf{S}, \mathbf{B}^t) + tr((\mathbf{B} - \mathbf{B}^t)^\top \nabla l(\mathbf{S}, \mathbf{B}^t)) + \frac{1}{2a}\|\mathbf{B} - \mathbf{B}^t\|_F^2, \tag{2.10}$$

where

$$0 < a \leq \min\{\frac{1}{\|\mathbf{h}_{1\star}\|_2^2}, \frac{1}{\|\mathbf{h}_{2\star}\|_2^2}, \cdots, \frac{1}{\|\mathbf{h}_{r_k\star}\|_2^2}\} \tag{2.11}$$

and

$$\nabla l(\mathbf{S}, \mathbf{B}^t) = -\mathbf{S}\mathbf{H}^\top + \mathbf{B}^t\mathbf{H}\mathbf{H}^\top.$$

By adding $\lambda\|\mathbf{B}\|_{2,1} + C$ to the both sides of (2.10), one has

$$F(\mathbf{B}) \leq G(\mathbf{B}, \mathbf{B}^t), \tag{2.12}$$

where

$$G(\mathbf{B}, \mathbf{B}^t) = l(\mathbf{S}, \mathbf{B}^t) + tr((\mathbf{B} - \mathbf{B}^t)^\top \nabla l(\mathbf{S}, \mathbf{B}^t)) + \frac{1}{2a}\|\mathbf{B} - \mathbf{B}^t\|_F^2 + \lambda\|\mathbf{B}\|_{2,1} + C, \tag{2.13}$$

i.e., $F(\mathbf{B})$ is majorized by $G(\mathbf{B}, \mathbf{B}^t)$.

It is easy to see that minimizing (2.13) is equivalent to minimizing (2.4). Without loss of generality, taking the derivative of (2.13) with respect to $\boldsymbol{\beta}_i$, i.e. the i^{th} column of $\mathbf{B}$, one has

$$\nabla G(\boldsymbol{\beta}_i, \boldsymbol{\beta}_i^t) = \frac{1}{a}(\boldsymbol{\beta}_i - \tilde{\boldsymbol{\beta}}_i^t) + \lambda\mathbf{u}_i, \tag{2.14}$$

where $\tilde{\boldsymbol{\beta}}_i^t = \boldsymbol{\beta}_i^t - a\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)$ and $\mathbf{u}_i$ is given in (2.6). Letting $\nabla G(\boldsymbol{\beta}_i, \boldsymbol{\beta}_i^t) = \mathbf{0}$ and with some algebra, we have $\boldsymbol{\beta}_i = \mathbf{0}$ if

$$\|\tilde{\boldsymbol{\beta}}_i^t\|_2 \leq a\lambda, \tag{2.15}$$

and otherwise

$$\boldsymbol{\beta}_i = \tilde{\boldsymbol{\beta}}_i^t \left(1 + \frac{a\lambda}{\|\boldsymbol{\beta}_i\|_2}\right)^{-1}. \tag{2.16}$$

Taking L_2-norm on both sides of (2.16) and combine the result of (2.15), one obtains

$$\|\boldsymbol{\beta}_i\|_2 = (\|\tilde{\boldsymbol{\beta}}_i^t\|_2 - a\lambda)_+, \tag{2.17}$$

where $(\cdot)_+$ is an operator that maps negative entries to zeros. Plug (2.17) into (2.16), one can have

$$\begin{aligned} \boldsymbol{\beta}_i &= \tilde{\boldsymbol{\beta}}_i^t \left[1 + \frac{a\lambda}{(\|\tilde{\boldsymbol{\beta}}_i^t\|_2 - a\lambda)_+}\right]^{-1} \\ &= \tilde{\boldsymbol{\beta}}_i^t \left(1 - \frac{a\lambda}{\|\tilde{\boldsymbol{\beta}}_i^t\|_2}\right)_+. \end{aligned} \tag{2.18}$$

The uniqueness solution of $\boldsymbol{\beta}_i$ in (2.16) is also shown in Appendix IV.

Proposition 1. (2.18) is the unique solution of (2.16).

Now we combine the updating rules with all columns of $\mathbf{B}$, which can be expressed as

$$\mathbf{B} = \tilde{\mathbf{B}}^t(\mathbf{I} - a\lambda\boldsymbol{\Sigma})_+, \tag{2.19}$$

where $\tilde{\mathbf{B}}^t = \mathbf{B}^t - a\nabla l(\mathbf{S}, \mathbf{B}^t) = [\tilde{\boldsymbol{\beta}}_1^t, \tilde{\boldsymbol{\beta}}_2^t, \cdots, \tilde{\boldsymbol{\beta}}_n^t]$ and

$$\boldsymbol{\Sigma} = \begin{pmatrix} \frac{1}{\|\tilde{\boldsymbol{\beta}}_1^t\|_2} & \cdots & 0 \\ \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\|\tilde{\boldsymbol{\beta}}_n^t\|_2} \end{pmatrix}$$

is a diagonal matrix. The updating rule (2.19) can be computed efficiently since $(\mathbf{I} - a\lambda\mathbf{\Sigma})_+$ is a diagonal matrix and some diagonal entries may be zero.

2.4 META-LEARNING VIA SGLASSO

In this section, we apply our proposed SGLasso model to meta-learning for automatic configuration recommendations. Suppose that there are m configurations in the search space $\mathbf{\Omega}_{\mathscr{A}}$ of the investigated algorithm $\mathscr{A}$, namely, $\mathbf{\Omega}_{\mathscr{A}} = \{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m\}$, then an m-dimensional classification performance vector on a historical dataset $\mathscr{D}$, denoted by $\mathbf{y} = [y_1, y_2, \cdots, y_m]^\top$, can be obtained, in which each entry y_i represents the classification performance of $\mathscr{A}$ instantiated with configuration $\boldsymbol{\omega}_i$. Let f be a meta-feature extraction function, then $\mathbf{x} = [x_1, x_2, \cdots, x_r]^\top = f(\mathscr{D})$ is the extracted meta-feature vector of $\mathscr{D}$. How to collect this metadata will be explained in the next section. Now suppose that there are n historical datasets $\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_n$, then a response matrix $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n]$ and a meta-feature matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$ can be obtained, where $\{\mathbf{x}_i, \mathbf{y}_i\}$ is the metadata of dataset $\mathscr{D}_i$.

2.4.1 SGLasso training

The training procedures of SGLasso are summarized in `Algorithm 1` where the superscript j refers to the j^{th} iteration. To show the alternative updating process, we adopt the notations used in formulae (2.2) and (2.3). The group index vector $\boldsymbol{\gamma}$ has the same dimension as $\mathbf{x}$, i.e., $\boldsymbol{\gamma} \in \mathbb{R}^r$, and each entry of γ specifies which group the meta-feature belongs to. In real scenarios, we can either generate $\boldsymbol{\gamma}$ randomly or determine it based on the natural grouping properties of meta-features. In step 4, we use $a_l = \frac{1}{10}\min\{\frac{1}{\|\mathbf{x}^l_{1\star}\|_2^2}, \frac{1}{\|\mathbf{x}^l_{2\star}\|_2^2}, \cdots, \frac{1}{\|\mathbf{x}^l_{r_l\star}\|_2^2}\}$, $l = 1, 2, \cdots, p$, for each group, where $\mathbf{x}^l_{j\star}$, $j = 1, 2, ..., r_l$, is the j^{th} row of $\mathbf{X}_l$, based on the

Algorithm 1 `SGLasso`($\mathbf{Y}, \mathbf{X}, \lambda, p, \boldsymbol{\gamma}$, `maxL`)

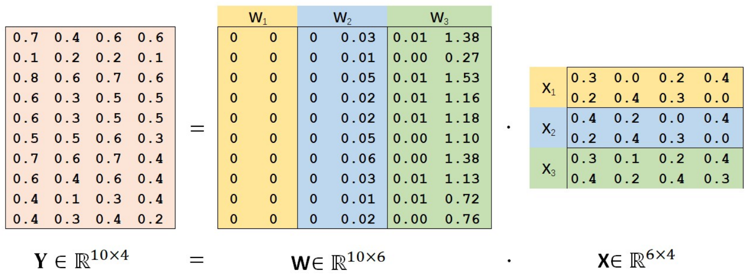
```
Input: Response matrix $\mathbf{Y}$, meta-features $\mathbf{X}$, tuning parameter $\lambda$, number of groups $p$, group index $\boldsymbol{\gamma}$, maximal allowable number of iterations maxL
Output: Estimated effects: $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p]$
 1: Categorize $\mathbf{X}$ into $p$ groups $\{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_p\}$ using $\boldsymbol{\gamma}$
 2: Initialize $\mathbf{W}^0 = [\mathbf{W}_1^0, \mathbf{W}_2^0, \cdots, \mathbf{W}_p^0]$, where $\mathbf{W}_l^0 \neq \mathbf{0}$, $l = 1, 2, \cdots, p$
 3: Initialize $\boldsymbol{\Theta}_l^0 = \mathbf{W}_l^0, l = 1, 2, \cdots, p$
 4: Calculate $a_l, l = 1, 2, \cdots, p$
 5: for $j = 1$ : maxL do
 6:     for $l = 1 : p$ do
 7:         compute gradient: $\mathbf{G}_l^{j-1} = \nabla l(\mathbf{S}_{-l}, \mathbf{W}_l^{j-1})$
 8:         update $\tilde{\mathbf{W}}_l^j \leftarrow \mathbf{W}_l^{j-1} - a_l \mathbf{G}_l^{j-1}$
 9:         update $\boldsymbol{\Theta}_l^j$ by $\boldsymbol{\Theta}_l^j \leftarrow \tilde{\mathbf{W}}_l^j (\mathbf{I} - a_l \lambda \boldsymbol{\Sigma})_+$
10:         update the center via a Nesterov step by
              $\mathbf{W}_l^j \leftarrow \boldsymbol{\Theta}_l^j + \frac{j}{j+3}(\boldsymbol{\Theta}_l^j - \boldsymbol{\Theta}_l^{j-1})$
11:     end for
12: end for
```

local steepest descent along with the objective function. In step 10, we apply the Nesterov momentum to accelerate the convergence process.

Within the inner loop, i.e., from step 7 to step 10, the most computationally expensive part is the calculation of the gradient in step 7. Recall that

$$\begin{aligned}\nabla l(\mathbf{S}_{-l}, \mathbf{W}_l) &= -\mathbf{S}_{-l}\mathbf{X}_l^\top + \mathbf{W}_l\mathbf{X}_l\mathbf{X}_l^\top \\ &= (-\mathbf{S}_{-l} + \mathbf{W}_l\mathbf{X}_l)\mathbf{X}_l^\top \\ &= (-\mathbf{Y} + \mathbf{W}\mathbf{X})\mathbf{X}_l^\top,\end{aligned} \tag{2.20}$$

which has the computational complexity of $O(mnr)$ where m, n, and r are the number of candidate configurations, number of historical datasets, and number of measures, respectively. In fact, $\mathbf{W}$ usually contains zero columns, which reduces the computational complexity to $O(mnr')$ where $r' < r$ is the number of nonzero columns of $\mathbf{W}$.



$$\begin{array}{cccc} 0.7 & 0.4 & 0.6 & 0.6 \\ 0.1 & 0.2 & 0.2 & 0.1 \\ 0.8 & 0.6 & 0.7 & 0.6 \\ 0.6 & 0.3 & 0.5 & 0.5 \\ 0.6 & 0.3 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.6 & 0.3 \\ 0.7 & 0.6 & 0.7 & 0.4 \\ 0.6 & 0.4 & 0.6 & 0.4 \\ 0.4 & 0.1 & 0.3 & 0.4 \\ 0.4 & 0.3 & 0.4 & 0.2 \end{array} = \begin{array}{cc|cc|cc} \multicolumn{2}{c|}{W_1} & \multicolumn{2}{c|}{W_2} & \multicolumn{2}{c}{W_3} \\ \hline 0 & 0 & 0 & 0.03 & 0.01 & 1.38 \\ 0 & 0 & 0 & 0.01 & 0.00 & 0.27 \\ 0 & 0 & 0 & 0.05 & 0.01 & 1.53 \\ 0 & 0 & 0 & 0.02 & 0.01 & 1.16 \\ 0 & 0 & 0 & 0.02 & 0.01 & 1.18 \\ 0 & 0 & 0 & 0.05 & 0.00 & 1.10 \\ 0 & 0 & 0 & 0.06 & 0.00 & 1.38 \\ 0 & 0 & 0 & 0.03 & 0.01 & 1.13 \\ 0 & 0 & 0 & 0.01 & 0.01 & 0.72 \\ 0 & 0 & 0 & 0.02 & 0.00 & 0.76 \end{array} \cdot \begin{array}{c|cccc} X_1 & 0.3 & 0.0 & 0.2 & 0.4 \\ & 0.2 & 0.4 & 0.3 & 0.0 \\ \hline X_2 & 0.4 & 0.2 & 0.0 & 0.4 \\ & 0.2 & 0.4 & 0.3 & 0.0 \\ \hline X_3 & 0.3 & 0.1 & 0.2 & 0.4 \\ & 0.4 & 0.2 & 0.4 & 0.3 \end{array}$$

$$\mathbf{Y} \in \mathbb{R}^{10\times4} \quad = \quad \mathbf{W} \in \mathbb{R}^{10\times6} \quad \cdot \quad \mathbf{X} \in \mathbb{R}^{6\times4}$$

Figure 2.1: An illustrating example that shows the training process of SGLasso. $\mathbf{Y}, \mathbf{X}$ and $\mathbf{W}$ represent the historical performance, meta-features, and estimated outcome, respectively.

2.4.2 An illustrating example

We here present an illustrating but simple example to show the effectiveness of our proposed SGLasso approach. Suppose that there are four historical problems and we extract the historical performance $\mathbf{Y} \in \mathbb{R}^{10\times4}$ and the meta-features $\mathbf{X} \in \mathbb{R}^{6\times4}$, respectively, see Figure 2.1. We partition the six meta-features into three groups, namely, $\boldsymbol{\gamma} = [1, 1, 2, 2, 3, 3]$, see Figure 2.1 (right) where each color indicates a group of $\mathbf{X}$. Then we compute $a_l, l = 1, 2, 3$, and we obtain $a_1 = 0.2990, a_2 = 0.2898$, and $a_3 = 0.2257$. We set $\lambda = 0.4$ and `maxL`$= 200$. When the training is completed under our proposed algorithm, we obtain the estimated $\mathbf{W}$ (in the middle of Figure 2.1). One can observe that $\mathbf{W}_1$ is a zero matrix and the first column of $\mathbf{W}_2$ is also zero, while $\mathbf{W}_3$ is dense, which indicates that the first three meta-features are assumed to be redundant and thus will be removed from the model. Taking a closer look into $\mathbf{X}$, we can see that the first and third rows of $\mathbf{X}$ are highly correlated to the fifth and sixth rows, respectively, and the second and fourth rows are exactly the same. Thus, the proposed SGLasso successfully identifies the correlated information of meta-features and provides the desirable solution.

2.4.3 Meta-feature selection

Our proposed SGLasso model can control both between-group and within-group sparsity by appropriately setting the tuning parameter λ. After the `maxL` iterations, the coefficient matrix $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p]$ can be obtained, which contains zero columns as expected. Let $\mathbf{v} \in \mathbb{R}^{r'}$ be a vector containing the indices of the nonzero columns of $\mathbf{W}$ where $r' < r$. Then $\mathbf{v}$ contains the indexes of the selected meta-features. The relationship between the historical performance and the selected meta-features over the n historical datasets can be rewritten as

$$\mathbf{Y} = \mathbf{W}_{[:,\mathbf{v}]}\mathbf{X}_{[\mathbf{v},:]}, \tag{2.21}$$

where $\mathbf{W}_{[:,\mathbf{v}]}$ ($\mathbf{X}_{[\mathbf{v},:]}$) represent the sub-matrices acquired by taking columns (rows) indexed by the index vector $\mathbf{v}$. Three advantages of meta-feature selection are ready to see: firstly, it reduces the number of the predictors so the complexity of the model can be largely reduced [68]; secondly, by removing those irrelevant or redundant meta-features for a specific problem, the model accuracy has been enhanced; thirdly, we only need to extract the selected meta-features from a new dataset, yielding the efficiency improvement for the online phase.

2.4.4 Performance prediction

For a new dataset $\mathscr{D}$ of interest, we can estimate the performance of algorithm $\mathscr{A}$ over the configuration space $\mathbf{\Omega}_{\mathscr{A}}$ via the trained SGLasso model. We are first to extract the meta-features of $\mathscr{D}$, denoted as $\mathbf{x}_{\mathbf{v}} = f_{\mathbf{v}}(\mathscr{D})$, where $\mathbf{x}_{\mathbf{v}} \in \mathbb{R}^{r'}$, and $f_{\mathbf{v}}$ means to extract the meta-features indexed by $\mathbf{v}$ only. Then the estimated performance is given by

$$\mathbf{y}_{\mathscr{D}} = \mathbf{W}_{[:,\mathbf{v}]}\mathbf{x}_{\mathbf{v}}, \tag{2.22}$$

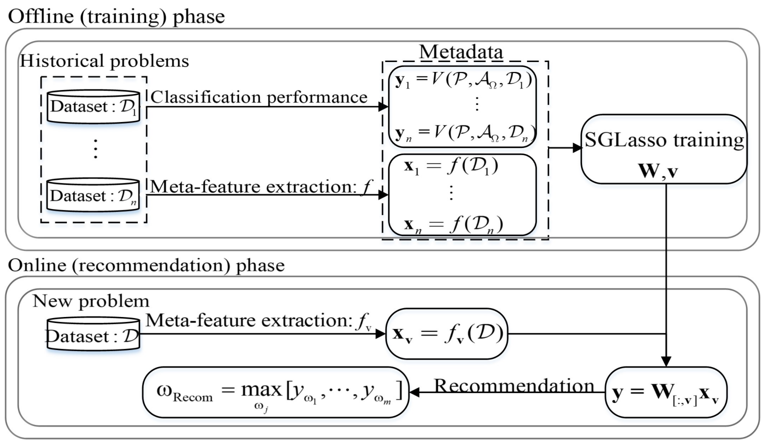

Offline (training) phase
Historical problems
Dataset : $\mathcal{D}_1$
Dataset : $\mathcal{D}_n$
Classification performance
Meta-feature extraction: f
Metadata
$\mathbf{y}_1 = V(\mathcal{P}, \mathcal{A}_\Omega, \mathcal{D}_1)$
$\mathbf{y}_n = V(\mathcal{P}, \mathcal{A}_\Omega, \mathcal{D}_n)$
$\mathbf{x}_1 = f(\mathcal{D}_1)$
$\mathbf{x}_n = f(\mathcal{D}_n)$
SGLasso training
$\mathbf{W}, \mathbf{v}$
Online (recommendation) phase
New problem
Dataset : $\mathcal{D}$
Meta-feature extraction: $f_\mathbf{v}$
$\mathbf{x}_\mathbf{v} = f_\mathbf{v}(\mathcal{D})$
$\omega_{\text{Recom}} = \max_{\omega_j}[y_{\omega_1}, \cdots, y_{\omega_m}]$
Recommendation
$\mathbf{y} = \mathbf{W}_{[:,\mathbf{v}]}\mathbf{x}_\mathbf{v}$

Figure 2.2: The automatic configuration recommendation architecture based on SGLasso model, where $V(\cdot,\cdot,\cdot)$ is the adopted 10-fold stratified cross-validation, and $\mathcal{P}$ is the balanced classification accuracy.

where $\mathbf{y}_\mathscr{D} \in \mathbb{R}^m$ and $(y_\mathscr{D})_j, j = 1, 2, \cdots, m$, is the predictive performance of $\boldsymbol{\omega}_j$ in $\boldsymbol{\Omega}_\mathscr{A}$.

2.5 AUTOMATIC CONFIGURATION RECOMMENDATION

The specific details of our proposed SGLasso-based automatic configuration recommendation architecture are shown in Figure 2.2. This architecture consists of two phases, namely, the offline (training) phase and the online (recommendation) phase.

2.5.1 Offline phase

The important ingredients of the offline phase are metadata extraction on a collection of historical datasets and SGLasso training. We next explain how to collect the metadata.

Classification performance evaluation

Suppose there are n historical datasets $\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_n$ and m configurations, namely, $\boldsymbol{\Omega}_\mathscr{A} = \{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \cdots, \boldsymbol{\omega}_m\}$. Then the historical performance matrix $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n] \in \mathbb{R}^{m\times n}$ can be obtained using 10-fold cross-validation, in which the stratified dataset splitting is considered to eliminate the influence of class imbalance that commonly existed in the real datasets. For the same reason, the employed classification performance metric is balanced

Algorithm 2 `Performance-evaluation`$(\mathscr{A}, \mathscr{D}_j, \boldsymbol{\omega}, \mathcal{P})$

Input: Classification algorithm $\mathscr{A}$, dataset $\mathscr{D}_j$, hyperparameter configuration $\boldsymbol{\omega}$, classification performance metric $\mathcal{P}$

Output: Evaluated performance: $y_{\boldsymbol{\omega}}$

```
1: Stratified dataset split: $\mathscr{D}_j = \{\mathscr{D}_{j1}, \mathscr{D}_{j2}, \cdots, \mathscr{D}_{j10}\}$
2: for $i$=1:10 do
3:     $\mathscr{D}_{test} = \mathscr{D}_{ji}$
4:     $\mathscr{D}_{train} = \cup_{k \neq i}^{10} \mathscr{D}_{jk}$
5:     $y_i = \mathcal{P}(\mathscr{A}_{\boldsymbol{\omega}}, \mathscr{D}_{train}, \mathscr{D}_{test})]$
6: end for
7: $y_{\boldsymbol{\omega}} = \frac{1}{10} \sum_{i=1}^{10} y_i$
```

classification accuracy. The details are elaborated on in the following.

Given a classification algorithm $\mathscr{A}$ with a configuration $\boldsymbol{\omega}_i$ and a dataset $\mathscr{D}_j, j = 1, 2, \cdots, n$, we first split $\mathscr{D}_j$ into 10 folds evenly as $\mathscr{D}_j = \{\mathscr{D}_{j1}, \cdots, \mathscr{D}_{j10}\}$. Here, we employ the stratified cross-validation strategy to ensure every fold contains samples that are from all the classes when the sizes of classes are unbalanced. Then a fold $\mathscr{D}_{ji}$ is selected as testing set $\mathscr{D}_{test}$ and the remaining 9 folds are used as the training set $\mathscr{D}_{train}$. Thus we can obtain the classification performance as

$$y_{\boldsymbol{\omega}_i} = \mathcal{P}(\mathscr{A}_{\boldsymbol{\omega}_i}, \mathscr{D}_{train}, \mathscr{D}_{test}),$$

where $\mathcal{P}(\cdot, \cdot, \cdot)$ represents the chosen classification performance metric, i.e., balanced classification accuracy, and $y_{\boldsymbol{\omega}_i}$ is the performance of $\mathscr{A}_{\boldsymbol{\omega}_i}$ trained on $\mathscr{D}_{train}$ and evaluated on $\mathscr{D}_{test}$. Repeat this procedure ten times such that each fold $\mathscr{D}_{ji}$ can be used as $\mathscr{D}_{test}$ and the remaining folds can be used as $\mathscr{D}_{train}$. In the end, a 10-dimensional performance vector is formed. The entry-wise mean of this vector is obtained as the final performance of $\mathscr{A}_{\boldsymbol{\omega}_i}$ on dataset $\mathscr{D}_j$. The whole process is depicted in `Algorithm 2`.

Table 2.1: The existing meta-features and the number of (adopted) measures.

Names	#measures (adopted)
Statistics&Information-theory (SIT)	73 (54)
Model structure (MS)	24 (24)
Problem complexity (PC)	35 (23)
Landmarking (LM)	14 (14)
Relative landmarking (RLM)	14 (14)
Structural information (SI)	14 (11)
Concept (CON)	8 (7)
Clustering (CLU)	8 (8)
Total	190 (155)

Meta-feature extraction

The extraction of meta-features involves the calculations of the measures defined in each meta-feature. There are eight types of meta-features that are widely used in meta-learning in practices [56, 57]. These meta-features focus on different aspects, such as statistics & information theory (SIT), landmarking (LM), relative landmarking (RLM), model structure (MS), structural information (SI), classification complexity (PC), clustering (CLU), and concept (CON). The meta-feature used by our SGLasso is the union of the eight meta-features.

In our approach, Python package `pymfe` [23] is used to extract these meta-features, and then several preprocessing tricks are applied to the raw meta-features: 1) every measure is normalized into $[0, 1]$ using min-max scaling; 2) some measures that contain only one value throughout all datasets or contain too many missing values are deleted; 3) we also delete those repetitive measures that have the exactly the same values and only one is kept. Therefore, the number of measures of each type of meta-feature adopted in the experiments is reduced, as shown in the parentheses in Table 2.1.

Here we study the multicollinearity existing in the aforementioned meta-features. Following the detection methods suggested by Mansfield et al. [69], we first extract each type of meta-features from 136 UCI datasets and obtain 136 meta-feature vectors accordingly, then we compute their correlation matrix separately after the meta-feature preprocessing, and calculate the eigenvalues of each correlation matrix. We display the results in Figure 2.3, from which one can see that some eigenvalues of the eight meta-features are quite small (close to 0), which indicates that certain variables of the meta-features are highly correlated. Therefore, the linear regression model without the meta-feature selection usually does not generate desirable performance due to the ill-conditioned problem that occurred.

2.5.2 Recommendation

We have shown that the performance of $\mathscr{A}$ on a new dataset $\mathscr{D}$ can be estimated by (2.22). Then the promising configurations can be recommended based on the order of the predictive performance. Detailedly, we sort $\mathbf{y}_{\mathscr{D}} = [y_{\boldsymbol{\omega}_1}, y_{\boldsymbol{\omega}_2}, \cdots, y_{\boldsymbol{\omega}_m}]^\top$ in descending order as

$$[y_{\boldsymbol{\omega}_{i_1}}, y_{\boldsymbol{\omega}_{i_2}}, \cdots, y_{\boldsymbol{\omega}_{i_m}}],$$

where $y_{\boldsymbol{\omega}_{i_1}} \geq y_{\boldsymbol{\omega}_{i_2}} \geq \cdots \geq y_{\boldsymbol{\omega}_{i_m}}$. Therefore, the configuration that has the highest predictive value is recommended for this new dataset $\mathscr{D}$, i.e., $\boldsymbol{\omega}_{Recom} = \boldsymbol{\omega}_{i_1}$, or

$$\boldsymbol{\omega}_{Recom} = \boldsymbol{\Omega}_{\mathscr{A}}\{i_1\}, \tag{2.23}$$

where $\boldsymbol{\Omega}_{\mathscr{A}}\{i_1\}$ stands for the ${i_1}^{th}$ element of the configuration space $\boldsymbol{\Omega}_{\mathscr{A}}$.

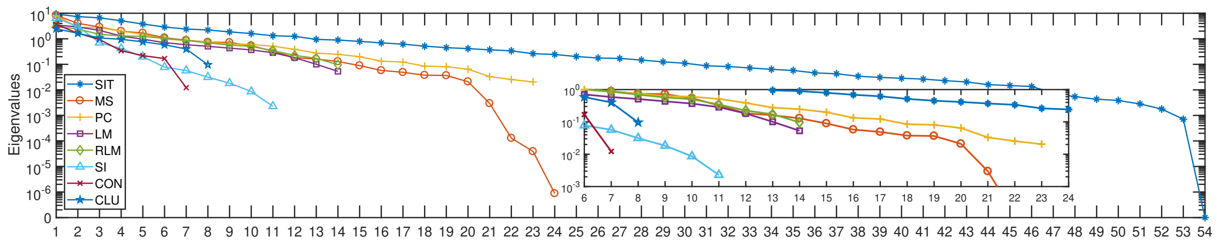


Figure 2.3: The problem of multicollinearity existed in the eight types of meta-features. Their eigenvalues being close to 0 imply that certain variables of meta-features are highly correlated.

2.6 EXPERIMENTS

In this section, we validate the recommendation effectiveness of our proposed model. We first present our experimental settings, including adopted datasets, classification algorithm with its configuration space, as well as various comparative methods with suitable evaluation metrics, and then present the empirical results and provide further elaboration.

2.6.1 Experimental setup

Datasets: We collect 136 classification datasets from the UCI machine learning repository [70], in which the smallest dataset contains 88 instances and the maximal sample size is about 30K; the number of attributes ranges from 3 to 1558, and the number of classes ranges from 2 to 15. The statistical information in terms of the number of instances, classes, and dimensions of the adopted datasets can be found in Appendix I. In experiments, 60, 20, and 56 datasets are randomly chosen as the training set, validation set, and testing set, respectively.

The raw datasets are initially processed by: 1) normalizing each attribute into the range [-1,1] using the min-max scaling to eliminate the difference among the domains of attributes; 2) removing samples that contain too many missing values and that are repetitive. Also, the attributes of some datasets that contain only one value are deleted; 3)removing the classes that have less than 10 instances from the datasets before implementing 10-fold stratified cross-validation.

Classification algorithm and configuration space: we evaluate our proposed configuration recommendation algorithm on the well-known SVM, due to its excellent classification performance shown in many real applications while its configuration evaluations, especially for large-scale datasets, are usually quite time-consuming.

Here, we consider SVM using RBF kernel because of the flexibility of kernel parameter setting. Thus, there are two numerical hyperparameters that should be determined: the regularization coefficient '`C`' and the RBF kernel width '`gamma`'. Following the guideline provided in [71], we preset `C` $\in 2^{-5:1:15}$ and `gamma` $\in 2^{-15:1:3}$. These two settings are effective for many datasets. Therefore, there are $21 \times 19 = 399$ points in total in the configuration space of SVM. In the experiment, we adopt the SVM software developed in `scikit-learn` library [72].

Comparative baselines: Three meta-learning baselines in literature are evaluated:

1. KNN [43] where Euclidean distance is used as the similarity measurement;
2. Kernel KNN (KKNN) [13, 55], and the performance estimation of a dataset $\mathscr{D}$ is given by $\mathbf{y}_{\mathscr{D}} = \frac{1}{S}\sum_{j=1}^{n} k(\mathbf{x}_j, \mathbf{x}_{\mathscr{D}})\mathbf{y}_j$ where $S = \sum_{j=1}^{n} k(\mathbf{x}_j, \mathbf{x}_{\mathscr{D}})$, and $k(\cdot,\cdot)$ is RBF kernel;
3. Multilayer perceptron (MLP) [22, 73]. The network structure is r-100-200-399. We select sigmoid as the activation function with the L_2 regularizer being added to avoid over-fitting.

We also compare our approach with multivariate Lasso (MLasso), the special case of (2.2) when $p = 1$, and multivariate sparse-group Lasso (MSGL) [66] to show the superiority of our feature selection model. All comparative approaches adopt the recommendation strategy proposed in (2.23).

On the other hand, we compare our method with four search-based algorithms: random search (RS), Bayesian optimization (BO), heteroscedastic and evolutionary Bayesian optimization (HEBO) [34], and Hyperband. During the configuration search process, 10-fold cross-validation is used to evaluate each set of configurations selected by search algorithms. Besides, the performance of the default setting of SVM used by `scikit-learn` [1] is also given to evaluate our approach.

Evaluation metrics: Three commonly used metrics in meta-learning are chosen for evaluating the performance of the recommendation approaches, namely, (average) classification accuracy rate (CA, ACA), (average) recommendation accuracy rate (RA, ARA), and hit rate (HR). Their definitions can be found in Appendix III. The ranges of all metrics are $[0, 1]$ and a larger value means the recommended configuration is closer to the real best one and thus it is preferred.

2.6.2 Experimental reports and elaborations

Our experimental reports consist of three parts. The first part shows the ability of meta-feature selection of SGLasso, and the second and the third parts compare the recommendation capacity between SGLasso and other meta-learning baselines as well as four classical search algorithms. The best parameters for SGLasso and the comparative baselines are determined by handout validation, i.e., evaluate the validation set on the model built on the training set and candidate parameters, the parameter that has the highest total perfor-

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

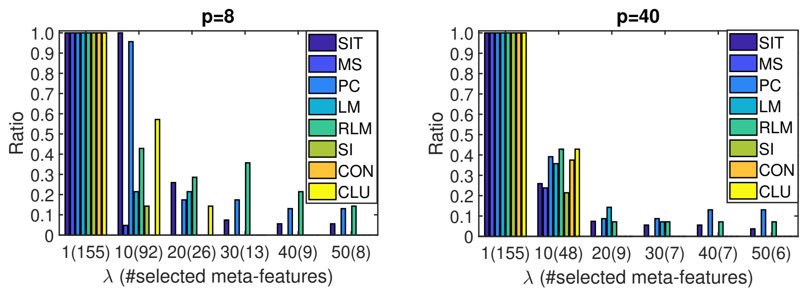


Figure 2.4: The importance of meta-features when different λ values are applied. The numbers of the selected meta-features are given in the parenthesizes on the horizontal axis. A ratio being 0 means the corresponding meta-feature is completely removed from the model.

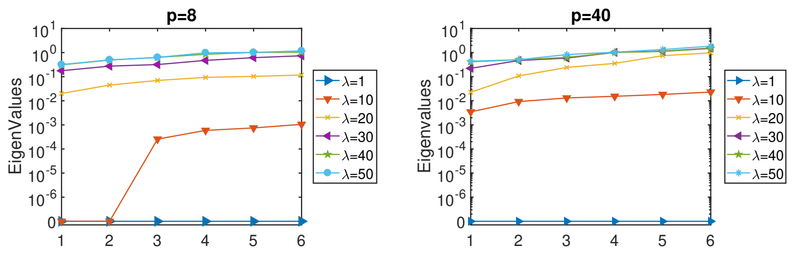


Figure 2.5: The 6 smallest eigenvalues of the correlation matrix generated by the selected mete-feature for each λ. Eigenvalues that are close to zero indicate a very high degree of multicollinearity existed in the meta-feature.

mance, i.e., ACA+ARA+HR, is leveraged, and the results of the testing set evaluated on the best model are reported.

Effectiveness of meta-feature selection

This subsection studies the SGLasso model based on three important questions. Firstly, which meta-features are essential for model construction? Secondly, can the problem of multicollinearity be reduced by meta-feature selection? Thirdly, how the recommendation performance of SGLasso will be affected by meta-feature selection?

To show the importance of each type of meta-feature for the investigated meta-model, the tuning parameter $\lambda = \{1, 10, 20, 30, 40, 50\}$ is set as an increasing sequence. As we discussed, a larger λ leads to a higher degree of sparsity of $\mathbf{W}$, and thus fewer measures

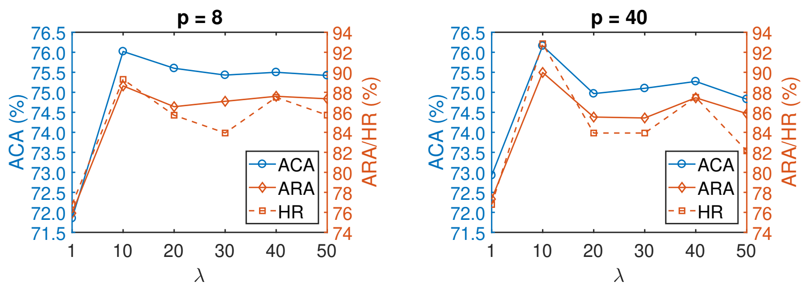


Figure 2.6: The recommendation performance fluctuation of SGLasso when different degrees of sparsity are applied.

would be retained. To train the SGLasso model, we set $p = 40$, i.e., the eight types of meta-features are randomly partitioned into 40 groups, and `maxL`$= 300$. Here $p = 40$ is determined by the grid search and it produces the best recommendation performance as shown in the following subsection. The results are summarized in Figure 2.4, where each data is the ratio between the number of the retained measures and the number of the total measures for an individual meta-feature. The number of the selected meta-features for each λ is given in the parenthesis below the horizontal axis. For the reason of comparison, we also provide results when $p = 8$ where each type of meta-feature is a group to see whether grouping strategy will affect the model construction of SGLasso.

One can see that for $p = 40$, no measures are deleted when $\lambda = 1$, but there are only 48 of them left when $\lambda = 10$, and the amounts are reducing sharply when $\lambda \geq 20$, and there are only 6 measures when $\lambda = 50$. One can also observe that SIT, PC, LM, and RLM seem to provide the most important predictor variables for our meta-regression problem since others are totally removed from the model as λ increases. We can draw similar conclusions for $p = 8$, which indicates that the grouping strategy applied to the SGLasso model does not affect the conclusion about which type of meta-feature mainly describes the model. This is a fine property that evidences the model's reliability.

The second discussion is to see if the variable selection of SGLasso can reduce the multicollinearity as we showed previously. To save space, we only show the 6 smallest eigenvalues of the correlation matrix obtained from the selected meta-features for each λ in Figure 2.5. Again, we show the results of both $p = 40$ and $p = 8$. As expected, variables are severely correlated when $\lambda = 1$ as many eigenvalues are close to zero; however, this situation is alleviated as a smaller proportion of measures is selected as λ increases, which means our SGLasso can effectively delete those redundant and correlated measures and thus improve the model performance. In addition, the tendencies of $p = 40$ and $p = 8$ shown in Figure 2.5 are consistent, which again suggests the model stability of SGLasso. On the other hand, by comparing Figure 2.3 ($\lambda = 1$) and Figure 2.5, we can see that the combined meta-features are more severely correlated than every single type of meta-feature because more eigenvalues are zero (we show only 6 of them but the total number is 20), which implies that there are similar measures cross different meta-features and thus feature selection can remove these redundancies as expected.

Next, we show how the recommendation performance of SGLasso is affected when meta-feature selection is performed. The results of ACA, ARA, and HR are depicted in Figure 2.6, from which one can observe that feature selection indeed improves the recommendation performance: when all measures are applied ($\lambda = 1$), SGLasso has the mediocre performance with an ACA of 72.92%, an ARA of 77.31%, and an HR of 76.79% for $p = 40$, and has the performance with an ACA of 71.83%, an ARA of 75.92%, and an HR of 76.79% for $p = 8$; however, it has the smallest ACA (ARA, HR) of 74.83% $(85.43\%, 82.14\%)$ after the feature selection for $p = 40$, and has the smallest ACA (ARA, HR) of 75.43% $(86.56\%, 83.92\%)$ after the feature selection for $p = 8$. On the other hand, the optimal recommendation

Table 2.2: The parameter settings for SGLasso, MLasso, MSGL, KNN, KKNN, and MLP.

Approach	Parameter	Usage	Range	References
SGLasso	p	number of groups	$\{2,3,\cdots,50\}$	–
	λ	sparsity regularization coefficient	$\{1,10,20,30,40,50\}$	
MLasso	λ	sparsity regularization coefficient	$\{1,10,20,30,40,50\}$	[67]
MSGL	p	number of groups	$\{2,3,\cdots,50\}$	[66]
	λ	sparsity regularization coefficient	$\{1,2,\cdots,10,20,\cdots,100\}\times 10^{-4}$	
	λ_G	L_2 regularization coefficient	$\{1,2,\cdots,10,20,\cdots,100\}\times 10^{-4}$	
KNN	K	number of neighbors	$\{1,2,\cdots,50\}$	[43]
KKNN	γ	parameter of RBF kernel	{1e-3,1e-2,1e-1,1e0,1,2,3,4,5,1e1,2e1,5e1,1e2}	[13, 55]
MLP	λ	L_2 regularization coefficient	{1e-7,1e-6,1e-5,1e-4,1e-3,1e-2}	[22, 73]

Table 2.3: Comparisons on ACA, ARA, and HR between MLasso, MSGL, KNN, KKNN, MLP, and SGLasso. The maximal value of each row is highlighted.

Metrics	SGLasso	MLasso	MSGL	KNN	KKNN	MLP
ACA	**76.16**	74.77	75.07	74.74	75.11	74.70
ARA	**89.97**	85.95	87.22	85.54	87.00	85.31
HR	**92.86**	85.71	89.89	80.36	85.71	85.18

Table 2.4: The Wilcoxon Signed-Rank tests between MLasso, MSGL, KNN, KKNN, MLP, and SGLasso. Here">" represents "is better than". p-Values that are larger than 0.05 are underlined.

Alternative hypothesis	win/tie/lose	p-value (CA)	p-value (RA)
SGLasso>MLasso	36/4/16	0.002	0.013
SGLasso>MSGL	31/5/20	$\underline{0.088}$	0.000
SGLasso>KNN	31/3/22	$\underline{0.052}$	0.024
SGLasso>KKNN	35/3/18	0.007	0.025
SGLasso>MLP	38/0/18	0.001	0.018

occurs when $\lambda = 10$ for both grouping strategies, where ACA (ARA, HR) mounts to 76.16% $(89.97\%, 92.86\%)$ for $p = 40$ and 76.02% $(88.64\%, 89.29\%)$ for $p = 8$. As a comparison, the ACA of the real optimal configurations on the testing problems is 78.70%.

Comparisons with meta-learning baselines

This section discusses the recommendation capacity of SGLasso compared to other meta-learning baselines, such as KNN, KKNN, and MLP, as well as Lasso-based approaches, namely, MSGL and MLasso. The parameter search space of each approach is listed in Table 2.2. The outcomes of ACA, ARA, and HR are summarized in Table 2.3, and the comparisons on each single dataset are shown in Figure 2.7. Notice that the comparative baselines (KNN, KKNN, and MLP) use the complete meta-features, i.e., no feature selection is imposed as SGLasso does.

From Table 2.3, one can observe that: 1) meta-learning-based recommendation can choose the applicable configurations since the achieved lowest ACA, ARA, and HR are 74.70% (MLP), 85.31% (MLP), and 80.36% (KNN), respectively, which proves the promising ability of meta-learning for automatic configuration selection; 2) Our method, SGLasso, outperforms other approaches with an ACA of 76.16%, an ARA of 89.97%, and an HR of 92.86%. MSGL wins the second place, KKNN, MLasso and MLP take the tertiary, fourth, and fifth place, respectively, while KNN performs the worst; 3) SGLasso outweighs MLasso with a large gap, which implies the importance of the grouping strategy added into the model.

By looking into the performance difference on each dataset (Figure 2.7), one can see that SGLasso overpasses or performs equally with other baselines on most of the datasets, and this conclusion is more evident in terms of RA and HR, which testifies that CA alone is not enough to evaluate the recommendation performance. To be specific, SGLasso has a RA that is larger than 80% on 48 over 56 cases and has a RA that is larger than 90% on 37 over 56 cases, and these statistics are 42/28, 45/32, 39/32, 41/30, 37/27 for MLasso, MSGL, KNN, KKNN, and MLP, respectively; our method hits on 52 over 56 cases, and MLasso, MSGL, KNN, KKNN, and MLP respectively hit on 48, 50, 45, 48, and 47 out of 56 cases. All approaches fail to hit on the dataset `madelon-valid`, and only MSGL hits on dataset `fertility` and `spiral`.

In addition, to show if SGLasso performs statistically better than other methods, we adopt Wilcoxon signed-rank tests at a 0.05 significant level. The null hypothesis is that SGLasso performs worse than or equally with MLasso (or MSGL, KNN, KKNN, MLP). We test this on both CA and RA, and the testing results are displayed in Table 2.4, where the p-values that are less than 0.05 means that the null hypothesis is rejected and we accept the alternative hypothesis, that is, SGLasso outweighs significantly. We also give the statistics about on how many datasets SGLasso wins, ties, and loses. One can see that SGLasso is significantly better than MLasso, KKNN, and MLP on both CA and RA, and better than MSGL and KNN on RA. Although SGLasso fails to overpass MSGL and KNN on CA statistically, the p-values are close to 0.05. It is worth mentioning that KNN has a distinct manner in Table 2.3 and Table 2.4, i.e., KNN has the worst HR but it has high p-values. This is due to KNN having an unstable performance, i.e., it performs well on some datasets but it also does not generate satisfying recommendations on other datasets because of the difficulty of the selection of parameter `K`.

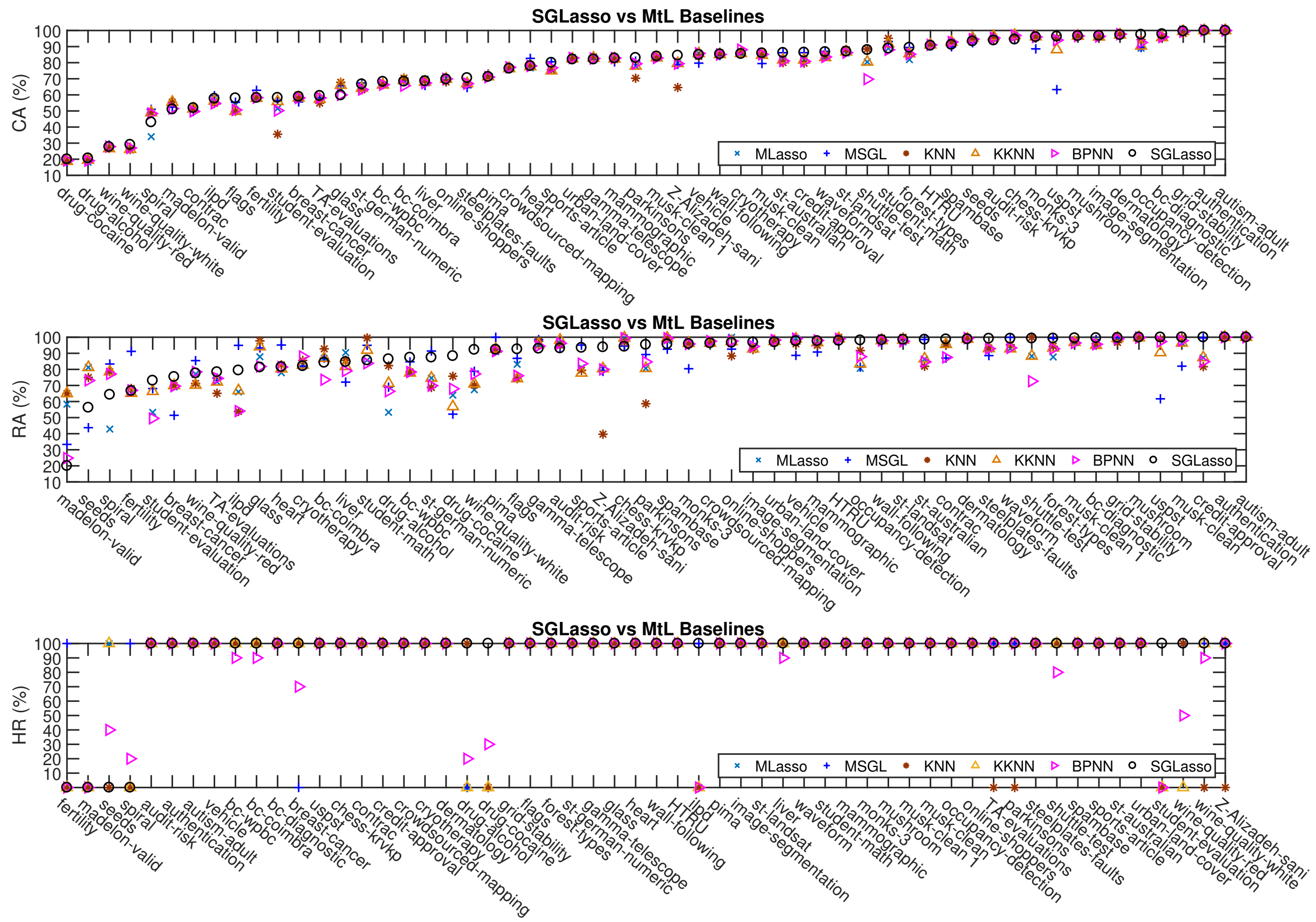


Figure 2.7: Comparisons on CA (top), RA (middle), and HR (bottom) between MLasso, MSGL, KNN, KKNN, MLP, and SGLasso on each single testing dataset.

Comparisons with search algorithms

In this section, we compare the effectiveness between SGLasso and search algorithms when the same time budget is allowed. The maximal time budget provided for search algorithms on each dataset is the time consumed by SGLasso for online applications, i.e., meta-feature extraction and performance estimation. Here we exclude the time of offline training since this can be done whenever needed. Notice that we set $\lambda = 10$ and $p = 40$, so only 48 out of 155 measures are extracted during the online phase. In a contrast, KNN is also compared to these search algorithms where KNN needs more time budget to extract the whole 155 measures. We select KNN because it has larger p-values in the statistical tests as given in Table 2.4. The comparative results are summarized in Table 2.5, and the details on each dataset are depicted in Figure 2.8 and 2.9. Because of the randomness, we repeat RS and Hyperband three times, and the average performance is reported.

From Table 2.5, one can see that: 1) the existing default configuration does not produce satisfying performance with an ACA of 72.59%, an ARA 77.09%, and an HR of 75.00%, which verifies the necessity of hyperparameter tuning; 2) RS, Hyperband, BO, and HEBO have the similar performance that is slightly better than the default; 3) there is no doubt that SGLasso outweighs both default setting and search algorithms with a wide gap. Looking into the per-dataset comparisons (Figure 2.8), we find that there are some cases where search algorithms surpass SGLasso, e.g., `madelon-valid`, `breast-cancer`, `heart`, and et al. These datasets have small dimensions, namely the small number of instances (attributes and classes), in which search algorithms are certainly more effective. As expected, SGLasso shows the superiority on those large-scale datasets, see `shuttle-test`, `online-shopper`,

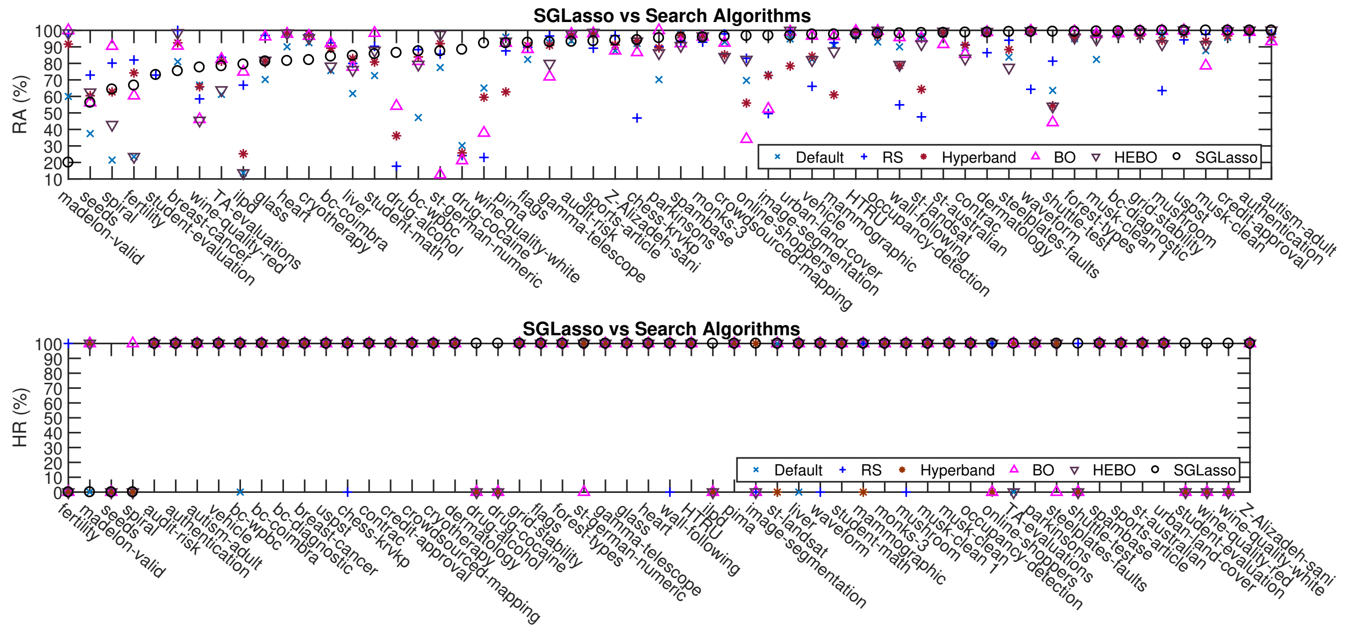


Figure 2.8: Comparisons on RA (top), and HR (bottom) between default, RS, Hyperband, BO, HEBO, and SGLasso on each single testing dataset.

Table 2.5: Comparisons on ACA, ARA, and HR between default, RS, Hyperband, BO, HEBO, and SGLasso (KNN). The maximal value of each row is highlighted.

Metrics	default	RS	Hyperband	BO	HEBO	SGLasso	RS	Hyperband	BO	HEBO	KNN
ACA	72.59	73.19	73.12	73.31	73.09	**76.16**	72.26	74.99	**75.95**	74.60	74.74
ARA	77.09	82.76	81.52	80.78	78.15	**89.97**	79.66	86.37	**89.57**	82.96	85.54
HR	75.00	75.00	76.79	76.79	78.57	**92.86**	80.36	80.36	**89.29**	82.14	80.36

`drug-cocaine`, etc., where cross-validation is time-consuming during each search. We also report the Wilcoxon signed-rank tests among the investigated approaches in Table 2.6, from which we can conclude that the superiority of our method over default, RS, Hyperband, BO, and HEBO is statistically significant.

However, the conclusions regarding KNN are different to some degree. Since the allowable time budget is increasing on each dataset, the performance of search algorithms has been improved, especially for BO, HEBO, and Hyperband. In Table 2.5, KNN outperforms default and RS, and it is inferior to Hyperband slightly, but BO outweighs KNN with a noticeable difference. The statistical tests in Table 2.6 also show that KNN fails to beat search algorithms, and BO is even significantly better than KNN. In Figure 2.9, it is clear to see that KNN is inferior to search algorithms on many datasets. These conclusions suggest that meta-learning without meta-feature selection is not efficient when the time-consuming extraction of meta-features is involved.

2.7 CONCLUDING REMARKS

The effective meta-learning mainly depends on the quality of meta-features, and thus the meta-feature selection appears to be critical for algorithm learning under the framework of machine learning. In this paper, by integrating features and algorithm learning as a whole, we develop a Lasso-type model, SGLasso, for the feature selection with the application to the automatic configuration recommendation. The main goal is to capture the intrinsic relationship between the dataset characteristics and the empirical performance through the development of a meta-learner, which can identify key meta-features and remove redundant or irrelevant ones. In the experiments, eight types of state-of-the-art meta-features are

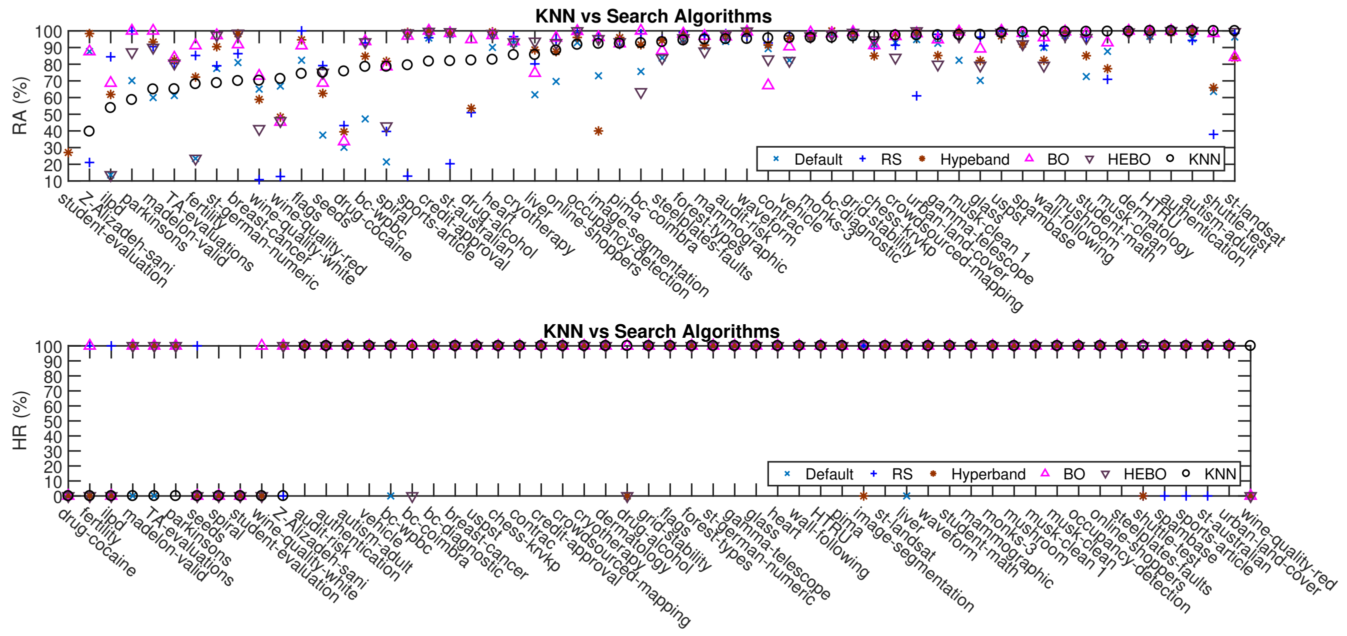


Figure 2.9: Comparisons on RA (top) and HR (bottom) between default, RS, Hyperband, BO, HEBO, and KNN on each single testing dataset.

Table 2.6: The Wilcoxon Signed-Rank tests between default, RS, Hyperband, BO, HEBO, and SGLasso (KNN).

Hypothesis	win/tie/lose	p-value (CA)	p-value (RA)	Hypothesis	win/tie/lose	p-value (CA)	p-value (RA)
SGLasso>default	45/1/10	0.000	0.000	KNN>default	39/1/16	0.003	0.002
SGLasso>RS	37/0/19	0.028	0.026	KNN>RS	24/1/31	0.694	0.706
SGLasso>Hyperband	39/0/17	0.003	0.001	KNN>Hyperband	25/1/30	0.436	0.460
SGLasso>BO	35/4/17	0.026	0.021	KNN>BO	19/4/33	0.965	0.965
SGLasso>HEBO	34/2/20	0.002	0.003	KNN>HEBO	25/3/28	0.821	0.787

adopted by our SGLasso to recommend configurations for SVM. The proposed SGLasso has shown superior performance that outperforms the existing meta-learning methods as well as the well-known search-based algorithms, such as random search, Bayesian optimization, and Hyperband. In addition, our approach is flexible and is ready to apply to other similar multivariate multiple regression problems where feature selection is involved with meta-learning .

CHAPTER 3

NONLINEAR META-FEATURE SELECTION

3.1 INTRODUCTION

Regarding feature selection, the least absolute shrinkage and selection operator, known as Lasso, has been widely studied. As an embedded method, its superior advantage over other wrapper or filter approaches lies in that the feature selection process is inlaid in the regression model training so that the computation burden can be alleviated. Quite often, the relationship between features and responses appears to be nonlinear in nature, and capturing the nonlinearity generally improves the performance for feature selection and regression estimation. Nevertheless, most of the existing Lasso models have been developed under the framework of linear approaches (see, e.g., a recent survey [74]). Kernelized Lasso, which is a nonlinear setting, can be used to perform the feature selection in the high-dimensional *Reproducing Kernel Hilbert Space*. However, the existing kernel Lasso approaches are not designated for meta-learning (see, e.g., [75, 76, 77] and references therein), and lack of the essential automatic learning process which is required to integrate the feature selection, the model, and the algorithm learning to form a meta-learner.

In this chapter, we develop a multivariate kernel group Lasso (KGLasso) approach that serves as both a meta-learner and meta-feature selector for the hyperparameter configuration recommendation under the framework of meta-learning. The significant difference between our proposed method and the current existing recommendation approaches lies in that our proposed KGLasso is able to automatically select the principal meta-features as well as the instances during the process of meta-learner training so that the most desirable configuration recommendations can be obtained. More specifically, we develop a KGLasso regression model on the metadata of historical datasets in which the meta-features are used as the predictors and the historical performance serves as the responses. To implement the KGLasso model, we establish an effective algorithm by constructing a corresponding auxiliary function so that convergence can be guaranteed. The trained model has the capacity to identify the critical meta-features and instances on which the performance of configurations for a new dataset can be based, boosting both efficiency and accuracy for recommendation tasks. Further, embedding with the proposed KGLasso, a meta-learning-based system is built for the configuration recommendation. In the experiments, we demonstrate its effectiveness in recommending configurations for SVM with desirable performance. Also, the extensive comparisons with other state-of-the-art recommendation algorithms as well as search algorithms over 120 UCI datasets further confirm the superiority of our KGLasso approach. Our main contributions in this chapter are summarized below.

1. A meta-feature selection model, called KGLasso, is developed under the nonlinear regression framework. It has the capacity to identify both meta-features and instances automatically. Our kernelization setting allows us to capture the correlated non-linearity between meta-features and instances. The algorithm associated with KGLasso is shown to be convergent mathematically.

2. Under the framework of meta-learning, a meta-learning-based system for configuration recommendation is established by embedding with the proposed KGLasso and the obtained system can provide competent configuration recommendation effectively.

3. Extensive experiments are conducted to demonstrate the effectiveness of our proposed

recommendation system, including the configuration recommendation for SVM, as well as comparisons with various search algorithms with standard evaluation metrics.

3.2 RELATED WORK

Motivated by the kernel support-vector regression, among others, Roth first discussed a generalized Lasso regression for a simple univariate regression problem [75]

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^{n\times 1}} \frac{1}{2}\|\mathbf{y}-\mathbf{K}\boldsymbol{\alpha}\|_2^2+\lambda\|\boldsymbol{\alpha}\|_1,$$

here $\mathbf{y}=[y_1,y_2,\cdots,y_n]^\top\in\mathbb{R}^{n\times 1}$ are the responses and $\mathbf{X}=[\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n]\in\mathbb{R}^{r\times n}$ are the predictors, where n is the number of training instances and r is the dimension of features, and $\mathbf{K}\in\mathbb{R}^{n\times n}$ is the kernel matrix generated on the predictor variables $\mathbf{X}$, and $K_{ij}=k(\mathbf{x}_i,\mathbf{x}_j)$ where $k(\cdot,\cdot)$ is a pre-defined kernel function. Its formulation focuses on selecting the training instances and is not applicable to feature selection. On the other hand, a feature-scaling kernel was proposed for the feature selection in [78, 79] in the form of

$$k_{\boldsymbol{\vartheta}}(\mathbf{x}_i,\mathbf{x}_j)=k(\mathbf{x}_i*\boldsymbol{\vartheta},\mathbf{x}_j*\boldsymbol{\vartheta}),$$

where $*$ denotes the standard Hardmard product, and $\boldsymbol{\vartheta}\in\{0,1\}^r$ is a binary scaling parameter vector that specifies each of the features being selected or not. However, solving for $\boldsymbol{\vartheta}$ could be an NP-hard problem since the objective function usually is nonconvex and discontinuous in terms of $\boldsymbol{\vartheta}$.

To address the aforementioned issue, Li and Yang et al. [80] proposed a feature-wise kernel Lasso method, called feature vector machine, by introducing the following optimization:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^{r\times 1}} \ \frac{1}{2}\|\phi(\mathbf{y}) - \sum_{i=1}^{r}\alpha_i\phi(\mathbf{x}_{i\star})\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1,$$

where $\phi : \mathbb{R}^{r\times 1} \to \mathbb{R}^{d\times 1}$ is a given nonlinear mapping, and $\mathbf{x}_{i\star} \in \mathbb{R}^{n\times 1}$ is the i^{th} feature (not instance) of $\mathbf{X}$. To predict the response of a new instance, it requires minimizing another optimization problem (see (6) in [80]) whose solution may be difficult to obtain when complex kernel functions are adopted. Another feature-wise selection method, called HSIC Lasso, was proposed by Yamada and Jitkrittum et al. [77] as

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^{r\times 1}} \ \frac{1}{2}\|\bar{\mathbf{L}} - \sum_{i=1}^{r}\alpha_i\bar{\mathbf{K}}^i\|_F^2 + \lambda\|\boldsymbol{\alpha}\|_1,$$

where $\bar{\mathbf{L}} \in \mathbb{R}^{n\times n}$ is the centered kernel matrix of responses $\mathbf{y}$, and $\bar{\mathbf{K}}^i \in \mathbb{R}^{n\times n}$ is the centered kernel matrix of the i^{th} feature (row) of $\mathbf{X}$. However, the estimation of a new instance response is lacking, which prevents the connection from learning. Also, a sparse additive model for the feature selection in [76, 81] is proposed in the following form:

$$\min_{\boldsymbol{\alpha}_i\in\mathbb{R}^{d_n\times 1}} \ \frac{1}{2}\|\mathbf{y} - \sum_{i=1}^{r}\boldsymbol{\Psi}_i\boldsymbol{\alpha}_i\|_2^2 + \lambda\sum_{i=1}^{r}\|\boldsymbol{\Psi}_i\boldsymbol{\alpha}_i\|_2,$$

where $\boldsymbol{\Psi}_i \in \mathbb{R}^{n\times d_n}$ is the matrix generated by a set of basis functions $\{\psi_{i1}, \psi_{i2}, \cdots, \psi_{id_n}\}$ on the i^{th} feature (row) of $\mathbf{X}$ and $\boldsymbol{\Psi}_i(j,l) = \psi_{i,l}(X_{ij})$. The sparse additive model requires the nonlinearity in the model to be additive and thus it belongs to a quite special case.

Other nonlinear Lasso models such as LCPHR [82], SLR-$L_{1/2}$ [83], and LPR [84] apply only to very specific situations. For example, LCPHR is only applicable for survival data, LPR assumes the underlying data follows the Poisson distribution, and SLR-$L_{1/2}$ is used for gene (feature) selection in cancer classification. These models commonly adopt logarithms or exponents to achieve nonlinear learning that only fits a certain type of growth problem. Most critically, all aforementioned approaches are constructed under the framework of *univariate* regression, which is quite restrictive and different from the multivariate cases in which variable interactions play important roles and must be taken into account.

Now it comes to our motivation for solving the configuration recommendation problem. First, the relationship between features and instance responses tends to be nonlinear, and thus the kernelization Lasso approach is a good candidate to capture the non-linearity. Second, features of high dimensional datasets are multiple, which implies that a multivariate approach deems to be necessary. Third, it is essential to develop an efficient learning process that can integrate the dataset, the (configuration recommendation) model, and algorithm learning together to provide the most competent configuration recommendation automatically under the general framework of machine learning. To the best of our knowledge, such an approach is not available in the current literature.

3.3 MULTIVARIATE KERNEL GROUP LASSO

Suppose that we are given n paired samples

$$\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_n, \mathbf{y}_n)\} \tag{3.1}$$

for a specific multivariate regression problem where $\mathbf{x}_i \in \mathbb{R}^{r \times 1}$ is the predictor vector and $\mathbf{y}_i \in \mathbb{R}^{m \times 1}$ is the response. Denote $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n] \in \mathbb{R}^{m \times n}$ and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{r \times n}$, and $\mathbf{X}$ is further row-wisely partitioned into p $(1 \leq p \leq r)$ disjoint groups, i.e., $\mathbf{X} = [\mathbf{X}_1; \mathbf{X}_2; \cdots; \mathbf{X}_p]$ where $\mathbf{X}_i \in \mathbb{R}^{r_i \times n}, i = 1, 2, \cdots, p$, and $\sum_{i=1}^{p} r_i = r$. Aiming at meta-learning via a multivariate regression approach, we propose the following multivariate kernel group Lasso (KGLasso) model:

$$\min_{\boldsymbol{\Theta}} \ F(\boldsymbol{\Theta}) = \frac{1}{2}\|\mathbf{Y} - \sum_{i=1}^{p} \mathbf{W}_i \mathbf{K}_i\|_F^2 + \lambda \sum_{i=1}^{p} \|\mathbf{W}_i\|_{2,1}, \tag{3.2}$$

where $\boldsymbol{\Theta} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p] \in \mathbb{R}^{m \times pn}$ is the model parameter with $\mathbf{W}_i \in \mathbb{R}^{m \times n}, i = 1, 2, \cdots, p$, and $\mathbf{K}_i = \phi^\top(\mathbf{X}_i)\phi(\mathbf{X}_i) \in \mathbb{R}^{n \times n}$ is the kernel matrix.

Here p is a parameter with a positive integer range. It often takes the number of the existing grouping of meta-features, but it also can be used for other random groupings when more subtle groups become necessary under our framework. The first term in (3.2) is the generalized kernel Lasso regression, while the second term is the regularization, and the adopted norm $\|\cdot\|_{2,1}$ is used to enforce the sparsity solution of $\mathbf{W}_i$. The sparse solution of (3.2) allows us to identify the most important features with optimization. The model (3.2) can be viewed as the generalized instance-wise Lasso when $p = 1$ and the feature-wise kernelized Lasso when $p = r$. Solving (3.2) is not a trivial task since the objective function is not convex and the procedure requires some special treatment. We will present the details next.

3.3.1 Alternating iteration

To solve the minimization of (3.2), we adopt the block-coordinate descent method. We fix $\mathbf{W}_i$ for all i but one and then solve for $\mathbf{W}_l, l \neq i$ alternatively during each iteration. Therefore, without loss of generality, we can rewrite (3.2) as

$$F(\mathbf{W}_l) = \frac{1}{2}\|\mathbf{R}_{-l} - \mathbf{W}_l \mathbf{K}_l\|_F^2 + \lambda \|\mathbf{W}_l\|_{2,1} + C_{-l}, \tag{3.3}$$

where $\mathbf{R}_{-l} = \mathbf{Y} - \sum_{i \neq l}^{p} \mathbf{W}_i \mathbf{K}_i$ and $C_{-l} = \sum_{i \neq l}^{p} \|\mathbf{W}_i\|_{2,1}$ are constants. Before we move on, we use short notations in (3.3) for the purpose of simplicity and rewrite it as

$$F(\mathbf{B}) = \frac{1}{2}\|\mathbf{H} - \mathbf{B}\mathbf{K}\|_F^2 + \lambda\|\mathbf{B}\|_{2,1} + C, \tag{3.4}$$

where $\mathbf{B} = \mathbf{W}_l$, $\mathbf{H} = \mathbf{R}_{-l}$, $\mathbf{K} = \mathbf{K}_l$, and $C = C_{-l}$.

The gradient of (3.4) with respect to $\mathbf{B}$ is given by

$$\nabla F(\mathbf{B}) = -\mathbf{H}\mathbf{K}^\top + \mathbf{B}\mathbf{K}\mathbf{K}^\top + \lambda\mathbf{U}, \tag{3.5}$$

where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n]$ and

$$\mathbf{u}_j = \begin{cases} \frac{\mathbf{b}_j}{\|\mathbf{b}_j\|_2}, & \text{if } \|\mathbf{b}_j\|_2 \neq 0, \\ \mathbf{z}_j, & \text{if } \|\mathbf{b}_j\|_2 = 0, \end{cases} \quad \text{and } j = 1, 2, \cdots, n,$$

where $\|\mathbf{z}_j\|_2 \leq 1$. Therefore, if $\|\mathbf{b}_j\|_2 \neq 0$, then $\mathbf{b}_j$ can be updated via

$$\mathbf{b}_j = \left[\mathbf{H}\mathbf{k}_{j\star}^\top - \sum_{i \neq j}^{n} \mathbf{b}_i \|\mathbf{k}_{i\star}\|_2^2\right] \left(\|\mathbf{k}_{j\star}\|_2^2 + \frac{\lambda}{\|\mathbf{b}_j\|_2}\right)^{-1}; \tag{3.6}$$

otherwise, if $\|\mathbf{b}_j\|_2 = 0$, it satisfies

$$\left\|\mathbf{H}\mathbf{k}_{j\star}^\top - \sum_{i \neq j}^{n} \mathbf{b}_i \|\mathbf{k}_{i\star}\|_2^2\right\|_2 \leq \lambda. \tag{3.7}$$

Similarly, the explicit solution (3.6) may not produce desirable outcomes, as indicated in [65], due to the numeric instability caused in the scenario $\|\mathbf{k}_{j\star}\|_2 \to 0$. The instability analysis can also refer to in Appendix IV. We will address this issue next.

3.3.2 Auxiliary function

Instead of solving $\mathbf{B}$ directly from (3.4), we resort to solving its auxiliary function constructed below. Without loss of generality, we consider the j^{th} column $\mathbf{b}_j$ of $\mathbf{B}$ only and ignore the constant term in our following discussion, leading to the loss function

$$\hat{F}(\mathbf{b}_j) := \frac{1}{2}\|\mathbf{H} - \mathbf{B}\mathbf{K}\|_F^2 + \lambda\|\mathbf{b}_j\|_2. \tag{3.8}$$

We further denote $L(\mathbf{b}_j) := \frac{1}{2}\|\mathbf{H} - \mathbf{B}\mathbf{K}\|_F^2$. Suppose that $\mathbf{K}$ has nonzero rows, i.e., $\frac{1}{\|\mathbf{k}_{j\star}\|_2^2}, j = 1, 2, \cdots, n$, is well-defined, then we have the following theorem.

Theorem 2. Given a constant vector $\mathbf{b}_j^t$, we define

$$G(\mathbf{b}_j, \mathbf{b}_j^t) := L(\mathbf{b}_j^t) + (\mathbf{b}_j - \mathbf{b}_j^t)^\top \nabla L(\mathbf{b}_j^t) + \frac{1}{2a_j}\|\mathbf{b}_j - \mathbf{b}_j^t\|_2^2 + \lambda\|\mathbf{b}_j\|_2, \tag{3.9}$$

where

$$0 < a_j \leq \frac{1}{\|\mathbf{k}_{j\star}\|_2^2}. \tag{3.10}$$

Then we have

1. $\nabla L(\mathbf{b}_j^t) = (-\mathbf{H}\mathbf{K}^\top + \mathbf{B}^t\mathbf{K}\mathbf{K}^\top)_j$;

2. $G(\mathbf{b}_j, \mathbf{b}_j^t) \geq \hat{F}(\mathbf{b}_j), \quad G(\mathbf{b}_j^t, \mathbf{b}_j^t) = \hat{F}(\mathbf{b}_j^t)$.

The function $G(\cdot, \cdot)$ is called the auxiliary function in literature (see, e.g. [85]). The proof of Theorem 2 is given in Appendix V. Now we seek $\mathbf{b}_j^\star$ that minimizes $G(\mathbf{b}_j, \mathbf{b}_j^t)$, namely, $\mathbf{b}_j^\star = \arg\min_{\mathbf{b}_j} G(\mathbf{b}_j, \mathbf{b}_j^t)$. The gradient of (3.9) with respect to $\mathbf{b}_j$ is given by

$$\nabla G(\mathbf{b}_j, \mathbf{b}_j^t) = \nabla L(\mathbf{b}_j^t) + \frac{1}{a_j}(\mathbf{b}_j - \mathbf{b}_j^t) + \lambda \mathbf{u}_j. \tag{3.11}$$

By letting $\nabla G(\mathbf{b}_j, \mathbf{b}_j^t) = \mathbf{0}$ and denote $\tilde{\mathbf{b}}_j^t = \mathbf{b}_j^t - a_j \nabla L(\mathbf{b}_j^t)$, one can have

$$\mathbf{b}_j = \tilde{\mathbf{b}}_j^t \left(1 + \frac{a_j \lambda}{\|\mathbf{b}_j\|_2}\right)^{-1} \tag{3.12}$$

if $\|\mathbf{b}_j\|_2 \neq 0$, or

$$\|\tilde{\mathbf{b}}_j^t\|_2 \leq a_j \lambda \tag{3.13}$$

if $\|\mathbf{b}_j\|_2 = 0$. It can be shown that (3.12) and (3.13) are equivalent to (3.6) and (3.7), respectively, when $a_j = \frac{1}{\|\mathbf{k}_{j\star}\|_2^2}$.

Now we take the L_2-norm for the both sides of (3.12) with some derivations, one can see

$$\|\mathbf{b}_j\|_2 = \left(\|\tilde{\mathbf{b}}_j^t\|_2 - a_j \lambda\right)_+. \tag{3.14}$$

Plug (3.14) into (3.12) and simplify the fraction, we arrive at

$$\mathbf{b}_j = \tilde{\mathbf{b}}_j^t \left(1 - \frac{a_j \lambda}{\|\tilde{\mathbf{b}}_j^t\|_2}\right)_+. \tag{3.15}$$

As we have shown in Chapter 2, (3.15) is the unique solution of (3.12). Combining the n columns of $\mathbf{B}$, then the updating criterion for $\mathbf{B}$ in (3.4) can be described by

$$\mathbf{B} = \tilde{\mathbf{B}}^t (\mathbf{I} - \lambda \boldsymbol{\Sigma} \mathbf{D})_+, \tag{3.16}$$

where $\tilde{\mathbf{B}}^t = [\tilde{\mathbf{b}}_1^t, \tilde{\mathbf{b}}_2^t, \cdots, \tilde{\mathbf{b}}_n^t] = \mathbf{B}^t - (-\mathbf{H}\mathbf{K}^\top + \mathbf{B}^t\mathbf{K}\mathbf{K}^\top)\mathbf{D}$, $\mathbf{D} = \text{diag}([a_1, a_2, \cdots, a_n])$, and $\mathbf{\Sigma} = \text{diag}([\frac{1}{\|\tilde{\mathbf{b}}_1^t\|_2}, \frac{1}{\|\tilde{\mathbf{b}}_2^t\|_2}, \cdots, \frac{1}{\|\tilde{\mathbf{b}}_n^t\|_2}])$. Here $\text{diag}(\mathbf{x})$ represents a diagonal matrix with diagonal elements $\mathbf{x}$.

Now we recover (3.16) without using the short notation in (3.3), yielding

$$\mathbf{W}_l = \tilde{\mathbf{W}}_l^t(\mathbf{I} - \lambda\mathbf{\Sigma}_l\mathbf{D}_l)_+, \tag{3.17}$$

where $\tilde{\mathbf{W}}_l^t = \mathbf{W}_l^t - (-\mathbf{R}_{-l}\mathbf{K}_l^\top + \mathbf{W}_l^t\mathbf{K}_l\mathbf{K}_l^\top)\mathbf{D}_l$, $\mathbf{D}_l = \text{diag}([a_1^l, a_2^l, \cdots, a_n^l])$ with $a_j^l \in (0, \frac{1}{\|(\mathbf{k}_l)_{j\star}\|_2^2}]$, and $\mathbf{\Sigma}_l = \text{diag}([\frac{1}{\|(\tilde{\mathbf{w}}_l^t)_1\|_2}, \frac{1}{\|(\tilde{\mathbf{w}}_l^t)_2\|_2}, \cdots, \frac{1}{\|(\tilde{\mathbf{w}}_l^t)_n\|_2}])$.

The following theorem shows the convergence of our algorithm. The proof can be found in Appendix V as well. Therefore, by alternatively updating coefficient matrices $\mathbf{W}_l, l = 1, 2, \cdots, p$, a local minima of (3.2) can be obtained.

Theorem 3. Given $\mathbf{W}_i, i \neq l$, are fixed as constant matrices, under the updating rule given in (3.17), the loss function (3.2) is non-increasing during each alternatively updating.

3.4 META-LEARNING VIA KGLASSO

In this section, we show how to apply our proposed KGLasso approach for the automated meta-feature selection as well as configuration recommendation under the framework of meta-learning. The general architecture embedded with KGLasso is depicted in Figure 3.1, which consists of offline and online stages respectively. We next explain the details.

3.4.1 Offline phase

The main tasks in the offline stage include metadata extraction and the proposed KGLasso learning.

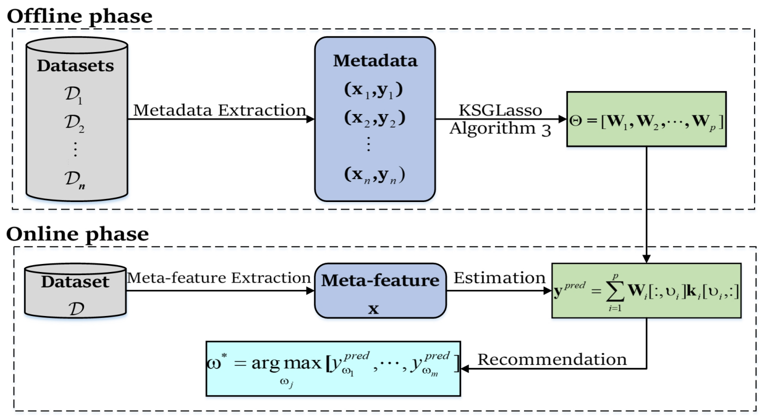

Offline phase
Datasets
$\mathcal{D}_1$
$\mathcal{D}_2$
$\vdots$
$\mathcal{D}_n$
Metadata Extraction
Metadata
$(\mathbf{x}_1,\mathbf{y}_1)$
$(\mathbf{x}_2,\mathbf{y}_2)$
$\vdots$
$(\mathbf{x}_n,\mathbf{y}_n)$
KSGLasso
Algorithm 3
$\Theta = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p]$
Online phase
Dataset
$\mathcal{D}$
Meta-feature Extraction
Meta-feature
$\mathbf{x}$
Estimation
$\mathbf{y}^{pred} = \sum_{l=1}^{p} \mathbf{W}_l[:, \upsilon_l]\mathbf{k}_l[\upsilon_l, :]$
Recommendation
$\omega^* = \arg\max_{\omega_j} [y^{pred}_{\omega_1}, \cdots, y^{pred}_{\omega_m}]$

Figure 3.1: The automated configuration recommendation architecture based on the proposed KGLasso model associated with meta-learning functionality.

Metadata extraction

Let $\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_n$ be the n collected historical datasets from which n pairs of metadata, $(\mathbf{x}_j, \mathbf{y}_j), j = 1, 2, \cdots, n$, are extracted, in which $\mathbf{x}_j \in \mathbb{R}^{r\times 1}$ is the meta-features of $\mathscr{D}_j$, and $\mathbf{y}_j \in \mathbb{R}^{m\times 1}$ is the classification performance evaluated on $\mathscr{D}_j$ over the search space $\mathbf{\Omega}_{\mathscr{A}}$. The Python library `pymfe` [23] is designed for extracting the state-of-the-art meta-features and therefore it is adopted in our system, while 10-fold stratified cross-validation is leveraged to evaluate the performance $\mathcal{P}$ of a configuration $\boldsymbol{\omega}_s \in \mathbf{\Omega}_{\mathscr{A}}$:

$$y_{\boldsymbol{\omega}_s} = \frac{1}{10}\sum_{i=1}^{10} \mathcal{P}(\mathscr{A}_{\boldsymbol{\omega}_s}, \cup_{i\neq q}^{10} \mathscr{D}_j^{(i)}, \mathscr{D}_j^{(q)}), s = 1, 2, \cdots, m,$$

where $\mathcal{P}(\cdot,\cdot,\cdot)$ represents the performance achieved when $\mathscr{A}_{\boldsymbol{\omega}_s}$ is trained on $\cup_{i\neq q}^{10} \mathscr{D}_j^{(i)}$ and evaluated on $\mathscr{D}_j^{(q)}$. $\mathscr{D}_j^{(i)}, i = 1, 2, \cdots, 10$, is a 10-fold partition of dataset $\mathscr{D}_j$.

In general, the intrinsic relationship between meta-feature and classification performance often appears to be nonlinear, and capturing the underlying nonlinearity usually benefits the accuracy of recommendation. In addition, selecting the most explainable meta-features is also vital for the improvement of both efficiency and effectiveness. Therefore, we develop a new meta-leaner, KGLasso, to fulfill both automated feature selection and configuration recommendation, which is formulated as

$$F(\mathbf{\Theta}) = \frac{1}{2}\|\mathbf{Y} - \sum_{i=1}^{p} \mathbf{W}_i \mathbf{K}_i\|_F^2 + \lambda \sum_{i=1}^{p} \|\mathbf{W}_i\|_{2,1}, \tag{3.18}$$

where $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n] \in \mathbb{R}^{m \times n}$ is the performance matrix, $\mathbf{K}_i = \phi^\top(\mathbf{X}_i)\phi(\mathbf{X}_i)$, and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] = [\mathbf{X}_1; \mathbf{X}_2; \cdots; \mathbf{X}_p] \in \mathbb{R}^{r \times n}$ is the grouped meta-features, and $\mathbf{\Theta} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p]$ is the model parameter.

KGLasso learning

As we stated before, we adopt the block-coordinate descent algorithm to update the coefficient matrices $\mathbf{W}_i, i = 1, 2, \cdots, p$. The updating procedures are summarized in `Algorithm` 3. To illustrate the alternative updating process, we use the notations defined in (3.2). In step 1, we can group the meta-features based on their natural grouping properties, such as each type of meta-features being a group or other strategies as needed in the applications. The RBF (radial basis) kernel function,

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2}{\sigma}\}, \sigma > 0,$$

is leveraged in our model because the selection of the kernel parameter σ is flexible and the kernel is within 0 and 1. In practice, the unified setting of σ in various groups may cause biases: when groups have different numbers of meta-features, i.e., $r_i \neq r_j$, $k(\mathbf{x}_1, \mathbf{x}_2)$ with a higher dimension of $\mathbf{x}$ tends to be smaller given the same σ across the p groups.

Algorithm 3 `KGLasso`

```
Input: Performance matrix $\mathbf{Y}$, meta-features $\mathbf{X}$, kernel function $k(\cdot,\cdot)$, tuning parameter $\lambda$, number of groups $p$, maximal allowable number of iterations maxL
Output: Model parameter $\mathbf{\Theta} = [\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_p]$
 1: Categorize $\mathbf{X}$ into $p$ disjoint groups $[\mathbf{X}_1; \mathbf{X}_2; \cdots; \mathbf{X}_p]$
 2: Compute kernel matrices $\mathbf{K}_i$ of $\mathbf{X}_i$ using $k(\cdot,\cdot)$
 3: Initialize $\mathbf{\Theta}^0 = [\mathbf{W}_1^0, \mathbf{W}_2^0, \cdots, \mathbf{W}_p^0]$, where $\mathbf{W}_i^0 \neq \mathbf{0}, i = 1, 2, \cdots, p$
 4: Calculate $\mathbf{D}_i = \text{diag}(a_1^i, a_2^i, \cdots, a_n^i)$, $i = 1, 2, \cdots, p$.
 5: for $t = 1$ : maxL do
 6:     for $i = 1 : p$ do
 7:         compute gradient $\nabla L(\mathbf{R}_{-i}, \mathbf{W}_i^{t-1})$
 8:         update $\tilde{\mathbf{W}}_i^{t-1} \leftarrow \mathbf{W}_i^{t-1} - \nabla L(\mathbf{R}_{-i}, \mathbf{W}_i^{t-1})\mathbf{D}_i$
 9:         update $\mathbf{W}_i^t \leftarrow \tilde{\mathbf{W}}_i^{t-1}(\mathbf{I} - \lambda \mathbf{\Sigma}_i \mathbf{D}_i)_+$
10:     end for
11: end for
```

In experiments, we set σ first, then σ_i in group i is determined by $\sigma_i = \sigma r_i, i = 1, 2, \cdots, p$, which can effectively reduce the potential biases. Also, we set $a_j^i = \frac{1}{10\|(\mathbf{k}_i)_{j\star}\|_2^2}, j = 1, 2, \cdots, n$, in our algorithm which is obtained by grid search within the given range of a_j^i in Theorem 2. In step 7, $\nabla L(\mathbf{R}_{-i}, \mathbf{W}_i^{t-1}) = -\mathbf{R}_{-i}\mathbf{K}_i^\top + \mathbf{W}_i^{t-1}\mathbf{K}_i\mathbf{K}_i^\top$ is used.

Meta-feature and instance selection

Based on the updating rule in step 9, each $\mathbf{W}_j$ can be column-wise sparse. If $\mathbf{W}_j = \mathbf{0}$, a zero matrix, then the corresponding meta-features are supposed to be least important and thus are excluded from the investigated recommendation problem; if some columns of $\mathbf{W}_j$ are zero, then the corresponding training instances are assumed to be less important and thus are deleted from the model. With appropriate (λ, σ) setting, our KGLasso model not only can select the principal meta-features and instances for a specific meta-learning task but also obtain a highly sparse solution that improves the training efficiency with simplifying complexity.

Let $\boldsymbol{v}_i, i = 1, 2, \cdots, p$, be the index vector that specifies the nonzero columns of $\mathbf{W}_i$ (if

$\mathbf{W}_i = \mathbf{0}$, then $\boldsymbol{\upsilon}_i$ is an empty set), then our trained KGLasso model can be written as

$$\mathbf{Y} = \sum_{i=1}^{p} \mathbf{W}_i[:, \boldsymbol{\upsilon}_i]\mathbf{K}_i[\boldsymbol{\upsilon}_i, :], \tag{3.19}$$

where $\mathbf{W}_i[:, \boldsymbol{\upsilon}_i]$ ($\mathbf{K}_i[\boldsymbol{\upsilon}_i, :]$) represents the submatrix acquired by taking columns (rows) indexed by $\boldsymbol{\upsilon}_i$.

3.4.2 Online phase

During the online phase, we need to estimate the empirical performance of configurations using the trained KGLasso meta-learner and make a configuration recommendation. For a new dataset $\mathscr{D}$, the performance of $\mathscr{A}$ over the search space $\boldsymbol{\Omega}_{\mathscr{A}}$ can be estimated only by the selected meta-features and instances. Denote $\mathbf{x}_{\mathscr{D}}$ as the meta-feature of $\mathscr{D}$ and it is partitioned into p groups $\mathbf{x}_{\mathscr{D}} = [\mathbf{x}_1^{\mathscr{D}}; \mathbf{x}_2^{\mathscr{D}}; \cdots; \mathbf{x}_p^{\mathscr{D}}]$, applying the same grouping principle used on the training stage, the performance of $\boldsymbol{\Omega}_{\mathscr{A}}$ on dataset $\mathscr{D}$ can be estimated by

$$\mathbf{y}^{pred} = \sum_{i=1}^{p} \mathbf{W}_i[:, \boldsymbol{\upsilon}_i]\mathbf{k}_i[\boldsymbol{\upsilon}_i, :],$$

where $\mathbf{k}_i = \phi^{\top}(\mathbf{X}_i)\phi(\mathbf{x}_i^{\mathscr{D}}) \in \mathbb{R}^{n\times 1}$, $\mathbf{X}_i$ is the meta-features of the i^{th} group of training datasets, defined in Section III. In practice, we just need to calculate the entries of $\mathbf{k}_j$ indexed by $\boldsymbol{\upsilon}_j$ and thus we are only required to extract those meta-feature groups where $\boldsymbol{\upsilon}_j$ is not empty. The configuration that has the highest predictive performance is recommended to the new dataset $\mathscr{D}$, i.e., $\boldsymbol{\omega}^{recom} = \arg\max_{\boldsymbol{\omega}}[y^{pred}_{\boldsymbol{\omega}_1}, y^{pred}_{\boldsymbol{\omega}_2}, \cdots, y^{pred}_{\boldsymbol{\omega}_m}]$, here $y^{pred}_{\boldsymbol{\omega}_i} \in \mathbf{y}^{pred}$ is the predictive performance of configuration $\boldsymbol{\omega}_i$ from the search space $\boldsymbol{\Omega}_{\mathscr{A}}$. Since the optimal configuration on a dataset usually is not unique, we thus recommend multiple configurations, i.e.,

$$\boldsymbol{\omega}^{recom} = \{\boldsymbol{\omega}_{j_1}, \boldsymbol{\omega}_{j_2}, \cdots, \boldsymbol{\omega}_{j_c}\} \tag{3.20}$$

where $1 \leq c \ll m$ and

$$y^{pred}_{\boldsymbol{\omega}_{j_1}} \geq y^{pred}_{\boldsymbol{\omega}_{j_2}} \geq \cdots \geq y^{pred}_{\boldsymbol{\omega}_{j_m}}.$$

We call $\boldsymbol{\omega}_{j_l}$ as the l^{th}-order recommendation.

3.5 EXPERIMENTS

In this section, we report the empirical results of our proposed KGLasso method in terms of the effectiveness of hyperparameter recommendation. We here first introduce the configurations of the meta-learning recommendation system, including the employed classification datasets, meta-features, classification algorithm, and its hyperparameter search space.

3.5.1 Experimental configurations

Classification datasets

120 classification datasets are collected from the well-known UCI machine learning repository in our experiments, and 60 of them are randomly picked as historical datasets for model training, and the other 60 datasets are leveraged as testing datasets. The datasets have a wide range of distributions about the number of instances (attributes and classes) and detailed information is given in Appendix I. New classification datasets can be further added to our system without restrictions.

Table 3.1: The adopted meta-features and the numbers of measures.

Names (Abbr.)	#meta-features
Statistics&Information theory (SIT)	54
Model structure (MS)	24
Landmarking (LM)	14
Relative landmarking (RLM)	14
Problem complexity (PC)	23
Structural information (SI)	11
Clustering (CLU)	8
Concept (CON)	7
United (UNI)	155

Meta-features

There are eight types of meta-features [23, 57] that are widely utilized by researchers, each of them characterizes different aspects of a classification dataset, namely statistics and information theory, classification complexity, structural information, (relative) landmarking, concept, clustering, and model structure. We extract these meta-features using the `pymfe` library, and each type of meta-feature is categorized into one group (thus $p = 8$). In the implementation, each meta-feature is normalized into $[0, 1]$ via min-max scaling to eliminate the domain differences that existed among the meta-features. In addition, some repetitive meta-features across diverse groups are removed at the beginning. The number of the total adopted meta-features in each type is summarized in Table 3.1.

Classification algorithm and hyperparameter space

We consider the hyperparameter recommendation problem for the well-known support vector machine (SVM) [86] to promote its efficiency in real applications. Specifically, we adopt SVM with the RBF kernel because of the flexibility of kernel parameter selection. Therefore, there are two tuning parameters of RBF-SVM, namely, regularization coefficient '`C`' and kernel width 'σ_{svm}'. As the settings leveraged in many related work [71, 14], we set `C`

$\in \{2^{-5}, 2^{-4}, \cdots, 2^{15}\}$ and $\sigma_{svm} \in \{2^{-15}, 2^{-14}, \cdots, 2^{3}\}$. Thus, the search space of RBF-SVM has $21 \times 19 = 399$ hyperparameters, i.e., $|\mathbf{\Omega}_{\mathscr{A}}| = 399$. We implement the function `SVC` developed in `scikit-learn` machine learning library [72] in our experiments.

Evaluation metrics

We evaluate the performance of each approach using four metrics: classification accuracy rate (CA), recommendation accuracy rate (RA), hit rate (HR), and normalized discounted cumulative gain (NDCG), respectively. The first three are widely considered in meta-learning [15, 16, 87], while the last one is commonly used in recommender system studies [88]. Definitions of the metrics can be found in Appendix III.

3.5.2 Experimental results and analyses

The experimental reports are given in four parts. The first and the second parts illustrate the capacity of meta-feature and instance selection of KGLasso and its parameter sensitivity, and the third and the fourth reports compare the recommendation performance of KGLasso to the state-of-the-art meta-learners and search algorithms. For each testing dataset, the top-3 configurations are recommended, i.e., $c = 3$ in (3.20), and their average performance is presented.

Capacity of meta-feature and instance selection

There are two tuning parameters for KGLasso, namely, the RBF-kernel width σ and the regularization parameter λ, and their search ranges are given in Table 3.2. The optimal tuning parameter pair $(\lambda^{\star}, \sigma^{\star}) = (6, 0.009)$ is determined by grid search. The evaluation metric of the grid search is the mean of CA, RA, HR, i.e., (CA+RA+HR)/3. To show the influence of tuning parameters on the meta-feature and instance selection, we illustrate the

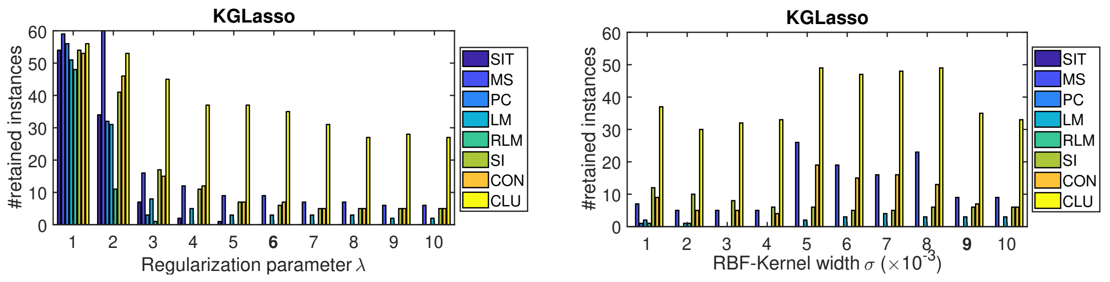


Figure 3.2: The number of retained meta-features and instances in our hyperparameter recommendation experiments. The y-axis shows the number of retained instances and the x-axis shows the various applied model parameters, i.e., λ (left) and σ (right). A zero means the corresponding meta-feature is not selected by the model.

Table 3.2: Settings for the tuning parameters of KGLasso, KNN, KKNN, MLP, and WarmCF.

Approach	Parameters	Usage	Range
KGLasso	λ	$L_{2,1}$ regularization parameter	`{1,2,⋯,10}`
	σ	parameter of RBF kernel	`{1e-3,2e-3,⋯,1e-2}`
KNN	`K`	number of neighbors	`{1,2,⋯,50}`
KKNN	σ	parameter of RBF kernel	`{1e-3,1e-2,1e-1,1e0,1,2,3,4,5,1e1,2e1,5e1,1e2}`
MLP	λ	L_2 regularization parameter	`{1e-7,1e-6,1e-5,1e-4,1e-3,1e-2}`
	s	Activation function	`Sigmoid`
	`neural_struc`	Neural network structure	`r-100-300-m`
WarmCF	λ	L_2 regularization parameter	`{1e-7,1e-6,1e-5,1e-4,1e-3,1e-2}`
	ϵ	keeping ratio of random corruption	`{0.1,0.2,⋯,1}`
	r	number of the latent variables	`{10,20,30,40,50}`

number of the retained instances of each group as the changing of tuning parameters in Figure 3.2 where σ is fixed as 0.009 on the left while λ is fixed as 6 on the right.

From Figure 3.2 (left), one can observe that the regularization parameter λ has a significant impact on the meta-feature and instance selection, as suggested by (3.15). When σ is fixed, the number of the retained instances in each group is decreasing as the increase of λ. Specifically, no meta-features are removed and most of the instances in each group are kept as well when $\lambda = 1$; nevertheless, PC and RLM meta-features are deleted from the model after $\lambda = 4$ and the model tends to be stable after $\lambda = 5$ where SIT, PC, and RLM meta-features are assumed to be redundant and are thus excluded from the model. Only a small amount of instances are selected in most of the remaining groups except CLU, which evidences the high degree of sparsity obtained by our KGLasso and also shows the dominant role of the CLU meta-feature.

The impact of σ on the selection task is more complicated. Given λ is properly fixed, as shown in Figure 3.2 (right), we do not observe a monotonous tendency when σ is varying, i.e., the numbers of the retained instances in eight groups are not decreasing or increasing, but we still can draw some similar conclusions. For example, an extremely sparse model is obtained and the principal meta-features are MS, LM, SI, CON, and CLU; especially, at least half of the instances are selected for CLU meta-feature under each σ. Besides, σ seems to determine more on the instance rather than the meta-feature selection when λ is fixed. In summary, by setting (λ, σ) appropriately, KGLasso is able to select the most essential meta-features and instances. We next elaborate on how promising configurations are recommended.

We illustrate the fluctuation of recommendation performance when tuning parameters

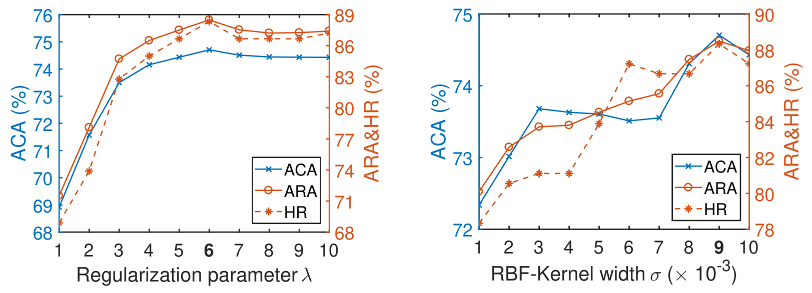


Figure 3.3: The influence of model parameters on the recommendation performance of KGLasso.

Table 3.3: Comparisons on ACA, ARA, and HR between CLU, CON, SI, LM, MS, SIT, PC, RLM, UNI, and KGLasso. The best value of each row is highlighted.

Metrics	CLU	CON	SI	LM	MS	SIT	PC	RLM	UNI	KGLasso
ACA	73.91	72.64	73.03	73.55	72.94	73.46	72.64	72.74	73.54	**74.70**
ARA	86.13	81.21	85.67	85.22	82.44	86.08	83.48	80.81	84.70	**88.49**
HR	86.67	78.89	86.67	83.33	82.22	85.00	82.78	76.67	81.11	**88.33**

(λ, σ) are varying in Figure 3.3 to demonstrate the impact of meta-features and instances selection on the recommendation performance. Similarly, σ is fixed to 0.009 on the left figure while λ is fixed to 6 on the right one. As we can see from Figure 3.3 (left), when all types of meta-features are kept ($\lambda = 1$), KGLasso has a mediocre performance with an ACA of 68.91%, an ARA of 71.51%, and an HR of 68.89%. As more instances are removed from the model ($\lambda = 2, 3$), the performance has a significant improvement (73.50%, 84.72%, and 82.78% on ACA, ARA, and HR). The optimal performance is obtained when $\lambda = 6$ where three types of meta-features are deleted and most of the instances within each remaining group are excluded. On the other hand, σ also has a considerable influence on the recommendation outcomes (Figure 3.3 right), which indicates the importance of instance selection in determining the recommendation performance, but the fluctuation range is less than that of λ.

Importance of meta-feature selection

In this experiment, our goal is to show the necessity of meta-feature selection. To fulfill this purpose, we compare our KGLasso with the generalized Lasso, i.e., the special case of KGLasso when $p = 1$, where every single type of meta-feature given in Table 3.1, including the united meta-feature (UNI), has been adopted. The comparisons in terms of ACA, ARA, and HR are summarized in Table 3.3, in which the first five columns are the results of the selected meta-features, and the middle three columns are the results of the removed meta-features. One can observe that 1) the best performance is obtained by KGLasso with an ACA of 74.70%, an ARA of 88.49%, and an HR of 88.33%, while the second performance is achieved by CLU, and the worst outcomes are generated by CON and RLM. UNI has medium performance but the extraction of all meta-features could be time-consuming. These results suggest the effectiveness of our KGLaso approach and the importance of meta-feature selection; 2) within the five selected meta-features, CLU has the best performance, which is coincident with the results shown in Figure 3.2, i.e., CLU has the largest amount of the retained instances so it is the most important one; 3) most importantly, the performance of SIT, PC, and RLM is close to that of CLU, MS, and CON, respectively, but SIT, PC, and RLM are removed by our model. Evidently, our KGLasso approach is able to capture the similarity among the groups of meta-features via learning and therefore exclude those redundant or nonsignificant ones. This is the central goal of our proposed method.

Moreover, to show if KGLasso is statistically better than other cases, we adopt Wilcoxon signed-rank tests [89] at a significance level of 0.05. The null hypothesis is that KGLasso does not outperform significantly. We test this on both CA and RA and the testing p-values are

Table 3.4: The Wilcoxon Signed-Rank test results between CLU, CON, SI, LM, MS, SIT, PC, RLM, UNI, and KGLasso. Here ">" represents "is better than". p-Values that are larger than 0.05 are underlined.

Hypothesis	win/tie/lose	p-value (CA)	p-value (RA)
KGLasso>CLU	32/5/23	<u>0.226</u>	<u>0.201</u>
KGLasso>CON	40/4/16	0.001	0.000
KGLasso>SI	35/4/21	0.048	<u>0.108</u>
KGLasso>LM	34/4/22	0.042	<u>0.084</u>
KGLasso>MS	35/6/19	0.003	0.003
KGLasso>SIT	36/4/20	<u>0.091</u>	<u>0.121</u>
KGLasso>RLM	38/4/18	0.004	0.003
KGLasso>PC	36/3/21	0.017	0.036
KGLasso>UNI	34/5/21	0.018	0.026

summarized in Table 3.4 where a p-value that is smaller than 0.05 means we reject the null hypothesis and accept the alternative hypothesis, i.e., the superiority of KGLasso is significant. In addition, we also present the number of datasets where KGLasso wins/ties/loses in each comparison in Table 3.4. We can see that, except for CLU and SIT, KGLasso outweighs other meta-features statistically on CA, and is significantly better than CON, MS, RLM, PC, and UNI on RA. Although the difference between CLU and KGLasso is not statistically significant, KGLasso still improves the overall performance with a large gap as given in Table 3.3. Our approach formulation by (3.2) can be seen as a combination of various meta-features (i.e., multiple-feature learning), so its recommendation performance can be effectively attained by leveraging all the underlying critical meta-features.

Comparisons with meta-learning baselines

Based on our formulation, we compare the proposed KGLasso approach to four regression-based meta-learning baselines for configuration recommendation, namely, KNN [16], kernel KNN (KKNN) [13], multi-layer Perceptron (MLP) [73], and warm-started collaborative filtering (WarmCF) [18]. The involved tuning parameters of each baseline as well

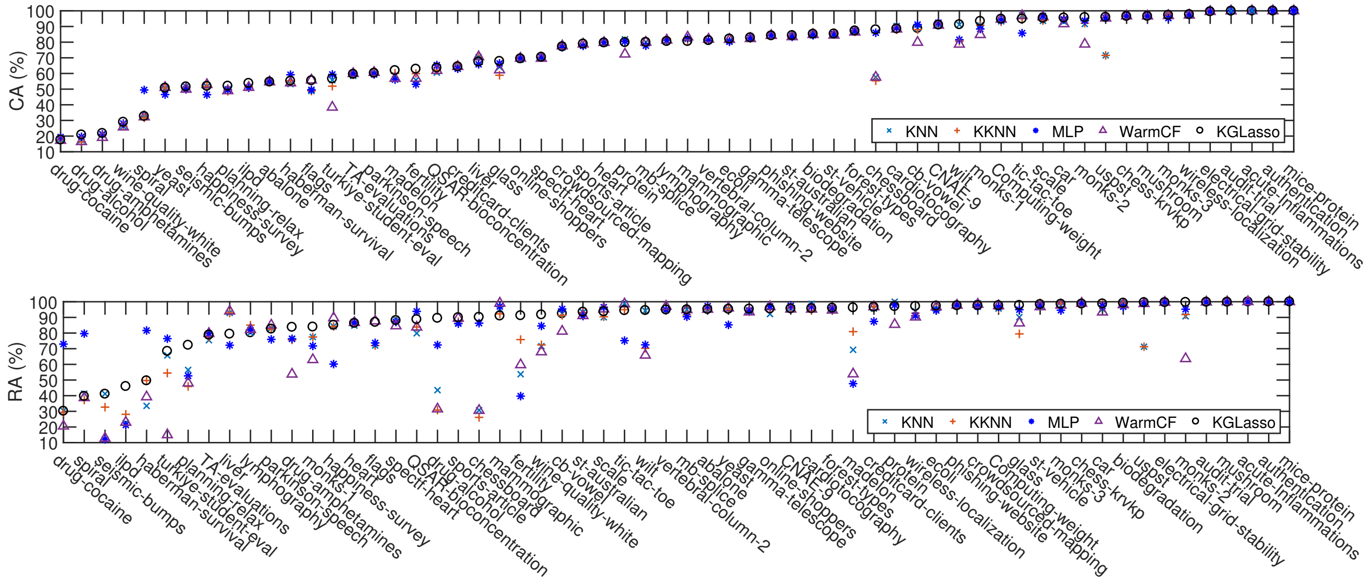


Figure 3.4: Comparisons on CA (top) and RA (bottom) between KNN, KKNN, MLP, WarmCF, and KGLasso on each single testing dataset.

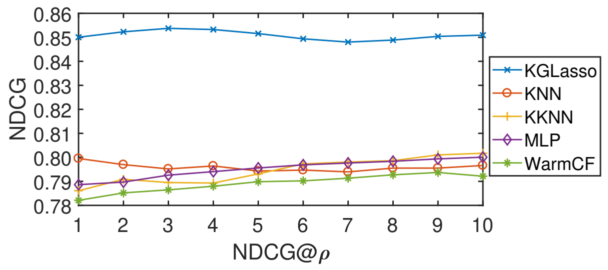


Figure 3.5: The comparisons on NDCG between KNN, KKNN, MLP, WarmCF, and KGLasso.

Table 3.5: Comparisons on ACA, ARA, and HR between KNN, KKNN, MLP, WarmCF, and KGLasso. The maximal value of each row is highlighted. The ACA of the real optimal configuration on the testing datasets is 77.58%.

Metrics	KGLasso	KNN	KKNN	MLP	WarmCF
ACA	**74.70**	73.01	72.74	73.87	72.38
ARA	**88.49**	82.97	82.56	84.76	80.09
HR	**88.33**	81.11	81.11	79.53	76.19

as their search ranges are given in Table 3.2 and the best parameters are determined by grid search. The meta-features employed by each baseline are the united one, i.e., UNI, since they do not have the meta-feature selection mechanism as KGLasso does. The overall comparative results are given in Table 3.5, and the comparison on each dataset is depicted in Figure 3.4.

From Table 3.5, one can observe that our KGLasso outweighs the comparative baselines with a wide gap, with an ACA of 74.70%, an ARA of 88.49%, and an HR of 88.33%. MLP has the secondary performance in terms of ACA and ARA (73.87% and 84.76%) but its HR (79.53%) is inferior to KNN and KKNN. This is because MLP has good recommendations on partial testing datasets but it performs mediocrely on other datasets. KNN and KKNN have similar tertiary performance. Although KKNN is a nonlinear version of KNN, it does not show any superiority in this experiment. WarmCF has the worst performance, with an ACA of 72.38%, an ARA of 80.09%, and an HR of 76.19%. For the per-dataset comparison (Figure 3.4), the differences on CA are subtle while they are more evident on RA. We can see that KGLasso has the best RA on most of the datasets, and the superiority on some datasets, such as `planning-relax`, `drug-alcohol`, and `madelon`, is significant. On the other hand, MLP evidently overpasses other approaches on datasets `drug-cocaine`, `spiral`, and `harberman-survival`, but it also has the worst recommendations on datasets

Table 3.6: The Wilcoxon Signed-Rank tests between KNN, KKNN, MLP, WarmCF, and KGLasso. Here ">" represents "is better than". p-Values that are larger than 0.05 are underlined.

Alternative hypothesis	win/tie/lose	p-value (CA)	p-value (RA)
KGLasso>KNN	40/5/15	0.000	0.000
KGLasso>KKNN	43/3/14	0.000	0.000
KGLasso>MLP	37/1/22	0.008	0.006
KGLasso>WarmCF	37/1/22	0.000	0.001

such as `happiness-survey`, `tic-tac-toe`, and `madelon`, which again shows its unstable performance.

In addition, we report the comparisons on NDCG to show the recommendation potentiality of each approach in Figure 3.5. We set 10 positions for calculating NDCG@ρ, i.e., $\rho = [1, 2, \cdots, 10]$. It is clear to see that KGLasso has an average NDCG of 0.85 at each position, which is dramatically better than that of the comparative baselines, in which the NDCG under each position is varying from 0.78 to 0.80. The higher NDCG value implies that the first ρ-order recommendations of KGLasso are closer to the real first ρ optimal configurations than the recommendations generated by other baselines, thus our proposed approach is quite competent across different orders of recommendations.

Finally, we report the statistical tests, as we leveraged in the previous discussion, in Table 3.6, and we can see that all p-values under both CA and RA are less than 0.05, so the superiority of our KGLasso against KNN, KKNN, MLP, and WarmCF is statistically significant.

Comparison with search algorithms

With the same allocated time budget, we compare the online recommendation performance of KGLasso to the performance of four popular search algorithms, namely, random search (RS) [7], hyperband (HB) [8], Bayesian optimization (BO) [6], and heteroscedastic

Table 3.7: Comparisons on ACA, ARA, and HR between DEF, BO, HEBO, HB, RS, and KGLasso. The maximal value of each row is highlighted.

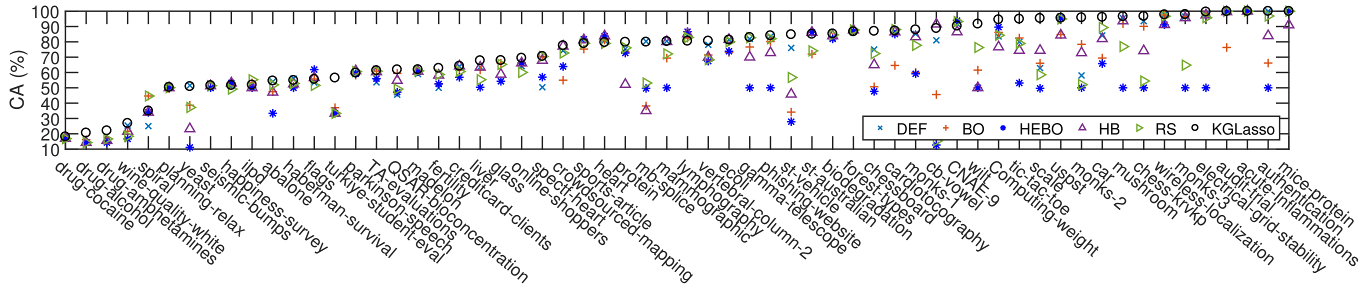
Metrics	DEF	BO	HEBO	HB	RS	KGLasso
ACA	69.00	64.66	57.57	66.37	65.72	**74.50**
ARA	70.67	63.31	42.41	65.13	64.65	**87.64**
HR	65.00	62.78	37.78	60.00	61.11	**86.67**

Table 3.8: The Wilcoxon Signed-Rank test results between DEF, BO, HEBO, HB, RS, and KGLasso. Here ">" represents "is better than". p-Values that are larger than 0.05 are underlined.

Hypothesis	win/tie/lose	p-value (CA)	p-value (RA)
KGLasso>DEF	45/3/12	0.000	0.000
KGLasso>BO	49/2/9	0.000	0.000
KGLasso>HEBO	49/2/9	0.000	0.000
KGLasso>HB	49/1/10	0.000	0.000
KGLasso>RS	49/2/9	0.000	0.000

evolutionary BO (HEBO) [34] which is the winner of NeurIPS 2020 competition. Notice that we only need to extract the selected meta-features from the new dataset during the online stage, i.e., CLU, CON, MS, LM, and SI. To avoid randomness, each search algorithm is repeated three times and their average is reported with 10-fold cross-validation being employed to evaluate the configurations. The performance of the default configurations (DEF) provided by `scikit-learn` is also compared. For simplicity, we only consider the first-order recommendation of our approach. The overall results are given in Table 3.7 and the per-dataset comparisons are shown in Figure 3.6.

As one can see that four search algorithms do not have satisfactory performance. The best performance results are from HB, where the ACA, ARA, and HR are 66.37%, 65.13%, and 60.00%, respectively. HEBO does not show its superiority in this comparison and it obtains the worst outcomes with an ACA of 57.57%, an ARA of 42.41%, and an HR of 37.38%. In contrast, our KGLasso greatly improves the recommendation performance with an ACA of 74.50%, an ARA of 87.64, and an HR of 86.67%, and the differences with four



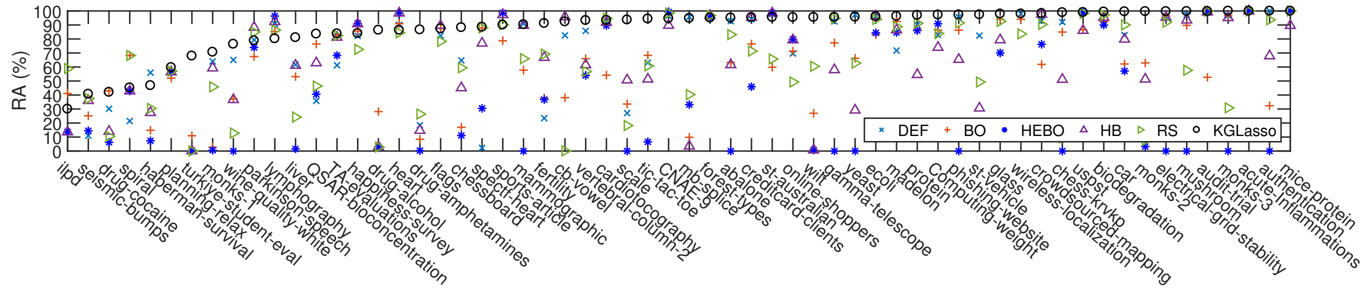


Figure 3.6: Comparisons on CA (top) and RA (bottom) between DEF, BO, HEBO, HB, RS, and KGLasso on each single testing dataset.

search algorithms are significant by the statistical tests given in Table 3.8, where KGLasso overpasses search algorithms on 49 out of 60 cases and the p-values on both CA and RA are pretty small. By taking a closer look at the per-dataset comparisons in Figure 3.6, a similar conclusion can be drawn. Only in a few cases where search methods outweigh KGLasso slightly, e.g., `lymphography`, `heart`, and `forest-types`. This is due to the small dimensions of these datasets (see Appendix I) and the cross-validations can be done efficiently from which the search algorithms benefit as expected. On the other hand, we notice that the default hyperparameter setting even outperforms search algorithms in this scenario with an ACA of 69.00%, an ARA of 70.67%, and an HR of 65.00%, but it is still inferior to our approach with a noticeable difference. By embedding the meta-feature selection via automated self-learning, our proposed KGLasso model demonstrates the superior ability in configuration recommendation, compared to various standard approaches.

3.6 CONCLUDING REMARKS

In this chapter, we develop a new hyperparameter recommendation approach embedded with meta-learning capacity in performing both meta-feature and instance selection automatically. The meta-learning is accomplished through our proposed KGLasso model which is constructed by a generalized multivariate kernel group Lasso approach. Our KGLasso setting is capable of capturing the underlying nonlinear characteristics between the features and the responses, including the univariate kernel Lasso as a special case under our proposed framework. The alternating minimization algorithm developed for the proposed KGLasso model is shown to be effective and robust via mathematical validation. The developed hyperparameter recommendation integrates the underlying datasets, the model, and the learning algorithm together to achieve the meta-learning task. The extensive experiments are based on the 120 UCI datasets and SVM for the configuration recommendation and demonstrate the superior performance of our proposed approach in terms of the common meta-learning baselines. Various comparisons with existing search algorithms such as random search, hyperband, and Bayesian optimization confirm the significant benefits of the proposed automated meta-learning approach.

CHAPTER 4

SPATIAL STRUCTURE PRESERVING

4.1 INTRODUCTION

In the current literature on meta-learning, the performance spaces of algorithms are usually instantiated as a vector. Although this framework is easy to implement with many existing models, the topographical structure of the performance space and the correlation existing in adjacent entries, which are critical characteristics for datasets, may be discarded. In this chapter, we attempt to structure the performance data into a multi-dimensional array whose dimension is governed by the number of different types of hyperparameters so that relevant information can be preserved. For example, when we consider the performance of RBF support-vector machine (SVM), which has two hyperparameters '`C`' and '`Gamma`', on the UCI classification dataset `firm-teacher-clave-direction` (see Figure 4.1), where the accuracy is determined by ten-fold cross-validation. The right figure shows the performance distribution when it is organized as a vector while the left figure shows that in a two-dimensional tensor (matrix). Clearly, the left figure presents the geometrical structure of the performance space and contains correlated information that reflects the data's spatial characteristic features.

In this chapter, we formulate the hyperparameter configuration recommendation task as a low-rank tensor completion (LRTC) problem. LRTC is a generalization of matrix completion [90] for high-dimensional arrays and it has been widely applied to both real applications and theoretical studies such as in image and video inpainting [91, 92, 93] and image denoising [94] where the low-rank property of matrices or tensors is the key of success.

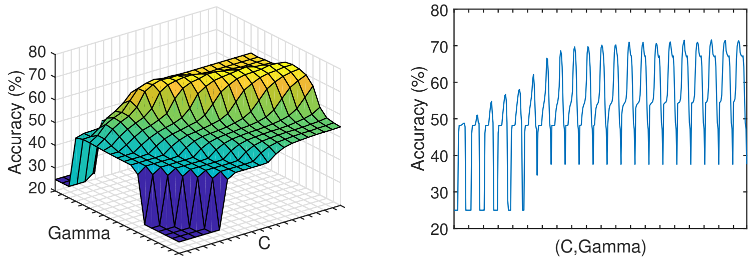


Figure 4.1: An example of performance data organization of RBF-SVM, which has two hyperparameters '`C`' and '`Gamma`', in tensor format (left) and in vector format (right) on the UCI problem `firm-teacher-clave-direction`.

The common approaches for LRTC are Tucker decomposition [95], CP decomposition [96], trace norm-based approaches [97, 92], and other variants [94, 93]. Similar to collaborative filtering, in the context of LRTC, we treat the unknown evaluations of the new problem as missing values and try to impute them based on the characteristics of existing entries. The motivation results from configurations usually lie in lower-dimensional space due to their mutual correlations. In terms of the LRTC setting, the proposed approach can further utilize the sparse performance data generated from historical problems in the offline stage, which in turn improves the overall efficiency. The significant difference between LRTC and collaborative filtering lies in that LRTC takes the spatial structure of the search space into account, providing more relevant information existing within the performance spaces. To the best of our knowledge, this is the first study in meta-learning under the tensor completion framework.

More specifically, we first acquire the performance data, which is organized as a higher-order (sparse) tensor, from a set of historical problems. Then suitable configurations on the new problems are evaluated to form a testing tensor. To infer the missing entries of the testing tensor, we design LRTC algorithms adopting the sum of the nuclear norm (SNN) [92] model.

Besides LRTC, we further establish a meta-based model via coupled matrix factorization (CMF), bridging the meta-features and performance over the historical problems such that the recommendations can be conducted directly through meta-features. Finally, a strategy combining LRTC and CMF for recommendation is proposed to overcome the shortness of LRTC resulting from the insufficient known entries. Based on the proposed algorithms, a configuration recommendation system is designed for the purpose of various applications. To verify the effectiveness of our approaches, we choose the SVM as well as deep neural networks such as ViT [98, 99] and ResNet [100] as the objective classifiers. The experimental results show that our approaches are able to generate more promising configurations than other existing meta-learning methods, showing the merits of our proposed approaches. In summary, our contributions to this chapter are given below.

- We formulate the configuration recommendation problem as a low-rank tensor completion (LRTC) task, and establish two corresponding algorithms under the nuclear norm model setting;
- A CMF-based configuration recommendation approach is developed, without requiring configuration evaluations for new problems;
- A development for integrating LRTC and CMF into a new algorithm is established to gain better recommendation capacity under the proposed framework;
- The effectiveness of configuration recommendation of the proposed approaches for SVM, ViT [98], and ResNet [100] is demonstrated and verified by comparing with existing approaches.

4.2 PRELIMINARY

Low-rank tensor completion (LRTC), suggested by its name, aims to predict the missing entries of a tensor via the existing entries while keeping its low-rank characteristics. For a given incomplete tensor $\mathcal{T}$, the optimization problem of LRTC is formulated as following:

$$\begin{aligned} &\min_{\mathcal{Y}} \operatorname{rank}_T(\mathcal{Y}) \\ &s.t.\ \mathcal{Y} * \mathcal{M} = \mathcal{T} * \mathcal{M}, \end{aligned} \tag{4.1}$$

where $\operatorname{rank}_T(\cdot)$ indicates a specific tensor rank, such as CP rank, Tucker rank, and Tensor-train rank [101], etc., and "$*$" implies the Hadamard product. $\mathcal{M}$ is a binary tensor, called a mask, that has the same dimension as $\mathcal{T}$, and it is defined as

$$\mathcal{M}_{i_1,\cdots,i_d} = \begin{cases} 1, & \mathcal{T}_{i_1,\cdots,i_d} \neq 0, \\ 0, & \mathcal{T}_{i_1,\cdots,i_d} = 0. \end{cases} \tag{4.2}$$

Especially, when Tucker rank is adopted, the objective function in (4.1) can be rewritten as

$$\begin{aligned} &\min_{\mathcal{Y}} g(\operatorname{rank}_t(\mathcal{Y})) \\ &s.t.\ \mathcal{Y} * \mathcal{M} = \mathcal{T} * \mathcal{M}, \end{aligned} \tag{4.3}$$

where g is a function integrating Tucker rank $\operatorname{rank}_t(\mathcal{Y}) = (\operatorname{rank}(\mathbf{Y}_{(1)}), \cdots, \operatorname{rank}(\mathbf{Y}_{(d)}))$ and $\mathbf{Y}_{(i)}, i = 1, 2, \cdots, d$, represents the unfolding matrix of tensor $\mathcal{Y}$ along mode i. The widely deployed one is a linear combination, i.e., $g(\operatorname{rank}_t(\mathcal{Y})) = \sum_{i=1}^d \alpha_i \cdot \operatorname{rank}(\mathbf{Y}_{(i)})$ where $\sum_{i=1}^d \alpha_i =$

1 are predefined coefficients.

With theoretical supports in the matrix completion problem, we know that the minimization of the rank of a matrix can be conducted by minimizing its nuclear norm, leading to a sum of nuclear norm (SNN) model [92] given by

$$\begin{aligned}
&\min_{\mathbf{Y}_{(i)}, i=1,\cdots,d} \sum_{i=1}^{d} \alpha_i \|\mathbf{Y}_{(i)}\|_* \\
&s.t.\ \mathbf{Y}_{(i)} * \mathbf{M}_{(i)} = \mathbf{T}_{(i)} * \mathbf{M}_{(i)},
\end{aligned} \tag{4.4}$$

where $\mathbf{M}_{(i)}$ is the unfolding matrix of $\mathcal{M}$ along mode i.

When $\mathbf{Y}_{(i)}$ has a low-rank decomposition $\mathbf{Y}_{(i)} = \mathbf{W}_i\mathbf{H}_i$ where $\mathbf{W}_i \in \mathbb{R}^{n_i \times r_i}$ and $\mathbf{H}_i \in \mathbb{R}^{r_i \times \Pi_{j \neq i}^{d} n_j}$, we then have $\|\mathbf{Y}_{(i)}\|_* = \min_{\mathbf{W}_i,\mathbf{H}_i}(\|\mathbf{W}_i\|_F^2 + \|\mathbf{H}_i\|_F^2)$, see e.g., [102]. Thus, the optimization model can be formulated as

$$\begin{aligned}
&\min_{\mathbf{W}_i,\mathbf{H}_i,\mathbf{Y}_{(i)}} \sum_{i=1}^{d} \alpha_i \left[\frac{1}{2}\|\mathbf{Y}_{(i)} - \mathbf{W}_i\mathbf{H}_i\|_F^2 + \frac{\beta_i}{2}(\|\mathbf{W}_i\|_F^2 + \|\mathbf{H}_i\|_F^2)\right] \\
&s.t.\ \mathbf{Y}_{(i)} * \mathbf{M}_{(i)} = \mathbf{T}_{(i)} * \mathbf{M}_{(i)},
\end{aligned} \tag{4.5}$$

where β_i's are the coefficients that control the degree of penalty on the factor matrices $\mathbf{W}_i$ and $\mathbf{H}_i$ in the objective function. Notice that model (4.5) is degraded to a probabilistic matrix factorization (PMF) model [48] if $\alpha_d = 1$, and $\alpha_i = 0, i = 1, 2, \cdots, d-1$.

4.3 KERNEL LOW-RANK TENSOR COMPLETION

This section presents our four sequential algorithms that are leveraged to predict the performance of the configurations on a new classification problem.

4.3.1 Data organization

Assume that $\mathscr{A}$ is the given classifier with $d-1$ hyperparameters, and each hyperparameter has the search space Ω_i with the cardinality of n_i, i.e.,

$$|\Omega_i| = n_i, i = 1, 2, \cdots, d-1. \tag{4.6}$$

Then the configuration search space $\boldsymbol{\Omega} = \Omega_1 \circ \Omega_2 \circ \cdots \circ \Omega_{d-1}$ lies in an $n_1 \times \cdots \times n_{d-1}$ multidimensional space. Here, $\circ$ denotes the outer product. Therefore, the performance data on a classification problem can be formed as a tensor $\mathcal{T}'$ with the dimension of $n_1 \times \cdots \times n_{d-1}$.

Now suppose that we have N historical problems $\mathscr{D}_1, \cdots, \mathscr{D}_N$ on which the performance of classifier $\mathscr{A}$ is already known, namely, $\mathcal{T}'_1, \cdots, \mathcal{T}'_N$, then we can generate a higher-order tensor $\bar{\mathcal{T}} \in \mathbb{R}^{n_1 \times \cdots \times n_{d-1} \times N}$ where $\mathcal{T}'_i = \bar{\mathcal{T}}(:, \cdots, :, i), i = 1, \cdots, N$. That is, $\mathcal{T}'_i$'s are the frontal slices of $\mathcal{T}$. Given a new problem $\mathscr{D}$ at hand in which the performance evaluations on some configurations are still not determined yet. Denoting this incomplete performance tensor as $\tilde{\mathcal{T}}'$, we can combine $\bar{\mathcal{T}}$ and $\tilde{\mathcal{T}}'$ as $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_{d-1} \times n_d}$ in which $n_d = N + 1$ and $\tilde{\mathcal{T}}' = \mathcal{T}(:, \cdots, :, n_d)$. In Section 4.4, we will further illustrate how to form the $\mathcal{T}$ for real problems. Our goal is to recover the missing entries of $\tilde{\mathcal{T}}$ as accurately as possible via the LRTC approach. An example of data organization is illustrated in Figure 4.2 where SVM-RBF has two hyperparameters `C` and `gamma` with a length of 21 and 19 respectively.

4.3.2 Performance estimation via LRTC

To complete the unknown entries of the performance tensor $\mathcal{T}$, we look for the optimal factor matrices $\mathbf{W}_i, \mathbf{H}_i, i = 1, \cdots, d$, as stated in problem (4.5). By fixing mode i, we have

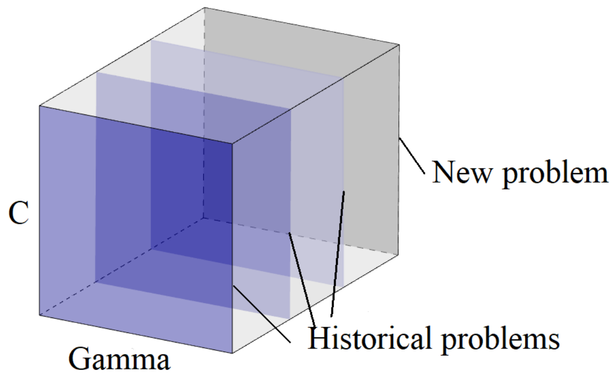


Figure 4.2: An example of 3D data organization. The first three frontal slices are the performance data of SVM on three historical problems while the fourth slice is the incomplete performance data on a new problem required to be completed.

the following sub-problem

$$\min_{\mathbf{W}_i,\mathbf{H}_i,\mathbf{Y}_{(i)}} \frac{1}{2}\|\mathbf{Y}_{(i)} - \mathbf{W}_i\mathbf{H}_i\|_F^2 + \frac{\beta_i}{2}(\|\mathbf{W}_i\|_F^2 + \|\mathbf{H}_i\|_F^2)$$

$$s.t.\ \mathbf{Y}_{(i)} * \mathbf{M}_{(i)} = \mathbf{T}_{(i)} * \mathbf{M}_{(i)}.$$

The objective function presented here is nonconvex because three variables, $\mathbf{W}_i, \mathbf{H}_i$, and $\mathbf{Y}_{(i)}$, are involved. Therefore, the block coordinate descent method is adopted which alternatively updates one variable while fixing the other two in each iteration. The updating rules for $\mathbf{W}_i, \mathbf{H}_i$, and $\mathbf{Y}_{(i)}$ in $(l+1)^{th}$ iteration are the following

$$\mathbf{H}_i^{l+1} = [\mathbf{W}_i^{l\top}\mathbf{W}_i^l + \beta_i\mathbf{I}_{r_i}]^\dagger \mathbf{W}_i^{l\top}\mathbf{Y}_{(i)}^l \tag{4.7}$$

$$\mathbf{W}_i^{l+1} = \mathbf{Y}_{(i)}^l\mathbf{H}_i^{l+1\top}(\mathbf{H}_i^{l+1}\mathbf{H}_i^{l+1\top} + \beta_i\mathbf{I}_{r_i})^\dagger \tag{4.8}$$

$$\mathbf{Y}_{(i)}^{l+1} = \mathbf{W}_i^{l+1}\mathbf{H}_i^{l+1} \tag{4.9}$$

$$\mathbf{Y}_{(i)}^{l+1} = \mathbf{T}_{(i)} * \mathbf{M}_{(i)} + \mathbf{Y}_{(i)}^{l+1} * (1 - \mathbf{M}_{(i)}) \tag{4.10}$$

where $\mathbf{A}^{\dagger}$ stands for the Moore-Penrose pseudoinverse if $\mathbf{A}$ is singular, otherwise it denotes the inverse of $\mathbf{A}$. Notice that $\mathbf{Y}_{(i)}^{0} = \mathbf{T}_{(i)}$. When the updating procedure is completed, the recovered tensor $\mathcal{Y}$ will be reconstructed as $\mathcal{Y} = \sum_{i=1}^{d} \alpha_i \cdot \text{fold}(\mathbf{Y}_{(i)})$ where fold$(\cdot)$ is to fold a matrix into a tensor. Hence, the predictive performance $\tilde{\mathcal{Y}}$ on the new problem can be extracted from $\mathcal{Y}$, i.e., $\tilde{\mathcal{Y}} = \mathcal{Y}(:, \cdots, :, n_d)$.

4.3.3 Performance estimation via kernel LRTC

In (4.5), we assume that each $\mathbf{Y}_{(i)}$ is linearly decomposed associated with $\mathbf{W}_i$ and $\mathbf{H}_i$; however, this underlying relationship essentially is nonlinear. To model this nonlinearity, in this subsection, we propose a kernelized LRTC (KLRTC) algorithm. Assume that ϕ is a nonlinear mapping, the objective function of KLRTC is given by

$$\begin{aligned} &\min_{\mathbf{W}_i, \mathbf{H}_i, \mathbf{Y}_{(i)}} \sum_{i=1}^{d} \alpha_i \left[\frac{1}{2} \|\phi(\mathbf{Y}_{(i)}) - \phi(\mathbf{W}_i)\mathbf{H}_i\|_F^2 + \frac{\beta_i}{2} (\|\mathbf{W}_i\|_F^2 + \|\mathbf{H}_i\|_F^2) \right], \\ &s.t.\ \mathbf{Y}_{(i)} * \mathbf{M}_{(i)} = \mathbf{T}_{(i)} * \mathbf{M}_{(i)}. \end{aligned} \tag{4.11}$$

Although we do not the know the explicit form of ϕ, their inner products can be calculated by a kernel function $k(\cdot,\cdot)$, i.e., $\phi(\mathbf{x}_1)^{\top}\phi(\mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2)$. The two we adopted are polynomial (i.e., $k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^{\top}\mathbf{x}_2 + c)^q$) and RBF (i.e., $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{1}{\sigma}\|\mathbf{x}_1 - \mathbf{x}_2\|^2)$) kernels. Notice that if $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^{\top}\mathbf{x}_2$, then (4.11) is equivalent to (4.5).

Denote $\mathbf{K}_{\mathbf{YY}} = \phi(\mathbf{Y})^{\top}\phi(\mathbf{Y})$, $\mathbf{K}_{\mathbf{YW}} = \phi(\mathbf{Y})^{\top}\phi(\mathbf{W})$, and $\mathbf{K}_{\mathbf{WW}} = \phi(\mathbf{W})^{\top}\phi(\mathbf{W})$. Then the objective function, for each mode i, can be formulated as

$$\begin{aligned} &\min_{\mathbf{W}_i, \mathbf{H}_i, \mathbf{Y}_{(i)}} \frac{1}{2}\text{tr}\left(\mathbf{K}_{\mathbf{Y}_{(i)}\mathbf{Y}_{(i)}} - 2\mathbf{K}_{\mathbf{Y}_{(i)}\mathbf{W}_i}^{\top}\mathbf{H}_i + \mathbf{H}_i^{\top}\mathbf{K}_{\mathbf{W}_i\mathbf{W}_i}\mathbf{H}_i \right) + \frac{\beta_i}{2}\left(\|\mathbf{W}_i\|_F^2 + \|\mathbf{H}_i\|_F^2 \right), \\ &s.t.\ \mathbf{Y}_{(i)} * \mathbf{M}_{(i)} = \mathbf{T}_{(i)} * \mathbf{M}_{(i)}. \end{aligned} \tag{4.12}$$

We also adopt block coordinate descent to update $\mathbf{W}_i, \mathbf{H}_i$, and $\mathbf{Y}_{(i)}$. Suppose $\mathbf{W}_i^l, \mathbf{H}_i^l$, and $\mathbf{Y}_{(i)}^l$ are known and we seek the next updating $\mathbf{W}_i^{l+1}, \mathbf{H}_i^{l+1}$, and $\mathbf{Y}_{(i)}^{l+1}$.

Update $\mathbf{H}_i$: As $\mathbf{H}_i$ is free of kernel functions, so the updating of $\mathbf{H}_i$ is straightforward. Removing the terms that do not involve variable $\mathbf{H}_i$ in (4.12), we have

$$\begin{aligned} \mathbf{H}_i^{l+1} &= \min_{\mathbf{H}_i} \frac{1}{2} \mathrm{tr}\left(\mathbf{H}_i^\top \mathbf{K}_{\mathbf{W}_i^l \mathbf{W}_i^l} \mathbf{H}_i - 2\mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i^l}^\top \mathbf{H}_i\right) + \frac{\beta_i}{2} \|\mathbf{H}_i\|_F^2 \\ &= (\mathbf{K}_{\mathbf{W}_i^l \mathbf{W}_i^l} + \beta_i \mathbf{I}_{r_i})^\dagger \mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i^l}^\top. \end{aligned} \tag{4.13}$$

Update $\mathbf{W}_i$: Removing the terms that do not involve variable $\mathbf{W}_i$ in (4.12), we have,

$$\mathbf{W}_i^{l+1} = \min_{\mathbf{W}_i} \frac{1}{2} \mathrm{tr}\left(\mathbf{H}_i^{l+1\top} \mathbf{K}_{\mathbf{W}_i \mathbf{W}_i} \mathbf{H}_i^{l+1} - 2\mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i}^\top \mathbf{H}_i^{l+1}\right) + \frac{\beta_i}{2} \|\mathbf{W}_i\|_F^2.$$

Next, we derive the updating rule for $\mathbf{W}_i$ when a specific kernel function is employed. For polynomial kernel, the gradient with respect to $\mathbf{W}_i$ is given by

$$\Delta_{\mathbf{W}_i} = -\mathbf{Y}_{(i)}^l \left(\mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i}^{(q-1)} * \mathbf{H}_i^{l+1\top}\right) + \mathbf{W}_i \left[(\mathbf{H}_i^{l+1} \mathbf{H}_i^{l+1\top}) * \mathbf{K}_{\mathbf{W}_i \mathbf{W}_i}^{(q-1)} + \beta_i \mathbf{I}_{r_i}\right],$$

where $(K_{YW}^{(q-1)})_{ij} = (\mathbf{y}_i^\top \mathbf{w}_j + c)^{q-1}$ and $(K_{WW}^{(q-1)})_{ij} = (\mathbf{w}_i^\top \mathbf{w}_j + c)^{q-1}$. Let $\Delta_{\mathbf{W}_i} = 0$. By using the iteratively reweighted optimization, i.e., fixing $\mathbf{W}_i$ as $\mathbf{W}_i^l$ in the kernel matrices $\mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i^l}^{(q-1)}$ and $\mathbf{K}_{\mathbf{W}_i^l \mathbf{W}_i^l}$, we solve for $\mathbf{W}_i$ as

$$\mathbf{W}_i^{l+1} = \mathbf{Y}_{(i)}^l \left(\mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i^l}^{(q-1)} * \mathbf{H}_i^{l+1\top}\right) \left[(\mathbf{H}_i^{l+1} \mathbf{H}_i^{l+1\top}) * \mathbf{K}_{\mathbf{W}_i^l \mathbf{W}_i^l}^{(q-1)} + \beta_i \mathbf{I}_{r_i}\right]^\dagger. \tag{4.14}$$

Similarly, the gradient with respect to $\mathbf{W}_i$, when RBF kernel is adopted, is given by

$$\Delta_{\mathbf{W}_i} = \frac{2}{\sigma}\mathbf{W}_i\left(\mathbf{\Gamma}_1 + \mathbf{Q}_2 - \mathbf{\Gamma}_2 + \frac{\sigma\beta_i}{2}\mathbf{I}_{r_i}\right) - \frac{2}{\sigma}\mathbf{Y}^l_{(i)}\mathbf{Q}_1,$$

where $\mathbf{Q}_1 = \mathbf{H}_i^{l+1\top} * \mathbf{K}_{\mathbf{Y}^l_{(i)}\mathbf{W}^l_i}, \mathbf{Q}_2 = (\mathbf{H}_i^{l+1}\mathbf{H}_i^{l+1\top}) * \mathbf{K}_{\mathbf{W}^l_i\mathbf{W}^l_i}$, and $\mathbf{\Gamma}_1 = \mathrm{diag}(\mathbf{1}^\top\mathbf{Q}_1), \mathbf{\Gamma}_2 = \mathrm{diag}(\mathbf{1}^\top\mathbf{Q}_2)$. Then, by iteratively reweighted optimization, one arrives at

$$\mathbf{W}_i^{l+1} = \mathbf{Y}^l_{(i)}\mathbf{Q}_1\left(\mathbf{\Gamma}_1 + \mathbf{Q}_2 - \mathbf{\Gamma}_2 + \frac{\sigma\beta_i}{2}\mathbf{I}_{r_i}\right)^\dagger. \tag{4.15}$$

Update $\mathbf{Y}_{(i)}$: Removing the terms that do not involve variable $\mathbf{Y}_{(i)}$ in (4.12), we have,

$$\mathbf{Y}^{l+1}_{(i)} = \min_{\mathbf{Y}_{(i)}} \frac{1}{2}\mathrm{tr}\left(\mathbf{K}_{\mathbf{Y}_{(i)}\mathbf{Y}_{(i)}} - 2\mathbf{K}^\top_{\mathbf{Y}_{(i)}\mathbf{W}^{l+1}_i}\mathbf{H}^{l+1}_i\right). \tag{4.16}$$

For polynomial kernel, the gradient of (4.16) with respect to $\mathbf{Y}_{(i)}$ is

$$\Delta_{\mathbf{Y}_{(i)}} = -\mathbf{Y}_{(i)}\left(\mathbf{K}^{(q-1)}_{\mathbf{Y}^l_{(i)}\mathbf{Y}^l_{(i)}} * \mathbf{I}_n\right) + \mathbf{W}^{l+1}_i\left((\mathbf{K}^{(q-1)}_{\mathbf{Y}^l_{(i)}\mathbf{W}^{l+1}_i})^\top * \mathbf{H}^{l+1}_i\right),$$

where $(K^{(q-1)}_{YY})_{ij} = (\mathbf{y}_i^\top\mathbf{y}_j + c)^{q-1}$, so the updating rule of $\mathbf{Y}_{(i)}$ is given by

$$\mathbf{Y}^{l+1}_{(i)} = \mathbf{W}^{l+1}_i\left((\mathbf{K}^{(q-1)}_{\mathbf{Y}^l_{(i)}\mathbf{W}^{l+1}_i})^\top * \mathbf{H}^{l+1}_i\right)\left(\mathbf{K}^{(q-1)}_{\mathbf{Y}^l_{(i)}\mathbf{Y}^l_{(i)}} * \mathbf{I}_n\right)^\dagger. \tag{4.17}$$

For RBF kernel, $\mathrm{tr}(\mathbf{K}_{\mathbf{Y}_{(i)}\mathbf{Y}_{(i)}})$ is removed from (4.16) since it is a constant. Its gradient for $\mathbf{Y}_{(i)}$ is given by

$$\Delta_{\mathbf{Y}_{(i)}} = \frac{2}{\sigma}\left(\mathbf{\Gamma}_3\mathbf{Y}_{(i)} - \mathbf{W}^{l+1}_i\mathbf{Q}_3\right),$$

Algorithm 4 `LRTC`$(\bar{\mathcal{T}}, \mathbf{r}, \boldsymbol{\alpha}, \mathscr{D})$

```
Input: Training tensor $\bar{\mathcal{T}}$, latent dimensions $\mathbf{r} = [r_1, \cdots, r_d]$, SNN coefficients $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_d]$, new problem $\mathscr{D}$
Output: Predictive performance $\tilde{\mathcal{Y}}$
 1: Generate the testing tensor $\tilde{\mathcal{T}}$ on problem $\mathscr{D}$
 2: Initialize low-rank tensor: $\mathcal{Y}^0 = \mathcal{T}$
 3: Initialize factor matrices: $\mathbf{W}_i^0 \in \mathbb{R}^{n_i \times r_i}, i = 1, \cdots, d$
 4: while stop criteria is unsatisfied do
 5:     for $i = 1$ to $d$ do
 6:         Update $\mathbf{H}_i^{l+1}$ using (4.7) or (4.13)
 7:         Update $\mathbf{W}_i^{l+1}$ using (4.8) or (4.14) or (4.15)
 8:         Update $\mathbf{Y}_{(i)}^{l+1}$ using (4.9) or (4.17) or (4.18)
 9:         Update $\mathbf{Y}_{(i)}^{l+1}$ using (4.10)
10:     end for
11:     $\mathcal{Y}^{l+1} = \sum_{i=1}^d \alpha_i \cdot \text{fold}(\mathbf{Y}_{(i)}^{l+1})$
12: end while
13: $\tilde{\mathcal{Y}}_{LRTC} = \mathcal{Y}^{l+1}(:, \cdots, :, n_d)$
```

where $\mathbf{Q}_3 = \mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{W}_i^{l+1}}^\top * \mathbf{H}_i^{l+1}$, and $\boldsymbol{\Gamma}_3 = \text{diag}(\mathbf{1}^\top \mathbf{Q}_3)$. Hence, the updating rule for $\mathbf{Y}_{(i)}^{l+1}$ is given by

$$\mathbf{Y}_{(i)}^{l+1} = \boldsymbol{\Gamma}_3^\dagger \left(\mathbf{W}_i^{l+1} \mathbf{Q}_3 \right). \tag{4.18}$$

The inverse matrices in (4.17) and (4.18) can be updated efficiently since $\mathbf{K}_{\mathbf{Y}_{(i)}^l \mathbf{Y}_{(i)}^l}^{(q-1)} * \mathbf{I}_n$ and $\boldsymbol{\Gamma}_3$ are diagonal. Finally, $\mathbf{Y}_{(i)}^{l+1}$ will be further updated through (4.10).

The workflow of (kernel) LRTC is depicted in `Algorithm 4`, and how to generate the testing tensor $\tilde{\mathcal{T}}$ on a problem $\mathscr{D}$ will be given in next section.

4.3.4 Coupled matrix factorization (CMF)

In this subsection, we develop an auxiliary algorithm that either can be employed independently or performed jointly with `Algorithm 4` to enhance the reliability of outcomes. Again, assume that there are N historical classification problems $\mathscr{D}_1, \cdots, \mathscr{D}_N$, and from each one we can extract a set of meta-features. Let f be the meta-feature extractor, then the meta-feature $\mathbf{x}_i \in \mathbb{R}^p, i = 1, \cdots, N$, can be represented as $\mathbf{x}_i = f(\mathscr{D}_i)$ where p is the dimension of meta-features.

Denote $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times p}$, then an objective function combining with the meta-feature $\mathbf{X}$ and training tensor $\bar{\mathcal{T}}$ can be established as follows

$$\min_{\mathbf{W},\mathbf{H},\mathbf{V}} \frac{1}{2}\|\bar{\mathbf{T}}_{(d)} - \mathbf{W}\mathbf{H}\|_F^2 + \frac{\gamma}{2}\|\mathbf{X} - \mathbf{W}\mathbf{V}\|_F^2 + \frac{\beta}{2}(\|\mathbf{W}\|_F^2 + \|\mathbf{H}\|_F^2 + \|\mathbf{V}\|_F^2), \tag{4.19}$$

where $\bar{\mathbf{T}}_{(d)} \in \mathbb{R}^{N \times \prod_{i=1}^{d-1} n_i}$, $\mathbf{W} \in \mathbb{R}^{N \times r}$, $\mathbf{H} \in \mathbb{R}^{r \times \prod_{i=1}^{d-1} n_i}$, and $\mathbf{V} \in \mathbb{R}^{r \times p}$. $\gamma > 0$ is the coefficient that adjusts the influence of the meta-feature in (4.19) and $\beta > 0$ is a regularization parameter. The rationale of (4.19) is to optimize the relationship between the meta-features and the classification performance over historical problems by sharing one common matrix $\mathbf{W}$.

Similarly, block-coordinate descent is applied to solve the optimization problem (4.19), and three updating rules can be acquired as follows

$$\mathbf{H}^{l+1} = (\mathbf{W}^{l\top}\mathbf{W}^l + \beta\mathbf{I}_r)^\dagger \mathbf{W}^{l\top}\bar{\mathbf{T}}_{(d)} \tag{4.20}$$

$$\mathbf{V}^{l+1} = (\mathbf{W}^{l\top}\mathbf{W}^l + \beta\mathbf{I}_r)^\dagger \mathbf{W}^{l\top}\mathbf{X}. \tag{4.21}$$

$$\mathbf{W}^{l+1} = (\bar{\mathbf{T}}_{(d)}\mathbf{H}^{l+1\top} + \gamma\mathbf{X}\mathbf{V}^{l+1\top})(\mathbf{H}^{l+1}\mathbf{H}^{l+1\top} + \gamma\mathbf{V}^{l+1}\mathbf{V}^{l+1\top} + \beta\mathbf{I}_r)^\dagger. \tag{4.22}$$

Next, we provide an effective way for performance estimation in the testing phase. Suppose $\mathscr{D}$ is a new classification problem at hand, its meta-feature is first extracted as $\mathbf{x} = f(\mathscr{D})$, then we have the decomposition $\mathbf{x}^\top = \mathbf{w}\mathbf{V}$, which leads to $\mathbf{w} = \mathbf{x}^\top\mathbf{V}^\dagger$, and the performance prediction is finally given by $\mathbf{y} = \mathbf{w}\mathbf{H} = \mathbf{x}^\top\mathbf{V}^\dagger\mathbf{H}$, and the prediction in

Algorithm 5 `CMF`$(\bar{\mathcal{T}}, \mathbf{X}, r, \beta, \gamma, \mathscr{D})$

```
Input: Training tensor $\bar{\mathcal{T}}$, training meta-features $\mathbf{X}$, latent dimension $r$, regularization coefficient $\beta$, balance coefficient $\gamma$, new classification problem $\mathscr{D}$
Output: Predictive performance $\tilde{\mathcal{Y}}$
1: Initialize factor matrix: $\mathbf{W}^0 \in \mathbb{R}^{N \times r}$
2: while stop criteria is unsatisfied do
3:     Update $\mathbf{H}$ using (4.20)
4:     Update $\mathbf{V}$ using (4.21)
5:     Update $\mathbf{W}$ using (4.22)
6: end while
7: Meta-feature extraction: $\mathbf{x} = f(\mathscr{D})$
8: $\tilde{\mathcal{Y}}_{CMF} = \text{fold}(\mathbf{x}^\top \mathbf{V}^\dagger \mathbf{H})$
```

tensor format is presented as $\tilde{\mathcal{Y}}_{CMF} = \text{fold}(\mathbf{y})$. Notice that this prediction strategy does not need any configuration evaluations, we treat it as coupled matrix factorization (CMF). This procedure is summarized in `Algorithm 5`.

4.3.5 Combining CMF and LRTC

The success of `Algorithm 4` relies on the ratio of the missing values, and the performance of completion could be poor if the known entries are very limited. Some theoretical analysis on how the missing ratio impacts the recovery performance can be found in [103, 104]. To overcome this potential issue in real scenarios, an algorithm integrating LRTC and CMF is developed, and we denote it as LRTC-CMF.

Let $\tilde{\mathcal{Y}}_{LRTC}$ and $\tilde{\mathcal{Y}}_{CMF}$ be the predictive performance tensor outputs obtained from the `Algorithm 4` and the `Algorithm 5`, respectively. Then the joint prediction of both LRTC and CMF is derived as

$$\tilde{\mathcal{Y}} = \theta \tilde{\mathcal{Y}}_{LRTC} + (1 - \theta) \tilde{\mathcal{Y}}_{CMF}, \tag{4.23}$$

where $\theta \in [0, 1]$ balances the weight of each output. Clearly, $\tilde{\mathcal{Y}} = \tilde{\mathcal{Y}}_{LRTC}$ when $\theta = 1$ and $\tilde{\mathcal{Y}} = \tilde{\mathcal{Y}}_{CMF}$ when $\theta = 0$. By adding auxiliary information (either meta-features or partial

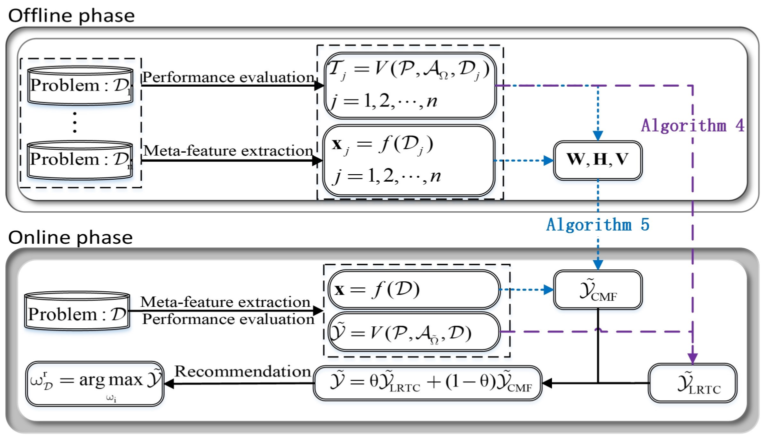

Offline phase
Problem : $\mathcal{D}_1$
Problem : $\mathcal{D}_n$
Performance evaluation
Meta-feature extraction
$\mathcal{T}_j = V(\mathcal{P}, \mathcal{A}_\Omega, \mathcal{D}_j)$
$j = 1, 2, \cdots, n$
$\mathbf{x}_j = f(\mathcal{D}_j)$
$j = 1, 2, \cdots, n$
$\mathbf{W}, \mathbf{H}, \mathbf{V}$
Algorithm 4
Algorithm 5
Online phase
Problem : $\mathcal{D}$
Meta-feature extraction
Performance evaluation
$\mathbf{x} = f(\mathcal{D})$
$\tilde{\mathcal{Y}} = V(\mathcal{P}, \mathcal{A}_{\tilde{\Omega}}, \mathcal{D})$
$\tilde{\mathcal{Y}}_{\mathrm{CMF}}$
$\tilde{\mathcal{Y}}_{\mathrm{LRTC}}$
$\tilde{\mathcal{Y}} = \theta\tilde{\mathcal{Y}}_{\mathrm{LRTC}} + (1-\theta)\tilde{\mathcal{Y}}_{\mathrm{CMF}}$
Recommendation
$\omega_{\mathcal{D}}^{r} = \arg\max_{\omega_i} \tilde{\mathcal{Y}}$

Figure 4.3: The automatic configuration recommendation framework based on our proposed approaches.

true evaluations) to the problem, both LRTC and CMF can interact with each other through the (4.23) setting.

4.4 AUTOMATIC CONFIGURATION RECOMMENDATION

To apply our proposed algorithms to configuration recommendation tasks, we design an automatic configuration recommendation system in conjunction with the general meta-learning framework [42], and its workflow, which comprises offline (training) and online (testing) stages, is illustrated in Figure 4.3. Detailed descriptions are given below.

4.4.1 Offline stage

The offline stage includes performance evaluation and meta-feature extraction on a group of historical problems.

Classification performance evaluation: The classification performance $\mathcal{T}$ on a specific problem $\mathscr{D}$ can be formulated as $\mathcal{T} = V(\mathcal{P}, \mathscr{A}_{\mathbf{\Omega}}, \mathscr{D})$ in which $V(\cdot,\cdot,\cdot)$ is a validation scheme and $\mathcal{P}$ is a performance evaluation metric that is should be determined. We set balanced classification accuracy as $\mathcal{P}$ and 10-fold stratified cross-validation as $V(\cdot,\cdot,\cdot)$, respectively, and they both take the imbalance of classes that usually exists in real-world problems into account. Since the training tensor of our approaches can be sparse, we can set a maximal budget for each configuration evaluation, which can potentially improve the efficiency of offline preparation. If the evaluation cannot be done within the maximal budget, then a zero will be returned. Obviously, a smaller allocated budget will result in a higher degree of sparsity in data.

Meta-feature extraction: The extraction of meta-features includes the computations of a set of predefined characteristics, or meta-measures, on the given problems. To the best of our knowledge, there are mainly eight types of meta-features that are widely leveraged for the meta-learning in literature, and they are based on various assumptions and hypothesizes; namely, statistics and information theory (SIT), model structure (MS), classification problem complexity (PC), clustering, structural information (SI), (relative) landmarking (LM), and concept. An overview of meta-features is available in [105]. The public Python software `pymfe` [23] is adopted in our system to perform meta-feature extraction, in which the users can customize which meta-measures to extract independently or jointly among the eight types of meta-features. Due to the variety of meta-measures, the min-max scaling strategy is applied to normalize the range of each meta-measure into interval $[0, 1]$ after the raw meta-measure values are computed such that each of them has equal importance for the given problem.

4.4.2 Online stage

In the online phase, we next explain how to prepare the testing tensor $\tilde{\mathcal{T}}$ and the recommendation scheme of our system when the predictive performance $\tilde{\mathcal{Y}}$ is generated.

Partial configuration evaluation: We deploy the exact same 10-fold cross-validation strategy we designed for the historical performance evaluation to partially evaluate the configurations to generate a testing tensor $\tilde{\mathcal{T}}$ for a new problem $\mathscr{D}$. The only difference lies in that a proportion of configurations $\tilde{\boldsymbol{\Omega}}_{\mathscr{A}}$ will be evaluated rather than the whole search space $\boldsymbol{\Omega}_{\mathscr{A}}$. Suppose that the prescribed *keeping ratio* is ϵ_{test} ($\epsilon_{test} \in (0,1)$), then $|\boldsymbol{\Omega}_{\mathscr{A}}|\epsilon_{test}$ points are going to be tested. The candidate configurations can be searched by either random selection or Bayesian optimization, or other search algorithms. We set a budget large enough to ensure that every configuration can be chosen and no budget is wasted in our experiments. When $|\boldsymbol{\Omega}_{\mathscr{A}}|\epsilon_{test}$ is pretty small compared to $|\boldsymbol{\Omega}_{\mathscr{A}}|$, a feasible trick to initialize $\tilde{\mathcal{T}}$ is to fill the zero entries with the average of the known entries. Let Γ be the index set of the trained configurations, then the testing tensor can be augmented by the following rule

$$\tilde{\mathcal{T}}(i, \cdots, j) = \begin{cases} \tilde{\mathcal{T}}(i, \cdots, j), & (i, \cdots, j) \in \Gamma, \\ \\ \frac{\texttt{sum}(\tilde{\mathcal{T}}_\Gamma)}{|\Gamma|}, & (i, \cdots, j) \notin \Gamma, \end{cases} \tag{4.24}$$

where $\texttt{sum}(\cdot)$ sums up all the entries of an array.

Configuration recommendation: Suppose that a predictive performance tensor $\tilde{\mathcal{Y}}$ is given for a problem $\mathscr{D}$, then the configuration(s) that has/have the highest predictive score is/are chosen as the candidate for problem $\mathscr{D}$. Detailedly, we are first to order the entries of $\tilde{\mathcal{Y}}$ in descending tendency, such as $y_{\boldsymbol{\omega}_{i_1}} \geq y_{\boldsymbol{\omega}_{i_2}} \geq \cdots \geq y_{\boldsymbol{\omega}_{i_m}}$ and $m = \prod_{i=1}^{d-1} n_i$. Then, configuration $\boldsymbol{\omega}_{i_1}$ is the selected candidate for problem $\mathscr{D}$. Since the optimal configuration on a problem is usually not unique, we can also recommend multiple candidates, i.e., $\boldsymbol{\omega}^r_{\mathscr{D}} = \{\boldsymbol{\omega}_{i_1}, \boldsymbol{\omega}_{i_2}, \cdots, \boldsymbol{\omega}_{i_c}\}$ where $c \ll m$, and $\boldsymbol{\omega}^r_{\mathscr{D}}$ implies the recommended candidates for problem

$\mathscr{D}$. Later, we call $\boldsymbol{\omega}_{i_j} \in \boldsymbol{\omega}^r_{\mathscr{D}}$ as the j^{th}-order recommendation.

4.5 EMPIRICAL VALIDATION

This section presents the effectiveness of our proposed method in real applications. Before the discussion of empirical results, we first introduce the experimental settings employed, including the evaluated classifiers, adopted classification problems, and the recommendation performance evaluation metrics.

4.5.1 Experimental settings

Evaluated classifiers: We here study the configuration selection for the classical support-vector machines (SVM) and the state-of-the-art deep neural network classifiers such as vision transformer (ViT) [98] and residual convolutional neural network (ResNet) [100] as well.

We consider RBF kernel SVM which has two important hyperparameters: regularization coefficient '`C`' and RBF kernel width '`gamma`'. Suggested by literature [71], we set the search ranges of '`C`' and '`gamma`' are $2^{-5:1:15}$ and $2^{-15:1:3}$, respectively. Therefore, we have $21 \times 19 = 399$ candidate configurations in total. For the implementations, we adopt the software `SVC` in the `scikit-learn` library.

On the other hand, we adopt the trained ViT (ViT-B/32 [98]) and ResNet (ResNet101 [100]) models and just fine-tune them on new problems. Studies about neural architecture search can be found in [106, 107, 108]. We consider five hyperparameters: '`dropout`' (rate is 0.5) or not, '`optmizier`' (Adam, SGD, RMSprop), '`learning_rate`' (with starting rate 0.1, 0.01, 0.001, 0.0001), '`batch_size`' (32, 128, 512), and '`epochs`' (10, 20, 30, 40, 50). In total, there are $2 \times 3 \times 4 \times 3 \times 5 = 360$ candidate configurations. We run all training on `Colab` GPUs or TPUs.

Table 4.1: The comparisons of ACA (%), ARA (%), HR (%), and MRR between LRTC and MFCF under various keeping ratios when three types of kernels are employed. The maximal value of each column is highlighted. The ACA of the real optimal configurations on the testing problems is 79.08%.

Methods	$\epsilon_{test} = 0.25\%$				$\epsilon_{test} = 0.50\%$				$\epsilon_{test} = 1\%$			
	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR
LRTC-Lin	74.09	81.17	77.12	0.0929	73.79	81.73	78.94	0.0889	73.99	82.41	78.33	0.0984
LRTC-Poly	74.27	81.58	77.73	**0.0964**	**74.65**	**82.48**	**79.24**	0.0921	**75.00**	**84.28**	**81.21**	0.0881
LRTC-RBF	**74.32**	**81.88**	77.73	0.0941	74.02	81.68	78.94	**0.0970**	74.90	83.77	81.06	**0.1085**
MFCF-Lin	73.78	80.09	77.12	0.0899	73.77	80.51	78.48	0.0934	73.39	80.70	75.76	0.0875
MFCF-Poly	73.81	80.58	**78.64**	0.0941	73.80	80.63	78.79	0.0939	73.67	80.81	76.36	0.0927
MFCF-RBF	74.03	80.28	75.45	0.0706	74.06	80.35	75.30	0.0801	74.66	81.61	77.73	0.0819

Methods	$\epsilon_{test} = 2\%$				$\epsilon_{test} = 3\%$				$\epsilon_{test} = 4\%$			
	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR
LRTC-Lin	75.98	87.67	84.24	0.1328	76.69	90.84	87.27	0.1432	77.11	91.75	89.09	0.1362
LRTC-Poly	**76.05**	**88.17**	**85.30**	0.1301	76.90	90.40	**88.79**	0.1490	**77.22**	92.32	**90.45**	0.1638
LRTC-RBF	**76.05**	87.90	85.15	**0.1347**	**76.93**	**90.94**	**88.79**	**0.1509**	77.20	**92.37**	**90.45**	0.1623
MFCF-Lin	75.77	87.18	83.64	0.1282	76.67	90.63	87.12	0.1383	77.11	91.74	89.09	0.1411
MFCF-Poly	75.92	87.39	84.39	0.1285	76.68	88.91	87.58	0.1402	77.16	91.88	90.15	**0.1639**
MFCF-RBF	75.81	87.46	85.00	0.1169	76.85	90.87	**88.79**	0.1263	77.13	92.02	**90.45**	0.1431

Methods	$\epsilon_{test} = 5\%$				$\epsilon_{test} = 10\%$				$\epsilon_{test} = 20\%$			
	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR
LRTC-Lin	77.63	**93.97**	91.82	0.1736	**78.22**	**96.35**	**96.21**	0.2401	**78.61**	97.87	**97.42**	0.3714
LRTC-Poly	**77.74**	93.84	**92.42**	0.1959	78.21	96.14	**96.21**	**0.2427**	76.60	97.49	97.27	0.3645
LRTC-RBF	**77.74**	93.90	**92.42**	**0.1974**	**78.22**	96.29	**96.21**	0.2417	**78.61**	**97.93**	**97.42**	**0.3732**
MFCF-Lin	77.63	93.86	91.82	0.1793	**78.22**	96.30	**96.21**	0.2401	**78.61**	97.92	**97.42**	0.3680
MFCF-Poly	77.67	93.22	91.36	0.1954	78.15	95.51	95.45	0.2390	78.49	96.26	95.76	0.3529
MFCF-RBF	77.71	93.87	92.42	0.1795	78.13	95.85	**96.21**	0.2226	78.48	97.07	97.12	0.3292

Classification problems: For SVM, we collected 136 classification problems from the UCI machine learning repository, and 66 of them are randomly selected as testing (new) problems while the rest of 70 are deployed as historical problems. These problems cover almost all the attribute types, such as discrete, continuous, binary, and categorical. The range of the problem sizes is from 78 to 30K. For ViT and ResNet, we collected 105 image classification problems from the Kaggle website, and 52 of them are randomly selected as testing (new) problems while the rest of the 53 are deployed as historical problems. The range of the problem sizes is from 194 to 64K. Detailed information about the selected UCI and Kaggle problems can be found in Appendix I and Appendix II.

Evaluation metrics: We adopt four metrics for evaluating the recommendation performance, namely, average classification accuracy (ACA), average recommendation accuracy (ARA), hit rate (HR), and mean reciprocal rank (MRR). The first three are widely used in meta-learning, see [15] and reference therein. The range of ACA, ARA, and HR is $[0, 1]$, and a larger value is preferred. MRR compares the ranks of the real optimal configuration and the recommended configuration and it also has a range of [0,1] and a larger value is better. Their definitions can also be found in Appendix III.

4.5.2 Effectiveness of LRTC on SVM

In this experiment, three kernels, i.e., linear, polynomial, and RBF kernel, are applied to LRTC, we denote them separately as LRTC-Lin, LRTC-Poly, and LRTC-RBF. We test 9 different keeping ratios on the testing problems, namely, $\epsilon_{test} = 0.25\%, 0.50\%, 1\%, 2\%, 3\%, 4\%, 5\%, 10\%,$ and 20%, to show the performance variance of LRTC when the number of the known entries is increasing. When $\epsilon_{test} = 0.25\%$, only 1 entry of

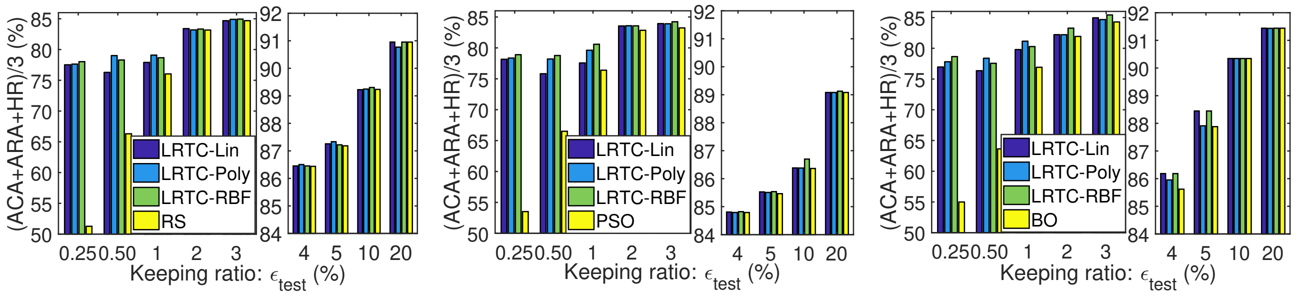


Figure 4.4: The comparisons between RS, PSO, BO, and LRTC, where the reported performance is the average of ACA, ARA, and HR.

Table 4.2: Wilcoxon signed-rank test results on CA between LRTC and MFCF. The p-values that are larger than 0.05 are underlined.

Alternative hypothesis	Keeping ratio ϵ_{test}								
	0.25%	0.50%	1%	2%	3%	4%	5%	10%	20%
LRTC-Lin > MFCF-Lin	$\underline{0.830}$	0.020	0.040	0.020	0.030	0.040	0.010	$\underline{0.500}$	$\underline{1.000}$
LRTC-Poly> MFCF-Poly	$\underline{0.910}$	$\underline{0.580}$	0.000	0.030	0.040	0.010	0.000	0.000	0.000
LRTC-RBF> MFCF-RBF	0.040	$\underline{0.550}$	0.000	0.000	0.000	0.000	0.000	0.000	0.000

each testing problem is known, while there are 80 known entries when $\epsilon_{test} = 20\%$.

LRTC versus MFCF

Here, we compare LRTC with MFCF [48] which is a special case of LRTC. The results under the four metrics are summarized in Table 4.1. The testing tensor is initialized by random search, so the experiment is repeated ten times for each ϵ_{test}, and only the average is reported to reduce the bias caused by the randomness of testing tensor initialization. Notice that the training tensor is dense, i.e., the budget is large enough for offline evaluations. For this comparison, we only consider the first-order recommendation.

From Table 4.1, one can observe that: 1) The size of ϵ_{test} has a dramatic impact on the recommendation performance of LRTC as expected. When $\epsilon_{test} = 0.25\%$, the ACA, ARA, HR, and MRR are about 74.09%, 81.17%, 77.12%, and 0.0929, respectively; however, these indexes mount to 78.61%, 97.87%, 97.42%, and 0.3714, respectively, when $\epsilon_{test} = 20\%$. MFCF has a similar tendency, but the performance difference between $\epsilon_{test} = 0.25\%, 0.50\%$, and 1% is minor; 2) Nonlinear kernels indeed improve the performance of LRTC. Especially when ϵ_{test} is small, kernel functions can capture the underlying relationships which guarantees better performance than the linear version, e.g., see $\epsilon_{test} = 0.25\%, 0.50\%$, and 1%. Nevertheless, when the known entries are increasing, the advantage of nonlinear kernels over the linear kernel is declining, e.g., see $\epsilon_{test} = 10\%$ and 20%. In contrast, nonlinear kernels do not show superiority for MFCF, and we observe that MFCF-Lin wins MFCF-Poly and MFCF-RBF on 8 over 9 cases; 3) When comparing to MFCF, LRTC wins in most of the cases no matter which kernel is applied, which proves that the performance space in a tensor structure preserves more useful information than the traditional vectorized setting. The gap between them is narrowing down with the increase of known entries. Especially when $\epsilon_{test} = 10\%$ and 20%, MFCF-Lin can perform equally as LRTC. In real applications, the entries may not be always fully available due to many unexpected reasons, in such a case, LRTC appears to be a better choice than MFCF.

To see if LRTC outperforms MFCF significantly, we apply the Wilcoxon signed-rank test [46] to each pair of LRTC and MFCF using the same kernel at a 5% significance level. The null hypothesis is that LRTC does not outperform MFCF significantly. The tested p-values of each pair are shown in Table 4.2, where p-values that are less than 0.05 mean that we reject the null hypothesis and accept the alternative hypothesis, i.e., the performance of LRTC is significantly better than that of MFCF. Here, we only report the test on CA, and similar results can be acquired on RA. As we can see, LRTC with any kernels is statistically better than MFCF when ϵ_{test} is between 1% and 5%. When $\epsilon_{test} = 10\%$ and 20%, we do not observe the obvious difference between LRTC-Lin and MFCF-Lin, yet the nonlinear variants of LRTC are still better than that of MFCF. Similarly, for the smaller ϵ_{test}, e.g., 0.25% and 0.50%, the superiority of our method is less obvious, we only observe 2 out of 6 cases where our method statistically outperforms MFCF.

LRTC versus search algorithms

Since LRTC needs partial configuration evaluations to start the completion and this can be done by search algorithms, e.g., RS, PSO, and BO, we prefer the recommendations of LRTC not to be inferior to the results found by the search algorithms. The comparisons between LRTC and RS (PSO, BO) are illustrated in Figure 4.4, where the reported performance for each ϵ_{test} is the mean of ACA, ARA, and HR. Because of the randomness of RS, we repeat the comparisons 10 times, and the average performance is adopted.

From Figure 4.4, one can observe that LRTC outperforms search algorithms dramatically when ϵ_{test} is small, e.g., $\epsilon_{test} = 0.25\%, 0.50\%$ and 1%. As the increase of ϵ_{test}, search algorithms can evaluate more configurations, so the difference between LRTC and RS (PSO, BO) is decreasing. Especially when $\epsilon_{test} = 20\%$, namely, 80 out of 399 configurations are tested, in which RS (PSO, BO) usually can find the optimal ones, while the LRTC does not appear to have the advantage.

An interesting phenomenon is that when LRTC is initialized by BO, its performance usually outperforms BO in a larger gap than that when it is initialized by RS, but LRTC does not show any advantage when ϵ_{test} is larger than 10%. A reasonable explanation is that BO searches the promising configurations for evaluations guided by the acquisition functions rather than randomly selecting one as RS does, so the initialization provided by BO can better describe the real performance space, or it is a good approximation of the real performance space, from which LRTC benefits. Therefore, BO should be a better choice for the initialization of the LRTC algorithm. Overall, LRTC can provide a better approach than search algorithms when the size of the search space of an algorithm is on a large scale since evaluating a large proportion of configurations via searching is often computationally unacceptable.

Performance of CMF

We present the effectiveness of CMF by comparing it to four meta-learning baselines, namely, KNN, kernel method (KKNN) [55], MLP [22], and warm-starting MFCF [18]. The meta-features employed are a combination of the eight state-of-the-art meta-features. The

Table 4.3: The comparisons of ACA (%), ARA (%), HR (%), and MRR between KNN, KKNN, MLP, MFCF, and CMF when three different orders of hyperparameters are recommended. The maximal value of each column is highlighted.

Methods	1st-order			2nd-order			3rd-order			
	ACA	ARA	HR	ACA	ARA	HR	ACA	ARA	HR	MRR
CMF	**76.39**	**89.28**	**87.88**	**76.06**	**88.30**	**86.36**	**76.47**	**88.61**	**86.36**	**0.1495**
KNN	74.22	81.59	78.79	74.38	82.27	78.79	73.90	81.26	77.27	0.0924
KKNN	74.24	81.26	77.27	73.97	81.30	75.76	74.53	82.11	77.27	0.0846
MLP	75.04	84.47	80.30	75.04	84.57	81.82	75.00	84.07	80.30	0.0459
MFCF	75.21	83.21	79.69	75.32	83.78	78.79	75.30	83.57	80.30	0.0308

Table 4.4: Wilcoxon signed-rank test results on CA between KNN, KKNN, MLP, MFCF, and CMF. The p-values that are larger than 0.05 are underlined.

Alternative hypothesis	The i^{th}-order recommendation		
	1	2	3
CMF > KNN	0.000	0.010	0.000
CMF > KKNN	0.000	0.010	0.000
CMF > MLP	0.000	0.000	0.000
CMF > MFCF	0.000	0.030	0.000

comparison results are given in Table 4.3 where the performance of the first three order recommendations is separately presented. The best parameter(s) for each method is/are determined by grid search.

From Table 4.3, one can observe that CMF outperforms the other four baselines with a large gap throughout the three different order recommendations, which shows the superiority of our method. In terms of ACA, ARA, and HR, MLP occupies the second place and MFCF takes the third place while KKNN and KNN rank fourth and fifth, respectively. For MRR, CMF ranks first but KNN and KKNN have a higher score than MLP and MFCF. When the first preference order configuration is recommended, the ACA, ARA, and HR of CMF are $76.39\%, 89.28\%$, and 87.88%, respectively, which are between the performance of LRTC when $\epsilon_{test} = 2\%$ and $\epsilon_{test} = 3\%$. For the second and third preference order recommendations, the performance of CMF is slightly decreasing, which implies that the first preference order recommendation is better than the second and third recommendations. However, for KNN, KKNN, MLP, and MFCF, the highest performance is obtained on the second or third recommendation.

Next, to check if CMF overpasses other baselines significantly, we report the Wilcoxon signed-rank test results in Table 4.4. Since all p-values are less than 0.05, we reach the conclusion that CMF is significantly better than KNN, KKNN, MLP, and MFCF over the first three ordered recommendations.

Performance of combined LRTC and CMF

This experiment aims to show how the performance of the combined algorithm LRTC-CMF is boosted when the ϵ_{test} is small. To show the overall performance difference between

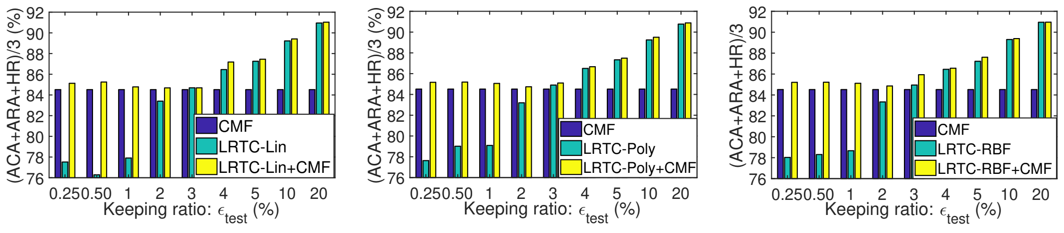


Figure 4.5: The comparisons between CMF, LRTC, and their combinations on the average of ACA, ARA, and HR.

LRTC, CMF, and LRTC-CMF, we only report the average of ACA, ARA, and HR for each ϵ_{test}. The details are depicted in Figure 4.5. Notice that here we only consider the first-order recommendation.

From Figure 4.5, one can observe that when ϵ_{test} is smaller than 3% where CMF outperforms LRTC, the combined algorithm can fairly improve the recommendation performance. However, when $\epsilon_{test} > 3\%$ where LRTC outperforms CMF, the impact of CMF on the combined algorithm is limited, although there are some improvements. A special case is that LRTC-CMF with linear kernel does not show any improvement when $\epsilon_{test} = 3\%$, but it is significantly better than LRTC-Lin when $\epsilon_{test} = 4\%$.

Overall, by integrating both partial real evaluations and meta-features, LRTC-CMF can benefit from each other when the known evaluations are insufficient. However, this combination is less effective as expected when a large proportion of configurations has been already evaluated.

Influence of tensor sparsity

This experiment aims at investigating how the recommendation capacity of LRTC would be affected when the different degrees of sparsity are applied to the training tensor. We set the training tensor keeping ratio $\epsilon_{train} = 10\%, 20\%, \cdots, 100\%$, which randomly select

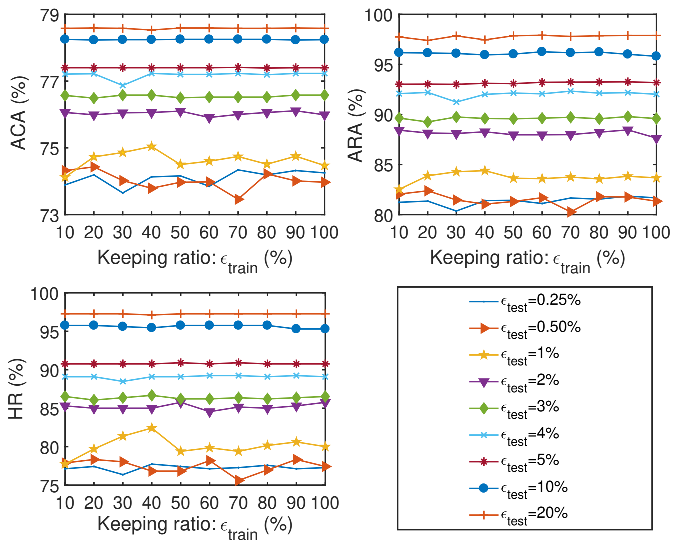


Figure 4.6: The performance fluctuation of LRTC-RBF on SVM when the keeping ratio on the training tensor is changing from 10% to 100%.

$90\%, 80\%, \cdots, 0\%$ of the points in the search space that will not be evaluated in the training stage. We repeat each ϵ_{train} ten times and the average performance is reported (Figure 4.6). Notice that we only present the results of LRTC-RBF and the following conclusions are also true for LRTC-Lin (Poly).

From Figure 4.6, one can observe that ϵ_{train} has some influences on the recommendation performance, according to the four evaluation metrics, when ϵ_{test} is small, e.g., $0.25\%, 0.50\%$, and 1%, while this impact is insignificant when a bigger ϵ_{test} is applied. Although there are fluctuations when $\epsilon_{test} = 0.50\%, 1\%$, most of the ϵ_{train} that are less than 100% can yield better performance. This is also true for $\epsilon_{test} = 0.25\%$, where a smaller ϵ_{train} does not affect too much the effectiveness of LRTC-RBF. In fact, an appropriate degree of sparsity can effectively prevent models from over-fitting and thus the sparse training tensors tend to produce more reliable outcomes than the dense version. Therefore, the sparsity of the training tensors not only saves time on the offline stage but also maintains the desirable recommendation capacity.

4.5.3 Effectiveness of LRTC on ViT

Due to the extensive burden of training, we do not apply 10-fold cross-validation to evaluate configurations on ViT and ResNet. We draw a proportion of instances from each class to constitute a validation set while the remaining instances are used for training and the performance is obtained on the validation set. In this section, we only report the performance of LRTC when ϵ_{test} is 0.25%, 0.50%, and 1% because we already knew that LRTC does not perform significantly better when ϵ_{test} is large in the previous experiment. Since the meta-features [23] do not apply to image datasets, we cannot present the results of CMF and LRTC-CMF.

LRTC versus MFCF

The settings for this comparison are similar to the case in SVM (4.5.2), i.e., the testing tensor is initialized by random search, and the average of ten runs is reported. Also, the training tensor is dense. We display the performance of the first-order recommendation in Table 4.5, from which one can observe that our approaches obtain desirable outcomes. Specifically, LRTC-Lin has the ACA, ARA, and HR of 87.21%, 94.90%, and 96.54% when $\epsilon_{test} = 0.25\%$, and they mount to 87.90%, 95.63%, and 97.88% when $\epsilon_{test} = 1\%$, and it produces the worst recommendation when $\epsilon = 0.50\%$ with an ACA, ARA, and HR of 86.08%, 93.34%, and 96.15%. On the other hand, the nonlinear variants achieve significant improvements for each ϵ_{test}. LRTC-RBF has the optimal performance when $\epsilon_{test} = 0.25\%, 1\%$ while LRTC-Poly shows its advantages when $\epsilon_{test} = 0.50\%$.

As a counterpart, MFCF-based methods show relatively lower performance in many

Table 4.5: The comparisons of ACA (%), ARA (%), HR (%), and MRR between LRTC and MFCF under various keeping ratios when three types of kernels are employed. The maximal value of each column is highlighted. The ACA of the real optimal configurations on the testing problems is 91.09%.

Methods	$\epsilon_{test} = 0.25\%$				$\epsilon_{test} = 0.50\%$				$\epsilon_{test} = 1\%$			
	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR
LRTC-Lin	87.21	94.90	96.54	**0.1736**	86.08	93.34	96.15	**0.1615**	87.90	95.63	97.88	0.1262
LRTC-Poly	87.41	95.03	97.50	0.1466	**87.87**	**95.50**	**99.23**	0.1323	87.79	95.52	98.46	0.1330
LRTC-RBF	**87.62**	95.20	**97.69**	0.1443	87.58	95.14	97.88	0.1370	**88.45**	**96.07**	**99.04**	0.1553
MFCF-Lin	87.58	**95.41**	96.15	0.1258	87.62	95.46	96.15	0.1283	87.76	95.63	95.96	0.1351
MFCF-Poly	87.08	94.37	96.15	0.1668	87.17	94.43	96.92	0.1284	87.08	94.73	96.15	**0.1669**
MFCF-RBF	87.11	94.79	95.38	0.1465	87.30	94.88	95.77	0.1429	87.42	95.01	96.73	0.1485

Table 4.6: Wilcoxon signed-rank test results on CA between LRTC and MFCF. The p-values that are larger than 0.05 are underlined.

Alternative hypothesis	Keeping ratio ϵ_{test}		
	0.25%	0.50%	1%
LRTC-Lin > MFCF-Lin	<u>0.580</u>	<u>0.318</u>	0.037
LRTC-Poly> MFCF-Poly	0.006	0.004	0.022
LRTC-RBF> MFCF-RBF	0.019	0.030	0.048

cases except for $\epsilon_{test} = 0.25\%, 0.50\%$ where MFCF-Lin has several higher scores in terms of ACA and ARA. However, because of the lower HR, it indicates that MFCF-Lin performs well only on some problems, but not for most of them. Our methods tend to recommend the applicable configurations for more problems. We also can see that the kernelized MFCF does not show any superiority, this may result from the insufficient nonlinear structure-preserving of the vectorized performance space. The statistical test in Table 4.6 reveals the significant difference between LRTC and MFCF where we only fail to accept the alternative hypothesis "LRTC-Lin > MFCF-Lin" when $\epsilon_{test} = 0.25\%, 0.50\%$, which suggests the superiority of our approaches.

At last, we notice that LRTC-Lin and MFCF-Poly have better MRR in some cases, e.g., $\epsilon_{test} = 0.25\%, 0.50\%$ for LRTC-Lin and $\epsilon_{test} = 0.25\%, 1\%$ for MFCF-Poly, but the other metrics appear to be average compared to other baselines, this is because they have very good recommendations just on several problems, but the performance on most of the problems is relatively mediocre.

LRTC versus search algorithms

Here we report the performance comparisons between RS, PSO, BO, and LRTC in Figure 4.7. Notice that the comparison between RS and LRTC is repeated ten times due to the uncertainty of RS and the mean performance of them is reported. We can observe that the tendencies of LRTC with three search algorithms are similar; when only one configuration ($\epsilon_{test} = 0.25\%$) is evaluated, search-based approaches do not perform well but our LRTC-based method dramatically improve the recommendation performance. As the allowed number of evaluated configurations is increasing, the performance gap between search

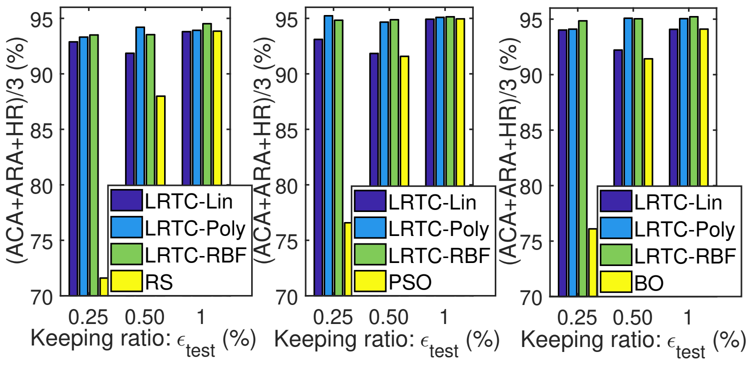


Figure 4.7: The comparisons between RS, PSO, BO, and LRTC, where the reported performance is the average of ACA, ARA, and HR.

methods and LRTC is narrowing but we still can see that, especially for kernel LRTC, our approaches obtain the desired performance. When $\epsilon_{test} = 1\%$, LRTC-Lin has similar performance as RS, PSO, and BO, but LRTC-Poly and LRTC-RBF still have a chance to win even though the superiority is minor in some occasions.

Influence of tensor sparsity

Similarly, here we discuss the performance fluctuation of LRTC-RBF on ViT when different degrees of sparsity are applied to the training tensor. To do so, we set $\epsilon_{train} = 10\%, 20\%, \cdots, 100\%$ as we did for SVM, and the results are displayed in Figure 4.8, from which one can observe that: 1) ϵ_{train} has a bigger influence on the performance of LRTC for $\epsilon_{test} = 0.25\%, 0.50\%$ where the main fluctuations are showed on HR and has a smaller impact for $\epsilon_{test} = 1\%$; 2) for $\epsilon_{test} = 0.25\%, 0.50\%$, we still have cases in which LRTC under a lower ϵ_{train} outperforms the counterparts when $\epsilon_{train} = 100\%$, e.g., $\epsilon_{train} = 70\%$ for $\epsilon_{test} = 0.25\%$ and $\epsilon_{train} = 80\%$ for $\epsilon_{test} = 0.50\%$. The effectiveness of LRTC under $\epsilon_{train} = 100\%$ does not show tremendous superiority over some other smaller ϵ_{train}; 3) the changing of ϵ_{train} does not degrade the performance of LRTC for $\epsilon_{test} = 1\%$ although there are several minor

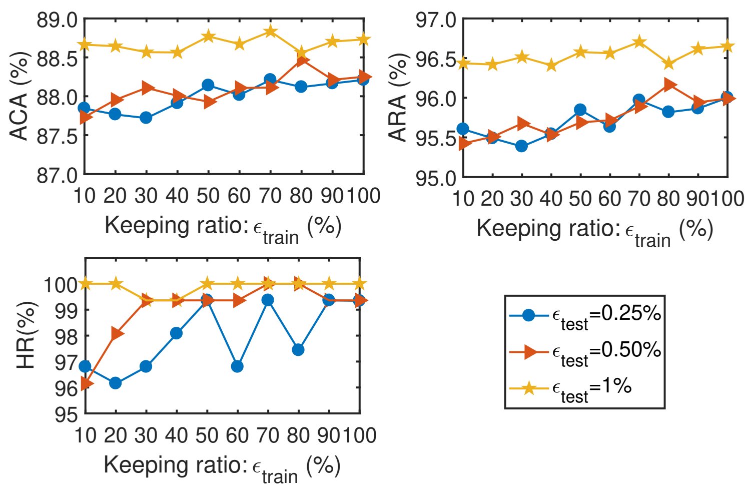


Figure 4.8: The performance fluctuation of LRTC-RBF on ViT when the keeping ratio on the training tensor is changing from 10% to 100%.

fluctuations. Overall, LRTC can still take advantage of the sparsity of tensors to attain efficiency improvement without losing its performance significantly on ViT.

4.5.4 Effectiveness of LRTC on ResNet

LRTC versus MFCF

The comparisons between LRTC and MFCF are summarized in Table 4.7 and the statistical test results on CA are listed in Table 4.8. We can observe that

- LRTC-based approaches attain the lowest HR of 98.08% and the highest HR of 100% in five and four cases out of nine in total, which indicates that we can recommend the applicable configurations to most of the testing problems. As a comparison, MFCF-based baselines have the lowest HR of 90.77% and attain the highest HR of 99.23% once.

- Our methods have the lowest (highest) ACA of 83.19% (85.16%) when $\epsilon_{test} = 0.25\%$, and have the lowest (highest) ACA of 84.82% (85.21%) when $\epsilon_{test} = 1\%$; however, they

Table 4.7: The comparisons of ACA (%), ARA (%), HR (%), and MRR between LRTC and MFCF under various keeping ratios ϵ_{test} when three types of kernel functions are employed. The maximal value of each column is highlighted. The ACA of the real optimal configurations on the testing problems is 87.57%.

Methods	$\epsilon_{test} = 0.25\%$				$\epsilon_{test} = 0.50\%$				$\epsilon_{test} = 1\%$			
	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR	ACA	ARA	HR	MRR
LRTC-Lin	83.19	93.66	98.08	0.0962	84.31	95.06	98.08	0.1035	**85.21**	**96.40**	98.08	0.1536
LRTC-Poly	85.00	**96.27**	**100.00**	0.1254	**84.99**	**96.19**	**100.00**	**0.1612**	**85.21**	**96.40**	98.08	**0.1801**
LRTC-RBF	**85.16**	96.24	**100.00**	0.1488	84.23	95.13	**100.00**	0.1263	84.82	95.70	98.08	0.1252
MFCF-Lin	80.80	91.04	90.77	0.1064	84.54	95.59	97.50	0.1562	84.94	96.13	**99.23**	0.1492
MFCF-Poly	84.54	95.43	98.08	0.1243	83.79	94.36	97.12	0.1057	85.01	96.13	98.08	0.1715
MFCF-RBF	84.91	95.83	97.69	**0.1492**	84.50	95.26	96.92	0.1461	84.61	95.53	97.69	0.1349

Table 4.8: Wilcoxon signed-rank test results on CA between LRTC and MFCF. The p-values that are larger than 0.05 are underlined.

Alternative hypothesis	Keeping ratio ϵ_{test}		
	0.25%	0.50%	1%
LRTC-Lin > MFCF-Lin	0.002	$\underline{0.757}$	0.043
LRTC-Poly> MFCF-Poly	0.049	0.000	$\underline{0.078}$
LRTC-RBF> MFCF-RBF	$\underline{0.155}$	$\underline{0.058}$	0.010

show relatively worse performance when $\epsilon_{test} = 0.50\%$ with the lowest and highest ACA of 84.23% and 84.99%. On the other hand, kernelized LRTC shows superior performance again. Especially, LRTC-Poly performs the best when $\epsilon_{test} = 0.50\%, 1\%$ while LRTC-RBF is the optimal for $\epsilon_{test} = 0.25\%$.

- For MFCF, linear kernel obtains the optimal performance for $\epsilon_{test} = 0.50\%$, and RBF and polynomial kernel win the first place for $\epsilon_{test} = 0.25\%$ and $\epsilon_{test} = 1\%$, separately. On average, LRTC outweighs MFCF even though MFCF-RBF has the highest MRR for $\epsilon_{test} = 0.25\%$ and MFCF-Lin has the highest ARA for $\epsilon_{test} = 1\%$, and MFCF-Lin outweighs LRTC-Lin when $\epsilon_{test} = 0.50\%$ in terms of ACA and ARA. The Wilcoxon signed-rank tests in Table 4.2 show that our proposed approaches are statistically better than MFCF in five over nine cases, and two other test scores are close to 0.05.

LRTC versus search methods

Here we report the performance comparisons between search algorithms and LRTC in Figure 4.9. The comparison between RS and LRTC is also repeated ten times and the mean performance of them is considered. We can draw similar conclusions as we had for ViT: the tendencies of LRTC associated with three search algorithms are similar; when only one configuration ($\epsilon_{test} = 0.25\%$) is evaluated, search-based approaches do not perform well but LRTC significantly improves the recommendation performance. As the allowed number of evaluations is increasing, the performance gap between RS, PSO, BO, and LRTC is narrowing down but we still can see that, especially for kernel LRTC, our approaches obtain the desired performance. When $\epsilon_{test} = 1\%$, LRTC-Lin has a similar performance as PSO and BO, but LRTC-Poly and LRTC-RBF still have chances to outperform with a significant difference.

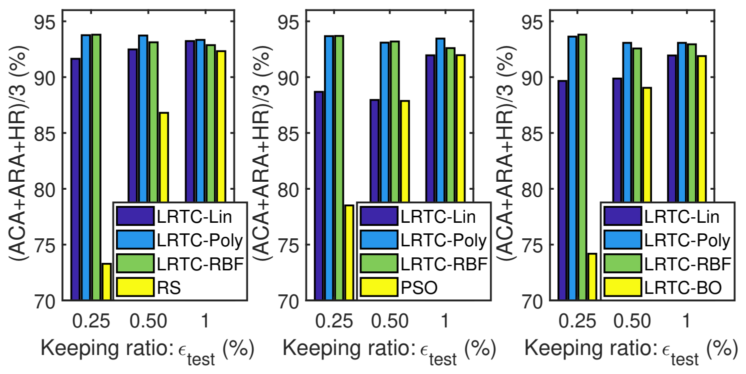


Figure 4.9: The comparisons between RS, PSO, BO, and LRTC, where the reported performance is the average of ACA, ARA, and HR.

Influence of tensor sparsity

We discuss the performance fluctuation of LRTC-RBF by setting the keeping ratio of the training tensor as $\epsilon_{train} = 10\%, 20\%, \cdots, 100\%$. The experimental results are illustrated in Figure 4.10, from which one can observe that: 1) the ϵ_{train} shows a bigger influence on the performance of LRTC-RBF for $\epsilon_{test} = 0.25\%, 0.50\%$ and has a smaller impact for $\epsilon_{test} = 1\%$; 2) for $\epsilon_{test} = 0.25\%$, a higher ϵ_{train} tends to produce a better performance and the best performance is obtained when $\epsilon_{train} = 80\%$; nevertheless, there is a contrary tendency for $\epsilon_{test} = 0.50\%$ and the best one is achieved when $\epsilon_{train} = 20\%$ which is preferred. The variations of effectiveness when $\epsilon_{test} = 1\%$ are relatively stabler in terms of ACA and ARA, and the negative impact of the training tensor sparsity is minor and the best performance is gained when $\epsilon_{train} = 40\%$; 3) notice that the overall best performance is attained when $\epsilon_{test} = 0.25\%$ and the worst one is obtained when $\epsilon_{test} = 0.50\%$, which are coincident with the results given in Table 4.1.

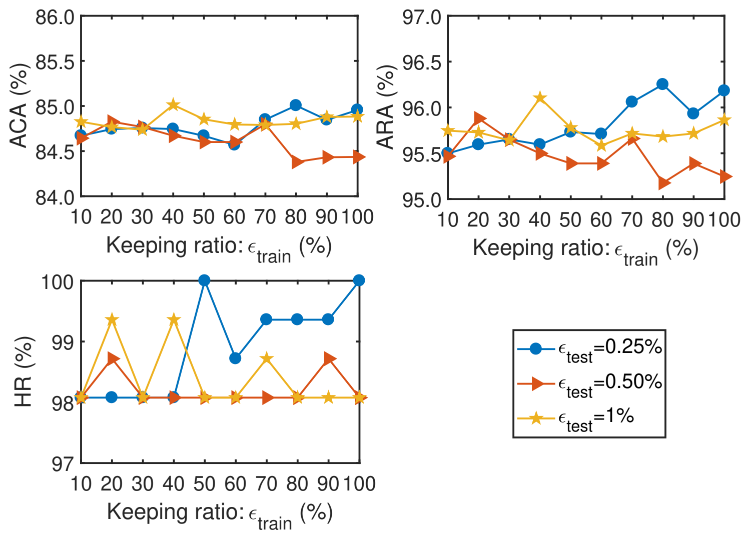


Figure 4.10: The performance fluctuation of LRTC-RBF when the keeping ratio ϵ_{train} on the training tensor is changing from 10% to 100%.

4.6 CONCLUDING REMARKS

This chapter studies the automatic hyperparameter recommendation problem in which the low-rank tensor completion (LRTC) technique is applied to estimate the performance of the unevaluated configurations on a new classification problem. We develop both linear and nonlinear completion algorithms based on the sum of the nuclear norm (SNN) model where we unfold a tensor along each model to process the data while maintaining the spatial correlated structure. Empirical results show the superiority of our proposed approach by comparing it to model-based collaborative filtering and search algorithms, such as random search, PSO, and Bayesian optimization. Besides LRTC, a coupled matrix factorization algorithm is proposed to bridge the relationship between performance space and the meta-features, which is a new method in meta-learning. Its effectiveness is verified by four well-known baselines in the literature.

For future work, it may be interesting to apply LRTC to the algorithm selection and hyperparameter optimization (CASH) problem, which aims to recommend the best algorithm(s) and their hyperparameter(s) simultaneously. Our current work can only deal with one algorithm because two or more search spaces with different dimensions may not be suitably merged into one single tensor, which could be a bottleneck for dealing with the CASH task. Further, how to handle the conditional hyperparameters or the hierarchical search space under the tensor framework would be another interesting topic.

CHAPTER 5

LATENT FEATURE LEARNING

5.1 INTRODUCTION

Selecting suitable classifiers along with adequate hyperparameter configurations is of great importance in real applications. Algorithms are desired with the ability to automatically search for both the best classifiers and configurations simultaneously. This is a fundamental research area in automated machine learning (AutoML) [109] that necessitates the AI applications, known as *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) [33]. Currently, there are very limited studies addressing the CASH problem despite its practical importance. The main challenge is how to perform the optimization process effectively and efficiently, with the high-dimensional configuration space appearing to be a hierarchical structure consisting of the integration of continuous (e.g., regularization parameter '`C`' and Gaussian kernel width '`gamma`' in SVM), or discrete (e.g., number of neighbors in KNN and the number of base learners in AdaBoost), or categorical (e.g., splitting criteria '`gini`' or '`entropy`' in the decision tree) variables, presenting the considerable level of difficulties.

One representative algorithm is *Sequential Model-based Optimization* (SMBO) [110], a framework that stems from Bayesian optimization. It not only can deal with many types of hyperparameters but also the hierarchical structure derived from the configuration space. A similar algorithm proposed by Leite et al. [111] is *active test*, which selects the most promising configurations based on previous comparisons between configurations on similar problems. However, since these two algorithms evaluate every selected configuration, such a procedure can be time-consuming if the problem size is large. The latest strategy for handling the CASH problem is the meta-learning-based recommendation approach, where matrix factorization-based collaborative filtering (MFCF) plays an important role [17, 18, 48, 47]. Following the same rationale in the recommender system, MFCF assumes two problems that have certain partially similar evaluations may maintain similar evaluations over the entire configuration space. Hence, one can predict the performance of the configuration space on a new problem by randomly evaluating some configurations. Nevertheless, within this framework, it requires performing matrix factorization for every recommendation task, which could be inefficient if the dimension of the matrix is high while this is typically true in CASH scenarios.

In this chapter, we propose a new configuration recommendation method by integrating meta-learning with the denoising autoencoder (DAE) approach [112, 113]. In fact, even if the search space of CASH is high-dimensional, the performance of configurations is usually correlated to each other to a certain degree and its main structure lies in a much lower-dimensional manifold that describes the performance distribution of the search space. By suitably manipulating the inputs, DAE is known as an effective tool to extract the robust latent features from data in which the attributes are highly correlated [112]. Our motivation lies in that, by learning the robust latent subspace of the configuration space of CASH via DAE, we are able to capture the intrinsic distribution of performance space and thus recommend suitable configurations. In practice, the effectiveness of latent features determines the performance of recommendation, which is similar to MFCF, but there are two significantly distinct differences. First, DAE employs a self-supervised learning strategy in its process; second, DAE is a neural network-based model. Although neural networks via learning have shown their powerful capacity in various applications, addressing the CASH problem under the neural network framework is still quite new in the meta-learning area. As far as we know, there is only one literature [22] in which a limited approach is considered by taking the multi-layer perceptron as a regressor for classifier selection purposes where the empirical investigations are quite insufficient.

In this chapter, we first define the candidate classifiers and their hyperparameters to form a configuration space of CASH and then evaluate it on a group of historical problems to obtain the training data, i.e., classification performance under a suitable metric. Based on a prior setting with a uniform probability distribution, some data entries are masked randomly by setting them to zero, and the rest of the unmasked data entries remain unchanged. Then both the encoder and decoder are trained by using the masked data as inputs and the unmasked data as labels to extract the latent features. Once the encoder and decoder have been developed, for a completely new problem at hand, we only need to evaluate some configurations by constructing a "masked" performance vector where the entries are zeros if the corresponding configurations are not selected and evaluated and feed it to the trained DAE to obtain the predictive performance of the configuration space. The configuration that has the highest predictive performance is thus recommended for the new problem. For real applications, to avoid any configuration evaluations for new problems, we develop a kernel multivariate multiple regression (MMR) model to directly connect the meta-features to the learned latent features of historical datasets so that the performance on the new problems can be estimated through the MMR and the *decoder* network, improving the efficiency of the recommendation process directly.

To show the recommendation capacity of our proposed approach, we establish an automatic classification configuration recommendation system that includes 82 historical classification problems that are collected from the UCI machine learning repository and 11 commonly-used classifiers with a total of 4983 configurations. Experimental outcomes on 45 testing problems demonstrate that our model outperforms the existing meta-learning baselines and search algorithms under various evaluation metrics, validating the ability to select well-suitable configurations for new problems. The main contributions of this chapter are summarized as follows:

1. We present a new model by integrating the encoder and decoder in DAE with meta-learning for the classifier and hyperparameter recommendation, which is a neural network-based model by focusing on latent feature learning to capture the essential characteristics of a data relationship and is able to provide the required capacity for hyperparameter recommendations under the machine learning framework.

2. A kernel multivariate multiple regression model between the acquired latent variables and the meta-features of datasets is developed to promote the recommendation efficiency, in which a new configuration evaluation for different problems is not required and thus better efficiency is achieved.

3. An automatic classification configuration recommendation system, including 82 historical problems and 11 common classifiers with a total of 4983 configurations, is established to show the effectiveness and efficiency of our proposed approach.

5.2 RELATED WORK

Following the convention, CASH is described as follows. Let $\mathscr{A} = \{\mathscr{A}^{(1)}, \mathscr{A}^{(2)}, \cdots, \mathscr{A}^{(a)}\}$ be the given candidate classifier pool, where $\mathscr{A}^{(j)}$, $j = 1, \cdots, a$, represent the classification algorithms, and denote $\Omega_{\mathscr{A}} = \mathbf{\Omega}^{(1)} \cup \mathbf{\Omega}^{(2)} \cup \cdots \cup \mathbf{\Omega}^{(a)}$ as the corresponding configuration space, where $\mathbf{\Omega}^{(j)}$ is the hyperparameter search space for classifier $\mathscr{A}^{(j)}$. For a given classification problem $\mathscr{D}$, the goal of CASH is to find a classifier $\mathscr{A}^{\star}$ instantiated with configuration $\boldsymbol{\omega}^{\star}$ such that

$$\mathscr{A}^{\star}_{\boldsymbol{\omega}^{\star}} = \operatorname*{arg\,opt}_{\mathscr{A}^{(j)} \in \mathscr{A}, \boldsymbol{\omega}_i \in \mathbf{\Omega}^{(j)}} V(\mathcal{P}, \mathscr{A}^{(j)}_{\boldsymbol{\omega}_i}, \mathscr{D}),$$

where $V(\cdot,\cdot,\cdot)$ is a validation strategy, e.g., k-fold cross-validation, and $\mathcal{P}$ is a performance evaluation metric such as the classification accuracy (error) rate and area under the curve.

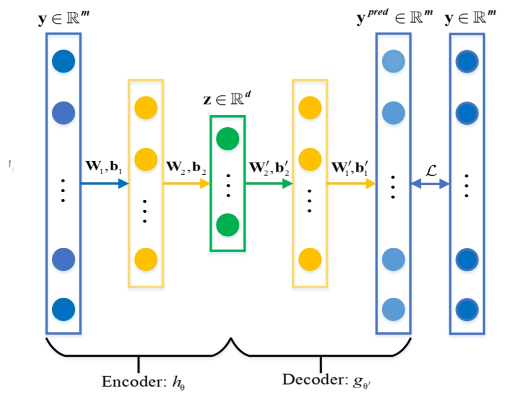
Currently, there are mainly three strategies used for addressing the CASH problem, namely, sequential model-based optimization, active testing, and meta-learning-based recommendation. The first two will be introduced next.

5.2.1 Sequential model-based optimization

Sequential model-based optimization (SMBO) [110] is a hyperparameter optimization strategy that derives from Bayesian optimization, and it comprises a surrogate model that fits the performance of the evaluated configurations and an acquisition function, such as the expected improvement, that chooses the most promising ones from the remaining search space for next evaluation. It generally outperforms the random search and the grid search with fewer iterations. The difference between various SMBO algorithms mainly appears in the surrogate models deployed. Two popular studies in hyperparameter optimization via machine learning are the random forest regressor and the tree-structured Parzen estimator because they can effectively handle the hierarchical structure and almost all types of hyperparameters can be found in the search space (i.e., discrete, continuous, and categorical). The representative framework adopting SMBO for solving the CASH problem is Auto-WEKA [114], where the WEKA classifiers and feature selectors are used as the candidate algorithms. By treating each classifier as a conditional hyperparameter, Auto-WEKA can be viewed as a single learning algorithm. Nevertheless, as a search-based method, the SMBO starts from scratch for new problems and is required to evaluate every promising configuration it finds, which could be costly when the sizes of problems are large.

5.2.2 Active testing

Active testing (AT) [111] employs a similar framework as search-based algorithms do and exploits the similarity between the testing problems and the historical problems during the testing process. The initialization of AT is done by ordering the average ranks of configurations over historical problems rather than random selection. The initialized configuration is denoted as a_{best}, the best one for the new problem, and then it is compared by the newly selected one determined by maximizing the *estimated performance gain* defined by the product of the relative landmarking and the similarity (both vary in the next round due to the participation of new evaluation) among the untouched search space. Every newly selected configuration is evaluated using 10-fold cross-validation and it becomes a_{best} once it outperforms the previous best one. When the allocated resource is exhausted, e.g., CPU time or the search-stopping criteria, a_{best} is recommended. Leite et al. [111] study the performance of AT on a small-scale search space of CASH (6 classifiers with a total of 292 configurations and 76 classification problems). Abdulrahman et al. [115] further incorporate running time into AT to enable slow but promising configurations that can be evaluated with sufficient time; however, its performance on the CASH problem has not been addressed. Since AT relies on



$\mathbf{y} \in \mathbb{R}^m$
$\mathbf{W}_1, \mathbf{b}_1$
$\mathbf{W}_2, \mathbf{b}_2$
$\mathbf{z} \in \mathbb{R}^d$
$\mathbf{W}'_2, \mathbf{b}'_2$
$\mathbf{W}'_1, \mathbf{b}'_1$
$\mathbf{y}^{pred} \in \mathbb{R}^m$
$\mathcal{L}$
$\mathbf{y} \in \mathbb{R}^m$
Encoder: h_θ
Decoder: $g_{\theta'}$

Figure 5.1: The illustration of a two-layer autoencoder, where $\theta = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\}$ and $\theta' = \{\mathbf{W}'_1, \mathbf{W}'_2, \mathbf{b}'_1, \mathbf{b}'_2\}$ are separately the collection of weight and bias parameters of encoder and decoder.

cross-validation during every iteration when terminating the test presents an important but unsolved issue in terms of how to balance between the required time usage and the desired performance.

5.3 DENOISING AUTOENCODERS

5.3.1 Autoencoder

Autoencoder (AE) is a special type of neural network that has a symmetric structure (see Figure 5.1). It comprises of two components: *encoder* h_θ and *decoder* $g_{\theta'}$. Encoder h_θ compresses or expands the inputs to a much lower or higher dimensional space, called codes, while the function of decoder $g_{\theta'}$ is to restore the original inputs from the encoded data. Suppose $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n]$ is a set of training data where $\mathbf{y}_i \in \mathbb{R}^m$ for any i, and denote $\mathcal{L}(\cdot, \cdot)$ be a loss metric, e.g., Euclidean distance, the loss function of a l-layer AE can be mathematically formulated as

$$F(\theta, \theta') = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\mathbf{y}_i, g_{\theta'}(h_\theta(\mathbf{y}_i))\right) + \mathcal{R}(\theta) + \mathcal{R}(\theta'), \tag{5.1}$$

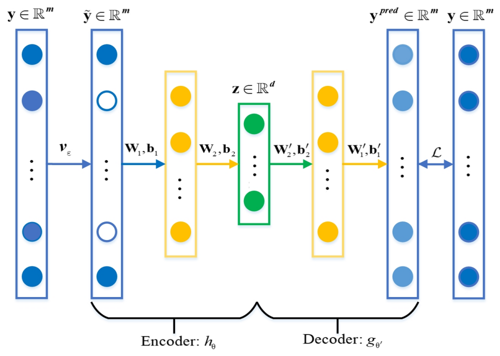

$\mathbf{y} \in \mathbb{R}^m$
$\tilde{\mathbf{y}} \in \mathbb{R}^m$
$\boldsymbol{v}_\varepsilon$
$\mathbf{W}_1, \mathbf{b}_1$
$\mathbf{W}_2, \mathbf{b}_2$
$\mathbf{z} \in \mathbb{R}^d$
$\mathbf{W}_2', \mathbf{b}_2'$
$\mathbf{W}_1', \mathbf{b}_1'$
$\mathbf{y}^{pred} \in \mathbb{R}^m$
$\mathcal{L}$
$\mathbf{y} \in \mathbb{R}^m$
Encoder: h_θ
Decoder: $g_{\theta'}$

Figure 5.2: The illustration of a two-layer denoising autoencoder, where the empty circles of $\tilde{\mathbf{y}}$ stand for the zero entries.

where $\theta = \{\mathbf{W}_1, \cdots, \mathbf{W}_l, \mathbf{b}_1, \cdots, \mathbf{b}_l\}$ and $\theta' = \{\mathbf{W}_1', \cdots, \mathbf{W}_l', \mathbf{b}_1', \cdots, \mathbf{b}_l'\}$ are separately the collection of weights and biases involved in the encoder (decoder) that should be determined via *learning*. The last two regularization terms are used to prevent the model from over-fitting, and $\mathcal{R}(\theta) = \beta \sum_{j=1}^{l} (\|\mathbf{W}_j\|_F^2 + \|\mathbf{b}_j\|_2^2)$ and $\mathcal{R}(\theta') = \beta \sum_{j=1}^{l} (\|\mathbf{W}_j'\|_F^2 + \|\mathbf{b}_j'\|_2^2)$ where $\beta > 0$ is the regularization parameter. We denote $\mathbf{z} = h_\theta(\mathbf{y}) \in \mathbb{R}^d$ as the encoded data and call it as the latent feature of $\mathbf{y}$ thereafter. As shown in the literature, when $d < n$, a single hidden layered AE with linear activation function is equivalent to the principal component analysis [116]. The situation becomes challenging when the number of hidden layers is increasing and nonlinear activation functions are deployed.

5.3.2 Denoising autoencoder (DAE)

To extract more useful latent features from data, a simple AE setting is not sufficient due to its limited capacity, and additional regularization is required for feature distinction. We consider *denoising* framework in our paper. To apply the denoising principle, we simply use the corrupted data as inputs and enforce AE to recover the original clean data, as illustrated in Figure 5.2. Denoting the corrupted data items as $\tilde{\mathbf{y}}_j, j = 1, 2, \cdots, n$, then the loss function of DAE can be set as

$$F(\theta, \theta') = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\mathbf{y}_i, g_{\theta'}(h_\theta(\tilde{\mathbf{y}}_i))) + \mathcal{R}(\theta) + \mathcal{R}(\theta'). \tag{5.2}$$

The corruption we selected is *masking noise*, namely, a prescribed proportion of entries of inputs are forced to be zero (here we assume that there are no zero entries in the clean input data) and others remain unchanged. A plausible geometric explanation of denoising is given in [112] and the key to success is the dependencies that existed among the attributes of the high-dimensional distributions. Also, by imposing sparsity on inputs, we can effectively prevent the DAE from over-fitting.

5.3.3 Loss metric

Since the final recommendation approach (shown in Section 5) depends on the orders of the predictive performance rather than the exact estimated values, it is expected that the outputs of DAE can better preserve the ranks of the entries of an input. Therefore, the rank-based loss functions are more suitable for our purpose, while the classical mean square loss is nonsensitive to the ranks of patterns. We adopt binary cross-entropy as the loss function in this paper, which is defined as

$$\mathcal{L}(\mathbf{y}, \mathbf{y}^{pred}) = \sum_{i=1}^{m} y_i \log y_i^{pred} + (1 - y_i) \log (1 - y_i^{pred}). \tag{5.3}$$

It is shown in the literature that binary cross-entropy (5.3) is rank-sensitive [117] and minimizing (5.3) is equivalent to maximizing NDCG [118], a popular rank-based evaluation metric, and a larger NDCG means the orders of entries from two items appear to be more

Algorithm 6 `DAE`

```
Input: Training data: $\mathbf{Y}$, keeping ratio: $\epsilon$, number of iterations of pre-training: max_iter_pre, number of iterations of fine-tuning: max_iter_fine
Output: Trained DAE model: encoder $h_\theta$ and decoder $g_{\theta'}$
 1: Preset regularization parameters: $\beta$
 2: Initialize network structure $h, g$ and parameters $\theta, \theta'$
 3: #Pre-training
 4: for j=1:max_iter_pre do
 5:     $\tilde{\mathbf{y}}_i = v_\epsilon(\mathbf{y}_i), i = 1, 2, \cdots, n$
 6:     Updating $\theta, \theta'$: $\theta, \theta' = \arg\min_{\theta,\theta'} F(\theta, \theta') = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{y}_i, g_{\theta'}(h_\theta(\tilde{\mathbf{y}}_i))) + \mathcal{R}(\theta) + \mathcal{R}(\theta')$
 7: end for
 8: #Fine-tuning
 9: for j=1:max_iter_fine do
10:     Updating $\theta, \theta'$: $\theta, \theta' = \arg\min_{\theta,\theta'} F(\theta, \theta') = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathbf{y}_i, g_{\theta'}(h_\theta(\mathbf{y}_i))) + \mathcal{R}(\theta) + \mathcal{R}(\theta')$
11: end for
```

similar. Utilizing the rank-based loss metric for meta-learning is one of our important approaches in this chapter.

5.4 META-LEARNING VIA DAE

In this section, we detail the training of the DAE model as well as how to make a performance prediction via the well-trained DAE. Suppose that we have a collection of historical classification problems $D = \{\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_n\}$, and the corresponding classification performance is evaluated, denoted as $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n]$, where $\mathbf{y}_j \in \mathbb{R}^m, j = 1, 2, \cdots, n$ and m is the dimension of the configuration space of CASH problem. The training procedure of DAE consists of pre-training and fine-tuning.

Pre-training: Let $\epsilon \in (0, 1)$ be the keeping ratio of corruptions, i.e., $(1-\epsilon)m$ entries of a data vector $\mathbf{y}_i$ will be randomly selected and replaced by zeros, and let v be the corruption function, namely, $\tilde{\mathbf{y}}_i = v_\epsilon(\mathbf{y}_i), i = 1, 2, \cdots, n$. During each iteration, training data $\mathbf{Y}$ is corrupted as $\tilde{\mathbf{Y}} = [\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, \cdots, \tilde{\mathbf{y}}_n]$ and it is fed to DAE to update parameters θ and θ' using `Adam` algorithm. The updating ends after a maximum number of iterations (`max_iter_pre`) is reached.

Fine-tuning: In contrast, during the fine-tuning phase, the uncorrupted training data $\mathbf{Y}$ is fed to the pre-trained DAE model in every iteration. Similarly, the updating ends after a maximum number of iterations (`max_iter_fine`) is reached.

This procedure is displayed in `Algorithm` 6. After the training process is finished, for a new given problem $\mathscr{D}$, the performance of the entire CASH search space is predicted by using the following two strategies: cold-starting and warm-starting recommendation, respectively.

5.4.1 Cold-starting recommendation

This method adopts the denoising principle. We first randomly select some configurations, denoted as $\tilde{\Omega}_A$, from $\Omega_{\mathscr{A}}$ and evaluate them on $\mathscr{D}$ to generate an incomplete performance vector $\tilde{\mathbf{y}}$, where the entries are zeros if the corresponding configurations are not evaluated, then it is fed to the trained DAE to obtain the predictive performance over the entire configuration space, denoted as

$$\mathbf{y}^{pred} = g_{\theta'}(h_\theta(\tilde{\mathbf{y}})). \tag{5.4}$$

Since this prediction depends on the evaluation of configurations, we call this approach cold-starting DAE. The functionality of the cold-starting DAE is presented in `Algorithm` 7, where `ceil`$(\cdot)$ is the function of rounding up to an integer.

5.4.2 Warm-starting recommendation

To improve efficiency in applications, we propose a warm-starting recommendation approach based on the meta-features of problems and the trained DAE model. Let f be the

Algorithm 7 `Cold-starting DAE`

```
Input: new problem: $\mathscr{D}$, keeping ratio: $\epsilon$, trained encoder: $h_\theta$, trained decoder: $g_{\theta'}$, configuration space: $\Omega_{\mathscr{A}}$, classification performance metric: $\mathcal{P}$
Output: predictive performance for $\mathscr{D}$: $\mathbf{y}^{pred}$
1: Corrupted performance vector: $\tilde{\mathbf{y}} = \mathbf{0}$
2: for $i = 1$ :ceil($|\Omega_{\mathscr{A}}|\epsilon$) do
3:     Pick a configuration randomly $(\mathscr{A}, \boldsymbol{\omega})_j$ from $\Omega_{\mathscr{A}}$, $j$ is the index of this configuration in $\Omega_{\mathscr{A}}$
4:     $\tilde{y}_j = V(\mathcal{P}, \mathscr{A}_{\boldsymbol{\omega}}, \mathscr{D})$
5:     Evaluation with no replacement: $\Omega_{\mathscr{A}} = \Omega_{\mathscr{A}} \setminus (\mathscr{A}, \boldsymbol{\omega})_j$
6: end for
7: Performance prediction: $\mathbf{y}^{pred} = g_{\theta'}(h_\theta(\tilde{\mathbf{y}}))$
```

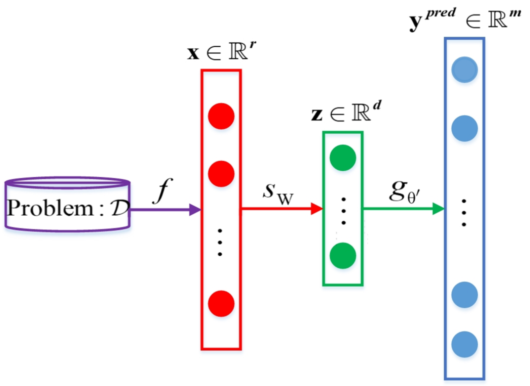

$\mathbf{y}^{pred} \in \mathbb{R}^m$
$\mathbf{x} \in \mathbb{R}^r$
$\mathbf{z} \in \mathbb{R}^d$
Problem : $\mathcal{D}$
f
s_W
$g_{\theta'}$

Figure 5.3: The illustration of the performance estimation of the warm-starting DAE combined with MMR $s_{\mathbf{W}}$ and the trained decoder $g_{\theta'}$, where f is the meta-feature extractor. $\mathbf{x}, \mathbf{z}$, and $\mathbf{y}^{pred}$ stand for meta-features, latent features, and the estimated performance, respectively.

meta-feature extractor, then the meta-features of the historical problems can be represented as

$$\mathbf{x}_i = f(\mathscr{D}_i) \in \mathbb{R}^r, i = 1, 2, \cdots, n.$$

Next, we extract the latent features from the training data, which is formulated as

$$\mathbf{z}_i = h_\theta(\mathbf{x}_i) \in \mathbb{R}^d, i = 1, 2, \cdots, n.$$

Notice that here we use clean inputs. Finally, the relationship between the meta-features and the latent features is modeled using a kernel multivariate multiple regression (MMR) where the meta-features are the predictors while the latent features are the responses. Denote this MMR model as $s_{\mathbf{W}}$, where $\mathbf{W}$ is a coefficient matrix, then the loss function is given by

$$F(\mathbf{W}) = \frac{1}{2}\sum_{i=1}^{n}\|\mathbf{z}_i - s_{\mathbf{W}}(\phi(\mathbf{x}_i))\|_2^2 + \frac{\beta'}{2}\|\mathbf{W}\|_F^2, \tag{5.5}$$

where $\beta' > 0$ is a regularization parameter, ϕ is a nonlinear mapping, and $s_{\mathbf{W}}(\phi(\mathbf{x}_i)) = \mathbf{W}^\top\phi(\mathbf{x}_i)$. The global optimal solution for (5.5) can be obtained by

$$\mathbf{W} = \phi(\mathbf{X})(\beta'\mathbf{I}_n + \mathbf{K}_{\mathbf{XX}})^{-1}\mathbf{Z}^\top,$$

where $(K_{XX})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top\phi(\mathbf{x}_j)$, $\phi(\mathbf{X}) = [\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_n)]$, and $\mathbf{Z} = [\mathbf{z}_1, \cdots, \mathbf{z}_n]$. Note that $k(\cdot,\cdot)$ represents various kernel functions, e.g., RBF or polynomial kernel. Then, it is combined with the decoder $g_{\theta'}$ of DAE to produce a new network architecture which is deployed for the performance estimation as described in Figure 5.3.

To make a prediction for a new problem $\mathscr{D}$, we first extract its meta-feature $\mathbf{x} = f(\mathscr{D}) \in \mathbb{R}^r$, then feed it to the combined network to obtain an estimation, namely,

$$\mathbf{y}^{pred} = g_{\theta'}(s_{\mathbf{W}}(\phi(\mathbf{x}))). \tag{5.6}$$

This warm-starting strategy along with prediction is shown in `Algorithm` 8.

Algorithm 8 `Warm-starting DAE`

```
Input: encoder: $h_\theta$, decoder: $g_{\theta'}$, training data: $\mathbf{Y}$, meta-feature extractor: $f$, historical problems: $\{\mathscr{D}_1, \cdots, \mathscr{D}_n\}$, new problem: $\mathscr{D}$, regularization parameter: $\beta'$
Output: predictive performance for $\mathscr{D}$: $\mathbf{y}^{pred}$
1: Meta-feature extraction: $\mathbf{x}_i = f(\mathscr{D}_i) \in \mathbb{R}^r, i = 1, 2, \cdots, n$
2: Latent feature extraction: $\mathbf{z}_i = h_\theta(\mathbf{y}_i) \in \mathbb{R}^d, i = 1, 2, \cdots, n$
3: $\mathbf{W} = \phi(\mathbf{X})(\beta' \mathbf{I}_n + \mathbf{K}_{\mathbf{XX}})^{-1}\mathbf{Z}^\top$
4: Prediction:
5: Meta-feature extraction: $\mathbf{x} = f(\mathscr{D}) \in \mathbb{R}^r$
6: Performance prediction: $\mathbf{y}^{pred} = g_{\theta'}(s_{\mathbf{W}}(\phi(\mathbf{x})))$
```

5.5 AUTOMATIC CONFIGURATION RECOMMENDATION

Based on our proposed DAE model and the general meta-learning architecture [42], we design an automatic classification configuration recommendation system and its workflow is illustrated in Figure 5.4. This system includes two phases: training and recommendation.

5.5.1 Training stage

The process of DAE model training and prediction using the given performance data and meta-features has been shown previously. Next, we will address how to obtain these two types of metadata.

Historical performance evaluation

Before we present the historical performance evaluation method, we first give the configuration space of the CASH problem being considered. To generate the candidate classifier pool, we adopt 11 commonly-used classifiers in the literature, they are SVM, decision tree (DT), extra tree (ET), random forest (RF), AdaBoost, Bagging, KNN, LDA, QDA, multi-layer perceptron (MLP), and logistic regression (LR), respectively. The important hyperparameters and their search ranges of each classifier are suggested by the literature [48, 47, 71] and `scikit-learn` library [72], and we totally have 4893 points in the configuration space

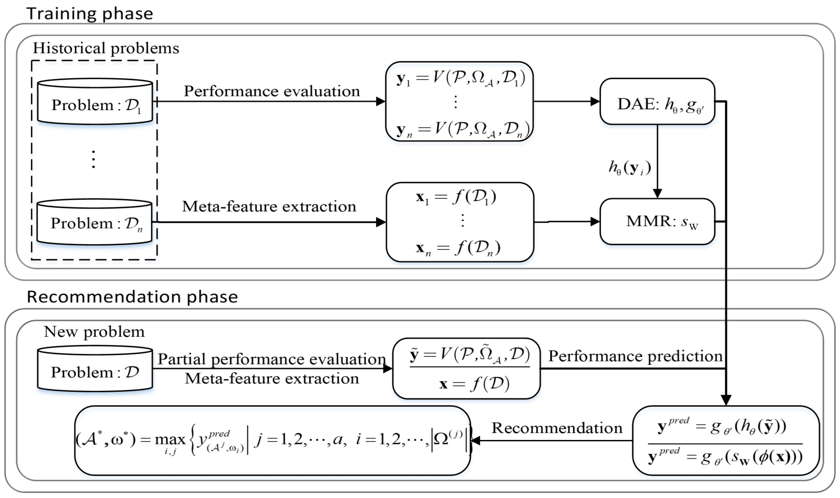

Training phase
Historical problems
Problem : $\mathcal{D}_1$
Problem : $\mathcal{D}_n$
Performance evaluation
Meta-feature extraction
$\mathbf{y}_1 = V(\mathcal{P},\Omega_{\mathcal{A}},\mathcal{D}_1)$
$\mathbf{y}_n = V(\mathcal{P},\Omega_{\mathcal{A}},\mathcal{D}_n)$
$\mathbf{x}_1 = f(\mathcal{D}_1)$
$\mathbf{x}_n = f(\mathcal{D}_n)$
DAE: $h_\theta, g_{\theta'}$
$h_\theta(\mathbf{y}_i)$
MMR: s_W
Recommendation phase
New problem
Problem : $\mathcal{D}$
Partial performance evaluation
Meta-feature extraction
$\tilde{\mathbf{y}} = V(\mathcal{P},\tilde{\Omega}_{\mathcal{A}},\mathcal{D})$
$\mathbf{x} = f(\mathcal{D})$
Performance prediction
$\mathbf{y}^{pred} = g_{\theta'}(h_\theta(\tilde{\mathbf{y}}))$
$\mathbf{y}^{pred} = g_{\theta'}(s_W(\phi(\mathbf{x})))$
Recommendation
$(\mathcal{A}^*,\omega^*) = \max_{i,j}\left\{y^{pred}_{(\mathcal{A}^j,\omega_i)} \middle| j=1,2,\cdots,a,\ i=1,2,\cdots,|\Omega^{(j)}|\right\}$

Figure 5.4: The illustration of meta-learning architecture based on our proposed DAE approach (cold-starting and warm-starting).

of CASH problem. This setting can cope with most of the problems in real scenarios. The number of hyperparameters and the size of each hyperparameter space of classifiers are summarized in Table 5.1, and more detailed information about the individual entries and ranges of hyperparameters can be found in Appendix VI.

Now suppose we have n historical classification problems $D = \{\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_n\}$ and the configuration search space $\Omega_{\mathscr{A}} = \mathbf{\Omega}^{(1)} \cup \mathbf{\Omega}^{(2)} \cup \cdots \cup \mathbf{\Omega}^{(a)}$. We determine the historical classification performance $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$ on the historical problems using 10-fold cross-validation. The detailed process of cross-validation can be found in Chapter 2.

Meta-feature extraction

The extraction of a meta-feature contains the computations of a set of metrics, called the meta-measures as defined. Based on our setting, there are eight types of meta-features [105] that are available, they are statistics & information-theory (SIT), landmarking (LM), relative landmarking (RL), problem complexity (PC), model structure (MS), structural information (SI), concept-based (Con), and clustering (Clu), respectively. To implement these meta-

Table 5.1: The summarization of the adopted candidate classifiers in terms of the number of hyperparameters and configurations.

Classifiers	#Hyperparameters	#Configurations
AdaBoost	2	70
Bagging	4	360
DT	3	760
ET	4	760
KNN	3	180
LDA	2	55
LR	3	796
MLP	4	72
QDA	1	11
RF	4	1520
SVM	2	399
Total	31	4983

features, we adopt the public Python library `pymfe` [23] in our system, the meta-measures deployed and their definitions can be found online[1]. After the meta-features are acquired, we normalize each meta-measure into interval $[0, 1]$ using the min-max scaling strategy to eliminate the bias possibly caused by some large-scale meta-measures. Some meta-measures that have too many missing values are deleted to avoid biases.

5.5.2 Recommendation

As we showed in Section 4, when a DAE is trained on the historical problems, then the classification performance of a new problem $\mathscr{D}$ can be predicted either by $\mathbf{y}^{pred} = g_{\theta'}(s_{\mathbf{W}}(\phi(\mathbf{x})))$ (warm-starting), where $\mathbf{x} = f(\mathscr{D})$ is the meta-feature vector of $\mathscr{D}$, or by $\mathbf{y}^{pred} = g_{\theta'}(h_\theta(\tilde{\mathbf{y}}))$ (cold-starting), where $\tilde{\mathbf{y}}$ is the evaluated real performance of $\mathscr{D}$ on a small proportion of the search space $\Omega_{\mathscr{A}}$. At last, we show how the recommended configurations for problem $\mathscr{D}$ are determined based on the estimated performance. We sort the entries of the predictive performance vector $\mathbf{y}^{pred} = [y_1^{pred}, y_2^{pred}, \cdots, y_m^{pred}]^\top$ in descending order as

$$\{y_{i_1}^{pred}, y_{i_2}^{pred}, \cdots, y_{i_m}^{pred}\},$$

[1]https://pymfe.readthedocs.io/en/latest/auto_pages/meta_features_description.html

where $y_{i_1}^{pred} \geq y_{i_2}^{pred} \geq \cdots \geq y_{i_m}^{pred}$. The configuration that has the highest predictive performance is thus recommended for this new problem $\mathscr{D}$, i.e.,

$$(\mathscr{A}, \boldsymbol{\omega})_r = \Omega_{\mathscr{A}}\{i_1\},$$

where $\Omega_{\mathscr{A}}\{i_1\}$ indicates the $(i_1)^{th}$ element of the configuration space $\Omega_{\mathscr{A}}$. Since the best configuration for a problem usually is not unique, multiple candidates for $\mathscr{D}$, i.e.,

$$\{(\mathscr{A}, \boldsymbol{\omega})_r^1, \cdots, (\mathscr{A}, \boldsymbol{\omega})_r^c\} = \Omega_{\mathscr{A}}\{i_1, \cdots, i_c\}, \tag{5.7}$$

where $(\mathscr{A}, \boldsymbol{\omega})_r^i$ indicates the i^{th}-order recommendation and $c \ll m$, can also be recommended.

5.6 EXPERIMENTS

In this section, we empirically test the effectiveness of our proposed recommendation approach. We first present our experimental settings, including evaluated classification problems, comparative methods, and recommendation performance evaluation metrics, and then discuss the empirical results from different aspects.

5.6.1 Experimental setup

Classification problems

We collect 127 classification problems from the well-known UCI machine learning repository[2], in which 82 problems are used as historical problems for training the meta-learner model (e.g., DAE) and the rest of 45 problems are used as testing ("new") problems. The range of the problem sizes is from 88 to 78K, which covers the most of problems often seen in

[2]http://archive.ics.uci.edu/ml/index.php

real applications. The range of the number of attributes (classes) is from 2 to 1558 (2 to 11). Other classification problems with a different number of dimensions and classes can be added to the proposed system directly without technical difficulty. The summarized information about the training and testing problems can be found in Appendix I.

Every selected problem is preprocessed by the following criteria: 1) data instances that contain missing values are removed; 2) each attribute is normalized using min-max scaling to ensure every attribute is equally important; 3) some attributes that have only one value are deleted too; 4) to perform a stratified 10-fold cross-validation, some classes of a problem that have less than 10 data instances are removed.

Comparative methods

Since the cold-starting DAE needs partially evaluate some configurations to obtain masked input data, we next verify if the performance of DAE is better than the best-evaluated configuration. Thus, we compare it to three search algorithms, such as random search (RS), SMBO, and active test (AT). For a given keeping ratio ϵ, then $\texttt{ceil}(|\Omega_A|\epsilon)$ configurations will be searched by each algorithm, and DAE will be trained and tested using the same ϵ. The cold-starting DAE is also compared to the cold-starting matrix factorization-based collaborative filtering [18, 47], denoted as MFCF hereafter.

The warm-starting DAE is compared to

- the warm-starting MFCF [18], where we adopt the warm-starting strategy we deployed in Section 4.2: an MMR $s_{\mathbf{W}}$ is trained to connect the meta-features of problems to the latent features of metadata learned by MFCF, then the predictive performance on new problem $\mathscr{D}$ is given by $\mathbf{y}^{pred} = \mathbf{W}^{\star} s_{\mathbf{W}}(\phi(\mathbf{x}))$ where $\mathbf{x}$ is the meta-feature of problem

Table 5.2: The parameter settings for DAE, MFCF, KKNN, and MLP deployed in the experiment with respect to names, usage, and values.

Approaches	Parameters	Usage	Values
DAE	`net_struc`	construct a DAE architecture	`m-2048-512-10` (symmetric)
	`activ_fun`	nonlinear transformation	`sigmoid`
	`max_iter_p`	iterations of pre-training	`300`
	`max_iter_f`	iterations of fine-tuning	`400`
	ρ	learning rate	`1e-3`
	β	regularization coefficient	`1e-2`
	$\mathcal{L}$	loss function	`binary cross-entropy`
	$k(\cdot,\cdot)$	kernel function for MMR	`RBF`
MFCF	d	dim of latent variable	`10`
	`max_iter`	iterations of training	`500`
	ρ	learning rate	`1e-4`
	α_{CF}	regularization coefficient	`100`
	β_{CF}	regularization coefficient	`100`
	$k(\cdot,\cdot)$	kernel function for MMR	`RBF`
KKNN	$k(\cdot,\cdot)$	kernel function	`RBF`
MLP	`net_struc`	neural network architecture	`r-10-512-2048-m`
	`activ_fun`	nonlinear transformation	`sigmoid`
	`max_iter`	iterations of training	`500`
	ρ	learning rate	`1e-3`
	β	regularization coefficient	`1e-3`
	$\mathcal{L}$	loss function	`binary cross-entropy`

$\mathscr{D}$ and $\mathbf{W}^\star$ is gained from MFCF;

- the similarity-based recommendation in which KNN [16] is used as the meta-learner and the Euclidean distance is adopted as the similarity measure;
- kernel KNN regression [55], where the prediction is given by $\mathbf{y}^{pred} = \frac{1}{S}\sum_{i=1}^{n} k(\mathbf{x}_i, \mathbf{x})\mathbf{y}_i$ where $S = \sum_{i=1}^{n} k(\mathbf{x}_i, \mathbf{x})$;
- multi-layer perceptron (MLP) [22] with the same structure of our warm-starting structure which is directly trained by feeding meta-features and the labels are the performance data on the historical problems. Our goal is to show that the learned latent features are capable of improving the performance of a neural network-based meta-learner;
- random search and SMBO whose search time is the time used for the meta-feature

extractions. Here the time consumed for the offline stage of our architecture is excluded since it can be accomplished with a flexible schedule. Here, we focus on the online stage, which is critical for the learning task discussed in this paper.

Evaluation metrics

We evaluate the performance of each approach using four metrics: classification accuracy rate (CA), recommendation accuracy rate (RA), hit rate (HR), and normalized discounted cumulative gain (NDCG), respectively. The first three are widely considered in meta-learning [15, 16, 87], while the last one is commonly used in recommender system studies [88].

Parameter setting in experiment

Since each method exploited in the experiment involves certain important parameters that should be well-determined, we summarize them in Table 5.2. Notice that DAE and MFCF share the same keeping ratio ϵ and the dimension of latent variables (we set it to 10); warm-starting DAE and MLP share the same network structure, activation function, learning rate, and loss function. Other important parameter settings will be specified later.

5.6.2 Empirical results

This section divides into four parts. The first and the second parts show the distribution of the extracted latent variables and the recommendation performance of cold-starting DAE, cold-starting MFCF, RS, AT, and SMBO. The third one reports the performance of warm-starting DAE, warm-starting MFCF, KNN, KKNN, and MLP. The fourth part studies the performance variation of warm-starting DAE when different levels of corruption are deployed. For each dataset, we recommend the first three best predictive configurations, i.e., $c = 3$ in

(5.7), and then the averaged performance is reported.

Latent variables

Under our framework, we show that the latent features not only can preserve the main structure of the high-dimensional configuration space but also maintain its generalization ability on unknown problems. To illustrate the distributions of latent features, we pursue the following steps, similar to [18]. Firstly, we split the training data into several clusters using the k-means algorithm where the number of clusters is suggested by Silhouette score [119]. Then the multidimensional scaling [120] is deployed to map the both configuration space and the latent space into a 2-dimensional space and each cluster is colored. Multidimensional scaling is known as a powerful tool that projects patterns to a lower dimensional subspace while maintaining the dissimilarity, or distance, among each pair of patterns. As shown in this section, the obtained 2-dimensional patterns can reflect the distributed information of the original spaces.

The visualizations of the latent features of DAE and MFCF are depicted in Figure 5.5. To show the impact of the keeping ratio on the latent features, we select three keeping ratios, i.e., $\epsilon = 0.001, 0.5$, and 1, which can be explained as very serious corruption, middle-level corruption, and no corruption, respectively. One can observe that: 1) when $\epsilon = 0.001$, latent features generated by MFCF do not present an explainable structure and the clusters are inseparable. As the increasing of ϵ, MFCF gradually produces the latent features that almost preserve the structure of the original configuration space, but the distribution along the y-axis is severely compressed; 2) in contrast, DAE gives a different latent space over these three keeping ratios. All points are lying on a parabola-like curve and each cluster gathers

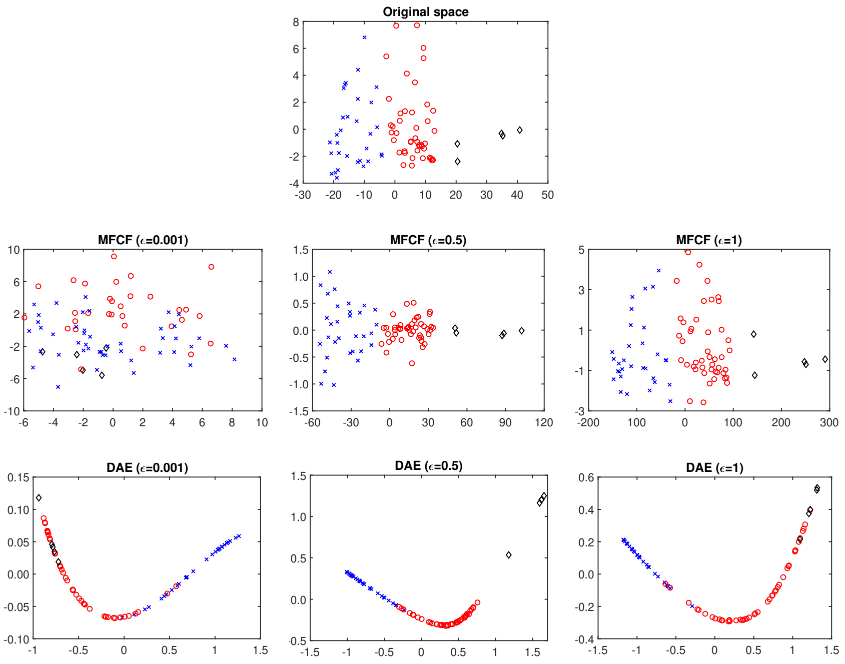


Figure 5.5: The distributions of the original configuration space (top), latent space of MFCF (middle), and latent space of DAE (bottom) on training instances when $\epsilon = 0.001, 0.5$, and 1. The two-dimensional data is acquired by adopting multidimensional scaling. One color represents one clustering and they are determined by the k-means algorithm.

around one location. The reason that leads to these two different distributions is that MFCF is a linear model so the extracted latent features tend to maintain their original distribution, our DAE model, however, is a neural network model, in which *sigmoid* activation function is adopted, and captures the nonlinear structure of the input space; 3) especially when $\epsilon = 0.5$, clusters can be easily classified for DAE. When there is no corruption for inputs ($\epsilon = 1$), DAE does not produce the best latent features, demonstrating that the denoising strategy indeed improves the model performance. Since MFCF specializes in primary distribution preservation, it may lose its ability to generalize for unknown instances. The later reported empirical results also support these conclusions.

Effectiveness of cold-starting DAE

Since evaluating a large group of configurations is time-consuming and search algorithms can usually find the applicable configurations, we set keeping ratio $\epsilon = 0.0006$ in this experiment, i.e., only three configurations are kept for training DAE on historical problems and testing on the new problems, to show the merits of our method. We repeat 10 times recommendations for DAE, MFCF, RS, and SMBO to reduce the influence of randomness, the comparison results are summarized in Table 5.3 where '$-$' means the corresponding data is unavailable. As we can see, our DAE method outperforms both MFCF and search algorithms such as RS, AT and SMBO under the applied evaluation metrics. DAE has an ACA of 77.02% whereas the ACA of MFCF is 75.94%. For those search algorithms, AT has the best ACA of 72.58%, and RS is slightly better than SMBO. This is because when search rounds are severely limited, finding a promising configuration is challenging. We can also reach the same conclusion from the results of ARA. Although the HR of MFCF is higher than that of DAE, the actual difference is only one out of 135 recommendations. The results of NDCG under different positions ρ show how close are the top-ρ recommended configurations to the real ranks. One can see that the NDCG@1 of DAE is 0.8919, and it is slowly declining when ρ is increasing but it is 0.8697 when $\rho = 20$, which shows that the effectiveness of the top-20 recommendations of DAE is stable with classification ability; however, MFCF has the lowest NDCG (0.7526) when $\rho = 1$ and it is increasing as ρ increases, which is not appreciated in real scenarios. On the other hand, we find that AT has the highest NDCG of 0.9164 for the optimal one evaluated so far but NDCG@3 declined sharply to 0.7747. SMBO has a higher NDCG than RS although it is inferior to RS in terms of ACA, ARA, and HR, which implies

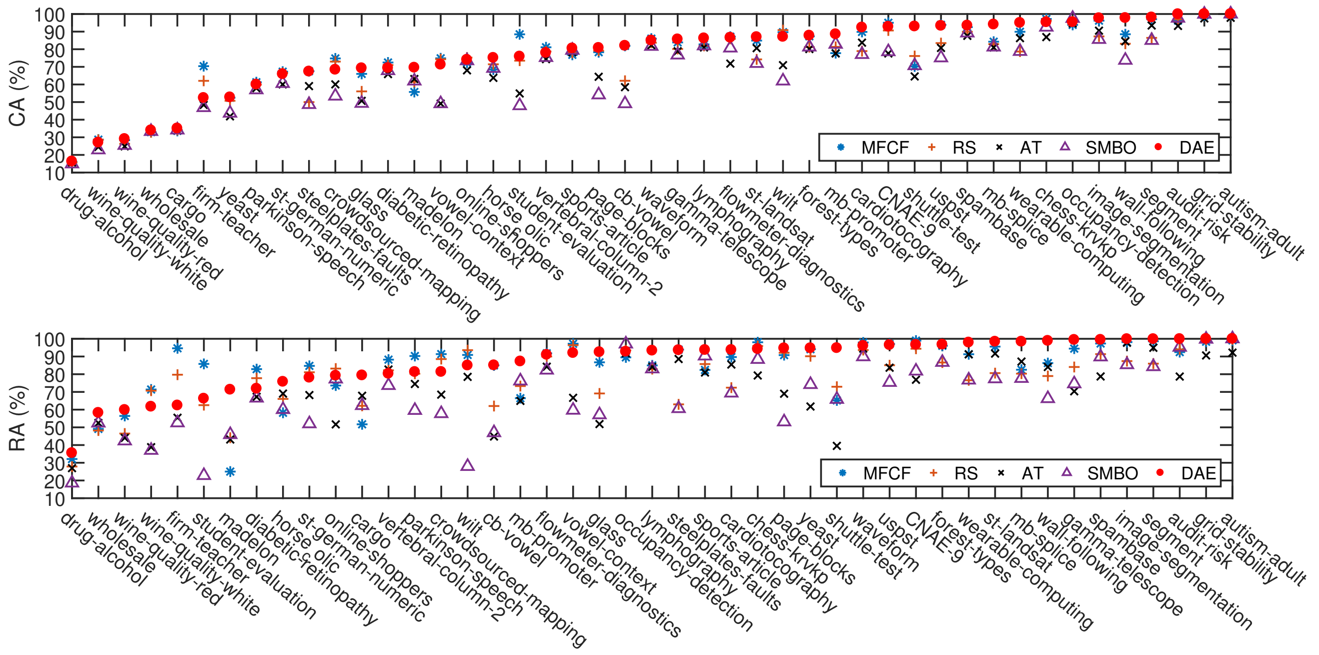


Figure 5.6: The comparisons of CA and RA between MFCF, RS, AT, SMBO, and DAE on each testing dataset.

Table 5.3: The comparisons of ACA, ARA, HR, and NDCG between MFCF, RS, SMBO, AT, and DAE. The ACA of the real optimal configurations over testing problems is 81.78%.

Methods	ACA	ARA	HR	NDCG@1	NDCG@3	NDCG@5	NDCG@7	NDCG@10	NDCG@20
DAE	**77.02**	**86.64**	74.81	0.8919	**0.8862**	**0.8794**	**0.8734**	**0.8739**	**0.8697**
MFCF	75.94	84.04	**75.56**	0.7526	0.7675	0.7770	0.7835	0.7889	0.7944
AT	72.58	78.94	60.00	**0.9164**	0.7747	-	-	-	-
RS	68.30	69.66	42.44	0.7506	0.5676	-	-	-	-
SMBO	67.10	68.32	40.74	0.8585	0.6662	-	-	-	-

Table 5.4: The p-values of Wilcoxon signed-rank test on CA and RA. We accept the alternative hypothesis H_a if the p-value is less than 0.05. Here '>' means overpass.

H_a	CA	RA
DAE>MFCF	0.038	0.033
DAE>RS	0.000	0.000
DAE>AT	0.000	0.000
DAE>SMBO	0.000	0.000

that the optimal configuration found by SMBO is better than that of RS but the other order recommendations of SMBO are mediocre.

To show the performance differences on each testing problem, we display the scatter plots in Figure 5.6 concerning CA and RA where the order of dataset ID is rearranged in ascending order based on the performance of DAE. One can observe that DAE can tie with or overpass MFCF, RS, AT, and SMBO on most of the problems with respect to CA, and the differences are more obvious for RA. Furthermore, to show if these differences are statistically significant, we applied the Wilcoxon signed-rank test at 0.05 significance for each pair, e.g., DAE versus MFCF, and the null hypothesis is that DAE does not outperform MFCF (or RS, AT, and SMBO) significantly. We report the p-values of the test in Table 5.4, and one can see that all p-values are less than 0.05 so we accept the alternative hypothesis, namely DAE overpasses MFCF, RS, AT, and SMBO statistically significantly. Above all, our method can suggest much better configurations than search algorithms when the allowable search rounds are severely insufficient.

Effectiveness of warm-starting DAE

To show the performance of the warm-starting DAE with other meta-learning baselines, we adopt all eight state-of-the-art meta-features, such as SIT, MS, PC, LM, RL, Con, SI, and Clu, and report their outcomes separately. We also unify these eight meta-features into a single meta-feature, dubbed Uni, whose performance is investigated too. The parameters

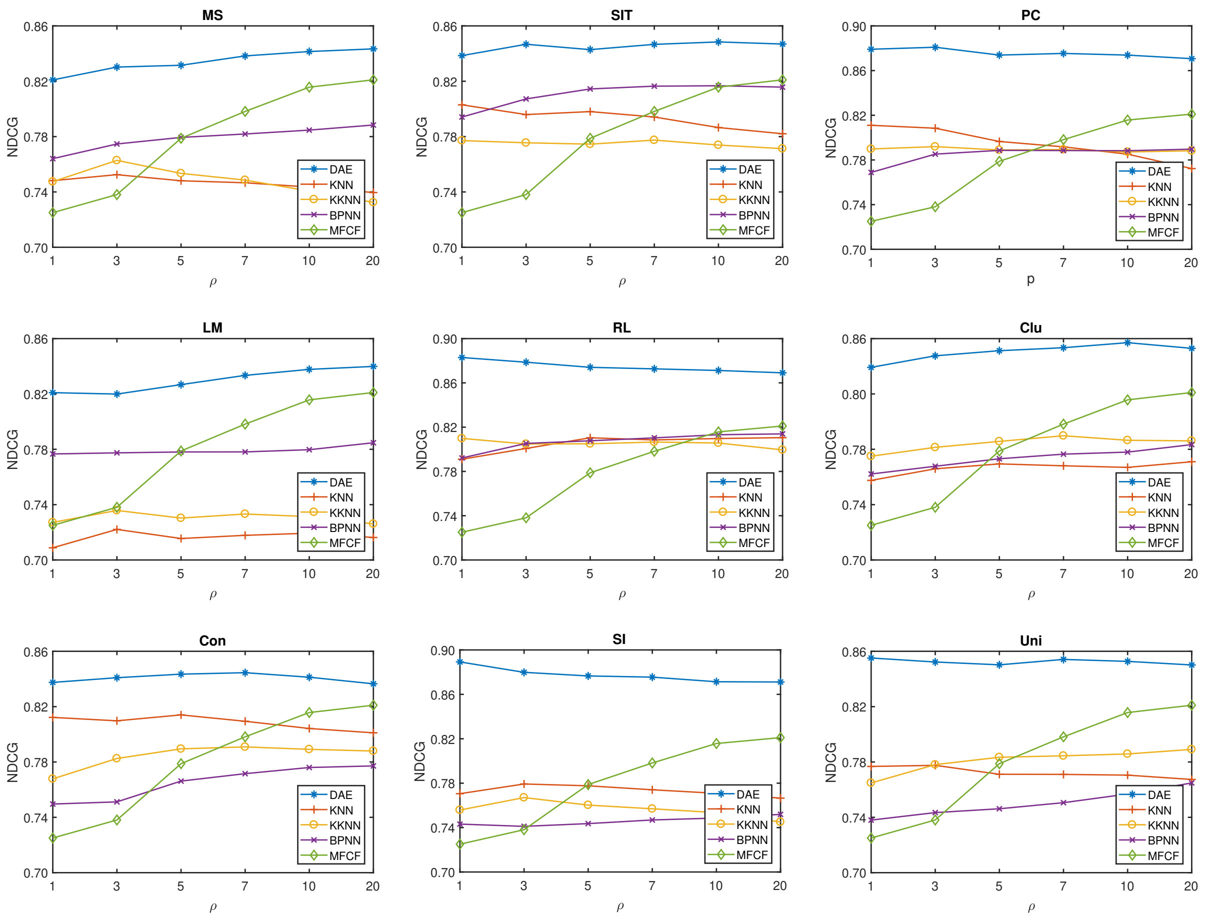


Figure 5.7: The comparisons of NDCG between KNN, KKNN, MLP, MFCF, and DAE. We do not give the NDCGs about RS and SMBO since they are not available.

Table 5.5: The comparisons of ACA (%), ARA (%), and HR (%) between KNN, KKNN, MLP, MFCF, RS, SMBO, and DAE when nine types of meta-features are employed. The maximal value of each column is highlighted.

Methods	MS			SIT			PC		
	ACA	ARA	HR	ACA	ARA	HR	ACA	ARA	HR
DAE	**77.31**	**87.45**	**81.48**	**77.33**	**87.39**	**85.19**	**77.91**	**89.30**	**85.19**
KNN	75.96	84.31	77.04	76.61	85.31	75.56	76.70	85.82	78.52
KKNN	75.94	84.17	76.30	76.40	85.00	76.30	76.75	86.23	78.52
MLP	75.97	84.59	77.04	76.60	86.07	79.26	76.05	84.75	74.81
MFCF	75.84	83.69	76.30	75.84	83.69	76.30	75.84	83.69	75.84
RS	65.30	64.25	13.33	64.91	64.42	13.33	65.78	65.23	17.78
SMBO	65.80	65.16	15.56	65.51	64.62	15.56	64.21	63.04	6.67
Methods	LM			RL			Clu		
	ACA	ARA	HR	ACA	ARA	HR	ACA	ARA	HR
DAE	**76.99**	**86.61**	**83.70**	**77.75**	**88.84**	**84.44**	**77.28**	**87.30**	**83.70**
KNN	74.85	83.29	74.81	77.27	86.35	75.56	75.61	84.67	74.81
KKNN	75.82	84.32	76.30	76.93	85.82	77.78	76.17	85.32	75.56
MLP	76.00	84.52	73.33	76.72	85.88	81.48	75.60	84.38	75.56
MFCF	75.84	83.69	76.30	75.84	83.69	76.30	75.84	83.69	75.84
RS	65.81	65.58	13.33	66.54	66.10	15.56	65.86	66.19	17.78
SMBO	65.06	63.10	13.33	65.81	65.80	22.22	65.14	64.58	13.33
Methods	Con			SI			Uni		
	ACA	ARA	HR	ACA	ARA	HR	ACA	ARA	HR
DAE	**77.40**	**86.98**	**81.48**	**76.98**	**87.72**	**78.52**	**77.35**	**87.38**	**85.19**
KNN	76.61	85.53	72.59	76.16	84.57	75.56	75.83	84.09	77.78
KKNN	76.60	86.10	77.78	76.09	85.11	73.33	76.13	85.49	80.74
MLP	76.12	84.55	76.30	75.87	84.57	73.33	75.91	84.64	77.78
MFCF	75.84	83.69	76.30	75.84	83.69	76.30	75.84	83.69	75.84
RS	65.88	65.33	11.11	64.91	63.18	8.89	65.09	64.03	11.11
SMBO	63.85	61.20	15.56	65.15	63.60	15.56	63.03	57.81	15.56

involved in each approach are determined by grid search, e.g., RBF kernel width '`gamma`' in MMR (warm-starting model for DAE and MFCF) and KKNN where the range for '`gamma`' is given by $\{1e-4, 1e-3, 1e-2, 1e-1, 5e-1, 1, 5, 1e1, 5e1, 1e2, 5e2, 1e3\}$, and the number of neighbors '`k`' in kNN where the range for '`k`' is given by $\{1, 2, \cdots, 50\}$. The keeping ratios ϵ for DAE and MFCF training are set to 0.1. Similarly, to eliminate the randomness, we report the average performance of ten times repetition for DAE, MFCF, MLP, RS, and SMBO. The comparative results are summarized in Table 5.5 (ACA, ARA, and HR) and Figure 5.7 (NDCG). Here the problem '`hand postures`'' is excluded from the historical problems in this comparison due to the fact that its PC, Con, and Clu meta-features contain too many missing values.

One can observe that: 1) meta-learning-based approaches outperform RS and SMBO dramatically throughout the nine meta-features, showing their effectiveness and efficiency. Since the allowable time for the search is limited, there is no obvious performance difference between RS and SMBO; 2) the underlying meta-learning approaches show different performance on each meta-feature deployed, except for MFCF, which performs consistently throughout the deployed meta-features. PC and RL generate the best results, SIT and Uni take the second place, MS, LM, and Clu occupy the tertiary, and Con and SI perform the worst; 3) DAE outperforms the comparative meta-learning baselines, namely, warm-starting MFCF, MLP, KNN, and KKNN, demonstrating its superior recommendation effectiveness for the CASH problem. Especially, DAE-PC has an ACA of 77.91%, ARA of 89.30%, and HR of 85.19%; however, other baselines achieve the best ACA of 76.75% (KKNN), ARA of 86.23% (KKNN), and HR of 78.52% (KNN and KKNN); 4) Among these four comparative baselines, cold-starting MFCF has the mediocre performance while the other three, e.g., KNN, KKNN, and MLP, have similar performance, and none of them show superior recommendation capacity; 5) In terms of NDCG, there is no doubt that DAE uniformly outweighs the other four baselines, demonstrating its potential of recommending promising configurations. Additionally, we notice that DAE-SI owns the highest NDCG while the average effectiveness of the first three recommendations is less satisfying as shown in Table 5.5. By looking into these three recommendations individually, we find that the second and third-order recommendations do not produce good results with an ACA of 76.82%; 6) The overall performance of warm-starting DAE outperforms that of the cold-starting DAE through different meta-features. Nevertheless, the cold-starting DAE tends to have a higher NDCG.

To show the performance differences among the baselines on each dataset, we only display the comparisons of RA in Figure 5.8 since we can get the same results from CA and it is easier to tell the performance differences of baselines on RA. We also only give the plots for SIT, PC, and RL to save space. As we can see from Figure 5.8, our method, DAE, can perform better than or the same as the other four baselines on most of the datasets, and it has stable performance. The RA of DAE is over 80% on 38 out of 45 cases under the three investigated meta-features, which proves the potential of recommending the hyperparameters with the high classification ability of our method. However, other baselines tend to produce mediocre recommendations sometimes; for instance, on dataset `drug-alcohol` (ID is 9), DAE has an RA that is over 90% but the RA of KNN, KKNN, and MFCF is less than 50% and MLP is better with an average RA of 60%. Similarly, we can take a look at dataset `shuttle-test` (ID is 40). We also observe that DAE failed on datasets `madelon` and `vargo` (ID is 21 and 31, respectively) when SIT meta-feature is employed but it does the best job

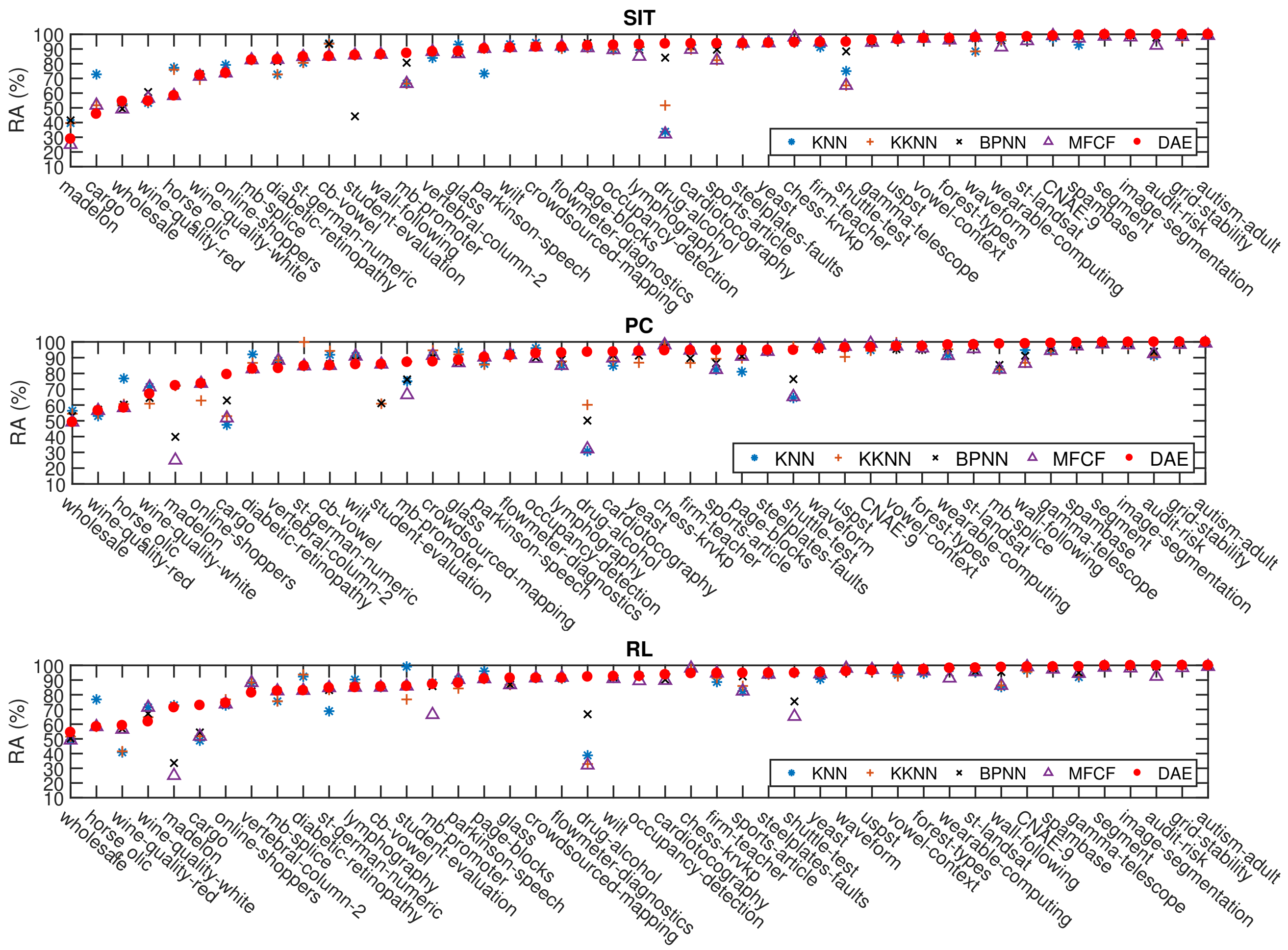


Figure 5.8: The comparisons of RA between KNN, KKNN, MLP, MFCF, and DAE on each testing dataset.

Table 5.6: The p-values of Wilcoxon signed-rank test on each type of meta-feature. We accept the alternative hypothesis H_a when the p-value is less than 0.05.

Metric	H_a	MS	SIT	PC	LM	RL	Clu	Con	SI	Uni
CA	DAE>KNN	0.016	0.043	$\underline{0.241}$	0.011	$\underline{0.131}$	0.009	0.035	$\underline{0.088}$	0.011
	DAE>KKNN	0.019	0.028	0.041	0.018	0.036	0.019	$\underline{0.138}$	0.036	0.033
	DAE>MLP	0.000	0.022	0.000	0.013	0.000	0.006	0.000	0.009	0.005
	DAE>MFCF	0.009	0.000	0.003	0.027	0.000	0.000	0.013	$\underline{0.072}$	0.002
	DAE>RS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	DAE>SMBO	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
RA	DAE>KNN	0.016	$\underline{0.058}$	$\underline{0.219}$	0.019	$\underline{0.138}$	0.028	0.022	$\underline{0.107}$	0.013
	DAE>KKNN	0.017	0.026	0.037	0.028	0.040	0.022	$\underline{0.168}$	0.038	$\underline{0.054}$
	DAE>MLP	0.000	0.024	0.000	0.025	0.000	0.009	0.001	0.015	0.007
	DAE>MFCF	0.008	0.000	0.004	$\underline{0.053}$	0.000	0.001	0.016	$\underline{0.062}$	0.003
	DAE>RS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	DAE>SMBO	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

using PC and RL meta-features.

We also report the Wilcoxon signed-rank test results in Table 5.6 to statistically compare DAE with other baselines. We can see that DAE significantly overpasses KNN, KKNN, MLP, and MFCF over 31 out of 36 cases for CA, and 28 out of 36 cases for RA, and it significantly outperforms RS and SMBO in all cases. It is worth mentioning that DAE fails to outweigh KNN on PC, RL, and SI meta-features, which suggests the potential of the KNN method. Even though some p-values of the test on SIT, LM, SI, and Uni are larger than 0.05, they are still close to 0.05 and the significant difference can be guaranteed if we gently decrease the confidence level.

In summary, we can conclude that our method can suggest more reliable outcomes compared to other popular meta-learning baselines.

Influence of keeping ratios

In this experiment, we are going to show the influence of the keeping ratio ϵ on the performance of the warm-starting DAE. To do so, we select twelve different values for ϵ, namely, $\epsilon \in \{0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1\}$, which cover the range between "severe corruption" and "no corruption". Especially when $\epsilon = 1$, we can see the performance of the

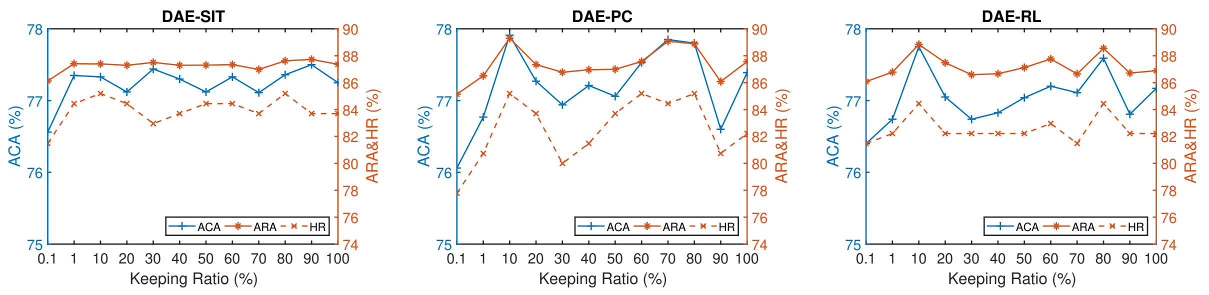


Figure 5.9: The effectiveness of warm-starting DAE when different keeping ratios (ϵ) are deployed.

warm-starting DAE when the denoising criterion is removed. Also, the optimal parameters for MMR are determined by grid search. The empirical outcomes (ACA, ARA, and HR) over three meta-features (SIT, PC, and RL) are depicted in Figure 5.9, and the results of the other six meta-features are omitted to avoid redundancy.

One can observe that: 1) the performance of DAE-SIT under RA metric is stable when ϵ is varying from 0.01 to 0.9, but CA and HR are fluctuating gently. For DAE-PC and DAE-RL, the changes of the three metrics are dramatic and have similar tendencies, i.e., the highest scores are obtained when $\epsilon = 0.1$ and 0.8. By examining the extracted latent features as we did in section 6.2.1, the best DAE model is obtained when keeping ratios are 0.1 and 0.8 where the extracted latent features appear to have the optimal distributions, i.e., three clusters are well-classified. This is mainly due to the relatively small ratio, as well as the relatively large ratio, which can better grasp both similarity and dissimilarity among the training datasets; 2) When data is either severely corrupted ($\epsilon = 0.001$) or completely clean ($\epsilon = 1$), DAE under the three meta-features does not provide the optimal performance, which implies the importance of the denoising principle and a proper degree of corruption is also the key of success; 3) Among the three investigated meta-features, although PC and RL are the most promising to yield satisfying outcomes, they do not have a good HR comparing to SIT. Also, the performance of PC and RL is sensitive to the applied keeping ratios.

5.7 CONCLUDING REMARKS

In this chapter, we developed a new approach to solving the well-known CASH problem via integrating the sparse denoising autoencoder approach with meta-learning. A DAE network is trained to extract the subtle latent features from the high-dimensional configuration space of CASH, based on the consensus that two problems should be similar and share the best configurations when their latent features are quantitatively close. We developed two strategies for latent feature generation for new problems, i.e., cold-starting and warm-starting. The first one applies the denoising principle as well, namely, a 'masked' performance vector on new problems is generated and fed into the DAE network; the second approach predicts the latent features using the meta-features of problems through regression learning. The acquired latent features are finally fed to the decoder network to generate a performance prediction over the entire configuration space.

We have conducted extensive experiments to show the effectiveness of our approach under various standard metrics. Compared to the existing baselines, DAE shows the best recommendation capacity under four commonly-used evaluation metrics. We found that the warm-starting DAE together with complex meta-features can produce the most desirable outcomes. A possible improvement in performance can be gained by combining with the SMBO algorithm which is initialized by the recommendations of DAE if the cost is permissible.

Future work could focus on learning the hierarchical relationships existing in the configuration space of the CASH problem. In current literature, including this paper, the configuration space is instantiated as a high-dimensional vector, which may lose some structural information and hyperparameter dependence. Maintaining a hierarchical structure more closely may improve the recommendation performance substantially. Such a study will be reported elsewhere.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSIONS

In this dissertation, we solved the problem of hyperparameter selection under the framework of meta-learning. Compared to traditional search-based algorithms, our approaches do not need any configuration evaluations during the recommendation phase, which facilitates the applications of efficient machine learning that are required by nowadays ever-increasing data. Focusing on the shortages of the existing approaches in meta-learning, we proposed four new methods which target automatic meta-feature selection and the learning of the spatial structure of hyperparameter search space. A significant amount of experiments were conducted on real-world classification problems to verify the effectiveness of our methods, and promising results have been obtained by comparing them to the state-of-the-art meta-learning baselines as well as search algorithms.

6.2 FUTURE WORK

Although meta-learning has shown its prospective performance in literature, there are still many interesting topics that deserve careful investigation in the future.

Meta-feature extraction: Most of the existing meta-features are only suitable for classification datasets, the applicable ones for other machine learning datasets, such as regression and clustering, are severely limited. This is because of the popularity of classification algorithms which have drawn the attention of researchers. Therefore, the study of regression and clustering algorithms in meta-learning is insufficient. In the future, the possible research topics can be the meta-feature extraction for regression, clustering, or other machine learning datasets. On the other hand, the current meta-features are only suitable for traditional machine learning datasets, there are no applicable ones for image datasets, which prevents us to perform automatic hyperparameter selection for state-of-the-art deep neural networks, such as residual networks and Vision Transformers. Except for the study in this dissertation, we did not find any other related work dealing with the problem of hyperparameter recommendation for deep neural networks under the framework of meta-learning.

Hierarchical structure-preserving: To preserve the spatial structure of the search space, we proposed to organize the search space as a multi-dimensional tensor, but this strategy is only suitable for those algorithms that have independent hyperparameters while there are many scenarios where the search spaces have conditional hyperparameters which present a complicated hierarchical structure, e.g., the search space of CASH problem. Therefore, if we can find a data organization approach that preserves the hierarchical structure of the search space, more useful information can be retained for a successful recommendation.

Deep learning for meta-learning: Deep learning is one of the most popular research topics in recent years because of its outstanding performance in image recognition and classification. Besides the aforementioned meta-feature extraction of the image datasets for hyperparameter recommendation of deep neural networks, another promising problem is to apply deep learning algorithms as recommendation algorithms in meta-learning. In current studies, the traditional machine learning algorithms are widely used as recommendation algorithms and only very simple neural network-based algorithms, such as multi-layer perceptron, are leveraged so far. We plan to investigate the representative deep convolutional neural networks (CNN) as a meta-learner and the main challenge is how to build a relationship between meta-features and the performance of hyperparameters.

REFERENCES

[1] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[3] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] C. M. Bishop *et al.*, *Neural networks for pattern recognition.* Oxford university press, 1995.

[5] M. Feurer and F. Hutter, “Hyperparameter optimization,” *Automated Machine Learning: Methods, Systems, Challenges*, pp. 3–33, 2019.

[6] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.

[7] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[8] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[9] R. Kohavi and G. H. John, “Automatic parameter selection by minimizing estimated error,” in *Machine Learning Proceedings 1995.* Elsevier, 1995, pp. 304–312.

[10] H. Mühlenbein and G. Paass, “From recombination of genes to the estimation of distributions I. Binary parameters,” in *International Conference on Parallel Problem Solving from Nature.* Springer, 1996, pp. 178–187.

[11] X. Guo, J. Yang, C. Wu, C. Wang, and Y. Liang, “A novel LS-SVMs hyper-parameter selection based on particle swarm optimization,” *Neurocomputing*, vol. 71, no. 16-18, pp. 3211–3215, 2008.

[12] J. Vanschoren, “Meta-learning,” *Automated Machine Learning: Methods, Systems, Challenges*, pp. 35–61, 2019.

[13] P. Brazdil, J. N. v. Rijn, C. Soares, and J. Vanschoren, “Metalearning approaches for algo-

rithm selection II," in *Metalearning.* Springer, 2022, pp. 77–102.

[14] T. A. Gomes, R. B. Prudêncio, C. Soares, A. L. Rossi, and A. Carvalho, "Combining meta-learning and search techniques to select parameters for support vector machines," *Neurocomputing*, vol. 75, no. 1, pp. 3–13, 2012.

[15] Q. Song, G. Wang, and C. Wang, "Automatic recommendation of classification algorithms based on data set characteristics," *Pattern Recognition*, vol. 45, no. 7, pp. 2672–2689, 2012.

[16] L. Deng, W.-S. Chen, and B. Pan, "Automatic classifier selection based on classification complexity," in *Proceedings of the first Chinese Conference on Pattern Recognition and Computer Vision*, 2018, pp. 292–303.

[17] D. Stern, R. Herbrich, T. Graepel, H. Samulowitz, L. Pulina, and A. Tacchella, "Collaborative expert portfolio management," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence AAAI-10*, 2010, pp. 179–184.

[18] M. Mısır and M. Sebag, "Alors: An algorithm recommender system," *Artificial Intelligence*, vol. 244, pp. 291–314, 2017.

[19] B. Bilalli, A. Abelló Gamazo, and T. Aluja Banet, "On the predictive power of meta-features in OpenML," *International Journal of Applied Mathematics and Computer Science*, vol. 27, no. 4, pp. 697–712, 2017.

[20] S. Y. Sohn, "Meta analysis of classification algorithms for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1137–1144, 1999.

[21] P. B. Brazdil, C. Soares, and J. Pinto da Costa, "Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[22] K. A. Smith, F. Woo, V. Ciesielski, and R. Ibrahim, "Modelling the relationship between problem characteristics and data mining algorithm performance using neural networks," in *Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining, and Complex Systems*, vol. 11. ASME Press, 2001, pp. 356–362.

[23] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. Garcia, J. T. Oliva, A. C. de Carvalho *et al.*, "MFE: Towards reproducible meta-feature extraction," *Journal of Machine Learning Research*, vol. 21, no. 111, pp. 1–5, 2020.

[24] H. Bensusan, C. G. Giraud-Carrier, and C. J. Kennedy, "A higher-order approach to meta-learning." *ILP Work-in-progress Reports*, vol. 35, pp. 33–42, 2000.

[25] Y. Peng, P. A. Flach, C. Soares, and P. Brazdil, "Improved dataset characterization for meta-learning," in *International Conference on Discovery Science.* Springer, 2002, pp. 141–152.

[26] H. Bensusan and C. Giraud-Carrier, "Discovering task neighbourhoods through landmark learning performances," in *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2000, pp. 325–330.

[27] R. P. W. Duin, E. Pekalska, and D. M. J. Tax, "The characterization of classification problems by classifier disagreements," in *Proceedings of the Seventeenth International Conference on Pattern Recognition*, 2004, pp. 140–143.

[28] C. Soares, J. Petrak, and P. Brazdil, "Sampling-based relative landmarks: Systematically test-driving algorithms before choosing," in *Portuguese Conference on Artificial Intelligence.* Springer, 2001, pp. 88–95.

[29] E. Perez and L. A. Rendell, "Learning despite concept variation by finding structure in attribute-based data," in *In Proceedings of the Thirteenth International Conference on Machine Learning.* Citeseer, 1996.

[30] R. Vilalta, "Understanding accuracy performance through concept characterization and algorithm analysis," in *Proceedings of the ICML-99 Workshop on Recent Advances in Meta-learning and Future Work*, 1999, pp. 3–9.

[31] B. A. Pimentel and A. C. de Carvalho, "A new data characterization for selecting clustering algorithms using meta-learning," *Information Sciences*, vol. 477, pp. 203–219, 2019.

[32] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 289–300, 2002.

[33] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning.* Springer, 2019.

[34] A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, A. M. Maraval, H. Jianye, J. Wang, J. Peters *et al.*, "HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation," *Journal of Artificial Intelligence Research*, vol. 74, pp. 1269–1349, 2022.

[35] J. Wu, M. Poloczek, A. G. Wilson, and P. Frazier, "Bayesian optimization with gradients,"

Advances in Neural Information Processing Systems, vol. 30, 2017.

[36] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 240–248.

[37] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[38] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[39] F. Glover, "Tabu search part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[40] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 193–225, 1997.

[41] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyperparameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, pp. 1–16, 2016.

[42] J. R. Rice, "The algorithm selection problem," in *Advances in Computers*. Elsevier, 1976, vol. 15, pp. 65–118.

[43] C. Soares, P. B. Brazdil, and P. Kuba, "A meta-learning method to select the kernel width in support vector regression," *Machine Learning*, vol. 54, no. 3, pp. 195–209, 2004.

[44] T. A. F. Gomes, R. B. C. Prudêncio, C. Soares, A. L. D. Rossi, and A. Carvalho, "Combining meta-learning and search techniques to select parameters for support vector machines," *Neurocomputing*, vol. 75, no. 1, pp. 3–13, 2012.

[45] M. Feurer, J. T. Springenberg, and F. Hutter, "Using meta-learning to initialize bayesian optimization of hyperparameters," in *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection-Volume 1201*. Citeseer, 2014, pp. 3–10.

[46] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, pp. 1–25, 2009.

[47] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell, "OBOE: Collaborative filtering for AutoML model selection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1173–1183.

[48] N. Fusi, R. Sheth, and M. Elibol, "Probabilistic matrix factorization for automated machine learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3348–3357.

[49] R. G. Mantovani, A. L. Rossi, J. Vanschoren, and A. C. de Carvalho, "Meta-learning recommendation of default hyper-parameter values for SVMs in classification tasks," in *MetaSel@ PKDD/ECML*, 2015, pp. 80–92.

[50] R. G. Mantovani, A. L. Rossi, J. Vanschoren, B. Bischl, and A. C. Carvalho, "To tune or not to tune: Recommending when to adjust SVM hyper-parameters via meta-learning," in *2015 International Joint Conference on Neural Networks (IJCNN).* IEEE, 2015, pp. 1–8.

[51] P. Brazdil, J. Gama, and B. Henery, "Characterizing the applicability of classification algorithms using meta-level learning," in *European Conference on Machine Learning.* Springer, 1994, pp. 83–102.

[52] S. Ali and K. A. Smith, "On learning algorithm selection for classification," *Applied Soft Computing*, vol. 6, no. 2, pp. 119–138, 2006.

[53] S. Dyrmishi, R. Elshawi, and S. Sakr, "A decision support framework for AutoML systems: A meta-learning approach," in *2019 International Conference on Data Mining Workshops (ICDMW).* IEEE, 2019, pp. 97–106.

[54] J. Gama and P. Brazdil, "Characterization of classification algorithms," in *Portuguese Conference on Artificial Intelligence.* Springer, 1995, pp. 189–200.

[55] H. Bensusan and A. Kalousis, "Estimating the predictive accuracy of a classifier," in *European Conference on Machine Learning.* Springer, 2001, pp. 25–36.

[56] I. Khan, X. Zhang, M. Rehman, and R. Ali, "A literature survey and empirical study of meta-learning for classifier selection," *IEEE Access*, vol. 8, pp. 10 262–10 281, 2020.

[57] P. Brazdil, J. N. van Rijn, C. Soares, and J. Vanschoren, "Dataset characteristics (Metafeatures)," in *Metalearning.* Springer, 2022, pp. 53–75.

[58] L. Todorovski, P. Brazdil, and C. Soares, "Report on the experiments with feature selection in meta-level learning," in *Proceedings of the PKDD-00 Workshop on Data Mining, Decision Support, Meta-learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions.* Citeseer, 2000.

[59] A. Kalousis and M. Hilario, "Feature selection for meta-learning," in *Pacific-Asia Conference*

on Knowledge Discovery and Data Mining. Springer, 2001, pp. 222–233.

[60] R. M. Cruz, R. Sabourin, and G. D. Cavalcanti, "META-DES. Oracle: Meta-learning and feature selection for dynamic ensemble selection," *Information Fusion*, vol. 38, pp. 84–103, 2017.

[61] C. X. Ren, D. Q. Dai, and Y. Hong, "Robust classification using 2,1-norm based regression model," *Pattern Recognition*, vol. 45, no. 7, pp. 2708–2718, 2012.

[62] Y.-F. Yu, C.-X. Ren, M. Jiang, M.-Y. Sun, D.-Q. Dai, and G. Guo, "Sparse approximation to discriminant projection learning and application to image classification," *Pattern Recognition*, vol. 96, p. 106963, 2019.

[63] C. Li, X. Wang, W. Dong, J. Yan, Q. Liu, and H. Zha, "Joint active learning with feature selection via cur matrix decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 6, pp. 1382–1396, 2018.

[64] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[65] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group Lasso," *Journal of Computational and Graphical Statistics*, vol. 22, no. 2, pp. 231–245, 2013.

[66] Y. Li, B. Nan, and J. Zhu, "Multivariate sparse group lasso for the multivariate multiple linear regression with an arbitrary group structure," *Biometrics*, vol. 71, no. 2, pp. 354–363, 2015.

[67] G. Obozinski, M. J. Wainwright, and M. I. Jordan, "Support union recovery in high-dimensional multivariate regression," *The Annals of Statistics*, vol. 39, no. 1, pp. 1–47, 2011.

[68] Ran-Chao, Ren, Chuan-Xian, Xiao-Lin, Yan, and Hong, "Sliced inverse regression with adaptive spectral sparsity for dimension reduction," *IEEE Transactions on Cybernetics*, vol. 47, no. 3, pp. 759–771, 2017.

[69] E. R. Mansfield and B. P. Helms, "Detecting multicollinearity," *The American Statistician*, vol. 36, no. 3a, pp. 158–160, 1982.

[70] D. Dua and C. Graff, "UCI machine learning repository [http://archive. ics. uci. edu/ml]. Irvine, CA: University of California, School of Information and Computer Science," *IEEE*

Transactions on Pattern Analysis and Machine Intelligence, 2019.

[71] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Tech. Rep., 2003.

[72] 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*, vol. 12, pp. 2825–2830, 2011.

[73] M. Kück, S. F. Crone, and M. Freitag, "Meta-learning with neural networks and landmarking for forecasting model selection an empirical evaluation of different feature sets applied to industry data," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 1499–1506.

[74] X. Li, Y. Wang, and R. Ruiz, "A survey on sparse learning models for feature selection," *IEEE transactions on cybernetics*, vol. 52, no. 3, pp. 1642–1660, 2020.

[75] V. Roth, "The generalized LASSO," *IEEE Transactions on Neural Networks*, vol. 15, no. 1, pp. 16–28, 2004.

[76] P. Ravikumar, J. Lafferty, H. Liu, and L. Wasserman, "Sparse additive models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 71, no. 5, pp. 1009–1030, 2009.

[77] M. Yamada, W. Jitkrittum, L. Sigal, E. P. Xing, and M. Sugiyama, "High-dimensional feature selection by feature-wise kernelized Lasso," *Neural Computation*, vol. 26, no. 1, pp. 185–207, 2014.

[78] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik, "Feature selection for SVMs," *Advances in Neural Information Processing Systems*, vol. 13, 2000.

[79] Y. Grandvalet and S. Canu, "Adaptive scaling for feature selection in SVMs," *Advances in Neural Information Processing Systems*, vol. 15, 2002.

[80] F. Li, Y. Yang, and E. Xing, "From Lasso regression to feature vector machine," *Advances in Neural Information Processing systems*, vol. 18, 2005.

[81] H. Chen, X. Wang, C. Deng, and H. Huang, "Group sparse additive machine," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[82] R. Tibshirani, “The Lasso method for variable selection in the Cox model,” *Statistics in Medicine*, vol. 16, no. 4, pp. 385–395, 1997.

[83] Y. Liang, C. Liu, X.-Z. Luan, K.-S. Leung, T.-M. Chan, Z.-B. Xu, and H. Zhang, “Sparse logistic regression with a L1/2 penalty for gene selection in cancer classification,” *BMC Bioinformatics*, vol. 14, no. 1, pp. 1–12, 2013.

[84] J. Friedman, T. Hastie, and R. Tibshirani, “Regularization paths for generalized linear models via coordinate descent,” *Journal of Statistical Software*, vol. 33, no. 1, p. 1, 2010.

[85] D. Seung and L. Lee, “Algorithms for non-negative matrix factorization,” *Advances in Neural Information Processing Systems*, vol. 13, pp. 556–562, 2001.

[86] D. A. Pisner and D. M. Schnyer, “Support vector machine,” in *Machine Learning*. Elsevier, 2020, pp. 101–121.

[87] X. Zhu, X. Yang, C. Ying, and G. Wang, “A new classification algorithm recommendation method based on link prediction,” *Knowledge-Based Systems*, vol. 159, pp. 171–185, 2018.

[88] K. Järvelin and J. Kekäläinen, “IR evaluation methods for retrieving highly relevant documents,” in *ACM SIGIR Forum*, vol. 51 (2). ACM New York, NY, USA, 2017, pp. 243–250.

[89] R. F. Woolson, “Wilcoxon signed-rank test,” *Wiley Encyclopedia of Clinical Trials*, pp. 1–3, 2007.

[90] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, “Robust recovery of subspace structures by low-rank representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 171–184, 2012.

[91] S. Gandy, B. Recht, and I. Yamada, “Tensor completion and low-n-rank tensor recovery via convex optimization,” *Inverse Problems*, vol. 27, no. 2, p. 025010, 2011.

[92] J. Liu, P. Musialski, P. Wonka, and J. Ye, “Tensor completion for estimating missing values in visual data,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 208–220, 2012.

[93] P. Zhou, C. Lu, Z. Lin, and C. Zhang, “Tensor factorization for low-rank tensor completion,” *IEEE Transactions on Image Processing*, vol. 27, no. 3, pp. 1152–1163, 2017.

[94] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, “Tensor robust principal component analysis: Exact recovery of corrupted low-rank tensors via convex optimization,” in *Pro-*

ceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 5249–5257.

[95] O. A. Malik and S. Becker, "Low-rank tucker decomposition of large tensors using tensorsketch," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[96] J. H. d. M. Goulart, M. Boizard, R. Boyer, G. Favier, and P. Comon, "Tensor CP decomposition with structured factor matrices: Algorithms and performance," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 4, pp. 757–769, 2015.

[97] M. Signoretto, L. De Lathauwer, and J. A. Suykens, "Nuclear norms for tensors and their use for convex multilinear estimation," *Submitted to Linear Algebra and Its Applications*, vol. 43, 2010.

[98] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[99] J. Zhang, H. Peng, K. Wu, M. Liu, B. Xiao, J. Fu, and L. Yuan, "MiniViT: Compressing vision transformers with weight multiplexing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 145–12 154.

[100] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[101] Y. Ji, Q. Wang, X. Li, and J. Liu, "A survey on tensor techniques and applications in machine learning," *IEEE Access*, vol. 7, pp. 162 950–162 990, 2019.

[102] N. Srebro and A. Shraibman, "Rank, trace-norm and max-norm," in *International Conference on Computational Learning Theory*. Springer, 2005, pp. 545–560.

[103] R. Tomioka, T. Suzuki, K. Hayashi, and H. Kashima, "Statistical performance of convex tensor decomposition," *Advances in Neural Information Processing Systems*, vol. 24, pp. 972–980, 2011.

[104] C. Mu, B. Huang, J. Wright, and D. Goldfarb, "Square deal: Lower bounds and improved relaxations for tensor recovery," in *International Conference on Machine Learning*. PMLR, 2014, pp. 73–81.

[105] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, and A. C. de Carvalho, "Characterizing classification datasets: A study of meta-features for meta-learning," *arXiv preprint arXiv:1808.10406*, 2018.

[106] H. Peng, H. Du, H. Yu, Q. Li, J. Liao, and J. Fu, "Cream of the crop: Distilling prioritized paths for one-shot neural architecture search," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 955–17 964, 2020.

[107] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 270–12 280.

[108] M. Chen, K. Wu, B. Ni, H. Peng, B. Liu, J. Fu, H. Chao, and H. Ling, "Searching the search space of Vision Transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8714–8726, 2021.

[109] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[110] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization.* Springer, 2011, pp. 507–523.

[111] R. Leite, P. Brazdil, and J. Vanschoren, "Selecting classification algorithms with active testing," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition.* Springer, 2012, pp. 117–131.

[112] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. 12, pp. 3371–3408, 2010.

[113] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 2006, pp. 1137–1144.

[114] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 847–855.

[115] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren, "Speeding up algorithm selection using average ranking and active testing by introducing runtime," *Machine Learning*, vol. 107, no. 1, pp. 79–108, 2018.

[116] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks*, vol. 2, no. 1, pp. 53–58, 1989.

[117] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: Theory and algorithm," in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 1192–1199.

[118] W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li, "Ranking measures and loss functions in learning to rank," *Advances in Neural Information Processing Systems*, vol. 22, pp. 315–323, 2009.

[119] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[120] I. Borg and P. J. Groenen, *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[121] D. R. Radev, H. Qi, H. Wu, and W. Fan, "Evaluating web-based question answering systems," in *LREC*, 2002, pp. 1153–1156.

APPENDIX A

UCI DATASETS

Table 6.1: Information Summarization of the UCI datasets adopted in the experiments with respect to names, number of instances, attributes, and classes.

ID	Name	#Instances	#Attributes	#Classes
1	abalone	4177	8	3
2	acute-Inflammations	120	7	2
3	audit-risk	776	26	2
4	audit-trial	4177	8	3
5	authentication	1372	4	2
6	autism-adult	704	19	2
7	avila	120	7	2
8	bc_wisconsin	776	26	2
9	bc_wpbc	198	33	2
10	bc-coimbra	116	9	2
11	bc-diagnostic	569	30	2
12	biodegradation	776	26	2
13	blogger	776	17	2
14	blood-transfusion	1372	4	2
15	breast	776	17	2
16	breast-cancer	286	9	2
17	car	1372	4	2
18	cardiotocography	704	19	2
19	cargo	704	19	2
20	cb-sonar	10425	10	11
21	cb-vowel	116	9	2
22	cb-vowel-context	569	30	2
23	chessboard	699	9	2
24	chess-krvkp	3196	36	2
25	climate	10425	10	11
26	CNAE-9	198	33	2
27	contrac	1473	9	3
28	credit-approval	690	15	2
29	creditcard-clients	100	5	2
30	crowdsourced-mapping	10845	28	6
31	cryotherapy	90	6	2
32	dermatology	366	34	6
33	diabetic-retinopathy	748	4	2
34	divorce	569	30	2
35	drug-alcohol	1885	12	7
36	drug-amphetamines	286	9	2
37	drug-cocaine	1882	12	6
38	ecoli2	569	30	2
39	fertility	100	9	2
40	firm-teacher	3942	24	3
41	flags	194	28	6
42	flowmeter-diagnostics	3942	24	3
43	forest-types	523	27	4
44	gamma-telescope	19020	10	2
45	glass	205	9	5
46	grid-stability	10000	13	2
47	haberman-survival	699	9	2
48	happiness-survey2	208	60	2
49	hayes-roth	990	13	11
50	heart	270	13	2
51	hepatitis	528	10	11

Continued on next page

Table 6.1 – continued from previous page

ID	Name	#Instances	#Attributes	#Classes
52	hill-valley	100	2	2
53	hill-valley-noise	1080	856	9
54	horse_colic	198	33	2
55	house_votes	1473	9	3
56	HTRU	17898	8	2
57	ilpd	583	10	2
58	image-segmentation	2310	19	7
59	internet_ad	30000	23	2
60	ionosphere	1055	41	2
61	iris	10845	28	6
62	knowledge-model	90	6	2
63	libras	1151	19	2
64	liver	345	6	2
65	lymphography2	170	54	2
66	madelon	1885	12	7
67	madelon_valid	600	500	2
68	mammographic	961	5	2
69	mb-promoter	1885	12	7
70	mb-splice	1882	12	6
71	mice-protein	327	7	5
72	monks-1	10000	13	2
73	monks-2	100	9	2
74	monks-3	554	6	2
75	mushroom	8124	21	2
76	musk-clean	6598	166	2
77	musk-clean 1	476	166	2
78	nursery	748	4	2
79	occupancy-detection	8143	5	2
80	online-shoppers	12330	17	2
81	optdigits	523	27	4
82	page-blocks	1728	6	4
83	parkinsons	195	22	2
84	parkinson-speech	116	9	2
85	pendigits	1055	41	2
86	phishing-website	540	18	2
87	pima	768	8	2
88	planning-relax	194	28	6
89	post-operation	19020	10	2
90	protein	205	9	5
91	QSAR-androgen-receptor	306	3	2
92	QSAR-bioconcentration	140	6	3
93	qualitative-bankruptcy	132	5	3
94	scale	270	13	2
95	seeds	210	7	3
96	segment	1212	100	2
97	seismic-bumps	286	9	2
98	shuttle-test	14494	9	5
99	spambase	4601	57	2
100	spectf-heart	1728	6	4
101	spect-heart	300	27	2
102	spiral	100	3	2
103	sports-article	1000	59	2
104	st-australian	690	14	2
105	steelplates-faults	1941	27	7
106	st-german-credit	435	16	2
107	st-german-numeric	1000	24	2
108	st-landsat	6435	36	6
109	student-evaluation	5820	32	3
110	student-math	395	33	2
111	student-performance	2126	22	3
112	student-performance-por	3196	36	2
113	st-vehicle	100	5	2

Continued on next page

Table 6.1 – continued from previous page

ID	Name	#Instances	#Attributes	#Classes
114	synthetic-control	10800	16	4
115	TA-evaluations	151	5	3
116	thoracic-surgery	569	30	2
117	tic-tac-toe	1212	100	2
118	urban-land-cover	675	147	9
119	uspst	2007	256	10
120	vehicle	846	18	4
121	vertebral-column-2	17898	8	2
122	vertebral-column-3	583	10	2
123	voice	2126	22	3
124	vowel-context	366	34	6
125	wall-following	5456	24	4
126	waveform	5000	21	3
127	wearable-computing	2310	19	7
128	website-phishing	155	19	2
129	wholesale	3279	1558	2
130	wilt	690	15	2
131	wine	208	60	2
132	wine-quality-red	1599	11	6
133	wine-quality-white	4893	11	6
134	wireless-indoor	351	34	2
135	yeast	150	4	3
136	Z-Alizadeh-sani	303	54	2

APPENDIX B

KAGGLE IMAGE DATASETS

Table 6.2: Information Summarization of the image classification datasets adopted in the experiments with respect to names, the number of (training, validation) instances, and the number of classes.

ID	Name	#Train	#Validation	#Classes
1	AID	8020	990	30
2	Alien vs Predator	694	100	2
3	amazon	2259	279	31
4	Animal 12	13794	1692	12
5	Animal 151	5062	604	151
6	Animal 5	11996	1500	5
7	Animal 90	4320	540	90
8	Animal Faces	13130	1500	3
9	Apparel	9058	1124	24
10	AppleScabLDs	954	118	2
11	architecture	3905	422	25
12	Arts Images	6867	856	5
13	Birds vs Drone	606	110	2
14	BloodCell	7472	2485	4
15	Boat Recognition	1206	142	9
16	Brain Tumor MRI	5143	569	4
17	Breast Ultra Images	583	71	3
18	Butterfly	9285	375	75
19	Caltech101	7400	878	102
20	Car_Brand_Logos	2203	242	8
21	Cats Vs Dogs	19956	2494	2
22	Cats vs Rabbits	1600	414	2
23	Cats-Dogs-Bird	10677	1333	3
24	Chest CT scan	613	72	4
25	Chest X-RAY	5216	16	2
26	Chinese Mnist	12000	1500	15
27	Corn seeds	14243	1779	4
28	COVID19 Chest X-ray	14041	1754	3
29	COVID-19 Lung CT Scans	599	74	2
30	COVIDG	17039	2128	2
31	CUB_200_2011	9478	1171	200
32	Dermnet	14011	1546	23
33	Dogs 70	7946	700	70
34	dslr	420	39	31
35	Elephant Asia vs Africa	756	84	2
36	Emontion Detection	18067	2004	5
37	EndoTect_2020	583	69	11
38	EuroSAT	7305	913	4
39	Eye diseases	572	70	3
40	Facemask Detection	7188	897	3
41	Fake-Real face	1032	128	2
42	FGVC-aricrafts	8000	1000	30
43	Flower Recog	3457	430	5
44	Flowers	10288	1279	13
45	flowers 102	6149	1020	102
46	Food 11	9866	3430	11
47	Forest Fire	4154	455	2
48	Formula One Cars	1927	239	8
49	Fresh vs stale fruits	11748	1467	2
50	Fruits 360	6231	3114	24

Continued on next page

Table 6.2 – continued from previous page

ID	Name	#Train	#validation	#Classes
51	Fruits&Veg	3114	351	36
52	Fundus	827	83	40
53	Garbage 12	12423	1545	12
54	Garbage 5	2025	251	6
55	Gender Classification	21736	2715	2
56	Glaucoma	469	130	2
57	Google SIRI-WHU_earth	1920	240	12
58	Grapevein leaf	400	50	5
59	Grocery Store	2640	296	43
60	Horses	542	64	7
61	Horses or Humans	925	256	2
62	imagenet-a	6244	661	200
63	Indian Food	3828	470	15
64	Intel image	9164	2488	5
65	IP02	45095	7508	102
66	kth_tips_col_200x200	650	80	10
67	Large-scale Fish_Dataset	344	43	9
68	Lung and Colon Cancer	20000	2500	5
69	Lung Cancer	879	109	3
70	Malicious UAV_Dataset	625	75	5
71	Marvel Heros	2329	451	8
72	MASATI-v2	5918	735	7
73	Metal Surface Defects	1656	72	6
74	MIT Indoor Scenes	12527	1531	67
75	Monkeys 10	991	272	10
76	Mushroom	5378	668	9
77	Natural Images	5527	686	8
78	Notre_Dame_dataset	837	397	5
79	OPTIMAL-31	1488	186	31
80	Oxford Pet	5908	738	37
81	pest	2430	270	9
82	Plant Disease Recognition	1322	60	3
83	Pneumothorax	1623	202	2
84	Pumpkin Disease	1147	137	18
85	Red Root Sugarcane	779	85	2
86	Room Clean vs Messy	174	20	2
87	RS_C11_Database	994	119	11
88	RSI-CB256	19803	2472	7
89	SARS-COV	1986	247	2
90	Sign Languange	1662	200	10
91	Skin Cancer ISIC	2020	219	9
92	Snakes	1583	175	2
93	Sports 100	13572	500	100
94	Stanford Dog	16548	2009	120
95	Tire Texture	634	69	2
96	Tsinghua Dog	57161	6988	130
97	UCMerced_LandUse	1680	210	21
98	Veg Images	15000	3000	15
99	Vehicle Detection	14210	1775	2
100	Waste Classification	20309	2255	2
101	Weather Dataset	902	110	4
102	Weather Image	5495	682	11
103	webcam	656	68	31
104	WHU-RS19	813	96	19
105	World Coins	6413	844	211

APPENDIX C

EVALUATION METRICS

Three commonly used metrics in meta-learning are chosen for evaluating the performance of the recommendation approaches, namely, classification accuracy rate (CA), recommendation accuracy rate (RA), and hit rate (HR). In addition, mean reciprocal rank (MRR) and normalized discounted cumulative gain (NDCG), which are widely used in recommender systems, are also adopted in our experiment. The definitions of the five metrics are given below.

Classification accuracy rate (CA): CA is widely used to evaluate the performance of classification algorithm $\mathscr{A}$ instantiated a specific configuration $\boldsymbol{\omega}$ on a dataset $\mathscr{D}$. In this paper, we adopt the balanced CA to handle the possible *imbalance* existing in datasets, so that CA will not be biased by the class sizes. Average CA (ACA) over n datasets is defined as

$$ACA = \frac{1}{n}\sum_{i=1}^{n} CA^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}} \times 100\%,$$

where $CA^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}}$ denotes the CA on the dataset $\mathscr{D}_i$ using recommended configuration $\boldsymbol{\omega}_{Recom}$. The range of both CA and ACA is [0,1], and a higher CA and ACA are preferred, which indicates that the configuration recommended on each dataset appears to be more suitable.

Recommendation accuracy rate (RA): CA is not so effective when all configurations have a similar CA. RA compares the recommended configurations with the truly best and worst configurations as

$$RA = \frac{CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}} - CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{worst}}}}{CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{best}}} - CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{worst}}}}, \tag{6.1}$$

where $CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{worst}}}$ and $CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{best}}}$ are the CA of the worst and the best candidate configurations, respectively. The average RA (ARA) over n datasets is given by

$$ARA = \frac{1}{n}\sum_{i=1}^{n} RA^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}} \times 100\%, \tag{6.2}$$

where $RA^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}}$ denotes the RA on the i^{th} dataset. The ranges of RA and ARA are set to be in [0,1]. A larger RA means the recommended configurations are much closer to the real best configurations.

Hit rate (HR): For a given dataset $\mathscr{D}$ and classifier $\mathscr{A}$, we say a configuration $\boldsymbol{\omega}$ is applicable if the CA of $\mathscr{A}_{\boldsymbol{\omega}}$ falls into the interval

$$\left[CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{best}}} - \alpha \times \sigma_{CA},\ CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}_{best}}}\right], \tag{6.3}$$

where σ_{CA} is the standard deviation of $\{CA^{\mathscr{D}}_{\mathscr{A}_{\boldsymbol{\omega}}} | \boldsymbol{\omega} \in \Omega_{\mathscr{A}}\}$ and α is a positive integer that adjusts the length of this interval (in this paper, we set $\alpha = 1$). If the recommended configurations are applicable, then we call this recommendation hit on dataset $\mathscr{D}$. HR over n datasets is defined as

$$HR = \frac{1}{n}\sum_{i=1}^{n} HR^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}} \times 100\%,$$

where $HR^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}} = 1$ if the recommendation on dataset $\mathscr{D}_i$ is hit, otherwise, $HR^{\mathscr{D}_i}_{\mathscr{A}_{\boldsymbol{\omega}_{Recom}}} = 0$. The range of HR is also [0,1] and a larger HR is preferred.

Mean-reciprocal rank (MRR): To evaluate the ranking of the best underlying hyperparameter in a performance prediction, we adopt MRR metric [121] which is defined as $MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\text{rank}_{\mathscr{D}_i}(\boldsymbol{\omega}_{best})}$

$$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\text{rank}_{\mathscr{D}_i}(\boldsymbol{\omega}_{best})}, \tag{6.4}$$

where $\text{rank}_{\mathscr{D}_i}(\boldsymbol{\omega}_{best})$ refers to the rank of the real best hyperparameter $\boldsymbol{\omega}_{best}$ on the predictive performance of dataset $\mathscr{D}_i$. Since the optimal hyperparameter for datasets usually is not unique, we modify it as

$$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\min\{\text{rank}_{\mathscr{D}_i}(\boldsymbol{\omega}_{best}^{(1)}), \cdots, \text{rank}_{\mathscr{D}_i}(\boldsymbol{\omega}_{best}^{(l)})\}},$$

where we assume that there are l equally optimal hyperparameters for dataset $\mathscr{D}_i$. The range of MRR is given by $(0, 1]$, and we expect that $\text{rank}_{\mathscr{D}}(\boldsymbol{\omega}_{best}) = 1$, so a larger MRR implies that the model is more likely to recommend the real optimal hyperparameters.

Normalized discounted cumulative gain (NDCG): Take the predictive performance vector $\mathbf{x}_i^{pred}$ as the scores the recommender system assigned to the list of items for a testing problem (query) $\mathscr{D}_i$, then the NDCG at a particular rank position ρ $(0 < \rho \leq m)$ on this problem (query) can be defined as

$$NDCG_{\mathscr{D}_i}^{\rho} = \frac{DCG_{\mathscr{D}_i}^{\rho}}{IDCG_{\mathscr{D}_i}^{\rho}},$$

where $DCG_{\mathscr{D}_i}^{\rho} = \sum_{j=1}^{\rho}\frac{2^{\pi[(x_i^{pred})_j]}-1}{\log_2(i+1)}$ is the discounted cumulative gain obtained from $\mathbf{x}_{\mathscr{D}_i}^{pred}$ and $IDCG_{\mathscr{D}_i}^{\rho} = \sum_{j=1}^{\rho}\frac{2^{\pi[(x_i)_j]}-1}{\log_2(i+1)}$ is the real discounted cumulative gain obtained from $\mathbf{x}_{\mathscr{D}_i}$ on problem $\mathscr{D}_i$; $\pi[(x_i^{pred})_j]$ $(\pi[(x_i)_j])$ indicates the rank of the j^{th} entry of $\mathbf{x}_{\mathscr{D}_i}^{pred}$ $(\mathbf{x}_{\mathscr{D}_i})$. The range of $NDCG_{\mathscr{D}_i}^{\rho}$ is $(0, 1]$ and a larger value implies the first ρ recommended configurations being better matching the real best top-ρ configurations. The NDCG over N testing problems is given by

$$NDCG@\rho = \frac{1}{N}\sum_{i=1}^{N} NDCG_{\mathscr{D}_i}^{\rho}.$$

Again, NDCG@$\rho \in (0, 1]$ and a larger NDCG@ρ is preferred.

APPENDIX D

AUTOMATIC META-FEATURE SELECTION

Instability analysis of equation (2.7): First, we show that equation (2.16) is equivalent to equation (2.7) when $a = \frac{1}{\|\mathbf{h}_{i\star}\|_2^2}$. Notice that $\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t) = -\mathbf{S}\mathbf{h}_{i\star}^\top + \boldsymbol{\beta}_i^t \|\mathbf{h}_{i\star}\|_2^2 + \sum_{j\neq i}^{r_k} \boldsymbol{\beta}_j \|\mathbf{h}_{j\star}\|_2^2$ and $\tilde{\boldsymbol{\beta}}_i^t = \boldsymbol{\beta}_i^t - a\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)$, then

$$\begin{aligned}
\textbf{(2.16)} \iff \boldsymbol{\beta}_i &= \tilde{\boldsymbol{\beta}}_i^t \left(1 + \frac{a\lambda}{\|\boldsymbol{\beta}_i\|_2}\right)^{-1} \\
&= \left[\boldsymbol{\beta}_i^t - a\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)\right] \left(1 + \frac{a\lambda}{\|\boldsymbol{\beta}_i\|_2}\right)^{-1} \\
&= \left[\boldsymbol{\beta}_i^t - \frac{1}{\|\mathbf{h}_{i\star}\|_2^2}\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)\right] \left(1 + \frac{\lambda}{\|\mathbf{h}_{i\star}\|_2^2\|\boldsymbol{\beta}_i\|_2}\right)^{-1} \\
&= \left(\mathbf{S}\mathbf{h}_{i\star}^\top - \sum_{j\neq i}^{r_k} \boldsymbol{\beta}_j \|\mathbf{h}_{j\star}\|_2^2\right) \left(\|\mathbf{h}_{i\star}\|_2^2 + \frac{\lambda}{\|\boldsymbol{\beta}_i\|_2}\right)^{-1} \iff \textbf{(2.7)}.
\end{aligned}$$

Therefore, we can rewrite equation (2.7) as equation (2.18) where $a = \frac{1}{\|\mathbf{h}_{i\star}\|_2^2}$, and obtain

$$\begin{aligned}
\boldsymbol{\beta}_i &= \tilde{\boldsymbol{\beta}}_i^t \left(1 - \frac{a\lambda}{\|\tilde{\boldsymbol{\beta}}_i^t\|_2}\right)_+ \\
&= [\boldsymbol{\beta}_i^t - a\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)] \left(1 - \frac{a\lambda}{\|\boldsymbol{\beta}_i^t - a\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)\|_2}\right)_+ \\
&= \frac{\mathbf{c}}{\|\mathbf{h}_{i\star}\|_2^2} \left(1 - \frac{\lambda}{\|\mathbf{c}\|_2}\right)_+,
\end{aligned}$$

where $\mathbf{c} = \mathbf{S}\mathbf{h}_{i\star}^\top - \sum_{j\neq i}^{r_k} \boldsymbol{\beta}_j \|\mathbf{h}_{j\star}\|_2^2$ that is free of $\|\mathbf{h}_{i\star}\|_2$. As one can see, when $\|\mathbf{h}_{i\star}\|_2 \to 0$, $\frac{\mathbf{c}}{\|\mathbf{h}_{i\star}\|_2^2} \to \infty$. Thus, we have if $(1 - \frac{\lambda}{\|\mathbf{c}\|_2})_+ > 0$ then $\boldsymbol{\beta}_i \to \infty$. Obviously, this situation will lead to unstable and unsatisfying simulations.

Proof of Theorem 2.3.1: To prove

$$l(\mathbf{S}, \mathbf{B}) \leq l(\mathbf{S}, \mathbf{B}^t) + tr((\mathbf{B} - \mathbf{B}^t)^\top \nabla l(\mathbf{S}, \mathbf{B}^t)) + \frac{1}{2a}\|\mathbf{B} - \mathbf{B}^t\|_F^2, \tag{6.5}$$

we consider its subproblem, namely, fixing all columns of $\mathbf{B}$ except $\boldsymbol{\beta}_j$,

$$l(\mathbf{S}, \boldsymbol{\beta}_j) \leq l(\mathbf{S}, \boldsymbol{\beta}_j^t) + (\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t)^\top \nabla l(\mathbf{S}, \boldsymbol{\beta}_j^t) + \frac{1}{2a}\|\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t\|_2^2. \tag{6.6}$$

As a quadratic function, $l(\mathbf{S}, \boldsymbol{\beta}_j)$ can be expressed at point $\boldsymbol{\beta}_j^t$ as

$$l(\mathbf{S}, \boldsymbol{\beta}_j) = l(\mathbf{S}, \boldsymbol{\beta}_j^t) + (\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t)^\top \nabla l(\mathbf{S}, \boldsymbol{\beta}_j^t) + \frac{1}{2}(\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t)^\top [\nabla^2 l(\mathbf{S}, \boldsymbol{\beta}_j)](\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t), \tag{6.7}$$

where $\nabla^2 l(\mathbf{S}, \boldsymbol{\beta}_j) = \|\mathbf{h}_{j\star}\|_2^2\mathbf{I}$ is the second-order derivative of l with respect to $\boldsymbol{\beta}_j$. Then, it is equivalent to showing that

$$\frac{1}{2}(\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t)^\top (\|\mathbf{h}_{j\star}\|_2^2\mathbf{I})(\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t) \leq \frac{1}{2a}\|\boldsymbol{\beta}_j - \boldsymbol{\beta}_j^t\|_2^2, \tag{6.8}$$

which requires $0 < a \leq \frac{1}{\|\mathbf{h}_{j\star}\|_2^2}$. Now take the all columns of $\mathbf{B}$ into account, one can have

$$0 < a \leq \min\{\frac{1}{\|\mathbf{h}_{1\star}\|_2^2}, \frac{1}{\|\mathbf{h}_{2\star}\|_2^2}, \cdots, \frac{1}{\|\mathbf{h}_{r_k\star}\|_2^2}\}.$$

Proof of Proposition 1: We prove Proposition 1 by contradiction. Suppose that $\boldsymbol{\beta}^{(1)}$ and $\boldsymbol{\beta}^{(2)}$ are two different solutions for

$$\boldsymbol{\beta}_i = \tilde{\boldsymbol{\beta}}_i^t \left(1 + \frac{a\lambda}{\|\boldsymbol{\beta}_i\|_2}\right)^{-1}.$$

Without loss of generality, let $\|\boldsymbol{\beta}^{(1)}\|_2 = 1$. Then, we have

$$\begin{aligned} \boldsymbol{\beta}^{(1)} &= \tilde{\boldsymbol{\beta}}_i^t(1 + a\lambda)^{-1} && (6.9) \\ \boldsymbol{\beta}^{(2)} &= \tilde{\boldsymbol{\beta}}_i^t \left(1 + \frac{a\lambda}{\|\boldsymbol{\beta}^{(2)}\|_2}\right)^{-1} && (6.10) \end{aligned}$$

Notice that $\tilde{\boldsymbol{\beta}}_i^t = \boldsymbol{\beta}_i^t - a\nabla l(\mathbf{S}, \boldsymbol{\beta}_i^t)$ is free of $\boldsymbol{\beta}_i$. Now we consider two different occasions:

1. $\|\boldsymbol{\beta}^{(2)}\|_2 = 1$. Then it is quite straightforward to see that $\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(2)}$. A contradiction.

2. $\|\boldsymbol{\beta}^{(2)}\|_2 \neq 1$. From (6.9), one can have $\tilde{\boldsymbol{\beta}}_i^t = (1 + a\lambda)\boldsymbol{\beta}^{(1)}$, and plug it into (6.10),

$$\boldsymbol{\beta}^{(2)} = (1 + a\lambda)\boldsymbol{\beta}^{(1)} \left(1 + \frac{a\lambda}{\|\boldsymbol{\beta}^{(2)}\|_2}\right)^{-1} = \boldsymbol{\beta}^{(1)} \frac{1 + a\lambda}{1 + \frac{a\lambda}{\|\boldsymbol{\beta}^{(2)}\|_2}}. \tag{6.11}$$

   Taking 2-norm on both sides of (6.11), one arrives at

$$\|\boldsymbol{\beta}^{(2)}\|_2 = \frac{1 + a\lambda}{1 + \frac{a\lambda}{\|\boldsymbol{\beta}^{(2)}\|_2}} \Rightarrow \|\boldsymbol{\beta}^{(2)}\|_2 + a\lambda = 1 + a\lambda \Rightarrow \|\boldsymbol{\beta}^{(2)}\|_2 = 1,$$

   which contradicts the assumption of $\|\boldsymbol{\beta}^{(2)}\|_2 \neq 1$.

The proof of Proposition 1 is completed.

APPENDIX E

NONLINEAR META-FEATURE SELECTION

Proof of Theorem 2: To prove

$$G(\mathbf{b}_j, \mathbf{b}_j^t) = L(\mathbf{b}_j^t) + (\mathbf{b}_j - \mathbf{b}_j^t)^\top \nabla L(\mathbf{b}_j^t) + \frac{1}{2a_j}\|\mathbf{b}_j - \mathbf{b}_j^t\|_2^2 + \lambda\|\mathbf{b}_j\|_2, \tag{6.12}$$

with $L(\mathbf{b}_j) = \frac{1}{2}\|\mathbf{H} - \mathbf{BK}\|_F^2$ is the auxiliary function of

$$\hat{F}(\mathbf{b}_j) = \frac{1}{2}\|\mathbf{H} - \mathbf{BK}\|_F^2 + \lambda\|\mathbf{b}_j\|_2 = L(\mathbf{b}_j) + \lambda\|\mathbf{b}_j\|_2, \tag{6.13}$$

we first need to show that $\hat{F}(\mathbf{b}_j) \leq G(\mathbf{b}_j, \mathbf{b}_j^t)$. It is easy to verify that the second-order gradient of L with respect to $\mathbf{b}_j$ is given by $\nabla L^2(\mathbf{b}_j^t) = \|\mathbf{k}_{j\star}\|_2^2\,\mathbf{I}$. In fact, a direct calculation yields

$$\nabla L(\mathbf{b}_j^t) = (-\mathbf{HK}^\top + \mathbf{B}^t\mathbf{KK}^\top)_j = -\mathbf{Hk}_{j\star}^\top + [\mathbf{b}_j^t(\mathbf{Kk}_{j\star}^\top)_j + \sum_{i\neq j}^{n} \mathbf{b}_i^t(\mathbf{Kk}_{j\star}^\top)_i],$$

which leads to

$$\nabla L^2(\mathbf{b}_j^t) = (\mathbf{Kk}_{j\star}^\top)_j\,\mathbf{I} = \|\mathbf{k}_{j\star}\|_2^2\,\mathbf{I}.$$

We expand $L(\mathbf{b}_j)$ in a Taylor series at the point $\mathbf{b}_j^t$ as

$$L(\mathbf{b}_j) = L(\mathbf{b}_j^t) + (\mathbf{b}_j - \mathbf{b}_j^t)^\top \nabla L(\mathbf{b}_j^t) + \frac{1}{2}(\mathbf{b}_j - \mathbf{b}_j^t)^\top \nabla L^2(\mathbf{b}_j^t)(\mathbf{b}_j - \mathbf{b}_j^t), \tag{6.14}$$

Then it is equivalent to showing that

$$\frac{1}{2}(\mathbf{b}_j - \mathbf{b}_j^t)^\top(\|\mathbf{k}_{j\star}\|_2^2\,\mathbf{I})(\mathbf{b}_j - \mathbf{b}_j^t) \leq \frac{1}{2a_j}\|\mathbf{b}_j - \mathbf{b}_j^t\|_2^2,$$

when the following condition

$$0 < a_j \leq \frac{1}{\|\mathbf{k}_{j\star}\|_2^2},$$

is met. The proof of $\hat{F}(\mathbf{b}_j^t) = G(\mathbf{b}_j^t, \mathbf{b}_j^t)$ is straightforward.

Proof of Theorem 3: By Theorem 1, we have showed that $G(\mathbf{b}_j, \mathbf{b}_j^t)$ is the auxiliary function of $\hat{F}(\mathbf{b}_j)$. For given $\mathbf{b}_j^{t+1} = \arg\min_{\mathbf{b}_j} G(\mathbf{b}_j, \mathbf{b}_j^t)$, we have

$$\hat{F}(\mathbf{b}^{t+1}) \leq G(\mathbf{b}_j^{t+1}, \mathbf{b}_j^t) \leq G(\mathbf{b}_j^t, \mathbf{b}_j^t) = \hat{F}(\mathbf{b}^t),$$

which implies that $\hat{F}(\mathbf{b})$ is nonincreasing under the updating rule

$$\mathbf{b}_j = \tilde{\mathbf{b}}_j^t \left(1 - \frac{a_j \lambda}{\|\tilde{\mathbf{b}}_j^t\|_2}\right)_+ .$$

Now we can generalize the proof to all columns of $\mathbf{W}_j$, which leads to the conclusion.

APPENDIX F

LATENT FEATURE LEARNING

Table 6.3: The deployed hyperparameter settings for the candidate classifiers with respect to types, values, and steps.

Classifiers	Hyperparameters	Types	Values	Steps
AdaBoost	`learning_rate`	numerical	`0.1:0.1:1`	10
	`n_estimators`	numerical	{`10,50,100,200,300,400,500`}	7
Bagging	`bootstrap`	binary	{'`True`', '`False`'}	2
	`bootstrap_fea`	binary	{'`True`', '`False`'}	2
	`max_samples`	numerical	`0.1:0.1:0.9`	9
	`n_estimators`	integer	`100:100:1000`	10
Decision tree	`split`	categorical	{'`gini`', '`entropy`'}	2
	`min_samples_leaf`	integer	`1:1:20`	20
	`min_samples_split`	integer	`2:1:20`	19
Extra tree	`split`	categorical	{'`gini`', '`entropy`'}	2
	`min_samples_leaf`	integer	`1:1:20`	20
	`min_samples_split`	integer	`2:1:20`	19
KNN	`weight`	categorical	{'`uniform`', '`distance`'}	2
	`metric`	categorical	{'`euclidean`','`manhattan`','`chebyshev`'}	3
	`n_neighbors`	integer	`1:1:30`	30
LDA	`tolerance`	numerical	{`1e-5,1e-4,1e-3,1e-2,1e-1`}	5
	`shrinkage_rate`	numerical	`0:0.1:1`	11
LR	`solver`	categorical	{'`liblinear`', '`saga`'}	2
	`C`	numerical	`0.01:0.01:1,1:1:100`	199
	`penalty`	categorical	{'`l1`', '`l2`'}	2
MultiP	`solver`	categorical	{'`sgd`', '`adm`'}	2
	`activation`	categorical	{'`identity`','`logistic`','`tanh`','`relu`'}	4
	`alpha`	numerical	{`1e-4,1e-3,1e-2`}	3
	`learning_rate`	numerical	{`1e-4,1e-3,1e-2`}	3
QDA	`reg_param`	numerical	`0:0.1:1`	11
RF	`split`	categorical	{'`gini`','`entropy`'}	2
	`bootstrap`	binary	{'`True`','`False`'}	2
	`min_samples_leaf`	integer	`1:1:20`	20
	`min_samples_split`	integer	`2:1:20`	19
SVM	`C`	numerical	$2^{-5:1:15}$	21
	`gamma`	numerical	$2^{-15:1:3}$	19

VITA

Graduate School
Southern Illinois University Carbondale

Liping Deng

liping.deng@siu.edu

Shenzhen University, Shenzhen, Guangdong Province, China
Master of Science, Mathematics, June 2019

Anshan Normal University, Anshan, Liaoning Province, China
Bachelor of Science, Mathematics, June 2016

Special Honors and Awards:
Nomination for Excellent Teaching Assistant Award, 2022

Dissertation Title:
A Research on Automatic Hyperparameter Recommendation via Meta-learning

Major Professor: Dr. MingQing, Xiao

Publications:

Deng L., Xiao M.Q., "A New Automatic Hyperparameter Recommendation Approach under Low-Rank Tensor Completion Framework", *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), vol. 45, no. 4, pp. 4038-4050.

Deng L., Xiao, M.Q., "Latent Feature Learning via Autoencoder Training for Automatic Classification Configuration Recommendation", *Knowledge-Based Systems* (2023), vol. 261, pp. 110218

Deng L., Chen W.-S., Xiao, M.Q., "Meta-feature Selection via A Novel Multivariate Sparse-Group Lasso Method for Automatic Hyperparameter Configuration Recommendation", in *IEEE Transactions on Neural Networks and Learning System*, 2023. **(In Press)**