Published: 07 August 2025
Online AM: 26 June 2025
Accepted: 12 June 2025
Revised: 13 April 2025
Received: 06 October 2024

RESEARCH-ARTICLE

Towards Recommendation on Good Quality Data Science Solutions

JIAN CHEN, South China University of Technology, Guangzhou, Guangdong, China

YILE CHEN, South China University of Technology, Guangzhou, Guangdong, China

ZEYI WEN, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong, China

YAWEN CHEN, Henan University, Kaifeng, Henan, China

JIN HUANG, South China Normal University, Guangzhou, Guangdong, China

Open Access Support provided by:

South China University of Technology

The Hong Kong University of Science and Technology (Guangzhou)

Henan University

South China Normal University

Towards Recommendation on Good Quality Data Science Solutions

JIAN CHEN and YILE CHEN, South China University of Technology, Guangzhou, China
ZEYI WEN, Data Science and Analytics Thrust, Information Hub, Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China
YAWEN CHEN, School of Artificial Intelligence, Henan University, Zhengzhou, China
JIN HUANG, South China Normal University, Guangzhou, China

Data science aims to solve real-world problems with the knowledge derived from data. Successfully tackling a data science problem requires practitioners to choose an appropriate solution, which potentially comprises various components such as pre-processing techniques, learning algorithms, hyper-parameters, and so on. Therefore, a problem-driven recommendation for the promising solution is invaluable, as it facilitates efficient and convenient problem-solving. However, existing solution recommendation approaches confront notable challenges when dealing with limited and sparse prior experience in practical applications. Learning from such prior easily leads to overfitting and poor generalization in solution recommendations. To address this issue, we propose a novel solution recommendation method that can predict a good-quality data science solution, including the pre-processing, the learning algorithm, and hyper-parameters, for a given problem. The foundation of our method is a carefully designed ranking model that exploits a weight-sharing structure and a newly proposed loss. The ranking model focuses on incorporating relative ranking information into the predicted performance score of each solution. With these techniques, our method can recommend the solution with the highest score and effectively mitigate the limitations of using sparse prior experience. Our experiments demonstrate the superiority of our method in predicting solutions with higher accuracy and rank, even trained on highly sparse historical performance records. It also reduces recommendation time significantly compared to the baselines, offering remarkable efficiency and convenience for practitioners.

CCS Concepts: • **Computing methodologies** → **Search methodologies**; *Machine learning*;

Additional Key Words and Phrases: Data science, Solution recommendation, Sparse prior

Associate Editor: Xiaohui Yu

ACM Reference format:
Jian Chen, Yile Chen, Zeyi Wen, Yawen Chen, and Jin Huang. 2025. Towards Recommendation on Good Quality Data Science Solutions. *ACM Trans. Knowl. Discov. Data.* 19, 7, Article 133 (August 2025), 19 pages.
https://doi.org/10.1145/3746235

This work was supported by the National Natural Science Foundation of China (Grant No. 62376099), the Natural Science Foundation of Guangdong Province (Grant No. 2024A1515010989), the China Postdoctoral Science Foundation (Grant No. 2024M760785), and the Key R&D and Promotion Projects of Henan Province (Grant No. 252102210047).
Authors' Contact Information: Jian Chen, South China University of Technology, Guangzhou, China; e-mail: ellachen@scut.edu.cn; Yile Chen, South China University of Technology, Guangzhou, China; e-mail: jireh.x6@gmail.com; Zeyi Wen, Data Science and Analytics Thrust, Information Hub, Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China; e-mail: wenzeyi@ust.hk; Yawen Chen (corresponding author), School of Artificial Intelligence, Henan University, Zhengzhou, China; e-mail: ywchen@henu.edu.cn; Jin Huang, South China Normal University, Guangzhou, China; e-mail: huangjin@m.scnu.edu.cn.

ACM 1556-472X/2025/8-ART133
https://doi.org/10.1145/3746235

1 Introduction

Data science focuses on extracting knowledge from data and applying the knowledge to tackle real-world problems [38]. The problem-solving process in data science typically involves several key components, for instance, data pre-processing, algorithm learning, hyper-parameters tuning, performance evaluation, and so on, as illustrated in Figure 1. While selecting appropriate individual components, such as algorithm selection [30] or hyper-parameter configuration [12, 30], is crucial to effectively address data science problems, there is a growing demand for recommendations on a more general solution that considers multiple components within the process holistically. Problem-driven recommendations for high-quality solutions can help practitioners solve problems in a more streamlined and efficient manner [17, 21].

Many existing solution recommendation approaches strive to establish a mapping from the data science problem to the solution that achieved the best performance or a mapping from the solution to the exact performance score (e.g., accuracy) [4, 19, 30]. The effectiveness of these methods highly relies on the quality and accessibility of the prior experience as the mapping model is trained using the historical solutions and their associated performance records obtained on the historical datasets. Moreover, most of these methods require that each historical solution must have been evaluated on every available historical dataset. With this requirement, a performance matrix can be formed as in Figure 2(a), where each entry is assigned the performance record achieved by training the corresponding solution on that dataset. We refer to such performance records as dense historical performance records. Obtaining dense records can be challenging due to various factors, such as limited computation resources, time constraints, or the unavailability of certain solutions for evaluation on some datasets. Therefore, in practice, it is more common for only a subset of the historical solutions to have been evaluated on the historical datasets. As these performance records cannot fill in the whole matrix as in Figure 2(b), we regard them as sparse historical performance records. Unfortunately, training with limited and sparse historical performance records can restrict the method's capacity to learn from prior experience, leading to incomplete knowledge. This increases the risk of overfitting and diminishes the adaptability to future problems. Some researchers attempt to enhance the recommendation by further optimizing or fine-tuning the recommended solutions on the encountered problem [17, 21], which may cause overfitting in few-shot tasks.

To overcome the limitations of using sparse historical performance records for training and provide accurate recommendations on data science solutions for unseen problems, we propose a novel solution recommendation method. The data science solution considered in this work consists of the data, pre-processing, the learning algorithm, and hyper-parameters. The key idea of our method is to learn the ranking relationships of solutions using their representation with meta-features. We design a ranking model which is equipped with two weight-sharing **Performance Measurement Machines (PMMs)** and a balancing mechanism. We propose a modified contrastive loss for our ranking model which constrains PMMs to learn scores that capture the ranking relationship of solutions, instead of learning the precise performance score. The well-trained PMM then predicts scores for the potential solutions, and the solution with the highest score is selected as the most promising solution to a given data science problem. By incorporating these techniques, our method can avoid overfitting to the small number of sparse historical records and enhance its adaptability to address future problems.

In our experiment, we compare our method with the existing solution recommendation methods using the historical performance records of various sparsity levels. The experimental results reveal that our method consistently outperforms the baselines by recommending the solution with higher accuracy and rank, even using the highly sparse historical records for training. We also conduct

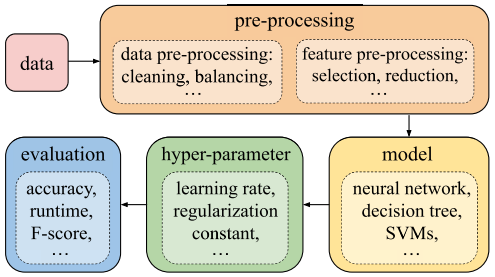
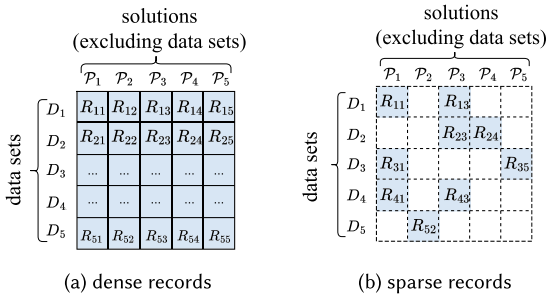


Fig. 1. Key processes of solving data science problems.



(a) dense records (b) sparse records

Fig. 2. Examples of performance matrices.

a case study to further fine-tune the recommended solution on the given problem and observe the performance of the tuned solution. The results affirm that our method can provide a superior choice of solution while reducing the recommendation time by one to four orders of magnitude compared with the baselines. In summary, we make the following major contributions:

- —We propose a novel method to recommend the data science solution which consists of the pre-processing techniques and the learning algorithm with hyper-parameters while considering the dataset itself.
- —Our method is built upon a ranking model which contains two weight-sharing PMMs with a newly defined contrastive loss. Benefiting from the techniques designed for sparse training records, our method is able to mitigate the negative influence of sparse historical performance records on solution recommendation.
- —We conduct comprehensive experiments to evaluate our solution recommendation method. The results verify the higher accuracy and rank of the solution recommended by our method as well as the better generality of our method over the existing methods.

2 Related Work

Recommendation systems have evolved significantly from their traditional forms to modern applications such as data science recommendations. Classical approaches include collaborative filtering [54] which exploits user–item interaction patterns, content-based methods [37] that rely on item similarities, and more recently, deep learning-based recommenders [57] which model complex user preferences. These techniques have demonstrated remarkable success in various domains such as product recommendations in e-commerce [29], personalized course selection in e-learning [52], and medical treatment tailor in e-health [5]. However, recommending data science solutions presents unique challenges that distinguish it from these traditional applications. Unlike conventional recommendation targets, data science solutions are complex compositions of structural and unstructured components, such as algorithms, hyper-parameters, and data pre-processing techniques [15]. The recommendation not only relies on the problems but also on the interaction of components inside the solutions. Traditional approaches struggle to capture the hierarchical dependencies and combinatorial effects inherent in data science solutions [45]. In addition, the nonlinear dependence of solution performance on dataset variations [36] renders data similarity-based recommendation approaches ineffective. This limitation is further compounded by the intrinsic difficulty in quantifying similarities of solution components such as similarities of different algorithms. These factors impede the direct application of conventional similarity-driven recommendation frameworks to data science solution recommendation tasks [17, 21].

To address these distinctive requirements, the data science community has developed specialized methods based on the traditional recommender systems for identifying promising solutions or their components. Various approaches such as algorithm selection [12], hyper-parameter optimization [35], neural architecture search [58], and so on have been explored. These strategies for solution recommendation can be broadly categorized into two types: inference-based recommendation and optimization-based recommendation.

2.1 Inference-Based Solution Recommendation

Inference-based recommendation operates analogously to model-based collaborative filtering [1], which leverages insights gained from previous experience, domain expertise, or heuristics to guide the decision-making process. It analyzes the characteristics of problems as well as solutions and recommends the solution (or its components) that is likely to perform well. For instance, various classifiers such as k-NN [20, 43], decision trees [44], **Random Forests (RFs)** [32], and **Support Vector Machines (SVMs)** [46] have been explored in solution recommendation. In these approaches, existing datasets are used as inputs to the classifier, where the label of each input represents the solution (or component) that achieved the best performance among all historical solutions on that problem. Classification models are trained to classify the problem into the class that indicates the best solution (or component). While classification models can predict the best solution, regression models are used to predict the performance of a solution. Inference-based recommendation methods employ regression models such as deep neural networks [39], decision trees [28], RFs [13], and **AdaBoost (AdaB)** [14] to model the performance of a solution or component. Common performance metrics used include predictive accuracy [23] and runtime [40, 55]. A multi-criteria metric **Adjusted Ratio of Ratios (ARR)** [30] and extended ARR [53] use the combination of accuracy and execution time to measure performance. Brazdil et al. [9] used the results of cross-validation as verification metrics. Some studies [3, 6] integrated expert knowledge into the evaluation metrics. The solution or components recommended by inference-based methods can also serve as the warm start in optimization-based recommendation.

Inference-based recommendation learns patterns from previous experience and directly predicts the promising solution to a new problem. These methods do not need to further tune the solution to the new problem. However, they can only predict solutions that have been run on previous problems and may not consider other solutions with potential good performance for new problems. Besides, most studies of inference-based recommendation focus on recommending specific components in the solution (e.g., algorithm selection [30]) and require dense historical performance records [19] to improve recommendation, which can be expensive to acquire.

2.2 Optimization-Based Solution Recommendation

Optimization-based recommendation utilizes search and optimization algorithms to find the best solution components automatically. In contrast to inference-based solution recommendation, optimization-based recommendation requires evaluating solutions on the encountered problem before making predictions. It involves training multiple solutions on the problem at hand and using the training results to guide the next round of solution search [27, 56]. **Auto-sklearn (AutoSK)** [17, 19] incorporates meta-learning and ensemble components into **Bayesian Optimization (BO)** [49]. It chooses a highly specialized BO variant known as Sequential Model-Based Algorithm Configuration [26] to enhance solution search. Fusi et al. [21] proposed a **Probabilistic Matrix Factorization (PMF)** approach for solution recommendation. PMF uses non-linear matrix factorization [34], inspired by collaborative filtering, to generate latent embeddings of solutions performed on different datasets. Then, it adopts BO with a customized acquisition function to iteratively explore possible solutions. Additionally, PMF initializes BO by selecting top-performing

solutions from the k-nearest datasets [18], which is similar to user-based collaborative filtering. Recently, researchers have further investigated deep learning techniques to better extract solution features [15, 45] as extracting user or item characteristics in conventional recommender systems. Arango and Grabocka [45] introduced **DeepPipe (DP)** which first exploits an encoder network to generate a latent embedding for each solution component. Then, DP feeds the embeddings into BO with a deep-kernel Gaussian process to search promising solutions. Drori et al. [15] choose a recurrent neural network, while Rakotoarison et al. [47] use a surrogate model as the estimators of solution performance, respectively. Based on the estimated performance, they use Monte Carlo Tree Search [31] to generate potential solutions. The performance evaluation of these solutions on the new task is back propagated to improve the estimator. Some studies [22, 41] use genetic algorithms to search algorithm configurations.

Optimization-based recommendation proceeds with iterative evaluations on the encountered dataset, which may lead to poor robustness when practitioners only have a few shots for their tasks. Moreover, evaluations on the new dataset add extra computational burden and can be time-consuming.

3 Recommendation on Data Science Solution

Our work aims to recommend the most promising data science solution for a given dataset, and there are three key challenges in solution recommendation. First, one of the main challenges is the limited number of historical performance records available for training the recommendation method. This limitation can hinder the ability of the method to learn the complex relationship between a solution and its performance, as well as its capacity to adapt to new datasets. Second, the search space in solution recommendation is influenced by prior experience. Sparse historical performance records can lead to a smaller search space, which, in turn, reduces the possibility of discovering superior solutions. Third, the dominance of certain types of features, such as data property features, over other types of features like algorithm features, can impact the predictive accuracy of the recommendation method. To tackle these challenges, we propose an effective solution recommendation method that leverages the sparse historical experience.

3.1 Overview of Our Proposed Solution Recommendation Method

Before delving into the details, we present an overview of our method as illustrated in Figure 3. Our method begins by generating a representation for each solution using a set of carefully extracted features. Then, it feeds pairs of solution into a ranking model. The ranking model is composed of two weight-sharing PMMs, and each PMM can learn a performance score for a given solution. The new contrastive loss proposed for our ranking model can guide the performance scores to memorize ranking information. After the training of the ranking model, the optimized PMM predicts performance scores for a set of candidate solutions. The solution with the highest predicted score is then recommended to the practitioner as the most promising data science solution to the given dataset. The carefully designed feature extraction, ranking model, and contrastive loss all contribute to enhancing the accuracy and reliability of the solution recommendation. In the following sections, we elaborate the specific techniques adopted in our method.

3.2 Solution Representation

To recommend data science solutions, it is paramount to represent the solution in a format that our method can effectively utilize. In our approach, we construct the solution representation as depicted on the left side of Figure 3. The solution considered in this work comprises four main components: the given dataset, the pre-processing methods, the learning algorithm, and the hyper-parameters

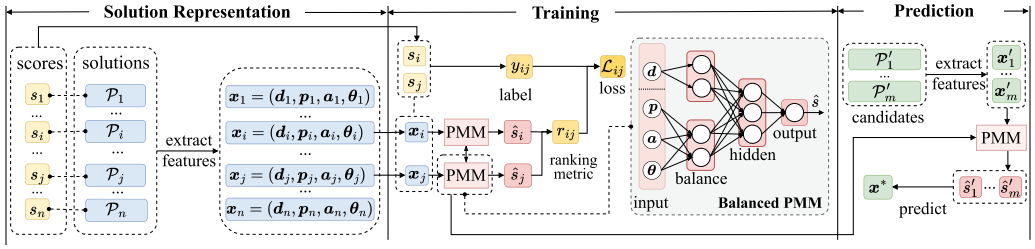


Fig. 3. Workflow of solution recommendation using our method.

utilized in the solution. We exploit meta-learning techniques to extract meta-features from each component, which enables us to represent the solution using these extracted features.

For the dataset component, we extract the following five groups of features: (i) Basic information about the problem, such as the total number of attributes and the number of classes; (ii) statistical features including the canonical correlations of the data, the geometric mean of each attribute; (iii) information theory-related measures, such as Shannon's entropy for each predictive attribute and the concentration coefficient of each pair of distinct attributes; (iv) landmarking measures such as the best performance of the decision tree trained on the data; (v) model-based measures, for example, the number of leaf nodes in the decision tree trained on the data. Combining these five groups of features forms the representation vector for the given dataset, where the vector is denoted as $\boldsymbol{d} \in \mathbb{R}^{n_d}$ and n_d is the total number of meta-features for the dataset.

In addition to the dataset representation, we also consider the representation of pre-processing methods. Since most applications involve pre-processing the data to boost learning, we explore the usage of pre-processing methods as features. We encode the pre-processing methods using a vector $\boldsymbol{p} \in \mathbb{R}^{n_p}$, where each dimension corresponds to a specific pre-processing method. In the vector, the dimensions corresponding to pre-processing methods used in the current solution are marked with ones, while the remaining dimensions which indicate other available pre-processing methods are filled with zeros.

To represent the learning algorithm, we consider not only the algorithm itself but also its category and implementation scheme. Formally, we use three one-hot vectors: $\boldsymbol{a}^{(alg)}$, $\boldsymbol{a}^{(cate)}$, and $\boldsymbol{a}^{(imp)}$, to indicate the algorithm, algorithm category, and implementation scheme, respectively. The dimension of $\boldsymbol{a}^{(alg)}$ is equal to the total number of distinct algorithms and $a_i^{(alg)}$ equals 1 when the ith algorithm is used. Each dimension in vector $\boldsymbol{a}^{(cate)}$ refers to an algorithm category, and the category that algorithm A belongs to is labeled with 1. We expand the range of categories mentioned in AutoSK [18] and include the following categories: (i) general linear models; (ii) SVMs; (iii) discriminant analysis; (iv) nearest neighbors; (v) naive Bayes; (vi) decision trees, and (vii) ensemble including vote, bagging, boosting, and forest; (viii) neural networks. As for the implementation schemes, each element in vector $\boldsymbol{a}^{(imp)}$ indicates a library in which the algorithm is implemented. The libraries considered cover multiple popular machine learning libraries such as Weka [25], AutoSK [18], Machine Learning in R [8], Massive Online Analysis [7], and so on. The three one-hot vectors are concatenated as the representation of the learning algorithm, which is defined as $\boldsymbol{a} = [\boldsymbol{a}^{(alg)}, \boldsymbol{a}^{(cate)}, \boldsymbol{a}^{(imp)}]$. The resulting vector $\boldsymbol{a}$ is an n_a dimensional vector, where n_a is equal to the sum of the dimensions of $\boldsymbol{a}^{(alg)}$, $\boldsymbol{a}^{(cate)}$, and $\boldsymbol{a}^{(imp)}$. Practitioners may incorporate new algorithms into the representation following the above process.

We take $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ to serve as the representation vector of hyper-parameters where n_θ is the total number of distinct hyper-parameters across all algorithms and pre-processing methods. Each

distinct hyper-parameter is regarded as an element of $\boldsymbol{\theta}$. All the elements in $\boldsymbol{\theta}$ are set to zero except for the hyper-parameters used in the current solution, which are assigned their respective values.

By utilizing the aforementioned representation vectors for each component, we can construct the representation of a solution. Let $\mathcal{P}$ be the set of n solutions used for training, and $\mathcal{P}$ can be denoted as $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$. By concatenating the representation vectors of the four components, we define the representation vector for a solution $\mathcal{P}_i \in \mathcal{P}$ as $\boldsymbol{x}_i$ where $\boldsymbol{x}_i = (\boldsymbol{d}_i, \boldsymbol{p}_i, \boldsymbol{a}_i, \boldsymbol{\theta}_i)$.

3.3 Ranking Model

With the representation vectors of solutions, we proceed to train a ranking model for the solutions. The training process is shown in the middle of Figure 3.

Ranking Model Structure. We design a weight-sharing network structure for our ranking model which is based on the Siamese neural network [11]. The Siamese neural network was originally intended for learning the similarity between pairs of training instances. In our case, however, the ranking model aims to learn the relative ranking between the performance of two solutions. Our ranking model consists of two weight-sharing PMMs. Each PMM is responsible for measuring the performance and generating a performance score for an input solution. A simple PMM network can be built with multiple fully connected layers and a single neuron in the last layer. However, such a network may suffer from certain features, e.g., the data features, dominating over other features, e.g., the algorithm or the hyper-parameter features. This imbalance becomes more apparent when we have insufficient training solutions. To address this issue, we propose a balanced structure for PMM that learns the features separately.

Balanced Structure. The balanced PMM, illustrated in the middle of Figure 3, contains two parallel balanced layers. One balanced layer is used to map the dataset features (i.e., $\boldsymbol{d}$), while the other balanced layer is used to map the remaining features (i.e., $\boldsymbol{a}$, $\boldsymbol{p}$, and $\boldsymbol{\theta}$). The outputs of these balanced layers are then fed into a hidden layer together. The last layer of the PMM network contains a single neuron that outputs the learned performance score for an input solution. During the training process, the ranking model takes a pair of solution vectors as inputs each time. The input pair is denoted as $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the ith and jth training solutions, respectively. Then, the weight-sharing PMMs generate two performance scores $\hat{s}_i$ and $\hat{s}_j$ for $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively.

Loss of the Ranking Model. The objective of training the ranking model is to learn the ranking relationship between two performance scores. To represent the ranking relationship, we introduce a ranking metric r_{ij} which computes the distance from the performance score $\hat{s}_i$ to $\hat{s}_j$ as follows:

$$r_{ij} = \text{sigmoid}(\hat{s}_i - \hat{s}_j) = \frac{1}{1 + e^{-(\hat{s}_i - \hat{s}_j)}}. \tag{1}$$

We use the sigmoid function to normalize the distance as computing the distance directly with $\hat{s}_i - \hat{s}_j$ leads to extremely large or small values of the predicted score. Our ranking metric is asymmetric which means that r_{ij} of the input pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is not equal to r_{ji} of the input pair $(\boldsymbol{x}_j, \boldsymbol{x}_i)$. The ranking metric's value depends on the order of the input solutions, which provides an opportunity to generate additional training pairs. By swapping the order of the solutions in an input pair, we can create quadratic times more training pairs for our model.

As the ranking metric reflects predictive rank, we use historical performance records of solutions to compute true ranks. The true rank of an input pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is denoted by y_{ij} as follows, which relies on the input order of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

$$y_{ij} = \mathbb{I}(s_i > s_j), \tag{2}$$

where $\mathbb{I}(\cdot)$ is an indicator function and s_i is the true performance score of the solution $\boldsymbol{x}_i$. The indicator function equals 1 when the inequality $s_i > s_j$ holds, and 0 otherwise. We use the historical performance (i.e., the predictive accuracy) as the true performance score. Note that practitioners have the flexibility to substitute the performance score with another evaluation metric of interest.

Once we have the predictive rank r_{ij} and the true rank y_{ij}, we can derive the objective function of our ranking model. To capture the ranking relationship in the loss, based on the common contrastive loss [24], we propose a new contrastive loss as shown below:

$$\mathcal{L} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2}((1 - y_{ij}) \cdot r_{ij}^2 + y_{ij} \cdot \max(\epsilon - r_{ij}, 0)^2), \tag{3}$$

where n is the number of training solutions and the hyper-parameter ϵ is a margin larger than which r_{ij} is ignored. The margin constrains the range of the learned scores. When the true performance score s_i is larger than the true performance score s_j (i.e., $y_{ij} = 1$), minimizing the loss in Equation (3) corresponds to maximizing the ranking metric. On the other hand, when the true rank y_{ij} is equal to 0, the objective is to minimize the squared value of the ranking metric. Since the ranking metric monotonically increases and the value of which is always positive, minimizing the squared value of the ranking metric is equivalent to minimizing the value of the ranking metric itself. As shown in Equation (1), the minimization (resp. maximization) of our ranking metric constrains the predicted performance score $\hat{s}_i$ to be smaller (resp. larger) than the performance score $\hat{s}_j$, while maintaining a large gap between $\hat{s}_i$ and $\hat{s}_j$. By optimizing this loss function, our proposed PMMs learn to capture the relative score of each solution's performance rather than the exact performance itself. This enables PMMs to be less susceptible to memorizing specific performance scores of individual solutions and helps mitigate the risk of overfitting.

Training of Our Ranking Model. Based on the proposed ranking model structure and loss, the training process of our ranking model is summarized in Algorithm 1. Initially, we generate the representation vectors for all training solutions (Line 2). Next, each pair of solutions is fed into the PMM network which predicts their performance scores (Line 7). We then calculate the ranking metric r_{ij} using the predicted performance scores (Line 8) and determine the true rank y_{ij} based on the historical performance records (Line 9). Afterward, we compute the loss using our proposed contrastive loss function (Line 10). This loss is used to update the PMM network's weights through backpropagation (Line 11). This iterative training process continues for multiple epochs until convergence is achieved.

3.4 Solution Prediction

The well-trained PMM from Algorithm 1 is used to predict the promising solution for a given dataset. The process of the solution prediction is illustrated in the right part of Figure 3. We first generate candidate solutions by considering various combinations of the given data, different pre-processing methods, and algorithms with their hyper-parameters. These candidate pre-processing methods, algorithms, and hyper-parameters are the widely used ones on the OpenML platform (see Section 4 for more details). We represent the candidate solutions in vectors as demonstrated in Section 3.2. Then, one candidate solution at a time is fed into the optimized PMM, which predicts a score for each candidate solution. The candidate solution with the highest score is recommended to the practitioner as the most promising choice.

3.5 Theoretical Analysis on the Computational Complexity

We further provide an analysis on the computational complexity of our method. During the solution representation phase, the complexities of generating the features for pre-processing

Algorithm 1: Ranking Model Training

Input: Historical solutions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$ with their historical performance scores $S = \{s_1, s_2, \ldots, s_n\}$

Output: The optimized PMM

```
foreach P_i ∈ P do
    x_i ←constructSolutionRepresentation(P_i)                    // Section 3.2
foreach epoch do
    for i ← 1 to n do
        for j ← 1 to n do
            ŝ_i ←PMM(x_i)
            ŝ_j ←PMM(x_j)
            r_ij ←rankMetric(ŝ_i, ŝ_j)                           // Equation (1)
            y_ij ←rankLabel(s_i, s_j)                            // Equation (2)
            L+ =rankLoss(r_ij, y_ij)                             // Equation (3)
    PMM←updatePMM(L/n², PMM)                                     // Back propagation
```

methods, algorithms, and hyper-parameters scale linearly with the number of each aforementioned component, respectively. Since the complexity of generating meta-features for datasets depends on the specific methods applied in practice, as shown in our experimental studies, we refrain from providing a detailed analysis for this aspect.

Next, we proceed to analyze the computational complexity of our ranking model. Let d_{b1} and d_{b2} represent the dimensions of fully-connected balance layers responsible for processing data features and the residual features of a solution, respectively. Additionally, assume d_l denotes the dimensionality of the layer immediately succeeding the balance layers. Given that the primary components of our model are PMM, label generation, and loss computation, with their respective complexities denoted as O_{PMM}, O_{label}, and O_{loss}, the overall computational complexity O of our model can be expressed as $O = O_{PMM} + O_{label} + O_{loss}$. According to Equations (2) and (3), we can deduce that $O_{loss} = O_{label} = O(n^2)$. As for PMM illustrated in Figure 3, we have $O_{PMM} = O(n[n_d d_{b1} + (d_s - n_d)d_{b2} + d_l(d_{b1} + d_{b2})]) \approx O(n[d_s(d_{b1} + d_{b2}) + d_l(d_{b1} + d_{b2})]) = O(n(d_s + d_l)(d_{b1} + d_{b2}))$, where $d_s = n_d + n_p + n_a + n_\theta$ represents the total dimensionality of our solution representation. Since the dimensions of the layers within PMM are practically much smaller than the solution dimension, i.e, $d_{b1}, d_{b2}, d_l << d_s$, the complexity of PMM can be simplified to $O_{PMM} = O(nd_s)$. By aggregating individual complexities, we derive the overall complexity of our model as follows:

$$O = O_{PMM} + O_{label} + O_{loss} \approx O(2n^2 + nd_s) \approx \begin{cases} O(n^2), & n \geq d_s, \\ O(nd_s), & n < d_s. \end{cases} \tag{4}$$

3.6 Advantages of Our Recommendation Method

In comparison with other solution recommendation methods, our method boasts the following advantages that concurrently tackle the challenges outlined at the beginning of Section 3. First, our method is equipped with techniques designed to handle limited and sparse training records. On the one hand, our method incorporates an inherent solution augmentation. This augmentation is facilitated by the asymmetric ranking metric, which enables our method to take pairs of solutions as inputs and the number of inputs is quadratic in the number of training solutions. On the other

Table 1. Advantages of Our Method over Existing Methods

Recommendation method	Inference-based recommendation	Optimization-based recommendation	Ours
Recommendation strategy	Learn a mapping to the exact solution or performance	Evaluate and optimize solutions on the given data	Incorporate solution ranking into performance scores
Trainable with sparse records	✗	✓	✓
Diverse search space	✗	✓	✓
Not requiring evaluation on the given data	✓	✗	✓
Augmentation of solutions	✗	✗	✓

hand, our method learns a performance score that indicates the relative rank of each solution. Therefore, our method focuses on the underlying patterns and trends that determine the relative performance of solutions, rather than relying on precise performance values. This leads to a more robust and generalizable recommendation. Second, we use a meta-learning-based representation to characterize solutions which decouples the search space in the recommendation from the prior experience. This allows our method to search and recommend candidate solutions that may not have been evaluated on any historical datasets. The search space is thus expanded. Third, as detailed in Section 3.3, we propose a balanced PMM to effectively address the dominance issues associated with certain types of features. Last but not least, our recommendation method does not require additional evaluation on the encountered dataset, which saves much computation burden. We highlight the superiority of our method compared with other baselines in Table 1.

4 Experimental Studies

In this section, we present the empirical studies conducted to evaluate the performance of our proposed method. We compare our method with existing solution recommendation approaches under two different scenarios: one with dense historical records used in training, and the other with sparse historical records.

4.1 Benchmarks and Experimental Setup

Benchmarks. In our experimental studies, we use two benchmarks: the *classifier-110* benchmark and the *openml-75%* benchmark, which include the dense and sparse historical performance records, respectively. The available performance records include the training and test accuracy.

— *classifier-110* benchmark: The *classifier-110* benchmark consists of solutions used in a research study [16] that investigated the effects of 179 classifiers on 121 datasets. In our experiments, we chose the classifiers that were executed on all the datasets and excluded the ones that were not. This results in a subset of 110 classifiers, which allows us to construct a set of 121×110 solutions with dense performance records. We use the Pymfe library [2] to generate 86 meta-features for each dataset since the original study does not provide meta-features for the datasets.

— *openml-75%* benchmark: We collect 956 datasets and 681 different combinations of widely used pre-processing (e.g., scaling, imputing) and machine learning algorithms (e.g., SVM, RF) with various hyper-parameters from the OpenML platform [51]. This results in a total of 956×681 solutions. It is important to note that the performance records of approximately 75% (77.47% to be precise) solutions in this benchmark are not available on OpenML. This level of sparsity reflects real-world scenarios where performance records may not be available for a significant portion of solutions. We use the 69 data features provided by OpenML as the meta-features for data. More details of the benchmarks and meta-features used in our experiments can be found in our Supplementary Material.

Experimental Setup. To provide a clearer understanding of the experimental settings, we introduce the concept of tasks. In our experiments, a task consists of a dataset along with all the possible solutions and their corresponding performance records on that dataset. For each benchmark, we divide the datasets into training tasks and test tasks as follows:

- We take 80% datasets in a benchmark (e.g., in *classifier-110*, we take 121×80%≈97 datasets), along with their corresponding solutions and performance records, to create training tasks. 20% of the training tasks serve as validation tasks for model configuration.
- The remaining datasets in a benchmark, along with their solutions and performance records, form the test tasks. These datasets have different distributions from those in training tasks, as they originate from distinct areas. Within each test or validation task, the solutions are considered candidate solutions, and among them, the one with the highest performance record (i.e., accuracy in our experiments) is identified as the best solution for that task.

We conducted our experiments on a server with an Intel(R) Xeon(R) CPU of 126GB memory running on a Linux OS. Our method is implemented with PyTorch [42]. The hyper-parameters of both the baseline models and our proposed method were tuned with the following configurations. The stochastic gradient descent with momentum set to 0.9 serves as the optimizer, and the learning rate is selected from the set {0.1, 0.01, 0.001}. All the methods are trained for 200 epochs. The balanced PMM in our method uses two parallel fully connected layers as the balance layer which are followed by a hidden layer. Each parallel layer or the hidden layer has n neurons, where n is selected from {20, 50, 100}. The margin ϵ in our objective function is set as 0.9. Moreover, to ensure a fair comparison, we use the *same* solution representation approach as detailed in Sections 3.2 and 4.1 for both the baselines and our method. Our Supplementary Material, source code and benchmarks are available on the GitHub.[1]

4.2 Analysis on the Performance of Solution Recommendation

Given the practical limitations in establishing ground-truth optimal solutions for real-world data science problems, we adopt a comparative analysis against existing methods to demonstrate the effectiveness and superiority of our proposed approach. The baseline methodologies evaluated in our study include the following:

- *Inference-Based Recommendation.* This type of recommendation method learns the mapping from the dataset to the most promising solution. We choose the widely used mapping models for our inference-based baselines which are: (i) AdaB [33]; (ii) **Classification Neural Networks (ClsNN)**, and (iii) **Regression Neural Networks (RegNN)** [10, 39]; (iv) RF [50]. Each network has two fully connected layers as hidden layers. The first hidden layer has $2n$ neurons and the next hidden layer has n neurons, which are consistent with our ranking model. The tuning of the hyper-parameter n is executed as described in the last paragraph of Section 4.1.
- *Optimization-Based Recommendation.* (i) DP [45]; (ii) PMF [21]; (iii) AutoSK [17]. These optimization-based baselines select k solutions as warm start and then fine-tune the warm start solutions using BO variants as described in Section 2.2. We set $k = 5$ following the default setting [17]. Note that PMF, AutoSK, and DP require fine-tuning of the solutions on the *validation/test* datasets, which is not necessary in our method. Hence, for a fair comparison, we focus solely on the k solutions selected prior to applying BO. We leave the discussion about the solutions tuned after BO in Section 4.3.
- **Heuristic Search (HS)** [48]. The solution with the highest average rank over all the datasets in training tasks is selected as the best solution for a validation or test dataset.

[1] https://github.com/JirehChan/SolutionRecommendation.

Table 2. Comparison on the Averaged Performance

Method	classifer-110			openml-75%			openml-85%			openml-95%		
	Rank	Accuracy (%)	NDCG@5	Rank	Accuracy (%)	NDCG@5	Rank	Accuracy (%)	NDCG@5	Rank	Accuracy (%)	NDCG@5
AdaB	2.12 ± 1.45	83.71 ± 12.56	0.993	6.45 ± 2.55	77.74 ± 15.25	0.973	6.59 ± 2.83	77.74 ± 15.73	0.979	6.29 ± 2.51	76.28 ± 15.95	0.973
ClsNN	2.64 ± 2.24	83.62 ± 12.46	0.985	5.36 ± 2.51	81.26 ± 14.80	0.978	5.54 ± 2.27	81.50 ± 13.93	0.989	4.56 ± 2.57	81.53 ± 16.45	0.984
RegNN	2.12 ± 1.45	83.71 ± 12.56	0.993	6.23 ± 2.91	72.65 ± 22.74	0.961	5.46 ± 2.76	79.72 ± 16.32	0.982	4.96 ± 2.61	80.59 ± 15.93	0.983
RF	5.00 ± 3.51	82.16 ± 11.88	0.989	4.09 ± 2.82	82.94 ± 15.58	0.986	3.82 ± 2.44	83.49 ± 14.35	0.989	3.76 ± 2.44	83.82 ± 14.95	0.989
AutoSK	5.20 ± 3.52	81.54 ± 12.90	0.987	3.93 ± 2.61	83.23 ± 15.09	0.987	3.89 ± 2.76	82.61 ± 15.50	0.986	4.16 ± 2.52	82.50 ± 15.65	0.985
DP	8.08 ± 2.94	73.63 ± 15.12	0.969	5.52 ± 2.86	78.13 ± 18.32	0.972	5.89 ± 2.87	77.35 ± 17.18	0.972	5.74 ± 2.62	77.51 ± 17.40	0.974
PMF	4.32 ± 3.09	82.99 ± 12.24	0.991	5.53 ± 3.21	63.17 ± 37.85	0.750	5.36 ± 3.35	63.12 ± 37.87	0.748	9.75 ± 1.04	6.89 ± 21.49	0.099
HS	2.12 ± 1.45	83.71 ± 12.56	0.991	6.46 ± 2.48	80.78 ± 14.44	0.987	6.57 ± 2.58	80.74 ± 14.51	0.987	4.75 ± 2.19	83.65 ± 14.20	0.989
RS	7.36 ± 2.50	82.14 ± 12.18	0.991	6.68 ± 2.75	73.52 ± 18.04	0.965	6.72 ± 2.85	73.52 ± 18.04	0.965	6.51 ± 2.42	73.52 ± 18.04	0.965
Ours	**2.04 ± 1.56**	**84.49 ± 11.71**	**0.994**	**2.87 ± 1.99**	**85.37 ± 14.05**	**0.990**	**3.47 ± 2.11**	**84.26 ± 13.36**	**0.994**	**2.74 ± 1.96**	**85.39 ± 13.82**	**0.991**

The best results are highlighted in bold.

—**Random Search (RS)**. It selects a solution randomly from the candidates.

Overall Performance. The performance of solution recommendation based on the *classifier-110* and *openml-75%* benchmarks is detailed in the left half of Table 2. The primary metrics for evaluating a recommendation method $\mathcal{M}$ include the average rank, average test accuracy, and average **Normalized Discounted Cumulative Gain (NDCG)** which are computed as follows:

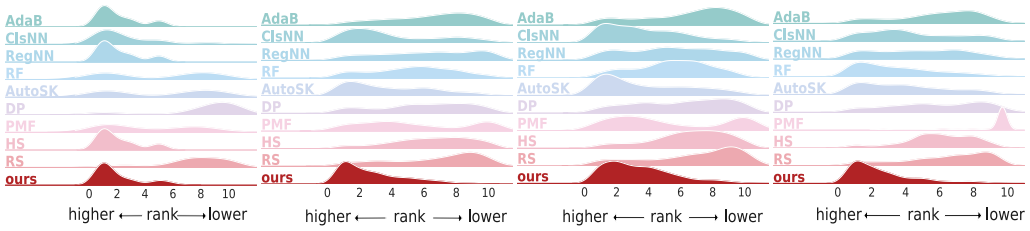
$$\text{average rank}_{\mathcal{M}} = \frac{1}{D_B}\sum_{i=1}^{D_B} relRank_{\mathcal{M},i}, \quad \text{average accuracy}_{\mathcal{M}} = \frac{1}{D_B}\sum_{i=1}^{D_B} testAcc_{\mathcal{M},i},$$

$$\text{NDCG}_{\mathcal{M}}@5 = \frac{1}{D_B}\sum_{i=1}^{D_B}\left[\sum_{j=1}^{5}\frac{testAcc^{(j)}_{\mathcal{M},i}}{\log(j+1)} \Big/ \sum_{j=1}^{5}\frac{testAcc^{(j)}_{\mathcal{M},i}}{\log(rank(\mathcal{M},j)+1)}\right],$$

$$\mathcal{M} \in \{\text{AdaB, ClsNN, RegNN, RF, AutoSK, DP, PMF, RS, HS, ours}\},$$

where D_B denotes the total number of test datasets in the benchmark B. With each of the 10 recommendation methods in $\mathcal{M}$ predicting the most promising solution for the ith test dataset, we can obtain 10 potential solutions. Organizing these solutions in the descending order of test accuracy gives us the relative rank of the solution predicted by method $\mathcal{M}$ as $relRank_{\mathcal{M},i}$ and $relRank_{\mathcal{M},i} \in \{1, \ldots, 10\}$. Moreover, we refer to the test accuracy of the top-1 solution recommended by the method $\mathcal{M}$ for the ith dataset as $testAcc_{\mathcal{M},i}$. In addition to evaluating the performance of the top-1 solution recommended by each method, we also assess the ranking quality of a list of predicted solutions. $\text{NDCG}_{\mathcal{M}}@5$ measures the similarity between the predicted rankings and the true rankings of the top-5 solutions predicted by method $\mathcal{M}$. In the formula of $\text{NDCG}_{\mathcal{M}}@5$ metric, $testAcc^{(j)}_{\mathcal{M},i}$ indicates the test accuracy of the jth best solution predicted by method $\mathcal{M}$ for the ith test datasets. We denote the true rank of the jth best solution recommended by method $\mathcal{M}$ as $rank(\mathcal{M}, j)$ where j, $rank(\mathcal{M}, j) \in \{1, \ldots, 5\}$. Given that inference-based baselines cannot be applied to the sparse benchmark (e.g., *openml-75%*), we assigned a performance score of zero to solutions without historical performance records. As indicated in Table 2, our method consistently outperforms all the baselines in accuracy, rank, and NDCG values. This sustained performance demonstrates the ability of our method to predict solution rankings that closely align with the true rankings.

Overall Performance on Different Sparsities of Training Records. To investigate the impact of different training record sparsities on solution recommendation, we vary the sparsity level of the *openml-75%* benchmark by randomly removing the historical records in it. The recommendation results which were provided by the methods trained on different sparsity levels are presented in the right half of Table 2. The *openml-85%* (resp. *openml-95%*) benchmark indicates that 85%



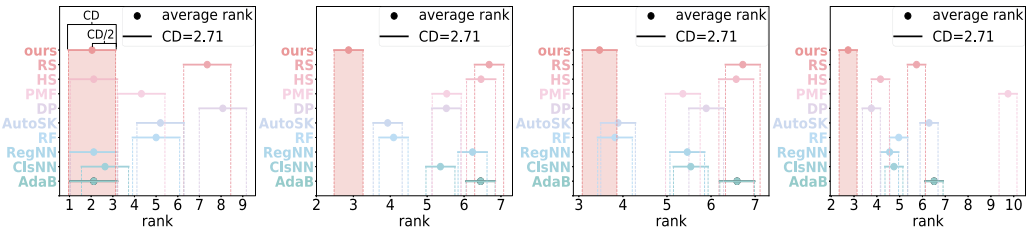
(a) classifier-110 (b) openml-75% (c) openml-85% (d) openml-95%

Fig. 4. Rank distributions of the solutions recommended for all the tested datasets.

(resp. 95%) performance records of the solutions are unavailable. Even with highly sparse training tasks (e.g., *openml-95%*), our method still outperforms the baselines. Notably, PMF experiences a significant decrease in accuracy, because it is difficult for PMF to select suitable warm start solutions from highly sparse historical records. Although the randomly selected solutions remain unaffected by changes in sparsity levels, their average ranks correlate with the variations of sparsity. This phenomenon arises due to the relative nature of rank, which is contingent on the ranks of solutions predicted by other methods. Alteration in sparsity levels shifts the recommendations made by both other baselines and our method, leading to changes in the relative rank of random selection. Moreover, the superior performance of our method achieved on *openml-95%* relative to *openml-85%* may primarily stem from the higher signal-to-noise ratio of *openml-95%* benchmark. Fewer noisy samples and a greater proportion of high-quality solutions in the training set allow models to more effectively identify discriminative features for solution recommendation. The empirical results of the baselines reaffirm our findings. As shown in Table 2, not only our method but also the majority of the tested baselines (i.e., 6 out of 9 baselines) achieve better performance on *openml-95%* compared with *openml-85%*. Analogous observation can be found in the previous study [50]. Our experimentation with different sparsity levels simulates real-world scenarios and demonstrates the potential applicability of our method in industrial settings.

Detailed Results. In addition to the averaged performance presented in Table 2, we provide a detailed analysis on the relative rank $relRank_{\mathcal{M},i}$ for each test datasets. Figure 4 depicts the distributions of relative ranks for the solutions recommended by each method for all the test datasets. Notably, our method shows a concentrated distribution around the first place, marked by a higher peak in the distribution. On the contrary, other methods exhibit more diverse distributions with longer and wider tails. Our method stands out for providing more accurate solution recommendations for unseen problems, even when trained on highly sparse historical records. This demonstrates the effectiveness and superior generality of our method.

Tests of Statistical Significance. We validate the significance of the comparison among all the tested recommendation methods on different datasets using the Friedman test. As listed in the middle column of Table 4, the p-value on each benchmark is smaller than the significance level of 0.05, which indicates that the differences among the 10 tested recommendation methods, including the baselines and our method, are significant. To delve deeper into the significance of the difference between our method with each baseline, we adopt the Nemenyi *post hoc* test, and the results are visually presented in Figure 5. The width of each bar corresponds to the **Critical Difference (CD)** on that benchmark. If the difference between the average rank of our method and that of a baseline exceeds the CD, meaning that the baseline bars do not intersect with the red bar, the result is deemed significant. Observing Figure 5(b) through (d), none of the baselines overlaps with the red



(a) classifier-110 (b) openml-75% (c) openml-85% (d) openml-95%

Fig. 5. Nemenyi *post hoc* test.

bar on the sparse benchmarks, apart from AutoSK and RF on *openml-85%*. In Figure 5(a), the results indicate less significance on the dense benchmark, with AdaB, RegNN, ClsNN, and the heuristic method showing comparable performance to our method. In summary, the statistical significance tests provide evidence that our method performs significantly better than the baselines on most benchmarks, especially sparse ones, demonstrating its effectiveness in solution recommendations.

4.3 Case Studies: Fine-Tuning on the Test Tasks

In this case study, we conduct the full process of our optimization-based baselines where the solutions selected in the warm start stage are further fine-tuned on the test tasks. We divide the dataset in each test task into two subsets where 80% of the data serves as the support set and the remaining 20% serves as the query set. The warm start solutions are first tuned on the support set using the optimization techniques in PMF and AutoSK, respectively. The solution which achieved the highest predictive accuracy on the support set is recommended as the best solution. Then, we apply the recommended solution on the query set to compute the test accuracy. We denote the number of tuning iterations as t and take $t = 50$ in our experiments. As AutoSK, DP, or PMF evaluates t solutions in total (one in each iteration) during fine-tuning, for a fair comparison, we evaluate the top-t solutions directly predicted by our method on the support set to find the solution with the best performance. Similarly, the random strategy selects one solution in each iteration and evaluates the t chosen solutions on the support set.

Overall Performance of Solution Recommendation with Fine-Tuning. We list the average rank, average test accuracy, and average NDCG@5 of the solutions recommended after fine-tuning in Table 3. The results show that our method not only attains the best test accuracy compared with the fine-tuned baselines but also produces a recommendation list of the top-5 solutions with higher quality. Furthermore, we illustrate the variation of average test accuracy and average rank throughout iterations in Figures 6 and 7, respectively. We observed that the solution recommended by our method consistently achieves the highest accuracy within the same number of iterations while maintaining small ranks. Remarkably, the top-t solutions directly predicted by our method outperform most of those tuned with PMF and AutoSK. The significance of these fine-tuned results is confirmed by the Friedman test, as demonstrated in the final column of Table 4.

Detailed Results. We draw the distributions of relative ranks for the solutions recommended after fine-tuning in Figure 8. The relative ranks of our method converge to the highest rank with a higher peak, particularly on the sparse benchmarks. While AutoSK shows a slight advantage on the dense benchmark, our method still performs competitively. These compact distributions of our method reaffirm its superior performance compared to the carefully tuned baselines.

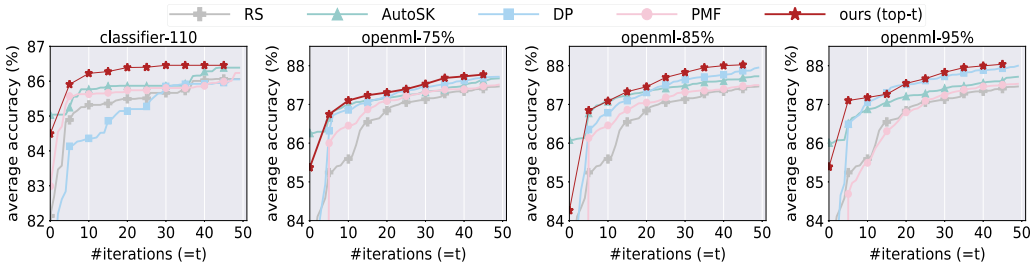


Fig. 6. Average accuracy varied with iterations.

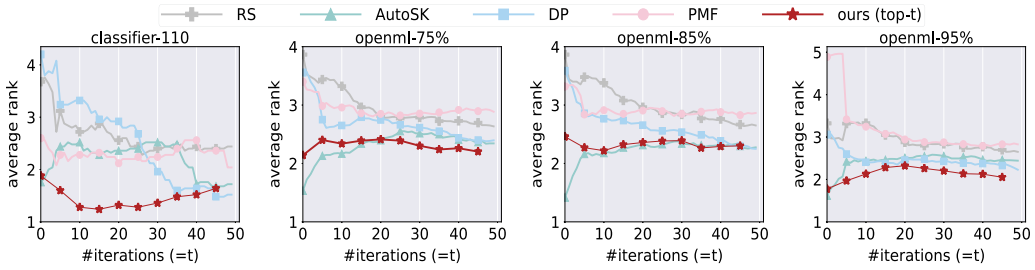
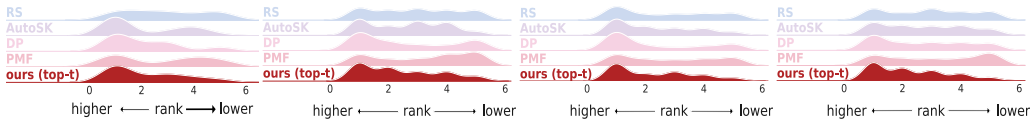


Fig. 7. Average rank varied with iterations.



(a) classifier-110 (b) openml-75% (c) openml-85% (d) openml-95%

Fig. 8. Rank distributions of fine-tuned solutions for all the tested datasets.

Efficiency Comparison. In the meantime, our method offers substantial time savings compared with the other baselines. Our approach requires just one training phase of about 210 seconds per benchmark and can be promptly applied to multiple new problems without additional evaluations. Conversely, optimization-based methods demand extra evaluations for every new problem. As shown in Figure 9, our method takes few seconds to recommend the top-50 solutions which are 30 to 10^4 times faster than AutoSK, DP, or PMF. This efficiency makes our method highly practical as it delivers higher-quality solutions with far less recommendation time than those heavily tuned.

4.4 Ablation Study

In this study, we empirically observe the effectiveness of our balanced structure and asymmetric ranking metric, respectively. In the first part of the study, we replace the balanced structure with a variant that retains the same number of layers and layer sizes as the balanced PMM, but omits the division of one layer into two balanced layers. In the second part, we modify the ranking metric to use a symmetric distance measurement. The results of this comparison, including rank and test accuracy for the recommended solutions across test tasks, are summarized in Table 5. The findings indicate that our method consistently outperforms alternatives, regardless of whether the historical performance records are dense or sparse. This reveals the importance of our balanced structure and asymmetric ranking measurement in enhancing the efficacy of solution recommendations.

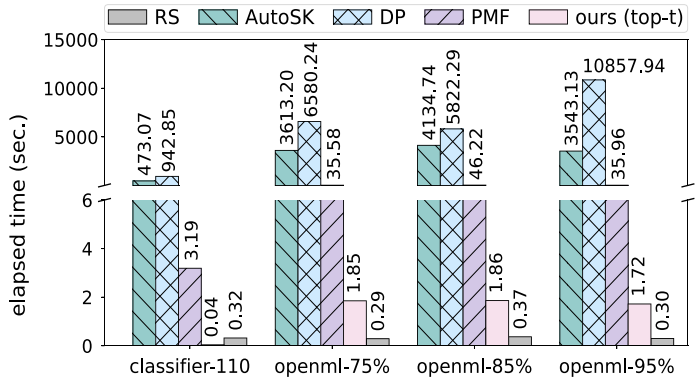


Fig. 9. Total elapsed time on solution recommendation.

Table 3. Averaged Performance with Fine-Tuning

Benchmark	Method	Accuracy (%)		Rank		NDCG@5	
		t=25	t=50	t=25	t=50	t=25	t=50
classifier-110	AutoSK	85.87	86.39	2.36	1.68	0.979	0.977
	DP	85.14	85.95	2.92	1.64	0.992	**0.982**
	PMF	85.74	85.99	2.12	2.36	0.979	0.978
	RS	85.49	86.06	2.56	2.40	0.984	0.979
	Ours (top-t)	**86.39**	**86.46**	**1.32**	**1.64**	**0.992**	0.975
openml-75%	AutoSK	87.16	87.61	2.40	2.38	0.974	0.971
	DP	87.21	87.68	2.74	2.40	0.972	0.978
	PMF	87.09	87.44	2.85	2.91	0.976	0.968
	RS	86.84	87.40	2.81	2.68	0.961	0.976
	Ours (top-t)	**87.31**	**87.76**	**2.39**	**2.23**	**0.977**	**0.979**
openml-85%	AutoSK	87.33	87.65	2.31	2.33	0.983	0.982
	DP	87.31	87.85	2.66	2.31	0.981	0.980
	PMF	87.05	87.46	2.91	2.84	0.975	0.972
	RS	86.84	87.40	2.96	2.66	0.961	0.976
	Ours (top-t)	**87.46**	**88.02**	**2.36**	**2.30**	**0.983**	**0.982**
openml-95%	AutoSK	87.21	87.60	2.51	2.45	0.978	0.975
	DP	87.52	87.93	2.46	2.35	**0.979**	0.981
	PMF	86.80	87.49	2.94	2.84	0.973	0.974
	RS	86.84	87.40	2.88	2.70	0.961	0.976
	Ours (top-t)	**87.54**	**88.03**	**2.32**	**2.05**	0.978	**0.982**

The best results are highlighted in bold.

Table 4. Significance Test (Significance Level $\alpha = 0.05$)

Benchmark	p-value	p-value (w/ tuning)
classifier-110	$8.40e^{-12}$	$5.97e^{-03}$
openml-75%	$3.66e^{-41}$	$7.74e^{-64}$
openml-85%	$1.23e^{-34}$	$1.23e^{-64}$
openml-95%	$4.02e^{-122}$	$8.15e^{-110}$

Table 5. Ablation Study of Our Method

Benchmark	Method	Accuracy (%)	Rank
classifier-110	w/o balance structure	80.43 ± 14.27	1.52 ± 0.81
	w/o asymmetric rank	78.70 ± 16.17	2.32 ± 0.84
	Our method	$\mathbf{84.49 \pm 11.71}$	$\mathbf{2.04 \pm 1.56}$
openml-75%	w/o balance structure	78.47 ± 15.91	2.16 ± 0.71
	w/o asymmetric rank	75.00 ± 16.47	2.23 ± 0.89
	Our method	$\mathbf{85.37 \pm 14.05}$	$\mathbf{2.87 \pm 1.99}$
openml-85%	w/o balance structure	77.03 ± 18.55	2.05 ± 0.77
	w/o asymmetric rank	74.75 ± 17.80	2.23 ± 0.88
	Our method	$\mathbf{84.26 \pm 13.36}$	$\mathbf{3.47 \pm 2.11}$
openml-95%	w/o balance structure	77.39 ± 15.95	2.21 ± 0.68
	w/o asymmetric rank	72.24 ± 22.24	2.25 ± 0.90
	Our method	$\mathbf{85.39 \pm 13.82}$	$\mathbf{2.74 \pm 1.96}$

The best results are highlighted in bold.

5 Conclusion

To provide promising data science solutions, encompassing pre-processing, learning algorithms, and hyper-parameters tailored to specific problems, we have proposed a novel solution recommendation method for data science problems. Our approach involves extracting meta-features from solutions and applying a ranking model to learn their performance scores, which imply the relative ranks of them. The ranking model is equipped with a series of techniques including (i) two weight-sharing PMMs, (ii) a balancing mechanism, and (iii) a new contrastive loss function. By leveraging these techniques, our method effectively addresses the challenges related to sparse historical performance records, such as inadequate representation of solution space and risk of overfitting. The experimental results affirm the superior generalization of our proposed method to future tasks, when compared to existing inference-based and optimization-based baselines. Our method consistently

recommends solutions with higher accuracy and rank for the encountered tasks, regardless of whether trained on dense or highly sparse historical records. Our case study further demonstrates its significant outperformance over heavily tuned baselines. This solution recommendation method empowers practitioners to efficiently choose high-quality solutions for their data science problems. Furthermore, exploring adaptive incremental learning frameworks to dynamically incorporate evolving data science problems, solutions, and their historical records in training is an important research direction for solution recommendation. It is also promising to develop interpretable recommendation mechanisms that provide transparent rationales for suggested solutions. We leave these interesting research ideas for future work.

References

[1] Inas Amjed Al Mani, Ali M. Ahmed Al-Sabaawi, and Mohsin Hasan Hussien. 2022. A review paper of model based collaborative filtering techniques. In *Proceedings of the 2022 IEEE International Conference on Data Science and Intelligent Computing (ICDSIC)*, 52–57.

[2] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís Paulo F. Garcia, Jefferson Tales Oliva, André C. P. L. F. de Carvalho. 2020. MFE: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research* 21 (2020), 1–5.

[3] Rahman Ali, Sungyoung Lee, and Tae Choong Chung. 2017. Accurate multi-criteria decision making methodology for recommending machine learning algorithm. *Expert Systems with Applications* 71 (2017), 257–278.

[4] Shawkat Ali and Kate A. Smith-Miles. 2006. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing* 70, 1-3 (2006), 173–186.

[5] Zafar Ali, Yi Huang, Irfan Ullah, Junlan Feng, Chao Deng, Nimbeshaho Thierry, Asad Khan, Asim Ullah Jan, Xiaoli Shen, Wu Rui, et al. 2023. Deep learning for medication recommendation: A systematic survey. *Data Intelligence* 5, 2 (2023), 303–354.

[6] Abraham Bernstein, Foster Provost, and Shawndra Hill. 2005. Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering* 17, 4 (2005), 503–518.

[7] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. 2010. Moa: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the 1st Workshop on Applications of Pattern Analysis (WAPA)*, 44–50.

[8] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. 2016. Mlr: Machine learning in R. *Journal of Machine Learning Research* 17, 1 (2016), 5938–5942.

[9] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. 2003. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning* 50, 3 (2003), 251–277.

[10] Sibei Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2023. Haipipe: Combining human-generated and machine-generated pipelines for data preparation. In *Proceedings of the ACM on Management of Data*, 1–26.

[11] Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, 539–546.

[12] Tiago Cunha, Carlos Soares, and André Cplf de Carvalho. 2018. Metalearning and recommender systems: A literature review and empirical study on the algorithm selection problem for collaborative filtering. *Information Sciences* 423 (2018), 128–144.

[13] Hans Degroote, Patrick De Causmaecker, Bernd Bischl, and Lars Kotthoff. 2018. A regression-based methodology for online algorithm selection. In *Proceedings of the 11th Annual Symposium on Combinatorial Search*, 37–45.

[14] Tri Doan and Jugal Kalita. 2015. Selecting machine learning algorithms using regression models. In *Proceedings of the IEEE International Conference on Data Mining Workshop (ICDMW)*, 1498–1505.

[15] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni De Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2021. AlphaD3M: Machine learning pipeline synthesis. In *Proceedings of the International Conference on Machine Learning AutoML Workshop*.

[16] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research* 15, 1 (2014), 3133–3181.

[17] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research* 23, 261 (2022), 1–61.

[18] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. 2016. Efficient and robust automated machine learning. *Advances in Neural Information Processing Systems (NeurIPS)* 28 (2016), 2944–2952.

[19] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. Chapter 6: Auto-sklearn: Efficient and robust automated machine learning. In *Automated Machine Learning: Methods, Systems, Challenges*. Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.), Springer International Publishing, Cham, 123–143.

[20] Andrey Filchenkov and Arseniy Pendryak. 2015. Datasets meta-feature description for recommending feature selection algorithm. In *Proceedings of the Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, 11–18.

[21] Nicolo Fusi, Rishit Sheth, and Melih Elibol. 2018. Probabilistic matrix factorization for automated machine learning. *Advances in Neural Information Processing Systems (NeurIPS)* 31 (2018), 3348–3357.

[22] Pieter Gijsbers, Joaquin Vanschoren, and Randal S. Olson. 2018. Layered TPOT: Speeding up tree-based pipeline optimization. arXiv:1801.06007. Retrieved from https://arxiv.org/abs/1801.06007

[23] Silvio B. Guerra, Ricardo B. C. Prudêncio, and Teresa B. Ludermir. 2008. Predicting the performance of learning algorithms using support vector machines as meta-regressors. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 523–532.

[24] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the 2nd IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, 1735–1742.

[25] Geoffrey Holmes, Andrew Donkin, and Ian H. Witten. 1994. Weka: A machine learning workbench. In *Proceedings of the ANZIIS'94-Australian New Zealand Intelligent Information Systems Conference*. IEEE, 357–361.

[26] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[27] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature.

[28] Anja Jankovic, Gorjan Popovski, Tome Eftimov, and Carola Doerr. 2021. The impact of hyper-parameter tuning for landscape-aware performance regression and algorithm selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 687–696.

[29] Houye Ji, Junxiong Zhu, Xiao Wang, Chuan Shi, Bai Wang, Xiaoye Tan, Yanghua Li, and Shaojian He. 2021. Who you would like to share with? A study of share recommendation in social e-commerce. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 232–239.

[30] Irfan Khan, Xianchao Zhang, Mobashar Rehman, and Rahman Ali. 2020. A literature survey and empirical study of meta-learning for classifier selection. *IEEE Access* 8 (2020), 10262–10281.

[31] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, 282–293.

[32] Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. 2015. Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In *Proceedings of the International Conference on Learning and Intelligent Optimization*. Springer, 202–217.

[33] Doron Laadan, Roman Vainshtein, Yarden Curiel, Gilad Katz, and Lior Rokach. 2019. Rankml: A meta learning-based approach for pre-ranking machine learning pipelines. arXiv:1911.00108. Retrieved from https://arxiv.org/abs/1911.00108

[34] Neil D. Lawrence and Raquel Urtasun. 2009. Non-linear matrix factorization with gaussian processes. In *Proceedings of the International Conference on Machine Learning*, 601–608.

[35] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Efficient automatic CASH via rising bandits. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 4763–4771.

[36] Weixin Liang, Yining Mao, Yongchan Kwon, Xinyu Yang, and James Zou. 2023. Accuracy on the curve: On the nonlinear correlation of ml performance between data subpopulations. In *Proceedings of the International Conference on Machine Learning*, 20706–20724.

[37] Fangyuan Luo, Jun Wu, and Tao Wang. 2023. Discrete listwise content-aware recommendation. *ACM Transactions on Knowledge Discovery from Data* 18, 1 (Aug. 2023), Article 7, 1–20. DOI: https://doi.org/10.1145/3609334

[38] Fernando Martínez-Plumed, Lidia Contreras-Ochando, Cèsar Ferri, José Hernández-Orallo, Meelis Kull, Nicolas Lachiche, María José Ramírez-Quintana, and Peter Flach. 2021. CRISP-DM twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering* 33, 8 (2021), 3048–3061.

[39] Mario A. Muñoz, Michael Kirley, and Saman K. Halgamuge. 2012. A meta-learning prediction model of algorithm performance for continuous optimization problems. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, 226–235.

[40] Mario A. Muñoz, Yuan Sun, Michael Kirley, and Saman K. Halgamuge. 2015. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences* 317 (2015), 224–245.

[41] Randal S. Olson and Jason H. Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the Workshop on Automatic Machine Learning*, 66–74.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019), 8026–8037.
[43] Yonghong Peng, Peter A. Flach, Carlos Soares, and Pavel Brazdil. 2002. Improved dataset characterisation for meta-learning. In *Proceedings of the International Conference on Discovery Science (DS)*, 141–152.
[44] Bernhard Pfahringer, Hilan Bensusan, and Christophe G. Giraud-Carrier. 2000. Meta-learning by landmarking various learning algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 743–750.
[45] Sebastian Pineda Arango and Josif Grabocka. 2023. Deep pipeline embeddings for AutoML. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1907–1919.
[46] Fábio Pinto, Carlos Soares, and Joao Mendes-Moreira. 2016. Towards automatic generation of metafeatures. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 215–226.
[47] Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. 2019. Automated machine learning with Monte-Carlo tree search. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 3296–3303.
[48] Wojciech Sałabun and Karol Urbaniak. 2020. A new coefficient of rankings similarity in decision-making problems. In *Proceedings of the International Conference on Computational Science*, 632–645.
[49] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 2951–2959.
[50] Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. 2020. Extreme algorithm selection with dyadic feature representation. In *Proceedings of the International Conference on Discovery Science*, 309–324.
[51] Joaquin Vanschoren, Jan N. Van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.
[52] Shanshan Wan and Zhendong Niu. 2019. A hybrid e-learning recommendation approach based on learners' influence propagation. *IEEE Transactions on Knowledge and Data Engineering* 32, 5 (2019), 827–840.
[53] Guangtao Wang, Qinbao Song, Heli Sun, Xueying Zhang, Baowen Xu, and Yuming Zhou. 2013. A feature subset selection algorithm automatic recommendation method. *Journal of Artificial Intelligence Research* 47 (2013), 1–34.
[54] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. 2022. A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4425–4445.
[55] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBOE: Collaborative filtering for AutoML model selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1173–1183.
[56] Zhenqian Shen, Yongqi Zhang, Lanning Wei, Huan Zhao, and Quanming Yao. 2024. Automated machine learning: From principles to practices. arXiv:1810.13306. Retrieved from https://arxiv.org/abs/1810.13306
[57] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys* 52, 1 (2019), 1–38.
[58] Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. arXiv:1611.01578. Retrieved from https://arxiv.org/abs/1611.01578

Received 6 October 2024; revised 13 April 2025; accepted 12 June 2025