Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Auto-CASH: A meta-learning embedding approach for autonomous classification algorithm selection

Tianyu Mu [a,*,1], Hongzhi Wang [a,b,*], Chunnan Wang [a], Zheng Liang [a], Xinyue Shao [a]

[a] Harbin Institute of Technology, No. 92, West Dazhi Street, Harbin, China
[b] Peng Cheng Laboratory(PCL), No. 2, Xingke 1st Street, Nanshan, Shenzhen, China

A R T I C L E I N F O

Article history:
Received 14 July 2021
Received in revised form 26 December 2021
Accepted 19 January 2022
Available online 24 January 2022

Keywords:
Classification
Autonomous algorithm selection
Hyperparameter optimization
Deep Q-Network
Meta-learning
AutoML

A B S T R A C T

With years of development, machine learning algorithms have excellent performance in some tasks of data analysis and data mining. To apply machine learning to new tasks, suitable algorithm and hyperparameters selection techniques, which is known as *Combined Algorithm Selection and Hyperparameter optimization problem*, are in demand. In the field of data analysis, how to automate the algorithm selection process has become a hot research topic in recent years. Most of the existing approaches are developed under the background of Automated Machine Learning with high time or space complexity. To alleviate the issue, an approach extracts and learns from prior experience based on *meta-learning* theory named Auto-CASH is proposed in this paper. One of the major drawbacks of existing meta-learning methods is that they rely too much on human expertise to extract and filter knowledge that guides subsequent training. Auto-CASH can automatically select features of tasks by introducing a reinforcement learning strategy. Thus Auto-CASH becomes less dependent on human expertise. Besides, two pruning strategies when processing Hyperparameter Optimization to improve efficiency are firstly proposed. Extensive experiments on classification tasks are conducted and results demonstrate that Auto-CASH outperforms state-of-the-art CASH approaches and popular AutoML systems with less time cost.

1. Introduction

Machine learning (ML) approaches have been used widely. Many algorithms (or models) [1] have been developed for specific problems [2]. However, for different datasets, the performance of these algorithms varies considerably. One learning algorithm cannot outperform another one in all aspects and problems [3]. Therefore, it is necessary to choose the most suitable algorithm and hyperparameters for a specific task. Such job is usually accomplished by domain experts according to their experiences and a series of experiments. While each algorithm has a complex hyperparameter configuration space. Even for an expert with adequate domain knowledge, it is hard to make an ideal selection among various algorithms and their complex hyperparameter space. Nonetheless, the suitable solution for the particular task instance is still desperately

* Corresponding authors at: Harbin Institute of Technology, No. 92, West Dazhi Street, Harbin, China.
E-mail addresses: mutianyu@hit.edu.cn (T. Mu), wangzh@hit.edu.cn (H. Wang), WangChunnan@hit.edu.cn (C. Wang), lz20@hit.edu.cn (Z. Liang), shaoxinyue@hit.edu.cn (X. Shao).
URL: http://homepage.hit.edu.cn/wang (H. Wang).
[1] Tianyu Mu is the first author.

https://doi.org/10.1016/j.ins.2022.01.040
0020-0255/© 2022 Elsevier Inc. All rights reserved.

needed in practice. In the face of such situation, Thornton et al. presented the Combined Algorithm Selection and Hyperparameter optimization problem(CASH) [4], aiming at selecting a suitable algorithm and configuring its hyperparameters for specific scenarios automatically.

Despite several works have already shown similar ideas [5], Thornton et al. first gave the formulaic definition of the CASH problem and introduced the model they proposed - Auto-WEKA [4]. It uses a hyperparameter to represent candidate algorithms, thereby converting the CASH problem into a hyperparameter optimization problem (HPO). However, Auto-WEKA will iterate online round by round to find the best solution, thus suffering from high time and space cost. One of the current phenomena is that many research papers related to ML have been proposed with extensive experiments, which detailed analyze the performance of many related algorithms with certain hyperparameter settings on different tasks. Such reported experimental results are pretty valuable to guide effective algorithm selection and reduce the search space. Thus, an approach adopt these experience to deal with the CASH problem are proposed and named Auto-Model [6]. Different from Auto-WEKA, Auto-Model extracts experimental results from previously published ML papers to create a knowledge base, making the selection of algorithms more intelligent and automated. The knowledge base can be updated with continuous training. A steady flow of training data will enhance the knowledge base gradually replacing the experience of experts. To the best of our knowledge, Auto-Model is the state-of-the-art CASH solver and performs better than Auto-WEKA on classification problems. Nevertheless, the quality of used paper will affect the effectiveness of the entire model, too much manual work is needed for evaluating each paper's contribution to the knowledge base. Two main shortcomings are still need to be addressed. a) The use of experience is still insufficient. In addition to using experience for training acceleration and assisting in algorithm selection, we should dig deeper into the information hidden inside the experience and find a suitable way to express the experience in order to fully demonstrate its value. b) Auto-Model extracts features of each task manually, which is a very inefficient and poorly interpretable choice. Although continuous experiments and attempts have been made to select the best set of feature combinations in the experimental results, final result may still fall into the local optimal solution.

How to reduce manual intervention in ML applying has gradually become a hot research topic [7]. The term Automated Machine Learning (AutoML) usually describes techniques (or systems) which can automate the ML pipeline. Although AutoML methods such as Auto-sklearn [8] can solve the CASH problem, they need to evaluate the candidate algorithms on data through a continuous exploration–exploitation trade off, which requires a large calculation and time consumption, and may lead to a local optimal solution. Another effective approach to address the CASH problem is *meta-learning* [9]. With training on the meta-data extracted from prior experience, the meta-learner is capable of recommending an optimal algorithm for new tasks [10]. The process of using experience to train the meta-learner is a deeper mining and application of experience, which also solves the *first* defect of Auto-Model we proposed in the previous paragraph. However, existing meta-learning-based methods cannot automatically determine the extracted meta-data, which still relies on human expertise. While existing solutions are effective for some scenarios, two major challenges remain unresolved. On the one hand, the whole workflow needs to be more automated. An effective strategy should automatically choose the meta-features [2]used. Selection of meta-features is a crucial step in meta-learning, while current approaches almost rely on human expertise. On the other hand, CASH has buckets effect. Although existing Hyperparameter Optimization (HPO) approaches such as Bayesian Optimization can achieve optimal performance [11], the high computation cost and massive computing requirements prevent them applying on the real CASH scenarios with complex search space.

Above all, existing work in AutoML cannot tackle those challenges well, which makes them inefficient in practice. The most effective way to improve efficiency is utilizing prior experience to train the model. Thus, we propose *Auto-CASH*, an approach based on meta-learning. The main research topic of our work is automatic model selection and hyperparameter optimization in the field of AutoML, mainly using meta-learning to guide algorithm selection and reinforcement neural networks to achieve automatic feature extraction and selection. Our approach focuses on the following 3 objectives.

a. Utilization of experiences. For the first challenge (also the *second* defect of Auto-Model), Auto-CASH represents the task as a feature vector and model the algorithm selection experience as the mapping from analysis task to its optimal process algorithm. A meta-learner is trained with the knowledge to identify high-performing algorithms for solving new tasks. By sufficiently utilizing the extracted prior experience, our approach is lightweight and efficient in practice. Auto-CASH utilizes *Deep Q-Network* (DQN)[12], a reinforcement learning approach, to automatically select meta-features. Compared to previous work, including Auto-Model, Auto-CASH utilizes a smaller number of meta-features, yet achieves better results. Then it leverages the meta-data of raw training datasets using the meta-feature to train a meta-learner. Since Random Forest (RF) is sensitive to the complex relationship among meta-features, it is used for the meta-learner. With RF, Auto-CASH achieves good performance and an acceptable time cost in algorithm selection. The advantages of DQN and RF and why they are better suited to our approach are as follows.

Advantages of Deep Q-Network. In Auto-CASH, the selection of meta-feature is transformed into a continuous action decision problem, which can be solved by Reinforcement Learning (RL) approaches. To the best of our knowledge, this is the first time that a reinforced policy is used to simulate a human expert using learned empirical knowledge to perform automatic feature selection. An RL approach includes two entities: the agent and the environment. The interactions between the two entities are as follows. Under the state s_t, the agent takes an action a_t and get a corresponding reward r_t from the environ-

[2] Meta-feature is the characterizations of a task and is considered to be a crucial source of meta-data. Each task $t_i \in T$ is described as a meta-feature vector $m(t_i) = (m_{i,1}, m_{i,2} \ldots, m_{i,j})$ of j meta-features.

ment, and then the agent enters next state s_{t+1}. The action decision process will be repeated until the agent meets termination conditions.

Continuous action decision problems have the following characteristics.

- For various actions, the corresponding rewards are usually different.
- The reward for an action is delayed.
- The value of the reward for an action is influenced by the current state.

The feature selection problem is also a sequential decision problem, where the importance of each feature becomes different with the combination of features that have been selected. Whether a feature is selected or not also plays a significant role in the selection of other features. Q-learning [13] is a classical value-based RL algorithm to solve the continuous action decision problems. Let $Q(s_t, a_t)$ value represent the reward of action a_t in state s_t. The main idea of Q-learning is to fill in the Q-table with $Q(s_t \in allStates, a_t \in allActions)$ value by iterative training. At the beginning of the training phase, i.e. exploring the environment, the Q-table is filled up with the same random initial value. As the agent explores the environment continuously, the Q-table will be updated using Eq. (1).

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s,a)] \tag{1}$$

For large enough states, it is clear that Q-table can no longer cope with such a situation, because it will lead to memory explosion and extreme consumption of hardware resources. Deep Q-Network (DQN)[12] is proposed, which uses neural network (NN) to analyze the reward of each action under a specific state. The input of DQN is current state, and the output is the estimated reward for each action under this state. The agent then randomly chooses actions with a probability of $\varepsilon(0 < \varepsilon < 1)$ and chooses actions with a probability of $1 - \varepsilon$ that can bring the maximal reward. It is called $\varepsilon - greedy$ exploration strategy, which can balance the exploration and the exploitation. In the beginning, the system will maximize the exploration space completely randomly. As training continues, ε will gradually decrease from 1 to 0. Finally it will be fixed to a stable exploration rate. In the training phase, DQN uses the strategy shown in Equation (1) to update the parameter values of NN. Under the state s, the agent selects action a which can obtain a maximum cumulative reward r according to NN, and then enters state s'. The parameters should be updated right now. γ, α denote the discount factor and learning rate, respectively. The difference between the true Q-value and the estimated Q-value is $\alpha[R(s,a) + \gamma max_{a'} Q(s', a') - Q(s,a)]$[14]. The value of α determines the speed of learning new information, and the value of γ means the importance of future rewards.

DQN has two mechanisms to make it act more like a human being: *1) Experience Replay.* Each time the DQN is updated, we can randomly extract some previous experiences stored in the experience base to learn. Randomly extracting disrupts the correlation between experiences and makes update process more efficient. *2) Fixed Q-target.* We use two NNs with the same structure but different parameters to predict the estimated and target Q-value, respectively. NN for the estimated Q-value has the latest parameter values, while for target Q-value, it has previous parameters. With these two mechanisms, DQN becomes more intelligent and makes the application of its selected features better.

Advantages of Random Forest for algorithm selection. Random Forest is chosen as the meta-learner to rank the algorithms and select the one that best fits the input task for two reasons. First, this model has high predictive ability for classification tasks and do not require many computational resources and complex hyperparameter tuning processes [15]. Second, Random Forest is sensitive to the internal influences among the complex meta-features selected by DQN when training.

b. Efficiency of Hyperparameter Optimization. Hyperparameter Optimization (HPO) algorithm aims to search optimal hyperparameter configurations that can maximize the performance of the algorithm selected by the meta-learner. Some hyperparameters with a wide range of values make the HPO space very complicated. Considering the variation and evolution mechanisms of *Genetic Algorithm* (GA) can approximate optimal solution with time constraints [16]. So, for the second challenge, a new HPO approach is designed based on it. Two mechanisms are used to ensure fast convergence of the GA in order to improve the efficiency of HPO process. a) *Potential Priority*. Before tuning, Auto-CASH measures the impact of each hyperparameter on performance improvement and then prune based on it. This allows prioritizing the limited time and computational resources for optimizing high-potential hyperparameters to obtain the highest performance improvement. b) *Fast-forward Initialization*. The Hyperparameter configuration of each algorithm in the experience base extracted from the papers is used as the initialization population of the GA instead of the random initialization population. The biggest advantage of this is to use prior experience to fast-forward the genetic process from the initial to the intermediate generations as a way to increase the convergence speed. Experimental results demonstrate that our approach can save a quarter of the time compared with existing work, which improves the efficiency significantly.

c. Automation of the entire approach. Auto-CASH employs a reinforcement policy and trains a network that *automatically selects the task features to be extracted and transforms them into meta-data*, which not only reduces human intervention but also takes into account the complex correlations among features. To the best of our knowledge, this is the first time that a reinforcement learning strategy is used for automatic feature engineering. In this way, feature selection, which used to rely heavily on expert experience, is turned into a task that automatically extracts features based on the user's input dataset. Fig. 1 is the brief flow chart of Auto-CASH. When users invoke our algorithm, they can only provide the data of tasks and candidate algorithms. Our approach automatically selects the best algorithm for the uploaded task in turn, and performs hyperparameter optimization to provide the user with a configured model.

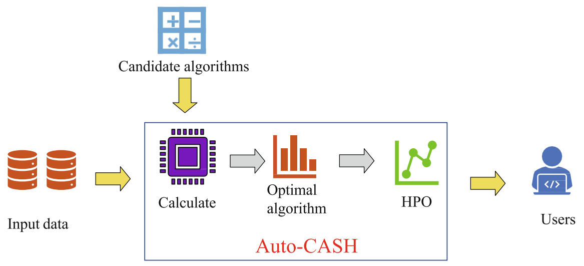


Fig. 1. Brief workflow of Auto-CASH.

Next is the introduction of the potential application scenarios for Auto-CASH. Firstly, the most possible scenario is for a user with little or no experience in classification algorithms who wants to quickly choose a better algorithm for his task, and Auto-CASH can eliminate some of his learning time for this expertise. Secondly, if a team or a user does not have enough time or human effort to make an algorithm selection, they can use our approach to save time. Thirdly, if the number of algorithms to be selected is too large, Auto-CASH can be applied for initial screening of algorithms to select a small range of high performance algorithms to ease the burden of subsequent algorithm selection.

The major contributions of this paper are summarized as follows:

1. **A novel approach to solve Combined Algorithm Selection and Hyperparameter Selection problem.** *Auto-CASH*, a meta-learning based approach for the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem, is proposed. It represents the task as a feature vector and model the algorithm selection experience as the mapping from analysis task to its optimal process algorithm. A meta-learner is trained with the knowledge to identify high-performing algorithms for solving new tasks. By sufficiently utilizing the extracted prior experience, our approach automatically selects the optimal algorithm for a new task, while being very efficient and lightweight in practice. Then a genetic search with our two prune strategies is applied for configuring the hyperparameters for selected algorithm before.
2. **A new algorithm for automatic feature selection.** Meta-data plays a crucial role in the field of meta-learning, and the selection and processing of meta-data has a great impact on the final results. In practice, automatically selection of meta-features can lead to efficient and intelligent methods. Auto-CASH employs a reinforced policy and trains a network that *automatically selects the task features to be extracted and transforms them into meta-data*, which not only reduces human intervention but also takes into account the complex correlations among features. To the best of our knowledge, this is the first study that combines reinforcement learning policy and meta-learning to solve CASH.
3. **Improve the efficiency of Combined Algorithm Selection and Hyperparameter Selection problem solving from multiple perspectives.** First, workflow in Auto-CASH reduces the search space, which is the most critical part of the efficiency improvement. Suppose that there are m algorithms to be selected, and each algorithm has n hyperparameters with p values. Existing work have a $m \cdot n \cdot p$ search space while it is reduced to $m + n \cdot p$ in ours. Accordingly, the complexity of the approach is reduced from $O(m \cdot n \cdot p)$ to $O(m + n \cdot p)$. Second, the HPO process is the most time consuming process, we propose two empirical-based optimization strategies, i.e. Potential Priority and Fast-forward Initialization, to improve the optimization efficiency of the genetic algorithm and the overall efficiency of solving CASH. We believe that limited time should be prioritized to tune parameters with higher potential. Before tuning, we *measure the impact of each hyperparameter* on performance improvement and then prune the hyperparameter search space based on it. Then we perform genetic search with the *hyperparameter configuration of the historical experience closest to the new task*. With these two strategies, the complexity of our approach is reduced from $O(m + n \cdot p)$ to $O(m + n' \cdot p')$, where $n' < n, p' < p$.
4. **Performance validation.** Extensive experiments are conducted on classification tasks, and the results show that our method outperforms existing AutoML approaches with the efficiency significantly improved.

The structure of this paper is as follows. Section 2 analyzes related work and introduces concepts about HPO. Section 3 describes the workflow and some implementations of our model. Section 4 introduces the methodology for automatically meta-feature selection. We evaluates the performance of Auto-CASH and compares it with early work in Section 5. Finally, we draw a conclusion and give our future research directions in Section 6.

2. Related work

In this section, we will introduce the latest studies on automatic algorithm selection and hyperparameter optimization, respectively.

2.1. Automatic algorithm selection

There has been an increasing trend to apply AutoML approaches [2] to automate algorithm selection [15]. Existing approaches are typically based on Bayesian Optimization [11], tree-based classifier [17], Deep RL [18], and Genetic Algorithm

[19]. Based on the Bayesian Optimization framework SMAC, Auto-WEKA [4] is one of the first approaches to automated CASH, which converts the CASH problem into an HPO problem. However, Auto-WEKA simultaneously optimizes hyperparameters for 40 ML algorithm models, suffering from high time and space cost. It is inefficient to choose only one algorithm as the optimal one in the end, although it takes a long time for all these algorithms to be tuned. In the search and verification process, although some algorithms have similar performance to the optimal algorithm, only the algorithm with the best performance is selected in Auto-WEKA, which leads to a waste of a valuable part of the verification process. To avoid this situation, Auto-sklearn [8] combines some algorithms with similar performance into an optimal model by assigning weights to them. Auto-sklearn is based on the implementation of scikit-learn library, so the algorithm list is not as abundant as Auto-WEKA, but the combination of models with different weights makes a variety of model combinations to be chosen in the end. Auto-sklearn tries to draw on some existing validation results, and similar ideas are worthy of more widespread application. Auto-Model [6] leverages experimental results of published ML papers to create a knowledge base, for priority relationship among candidate algorithms. It is not an ideally automated framework, since heavy human labor is required to rate these papers. Previous work has conducted in-depth research on how to extract experience manually, but neglected how to automatically convert experience into data that can be used for training. Our approach solves this defect through the meta-data.

There is also a category of approaches based on meta-learning that considers the role of experience and meta-features to guide the task in the present moment. Existing meta-learning based studies for automatic algorithm selection more concentrate on expressing relationships between algorithm performance and tasks characteristics [20]. Therefore, a great number of meta-features have been employed by existing works, which can be divided into three categories:

1. *Statistical information* meta-feature is the most intuitive way to describe datasets. These meta-features involves the number of records, mean, variance, entropy, etc. All of them have formulae and proven theoretical implications. They are Efficient, easy to calculate and intuitive, which are the biggest advantages.
2. *Hyperparameter setting* meta-feature is used to represent characteristics of the model applied to datasets. For example, the number of max depth of random forest, the learning rate, etc.
3. *Evaluation* meta-feature is based on estimated performance of a model on the test datasets, e.g. time, accuracy, AUC, etc. They are usually more expensive to acquire, since some complex models cannot be easily evaluated.

AutoGRD [17] is very innovative in proposing a new meta-feature, which extracts graphical meta-features from a graph-based representation for dataset. However, graph-based representation of the relationships between tasks (datasets) is not straightforward, and the measures are difficult to define. Auto-Di [21] combines dataset-based and embedding-based meta-features to train a meta-learner. After training, it can estimate and rank a list of algorithms on specific datasets. These works contributing to meta-feature extraction and generation have led to a wider range of meta-feature selection as well.

Automatic algorithm selection also plays an important role in the database field, and in the last two years some methods have emerged to optimize and build intelligent databases. Oracle first proposed a complete CASH system and brought it online named Oracle AutoML [22], adding it to their paid database system application. In their approach they use parallel processing to evaluate different algorithms and hyperparameters optimization for the task, and update the algorithms and hyperparameters configuration through communication among pipelines. Such an approach is not complicated, but we believe that it requires Oracle's powerful servers and mature parallel technology and environment to be stable and efficient, and it is expensive to use. AutoSmart [23] is the first automated process in the database field, including data processing, model selection, table merge and other operations. This is also the latest result in the field up to the time of writing this paper. However, we believe that there is still room for optimization by combining experience to accelerate.

2.2. Hyperparameter optimization techniques

The problem of HPO has a long history, dating back to almost last century [24]. This problem aims at optimizing some specific parameters related to the model in order to improve performance of the model and reproducibility of the work. In this section, we will give definition of HPO problem and introduce some existing studies, respectively.

Let $\mathcal{A}$ and $\mathcal{D}$ denote a learning algorithm with n hyperparameters $\lambda_1, \lambda_2, \ldots \lambda_n$ and a dataset, respectively. The domain of λ_i is denoted by Λ_i. So the overall hyperparameter space Λ is a subset of Cartesian product of these domains: $\Lambda \subseteq \Lambda_1 \times \Lambda_2 \times \ldots \Lambda_n$. Given a score function $\mathcal{F}(\mathcal{A}, \mathcal{D})$, the HPO problem can be written as Eq. (2), where $\mathcal{A}_\lambda$ means algorithm $\mathcal{A}$ with a hyperparameter configuration λ ($\lambda \in \Lambda$).

$$\lambda^* = \arg\max_{\lambda \in \Lambda} \mathcal{F}(\mathcal{A}_\lambda, \mathcal{D}) \tag{2}$$

Some approaches are proposed, including Grid Search [25], Random Search [26], Hyperband[27], Bayesian Optimization [28], and Genetic Algorithm[29].

Grid Search is the most basic HPO approach. Users define a finite set of values for each hyperparameter, then grid search can evaluate the Cartesian product of these sets. Obviously, this approach is strongly influenced by dimensions of the hyperparameters, and the search space is geometrically expanded for each additional hyperparameter. So an intuitive improvement is applying random combinations of hyperparameters to replace the Cartesian product, which is named Random Search. But this method is not stable, because no one can guarantee a random result.

Therefore, some methods that can predict the performance in the search process are proposed. Learning curve-based HPO approach more focus on the predictive termination, where a continuous learning curve is used to represent the performance trend during optimization, and the HPO process stops if the configuration is predicted to not reach the expected value. Usually a search result is evaluated using the weight values of multiple metrics to make a comprehensive evaluation. Freeze–thaw Bayesian optimization [30] is a typical representative of this type of approach. While it is limited by no sharing information among evaluated configurations and the new one, in other words no leveraging of assessment experience.

A more intuitive and theoretically based HPO method based on Bayesian theory was proposed, which computes and explores the hyperparameter distribution of possible higher performance gains through Bayesian distribution. Bayesian Optimization (BO)[31] is a black-box global optimization approach that almost has the best performance among the above-mentioned HPO approaches. It uses a surrogate model to fit the target function, then predict the distribution of the surrogate model based on Bayesian theory iterative. However, it is time-consuming to explore the surrogate model using Bayesian theory and historical data. When encountering a model with high complexity, although BO can provide the optimal HPO results, the execution time is hardly unacceptable. Kirthevasan et al. [32] optimizes these shortcomings, while it has little effect on data sets with large amounts of data.

Genetic Algorithm (GA)[19] adopts an evolutionary optimization method to accelerate convergence. It is a stochastic global search and optimization method developed by imitating the biological evolution mechanism of nature. GA works by encoding hyperparameters configuration and random initializing population, then iteratively produces the next generation through selection, crossover and mutation steps. The iteration stops when one of the stopping criteria is met, and finally the optimal individual (i.e., configuration) is treated as the solution for HPO problem. GA sometimes has difficulty in controlling the convergence rate, resulting in easy to fall into local optimal solutions or take a long search time. Sherpa [33] is the latest HPO toolkit so far, it implements the HPO algorithms mentioned above, but the improvement effect for them is not good enough. While in this paper we propose two mechanisms based on experience to resolve this issue.

Literature Summary. The existing work in the area of algorithm selection has the following drawbacks.

1) The algorithm has a long running time and low efficiency due to the large search space. The running time is mainly limited by the search space, so here we analyze the space complexity of feasible solutions. Suppose that there are m algorithms to be selected, and each algorithm has n hyperparameters with p values. Apparently the search space of Auto-WEKA is $m \cdot n \cdot p$. Auto-sklearn takes a smart approach to reduce the complexity of the search by reducing the number of candidate algorithms, but the complexity is difficult to evaluate because of the need to constantly measure the combination of weight coefficients. However, from the actual use of the results, the complexity is not reduced.

2) The use of existing approaches requires more human expertise in the process. For example, Auto-Model uses papers to build a knowledge base, but the papers used need to be continuously updated to ensure the excellent performance of the knowledge base, and how to measure the papers used to build the knowledge base is extremely demanding human expertise, which is difficult to decide even for some researchers.

3) The collection and use of historical experience is neglected. Work in the field of HPO also faces the problem of longer running times due to the more complex search space. Since most HPO algorithms are randomized and cannot analyze the time complexity, it is difficult to analyze and optimize them from the perspective of computational complexity. Our work investigates and optimizes the above defects, and the details are shown in Section 3 and 4.

3. Approach overview

In this section, a formulaic definition of the problem in this study and introduction of the workflow in Auto-CASH are described. Then the criterion used in Auto-CASH and its advantage are discussed. In the end, the implementations of algorithm selection and GA-based HPO are shown.

3.1. Problem definition

Definition 1 (*Combined Algorithm Selection and Hyperparameter Optimization problem*) Given a new task T_{new}, our goal is to find an algorithm with a tuned hyperparameter configuration that can have maximal performance. It can be expressed as Eq. 3, where the $\mathcal{V}(\mathcal{L}, \mathcal{A}_\lambda, T_{old}, T_{new})$ measures the performance of a model generated by algorithm $\mathcal{A}$ with hyperparameters λ leveraging experience observed from previous tasks T_{old} and evaluated on T_{new}. In the real world, we can only have finite tasks $(T_{old}, T_{new}) \sim \mathcal{T}$ for experience extracting and training, thus we need try to approximate the expectation.

$$(\mathcal{A}, \lambda)^* = \arg\max_{\lambda \in \Lambda} \mathbb{E}_{(T_{old}, T_{new}) \sim \mathcal{T}} \mathcal{V}(\mathcal{L}, \mathcal{A}_\lambda, T_{old}, T_{new}) \tag{3}$$

3.2. Workflow

As described by [17], current CASH approaches mostly suffer from high time cost [34]. On the one hand, they merge the hyperparameter space for all candidate algorithms and optimize them simultaneously, which is a high dimensional search space. The complex space also makes the method easy to fall into a locally optimal solution if there is no enough time. On the other hand, the optimization strategy is based on Bayesian Optimization [8], a method that has been illustrated to be less efficient in high dimensional space [35].

Algorithm 1: Auto-CASH approach.

```
Input: The training datasets $D_{train}$; The candidate algorithm set $Alg$; The datasets needs for autonomous algorithm
   selection and HPO $D$;
Output: An optimal algorithm $alg$ and its hyperparameter setting $\lambda_{alg}$;
1: for all $\mathcal{D} \in D_{train}$ do
2:     $\mathcal{A}^* = \arg\max_{\mathcal{A}\in alg} F_{score}(\mathcal{A}, \mathcal{D})$
3:     Experience base $\leftarrow (\mathcal{A}^*, \mathcal{D})$
4: end for
5: Select $M_{list}$ according to $D_{train}$          ▹ Refer to Algorithm 2
6: for all $\mathcal{E} \in$ Experience do
7:     $\mathcal{E} + M_{list} \rightarrow$ meta-vector $V_{\mathcal{E}}$
8:     $V_{all} \leftarrow V_{all} \cup V_{\mathcal{E}}$
9: end for          ▹ Leverage selected meta-features to convert experience to meta-data.
10: Initialize a Random Forest $rf$          ▹ Meta-learner
11: $rf$.fit($V_{all}$)
12: $alg \leftarrow rf$.predict($T_{New}$)          ▹ The trained meta-learner is used to automatic algorithm selection
13: for all $\lambda \in \Lambda$ of alg do          ▹ Potential Priority
14:     $\Lambda^* \leftarrow$Top-K potential $\lambda$
15: end for
16: Initialize $\Lambda^*$ with Experience base          ▹ Fast-forward Initialization
17: $\lambda_{alg} \leftarrow GeneticSearch(alg, \Lambda^*)$          ▹ Refer to Section 3.5
18: return $alg, \lambda_{alg}$;
```

In order to improve the efficiency of CASH, an increasingly popular trend is to select algorithms first, and then optimize hyperparameters [34]. Table 1 shows experimental results for Auto-WEKA and Auto-Model on 3 datasets. We use Auto-WEKA and Auto-Model to perform algorithm selection for the datasets in the table, output the currently selected algorithms at different time points, and finally summarize them into a table. The experimental results are very informative, and the final algorithm is fixed in almost 2 min, which means most time is used for HPO. If the two steps discussed above can be optimized separately, the time cost will be reduced apparently. Therefore, we determine to accelerate algorithm selection based on meta-learning. Then HPO technique is used to optimize selected algorithm performance. There are two main advantages: (a) **The search space is reduced significantly**. Suppose there are m algorithms to be selected, and each algorithm has n hyperparameters with p values. Auto-WEKA has a $m \cdot n \cdot p$ search space while it is $m + n \cdot p$ in Auto-CASH. (b) The algorithm selected by the **meta-learner making full use of knowledge** can improve the lower limit of the result performance.

The workflow and pseudo code of Auto-CASH is shown in Fig. 2 and Algorithm 1, respectively. The whole workflow is divided into two phases, the experience extraction and offline working phase. The trained meta-learner can handle new tasks, and our HPO approach makes sure maximize the speed of genetic search convergence. Specifically, firstly in the experience base building phase, Auto-CASH extracts the experiences from previous tasks and training the meta-learner for the next step. Then the RF trained with previous experience selects an optimal algorithm for new tasks. At last, HPO is executed to search the optimal hyperparameter configuration for the algorithm selected before.

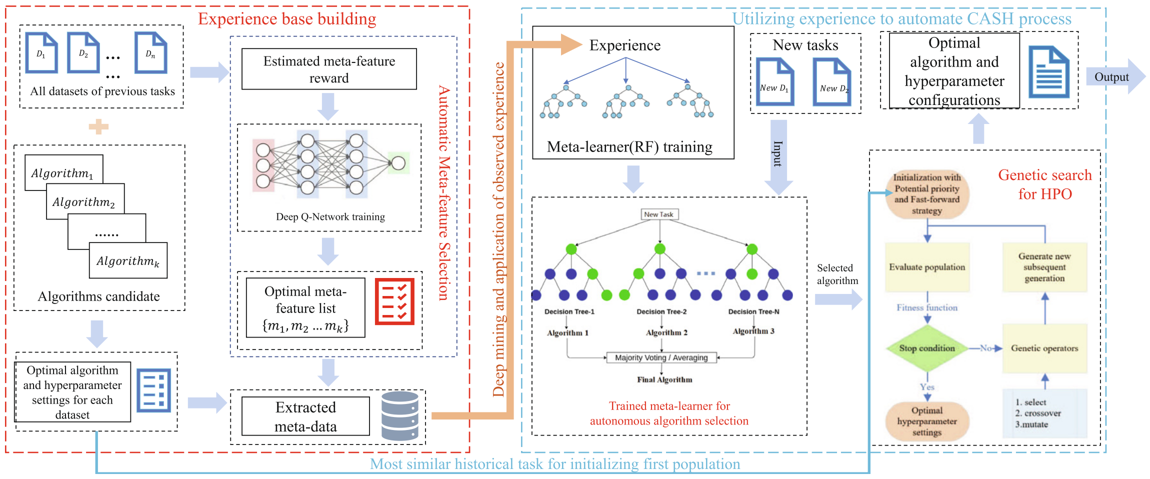
Analysis of Algorithm 1 line by line are as follows. First of all, from Line 1 to 4 Auto-CASH chooses the optimal algorithm with our new metric criterion shown in Section 3.3. In the next phase, we employ DQN to automatically determine the meta-features for representing datasets, as shown in Line 5. Then the experience we observed are converted to the meta-data, as shown from Line 6 to 9. A random forest model are initialized as the surrogate model of a meta-learner, which will fit and train on the meta-data. Given a meta-feature vector of a new task, the trained meta-learner can predict the label for it, as described in Line 12 and Section 3.4. At last, in Line 17, we apply GA-based approach described in Section 3.5 to search for the optimal hyperparameter configuration.

3.3. Metric criterion

For classification algorithms, the most commonly used metric criterion is accuracy. However, accuracy is highly influenced by the test-train data set splitting. Thus, a more balanced metric criterion should be considered from multiple perspectives. AUC is the area under the ROC [36] curve, which represents the classifier's ability to classify positive and negative examples [37]. The accuracy and AUC usually turn out to be conflicted on an unbalanced dataset. For example, in a cancer dataset, there may be only 1% of cancer records(negative cases). If all records are classified as positive cases, the accuracy value is 0.99, but the AUC value is only 0.5. Therefore, optimizing both accuracy and AUC can be treated as a multi-

Table 1
Algorithms selected in different time limit. The final algorithm is fixed in a subset very early.

Dataset	Approach	1 min	2 min	4 min	8 min
Audiology	Auto-WEKA	LMT	KStar	KStar	KStar
	Auto-Model	LMT	KStar	KStar	KStar
Mammo	Auto-WEKA	DecisionTable	RandomTree	RandomTree	RandomTree
	Auto-Model	RandomTree	RandomForest	RandomTree	RandomForest
Balance-scale	Auto-WEKA	LMT	LWL	LWL	LWL
	Auto-Model	LMT	LMT	LWL	LWL



Experience base building
All datasets of previous tasks
Algorithms candidate
Optimal algorithm and hyperparameter settings for each dataset
Estimated meta-feature reward
Deep Q-Network training
Optimal meta-feature list $\{m_1, m_2 \dots m_k\}$
Extracted meta-data
Automatic Meta-feature Selection
Deep mining and application of observed experience
Utilizing experience to automate CASH process
Experience
Meta-learner(RF) training
New tasks
Input
Selected algorithm
Trained meta-learner for autonomous algorithm selection
Majority Voting / Averaging
Final Algorithm
Optimal algorithm and hyperparameter configurations
Output
Genetic search for HPO
Initialization with Potential priority and Fast-forward strategy
Evaluate population
Generate new subsequent generation
Fitness function
Stop condition
No
Yes
Genetic operators
1. select 2. crossover 3.mutate
Optimal hyperparameter settings
Most similar historical task for initializing first population

Fig. 2. The Auto-CASH workflow. With training based on the observed experience extracted from new tasks, our approach can work offline when it is trained comprehensively. Both in the training of the meta-learner and the initialization phase of the genetic search, we make full use of experience to improve efficiency.

objective optimization problem. A classic multi-objective optimization method [38] is weighted sum, represented as $F_{score} = w_1 \cdot accuracy + w_2 \cdot AUC$ in our problem. While these two are important indicators for evaluating a model from different perspectives, more complicated calculations are needed for computing reasonable weight coefficients. Therefore, we adopt a concise way to represent the score function of Auto-CASH, shown as Eq. (4).

$$F_{score} = accuracy \cdot AUC \tag{4}$$

3.4. Automatic algorithm selection

Algorithm selection is based on the meta-learning theory [9], a subfield of AutoML, aims to learn from prior task experience to reduce computation for similar task [39]. It can be divided into 3 steps.

1. The first step is extracting the meta-features(such as, class entropy, variance) of training datasets. Different from classical meta-learning frameworks, our approach automatically determines the used meta-features through DQN (refer to Section 4). Each task $t_j \in \mathcal{T}$ is described with a meta-feature vector $m(t_j) = (m_{j,1}, \dots m_{j,k})$ of k meta-features. This can be used to measure similarity, for instance, L1 distance among tasks. Thus information from the most similar tasks can be transferred to new task.
2. Then for each previous task, meta-feature vector and its optimal algorithm are collected as meta-data. Assume that there are totally n previous tasks and k selected meta-features, the finally meta-data is described as $\mathbb{E}_{n\times(k+1)}$. The $k+1^{th}$ variable represents its optimal algorithm. Meta-data is regarded as the experience and the basis of meta-learner training.
3. Finally, the trained meta-learner will recommend an algorithm for a new task according to its extracted meta-feature vector. In this way, our work reduces human labor by extracting and applying the experience through meta-learning.

A RF model serves as the surrogate model of meta-learner, which is blessed with three advantages. (a) We use some complex meta-features to represent the datasets. RF is sensitive to the internal influences among these meta-features when training. (b) RF has a high prediction accuracy after a series of experiments [15]. (c) Training a RF needs less human labor than Auto-Model [6], which has to evaluate and extract rules in published papers. On this basis, the autonomous algorithm selection only cost a few seconds.

3.5. Hyperparameter optimization

Some hyperparameters with a wide range of values make the HPO space very complicated. Under this situation, Genetic Algorithm can obtain better results within a shorter time [16], so Genetic-Algorithm-based HPO is designed. However it still has some drawbacks that lead to less convergence in the case of very complex data sets. Auto-CASH employs two optimization strategies that can reduce search space of GA leveraging experience and some estimates.

3.5.1. Potential priority

First, not all hyperparameters have the same influence on the final optimization results [26], so reducing the dimension of hyperparameters is the first step. Auto-CASH chooses the hyperparameters based on the performance improvement after tuning. According to the Occam's razor principle, in order to reduce the complexity of algorithm, we only select the hyperparameters that will bring a relatively large effect improvement for tuning. This kind of pruning will minimize the feasible solution search space and make GA converge more efficiently.

Let $\mathcal{A}$ and $\mathcal{D}_{val}$ denote a learning algorithm with n hyperparameters $\lambda_1, \lambda_2, \dots \lambda_n$ and validation dataset, respectively. Given a performance improvement function $I(\mathcal{A}_{\lambda_i}, \mathcal{D}_{val})$, the pruning can be formulated as Eq. 5. θ is the threshold we set, which is 0.3 times the maximum of performance improvement. $\mathcal{A}_{\lambda_i}$ denotes algorithm $\mathcal{A}$ with an optimized hyperparameter λ_i.

$$\begin{aligned} \lambda_{list} &= \{\lambda_i | i \in [1, n], I(\mathcal{A}_{\lambda_i}, \mathcal{D}_{val}) > \theta\} \\ \theta &= 0.3 \cdot \max_{i \in [1,n]} I(\mathcal{A}_{\lambda_i}, \mathcal{D}_{val}) \end{aligned} \tag{5}$$

3.5.2. Fast-forward initialization

After selecting more "crucial" hyperparameters, we want to reduce the search space even further. So it is natural to think of reducing the search range of each hyperparameter. In our experience base, we not only record the optimal algorithms for each dataset, but also the hyperparameter configurations of these algorithms. To obtain such optimal configuration, some analysis and discussion are given as follows.

Definition 2 (*δ-k-neighborhood*): Given a Genetic Algorithm GA with hyperparameter space Λ, a ratio bound δ and a given original hyperparameter λ_0, $GA^k(\lambda_0)$ represents the random variable $\lambda \in \Lambda$ after k generations of GA. Thus, the δ-k-neighborhood of λ_0 is defined as follows.

$$N(\lambda_0, \delta, k) = \{\lambda | P\{GA^k(\lambda_0) = \lambda\} > \delta\} \tag{6}$$

$N(\lambda_{\mathcal{T}_{new}}, k, \delta)$ is a neighborhood filled with values from where can we reach the optimal solution after k generations with possibility of δ. Therefore, given two possiblity bound $\delta_1 \leqslant \delta_2$, and two numbers of generations $k_1 \leqslant k_2$, we have:

$$\begin{aligned} &N(\lambda_{\mathcal{T}_{new}}, k, \delta_1) \subset N(\lambda_{\mathcal{T}_{new}}, k, \delta_2) \\ &N(\lambda_{\mathcal{T}_{new}}, k_1, \delta) \subset N(\lambda_{\mathcal{T}_{new}}, k_2, \delta) \end{aligned} \tag{7}$$

Our observation is that the hyperparameter settings are similar for the same algorithm for different tasks. We assume that there is a sweet spot for each algorithm and the hyperparameter combinations within this region are more widely adaptable. For a target task $\mathcal{T}_{new}$, a task $\mathcal{T}_1$ in observed experiences who share the same candidate algorithm with $\mathcal{T}_{new}$, and the number of generation k, our assumption can be demonstrated with the following equation:

$$\delta_0 = min_{\delta}\{\lambda_1 \in N(\lambda_{new}, k, \delta)\} \propto sim(\mathcal{T}_1, \mathcal{T}_{new}) \tag{8}$$

where $sim(\mathcal{T}_1, \mathcal{T}_{new})$ denotes the similarity between tasks, λ_{new} represents the optimal hyperparameter configuration for $\mathcal{T}_{new}$.

To accelerate the convergence of GA, our goal is to find an original hyperparameter λ_0 where:

$$\begin{aligned} \lambda_0 &= \arg\max min_{\delta}\{\lambda_1 \in N(\lambda_{new}, k, \delta)\} \\ &= \arg\max sim(\mathcal{T}_1, \mathcal{T}_{new}) \\ &= \arg\min dis(\lambda_1, \lambda_{new}) \end{aligned} \tag{9}$$

Based on the discussion above, if the primordial population can be initialized from this region, the process of random initialization followed by evolution to reach here can be omitted. In other words, Auto-CASH selects the "closest" (similar) historical experience as the initialization of GA by calculating the L2 distance between the meta-feature vectors representing different tasks, which is another way to leverage the experiences that we worked hard to gain. Effect of Fast-forward Initial-

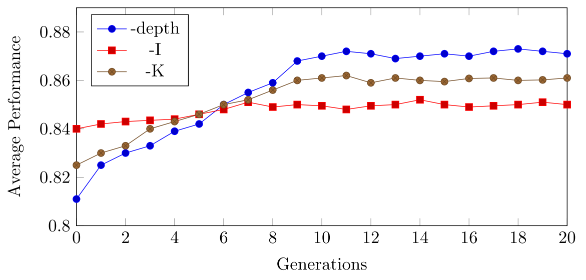


Fig. 3. Performance improvement for tuning three hyperparameters of Random Forest, respectively. All reach convergence within 20 generations, even 15 generations.

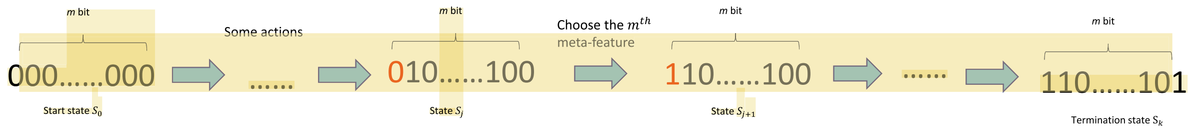


Fig. 4. Transition between states examples.

ization is demonstrated on algorithm Random Forest[3] with all training datasets and show the experimental result in Fig. 3. Genetic search is performed on each of the three hyperparameters and the acceleration was perfect, all reaching the performance optimum within 15 generations. Experimental results in [40] also illustrate this strategy significantly speed up convergence when tuning for SVM. The optimal hyperparameter combination was found within almost 10 generations.

Above all, the HPO process can be summarized in the following four steps:

1. In the beginning, we utilize binary code to encode hyperparameters and initializes the original population with our two optimization strategies.
2. Then, the batch of individuals with the best fitness are selected [4] for next generation.
3. Two binary sequences (individuals) randomly exchange their sub sequences in the same position to show the crossover process. And the binary digits of individuals alters randomly as mutation.
4. For each subsequent generation, the hyperparameter configuration is returned as the HPO result if the termination condition has been reached; otherwise, the above steps will be iteratively executed.

4. Automatic meta-feature selection

The major challenge of meta-learning is the quality of selected meta-features [41]. Due to the fact that meta-features have complicated correlation between each other, it is difficult to determine which to be used based on human expertise. An ideal scenario is to train a model to automatically select the appropriate meta-features from a set to be selected based on the characteristics of the dataset. A well-studied approach focusing on the influence of multiple candidate selection is DQN [12]. The key is how to transform meta-feature selection into the automatic continuous decision problem that DQN outperforms. This goal can only be achieved through proper definition of the state, action, reward, and transition between states in the DQN environment. It is also a major contribution of our work. In the rest of this section, we will introduce the methodology of DQN environment construction and details of selection process.

Given a collection of candidate meta-features $MF(|MF| = m)$, the **state** s is the meta-feature selected from MF. Each **action** a selects a specific meta-feature $mf \in MF$. The eventually selected meta-features form an optimal meta-feature list $M_{list}(M_{list} \subseteq MF, |M_{list}|_{max} = n, n < m)$. The **reward** r_a of action a is the probability of selecting the optimal algorithm by performing action a.

According to above stipulation, we adopt an m-bit binary number to encode all states. Each bit represents a meta-feature in MF. In a specific state s, if the meta-feature mf is selected, its corresponding bit is encoded as 1. Otherwise, it is encoded as 0. Thus, there are totally 2^m states and m actions. The example in Fig. 4 can explain the transition among states more clearly. Under the start state S_0, no meta-feature is selected, so all m bits are 0. After performing some actions, the state is S_j. The next action is to choose mf_m, so the mth bit of the number is set 1. These steps will be repeated until the termination state.

[3] The algorithm is implemented by WEKA. '-K' denotes number of attributes to randomly investigate. '-I' denotes number of trees in the random forest. '-depth' denotes the maximum depth of the tree.

[4] In our approach, fitness means the algorithm performance with specific hyperparameter configuration

Algorithm 2: Automatic meta-feature selection approach.

```
Input: The meta-feature candidates list $MF(|MF| = m)$; The limit of optimal meta-feature list $|M_{list}|_{max} = n(n < m)$; The
  limit of episode $e_{max}$;
Output: The optimal meta-feature list $M_{list}$;
1: Construct state set $S$ and action set $A$;
2: Estimate reward $R$ of each $a \in A$;
3: for $i = 1; i < e_{max}; i++$ do
4:   Initialize $M_{list} = \emptyset$ and all candidate $M_{list}$ set $M_{all} = \emptyset$;
5:   Start state = $s_0$, current state $s = s_0$;
6:   while $|M_{list}| < n$ do
7:     Initialize the DQN using $S, A$, and $R$;
8:     Use DQN to find the optimal action $a_{best}$ for $s$;
9:     $M_{list} \leftarrow M_{list} \cup a_{best}$;
10:      $s \leftarrow s$ perform $a_{best}$;
11:   end while
12:   $M_{all} \leftarrow M_{all} \cup M_{list}$;
13: end for
14: return $M_{list} = argmax(M_{all})$;
```

All meta-feature selection steps are summarized in Algorithm 3. At first, as shown in Line 1, we construct the state set S and action set A. Then we estimate the reward of each action R, which is shown in Line 2. The DQN environment is initialized as (S, A, R). For each training iteration, DQN starts from S_0, and chooses the maximal reward action in each next step (Line 4–10). After decoding the termination state, the training results are obtained. We repeat above steps and eventually obtain the optimal meta-feature list from numerous training results(Line 14).

Compared with traditional feature selection methods(filter-based [42], wrapper-based [43]), the most significant advantage of DQN is that it will continuously improve the selection results according to the reinforced policy. At start, the lack of experience makes the selection of DQN have a deviation from reality. As the training progresses, DQN will adjust its parameters such as learning rate and discount rate according to the deviation and make next selection more reasonable. Eventually, the network parameters become stable, and the selected meta-features achieve optimal effect. This progressive learning mechanism optimizes the selection result. Experimental results demonstrate the effectiveness of our proposed approach and are shown in Section 5.

5. Evaluation

To verify the performance of the proposed approach, extensive experiments on classification tasks are conducted and the results are shown as follows.

5.1. Experimental setup

For all experiments in this paper, the setup is as follows:

1. **Datasets** All datasets used are real-world datasets from UCI Machine Learning Repository and Kaggle. The datasets cover fields such as biology, medicine, classification of common objects in life, statistics of natural phenomena. The most significant advantage of using real-world datasets is that it can improve the practicality of our model in real life and lay the foundation for future research. Data list is summarized in Table 2.
2. **Evaluation** 10-fold cross validation accuracy, AUC, and relative maximum value (RMV) are used as the indicator to evaluate performance of classification algorithms on datasets. RMV is the ratio between result's capability to the best capability achieved by evaluated approach on the same dataset.
3. **Baselines** This part compares Auto-CASH to four baselines, which can be divided into two categories. (a) Optimized approaches to the CASH problem. Auto-WEKA: a framework transforms CASH into a hierarchical HPO problem, in which the selection of algorithms is recognized as a hyperparameter. Auto-Model [6], utilizing a knowledge base to store some experiences. (b) Popular AutoML systems. Auto-sklearn [8]: based on Bayesian Optimization to give a weighted mixture of algorithms. TPOT [19]: a genetic machine learning pipeline designer. Compared with another type of baseline, the biggest disadvantage of the AutoML system is that they ignore the optimization of time cost.
4. **Performance convergence and iteration limits** When the performance fluctuates less than 0.1% for five consecutive iterations, it is considered to have converged. To ensure efficient execution, the maximum number of iterations of the GA for HPO is 50.

Table 2
All datasets used in our experiments.

nursery	squash-unstored	anneal.ORIG	credit-a	eucalyptus
optdigits	unbalanced	arrhythmia	credit-g	eye_movements
page-blocks	vehicle	audiology	cylinder-bands	glass
pasture	vote	australian	dermatology	grub-damage
pendigits	vowel	autos	diabetes	heart-c
postoperative	waveform-5000	balance-scale	ecoli	heart-h
segment-challenge	weather.nominal	bank	lung-cancer	heart-statlog
segment-test	weather.numeric	breast-cancer	mbagrade	hepatitis
segment	white-clover	breast-w	molecular-biology	hypothyroid
sick	wine	bridges_version1	monks-1_test	ionosphere
sonar	zoo	bridges_version2	monks-1_train	iris.2D
soybean	schlvote	car	monks-2_test	iris
spambase	solar-flare_1	cmc	monks-2_train	kr-vs-kp
spectf_test	solar-flare_2	colic	monks-3_test	labor
squash-stored	anneal	contact-lenses	monks-3_train	landsat_test
Acut	EEG Eye	Yeast	Balloon	MAGIC Gamma
Cardiotocography	ISOLET	abalone	primary-tumor	mushroom
Hayes-Roth	mfeat-karhunen	MEU-Mobile KSD	Musk2	Mammo(graphic)
landsat_train	mfeat-morphological	blood	Pittsburghv1_material	Teaching assistant
letter	molecular	Wall following	Pittsburghv1_type	climate
liver-disorders	Flag	Hepatitis	Pittsburghv2_type	work
lymph	Wholesale	Cylinder bands	Bupa	Avila
mfeat-factors	Crowdsourced Mapping	Echocardiogram	Vertebral	parkinsons
mfeat-fourier	Lenses	Musk1	relax	Shuttle

Table 3
For Random Forest, the performance improvement after tuning the three hyperparameters respectively.

hyperparameter	$-K$	$-depth$	$-I$
Improvement(%)	3.5($\pm$ 0.1)	6($\pm$ 0.1)	1.1($\pm$ 0.1)

Referring to the methodology in Section 3.5, we first test the performance improvement of hyperparameters for each *alg*. We use GA to tune hyperparameter $-K$, $-depth$, and $-I$, respectively for *alg* Random Forest. The effect of each parameter on the final performance improvement is different, which is shown in Table 3. After tuning $-depth$, we can achieve a 6 percent improvement, while $-I$ can only improve 1.1 percent(lower than $\theta = 0.3 \times 6 = 1.8$). Thus for *alg* RF, we decide to tune $-depth$ and $-K$ in HPO. We run the above process for each candidate algorithm to determine the hyperparameters to be tuned. All 23 candidate algorithms and their categories are summarized in Table 4. To ensure the fairness of the performance, we use the algorithm implemented in WEKA, which is an open source software. The source code can refer to https://svn.cms.waikato.ac.nz/svn/weka/branches/stable-3-8/. We wrap the jar package and invoke it using Python.

The candidate meta-feature set also plays a crucial role, which is supposed to capture various aspects of the datasets and easy to calculate. Our work adapts dataset-based statistical meta-features because they can intuitively describe the features of datasets. Refer to works of Wang et al. [6], eventually candidate meta-features are summarized as follows. These meta-features are easy to calculate during algorithm selection.

- mf_0: Class number in target attribute.
- mf_1: Class information entropy of target attribute.
- mf_2: Maximum proportion of single class in target attribute.
- mf_3: Minimum proportion of single class in target attribute.
- mf_4: Number of numeric attributes.
- mf_5: Number of category attributes.
- mf_6: Proportion of numeric attributes.
- mf_7: Total number of attributes.
- mf_8: Records number in the dataset.
- mf_9: Class number in category attribute with the least classes.
- mf_{10}: Class information entropy in category attribute with the least classes.
- mf_{11}: Maximum proportion of single class in category attribute with the least classes.
- mf_{12}: Minimum proportion of single class in category attribute with the least classes.
- mf_{13}: Class number in category attribute with the most classes.
- mf_{14}: Class information entropy in category attribute with the most classes.
- mf_{15}: Maximum proportion of single class in category attribute with the most classes.
- mf_{16}: Minimum proportion of single class in category attribute with the most classes.

Table 4
All classification algorithms in Auto-CASH. To ensure the fairness of the performance, we use the code implemented in WEKA, which is an open source software.

Category	Algorithm
lazy	IBK, KStar, LWL
meta	LogitBoost, RandomSubSpace, RandomCommittee, MultiClassCls, AttributeSelectedCls, Vote, ClassificationViaRegression, Bagging, AdaBoost
trees	J48, LMT, RandomForest, RandomTree
rules	JRip, DecisionTable
bayes	NaiveBayes, BayesNet
functions	Logistic, MultilayerPerceptron, SMO

- mf_{17}: Minimum average value in numeric attributes.
- mf_{18}: Maximum average value in numeric attributes.
- mf_{19}: Minimum variance in numeric attributes.
- mf_{20}: Maximum variance in numeric attributes.
- mf_{21}: Variance of average value in numeric attributes.
- mf_{22}: Variance of variance in numeric attributes.

5.2. Experimental results

In order to adequately validate the performance of Auto-CASH, statistical tests and experimental results were conducted to compare with the two categories of methods in baselines respectively.

5.2.1. Comparison with solutions of Combined Algorithm Selection and Hyperparameter Selection problem

The experiment is conducted on 120 datasets, 15 of which serve as validation sets and the rest are training sets. After training the DQN, Auto-CASH can obtain the optimal meta-feature vector. Then we extract meta-data and train the meta-model for evaluating performance of Auto-CASH on validation datasets. Primary results of the experiment are shown in Table 5 and Table 6. From the results in Table 5, it is clear that our approach has the highest average RMV on both accuracy and AUC of all of the CASH approaches evaluated. And it demonstrates that the new metric criterion plays an important role in optimizing both accuracy and AUC with few computation cost. The fine-tuned algorithms recommended by Auto-CASH achieve top performance at a ratio of 73.3% on *accuracy*, which is a significant improvement compared with the baselines. For *AUC*, 40% datasets reach the optimal value, which is equal to Auto-Model and slightly higher than Auto-WEKA. This conclusion is also supported by Table 6, which provides AUC and accuracy value given by evaluated approaches on each validation dataset.

Analysis. The definition of the search space has a greater impact on the results of the automatic selection of the algorithm than the design of an efficient search method. The results of statistical tests and significance analysis show that Auto-CASH achieves SOTA results in the current methods. This is achieved thanks to a sequential process of algorithm selection and HPO for the selected algorithm. In CASH problems, the verification process of HPO is extremely costly, so the number of HPOs in one algorithm selection process should be minimized, which will significantly improve the efficiency and probability of finding the optimal result. With the sequential search strategy, the search space is divided into two categories: algorithm-only and hyperparameter-only. Auto-CASH can efficiently select a quite suitable classification algorithm with the help of well-trained meta-learner, and utilize the left time to find the optimal hyperparameter setting for optimizing the performance of selected algorithm, whereas, Auto-WEKA considers a huge search space which contains the algorithms and their hyperparameters, and unable to find out suitable algorithms in a short time. As the comparison, Auto-WEKA needs to waste much time on evaluating inappropriate classification algorithms with various hyperparameter settings. Therefore, its performance is lower than that of Auto-CASH.

The knowledge developed from historical experience can guide and help in the selection of algorithms for new tasks, but the format in which the knowledge is employed plays an important role. Auto-model, also known as a CASH solution, performs better than Auto-WEKA, which we believe is with the help of historical experiences, and clearly facilitates algorithm selection for current tasks. However, I think that our approach is more comprehensive in extracting and processing historical experience, because Auto-Model extracts knowledge from the data in papers and builds a knowledge base, but the experimental hardware conditions are not the same between papers, so the validity of many measures is doubtful. In addition, the experience in the knowledge base needs to be searched to match the knowledge needed of the current task when guiding a new task, which reduces the efficiency when recommending algorithms for new tasks. Therefore, in Auto-CASH, after in-depth research and comparison, it is more efficient to train a machine learning agent model using historical experience than just building a knowledge base, and it also has more theoretical performance guarantee. As a comparison, our method is more consistent in terms of the experimental hardware conditions among different papers. We adopted model-free reinforcement learning strategy to select features according to specific datasets, and the effect of feature selection from the data-driven perspective is better than the traditional feature selection method. We also designed experiments in Section 5.3 to prove our idea.

Table 5
The average RMV of accuracy, AUC, and F_{score} on validation datasets.

Measure	Auto- CASH	Auto- Model	Auto- WEKA
Average RMV(accuracy)	**0.991**	0.974	0.903
Average RMV(AUC)	**0.977**	0.973	0.941
Average RMV(Fscore)	**0.995**	0.934	0.917

Table 6
Accuracy and AUC of Auto-CASH, Auto-Model, and Auto-WEKA on validation datasets. Bold font denotes the best result.

Dataset	Records	Attributes	Classes	Accuracy			AUC		
				Auto-CASH	Auto-Model	Auto-WEKA	Auto-CASH	Auto-Model	Auto-WEKA
Avila	20867	10	12	**0.998**	0.991	0.988	**0.996**	0.994	0.881
Nursery	12960	8	3	**0.966**	0.936	0.951	0.988	**0.992**	0.967
lymph	148	18	6	**0.919**	0.855	0.878	**0.94**	0.923	0.93
Climate	540	19	2	0.887	**0.90**	0.751	0.971	**0.988**	0.788
Australian	690	14	2	**0.851**	0.806	0.804	0.986	**0.994**	0.982
Anneal	798	38	6	**0.993**	0.961	0.978	0.996	0.992	**0.997**
Squash-stored	52	24	3	**0.712**	0.648	0.60	0.603	0.628	**0.785**
Vowel	990	14	11	**0.912**	0.847	0.154	**0.995**	0.903	0.674
Zoo	101	18	7	**0.989**	0.980	0.95	0.998	**1.0**	**1.0**
Breast-W	699	9	2	0.947	**0.964**	**0.964**	**0.992**	0.991	0.987
Iris	150	4	3	0.971	**0.973**	0.968	0.996	**0998**	**0.998**
Diabetes	768	9	2	**0.788**	0.752	0.772	**0.841**	0.839	0.820
Musk1	476	166	2	**0.958**	0.941	0.951	0.958	0.944	**0.993**
Promoter	106	57	2	0.853	**0.891**	0.845	0.854	**0.892**	0.844
Blood	748	5	2	**0.716**	0.653	0.693	**0.712**	0.69	0.609

Performance validation of Auto-CASH. In order to show the significance of the Auto-CASH results, a statistical validation test is used to prove our conclusion. The *Wilcoxon test* is a famous non-parametric statistical test method outperforms t-test [44]. Wilcoxon test ranks different performances of two algorithms for each sampled dataset and compares ranks for positive and negative differences by ignoring the signs. We check the results obtained from the different CASH solutions against our approach and use them to obtain the performance distribution of different methods. The true performance of each approach is demonstrated by calculating the variability between the performance distributions.

As an example, we will show the steps of Wilcoxon test with a single sample of all datasets in Table 6. Comparison of accuracy for Auto-CASH and Auto-Model is shown in Table 7. Let d_i be the difference between accuracies of the two approaches on the i-th sampled dataset. The rank are obtained according to the absolute values of differences. Let R^+ and R^- be the sum of positive differences rank and negative differences rank, respectively. Their computing formulas are shown as Eq. 10 and 11, respectively. Let T denote the smaller of sums, which is calculated by $T = min(R^+, R^-)$. The hypothesis of the Wilcoxon test is H_0:The performance scores generated by the two approaches are from the same distribution and do not differ significantly. H_1:The performance scores generated by the two methods come from different distributions.

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{10}$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{11}$$

Results. By calculation, for this sampling, Wilcoxon test of Auto-CASH against Auto-WEKA $p - value = 0.0015$. This result implies that we have a 0.997 probability (confidence level of $\alpha = 0.01, N = 15$) of rejecting H_0 and accepting H_1. We uses the same method to calculate Auto-CASH against Auto-Model, and the $p - value = 0.025$. The result is also obvious that there is a 0.95 probability (confidence level $\alpha = 0.05, N = 15$) of rejecting H_0 and receiving H_1. To avoid the specificity of single sampling, we repeated the average sampling ten times, and the results of $p - value$ are shown in Fig. 5. Finally, the average $p - value$ of accepting H_1 for the Auto-CASH against Auto-WEKA and Auto-Model is 0.002 and 0.019, respectively. The multi-sampling results show that the Auto-CASH method performs better than two baselines, and there are significant advantages.

5.2.2. Comparison with automated machine learning systems

To illustrate the practicality of Auto-CASH, two popular AutoML systems are chosen as baselines. Auto-sklearn employs a search strategy based on Bayesian optimization, and TPOT adopts evolutionary algorithms to find the optimal solution. Unlike our approach, these AutoML systems pay more attention to the final performance, ignoring the optimization of effi-

Table 7
Comparison of accuracy for Auto-CASH and Auto-WEKA for each dataset.

Data	Auto-WEKA	Auto-CASH	difference	Rank
Avila	0.988	0.998	+ 0.01	4
Nursery	0.951	0.966	+ 0.015	5.5
lymph	0.878	0.919	+ 0.041	11
Climate	0.751	0.887	+ 0.136	14
Australian	0.804	0.851	+ 0.047	12
Anneal	0.978	0.993	+ 0.015	5.5
Squash-stored	0.60	0.712	+ 0.112	13
Vowel	0.154	0.912	+ 0.758	15
Zoo	0.95	0.989	+ 0.039	10
Breast-W	0.964	0.947	- 0.017	8
Iris	0.968	0.971	+ 0.003	1
Diabetes	0.772	0.788	+ 0.016	7
Musk1	0.951	0.958	+ 0.007	2
Promoter	0.845	0.853	+ 0.008	3
Blood	0.693	0.716	+ 0.023	9

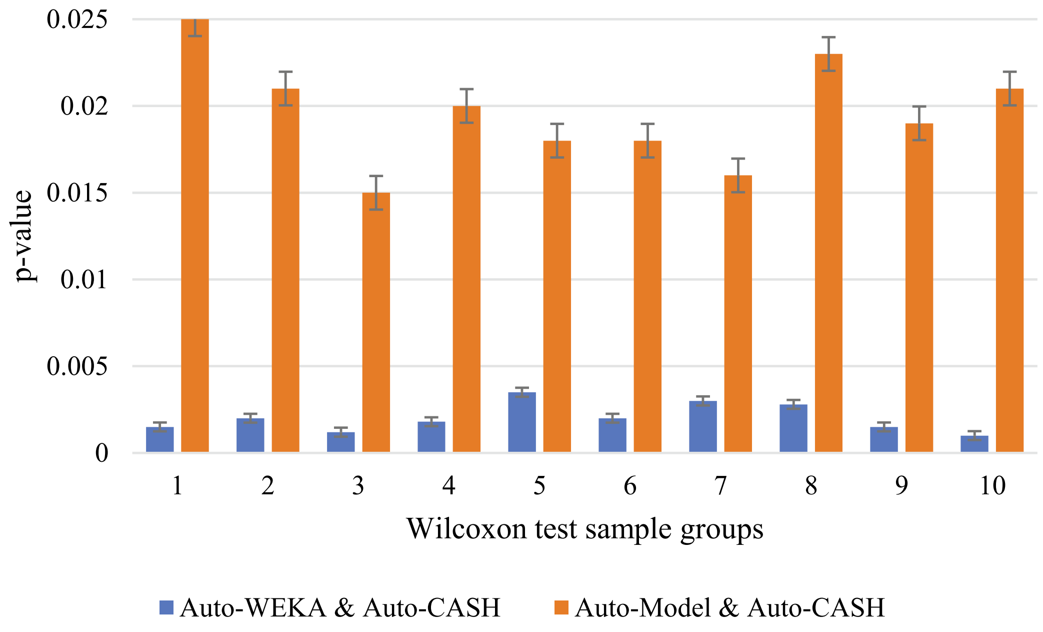


Fig. 5. Results of ten times random sample for Wilcoxon test among three CASH solutions.

Table 8
Test accuracy of compared AutoML systems on 12 validation datasets. Auto-CASH uses the same training datasets as above experiment. Because Auto-sklearn gives result of mixed multiple algorithms (A weighted classifier combination, e.g. $0.4 \cdot adaboost + 0.32 \cdot libsvm\_svc + 0.22 \cdot passive\_aggressive + 0.06 \cdot gaussian\_nb$), the AUC value is meaningless.

Dataset	Records	Attributes	Classes	TPOT	Auto-sklearn	Auto-CASH
Cmc	1473	10	3	0.538	0.547	**0.554**
Musk1	476	166	2	0.95	0.909	**0.958**
Vertebral	310	6	2	0.795	0.892	**0.913**
Mammo(graphic)	961	5	2	0.843	0.815	**0.855**
Dermatology	358	34	6	0.97	0.963	**0.986**
Diabetes	768	8	2	0.77	0.753	**0.788**
Live-disorder	345	6	2	0.721	0.75	**0.756**
Opdigits	5620	64	10	0.986	**0.989**	0.983
Sonar	208	60	2	0.73	0.841	**0.865**
Teaching assistant	151	5	3	0.56	0.652	**0.657**
Vehicle	946	18	4	0.763	0.744	**0.863**
Wine	178	13	3	0.962	0.962	**0.977**

ciency. As a result, they suffer from low efficiency with time constraints. Detailed results shown in Table 8 also proves our approach is more efficient on the 12 validation datasets.

Analysis. There are two main reasons why our work achieves substantially performance improvement: (a) The meta-model chosen by our approach has an excellent predictive performance for the optimal algorithm. It is more efficient than Auto-WEKA's continuous iteration to predict the performance distribution. Meta-learning theory maximizes the use of knowledge contained in experience; (b) The meta-feature selected by DQN can represent the characteristics of the dataset more comprehensively. DQN deeply excavates the relationship among the meta-features and avoids the overlapping of feature information representation to the greatest extent. We reduce the number of meta-features to 7, and it is 11 in Auto-Model. All in nuts, the design of Auto-CASH that utilizes the experience learned before to give better results for new tasks is reasonable and meaningful, and the results compared to the state-of-the-art CASH solutions, and AutoML systems reveal the merits of our approach.

A combination of empirical learning (e.g., meta-learning) is more effective in the field of automatic algorithm selection than iterative search-only. Although iterative optimization search round after round may find the optimal algorithm, it is easy to fall into time constraints. For the AutoML method, it is common that the runtime requirement is long because in order to be "Auto" enough, many search techniques are used to enhance the automation, which is a time for performance approach. In the design and optimization of Auto-CASH, we chose to balance the execution time with the performance of the algorithm. Therefore, the most prominent feature of Auto-CASH compared to these methods is that the algorithm selection step is accelerated by meta-learning instead of iteratively trying to select from many algorithms. Although it may not be possible to select the optimal algorithm, the meta-learner trained with sufficient experience has good prediction performance and combined with our proposed two pruning strategies genetic search, so that even if the best algorithm is not selected, it will still approach the optimal performance. This is why our method has the highest RMV value (Table 5), which means that the algorithm and hyperparameters chosen by Auto-CASH for the task has the highest probability of being close to the actual optimal algorithm.

Performance validation of Auto-CASH. In order to show the significance of the Auto-CASH results, a statistical validation test is used to prove our conclusion. *Wilcoxon test* is stilled used here the same as above section to verify the performance of our approach. The calculation of statistic value and hypothesis readers can refer to Section 5.2.1. In addition, in Section 5.2.1 we have provided an example to demonstrate the specific calculation process, so we will not repeat it here. Wilcoxon test is applied for Auto-CASH against TPOT and Auto-sklearn, respectively. To avoid the specificity of single sampling, we repeat the average sampling ten times, and results of $p - value$ are shown in Fig. 6. According to the calculation of sampled data, the average $p - value$ of Auto-CASH against TPOT and Auto-sklearn is 0.001 and 0.002 respectively (confidence level $\{\alpha = 0.002, N = 15\}$ and $\{\alpha = 0.004, N = 15\}$, respectively), which means the probability of accepting H_1 is 0.998 and 0.996. Statistical data and experimental results demonstrate the significant superiority of our approach compared to baselines.

5.3. Discussions

We show the major results and analyze them in above section, and additional observations and analysis during the experiments are discussed in this section.

The impact of meta-features is in a large range. For each meta-feature, we randomly select some batches containing it. Based on each batch of meta-features, we construct and train a RF model. We repeat above steps multiple times for each batch size, and the average accuracy of RF is the reward. Fig. 7 shows the estimated reward for meta-features in Auto-CASH. From the results, these meta-features increase the RF predictive accuracy by up to 24 percent, which shows the significant impact on the RF training effect and conveys the importance of the automatic selection of meta-feature methods.

Too many meta-features do not mean enough information gain. The relationship between the number of meta-feature for representing a dataset and the accuracy RMV are presented in Fig. 8. It is obvious that when the number is relatively small

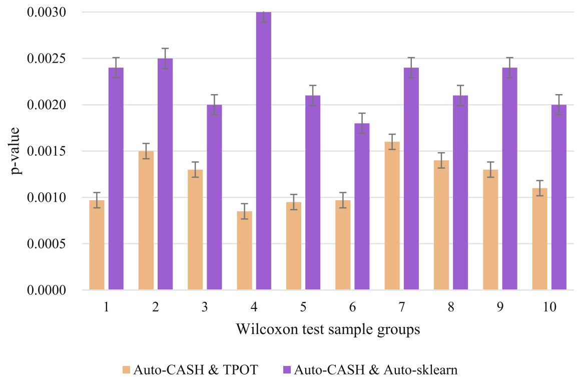


Fig. 6. Results of ten times random sample for Wilcoxon test between two AutoML solutions.

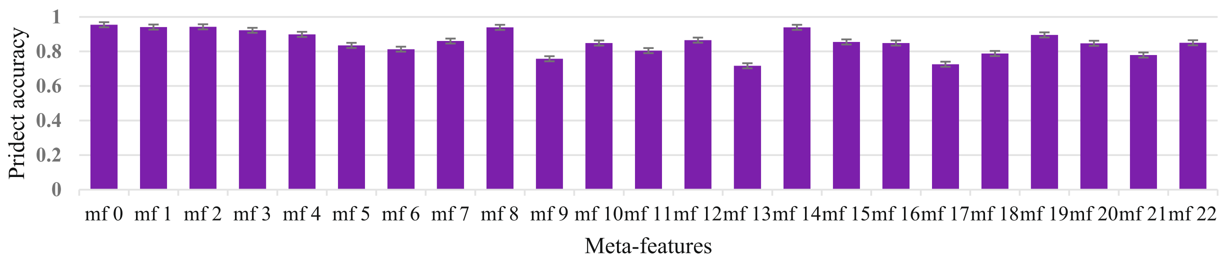


Fig. 7. Performance for each meta-feature.

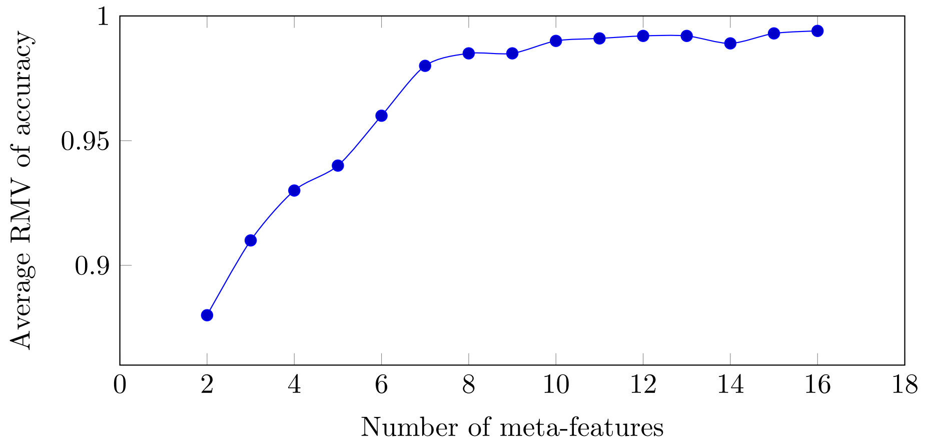


Fig. 8. The number of meta-feature has a great influence on the average accuracy RMV.

Table 9
Categories of 23 candidate statistical meta-features. For category attributes, we concentrate on the inter-class dispersion degree and the maximum range of class proportion. For numeric attributes, we are more concerned about the center and extent of fluctuation. Besides, we also take the global numeric information of records and attributes into consider.

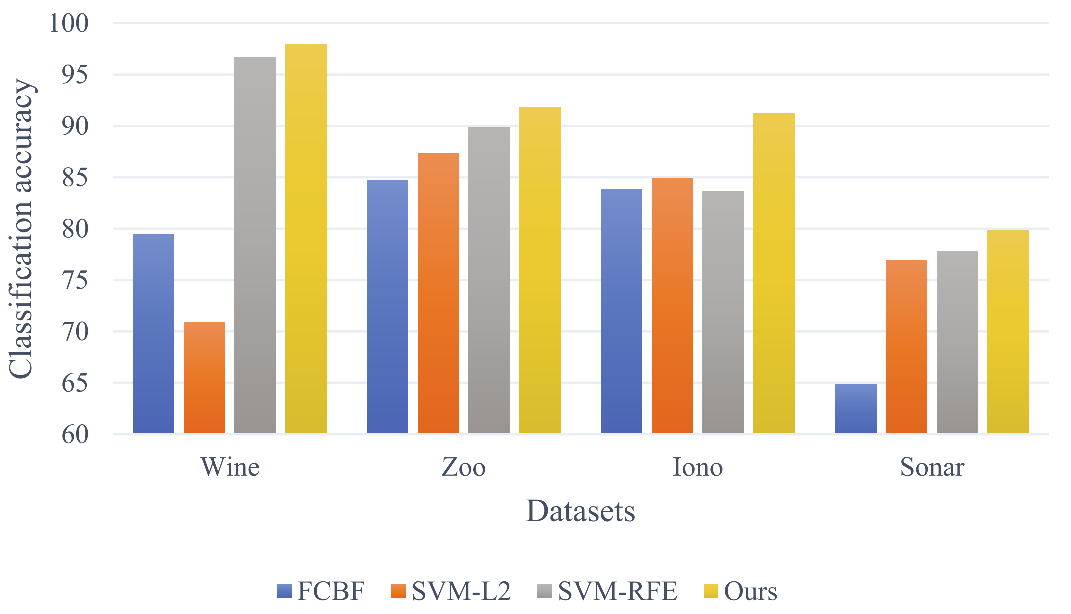
Category	mf index
Category information entropy	1,10,14
Proportion of classes in different type of attributes	2,3,6,11,12,15,16
Average value	17,18
Variance	19,20,21,22
Number of instances	0,4,5,7,8,9,13

Table 10
After fine-tuning, the hyperparameter values setting of DQN.

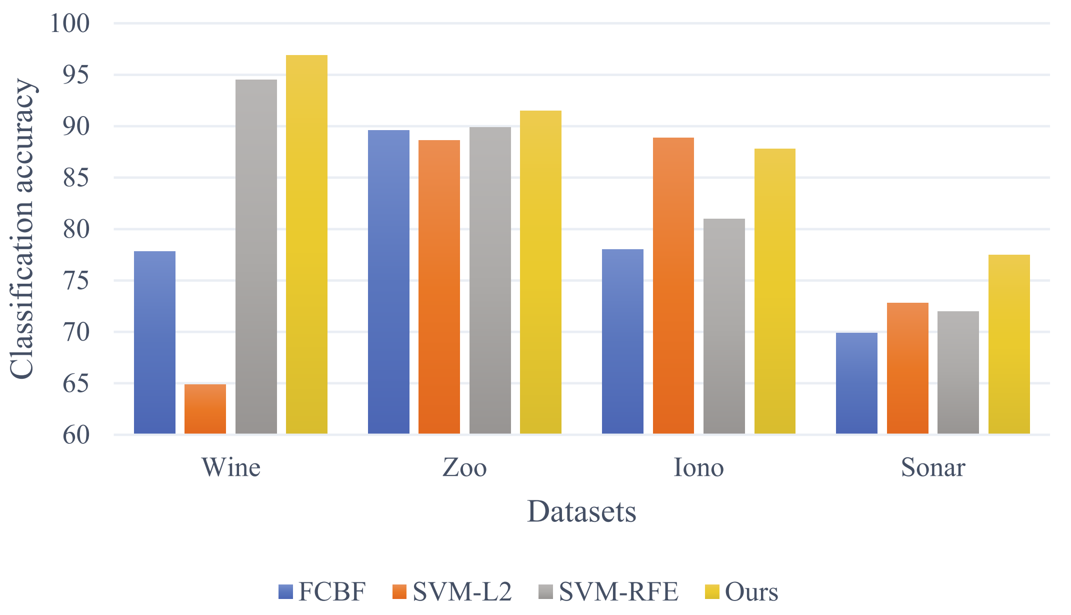
HYP	State	Action	Memory	γ	α
Value	2^{23}	23	800	0.95	0.0001

(less than 7), each additional meta-feature could achieve a large improvement on average RMV. It indicates that the meta-features adopted are not enough to represent all the essence of the training datasets. However, the information gain brought by more meta-features is gradually decreasing with the increase on number. DQN is introduced to find a group of minimum meta-features that can represent the feature of the datasets to the maximum extent. In this way, Auto-CASH optimizes the training effect of meta-model and achieves better algorithm selection results than baselines.

The meta-feature selected by DQN can comprehensively represent the datasets. Compared with Auto-Model, we use fewer meta-features, and Auto-CASH achieves a better performance in most cases as shown in Table 6. It proves that DQN does improve the efficiency of CASH through meta-feature selection. However, our approach suffers from stability because that of DQN depends on the setting of hyperparameters slightly. That is one possible shortcoming of our approach. After tuning various hyperparameter combinations, we finally controlled DQN to converge around 1,500 epochs. The hyperparameter of our DQN model are listed in Table 10. According to [45], we adapt two mechanisms to make DQN acts more like human being: *Experience Replay* and *Fixed Q-target*. Finally we get the $M_{list} = \{mf_0, mf_2, mf_4, mf_6, mf_7, mf_9, mf_{13}\}$ among the outputs of fine-tuned DQN. Let us analyze the result given by DQN and our previous manual selected meta-feature list. In Auto-

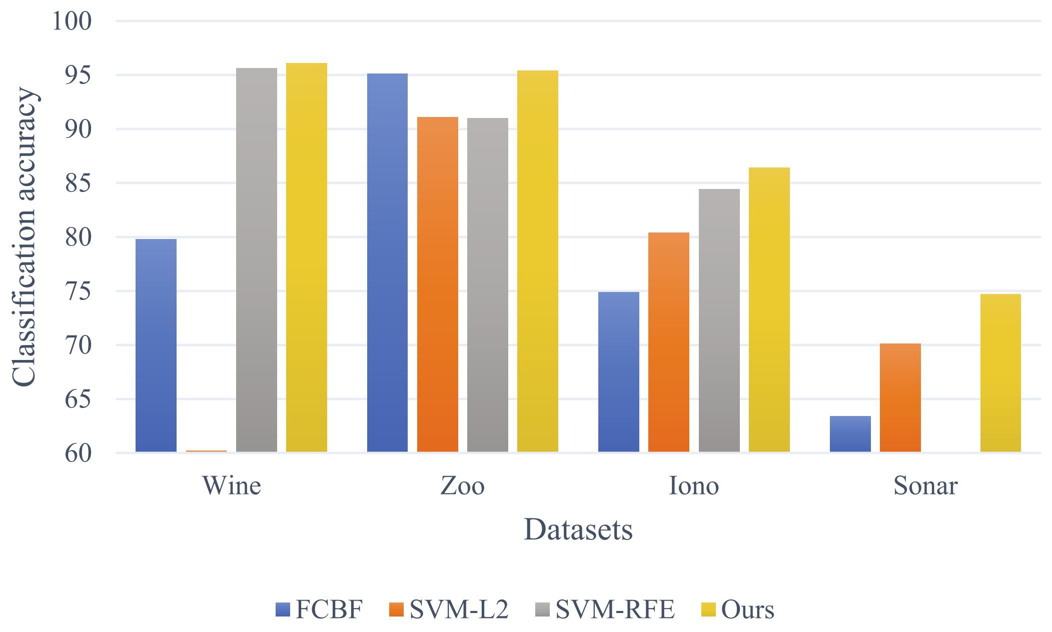


(a) Comparison of classification accuracy of different feature selection algorithms under KNN classifier.

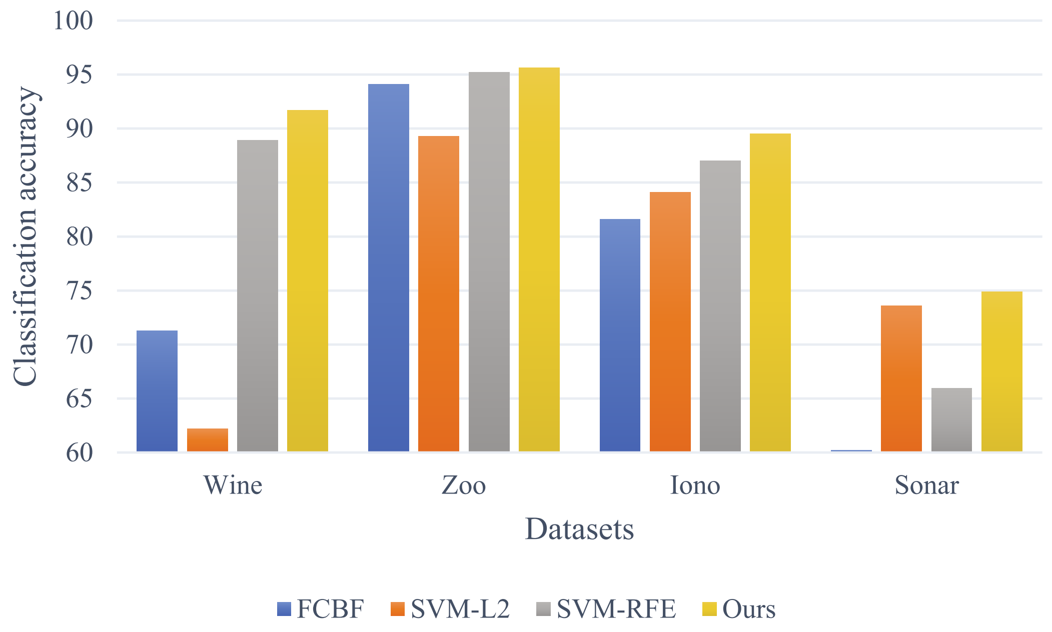


(b) Comparison of classification accuracy of different feature selection algorithms under LSVM classifier.

Fig. 9. Experimental results of different feature selection approaches.



(c) Comparison of classification accuracy of different feature selection algorithms under Naive Bayes classifier.



(d) Comparison of classification accuracy of different feature selection algorithms under Decesion Tree classifier.

Model, we finally define the $M_{list}^{AM} = \{mf_1, mf_3, mf_4, mf_5, mf_8, mf_9, mf_{10}, mf_{11}, mf_{17}, mf_{18}, mf_{19}\}$ with series of experiments. To facilitate the analysis of different points, the 23 meta-features are classified here into the following five categories and are shown in Table 9. According to the classification, the meta-features used in Auto-Model are spread over these five categories, while the results given by DQN are more concentrated in the second and fifth categories. While this is an interesting phenomenon, it does make sense. Once we know the number of instances and the proportion of different categories among them, we can calculate the values of mean, variance, and information entropy. In other words, the first, third and fourth categories of meta-features can be derived from the second and fifth meta-features as a basis. DQN's selection prefers these foundations, while being richer in informative meta-features.

To demonstrate our automatic meta-feature selection approach is effective, we also conduct comparative experiments with traditional feature selection approaches. We used different feature extraction algorithms for the classification task dataset to extract features and then ran the classification algorithm on the new dataset with the final accuracy as a measure. The baselines are as follows. a) FCBF [46], which is a fast filtering feature selection algorithm, a symmetrical uncertainty (SU) based method. b) SVM-RFE [47], which is a sequential backward selection algorithm based on the maximum interval principle of SVM and is an embedding approach. It trains the sample by the model, then sorts each feature with a score, removes the feature with the smallest score, then trains the model again with the remaining features for the next iteration, and finally selects the desired number of features. c) SVM with L2 regularization feature selection approach, which is a wrapper based approach. Experimental results on different classifiers and datasets are shown in Fig. 9. Experimental results demonstrate that the features extracted using the reinforcement learning strategy can better represent the dataset and achieve the highest classification accuracy.

Auto-CASH achieves better performance in shorter time. Compared to baselines, we save about a quarter of time cost while obtaining the same or better results (5 min for CASH solutions and 30 min for AutoML systems) with 230 s average. The experimental results demonstrate that our approach reduces the search space of the optimal solution of the CASH problem and makes full use of experience, thereby improving the efficiency of solving the CASH problem.

Limitations of our work. Although our approach has many innovative points, it also has some limitations. First, the list of algorithms and the set of meta-features are limited, which may not be suitable for some classification tasks, and we hope to improve the scalability of the method in future studies, for example, by supporting users to upload custom algorithms to the list of algorithms to be selected. Second, at present, Auto-CASH only supports the dataset of classification tasks in tabular format, and we would like to extend it to other tasks, such as time series classification tasks, in the future.

6. Conclusion and future work

In this paper, we propose Auto-CASH, an approach combined with meta-learning and reinforcement learning for the CASH problem. By transforming the selection of meta-feature into a continuous action-selection problem, Auto-CASH can automatically solve it utilizing Deep Q-Network. Thus it significantly reduces human labor in the training. For a particular task, Auto-CASH enhances the performance of the recommended algorithm within an acceptable time. Experimental results demonstrate that Auto-CASH outperforms state-of-the-art CASH approaches and popular AutoML systems on efficiency. While our approach still have some limitations, for example, the candidate algorithm list is fixed resulting in less flexibility.In future work, we plan to improve the scalability of Auto-CASH, for example, by customizing the list of algorithms and extending the types of tasks supported. Besides, we plan on applying neural architecture search (NAS) techniques to find a DQN structure that can improve its stability and speed up the training of the network.

Credit authorship contribution statement

Tianyu Mu: Writing – original draft, Writing – review & editing, Conceptualization, Methodology, Formal analysis, Visualization, Software, Validation, Investigation. **Hongzhi Wang:** Writing – original draft, Writing – review & editing, Conceptualization, Methodology, Supervision, Project administration, Funding acquisition. **Chunnan Wang:** Visualization, Validation, Conceptualization. **Zheng Liang:** Methodology, Visualization, Writing – original draft. **Xinyue Shao:** Writing – original draft, Resources.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This paper was supported by NSFC grant (U1866602, 71773025). The National Key Research and Development Program of China (2020YFB1006104).

References

[1] M. Lindauer, J.N. van Rijn, L. Kotthoff, The algorithm selection competitions 2015 and 2017, Artif. Intell. 272 (2019) 86–100.
[2] B. Taylor, V.S. Marco, W. Wolff, Y. Elkhatib, Z. Wang, Adaptive deep learning model selection on embedded systems, ACM SIGPLAN Notices 53 (6) (2018) 31–43.

[3] C. Schaffer, Cross-validation, stacking and bi-level stacking: Meta-methods for classification learning, in, Selecting Models from Data, Springer (1994) 51–59.
[4] C. Thornton, F. Hutter, H.H. Hoos, K. Leyton-Brown, Auto-weka Combined selection and hyperparameter optimization of classification algorithms, in, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013, pp. 847–855.
[5] A. Ławrynowicz, J. Potoniec, Pattern based feature construction in semantic data mining, International J. Semantic Web Inf. Syst. 10 (1) (2014) 27–65.
[6] C. Wang, H. Wang, T. Mu, J. Li, H. Gao, Auto-model: Utilizing research papers and hpo techniques to deal with the cash problem, in: 2020 IEEE 36th International Conference on Data Engineering (ICDE), IEEE, 2020, pp. 1906–1909.
[7] M. Feurer, F. Hutter, Hyperparameter optimization, in: Automated Machine Learning, Springer, Cham, 2019, pp. 3–33..
[8] M. Feurer, A. Klein, K. Eggensperger, J.T. Springenberg, F. Hutter, Auto-sklearn: Efficient and robust automated machine learning..
[9] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning, Springer, 2019.
[10] B.M. Lake, T.D. Ullman, J.B. Tenenbaum, S.J. Gershman, Building machines that learn and think like people, Behavioral and brain sciences 40..
[11] M. Feurer, J.T. Springenberg, F. Hutter, Initializing bayesian hyperparameter optimization via meta-learning, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015..
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602..
[13] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.
[14] F.S. Melo, Convergence of q-learning: A simple proof, Institute Of Systems and Robotics, Tech. Rep. (2001) 1–4.
[15] T. Doan, J. Kalita, Selecting machine learning algorithms using regression models, in: 2015 IEEE International Conference on Data Mining Workshop (ICDMW), IEEE, 2015, pp. 1498–1505.
[16] N. Mori, M. Takeda, K. Matsumoto, A comparison study between genetic algorithms and bayesian optimize algorithms by novel indices, in: Proceedings of the 7th annual conference on Genetic and evolutionary computation, 2005, pp. 1485–1492.
[17] N. Cohen-Shapira, L. Rokach, B. Shapira, G. Katz, R. Vainshtein, Autogrd: Model recommendation through graphical dataset representation, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 821–830.
[18] I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. One, K. Cho, C. Silva, J. Freire, Alphad3m: Machine learning pipeline synthesis, in: AutoML Workshop at ICML, 2018..
[19] R.S. Olson, J.H. Moore, Tpot A tree-based pipeline optimization tool for automating machine learning, in, Automated Machine Learning, Springer (2019) 151–160.
[20] S.N. das Dôres, L. Alves, D.D. Ruiz, R.C. Barros, A meta-learning framework for algorithm recommendation in software fault prediction, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 1486–1491.
[21] R. Vainshtein, A. Greenstein-Messica, G. Katz, B. Shapira, L. Rokach, A hybrid approach for automatic model recommendation, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 2018, pp. 1623–1626.
[22] A. Yakovlev, H.F. Moghadam, A. Moharrer, J. Cai, N. Chavoshi, V. Varadarajan, S.R. Agrawal, S. Idicula, T. Karnagel, S. Jinturkar, et al, Oracle automl: a fast and predictive automl pipeline, Proc. VLDB Endowment 13 (12) (2020) 3166–3180.
[23] Z. Luo, Z. He, J. Wang, M. Dong, J. Huang, M. Chen, B. Zheng, Autosmart: An efficient and automatic machine learning framework for temporal relational data, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 3976–3984.
[24] R. Kohavi, G.H. John, Automatic parameter selection by minimizing estimated error, Machine Learning Proceedings 1995, Elsevier (1995) 304–312.
[25] D.C. Montgomery, Design and analysis of experiments, John Wiley & Sons (2017).
[26] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305.
[27] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. 18 (1) (2017) 6765–6816.
[28] E. Brochu, V.M. Cora, N. De Freitas, A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, arXiv preprint arXiv:1012.2599..
[29] D. Whitley, A genetic algorithm tutorial, Stat. Comput. 4 (2) (1994) 65–85.
[30] K. Swersky, J. Snoek, R.P. Adams, Freeze-thaw bayesian optimization, arXiv preprint arXiv:1406.3896..
[31] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, R. Adams, Scalable bayesian optimization using deep neural networks, in: International conference on machine learning, 2015, pp. 2171–2180.
[32] K. Kandasamy, K.R. Vysyaraju, W. Neiswanger, B. Paria, C.R. Collins, J. Schneider, B. Poczos, E.P. Xing, Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly, J. Mach. Learn. Res. 21 (81) (2020) 1–27, http://jmlr.org/papers/v21/18-223.html.
[33] L. Hertel, J. Collado, P. Sadowski, J. Ott, P. Baldi, Sherpa: Robust hyperparameter optimization for machine learning, SoftwareX 12 (2020) 100591.
[34] Y. Li, J. Jiang, J. Gao, Y. Shao, C. Zhang, B. Cui, Efficient automatic cash via rising bandits, AAAI (2020) 4763–4771.
[35] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown, Towards an empirical foundation for assessing bayesian optimization of hyperparameters, in: NIPS workshop on Bayesian Optimization in Theory and Practice, vol. 10, 2013, p. 3..
[36] D.M. Powers, Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation..
[37] T. Fawcett, An introduction to roc analysis, Pattern Recogn. Lett. 27 (8) (2006) 861–874.
[38] L. Xiujuan, S. Zhongke, Overview of multi-objective optimization methods, J. Syst. Eng. Electron. 15 (2) (2004) 142–146.
[39] F. Pinto, C. Soares, J. Mendes-Moreira, Towards automatic generation of metafeatures, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2016, pp. 215–226.
[40] T.A. Gomes, R.B. Prudêncio, C. Soares, A.L. Rossi, A. Carvalho, Combining meta-learning and search techniques to select parameters for support vector machines, Neurocomputing..
[41] P. Brazdil, C.G.G. Carrier, C. Soares, R. Vilalta, Metalearning: Applications to data mining, Cognitive Technologies..
[42] C. Lazar, J. Taminau, S. Meganck, D. Steenhoff, A. Coletta, C. Molter, V. de Schaetzen, R. Duque, H. Bersini, A. Nowe, A survey on filter techniques for feature selection in gene expression microarray analysis, IEEE/ACM Trans. Comput. Biol. Bioinf. 9 (4) (2012) 1106–1119.
[43] A.G. Karegowda, M. Jayaram, A. Manjunath, Feature subset selection problem using wrapper approach in supervised learning, Int. J. Comput. Appl. 1 (7) (2010) 13–17.
[44] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[45] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[46] L. Yu, H. Liu, Feature selection for high-dimensional data: A fast correlation-based filter solution, in: Proceedings of the 20th international conference on machine learning (ICML-03), 2003, pp. 856–863.
[47] P.A. Mundra, J.C. Rajapakse, Svm-rfe with mrmr filter for gene selection, IEEE Trans. Nanobiosci. 9 (1) (2009) 31–37.