Assassin: an Automatic claSSificAtion system baSed on algorithm SelectIoN

Tianyu Mu
Harbin Institute of Technology
mutianyu@hit.edu.cn

Hongzhi Wang
Harbin Institute of Technology
Peng Cheng Laboratory(PCL)
wangzh@hit.edu.cn

Shenghe Zheng
Harbin Institute of Technology
1190300321@stu.hit.edu.cn

Shaoqing Zhang
Harbin Institute of Technology
1190200721@stu.hit.edu.cn

Cheng Liang
Harbin Institute of Technology
1190301804@stu.hit.edu.cn

Haoyun Tang
Harbin Institute of Technology
1190201526@stu.hit.edu.cn

ABSTRACT

The increasing complexity of data analysis tasks makes it dependent on human expertise and challenging for non-experts. One of the major challenges faced in data analysis is the selection of the proper algorithm for given tasks and data sets. Motivated by this, we develop Assassin, aiming at helping users without enough expertise to *automatically select optimal algorithms* for classification tasks. By embedding meta-learning techniques and reinforced policy, our system can automatically extract experiences from previous tasks and train a meta-classifier to implement algorithm recommendations. Then we apply genetic search to explore hyperparameter configuration for the selected algorithm. We demonstrate Assassin with classification tasks from OpenML. The system chooses an appropriate algorithm and optimal hyperparameter configuration for them to achieve a high-level performance target. The Assassin has a user-friendly interface that allows users to customize the parameters during the search process.

PVLDB Reference Format:
Tianyu Mu, Hongzhi Wang, Shenghe Zheng, Shaoqing Zhang, Cheng Liang, and Haoyun Tang. Assassin: an Automatic claSSificAtion system baSed on algorithm SelectIoN . PVLDB, 14(12): 2751 - 2754, 2021.
doi:10.14778/3476311.3476336

1 INTRODUCTION

With explosive growth of digital information, the data captured in real life becomes more complicated. As a result, the complexity of data analysis tasks gradually increases according to application scenarios. The term Automatic Statistician (also known as Automatic Data Analysis) [9] is often used to describe systems with the ability to automate the process of data analysis such as model selection, data cleansing (or restoration) and producing predictions with minimal human intervention. The algorithm(or model) employed to process the data is the heart of overall data analysis task, either in the data pre-processing phase or in the data analysis phase. Although many algorithms have been developed for the same type of tasks (e.g., classification), one algorithm can hardly outperform others in all aspects and scenarios [8]. Currently, most of such work are accomplished by domain experts based on their experiences and a series of experiments. Recently, data become too complex for experts to design suitable analysis model. Thus, a system that can leverage observed experiences to automatically select an appropriate algorithm for the task is in demand.

While existing approaches [2, 4, 10] attempt to address the algorithm selection problem, two major problems remain unresolved. On the one hand, existing approaches fails to make sufficient use of experiences, which are useful for new algorithm selection jobs. On the other hand, hyperparameter configuration of the selected algorithm is still inefficient due to the large search space.

To address above issues, we developed Assassin, an automatic classification algorithm selection system. Our system focuses on the following objectives. **1) Utilization of experiences.** To address the first problem, we represent 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. **2) Efficiency of hyperparameter optimization(HPO).** To address the second problem, two mechanisms are designed to narrow the search space, aiming at ensuring fast convergence of the genetic search. 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*. **3) Automation of the entire system.** Assassin 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.

To summarize, Assassin has following features: **1) Full automation.** Our system automates the whole process of algorithm selection and HPO. Users only need to upload the data set of classification task, and the optimal algorithm and its hyperparameters will be provided without human efforts. **2) High precision.** Experimental results show that Assassin can select a higher performance algorithm for a user task, which demonstrates that our system is effective. **3) Friendly to users.** Whatever experts or non-expert

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476336

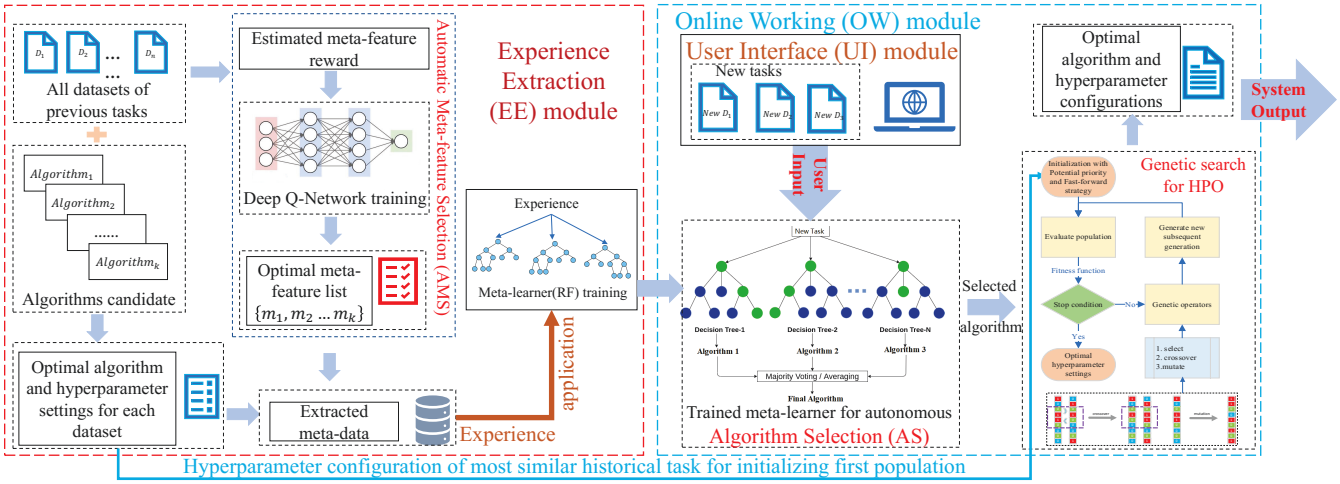


Figure 1: The Assassin workflow. With the training based on the experiences extracted from previous tasks, our system can select the algorithm online. Optionally, if users has enough time, the model can be re-trained and some parameters such as algorithm lists can be specified.

users can get started quickly through the graphical interface. Specially, we set up the customization mode to facilitate users to modify the parameter settings of system.

2 SYSTEM OVERVIEW

The goal of our system is to select an algorithm with a tuned hyperparameter configuration to gain maximal performance for a given data analysis task. To achieve this goal, we develop automatic algorithm selection strategy based on historical experiences and search space pruning algorithm for HPO.

System Architecture As shown in Figure 1, Assassin consists of following 3 modules, in which EE module accomplishes the experience extraction, and OW is in charge of algorithm selection and HPO. Besides, we develop user interface module for interaction. Then we will introduce these modules and the workflow of our system.

Experience Extraction (EE) module. Since the prior experiences are crucial for guiding algorithm selection, we develop EE module to capture the experiences. In this module, we represent a data analysis task as a meta-feature[1] vector and model the algorithm selection experiences as the mapping from analysis task to its optimal algorithm. Thus, the selected meta-features for representing the tasks are crucial to the effectiveness of experiences to the future algorithm selection tasks. We design AMS algorithm employing reinforced policy to select proper features. Such policy can quantify the importance of each meta-feature and the benefits of different combinations of meta-features through continuous exploration to maximize the representation ability of the selected features. Then the meta-learner is trained on the extracted experiences. The details of the feature selection algorithm AMS will be introduced in Section 3.1.

Online Working (OW) module. To select proper algorithm and determine its effective hyperparameter configuration, we develop this module. Algorithm selection and HPO are executed in order, which has gradually become a rising trend [10, 11]. Firstly, this module calls meta-learning-based AS algorithm, which uses the trained meta-learner in EE module to select a high-performance algorithm for the task given by users. Secondly, we perform genetic search for the algorithm selected in last step. Specifically, we design two HPO search space pruning strategies to increase the convergence speed. The AS and HPO algorithms adopted in our system will be introduced in Section 3.2 and Section 3.3, respectively.

User Interface (UI) module In this module, we provide graphical interface for users, as shown in Section 4. The GUI permits user to upload local data sets and set parameters by themselves. Our system provides two modes for users: 1) *Customization mode* allows users modify some settings, such as candidate algorithm list, running time limit, the number of tasks as experience and meta-feature list generated by DQN. This mode is suitable for experts. 2) *Automation mode* only requires users to upload the data set and the analysis task, and our system analyzes the data set based on the default setting and automatic algorithms. This mode is suitable for non-expert users. In this module, we also provide the interface for data analysis performance evaluation and statistical information of data.

The workflow is summarized as follows. First of all, Assassin employs EE module for extracting experience from previous tasks and training the meta-learner for next step. Then OW module invokes AS algorithm to select an optimal algorithm for tasks uploaded through UI module by users. At last, HPO is executed to search the optimal hyperparameter configuration for the algorithm selected by AS.

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

3 TECHNIQUES

In this section, we give the details of the core algorithms in our system.

3.1 Automatic Meta-feature Selection (AMS)

AMS algorithm aims to automatically choose proper meta-features to represent tasks. Unfortunately, the connection between meta-features is complex, and it is hard to quantify the effect of different combination of meta-features. Consider that reinforcement learning achieves high performance on multiple candidate selection problem [6]. Such approach is suitable for our automatic meta-feature selection demand.

Based on above discussions, we develop a reinforcement learning algorithm for the meta-feature selection. 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 an action a is the probability of selecting the optimal algorithm by performing a. The Q table update function is as follows.

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

The Deep Q-Network updates the correlations among meta-features according to Equation 1. Under the state s, the agent selects action a with a maximum cumulative reward r according to Q-value calculated by a neural network, and then enters state s'. The Q-value should be updated right now. γ and α denote the discount factor and learning rate, respectively. The difference between the true and the estimated Q-value is $\alpha[R(s,a) + \gamma max_{a'} Q(s',a') - Q(s,a)]$. The value of α determines the speed of learning, and the value of γ means the importance of the future rewards. The Q-value reflects correlation and influence between all meta-features, and the final list consists of those with higher Q-values.

After selecting k meta-features representing the tasks, each task $t_j \in \mathcal{T}_{old}$ is described with a meta-feature vector $\mathcal{M}_{t_j} = (m_j^1, ...m_j^k)$. Then for each t_j, the meta-feature vector and its optimal algorithm are collected as meta-data $\mathbb{E}$. Assuming that we have 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 experiences and the basis of meta-learner training. The trained meta-learner $\mathcal{L}^{\mathbb{E}_{n\times(k+1)}}$ is trained based on the meta-data for AS algorithm in OW module.

We select Random Forest (RF) model as the meta-learner, which is blessed with following advantages. 1) We use some complex meta-features to represent the tasks. RF is sensitive to the internal influences among these meta-features when training. 2) Experimental results in [3] show that RF is more adept at selecting an optimal algorithm based on task features.

3.2 Algorithm Selection (AS)

AS aims to select an optimal algorithm for the data analysis tasks. It is based on the meta-learner from AMS algorithm.

AS is divided into following 2 steps. Firstly, AS calculates the meta-feature vector $\mathcal{M}_t$ for user-uploaded task t. Then, the trained meta-learner $\mathcal{L}^{\mathbb{E}_{n\times(k+1)}}$ selects a high-performance algorithm $\mathcal{A}^* = \mathcal{L}^{\mathbb{E}}(\mathcal{M}_t)$ according to the meta-vector.

3.3 Hyperparameter Optimization (HPO)

HPO algorithm aims to search optimal hyperparameter configurations that can maximize the performance of the algorithm selected by AS. 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 [7], we design the GA-based HPO algorithm. In the beginning, we encode hyperparameter configurations to binary sequences and initializes the original population. Then after crossover and mutation, we select the batch of individuals high performance for next generation. For each subsequent generation, the hyperparameter configuration is returned as the result if the termination condition has been reached. Otherwise, the above steps will be executed iteratively.

However, GA still has some drawbacks that lead to slow convergence in the case of large scale hyperparameter space [10]. To tackle this problem, we propose two search space pruning strategies before GA. Experimental results in [5] also illustrate that such pruning strategies significantly speed up convergence when tuning for SVM and the optimal hyperparameter combination was found within almost 10 generations. We then discuss these two pruning strategies.

Potential Priority. Since not all hyperparameters are useful for the final tuning results [1], we first reduce the dimension of hyperparameters for acceleration. "Potential" evaluation are based on the performance improvement after tuning. According to the Occam's razor principle, we only select the hyperparameters that brings a relatively large effect improvement for tuning. Let $\mathcal{A}$ and $\mathcal{D}_{val}$ denote an algorithm with n hyperparameters $\lambda_1, \lambda_2, ...\lambda_n$ and validation data set, respectively. Given a performance improvement evaluation function $I(\mathcal{A}_{\lambda_i}, \mathcal{D}_{val})$, the pruning can be formulated as Equation (2) .

$$\begin{aligned} \lambda_{list} &= \{\lambda_i | i \in [1,n], I(\mathcal{A}_{\lambda_i}, \mathcal{D}_{val}) > \theta\} \\ \theta &= 0.3 \cdot \max_{i\in[1,n]} I(\mathcal{A}_{\lambda_i}, \mathcal{D}_{val}) \end{aligned} \quad (2)$$

Fast-forward Initialization. We use L1 distance of the meta-feature vectors to measure the distance between tasks, and choose the hyperparameter configuration for the historical task that is closest to user task as the initialization of GA. For the optimization of target task $\mathcal{T}_{new}$ using the genetic algorithm(GA), we give the following definition describing the change in performance during the iterative process.

Definition 1 *k-neighborhood*: For a GA algorithm with hyperparameter space Λ and a given original hyperparameter λ_0, $GA^k(\lambda_0)$ represents the hyperparameter $\lambda \in \Lambda$ after k generations of GA. Thus the k-neighborhood of λ_0 is as follows.

$$N(\lambda_0, k) = \{\lambda | P\{GA^k(\lambda_0) = \lambda\} > 0\} \quad (3)$$

For a target task $\mathcal{T}_{new}$ and a task $\mathcal{T}_1$ in observed experiences, the number of generation k and a given bound μ, we assume that:

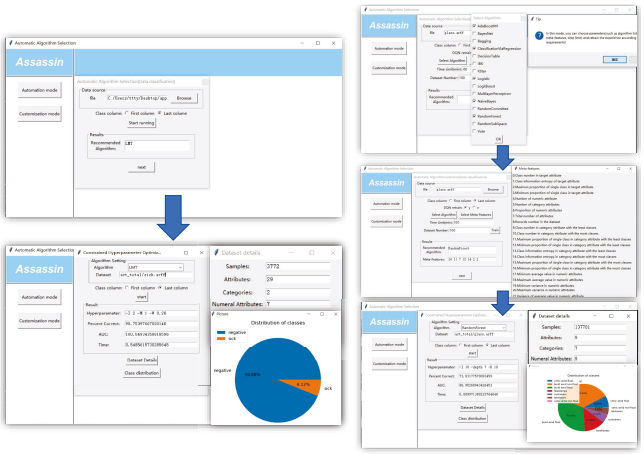
$$P\{Jaccard(N(\lambda_{\mathcal{T}_1}, k), N(\lambda_{\mathcal{T}_{new}}, k)) \geq \mu\} \propto sim(\mathcal{T}_1, \mathcal{T}_{new}) \quad (4)$$

where $sim(\mathcal{T}_1, \mathcal{T}_{new})$ denotes the similarity between tasks. $\lambda_{\mathcal{T}_{new}}$ represents the optimal hyperparameter configuration for $\mathcal{T}_{new}$, so $N(\lambda_{\mathcal{T}_{new}}, k)$ is a small neighborhood filled with values that can reach the optimal solution after k generations. To accelerate the convergence of GA, our goal is to find an original hyperparameter λ_0 where:

$$\begin{aligned} \lambda_0 &= \arg\max P\{Jaccard(N(\lambda_0, k), N(\lambda_{\mathcal{T}_{new}}, k)) \geq \mu\} \\ &= \arg\max sim(\mathcal{T}_1, \mathcal{T}_{new}) = \arg\min dis(\mathcal{T}_1, \mathcal{T}_{new}) \end{aligned} \tag{5}$$

We demonstrate these two strategies on algorithm Random Forest with all training data sets. By performing Potential Priority strategy, the hyperparameters to be tuned are '-K', '-I', '-depth'[2]. Experimental results show that after tuning the three hyperparameters separately, the performance improvement is 3.8%, 1.5% and 4.9% respectively and the '-depth' has the greatest "potential". With Fast-forward Initialization strategy, GA reaches convergence in less than 15 generations. The experimental results show that our pruning strategy effectively improve the speed of convergence.

4 DEMONSTRATIONS



(a) Automation Mode (b) Customization Mode

Figure 2: System Interfaces

We plan to demonstrate both two algorithm selection modes with the visualization of each step, and show how Assassin works with various data sets.

Data sources. We demonstrate the whole algorithm selection and hyperparameter optimization process with two open classification data sets from OpenML.

(1) *Thyroid disease data. (OpenML CC-18)* The data comes from thyroid disease records, with totally 29 classification attributes and 3772 instances. This is an imbalanced binary classification data set as shown in Figure 2(a).

(2) *Glass identification database data.* The data comes from the analysis of glass composition, with totally 9 classification attributes and 137K instances as shown in Figure 2(b).

[2]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.

We also permit users to input their own data set in the specific format. The data should be uploaded in a classification data set format, with each row as a tuple, where the last attribute represents the class of this tuple. In our system, 80% of the data set is used as the training set and 20% as the test set.

Demo scenarios. Assassin first allows users to choose algorithm selection mode. After that, the data set is uploaded to Assassin. We use an example to illustrate the demonstration of these two modes, respectively.

Automation mode. This mode is designed with automation as the primary goal. As shown in Figure 2 (a), after uploading data set and choosing target attribute(which column represents the class label, first or last), our system automatically invokes the experience-based trained meta-classifier. The optimal algorithm (LMT here) is selected. Then users can click "next" for HPO and data set information visualization. Finally, Assassin gives a performance measure for recommended algorithm with metrics shown in Figure 2 (a).

Customization mode. In this mode, users can modify default parameters. After uploading data set, users can determine candidate algorithm list, meta-feature list manually selected or regenerated using DQN, time limit, or experience tasks number. As shown in Figure 2 (b), we limit the algorithm list to 'AdaBoost, ClassificationViaRegression, Logistic, NaiveBayes, RandomForest' and choose to retrain the DQN (middle picture shows retrained results). This mode allows users to retrain our system and give an optimal algorithm with user's setting. After modifying the parameters, Assassin runs the same process as Automation mode.

ACKNOWLEDGMENTS

This paper was supported by NSFC grant U1866602. Hongzhi Wang and Tianyu Mu contributed to the work equally and should be regarded as co-first authors. Hongzhi Wang is the corresponding author.

REFERENCES

[1] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, Feb (2012), 281–305.
[2] Noy Cohen-Shapira and et.al. Rokach. 2019. AutoGRD: Model Recommendation Through Graphical Dataset Representation. In *Proceedings of the 28th ACM CIKM*. 821–830.
[3] Tri Doan and Jugal Kalita. 2015. Selecting machine learning algorithms using regression models. In *2015 ICDMW*. IEEE, 1498–1505.
[4] Matthias et.al. Feurer. 2019. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*. Springer, Cham, 113–134.
[5] Taciana AF Gomes and et.al. Prudêncio. 2012. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing* (2012).
[6] Volodymyr Mnih, Koray Kavukcuoglu, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[7] Naoki Mori, Masayuki Takeda, and Keinosuke Matsumoto. 2005. 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*. 1485–1492.
[8] Cullen Schaffer. 1994. Cross-validation, stacking and bi-level stacking: Meta-methods for classification learning. In *Selecting Models from Data*. Springer, 51–59.
[9] et.al. Steinruecken. 2019. The automatic statistician. In *Automated Machine Learning*. Springer, Cham, 161–173.
[10] Chunnan Wang, Hongzhi Wang, Tianyu Mu, Jianzhong Li, and Hong Gao. 2020. Auto-Model: Utilizing Research Papers and HPO Techniques to Deal with the CASH problem. In *2020 IEEE 36th ICDE*. IEEE, 1906–1909.
[11] Anatoly Yakovlev and et.al Moghadam. 2020. Oracle automl: a fast and predictive automl pipeline. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3166–3180.