Model Hyper Parameter Tuning using Ant Colony Optimization

Aleksandar Trajkovski
FCSE, Ss Cyril and Methodius University, Skopje
Elevate Global LLC
Skopje, North Macedonia
aleksandar.trajkovski.2@students.finki.ukim.mk

Gjorgji Madjarov
FCSE, Ss Cyril and Methodius University, Skopje
Elevate Global LLC
Skopje, North Macedonia
gjorgji.madjarov@finki.ukim.mk

***Abstract*—The process of adjusting the best hyper-parameters for machine learning models is a complex optimization problem over the models that in reality are ”black-boxes”. This process entails several challenges such as the time complexity of finding optimal results, different types of hyper-parameters (continuous and categorical hyper-parameters) and model over-fitting. The main benefits of the hyper-parameter adjustment process are the improvement of the efficiency and quality of the models. This paper presents an implementation for adjusting hyper-parameters of time series prediction regression models, which is based on pheromone paths of ant colonies used as a way of finding food in nature. This approach of leaving pheromone pathways has been used to quickly find values for the hyper-parameters of the regression models for which they give the most optimal results.**

The proposed implementation for Ant Colony optimization is compared with three other parameter optimization approaches (Grid search, Bayesian optimization and Tree Parzen Estimators) on ten different regression datasets according to the time required to perform the optimization and the quality of the model prediction. The experimental evaluation shows that the proposed method needs minimal time to optimize the hyper-parameters, while preserving very good predictive performance in comparison to the competing approaches.

***Index Terms*—hyper-parameter optimization, hyper-parameters, Ant colony, Grid search, HPO, time series**

A. Acronyms

MSE - Mean Squared Error;
ACO - Ant Colony Optimizer;
SVR - Support Vector Regressor;
HGB - Histogram Gradient Boosting;
GS - Grid search;
RS - Random Search;
TPE - Tree Parzen Estimators;
HPO - Hyper Parameter Optimization;
BS - Bayesian Search;

I. Introduction

Machine learning models use parameters that cannot be directly deduced from the data. These types of parameters are called hyper-parameters of a model [1]. The overall work and behavior of the model in the learning and prediction phases greatly depend on these parameters. Their optimization is a difficult and tedious task, which depends on the expertise of the engineer and the domain of the problem the engineers are trying to solve with the help of data modeling. The hyper-parameter tuning is a process of choosing optimal values of the corresponding hyper-parameters, for which the model will have the best results for a predefined metric used for assessing the quality of the forecast. The most well-known and widely used technique for hyper-parameter tuning of models is the Grid Search technique, which gives the best results when searching the entire hyper-parameter space. However this technique has a high level of time complexity. To avoid the most common problems with selecting hyper-parameters values (manual adjustment by the engineer, use of ”recommended” values, use of inadequate metrics to assess the quality of the forecast and the occurrence of model over fitting), automatic tools are made that significantly aid in the tuning process and minimize the prediction error [2]. The purpose of this paper is to describe, review and experimentally compare approaches for hyper-parameter tuning on regression models which use time series as input data. In this paper the tools AutoSklearn, Hyper-Opt will be considered as representatives of Bayesian optimization and Tree Parzen Estimators respectively. Moreover as third and fourth method of hyper-parameter tuning evaluation, the method Grid search and the proposed implementation using the principle of pheromone paths of ant colonies will be used. The proposed implementation for the Ant Colony Optimizer can be easily adapted to any type of problem and model. Furthermore the same implementation can be used for parallel execution and skipping of combinations through the pheromone path system, which significantly increases the speed of parameter tuning.

II. Related work

Hyper-parameter optimization (HPO) is the final step in model design and the initial step in model training [3]. There are two types of hyper-parameter optimization:

- Manual:
  For models that are widely used, the adjustment of these parameters by the scientist is possible and depends on the experience of previously solved problems. With the increasing dimensionality of hyper-parameter space and its values, this approach becomes very difficult. To overcome this problem, a second type of optimization has been proposed.
- Automatic:
  As the search space and parameter settings expand, the demand for computing power grows, as does the necessity for continuous trial-and-error access. As a result of that new types of optimizers emerged such as Grid search (GS), random search (RS), bayesian search (BS) and ant colony optimization (ACO). For this kind of HPO, the scientist is expected to set a starting configuration for the process of optimization. Afterwards, the best hyper-parameters are automatically

chosen after certain time. With this method, the procedure of manually specifying the hyper-parameters and analyzing the outcomes is no longer necessary [4].

A. Automatic hyper-parameter optimization approaches

There are two primary types of automatic hyper-parameter tuning approaches, depending on whether they can be adapted to iterations [5]:

- non-iterative and non-adaptive
- iterative and adaptable

Non-iterative and non-adaptive approaches are represented by the GS (Grid search) method.

Grid search is one of the most utilized hyper-parameter tuning approaches. This fundamental approach of parameter tuning begins by defining a hyper-parameter search space that must be explored in order to achieve a certain global optimum. It is the most commonly utilized approach due to its mathematical simplicity and the ability to adjust parameters in parallel. The most significant disadvantages of this strategy are its high time complexity and the so-called curse of dimensionality. The number of combinations, and thus the resources required to calculate them, grows exponentially as the number of parameters to be adjusted grows [6].

Furthermore, Bayesian search (BS) and Tree Parzen Estimators (TPE) are representatives for the iterative and adaptable type of approach.

Most of the time, the functions in the models that require parameters are "black-box" functions. These functions are difficult to evaluate and demand a substantial amount of computer resources to identify the global extreme. To optimize these types of functions, Bayesian optimization is most typically utilized. With each evaluation and new sample, it grows more and more accurate over time. The basic idea behind this strategy is to establish a balance between exploration and exploitation. By balancing these two characteristics, the problem of becoming trapped in a local optimum is avoided. When compared to GS, the main advantage is that it can be utilized for a wide range of functions, including convex, non-convex, and stochastic functions. Bayesian optimization is comprised of two parts: a surrogate model for modeling a function and a posterior probability-dependent activation function that seeks to choose the next sample of the distribution. The surrogate model is often a Gaussian process that assumes similar inputs will result in similar outcomes. Unlike GS method, this approach for hyper-parameter tuning explicitly determines the error function [4], [6]–[8]. Finally, the Tree Parzen Estimator (TPE) is utilized as an alternative to a suitable surrogate Bayesian model source. This approach works with a variety of search spaces, including values from a uniform distribution, category spaces, and normally distributed real values represented by a tree structure. The inability to represent interactions between hyper-parameters is a significant shortcoming of this method. The basic method of the Gaussian process outperforms TPE for hyper-parameters with no interaction.

B. Related work for Ant Colony Optimization

In the literature, the ant colony optimization algorithm is most commonly used to solve the traveling salesman problem. This technique has recently been introduced to hyper-parameter tuning for hyper-parameters in neural network layers and the construction of complete neural architectures, as well as for improving predictions using fuzzy logic [9], [12]. The next paragraphs will go over some of the more noteworthy publications that utilize this concept. The Deep Swarm system, which is based on flock intelligence, is presented in [9] To discover the optimum neural network architectures, the authors apply the Ant Colony Optimizer (ACO) approach. They have discovered a balance in the process of exploitation and exploration by using this concept. The authors are using ACO and cluster intelligence for the process of NAS (Neuron Architecture Search) due to some benefits such as decentralization, scalability and the ability to share already acquired knowledge.

Deep Swarm begins by generating a graph with a single starting vertex. After a certain number of ants have been generated, the ACO tuning process begins. Each of these ants aims to determine what the next layer in neural architecture will be based on certain selection rules. Following this selection, each ant makes additional adjustments to the internal hyper-parameters of each layer, thereby resolving the HPO problem (Hyper parameter optimization). To a certain depth, this procedure is repeated for each ant. Following the completion of this procedure, the pheromone matrix is locally updated. Only the best ant is capable of updating the global pheromone matrix. The authors emphasize as an important point in the evaluation process that by limiting the number of ants to one, the research and exploitation of the search space does not show its full potential, but by increasing the number of ants from four to eight, the time required for evaluation increases significantly, and model improvement is minimal (less than 0.13%). Finally, the authors report error rates of 0.46%, 1.79%, and 1.68% when comparing the new system, RS search, and GS search, respectively.

The authors of [10] present the OpenNAS system, which generates CNN (convolutional neural networks) for data sets that contain black and white or colored images. PSO (partial cluster optimization) and ACO are used to select the layers for neural networks. The idea for such a system emerged from the brute force algorithm's impracticality and the problem's complexity. For ACO tuning, this before mentioned system OpenNAS utilizes an implementation from the previous Deep Swarm system. According to the authors, as the number of ants increases, so does the search time. For the configuration chosen in that paper, PSO outperforms ACO by 3% on one data set and 1% on the second data set. The authors point out that the networks generated by ACO are much simpler, but with relatively well-adjusted parameters, based on their configuration. The paper concludes that increasing the depth of an ACO network results in more complex and better networks, whereas choosing the number of layers in a PSO network is completely stochastic. Then, in [11], ACO is used to configure a modular neural architecture for image pattern recognition. The process of fine-tuning the modular neural architecture for image pattern recognition begins with the establishment of a colony of ants in various positions. For each of these ants, an adjustable graph is created. The offered implementation makes use of three modules, each with seven vertices. The final result is 98.82 percent pattern recognition accuracy. Finally, [12] uses a new heuristic to find a suitable region in the universe of discourse, which is a region in which values for certain observable variables are found. ACO, a widely accepted solution to the graph search problem, has been used to find such a suitable region. The paper utilizes time series with a high level of data noise. These are known as fuzzy series. Forecasting is based on TAIEX data and a special model for predicting this type of series, which is based on fuzzy logic.

III. Ant Colony optimizer

A. Pseudo algorithm for Ant colony optimization

Ant colony optimizer (ACO) is based on and inspired by ant behavior in the wild in search of food, as well as their communication via chemical pheromone pathways that lead from their nest to the food source. With this in mind, the optimizer's implementation is based on marking specific parts of the search space of the algorithm's hyper-parameters which will have the greatest contribution to reducing the prediction error. Initially, this algorithm was designed to solve the travel salesman problem, but later, specific variants of it were identified and used to optimize models using categorical, discrete, or continuous hyper-parameter values.

Algorithm 1 Ant colony optimization pseudo-code

```
Input: ($N$ No. of ants,$E$ function,max No. of iteration $n_{max}$)
  Configuration selection from the hyper-parameter space $P$
  Creation of matrix $NxP$
  Evaluation of solution matrix $S$ with function $E$
  Saving results into $R = E(S)$
  for $i = 1, 2, 3, n_{max}$ do
    Selection of new parameter configuration
    Creation of new solution matrix $S'$
    Evaluation of the newly created matrix $S'$
    Saving results into $R' = E(S')$
    Updating $S$ with the best solutions from $S$ and $S'$
Output: $S$, $R$
```

B. Proposed implementation for Ant Colony Optimization

The algorithm begins by creating a matrix of size NxP, where N is the number of selected ants for tuning and P is the number of hyper-parameters that need to be adjusted. This matrix is initially initialized with values chosen at random for the hyper-parameters. Following this step, the matrix is evaluated, and the results are placed in a new matrix of size Nx1. The RMSE metric is used to calculate how good the solution is. After calculating the metric value and storing the result in a matrix of size Nx1, the solutions of the matrix NxP are arranged in order of best to worst. The next step in the algorithm is to assign weights to each of the solutions. For this purpose, the following formula is used:

$$\frac{1}{qn * 2 * \pi} e^{\frac{-i^2}{2*q^2*N^2}} \tag{1}$$

where $qn = N * q$ and q is a parameter of the algorithm that is variable and can be experimented with, and i represents the ordinal number of each of the solutions after prior ordering from best to worst.

The Gaussian distribution is determined by this formula, which has a mean of 1 and a standard deviation of qn. The values of these weights are unchanged because they use the ordinal numbers of the solutions from 1 to N (the number of ants in the initial configuration). The algorithm then calculates the standard deviation for each of the parameters. A matrix of size Px1 is created for this purpose.This matrix will aid us in the creation of the Gauss kernel and the determination of hyper-parameter values. Furthermore, the pheromone evaporation parameter is introduced as a parameter that controls the solution's convergence.

Following the creation of the matrix with standard deviation, a new matrix of size NxP is created. To fill the hyper-parameter values, the previously generated standard deviation matrix is used in such a way that the value for the standard deviation corresponding to the parameter for which they are currently taken is multiplied by a randomly generated variable from the half-open interval [0,1), and this value is added to the value for this parameter from the best found solution so far. To avoid generating parameter values that are outside the parameter limits, all values that cross the upper or lower limit for a parameter are assigned the maximum or minimum value of the limits. The newly generated matrix is sent to the optimization process in this manner. This procedure runs all models on a predetermined number of processor cores. After completing the optimization and obtaining a matrix with metric values, the new solutions are combined with the best solutions thus far and sorted from best to worst. Only N of these solutions are saved, and the procedure is repeated z times until the maximum number of iterations is reached or the change in improvement does not exceed the threshold limit defined at the start. With the selection of N best solutions from $2{*}N$ solutions a balance between research and exploitation is introduced.

IV. Experimental evaluation

A. Model representatives and expectations

For the purpose of experimental evaluation as regression model representatives SVR (Support Vector Regression) and HGB (Histogram Gradient Boosting) models were chosen. SVR, or Support Vector Regression, is a supervised machine learning method. The primary goal of this type of prediction method is to locate a hyper plane in space that contains the greatest number of points used in training. If the problem cannot be solved linearly, the kernel function is used to find this plane, which aims to transcend the inputs to a higher dimension where they can be solved linearly. In the literature, RBF is most commonly used as a kernel function to generalize models. During the experiments, the RBF (Radial basis function) kernel function was used in each model. SVR differs from other types of regression because it seeks the best hyperplane with a predetermined limit, rather than focusing on implicitly minimizing the error between the actual and predicted values. When there are a large number of samples, SVR has a significant disadvantage in terms of complexity and training time. This model has the advantage of being resistant to anomalous points. In contrast, histogram gradient boosting is a decision tree ensemble model. It is distinct from other types of Boosting algorithms in that it discretizes continuous input values into a set of distinct values. This accelerates the training of the trees added to the ensembles. The disadvantage is that if the number of iterations is very large, over-fitting can occur. Furthermore, this model is not robust in terms of anomalous points, which are usually the result of a minor over-fitting. This is due to the fact that each new tree is updated based on the residuals and prediction errors of previous trees.

The training time of the SVR model is expected to be significantly faster for a smaller data set, while the training time of the HGB model is expected to be faster as the number of training data increases. The experimental evaluation in this paper consists of setting parameters for the two types of regressors (SVR and HGB) mentioned above using four methods: Grid Search, HyperOpt, Auto-Sklearn, and the proposed Ant Colony optimizer.

B. Data sets

Ten data sets from the UCI Machine learning repository [13] with varying sampling intervals and observations ranging from 61 to 69681 samples were chosen for a thorough evaluation of the proposed optimization algorithm.

Summary results for both types of models will be presented for each set, sorted by the chosen Mean Squared Error (MSE) comparison metric and the time required to find a solution.

TABLE I
DATASET NAME AND REFERENCE

Dataset Name	Reference
ai4i2020	[14]
Istanbul Stock Exchange	[15]
Garments worker productivity	[16]
Air Quality UCI	[17]
Energy Data	[18]
PRSA	[19]
Daily Demand Forecasting Orders	[20]
Metro Interstate Traffic Volume	[21]
ETT	[22]
Household Power Consumption	[23]

C. Metrics and hyper-parameters

For the purpose of model comparison Mean Squared Error (MSE) and time required to find a solution are used. Mean Squared Error (MSE) is the mean root value of the difference between the actual and the predicted value. A lower value with this metric determines a better model. The formula for this metric is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2 \qquad (2)$$

where MSE is $MeanSquaredError$, n is the number of samples, Y are the true values and $\widehat{Y}$ are the predicted values.

MSE error minimization is accomplished by calculating the MSE metric in regards to 80% of the total data without shuffling from each data set (due to the problem of continuous data in time series) and determining the mean value of the two trainings obtained through 2-fold cross validation. Following the tuning process on the training data sets, the value for the previously mentioned metric MSE, is calculated on the leftover 20% of the data sets.

SVR parameters C (regulatory parameter) and epsilon (a parameter that determines the extent to which no error is provided in the training function for observations provided in the interval from epsilon to the real value for that observation) are used as hyper-parameters for tuning. In all experiments for SVR, the kernel function is set to RBF (radial basis function). In contrast, the following parameters are chosen for Histogram Gradient Boosting (HGB): learning rate (parameter that controls the model's learning), maximum leaf nodes (minimum number of sheets in each tree), minimum sample leaf (minimum number of samples per sheet in wood), and max depth (maximum depth of each tree).

V. RESULTS AND DISCUSSION

A. Results

According to Table II, the proposed implementation ACO method is the most accurate in six of the ten data sets. On the other hand, the proposed implementation ACO for the HGB model (Histogram Gradient boosting) is the best when compared by MSE in three out of ten cases, equating with Grid search and Auto-Sklearn.

According to Table III, the proposed implementation ACO method is the fastest in six of the ten data sets. On the other hand, the proposed implementation ACO for the HGB model (Histogram Gradient boosting) is fastest when compared by time needed for tuning in nine out of ten cases.

Finally, whether using the SVR or HGB model, the proposed ACO implementation is always among the top three best methods.

B. Discussion

After evaluating the approaches made over the ten data sets that differ in the number of samples and the number of features they possess, as well as the size of the values in the predicted variable, it can be concluded that in seven of the ten SVR models the fastest solution is the proposed architecture. For the MSE metric ranking in many cases where ACO is in second place, it is preceded by GS, which is an expected result because the implementation of ACO generates approximate values for the parameters that in the experimental phase were set to have the above limit of 1000 for parameter C and 1 for epsilon parameter in SVR models. With that in mind, ACO is most accurate by MSE in six out of ten experiments, and Grid Search is most accurate in two out of ten cases. In one case, because no data preprocessing was performed, only Auto-Sklearn produced good results with a budget of 100 seconds and utilization of the frame's internal preprocessor. In the final experiment, the HyperOpt framework produces an MSE that differs from the ACO solution by one decimal place.

In nine out of ten experiments, ACO produces significantly faster results for the second type of model, HGB (Histogram Gradient Boosting). The results for this model are very close to MSE (differ by a third decimal place).

Another finding is that GS is the slowest of all models. When experimenting with Auto-Sklearn and HyperOpt, it is necessary to set a budget, which in Auto-Sklearn is fully utilized, whereas in HyperOpt it is possible but not always fully utilized. Determining the exact budget is difficult, which is why in this experiment, it is set to 500 and 100 seconds in Auto-Sklearn and 500 seconds in HyperOpt, respectively, and these numbers are chosen arbitrarily.

VI. CONCLUSION AND FURTHER WORK

In this paper we presented an implementation for Ant Colony Optimizer for hyper-parameter tuning. This implementation was compared with three different benchmark methods (Grid Search, Bayesian Optimization and Tree Parzen Estimators) on ten different datasets and on two different types of models (SVR and HGB).

The ant colonies optimization strategy for hyper-parameter tuning showed minimal time to tune the hyper parameters while providing competitive results for prediction accuracy compared with the other method mentioned above. It also has the potential for a lot of parallelism as well. Its accuracy is

TABLE II
RESULTS FOR MSE METRIC FOR MODELS SVR AND HGB FOR EACH OF THE TEN DATASETS

Dataset	**SVR MSE**				**HGB MSE**			
Method	**ACO**	**Grid search**	**AutoSklearn**	**HyperOpt**	**ACO**	**Grid search**	**AutoSklearn**	**HyperOpt**
Data set 1	84.41	**67.75**	71.95	94.66	2.17	**1.55**	2.76	3.94
Data set 2	**0.06**	0.07	0.06	0.069	**0.05**	0.051	0.05	0.05
Data set 3	**0.02**	0.43	0.13	0.43	**0.01**	0.01	0.01	0.01
Data set 4	0.24	0.32	0.23	**0.22**	0.29	0.29	**0.28**	0.29
Data set 5	2.95	2.17	**0.27**	3.50	0.37	0.36	0.48	**0.35**
Data set 6	6612.90	**6612.86**	8889.93	6616.26	6183.95	**6133.22**	6202.17	6212.68
Data set 7	**555.61**	574.62	989.49	573.82	1622.55	1669.85	**1069.75**	1669.85
Data set 8	**3658869.40**	3660087.39	3871753.23	3660049.28	3543492.39	3536480.03	**3533824.67**	3547464.79
Data set 9	**61.13**	61.37	60.90*	2225.12	67.94	**63.97**	68.66	67.94
Data set 10	**564.97**	565.82	72533.10	612.22	**2111.98**	2862.77	1163.81*	3079.90

* DummyRegressor used leading to not valid results.

TABLE III
RESULTS FOR TIME NEEDED FOR HYPER-PARAMETER TUNENING FOR MODELS SVR AND HGB FOR EACH OF THE TEN DATASETS

Dataset	**SVR Time**				**HGB Time**			
Method	**ACO**	**Grid search**	**AutoSklearn**	**HyperOpt**	**ACO**	**Grid search**	**AutoSklearn**	**HyperOpt**
Data set 1	**88.12726**	263.16185	500	350	**4.67375**	524.975	100	90
Data set 2	**0.46472**	4.19407	100	340	**2.204**	268.064	100	100
Data set 3	**0.68046**	0.89757	100	340	**2.83316**	328.047	500	100
Data set 4	**16.147**	57.844	100	344	**4.37914**	506.48	100	100
Data set 5	**357.07076**	2049.26036	500	400	**60.26691**	858.13465	100	100
Data set 6	898.712	3348.097	**100**	357	**6.81851**	649.3429	100	100
Data set 7	**1.63201**	2.24134	100	340	**5.48762**	144.80978	100	100
Data set 8	553.03885	4077.37868	500	**354**	**5.16121**	396.8439	100	101
Data set 9	3353.06297	17669.359	500	**409***	**6.21469**	955.67905	100	102
Data set 10	1218.08134	2804.524	500	**340**	384.21429	592.46702	500	**100**

* Ends on first iteration due to the big number of samples in the dataset.

heavily reliant on the parameter boundaries. These boundaries are arbitrarily chosen and set in all experiments without taking into account problem modeling knowledge. In practice, these boundaries would be precisely defined based on the set, model, and the domain knowledge.

Moreover in this paper only the concept of parameter tuning is applied to two previously selected models in the domain of time series regression. In addition to this paper, it is possible to implement an AutoML (Automated Machine Learning) system, which is a system for searching algorithms and setting hyper-parameters for them without any human intervention [24]. Such systems consist of several processes, such as preparing the dataset, generating key features from that set, generating machine learning models, optimizing them, and finally evaluating them. In the literature, the part of the selection of machine learning models and their optimization is known as the Combined Algorithm Selection and Hyperparameter Optimization, or CASH for short.

REFERENCES

[1] M. Kuhn and K. Johnson, Applied Predictive Modeling. Springer Science Business Media, 2013.

[2] "objective function." https://xlinux.nist.gov/dads/HTML/objective.html

[3] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," arXiv.org, Mar. 12, 2020.

[4] "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization," Journal of Electronic Science and Technology, vol. 17, no. 1, pp. 26–40, doi: 10.11989/JEST.1674-862X.80904120.

[5] "Best Practices for Hyperparameter Tuning with MLflow," Databricks, May 06, 2019. https://databricks.com/session/best-practices-for-hyperparameter-tuning-with-mlflow

[6] "Hyperparameter Optimization With Random Search and Grid Search," Machine Learning Mastery, Sep. 13, 2020.

[7] P. I. Frazier, "A Tutorial on Bayesian Optimization," arXiv.org, Jul. 08, 2018. https://arxiv.org/abs/1807.02811.pdf

[8] A. Agnihotri and N. Batra, "Exploring Bayesian Optimization," Distill, vol. 5, no. 5, May 2020, doi: 10.23915/distill.00026

[9] E. Byla and W. Pang, "DeepSwarm: Optimising Convolutional Neural Networks using Swarm Intelligence," arXiv.org, May 17, 2019. https://arxiv.org/abs/1905.07350.pdf

[10] S. Lankford and D. Grimes, "[PDF] Neural Architecture Search using Particle Swarm and Ant Colony Optimization".

[11] "Ant colony optimization for the design of Modular Neural Networks in pattern recognition," IEEE Xplore.https://ieeexplore.ieee.org/document/7727194

[12] Q. Cai, D. Zhang, W. Zheng, and S. C. H. Leung, "A new fuzzy time series forecasting model combined with ant colony optimization and auto-regression," Knowledge-Based Systems, vol. 74, pp. 61–68, Jan. 2015, doi: 10.1016/j.knosys.2014.11.003.

[13] "UCI Machine Learning Repository." https://archive.ics.uci.edu/ml/index.php

[14] "UCI Machine Learning Repository: AI4I 2020 Predictive Maintenance Dataset Data Set." https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset

[15] "UCI Machine Learning Repository: ISTANBUL STOCK EXCHANGE Data Set." https://archive.ics.uci.edu/ml/datasets/ISTANBUL+STOCK+EXCHANGE

[16] "UCI MLR: Productivity Prediction of Garment Employees Data Set." https://archive.ics.uci.edu/ml/datasets/Productivity+Prediction+of+Garment+Employees

[17] "UCI Machine Learning Repository: Air Quality Data Set." https://archive.ics.uci.edu/ml/datasets/Air+Quality

[18] "Appliances energy prediction," data.world, https://data.world/uci/appliances-energy-prediction

[19] "UCI Machine Learning Repository: Beijing PM2.5 Data Data Set." https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

[20] "UCI Machine Learning Repository: Daily Demand Forecasting Orders Data Set." https://archive.ics.uci.edu/ml/datasets/Daily+Demand+Forecasting+Orders

[21] "UCI Machine Learning Repository: Metro Interstate Traffic Volume Data Set." https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume

[22] zhouhaoyi, "GitHub - zhouhaoyi/ETDataset:" GitHub. https://github.com/zhouhaoyi/ETDataset

[23] "Household power consumption." https://archive.ics.uci.edu/ml/machine-learning-databases/00235/

[24] I. Guyon et al., "A Brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning Without Human Intervention," PMLR, Dec. 04, 2016. https://proceedings.mlr.press/v64/guyon_review_2016.html https://arxiv.org/abs/2003.05689.pdf