Hyper-Parameter Optimization using PSO on Machine Learning Models

A DISSERTATION
SUBMITTED FOR THE MID TERM PROJECT EVALUATION (MTE)
FOR THE COMPLETION OF COURSE SWARM and Evolutionary Computing (CO-423)
IN
COMPUTER ENGINEERING

Submitted by:
Sourabh
2K18/CO/355
Tanyam Singhal
2K18/CO/371

Under the supervision of
Ms. Nishtha Ahuja

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi- 110042
November, 2021

DECLARATION

I, Sourabh, Roll No. 2K18/CO/355 along with Tanyam Singhal, Roll No. 2K18/CO/371 of B.Tech Computer Engineering, hereby declare that the project titled "Hyper-Parameter Optimization using PSO on Machine Learning Models" which is submitted by me to the Department of Computer Engineering, Delhi Technological University (DTU), Delhi for the mid-term project evaluation (MTE), is unique and not derivative from any source without proper citation.

New Delhi

INDIA

November, 2021

Abstract

Machine learning algorithms have been widely used in various programs and environments. In order to incorporate a machine learning model into various problems, their hyper-parameters must be tuned. Choosing the best parameter setting for machine learning models has a direct impact on model performance. It usually requires in-depth knowledge of machine learning algorithms and appropriate techniques for improving the parameter. Although several automation techniques are available, they have different strengths and obstacles when applied to different types of problems. In this project, optimization of hyper-standard machine learning parameters is studied. Many available libraries and frameworks for hyper-parameter optimization problems are provided, and the development of hyper-parameter optimization particle research is also discussed in this paper. In addition, tests were performed on benchmark data sets to compare the effectiveness of different development methods and provide practical examples for improving the hyper parameter. This project will help industrial users, data analysts, and researchers to better develop machine learning models by identifying the proper hyper-parameter configurations effectively.

List of Contents

1. Objective

Different ML algorithms are suitable for different types of problems or datasets. In general, building an effective machine learning model is a complex and time-consuming process that involves determining the appropriate algorithm and obtaining optimal model architecture by tuning its hyper-parameters (HPs).

The hyper-parameters of a machine learning model must be modified to fit different tasks. For machine learning models, choosing the appropriate hyper-parameter configuration has a direct impact on the model's performance.

It is crucial to select an appropriate optimization technique to detect optimal hyper-parameters. Traditional optimization techniques may not be suitable for HPO problems, since many HPO problems are not convex or differentiable optimization problems, and may result in a local instead of a global optimum.

In this study, we used particle swarm optimization to optimize the hyper parameters of typical machine learning models such as random forest, SVM, and KNN.

2. Introduction

Hyper Parameter

In machine learning models, there are two types of parameters: model parameters, which can be initialised and updated through the data learning process (for example, the weights of neurons in neural networks); and hyper-parameters, which cannot be directly estimated from data learning and must be set before training an ML model because they define the architecture of an ML model.Hyper-parameters are parameters that are used to either establish a machine learning model (for example,fine C parameter on the vector support machine and the level of learning to train the neural network) or define the algorithm used to reduce the loss function (e.g., activation function and types of optimizer on the neural network, and kernel type on vector machine support).

Hyper Parameter Tuning

A variety of options must be studied in order to develop an optimal ML model. Hyper-parameter tuning is the process of creating the ideal model architecture with the best hyper-parameter configuration. Tuning hyper-parameters is a critical part of creating a successful machine learning model, especially for tree-based ML models and deep neural networks, which contain a lot of them. Because different ML algorithms use different types of hyper-parameters, such as categorical, discrete, and continuous hyper-parameters, the hyper-parameter tuning procedure varies. Manual testing is a conventional method of tuning hyper-parameters that is still employed in graduate student research, despite the fact that it necessitates a thorough understanding of the ML algorithms and their hyper-parameter value settings. However, due to a high number of hyper-parameters, complex models, time-consuming model evaluations, and non-linear hyper-parameter interactions, manual tuning is unsuccessful for many issues.

3. Hyper-Parameter Optimization

These factors have prompted more research into strategies for automatic hyper-parameter optimization, also known as hyper-parameter optimization. The major goal of HPO is to automate the hyper-parameter tuning process and allow users to successfully apply machine learning models to real-world issues. After an HPO process, the ideal model architecture of an ML model should be obtained.The following are some of the reasons to use HPO approaches on ML models:

1. Since many ML developers spend a significant amount of time tuning hyper-parameters, especially for large datasets or complicated ML algorithms with a high number of hyper-parameters, it decreases the amount of human effort necessary.
2. It boosts the performance of machine learning models. Many machine learning hyper-parameters have distinct optimums for different datasets or situations.
3. It improves the reproducibility of models and studies. Multiple ML algorithms can only be compared properly when the same level of hyper-parameter tuning technique is used; thus, utilising the same HPO approach on different ML algorithms also aids in determining the best ML method for this given project.

4. Various Techniques Available

Various hyper-optimization techniques available which could prove out to be useful and have been somewhat used in previous research are as follows:

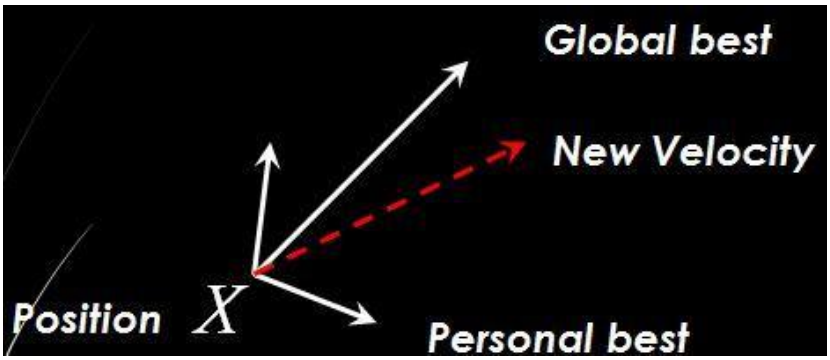
- Grid search,
- Random search,
- Hyper-band,
- Bayesian Optimization with Gaussian Processes (BO-GP),
- Bayesian Optimization with Tree-structured Parzen Estimator (BO-TPE),
- Particle swarm optimization (PSO),
- Genetic algorithm (GA).

All of them provide a basis to optimize hyper-parameters in a different way and have satisfactory results, but we employed particle swarm optimization method for the purpose of hyper-optimization.

5. Particle Swarm Optimization

Particle swarm optimization (PSO): Each particle in swarm interacts with other particles to obtain and update current global values in each recurrence until a maximum is achieved. Particle swarm optimization (PSO) is another set of evolutionary algorithms commonly used for development problems. PSO algorithms are motivated by biological statistics that reflect both individual and social behavior. The PSO works by enabling a group of particles (clusters) to cross the search area in a random way. PSO algorithms identify the optimal solution by interacting and sharing information between each particle in a group.

The basic concept of PSO lies in accelerating each particle toward its pbest(the best solution achieved so far by that particle) and the gbest(the best value obtained so far by any particle) locations, with a random weighted acceleration at each time. PSO uses a number of agents, i.e., particles that constitute a swarm flying in the search space looking for the best solution.



Each particle is considered to be a point (solution of the chosen one) in the N-dimensional space that adjusts its "flight" according to its flight information and flight information from other particles.

The Search Scheme of PSO

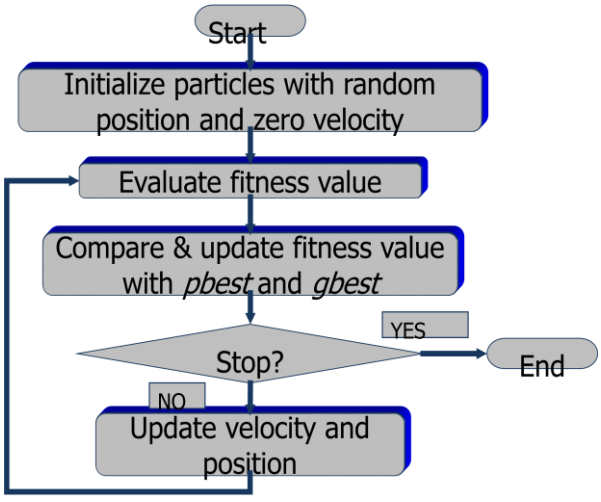
Each particle tries to modify its position X using the following formula:

1. $X(t+1) = X(t) + V(t+1)$

2. $V(t+1) = wV(t) + c_1 \times rand() \times (X_{pbest} - X(t)) + c_2 \times rand() \times (X_{gbest} - X(t))$

$V(t)$	velocity of the particle at time t
$X(t)$	Particle position at time t
w	Inertia weight
c_1 , c_2	learning factor or accelerating factor
rand	uniformly distributed random number between 0 and 1
X_{pbest}	particle's best position
X_{gbest}	global best position

The Algorithm



The PSO algorithm pseudocode is given as follows:

I/P: Randomly initialized position and velocity of Particles: Xi (0) and Vi (0)

O/P: Position of the approximate global minimum X

1: while terminating condition is not reached do

2: for i = 1 to number of particles do

3: Calculate the fitness functionf

4: Update personal best and global best of each particle

5: Update velocity of the particle using Equation 2

6: Update the position of the particle using equation 1

7: end for

8: end while

PSO v/s GA

Compared with GA, it is easier to implement PSO, since PSO does not have certain additional operations like crossover and mutation. In GA, all chromosomes share information with each other, so the entire population moves uniformly toward the optimal region; while in PSO, only information on the individual best particle and the global best particle is transmitted to others, which is a one-way flow of information sharing, and the entire search process follows the direction of the current optimal solution. The computational complexity of PSO algorithm is O(nlogn). In most cases, the convergence speed of PSO is faster than of GA. In addition, particles in PSO operate independently and only need to share information with each other after each iteration, so this process is easily parallelized to improve model efficiency.

The biggest limitation of the PSO is that it requires proper human implementation; otherwise, it may reach only the local area instead of the global beauty, especially at different hyper parameters. Proper implementation of a census requires prior engineer knowledge or application of population-based strategy. Many demographic methods have been proposed to improve the performance of evolutionary algorithms, such as the development-based development algorithm and the space conversion search method. Involving additional ways of initiating population growth will require more time to practice and resources.

Supports all types of hyper parameters and works best in large suspension spaces, as they can find the closest solutions even within very few repetitions. The PSO is capable of supporting great similarities, and is particularly well suited to ongoing and conditional HPO issues.

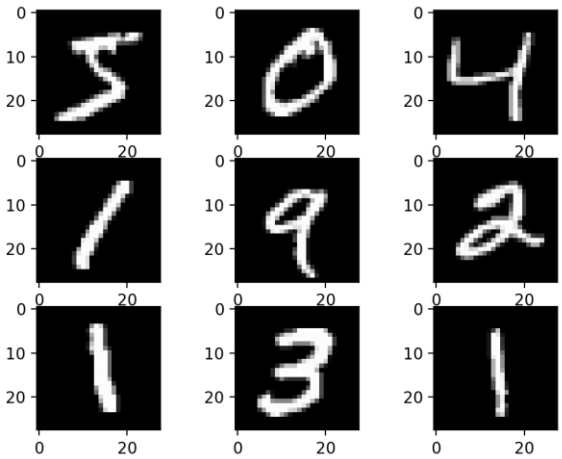
6. Dataset

Dataset used is MNIST from sklearn, The MNIST dataset summary representing the Modified National Institute of Standards and Technology dataset. It is a database of 60,000 photographs of a small 28 × 28 pixel grayscale one digit handwritten between 0 and 9.

```
datasets.load_digits()

 'pixel_7_3',
 'pixel_7_4',
 'pixel_7_5',
 'pixel_7_6',
 'pixel_7_7'],
'frame': None,
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
        [ 0.,  3., 15., ..., 11.,  8.,  0.],
        ...,
        [ 0.,  4., 11., ..., 12.,  7.,  0.],
        [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],
```

This is a snippet of the loaded dataset from MNIST which clearly shows some of the features like frames and the images.



7. Model Training

This is basically a general Machine learning code. We load the dataset, you can see a snippet of it. Now, initially we tested the classifiers without hyper-optimization which means that hyper-parameters are set to default. We fitted the models and calculated the accuracy for all three classifiers, here it is for random forest then for svm this for knn. After this we optimize the hyper parameters and see the results foe all three classifiers.

There are many existing HPO frameworks like sklearn, nevergrad, deap, TPOT, spearmint. We have used optunity. Optunity is a well-known HPO framework that provides a number of development strategies, including GS, RS, PSO, and BO-TPE. In Optunity, hyper-categorical parameters are converted into hyper-discrete parameters by reference, and hyper-discrete parameters are considered as continuous parameters; thus, it supports all types of hyper-parameters.

ML Algorithms Used

The following three Machine Learning algorithms were employed for the purpose:

- **Random forest (RF)**

  Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below). The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well. A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. Some of the prerequisites for random forest to perform well are: 1. There needs to be some actual signal in our features so that models built using those features do better than random guessing. 2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

- **Support vector machine (SVM)**

  Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, used for Scheduling and retraining problems. However, it is mainly used for Distribution Problems in Machine Learning. The goal of the SVM algorithm is to create a better line or decision line that can divide n-dimensional space into classes so that we can easily place a data point in the appropriate category in the future. This best decision limit is called the hyperplane. SVM selects the extra points / vectors that help create the hyperplane. These extreme cases are called supporting vectors, which is why the algorithm is called Vector Support Machine. Consider the diagram below in which there are two distinct categories divided by resolution or hyperplane:

- **K-nearest neighbor (KNN)**

  K-Nearest Neighbor is one of the simplest machine learning algorithms based on supervised learning strategies. The K-NN algorithm captures the similarities between new cases / data and available cases and places a new case in the category that closely resembles available categories. The K-NN algorithm stores all available data and classifies new data points based on similarities. This means that where new data comes from it can be easily categorized into a well suite section using the K-NN algorithm. The K-NN algorithm can be used for Undo and Editing but mainly for Editing problems. K-NN is a non-parameter algorithm, which means it does not make any assumptions about root data.

8. Performance Evaluation

Performance Metrics

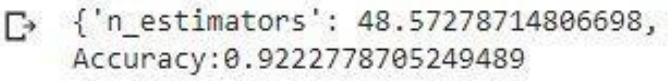
Performance Metric used was classification accuracy Classification accuracy. It is a metric that summarizes the performance of the division model as the correct guess number is divided by the number of predictions. It is easy to calculate and accurate to understand.To implement hpo, we have have used optunity which is an existing framework. It is suitable for pso and genetic algorithm as well. So while defining the objective function optunity is used by defining features like x_train, y_train, x_test, y_test,n_estimators=None, max_features=None,max_depth=None,min_samples_split, etc.

Results

1. Random Forest

   First for random forest, Initially we define the hyper prameter configuration space. Then we define the objective function, followed by fitting the model. Finally we predict the accuracy.

(With default hyperparameters)



(After optimization)

2. SVM

For SVM the hyper parameters of primary importance are penalty parameter C. So before fitting the model it is handled along with kernel type. Further model predictions are made and we can see the results as well.

```
Accuracy:0.9699499165275459
```

(For classifiers with default hyperparameters)

```
{'C': 15.2099609375, 'kernel': 2.435546875}
Accuracy:0.9738452977184195
```

(After optimization)

3. KNN

For KNN, the hyper parameter is k, the number of neighbors considered. So it is defined in beginning. Further the model is fit and the results are predicted in form of accuracy.

```
Accuracy:0.9627156371730662
```

(For classifiers with default hyperparameters)

```
{'n_neighbors': 3.49560546875}
Accuracy:0.9682804674457429
```

(After optimization)

We can see that results have been improved after hyperoptimization.

9. Future Scope

Although there are many existing HPO algorithms and functional frameworks, some problems still need to be addressed, and a few aspects of this domain can be improved. In this section, we discuss open challenges, current research questions, and potential future research guides.

Model Complexity

1. Costly Objective Function Evaluations

To evaluate the performance of a ML model with different hyper-parameter configurations, its objective function must be minimized in each evaluation. Depending on the scale of data, the model complexity, and available computational resources, the evaluation of each hyper-parameter configuration may take several minutes, hours, days, or even more.

2. Complex Search Space

In most cases where ML algorithms are used, only a few parameters have significant effects on model performance, and are large parameters that need to be adjusted. However, some hyper-parameters may interfere with performance and may be considered as continuous ML model enhancement, which increases the search space of the hyper parameter. As the number of hyper-parameter parameters and their values increases, they greatly increase the size of the search space and the complexity of the problems, and the total time to test the function of the object will also increase significantly.

Model Performance

1. Strong Anytime Performance and Final Performance

HPO techniques are often expensive and sometimes require more resources, especially for large databases or complex ML models. One example of a utility model is in-depth learning models, as they look at analyzing purpose work as black box functions and not considering their complexity. However, the overall budget is usually very limited in most practical contexts, so effective HPO algorithms should be able to prioritize objective performance assessments and have strong performance at any given time, demonstrating the ability to get complete or close-fitting and even more limited preparation budget.

2. Comparability of HPO Methods

To improve the hyper-models of ML models, different development algorithms can be used for each ML framework. Different optimization techniques have their strengths and barriers in different situations, and at the moment, there is no one best practice that surpasses all other methods when processing different data sets with different metrics and hyper parameter types.

10. Conclusion

In order to use ML models for real-world problems, their advanced parameters need to be tuned to fit specific data sets. However, as the scale of the generated data has increased significantly in real life, and manual manipulation parameters are more expensive, it has become necessary to increase the hyper parameters by default. In this survey paper, we discussed in more detail the advanced research in the field of hyper-parameter optimization and how to apply it to different ML models in theory and practice testing. Other available HPO tools and frameworks, open challenges, and potential research guides are also provided and highlighted for practical use and future research purposes. We also hope it helps to develop an understanding of the challenges that still exist within the HPO domain, and in doing so improve HPO and ML applications in future research.

References

1. L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” Neurocomputing, vol. 415, pp. 295–316, 2020, doi: https://doi.org/10.1016/j.neucom.2020.07.061.

2. Feurer, Matthias, and Frank Hutter. "Hyperparameter optimization." Automated machine learning. Springer, Cham, 2019. 3-33.

3. M.-A. Z¨oller and M. F. Huber, Benchmark and Survey of Automated Machine Learning Frameworks, arXiv preprint arXiv:1904.12054, (2019). https://arxiv.org/abs/1904.12054.

4. R. E. Shawi, M. Maher, S. Sakr, Automated machine learning: Stateof- the-art and open challenges, arXiv preprint arXiv:1906.02287, (2019). http://arxiv.org/abs/1906.02287.

5. M. Kuhn and K. Johnson, Applied Predictive Modeling., Springer (2013) ISBN: 9781461468493.

6. G.I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, H. Samulowitz, An effective algorithm for hyperparameter optimization of neural networks, IBM J. Res. Dev. 61 (2017) 1–20. https://doi.org/10.1147/JRD.2017.2709578.

7. F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., Automatic Machine Learning: Methods, Systems, Challenges, Springer (2019) ISBN: 9783030053185.

8. N. Decastro-Garc´ıa, ´A. L. Mu˜noz Casta˜neda, D. Escudero Garc´ıa, and M. V. Carriegos, Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm, Complexity 2019 (2019). https://doi.org/10.1155/2019/6278908.

9. S. Abreu, Automated Architecture Design for Deep Neural Networks, arXiv preprint arXiv:1908.10714, (2019).

10. J.O. Ogutu, T. Schulz-Streeck, H.P. Piepho, Genomic selection using regularized linear regression models: ridge regression, lasso, elastic net and their extensions, BMC Proceedings. BioMed Cent. 6 (2012).

11. J.M. Keller, M.R. Gray, A Fuzzy K-Nearest Neighbor Algorithm, IEEE Trans. Syst. Man Cybern. SMC-15 (1985) 580–585. https://doi.org/10.1109/TSMC.1985.6313426.

12. W. Zuo, D. Zhang, K. Wang, On kernel difference-weighted knearest neighbor classification, Pattern Anal. Appl. 11 (2008) 247–257. https://doi.org/10.1007/s10044-007-0100-z.
13. A. Smola, V. Vapnik, Support vector regression machines, Adv. Neural Inf. Process. Syst. 9 (1997) 155-161.

14. L. Yang, R. Muresan, A. Al-Dweik, L.J. Hadjileontiadis, Image-Based Visibility Estimation Algorithm for Intelligent Transportation Systems, IEEE Access. 6 (2018) 76728–76740. https://doi.org/10.1109/ACCESS.2018.2884225.