Hyperparameter Optimization Techniques for Machine Learning Using Diabetes data

Sweta Vashisth
Department of Computer Science and Engineering,
Lingaya's Vidyapeeth,
Faridabad, India

Md.Aftab Alam
Department of Computer Science and Engineering,
Lingaya's Vidyapeeth,
Faridabad, India

Manoj Yadav
Department of Computer Science and Engineering,
Lingaya's Vidyapeeth,
Faridabad, India

Abstract - Machine learning algorithms are widely used in various applications and areas. In order to adapt a machine learning model to different problems, its hyper parameters must be adjusted. When we select the best hyperparameter configuration for the machine learning model, it gives a direct impact on the performance and accuracy of the model. The optimal hyperparameter also lead to the better efficiency, faster convergence and better results.

It often requires a deep understanding of machine learning algorithms and appropriate high-parameter optimization techniques. Although there are several automatic optimization techniques, they have different strengths and disadvantages when applied to different types of problems. This paper studies the optimization of the hyperparameters of common machine learning models. We introduce several optimization techniques; these are Grid Search, Random Search, Bayesian optimization, Genetic algorithm, Optuna and discuss how to apply them to machine learning algorithms. To test the performance of various optimization methods and provides practical examples of hyperparameter optimization we use diabetes data set. Our results show that the Random forest Classifier offers the best accuracy after hyperparameter tuning, with the highest score of 82.42 obtained when using Optuna.

Keywords - Hyperparameter optimization, Machine learning, GridSearch, RandomSearch, Bayesian optimization, Genetic algorithm.

I. INTRODUCTION

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) [1]. Two types of parameters exist in machine learning models: one that can be initialized and updated through the data learning process (e.g., the weights of neurons in neural networks), named model parameters; while the other, named hyper-parameters, cannot be directly estimated from data learning and must be set before training a ML model because they define the architecture of a ML model [2]. Hyper-parameters are the parameters that are used to either configure a ML model (e.g., the penalty parameter in a support vector machine, and the learning rate to train a neural network) or to specify the algorithm used to minimize the loss function (e.g., the activation function and optimizer types in a neural network, and the kernel type in a support vector machine) [3].

To build an optimal ML model, a range of possibilities must be explored. The process of designing the ideal model architecture with an optimal hyper-parameter configuration is named hyper-parameter tuning. Tuning hyper-parameters is considered a key component of building an effective ML model, especially for tree-based ML models and deep neural networks, which have many hyper-parameters [4]. Hyper-parameter tuning process is different among different ML algorithms due to their different types of hyper-parameters, including categorical, discrete, and continuous hyper-parameters [5]. Manual testing is a traditional way to tune hyper-parameters and is still prevalent in graduate student research, although it requires a deep understanding of the used ML algorithms and their hyper-parameter value settings [6]. However, manual tuning is ineffective for many problems due to certain factors, including a large number of hyper-parameters, complex models, time-consuming model evaluations, and non-linear hyper-parameter interactions. These factors have inspired increased research in techniques for automatic optimization of hyper-parameters; so-called hyperparameter optimization (HPO) [7]. The main aim of HPO is to automate hyper-parameter tuning process and make it possible for users to apply machine learning models to practical problems effectively [1]. The optimal model architecture of a ML model is expected to be obtained after a HPO process. Some important reasons for applying HPO techniques to ML models are as follows [4]:

1. It reduces the human effort required, since many ML developers spend considerable time tuning the hyper-parameters, especially for large datasets or complex ML algorithms with a large number of hyper-parameters.
2. It improves the performance of ML models. Many ML hyper-parameters have different optimums to achieve best performance in different datasets or problems.
3. It makes the models and research more reproducible. Only when the same level of hyper-parameter tuning process is implemented can different ML algorithms be compared fairly; hence, using a same HPO method on different ML algorithms also helps to determine the most suitable ML model for a specific problem.

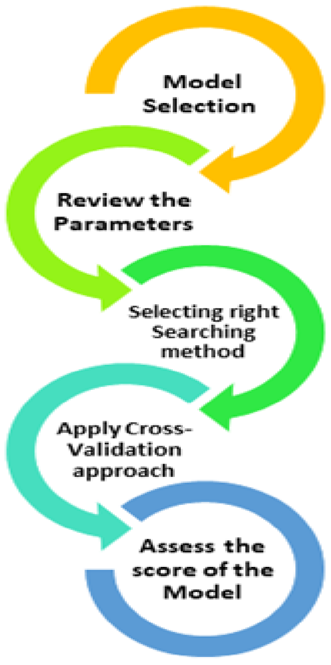
The objective of hyperparameter optimization techniques in machine learning is to find the best combination of hyperparameter values for a given model that maximizes its performance or achieves the desired outcome. Hyperparameters play a crucial role in determining the behavior and effectiveness of a machine learning algorithm, and choosing appropriate values for these hyperparameters is essential for obtaining accurate and reliable predictions.

By optimizing the hyperparameters, we aim to enhance the model's performance metrics, such as accuracy, precision, recall, or F1 score. The goal is to find the optimal configuration that balances model complexity, generalization, and computational efficiency to achieve the best possible trade-off.

Hyperparameter optimization techniques, such as grid search, random search, Bayesian optimization, or evolutionary algorithms, systematically explore the hyperparameter space to identify the combination of values that results in the highest model performance. By fine-tuning the hyperparameters, we can improve model accuracy, reduce overfitting, and ensure better generalization to unseen data. Ultimately, the objective is to maximize the model's predictive power and achieve superior performance on the task at hand.

Steps for perform Hyperparameter Tuning:

- Choose the right type of model.
- Examine the model's parameter list and construct the hyperparameter space.
- Find a way to search for hyperparameter space.
- Implement the cross-validation approach.
- Evaluate the model's performance to assess its effectiveness.


Model Selection
Review the Parameters
Selecting right Searching method
Apply Cross-Validation approach
Assess the score of the Model

In machine learning, the hyperparameter space refers to the range or set of values that can be assigned to the hyperparameters of a model. Hyperparameters are the configuration settings that determine how a machine learning algorithm operates, such as the learning rate, the number of hidden layers in a neural network, or the regularization parameter.

The hyperparameter space represents all the possible combinations or values that these hyperparameters can take. During the hyperparameter tuning process, different values from the hyperparameter space are selected and tested to find the optimal configuration that yields the best performance for a given machine learning task. This involves evaluating the model's performance using various combinations of hyperparameter values to identify the set of hyperparameters that produce the most accurate and reliable predictions.

II. Literature survey

Moshe Sipper [8] A Large Scale Study of Hyperparameter Tuning for machine learning algorithm by Moshe Sipper .This article provides a comprehensive study of Hyperparameter tuning and its impact on machine learning algorithm.
Yasser A. Ali , Emad Mahrous Awwad , Muna Al-Razgan and Ali Maarouf[9] this paper discuss the significance of hyperparameter search in machine learning algorithm and proposes a method for optimizing computational complexity and what are some of the latest advancements in hyperparameter search for machine learning algorithm.
Li Yang and Abdallah Shami [10] this paper is a survey of Hyperparameter optimization (HPO) techniques for machine learning algorithms. It discusses the importance of optimizing hyperparameters, the strengths and drawbacks of different automatic optimization techniques and provides state-of-the-art optimization techniques for common machine learning models.The paper also includes experiments on benchmark datasets to compare the performance of different optimization methods and provide practical examples of HPO.
Philipp Probst, Anne-Laure Boulesteix, Bernd Bischl [11] this paper discusses the importance of hyperparameters in machine learning algorithm and various options to setting them. The authors formalize the problem of tuning hyperparameters from a statistical point of view and conduct a large-scale benchmarking study to assess their tenability.

III. Methodology:

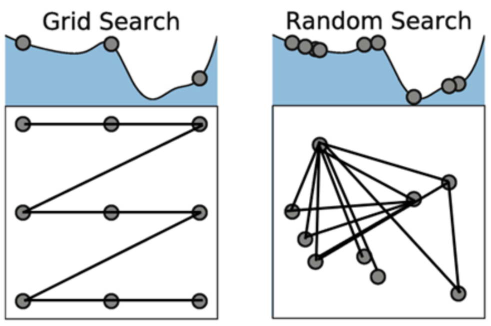
Hyperparameter tuning optimization techniques in machine learning aim to find the best combination of hyperparameter values for a given model. Here, we will discuss several common hyperparameter tuning techniques:

1. Grid Search (GS):

Grid search is a straightforward and exhaustive technique that involves defining a grid of possible hyperparameter values. It systematically explores all combinations of hyperparameters by evaluating the model's performance for each configuration. Grid search is easy to implement and guarantees that all possible configurations will be tested. However, it can be computationally expensive, especially in high-dimensional hyperparameter spaces.

2. Random Search (RS):

Random search selects hyperparameter values randomly from a predefined range for each iteration. Unlike grid search, it does not exhaustively explore all possible combinations. Instead, it focuses on randomly sampled configurations. Random search is computationally more efficient than grid search, especially when only a few hyperparameters have a significant impact on performance. It also has a higher chance of finding good configurations compared to grid search.



3. Bayesian Optimization (BO):

Bayesian optimization utilizes probabilistic models to guide the search process. It constructs a surrogate model to approximate the performance landscape of the model based on observed evaluations. By iteratively updating the model, it balances exploration (trying new configurations) and exploitation (focusing on promising regions). Bayesian optimization uses an acquisition function to determine the next hyperparameter configuration to evaluate. This technique adapts well to noisy or expensive evaluation functions and can efficiently handle high-dimensional search spaces.

4. Particle Swarm Optimization (PSO):

PSO is a population-based optimization technique inspired by the collective behavior of bird flocks or fish schools. In PSO, a group of particles represents potential solutions (hyperparameter configurations), and each particle adjusts its position based on its own experience and the best-known position in the swarm. This collective behavior enables PSO to efficiently explore the search space. PSO is suitable for continuous or discrete hyperparameters and has been applied to hyperparameter optimization tasks.

5. Genetic Algorithms (GA):

Genetic algorithms use a combination of selection, crossover, and mutation operations inspired by genetics and natural selection. In hyperparameter optimization, GA maintains a population of hyperparameter configurations as chromosomes. It applies selection to favor promising configurations, crossover to exchange genetic information, and mutation to introduce diversity. Genetic algorithms are flexible and can handle various types of hyperparameters, making them suitable for complex optimization problems.

6. Optuna:

Optuna is an automatic hyperparameter tuning software framework designed specifically for machine learning tasks. Hyperparameter tuning is the process of finding the best combination of hyperparameters (settings that are not learned by the model itself) for a given machine learning algorithm.

- Optuna provides a powerful and flexible platform to automate the search for optimal hyperparameter values. It offers various search algorithms and sampling methods to efficiently explore the hyperparameter space. By automatically searching for the best hyperparameters, Optuna helps improve the performance and accuracy of machine learning models.

- Optuna is compatible with popular machine learning frameworks such as PyTorch, TensorFlow, Keras, and SKlearn, making it easy to integrate into existing machine learning pipelines. It utilizes a define-by-run API, allowing users to write modular and dynamic code for constructing search spaces and defining objective functions.

- Overall, Optuna simplifies and automates the tedious process of hyperparameter tuning in machine learning, enabling researchers and practitioners to find optimal parameter configurations more efficiently.

Random Forest Algorithm:

Random Forest is a popular machine learning algorithm that belongs to the ensemble learning family. It is primarily used for classification and regression tasks. Random Forests have found applications in various fields, including finance, healthcare, image recognition, and natural language processing, due to their versatility, accuracy, and robustness.

The algorithm builds a collection of decision trees during the training phase. Each decision tree in the Random Forest is trained on a different subset of the data and considers only a subset of the available features. This process is known as bagging (bootstrap aggregating) and random feature selection.
The Random Forest algorithm has several hyperparameters that can be tuned to optimize its performance. Here are some of the key hyperparameters for the Random Forest algorithm:

1. n_estimators: This parameter specifies the number of decision trees to be included in the random forest. Increasing the number of trees can improve the model's performance, but it also increases computational complexity.

2. max_depth: It determines the maximum depth allowed for each decision tree in the random forest. Setting a higher value can lead to more complex trees, potentially increasing overfitting. Conversely, a lower value restricts the depth and can prevent overfitting.

3. min_samples_split: This hyperparameter specifies the minimum number of samples required to split an internal node during the construction of each decision tree. Increasing this value can help reduce overfitting by requiring a higher number of samples for a split.

4. min_samples_leaf: It sets the minimum number of samples required to be at a leaf node. Similar to min_samples_split, increasing this value helps control overfitting by ensuring a minimum number of samples in each leaf node.

5. max_features: This parameter determines the number of features to consider when looking for the best split at each node. It can be specified as a fixed number or a fraction of the total number of features. Smaller values reduce the randomness and make the algorithm more deterministic, while larger values introduce more randomness.

6. max_leaf_nodes: It sets the maximum number of leaf nodes in each decision tree. Constraining the maximum number of leaf nodes can prevent the trees from growing excessively and help control overfitting.

7. bootstrap: This parameter specifies whether to use bootstrapping when constructing individual decision trees. Bootstrapping involves sampling the training data with replacement, which introduces randomness and helps reduce variance.

8. Criterion: the function used to evaluate the quality of a split.

These are just a few examples of the hyperparameters that can be tuned for the Random Forest algorithm. The optimal values for these hyperparameters depend on the specific dataset and problem at hand. Hyperparameter tuning techniques, such as grid search, random search, or more advanced optimization algorithms like Bayesian optimization, can be used to find the best combination of hyperparameters for a given problem.

Here's a step-by-step overview of how the Random Forest algorithm works:

1. Data Sampling: Random subsets of the training data are selected through a process called bootstrapping. Each subset, also known as a bootstrap sample, is created by randomly selecting data points from the original training set with replacement.

2. Building Decision Trees: For each bootstrap sample, a decision tree is constructed. Decision trees are created by recursively splitting the data based on selected features, aiming to maximize the information gain or decrease in impurity at each split.

3. Random Feature Selection: At each node of the decision tree, only a random subset of features is considered for determining the best split. This randomness helps to reduce overfitting and increases the diversity of the individual decision trees.

4. Ensemble Creation: The individual decision trees created from different bootstrap samples and feature subsets are combined to form the Random Forest ensemble. The final prediction of the Random Forest is obtained by averaging or voting (in the case of classification) the predictions of all the trees.

The methodology for hyperparameter optimization techniques in machine learning typically involves the following steps:

- Define the Hyperparameters: Identify the hyperparameters that need to be optimized for the given machine learning model. These can include learning rate, regularization strength, number of layers, activation functions, and other parameters specific to the chosen algorithm or model architecture.

- Define the Hyperparameter Space: Determine the range or set of values that each hyperparameter can take. This defines the search space within which the optimization will be performed. It is essential to consider the possible range of values based on prior knowledge, domain expertise, and experimentation.

- Select an Optimization Technique: Choose an appropriate hyperparameter optimization technique based on the available resources, complexity of the problem, and desired trade-offs. Common techniques include grid search, random search, Bayesian optimization, evolutionary algorithms, and metaheuristic optimization algorithms.

- Split the Data: Split the available data into training and validation sets. The training set will be used to train the model, while the validation set will be used to evaluate the model's performance with different hyperparameter configurations.

- Define an Evaluation Metric: Select an evaluation metric that reflects the performance and goals of the machine learning task. This can be accuracy, precision, recall, F1 score, or any other relevant metric depending on the problem domain.

- Perform Hyperparameter Optimization: Apply the chosen optimization technique to explore the hyperparameter space and find the optimal configuration. This typically involves iteratively evaluating the model's performance with different hyperparameter values and updating the search strategy accordingly.

- Evaluate Model Performance: After each evaluation, assess the model's performance on the validation set using the chosen evaluation metric. Keep track of the performance for each hyperparameter configuration to identify the best-performing configuration.

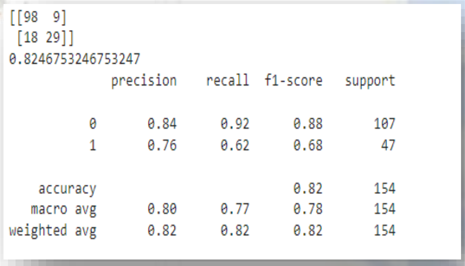
- Refine the Search: Based on the results obtained, refine the search strategy to focus on promising regions of the hyperparameter space. This can involve narrowing down the range of values, adjusting the search algorithm, or employing adaptive search techniques.

- Test the Final Model: Once the optimal hyperparameters are identified, retrain the model using the entire training dataset and the selected hyperparameter configuration. Evaluate the final model's performance on an independent test set to obtain an unbiased assessment of its predictive capability.

- Repeat and Iterate: Hyperparameter optimization is an iterative process. It may require multiple rounds of experimentation, refinement, and validation to fine-tune the model's performance and achieve the desired outcome.

Result and Analysis:
We analyze all the hyperparameter optimization techniques using the Random Forest algorithm on the Diabetes dataset.

Manual Hyperparameter tuning

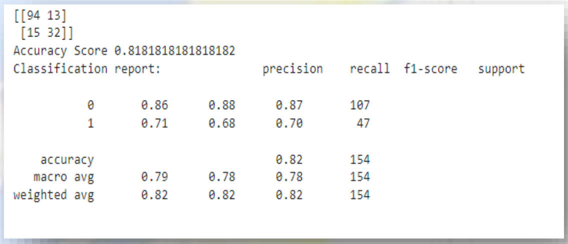


```
[[98  9]
 [18 29]]
0.8246753246753247
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.62      0.68        47

    accuracy                           0.82       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.82      0.82      0.82       154
```

Grid Hyperparameter tuning

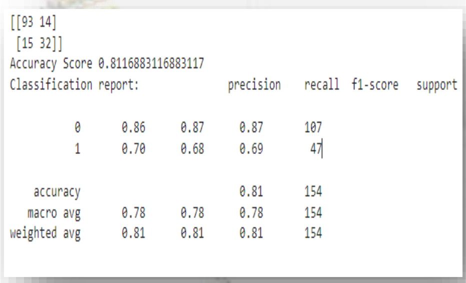


```
[[94 13]
 [15 32]]
Accuracy Score 0.8181818181818182
Classification report:              precision    recall  f1-score   support

           0       0.86      0.88      0.87       107
           1       0.71      0.68      0.70        47

    accuracy                           0.82       154
   macro avg       0.79      0.78      0.78       154
weighted avg       0.82      0.82      0.82       154
```

Random Search Hyperparameter tuning

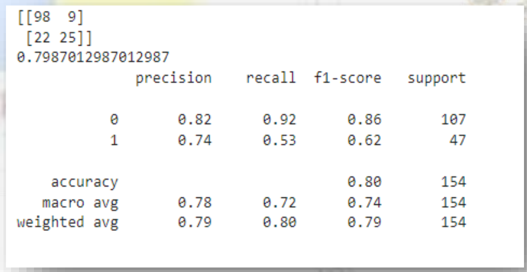


```
[[93 14]
 [15 32]]
Accuracy Score 0.8116883116883117
Classification report:              precision    recall  f1-score   support

           0       0.86      0.87      0.87       107
           1       0.70      0.68      0.69        47

    accuracy                           0.81       154
   macro avg       0.78      0.78      0.78       154
weighted avg       0.81      0.81      0.81       154
```

Bayesian Optimization Hyperparameter tuning

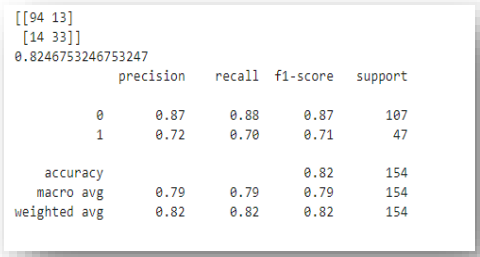


```
[[98  9]
 [22 25]]
0.7987012987012987
              precision    recall  f1-score   support

           0       0.82      0.92      0.86       107
           1       0.74      0.53      0.62        47

    accuracy                           0.80       154
   macro avg       0.78      0.72      0.74       154
weighted avg       0.79      0.80      0.79       154
```

Optuna



```
[[94 13]
 [14 33]]
0.8246753246753247
              precision    recall  f1-score   support

           0       0.87      0.88      0.87       107
           1       0.72      0.70      0.71        47

    accuracy                           0.82       154
   macro avg       0.79      0.79      0.79       154
weighted avg       0.82      0.82      0.82       154
```

IV. Conclusions

Hyperparameter optimization techniques provide a systematic and efficient way to search for the best hyperparameter values, ultimately enhancing the performance and effectiveness of machine learning models. Hyperparameter optimization techniques are that they play a crucial role in improving the performance and generalization capabilities of machine learning models. By fine-tuning the hyperparameters, we can find the optimal configuration that maximizes the model's accuracy and minimizes overfitting.

In this paper we study various techniques, such as grid search, random search, Bayesian optimization, and Genetic algorithm, Optuna. Our results show that the Random forest Classifier offers the best accuracy after hyperparameter tuning, with the highest score of 82.42 obtained when using Optuna. It is important to note that hyperparameter optimization is not a one-size-fits-all approach. The optimal hyperparameter values can vary depending on the dataset, the specific machine learning algorithm used, and the nature of the problem being addressed. Therefore, it is essential to carefully select and tune the hyperparameters for each specific task to achieve the best possible results.

V. References

1. R. E. Shawi, M. Maher, S. Sakr, Automated machine learning: State-of-the-art and open challenges, arXiv preprint arXiv:1906.02287, (2019).http://arxiv.org/abs/1906.02287.
2. M. Kuhn and K. Johnson, Applied Predictive Modeling, Springer (2013)ISBN: 9781461468493.
3. 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)120.https://doi.org/10.1147/JRD.2017.2709578.
4. F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., Automatic Machine Learning: Methods, Systems, Challenges, Springer (2019) ISBN:9783030053185.
5. 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.
6. S. Abreu, Automated Architecture Design for Deep Neural Networks, arXiv preprint arXiv:1908.10714, (2019).http://arxiv.org/abs/1908.10714.
7. O. S. Steinholtz, A Comparative Study of Black-box Optimization Algorithms for Tuning of Hyper-parameters in Deep Neural Networks, M.S.thesis, Dept. Elect. Eng., Lule˚a Univ. Technol., (2018)
8. Sipper, M. (2022). High Per Parameter: A Large-Scale Study of Hyperparameter Tuning for Machine Learning Algorithms Algorithms, 15(9), 315. https://doi.org/10.3390/a15090315.
9. Yasser A. Ali , Emad Mahrous Awwad , Muna Al-Razgan and Ali Maarouf A. Hyperparameter Search for Machine Learning Algorithms for Optimizing the Computational Complexity. Processes 2023, 11, 349. https://doi.org/10.3390/pr11020349
10. https://arxiv.org/abs/2007.15745
11. Probst, P., Boulesteix, A. L., & Bischl, B. (2019). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. Journal of Machine Learning Research, 20(32), 1-32. Retrieved from https://jmlr.org/papers/volume20/18-444/18-444.pdf