HYPER PARAMETER TUNING

PARAMETER VS HYPERPARAMETER TUNING

Parameters

Parameters are the internal variables of a model that are learned from the data during the training process. They define the model's representation of the underlying patterns in the data.

For example:
- In a linear regression model, the parameters are the coefficients of the predictors.
- In a neural network, the parameters are the weights and biases of the nodes.
- In a decision tree, the parameters are the split points and split criteria at each node.

The goal of the training process is to find the optimal values for these parameters, which minimize the discrepancy between the model's predictions and the actual outcomes.

Hyperparameters

In machine learning, hyperparameters are parameters whose values are set before the learning process begins. These parameters are not learned from the data and must be predefined. They help in controlling the learning process and can significantly influence the performance of the model.
- In a neural network, hyperparameters might include the learning rate, the number of layers in the network, or the number of nodes in each layer.
- In a support vector machine, the regularization parameter C or the kernel type can be considered as hyperparameters.
- In a decision tree, the maximum depth of the tree is a hyperparameter.

The best values for hyperparameters often cannot be determined in advance and must be found through trial and error.

Why the word 'hyper'?

The choice of the word is primarily a naming convention to differentiate between the two types of values (internal parameters and guiding parameters) that influence the behaviour of a machine learning model. It's also a nod to the fact that the role they play is a meta one, in the sense that they control the structural aspects of the learning process itself rather than being part of the direct pattern-finding mission of the model.

GridSearchCV

GridSearchCV refers to an algorithm that performs an exhaustive search over a specified grid of hyperparameters, using cross-validation to determine which hyperparameter combination gives the best model performance.

Understanding Grid Search CV with KNN

Suppose we have a dataset with CGPA and IQ as input features and an output column indicating whether a student gets placed. The dataset consists of 1,000 students. Our goal is to train a model that, given the CGPA and IQ of a new student, predicts whether they will be placed.

Now, suppose we decide to apply the K-Nearest Neighbors (KNN) algorithm to train our model. However, KNN has several hyperparameters that need to be set, such as:

- n_neighbors: The number of nearest neighbors to consider.
- metric: The distance metric used for similarity calculations.
- algorithm: The method used to compute nearest neighbors (e.g., brute force, KD-tree, Ball-tree).

As data scientists, we often do not know the optimal values for these hyperparameters.

To determine the best hyperparameter values, we use Grid Search Cross-Validation (GridSearchCV). This technique systematically evaluates different combinations of hyperparameter values to find the best-performing set.
Let's assume we are tuning only two hyperparameters: n_neighbors and metric.
Suppose:

- Possible values for n_neighbors : {3, 5, 10}
- Possible values for metric : {L1, L2}

We create a grid of all possible hyperparameter combinations:

Metric ↓ n_neighbors →	3	5	10
L1	L1,3	L1,5	L1,10
L2	L2,3	L2,5	L2,10

These six points represent all possible hyperparameter combinations. If we had more hyperparameters, the grid would expand into 3D, 4D, or higher dimensions.

Training Models Using Grid Search

Each point in the grid represents a possible hyperparameter setting. GridSearchCV sequentially trains models for each point, such as:

- Model 1 (m1): knn(n_neighbors=3, metric=L1)
- Model 4 (m4): knn(n_neighbors=3, metric=L2)

Similarly, all six models are trained. After training, we evaluate their performance and select the best-performing model. The hyperparameters of this model are then used for final training.

Cross-Validation in Grid Search CV

Cross-validation (CV) ensures that our model is evaluated on different subsets of the data. Instead of training on the entire dataset at once, we split the dataset into k folds and train multiple models.
For example, if we apply k-fold cross-validation with k=5, then:

- Each model is trained on 5 different training-validation splits.
- Since we have 6 hyperparameter settings, the total models trained would be: 6×5=30 models

By using Grid Search CV with Cross-Validation, we systematically explore different hyperparameter settings while ensuring model performance is evaluated robustly.

Advantages and Disadvantages of Grid Search CV

Advantages

1. Systematic and Exhaustive Search
   - Evaluates all possible combinations of hyperparameters, ensuring the best configuration is found.
2. Simple and Easy to Implement
   - Straightforward to apply using GridSearchCV in libraries like Scikit-Learn.
3. Works Well for Small Parameter Spaces
   - Effective when the number of hyperparameters and possible values is limited.
4. Ensures Reproducibility
   - Always produces the same results for the same dataset and hyperparameter grid.
5. Leverages Cross-Validation for Robust Performance
   - Reduces overfitting by validating models on multiple subsets of data.

Disadvantages

1. Computationally Expensive
    - If there are many hyperparameters or large datasets, training becomes time-consuming and resource-intensive.
2. Scalability Issues
    - The number of models trained grows exponentially with the number of hyperparameters, making it infeasible for high-dimensional tuning.
3. Ignores Intermediate Performance Trends
    - Treats hyperparameters as independent points rather than learning from previous results to optimize the search.
4. Not Suitable for Continuous Hyperparameter Ranges
    - Requires discrete values for parameters, missing potentially better settings between tested values.

Randomized Search CV

One of the major challenges with Grid Search CV is that it becomes computationally expensive when the number of hyperparameters increases. Some machine learning algorithms, such as XGBoost and Gradient Boosting, have a large number of hyperparameters. If we use Grid Search CV, it would require evaluating thousands of models, leading to excessive computation time, especially when working with large datasets.
To solve this problem, Randomized Search CV is used. Instead of testing every possible combination of hyperparameters, it randomly samples a fixed number of possible parameter settings and evaluates the models trained with these settings.

How Does Randomized Search CV Work?

1. Define the Parameter Space
    - Similar to Grid Search, we first specify a range of possible values for each hyperparameter. However, instead of evaluating all possible combinations, we only sample a fixed number of random combinations.
2. Randomly Select Hyperparameter Combinations
    - If there are 108 possible combinations in total, we can randomly pick 10, 20, or any other number of parameter settings instead of evaluating all 108.
3. Train and Evaluate Models
    - The selected parameter combinations are used to train multiple models, and cross-validation is applied to assess their performance.
4. Choose the Best Performing Model

    - Once all sampled models are evaluated, the best-performing model is selected based on the validation scores.

Example

Suppose we are tuning a KNN model and we have two hyperparameters:

- n_neighbors: Possible values → [3, 5, 10, 15, 20]
- metric: Possible values → ['euclidean', 'manhattan', 'chebyshev']

Grid Search CV would test 5 × 3 = 15 combinations.
Randomized Search CV, if set to try 6 random combinations, would only evaluate 6 out of these 15.
Example of 6 randomly chosen parameter combinations:

1. n_neighbors = 3, metric = 'euclidean'
2. n_neighbors = 10, metric = 'chebyshev'
3. n_neighbors = 5, metric = 'manhattan'
4. n_neighbors = 20, metric = 'euclidean'
5. n_neighbors = 15, metric = 'chebyshev'
6. n_neighbors = 3, metric = 'manhattan'

Instead of testing all 15, Randomized Search CV efficiently samples a subset and finds a good-performing model faster.

Advantages of Randomized Search CV

1. Computationally Efficient
    - It does not test all combinations, reducing computational cost significantly.
2. Works Well with High-Dimensional Hyperparameter Spaces
    - When many hyperparameters exist, Grid Search becomes impractical, but Randomized Search remains efficient.
3. Can Find Good Parameters Faster
    - It often finds near-optimal parameters in a fraction of the time compared to Grid Search.
4. Flexible in Terms of Resource Allocation
    - We can control how many parameter settings to test, adjusting based on available computing power.

Disadvantages of Randomized Search CV

1. Not Guaranteed to Find the Best Parameters
    - Since it does not test all combinations, it might miss the absolute best configuration.
2. Performance Depends on the Number of Random Samples

    - If too few random configurations are tested, it might not yield a good-performing model.
3. Less Systematic Than Grid Search
    - Grid Search ensures all possible hyperparameter values are tested systematically, whereas Randomized Search does not.

<u>Grid Search vs. Randomized Search</u>

Feature	**Grid Search CV**	**Randomized Search CV**
Computational Cost	High (Tests all combinations)	Lower (Tests a subset)
Best Parameter Guarantee	Yes	No (May miss the best one)
Use Case	Small hyperparameter space	Large hyperparameter space
Flexibility	Fixed, exhaustive search	User-defined number of random trials
Efficiency	Slow for large spaces	Faster, especially for complex models