Hyper-Parameter Tuning

Hyper Parameter 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.

Examples:

- 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 `CC` 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.

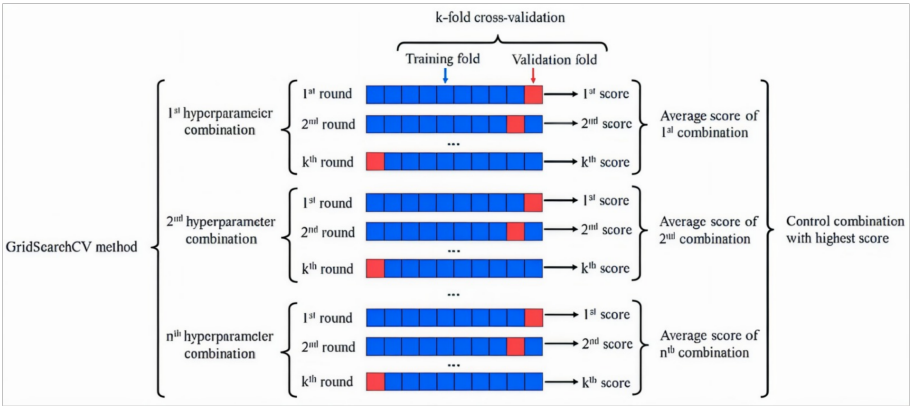
Types of hyperparameter tuning Methods :

1. GridSearch CV
2. Randomized Search CV

Advanced techniques ... (etc.)

Grid Search CV

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.



All the different combinations of hyperparameter are tested and CV is used to get the best (average) accuracy of using that hyperparameter combination.

Refer to the colab file ....

Pros of GridSearchCV

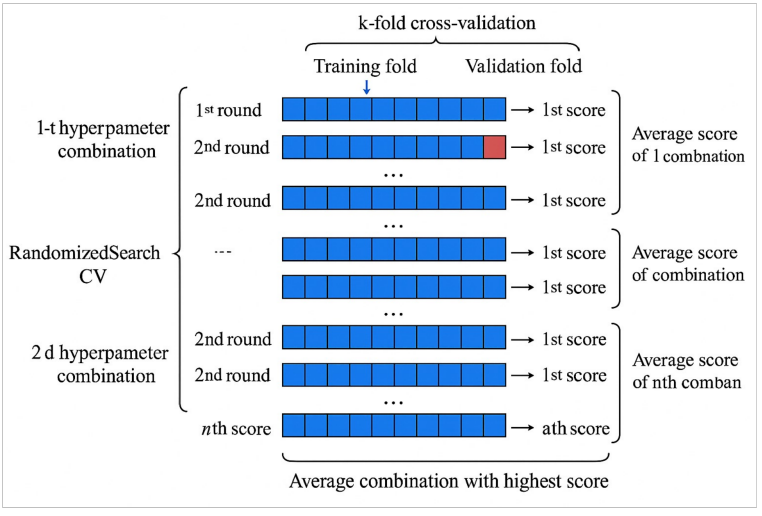
1. **Exhaustive search** – Tests all combinations of specified parameters.
2. **Guaranteed best result (within the grid)** – Finds the optimal hyperparameter set from your defined options.
3. **Deterministic** – Same input always gives the same result (no randomness).
4. **Simple to understand and implement** – Great for beginners and clear documentation.
5. **Works well for small parameter spaces** – Especially when the number of parameters and choices is limited.

Cons of GridSearchCV

1. **Very slow for large grids** – Becomes computationally expensive as the number of parameters or values increases.
2. **Inefficient search** – Tries all combinations, even if many are irrelevant or similar.
3. **Curse of dimensionality** – Time and resources scale poorly with more hyperparameters.
4. **May overfit to validation folds** – Especially if the grid is too fine-grained or data is limited.
5. **Limited flexibility** – Doesn't support probabilistic distributions for parameter values.

Randomized Search CV

Randomized Search CV is a hyperparameter tuning technique in machine learning that searches random combinations of parameters from a specified grid, rather than trying out every single possible combination like GridSearch CV



Pros of RandomizedSearchCV

1. **Faster than GridSearchCV** – Samples a limited number of combinations instead of trying all.
2. **Efficient for large search spaces** – Handles many hyperparameters and wide value ranges well.
3. **Flexible with distributions** – Allows use of probability distributions for parameter sampling.
4. **Good performance with fewer iterations** – Often finds a near-optimal solution faster.
5. **Early stopping possible** – You can limit the number of iterations with n_iter to save compute time.
6. **Avoids combinatorial explosion** – Especially helpful when parameters have many possible values.

Cons of RandomizedSearchCV

1. **No guarantee of best combination** – It might miss the global optimum due to random sampling.
2. **Requires careful distribution setup** – Choosing the right value ranges or distributions needs domain knowledge.
3. **Still computationally expensive** – Not ideal for very complex models or huge datasets without constraints.
4. **Results can vary** – Due to randomness, different runs may yield different "best" parameters (unless random_state is fixed).
5. **Less interpretability in search** – You may not fully understand which values were skipped without deeper inspection.