# Introduction to Hyperparameter Tuning

Data Science is made of mainly two parts. Data analytics and machine learning modeling. Although Data Science has a much wider scope, the above-mentioned components are core elements for any Data Science project. Let me quickly go through the difference between data analytics and machine learning.

For any Data Science project, having historical data is a key essence. This data might not be in the format that makes sense, or we might need to process the data to get insights from the same. To achieve insights, the below process is important.

1. **Data Analytics** – Historical data has a lot to say, so to hear what it has in store for us, we need to analyze it thoroughly. We can get insights into how variables are related to each other, which variable adds what value to the data, do we have any missing value, do we have any extreme (outlier) values, etc. Data description, data pre-processing, data munching, data cleaning, and exploratory data analysis all come under one umbrella, i.e**., Data Analytics**.
2. **Machine Learning**– Post doing data analytics, these insights should be used in the most sought-after way to predict the future values. How to do that? To answer this, we have machine learning models. When a machine learns on its own based on data patterns from historical data, we get an output which is known as a machine learning model. In a broad category, machine learning models are classified into two categories, **Classification**, and **Regression**.

There is a list of different machine learning models. They all are different in some way or the other, but what makes them different is nothing but input parameters for the model. These input parameters are named as **Hyperparameters.** These hyperparameters will define the architecture of the model, and the best part about these is that you get a choice to select these for your model. Of course, you must select from a specific list of hyperparameters for a given model as it varies from model to model.

Often, we are not aware of optimal values for hyperparameters which would generate the best model output. So, what we tell the model is to explore and select the optimal model architecture automatically. This selection procedure for hyperparameter is known as **Hyperparameter Tuning.**

Now, the question arises why we need this? How would this help in my model? What will happen if we skip this step? The list is endless. So, here we would discuss what questions this hyperparameter tuning will answer for us, and then all the above questions will automatically get answered.

- What should be the value for the maximum depth of the Decision Tree?
- How many trees should I select in a Random Forest model?
- Should use a single layer or multiple layer Neural Network, if multiple layers then how many layers should be there?
- How many neurons should I include in the Neural Network?
- What should be the minimum sample split value for Decision Tree?
- What value should I select for the minimum sample leaf for my Decision Tree?
- How many iterations should I select for Neural Network?

- What should be the value of the learning rate for gradient descent?
- Which solver method is best suited for my Neural Network?
- What is the K in K-nearest Neighbors?
- What should be the value for C and sigma in Support Vector Machine?

Above mentioned are just a few questions which could be answered by hyperparameter tuning. Each model has its own sets of parameters that need to be tuned to get optimal output. For every model, our goal is to minimize the error or say to have predictions as close as possible to actual values. This is one of the cores or say the major objective of hyperparameter tuning.

This can be particularly important when comparing how different machine learning models are performing on a dataset. Would it be justified to compare a Random Forest model with hyperparameters against an SVM model which is not optimized in terms of hyperparameter?

In this article, I would be explaining following approaches to Hyperparameter tuning:

- Manual Search
- Random Search
- Grid Search

## Manual Search

While using manual search, we select some hyperparameters for a model based on our gut feeling and experience. Based on these parameters, the

model is trained, and model performance measures are checked. This process is repeated with another set of values for the same hyperparameters until optimal accuracy is received, or the model has attained optimal error.

This might not be of much help as human judgment is biased, and here human experience is playing a significant role.

## Random Search

Though manual search is an iterative process, each time a specific value is given for hyperparameters to acquire optimal model performance measures, it is an exhaustive approach. Instead of doing multiple rounds of this process, it would be better to give multiple values for all the hyperparameters in one go to the model and let the model decide which one best suits. Those who are aware of hyperparameter tuning might say that I am talking about grid search, but no, this is slightly different. Out of the values mentioned, the model randomly makes combinations of its own and tries to fit the dataset and test the accuracy. Here, chances are there to miss on a few combinations which could have been optimal ones. Although, random search consumes quite less amount of time and most of the time it gives optimal solutions as well. So, in that case, it is a win-win situation.

## Grid Search

As mentioned above, in a random search, grid search also uses the same methodology but with a difference. In this method, each combination of

hyperparameter value is tried. This makes the process time consuming, or in short, inefficient. Say we have given 20 different hyperparameter values for 4 different hyperparameters. Then there would be 160,000 trials for a model. This sounds a big number and assumes how much time it would consume for a large dataset. Along with this, cross-validation is also added to the grid search, which increases this trial number to 1,600,000 (if 10-fold cross-validation is used).

This method is quite an expensive method in terms of computation power and time, but this is the most efficient method as there is the least possibility of missing out on an optimal solution for a model.

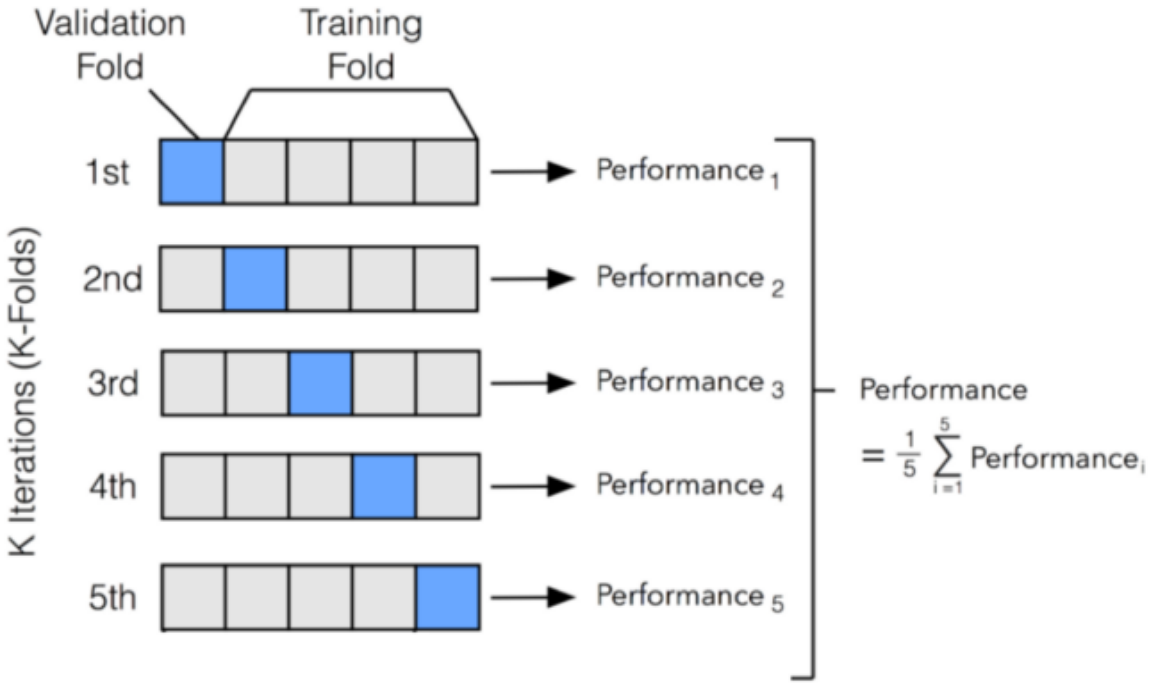
Let's quickly understand the concept of cross-validation also as it is preferred to use it while doing grid search or random search for hyperparameter tuning.

## Cross Validation

While creating any machine learning models, we generally divide the dataset into train sets and test sets. The train set is used to make machines learn the pattern and create a model for future prediction. The test dataset is used to test model performance such that it considers this data as unseen data. When we use cross-validation, even the train set is divided into N partitions to make sure that our model is not overfitting.

K-fold is the most commonly used cross-validation technique. Here training data is divided into K partitions and then the model is iteratively trained on the k-1 partition and test it with the leftover partition. Once having trained K time,

the model, we then average the training results obtained from each iteration to obtain overall model performance results.




Below are hyperparameters listed for few of machine learning models:

- **Decision Tree**
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- **Random Forest**
- n_estimators = number of trees in the forest
- max_features = max number of features considered for splitting a node

- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)
- **Artificial Neural Network**
- Hidden_layer_sizes = Number of neurons
- max_iter = max iterations required
- solver = solver method required for gradient descent
- tol = learning rate for minimization loss

## Conclusion

Here, we explored three methods for hyperparameter tuning. While this is an important step in modeling, it is by no means the only way to improve performance.