1. **What Is Hyperparameter Tuning?**

Hyperparameter tuning is the process of tuning the parameters present as the tuples while we build machine learning models. These parameters are defined by us which can be manipulated according to programmer wish. Machine learning algorithms never learn these parameters. These are tuned so that we could get good performance by the model. Hyperparameter tuning aims to find such parameters where the performance of the model is highest or where the model performance is best and the error rate is least. We define the hyperparameter as shown below for the random forest classifier model. These parameters are tuned randomly and results are checked.

RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mse', max\_depth=None, max\_features='auto', max\_leaf\_nodes=None, max\_samples=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False,  random\_state=None, verbose=0, warm\_start=False)

1. **What Steps To Follow For Hyper Parameter Tuning?**

* Select the type of model we want to use like RandomForestClassifier, regressor or any other model
* Check what are the parameters of the model
* Select the methods for searching the hyperparameter
* Select the cross-validation approach
* Evaluate the model using the score

**Implementation of Regression Model**

First, we will import all the required libraries and the dataset and do the [basic EDA](https://analyticsindiamag.com/tips-for-automating-eda-using-pandas-profiling-sweetviz-and-autoviz-in-python/) to understand the data. Use the below code to do the same

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

df  = pd.read(‘Boston.csv’)

print(df.isnull().sum())

print(df.info())

y = df['medv']

X = df.drop('medv', axis=1)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size= .30, random\_state=1)

from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()

dtr.fir(X\_train,y\_train)

print(dtr.score(X\_test,y\_test))

**Output:**

1. **Implementation of Model using GridSearchCV**

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. We have taken only the four hyperparameters whereas you can define as much as you want. If you increase the number of combinations then time complexity will increase. Use the below code to do the same.

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'bootstrap': [True],

'max\_depth': [5, 10, None],

'max\_features': ['auto', 'log2'],

'n\_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15]

}

Now we will define the type of model we want to build a random forest regression model in this case and initialize the GridSearchCV over this model for the above-defined parameters.

rfr = RandomForestRegressor(random\_state = 1)

g\_search = GridSearchCV(estimator = rfr, param\_grid = param\_grid, cv = 3, n\_jobs = 1, verbose = 0, return\_train\_score=True)

We have defined the estimator to be the random forest regression model param\_grid to all the parameters we wanted to check and cross-validation to 3. We will now train this model bypassing the training data and checking for the score on testing data. Use the below code to do the same.

g\_search.fit(X\_train, y\_train);

print(g\_search.best\_params\_)

**Output:**

We can check the best parameter by using the best\_params\_ function that is shown above.

print(best\_grid.score(X\_test, y\_test))

**Output:**

1. **Implementation of Model using RandomizedSearchCV**

First, we will define the library required for random search followed by defining all the parameters or the combination that we want to test out on the model. Similar to grid search we have taken only the four hyperparameters whereas you can define as much as you want. We have then defined the random grid. Use the below code to do the same.

import numpy as np

from sklearn.model\_selection import RandomizedSearchCV

n\_estimators = [int(x) for x in np.linspace(start = 5 , stop = 15, num = 10)] # returns 10 numbers

max\_features = ['auto', 'log2']

max\_depth = [int(x) for x in np.linspace(5, 10, num = 2)]

max\_depth.append(None)

bootstrap = [True, False]

r\_grid = {'n\_estimators': n\_estimators,

               'max\_features': max\_features,

               'max\_depth': max\_depth,

               'bootstrap': bootstrap}

print(random\_grid)

**Output:**

We will now define the random search passing the rf model with the randomly chosen hyperparameters and then train it. After this, we will check the score. Use the below code to do the same.

rfr\_random = RandomizedSearchCV(estimator=rfr, param\_distributions=r\_grid, n\_iter = 20, scoring='neg\_mean\_absolute\_error', cv = 3, verbose=2, random\_state=42, n\_jobs=-1, return\_train\_score=True)

rfr\_random.fit(X\_train, y\_train);

print(rf\_random.best\_params\_)

**Output:**

print(best\_random.score(X\_test , y\_test))

**Output:**

1. **Comparison of Different Models**

|  |  |
| --- | --- |
| **Models** | **Scores** |
| Regression Model (Without Hyperparameter Search) | 80.34 |
| Regression Model using GridSearchCV | 88.98 |
| Regression Model using RandomizedSearchCV | 90.17 |

**Conclusion**