Analytics using Python

Learning outcomes

1. You will learn Python , a useful language

2. Use programming for problem solving

Great Lakes Institute of Management

A guide to learn python for analytics

P. V. Subramanian

**A workbook on Analytics using Python**

**P. V. SUBRAMANIAN**

**Visiting faculty – Great Lakes Institute of Management**

*Copyright @ 2019 Great Lakes Institute of Management : All contents in this book, including (but not limited to) its contents are proprietary to Great Lakes Institute of Management, Chennai. Any effort at re-using, modifying and circulating both through Digital or Physical means, without the express written permission of the author, will amount to infringement of copyrights, and will be liable for suitable action.*

**Chapter 2. HYPER TUNING PARAMETERS**

Table of Contents

**[1)](#_Toc12996616)****[HYPER TUNING PARAMETERS](#_Toc12996616)** [4](#_Toc12996616)

[1.1. Introduction 4](#_Toc12996617)

[1.2. ComparE HYPERPARAMETERS AND MODEL PARAMETERS 4](#_Toc12996618)

[1.3. DIAGRAM SHOWING HYPERPARAMETERS AND MODEL PARAMETERS 5](#_Toc12996619)

[1.4. Why do we need hyperparameters? 5](#_Toc12996620)

[1.5. CLASSES OF Hyperparameter optimization 6](#_Toc12996621)

[**1.5.1.** **Exhaustive search of the space category** 6](#_Toc12996628)

[**1.5.2.** **Surrogate models** 7](#_Toc12996629)

[**1.5.3.** **Other models** 8](#_Toc12996630)

[1.6. Exhaustive search of the space ALGORITHMS -REGRESSION 8](#_Toc12996631)

[**1.6.1.** **Grid Search** 9](#_Toc12996654)

[**1.6.2.** **Random Search** 21](#_Toc12996655)

[1.7. SURROGATE MODEL ALGORITHMS - REGRESSION 27](#_Toc12996656)

[**1.7.1** **Bayesian Optimization** 27](#_Toc12996701)

[1.8. Exhaustive search of the space ALGORITHMS -CLASSIFICATION 33](#_Toc12996702)

[**1.8.1.** **Grid Search** 38](#_Toc12996712)

[**1.8.2.** **Random Search** 41](#_Toc12996713)

[1.9. SURROGATE MODEL ALGORITHMS - CLASSIFICATION 44](#_Toc12996714)

[**1.9.2** **Bayesian Optimization** 44](#_Toc12996718)

# **HYPER TUNING PARAMETERS**

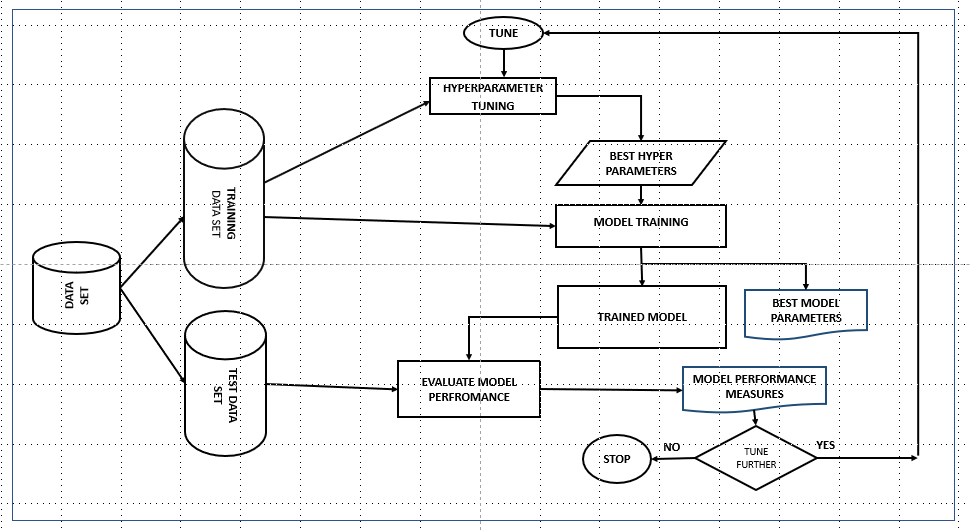
## Introduction

* In machine learning, **model** is the definition of a mathematical formula with a number of parameters that need to be learned from the data done through a process known as training.
* To explain, higher level properties of the model such as number of leaves or depth of a tree in Decision Trees, learning rate in many models including Gradient Boosting and Artificial Neural Network models, we need **hyperparameters** which are fixed before the model training process begins.

## ComparE HYPERPARAMETERS AND MODEL PARAMETERS

|  |  |
| --- | --- |
| **MODEL PARAMETERS** | **HYPER PARAMETERS** |
| * They are the properties of training data that learns on its own during the process of training by the model classifier such as Ridge Regression. Decision Trees, Random Forest, Artificial Neural Network, KNN etc. * Examples of model parameters are:   + - * Coefficients and slope in Ridge regression models       * Split points in decision trees       * Weights and bias in neural network | * They are the properties that govern the training process. * Examples of hyperparameters are:   a. Shrinkage parameter, lambda in Ridge Regression  b. Depth of tree in decision trees  c. Learning rate, Batch size, Number of layers, Activation functions in neural network |
| They are required by the model for making predictions for the new data | They are often specified prior to model building |
| They are usually not set manually | They can be set using heuristics |
| They are usually saved as part of the saved model | They are often tuned for a given modelling problem |

## DIAGRAM SHOWING HYPERPARAMETERS AND MODEL PARAMETERS



## Why do we need hyperparameters?

* Hyperparameters directly control the behaviour of the training algorithm and have a significant impact on the performance of the model is being trained.
* Choosing appropriate hyperparameters plays an important role in the success of the model architecture.

## CLASSES OF Hyperparameter optimization

* The hyperparameter optimization algorithms can be separated into three main categories:

a. Exhaustive search of the space

b. Surrogate models

c. Other models



### **Exhaustive search of the space category**

Grid search and random search belong to this class of hyper parameter tuning.

1. **GRID SEARCH**

* This is the traditional way of performing hyperparameter optimization
* Grid search or parameter sweep tries every combination of each setting of hyper-parameters.
* This approach guarantees to find the best set of values in search space guided by some performance metric.
* If the parameter space includes a real-valued space, set bounds manually and discretize before applying grid search.
* Refer:

<https://scikit-learn.org/stable/modules/grid_search.html>

<https://en.wikipedia.org/wiki/Hyperparameter_optimization>

1. **Random SEARCH**

* Random search replaces the exhaustive enumeration of all combinations by selecting them randomly.
* Random search algorithm randomly samples the search space instead of discretizing it with a Cartesian grid.
* For most data sets only a few of the hyper-parameters really matter. http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf
* The grid search approach spends redundant time exploring the unimportant parameter.
* Since the above models are performed in isolation, it is easy to parallelize this process.
* However, because each experiment was performed in isolation, they do not perform the **informed search** (i.e. choosing the next hyperparameter) to evaluate based on previous results.

### **Surrogate models**

1. **BAYESIAN METHOD**

* Bayesian methods, unlike random or grid search, keep track of past evaluation results and use them to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function: P(score | hyperparameters)
* This model is called surrogate for the objective function.
* Bayesian methods work by finding the next set of hyperparameters to evaluate on the actual objective function by selecting hyperparameters that perform best on the surrogate function.
* Bayesian optimization methods belongs to a class of Sequential model-based optimization (SMBO) algorithms. These methods run trials one after another, each time trying to improve hyperparameters by applying the results of previous iteration (Bayesian reasoning) and updating a probability model (surrogate).
* Bayesian optimization is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric.
* Several common choices for the surrogate model are Gaussian Processes, Random Forest Regressions, and Tree Parzen Estimators (TPE). These methods differ in how they construct the surrogate function. Selection function is the criteria by which the next set of hyperparameters are chosen from the surrogate function. The most common choice is Expected Improvement.
* Bayesian optimization methods are more efficient than random or grid search with better overall performance on the test data and less computation time.
* The probability model (aka surrogate or response surface) is easier to optimize than the actual objective function.
* There are now several libraries that can do SMBO in Python.
* **Spearmint and MOE** use a Gaussian Process for the surrogate, **Hyperopt** uses the Tree-structured Parzen Estimator, and **SMAC** uses a Random Forest regression. These libraries all use the Expected Improvement criterion to select the next hyperparameters from the surrogate model.

### **Other models**

1. **Gradient based optimization**

* This method is applied for specific learning algorithms.
* This method computes the gradient with respect to hyperparameters and then optimize the hyperparameters using gradient descent.
* The first usage of these techniques was focused on neural networks. Since then, these methods have been extended to other models such as support vector machines or logistic regression. *Refer:* [*https://en.wikipedia.org/wiki/Hyperparameter\_optimization*](https://en.wikipedia.org/wiki/Hyperparameter_optimization)

**We shall focus on the thirst three optimization models.**

## Exhaustive search of the space ALGORITHMS -REGRESSION

**We need the following to be defined for these algorithms:**

1. a search space
2. setting bounds for all the hyper parameters
3. adding a little prior knowledge on them (such as setting a non-uniform distribution for the search, example learning rates to be searched on a log distribution).

* In scikit-learn hyper-parameters are passed as arguments to the constructor of the estimator classes.
* It is highly recommended to search the hyper-parameter space for the best cross validation score.
* A search consists of:

a. An estimator (regressor or classifier such as sklearn.svm.SVC())

b. A parameter space (such as ParameterGrid in scikit-learn which gives a grid of parameters with a discrete number of values for each)

c. A method for searching or sampling candidates (such as expon, gamma, uniform and randint in scipy.stats module that provides a random variate sample)

d. A cross-validation scheme; and

e. A score function (such as precision, recall for unbalanced dataset for classification problems and R sqaure for regression problems) to evaluate a parameter setting



*We shall tune the hyper parameter for a decision tree regressor.*

* 1. **DECISION TREE**

DecisionTreeRegressor() from sklearn.tree creates a decision tree regressor object.

1. **max\_depth:** The maximum depth of the tree
2. **max\_features:** The number of features to consider when looking for the best split.
3. **max\_leaf\_nodes**: To grow a tree in the best-first fashion
4. **min\_impurity\_decrease**: The threshold for splitting a node; if the split induces a decrease of the impurity greater than or equal to threshold, node is split
5. **min\_impurity\_split**: The threshold for early stopping the tree growth
6. **min\_samples\_leaf**: The minimum number of samples required to be at a leaf node
7. **min\_samples\_split**: The minimum number of samples required to split an internal node
8. **min\_weight\_fraction\_leaf**: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node.

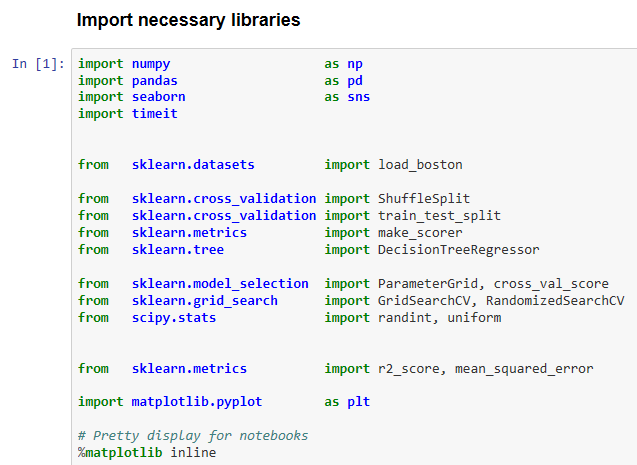
### **Grid Search**

* The grid search provided by GridSearchCV in Scikit-Learn exhaustively generates candidates by cross validation from a grid of parameter values specified with the param\_grid parameter.
* The Boston Housing Dataset contains the prices of houses in suburbs of Boston, Massachusetts. There are 506 observations with 13 predictor variables and 1 target variable. We need to predict the monetary value of the houses. This data set is already available via the scikit-learn package.
* The dataset originates from the UCI Machine Learning Repository. The Boston Housing data was collected in 1978.

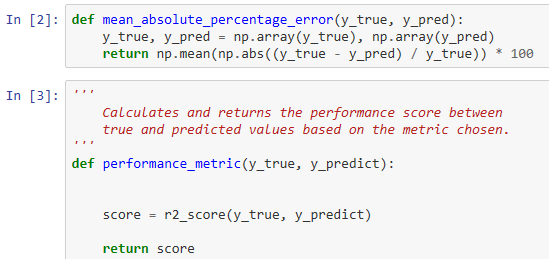
Refer: <https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html>

* We shall use grid search to select the best set of hyperparameters for Decision Tree Regressor to build a Decision Tree model for Regression on this data set.

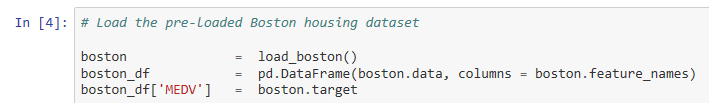
**Import required packages and load data**

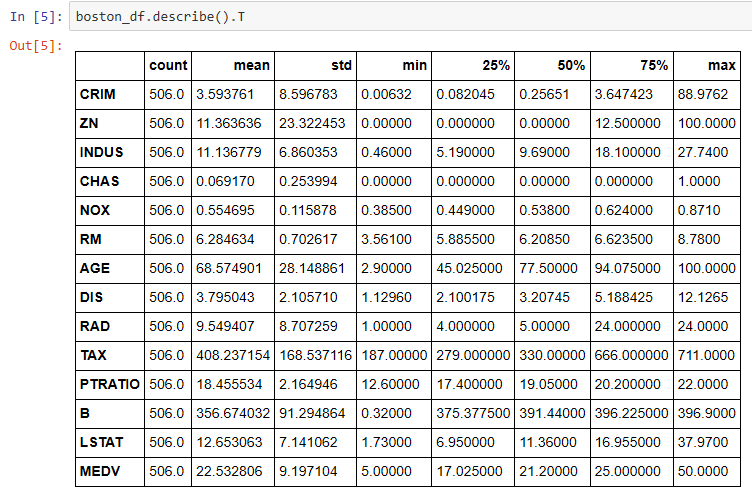
****

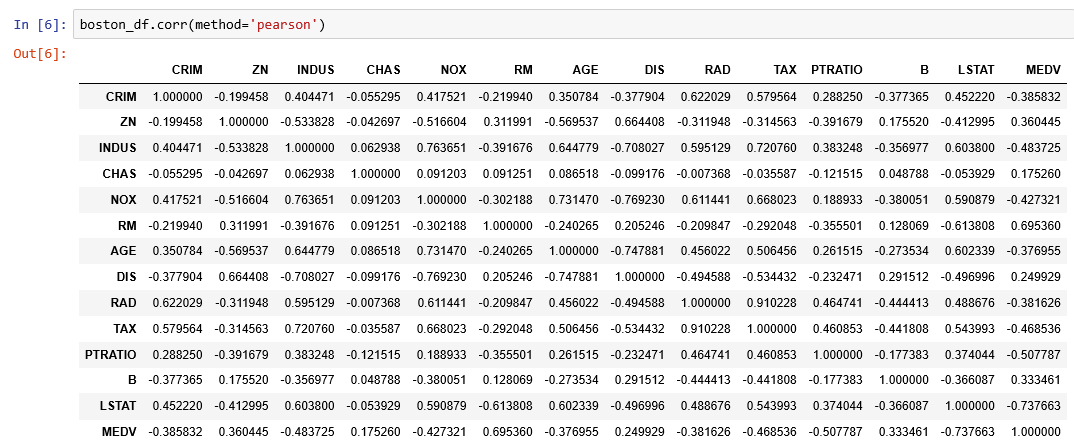
**Define useful functions**

****

**Load the data**

****

**Make a customary investigation into the data **

****

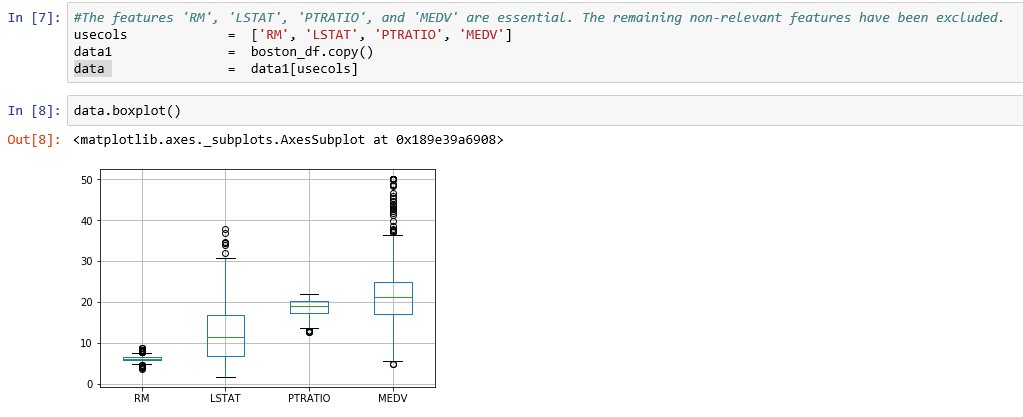
* From the above correlation matrix, we observe that

1. Strength of the relationship between MEDV, response variable and RM, a predictor variable is 0.695360 which implies for a higher RM, higher MEDV. This is because more rooms imply more space and cost assuming all other factors constant.
2. Strength of the relationship between MEDV, response variable and LSTAT, a predictor variable is -0.737663. For a higher LSTAT, lower MEDV.
3. Strength of the relationship between MEDV, response variable and PTRATIO, a predictor variable is -0.507787. For a higher PTRATIO, lower MEDV. This is because lower teacher to student ratio results in less attention dedicated to each student and thus impairing their performance at the school.
4. All the other predictor variables are weakly correlated with MEDV.

* We shall use the following three features from the Boston Housing data set:

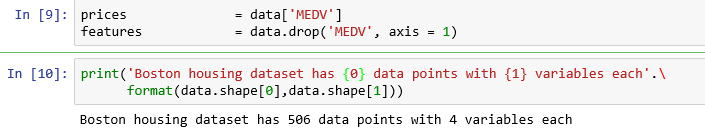
1. RM, which is the average number of rooms among the homes in the neighborhood
2. LSTAT, the percentage of homeowners in the neighborhood considered working poor class
3. PTRATIO, the ratio of students to teachers in primary and secondary schools in the neighborhood

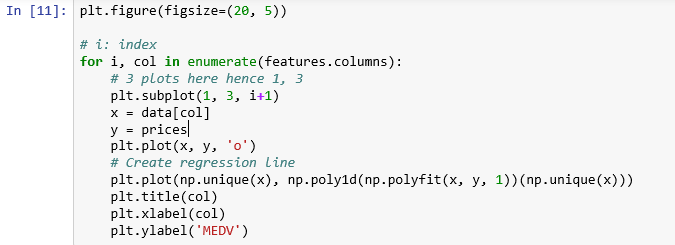
**Check for outliers**

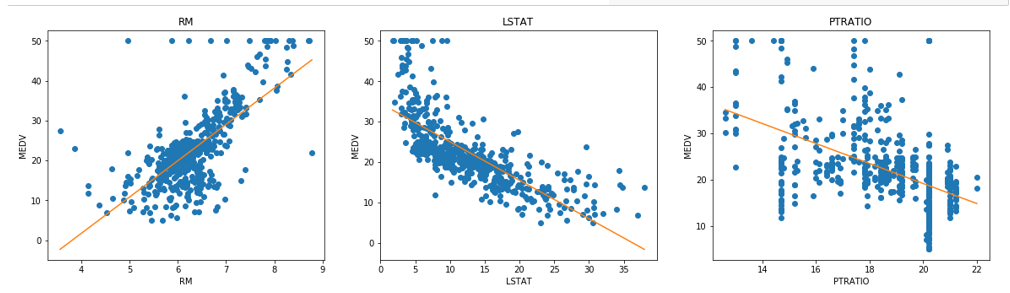
****

**Even though there are outliers for all the variables, looking at the maximum value, it appears that there are no major typos and we shall retain all the data points.**

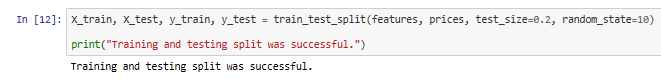
**Create predictor variables data set and response variable data set.**

****

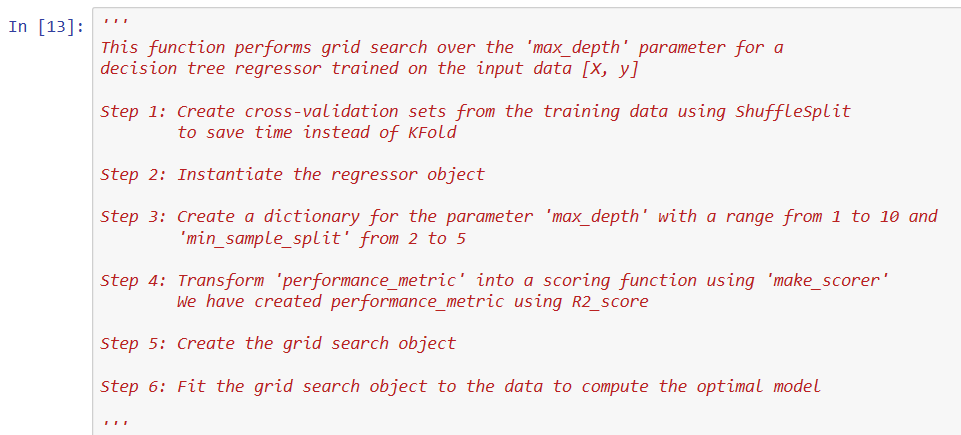
****

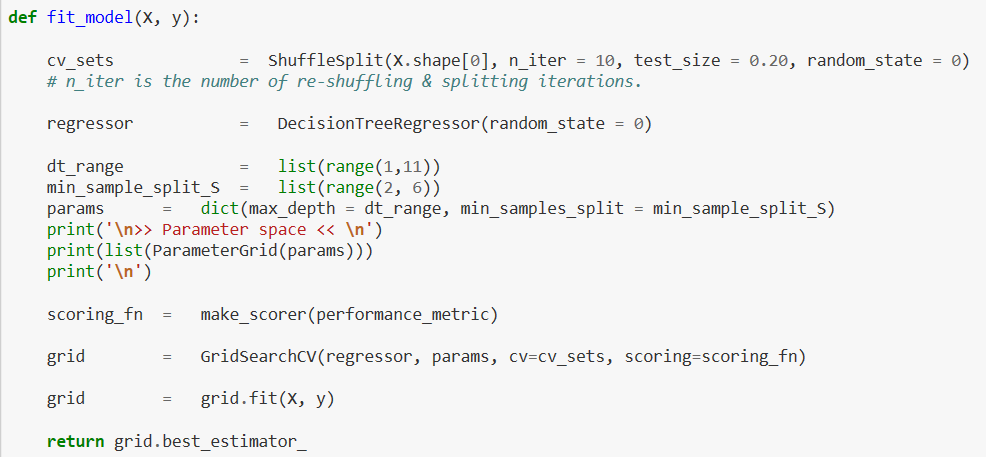
****

**Create Training data and test data**

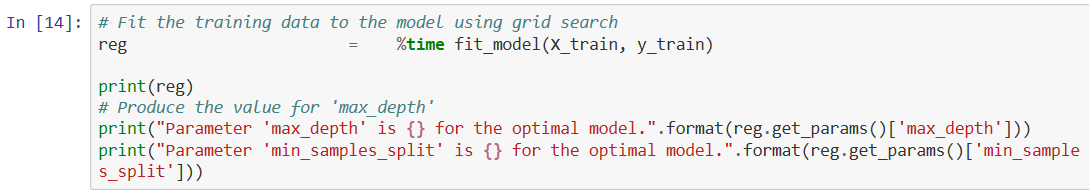
****

**Write a function for performing grid search for DecisionTreeRegressor**

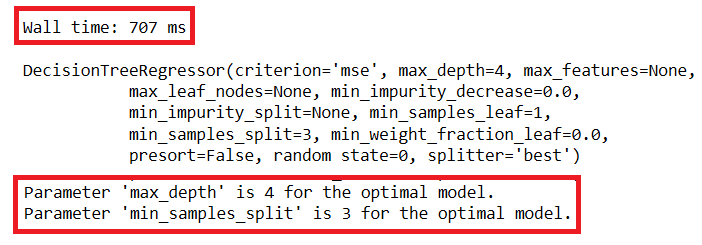
****

****

**Apply the function on the training data.**

****

****

****

**Here, we observe the following:**

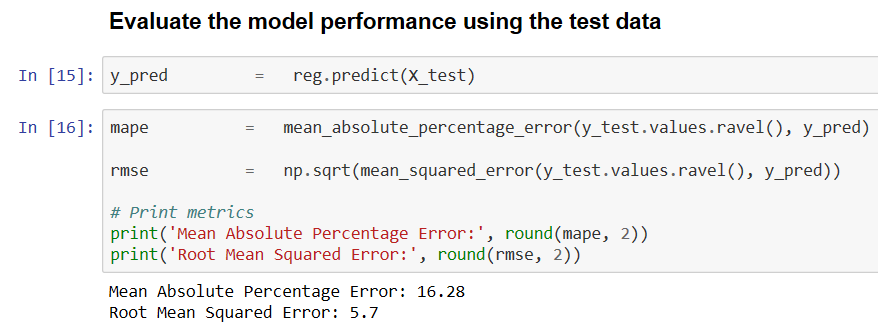
* Our grid search consists of

1. An estimator: sklearn.grid\_search. GridSearchCV
2. Parameter space: 40 pairs of {‘max\_depth’, ‘min\_sample\_split’} values ranging from

{(1,2) to (10, 5)} covering all combinations.

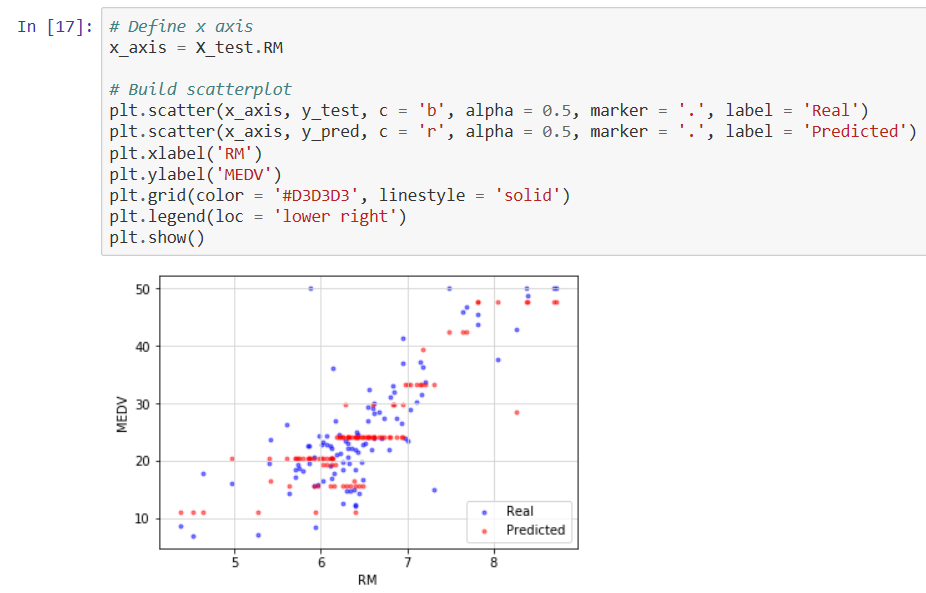
1. A method for searching or sampling candidates: Not Applicable since we need the entire candidates.
2. A cross-validation scheme: ShuffleSplit, a random permutation cross-validator with 10 number of re-shuffling & splitting iterations. This is used instead of KFold
3. A score function: R square as defined in our function, performance\_metric

* Execution time for the grid search is 707 milli-seconds. We need a large data set to compare model performance.
* Best value for the hyperparameter, max\_depth is 4 and min\_sample\_split is 3.

****

**Visual observed values with predicted values using RM, a predictor variable**

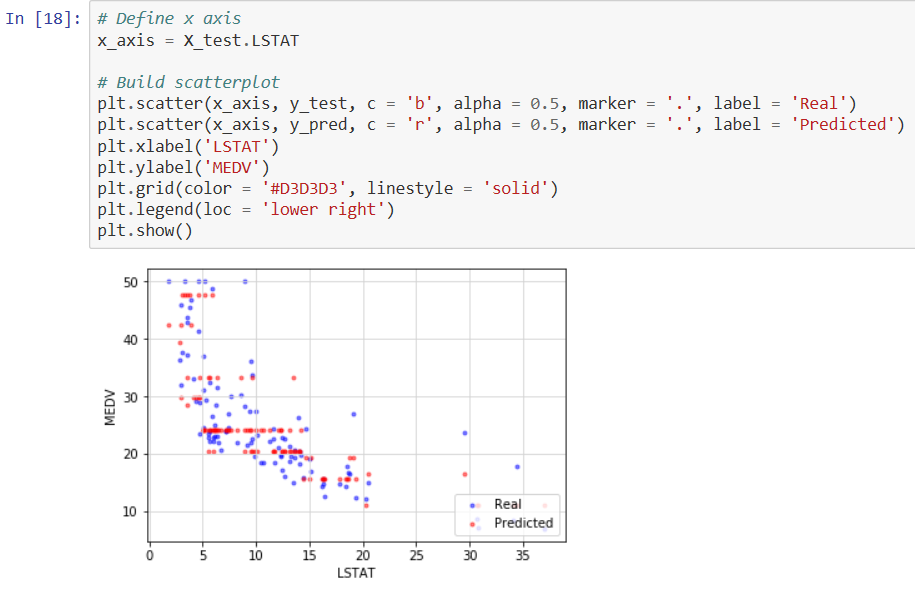
**Correlation coefficient with MEDV is 0.6954**



**You can observe how the model predicts MEDV for various values of RM. Check whether the model predicts MEDV for lower values of RM or higher values of RM.**

**Visual observed values with predicted values using LSTST, a predictor variable**

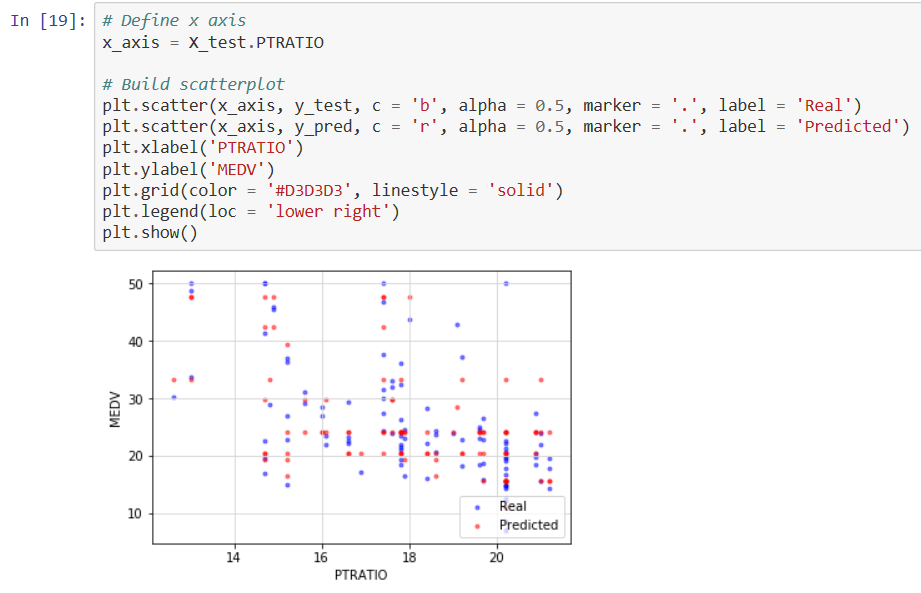
**Correlation coefficient with MEDV is -0.737663**

****

**You can observe how the model predicts MEDV for various values of LSTAT. Check whether the model predicts MEDV for lower values of LSTAT or higher values of LSTAT.**

**Visual observed values with predicted values using PTRATIO, a predictor variable**

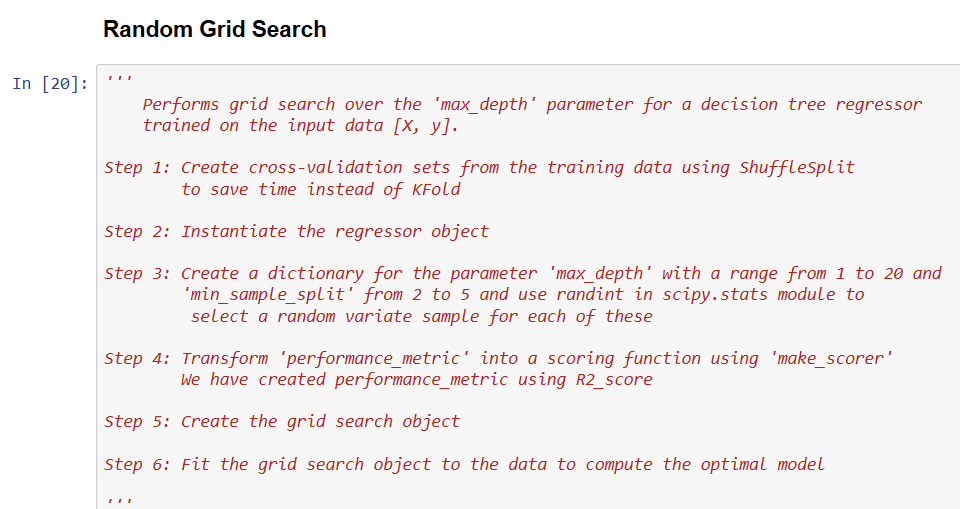
**Correlation coefficient with MEDV is -0.5078**

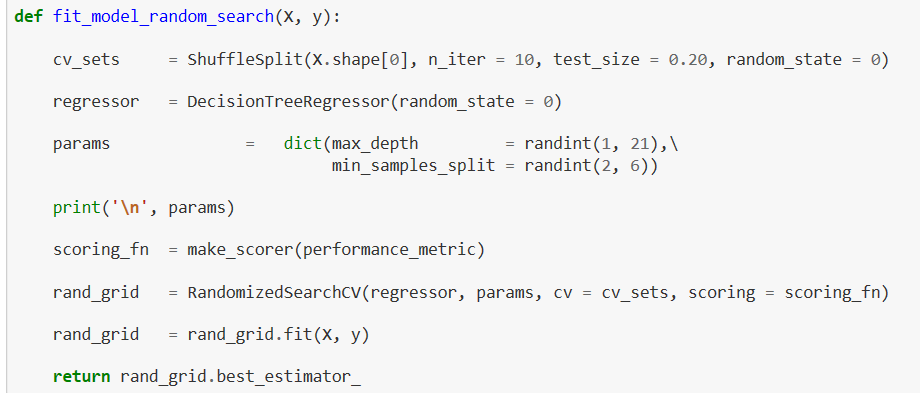
****

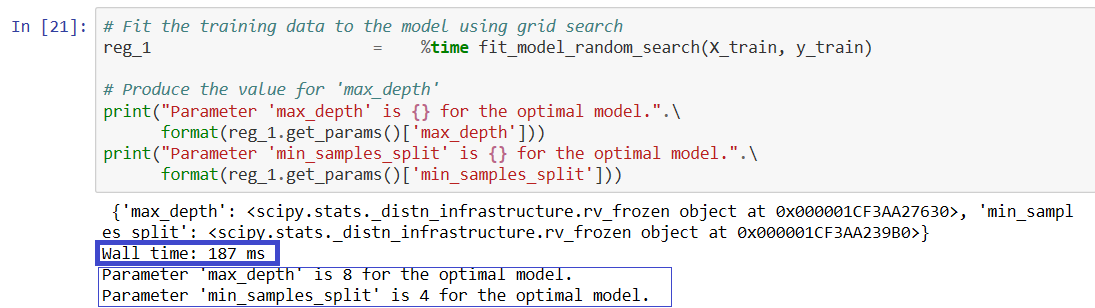
**You can observe how the model predicts MEDV for various values of PTRATIO. Check whether the model predicts MEDV for lower values of PTRATIO or higher values of PTRATIO.**

### **Random Search**

* Using Scikit-Learn’s RandomizedSearchCV method, you can define a grid of hyperparameter ranges, and randomly sample from the grid, performing cross validation with each combination of values.
* We shall use random grid search to select the best set of hyperparameters for Decision Tree Regressor to build a Decision Tree model for Regression on the Boston Housing data set.





****

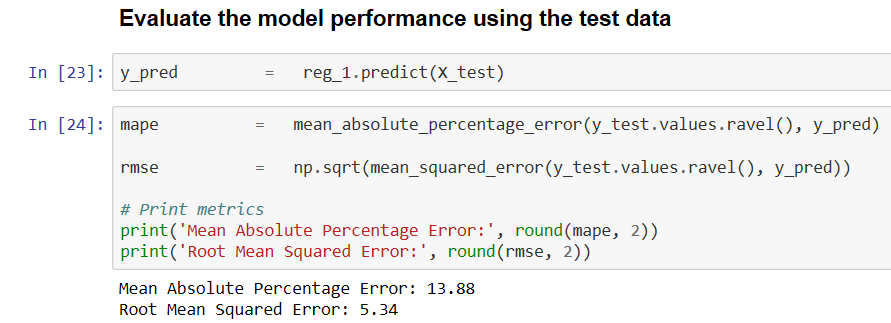
**Observations:**

* Both GridSearchCV and RandomizedSearchCV method of hyperparameter tuning of the DecisionTree Regression model yield the different pair of values for parameters:

GridSearchCV method gives the best hyperparameters as (max\_depth = 4, min\_samples\_split = 3)

RandomizedSearchCV method gives best hyperparameters as (max\_depth = 8, min\_samples\_split = 4)

* The method GridSearchCV took 707 milli seconds for execution while RandomizedSearchCV, method took only 187 milli seconds.

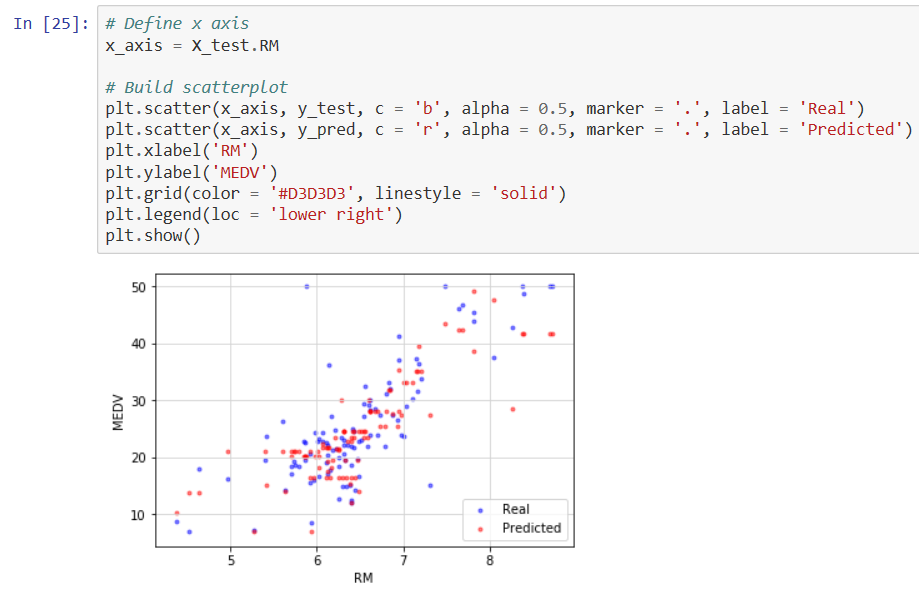


**Observations:**

* Both GridSearchCV and RandomizedSearchCV method of hyperparameter tuning of the DecisionTree Regression model yield the different model performance measures.
* RandomizedSearchCV method gives the MAPE as 13.88 and RMSE as 5.34 while GridSearchCV method gives MAPE as 16.28 and RMSE as 5.7.
* We observe that RandomizedSearchCV method performs better than GridSearchCV method.

**Visual observed values with predicted values using RM, a predictor variable**

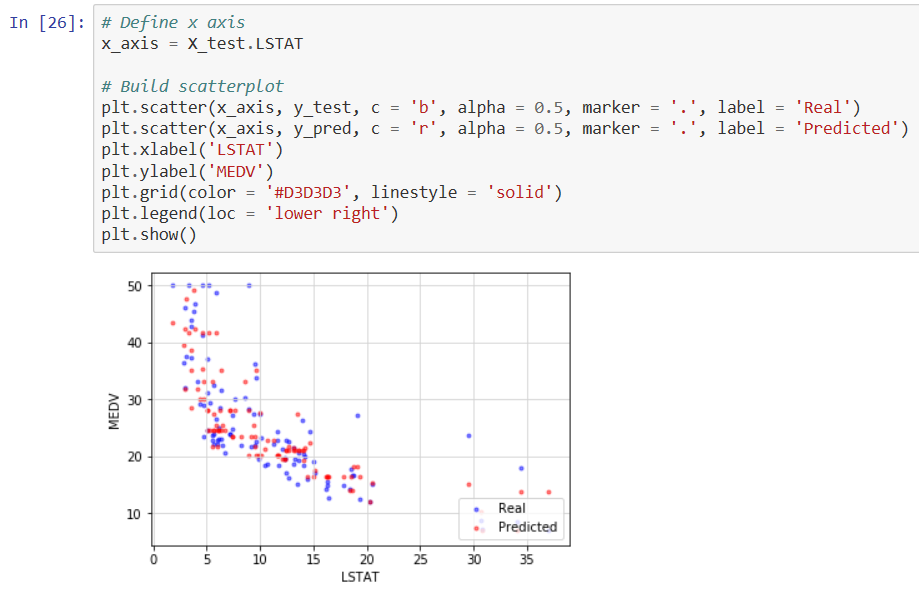
**Correlation coefficient with MEDV is 0.6954.**



**You can observe how the model predicts MEDV for various values of RM. Check whether the model predicts MEDV for lower values of RM or higher values of RM.**

**Visual observed values with predicted values using LSTST, a predictor variable**

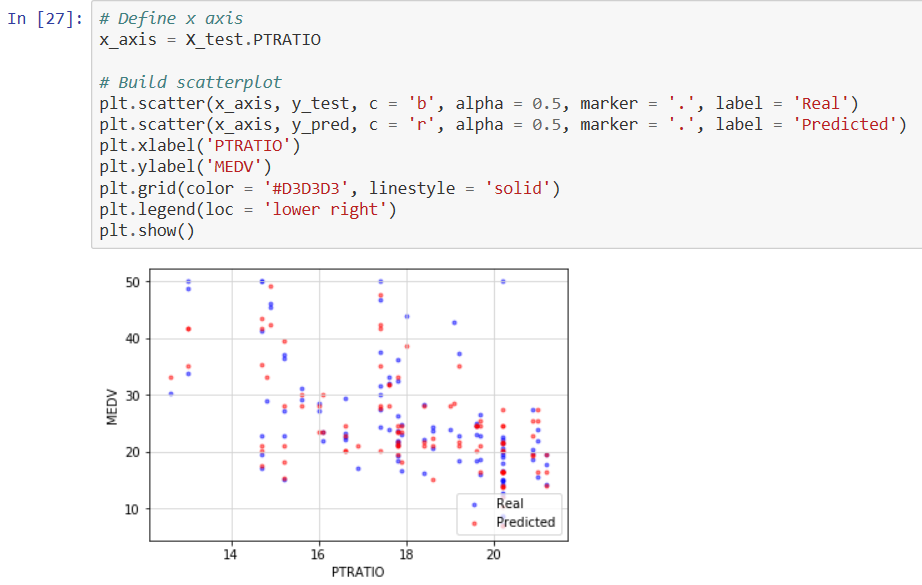
**Correlation coefficient with MEDV is -0.737663**



**You can observe how the model predicts MEDV for various values of LSTAT. Check whether the model predicts MEDV for lower values of LSTAT or higher values of LSTAT.**

**Visual observed values with predicted values using PTRATIO, a predictor variable**

**Correlation coefficient with MEDV is -0.5078**



**You can observe how the model predicts MEDV for various values of PTRATIO. Check whether the model predicts MEDV for lower values of PTRATIO or higher values of PTRATIO.**

*Refer:* [*https://www.ritchieng.com/machine-learning-project-boston-home-prices/*](https://www.ritchieng.com/machine-learning-project-boston-home-prices/)

## SURROGATE MODEL ALGORITHMS - REGRESSION

* There are five elements of model-based hyperparameter optimization:

1. *A domain of hyperparameters over which to search*
2. *An objective function which takes in hyperparameters and outputs a score that we want to minimize (or maximize)*
3. *The surrogate model of the objective function*
4. *A criteria, called a selection function, for evaluating which hyperparameters to choose next from the surrogate model*
5. *A history consisting of (score, hyperparameter) pairs used by the algorithm to update the surrogate model*

*We shall tune the hyper parameter for a decision tree regressor.*

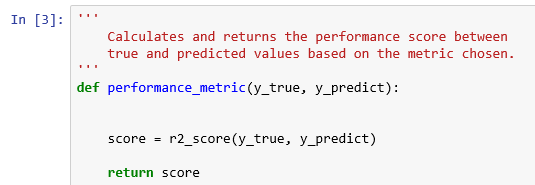


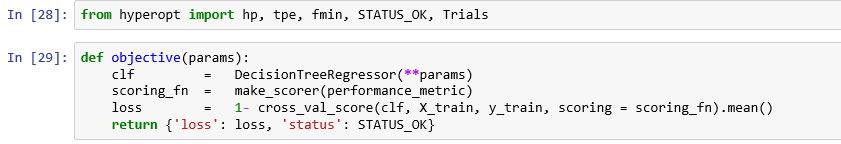
### **Bayesian Optimization**

We shall use Hyperopt (https://hyperopt.github.io/hyperopt/) to put Bayesian optimization into practice.

**Steps in formulating an optimization problem in hyperopt:**

* 1. Create objective function which takes hyper parameter space as input and returns a loss minimize. We are taking loss as 1 – R square which gives the percentage of variation in Y that the model cannot explain. We are making use of three fold cross validation and taking its mean to compute the loss for each iteration.

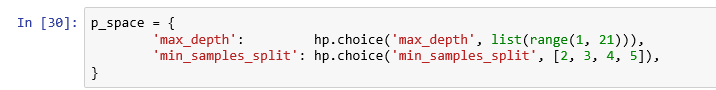




* We need to maximize the R square or minimize 1 – R square.
* Note: We need to define a function that return the negative of that metric since our objective function returns a real value that we want to minimize.
  1. Domain space which contains the range of input values to evaluate.

We are using both max\_depth and min\_samples\_split as hyper parameters to be tuned.

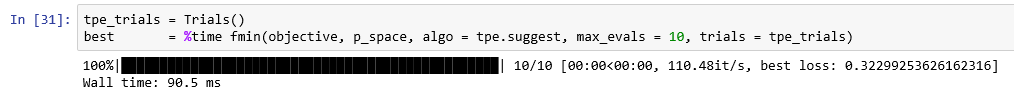
We define the range of values for max\_depth as 1 to 20 and for min\_samples\_split as 2 to 5



* 1. Optimization algorithm is the method used to construct the surrogate function and choose the next values to evaluate. We are using Tree structured Parzan Estimator (TPE)\* model and let hyperopt to configure it by using the suggest method.

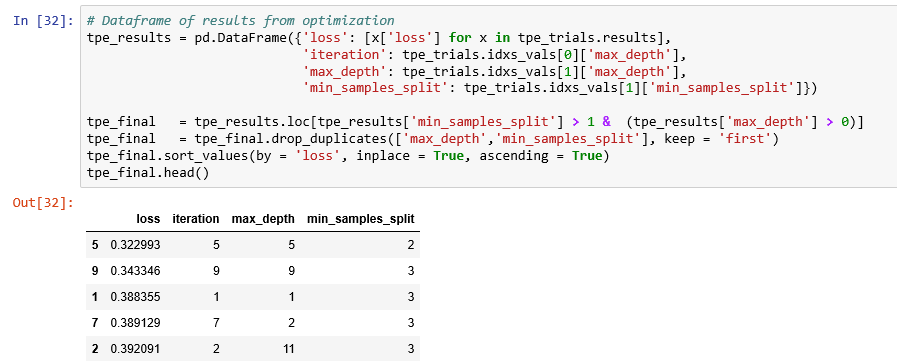
Note: Each iteration TPE collects new observation and at the end of the iteration, the algorithm decides which set of parameters it should try next.

* 1. Results which contains a score value pair that the algorithm uses to build the model



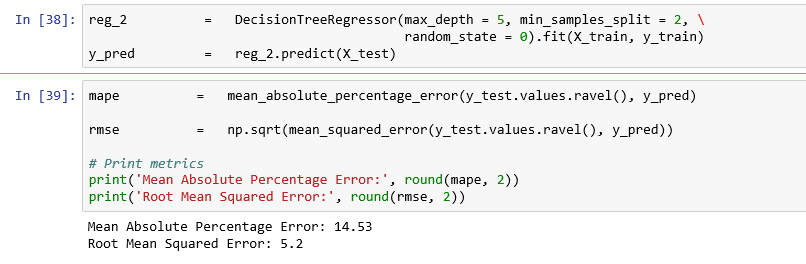
* We have created a Trials object that will record the values and the scores to find out the value - score pair for each trial.
* We have minimized our objective function by using fmin() function that takes four parts above as well as a maximum number of trials (10).
* Hyperopt method took 90.5 milli seconds while the method GridSearchCV took 655 milli seconds for execution and RandomizedSearchCV, method took only 184 milli seconds for the same data set and parameter space.

**Find the optimum hyper parameters**



* Hyperopt method (a Bayesian method) gives the pair of values (max\_depth = 5, min\_samples\_split = 2). We had seen that both GridSearchCV and RandomizedSearchCV method of hyperparameter tuning of the DecisionTree Regression model yield the same pair of values for parameters (max\_depth = 4, min\_samples\_split = 3)

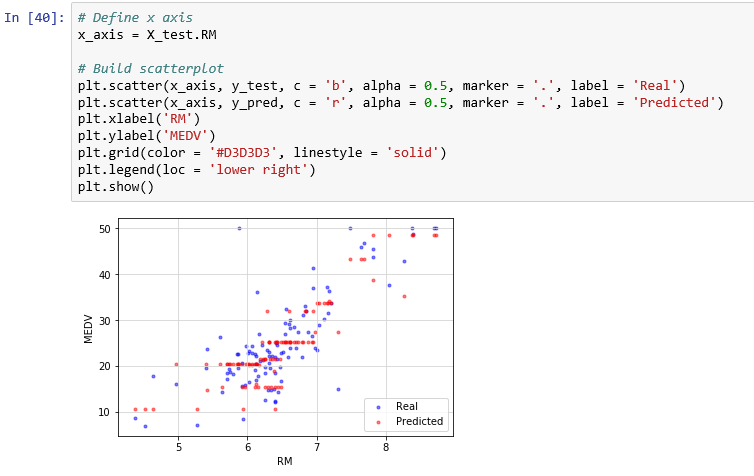
Build the model using these hyper parameters



* RandomizedSearchCV method gives the MAPE as 13.88 and RMSE as 5.34 while GridSearchCV method gives MAPE as 16.28 and RMSE as 5.7. Our hyperopt method (Bayesian method) report performance measures MAPE as 14.53 and RMSE as 5.2.

**Visual observed values with predicted values using RM, a predictor variable**

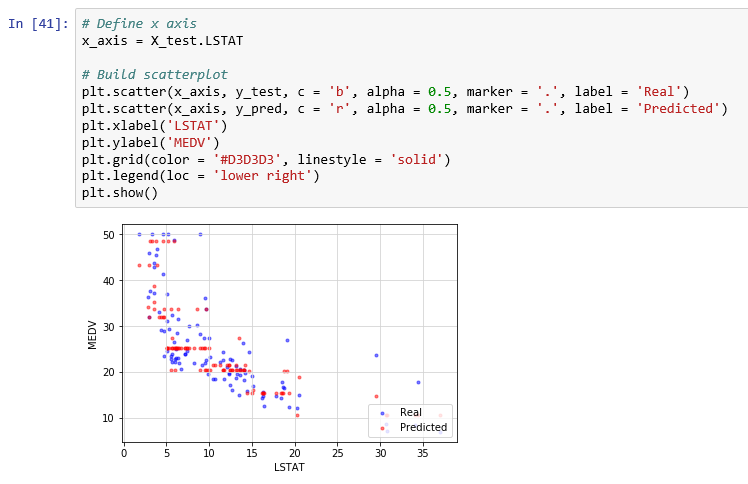
**Correlation coefficient with MEDV is 0.6954**



**You can observe how the model predicts MEDV for various values of RM. Check whether the model predicts MEDV for lower values of RM or higher values of RM.**

**Visual observed values with predicted values using LSTST, a predictor variable**

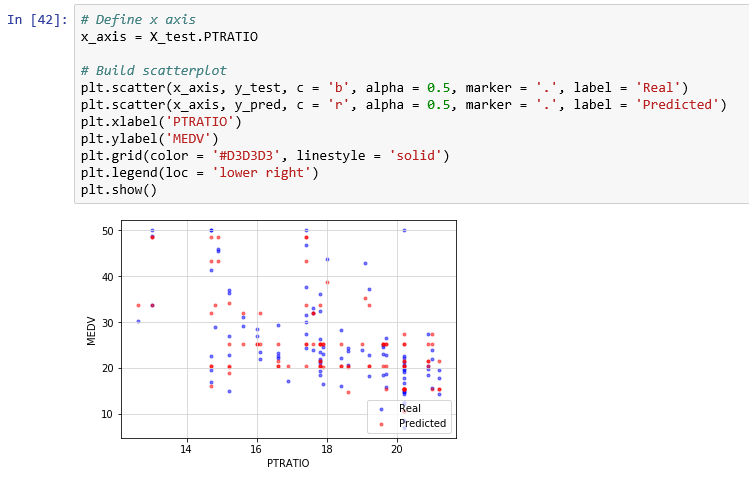
**Correlation coefficient with MEDV is -0.737663**



**You can observe how the model predicts MEDV for various values of LSTAT. Check whether the model predicts MEDV for lower values of LSTAT or higher values of LSTAT.**

**Visual observed values with predicted values using PTRATIO, a predictor variable**

**Correlation coefficient with MEDV is -0.5078**



**You can observe how the model predicts MEDV for various values of PTRATIO. Check whether the model predicts MEDV for lower values of PTRATIO or higher values of PTRATIO.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **RMSE** | **MAPE** | **Execution time** |
| **GridSearchCV** | **5.7** | **16.28** | **655 ms** |
| **RandomizedSearchCV** | **5.34** | **13.88** | **184 ms** |
| **Bayseian - Hyperopt** | **5.2** | **14.53** | **90.5 ms** |

## Exhaustive search of the space ALGORITHMS -CLASSIFICATION

* We shall focus on the random forest algorithm. This creates an ensemble of decision trees which find optimal partitions of the data which result in an accurate classification of the predicted response variable.
* In our example, hyperparameters include

1. Number of decision trees in the forest (n\_estimators)
2. Number of features considered by each tree when splitting a node (max\_features)
3. Maximum depth of a tree (max\_depth)
4. Minimum number of samples to split an internal node (min\_samples\_split)
5. Minimum number of samples required at a leaf node (min\_samples\_leaf)

* A search consists of:

a. An estimator (regressor, sklearn.ensemble.RandomForestClassifier )

b. A parameter space (such as ParameterGrid in scikit-learn which gives a grid of parameters with a discrete number of values for each). For example:

max\_depth : [ 3, 5, 10, 15, 20]

max\_features : [ 1, 3, 10, 15, 20]

min\_samples\_split : [1, 3, 10]

min\_samples\_leaf : [1, 3, 10]

n\_estimators: [150, 300, 500]

c. A method for searching or sampling candidates (such as expon, gamma, uniform and randint in scipy.stats module that provides a random variate sample)

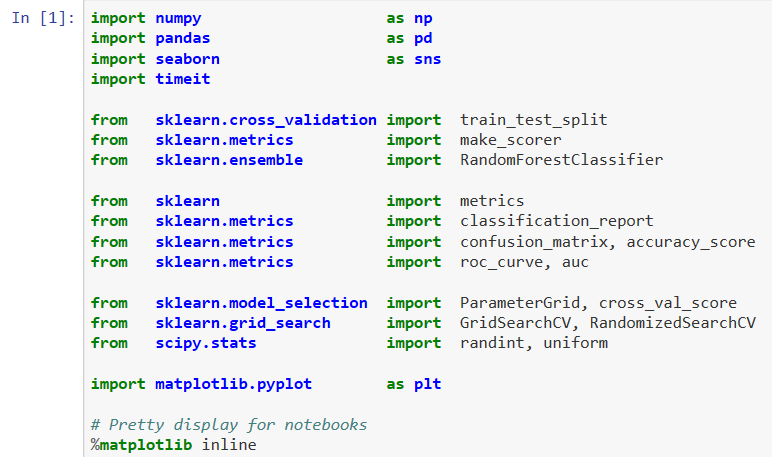
d. A cross-validation scheme. We use 10-fold cross-validation to choose the best combination as the final model

e. A score function is average weighted F1 ratio to evaluate a parameter setting. , F1 ratio is the harmonic mean of precision and recall. Here, average = 'weighted' calculates metrics for each label, and their average weighted by support (i.e. number of true instances for each label.

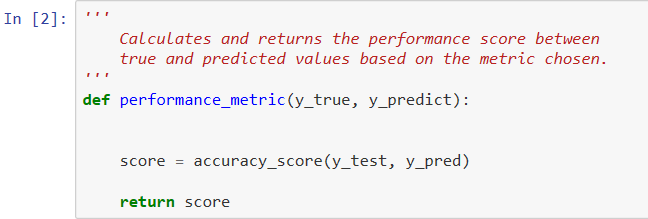
**Data preparation for Random Forest**

* In the first approach, we will use the default options for the random forest model, with two exceptions.
* The default number of trees made by a random forest in sklearn is a meager 10. We shall use 150.
* Parameter n\_jobs is set to = -1. This is the number of jobs to run in parallel for both fit and predict. None means 1 unless in a joblib.parallel\_backend context. -1 means using all processors. For more info, refer to <https://scikit-learn.org/stable/glossary.html#term-n-jobs>.

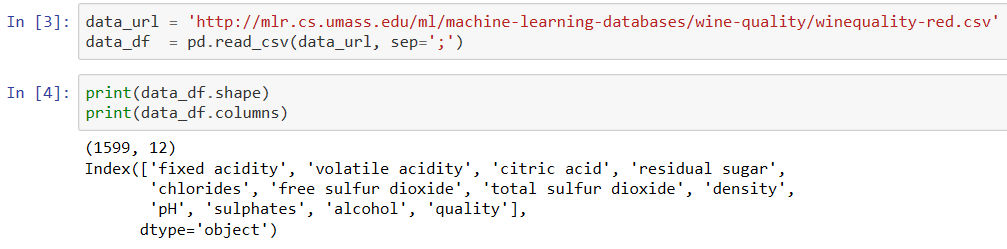
**Import required libraries**

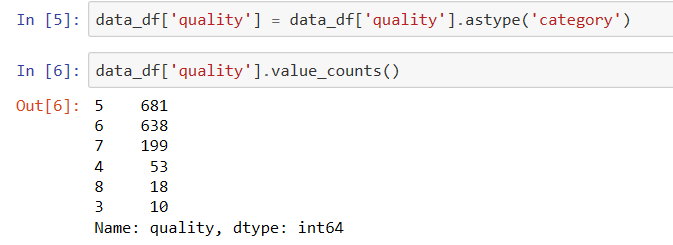


**Define a performance metric**

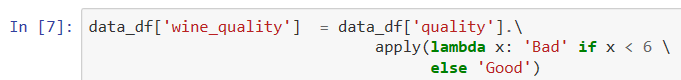


**Read the data**

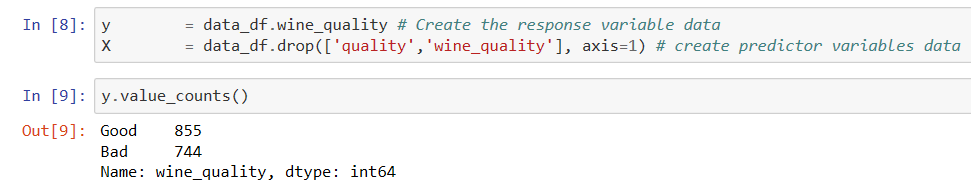
****

****

**Let’s classify the wines into good and bad based on their quality. If quality is below 6, it is bad else it is good.**

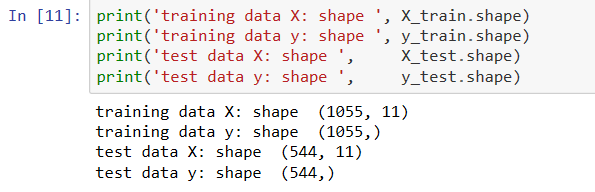
****

**Create predictor variables, X and the response variable, y. Find the distribution of classes in the response variable, y.**

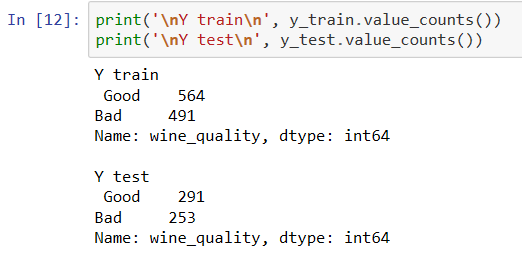
****

**Split the data into training data and test data in the ratio 66%: 34%.**

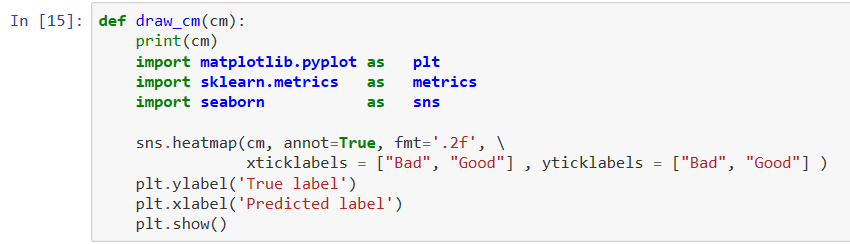
****

****

**Class composition in training and test data**

****

**Write a function to show confusion matrix**

****



### **Grid Search**

**Step 1: Create cross-validation sets**

**Step 2: Instantiate the classifier object**

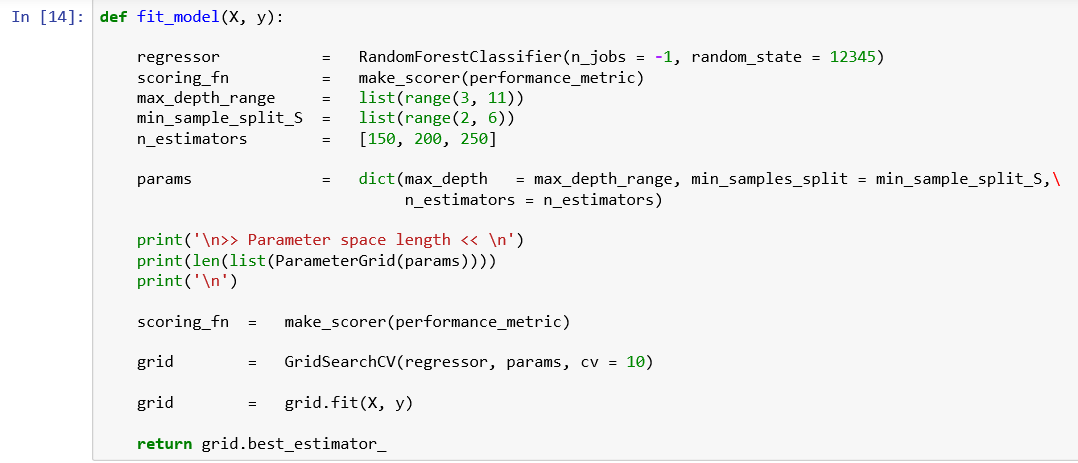
**Step 3: Create a dictionary for the parameter 'max\_depth' with a range from 3 to 10 , 'min\_sample\_split' from 2 to 5 and n\_estimators with value from the list 150, 200 and 250**

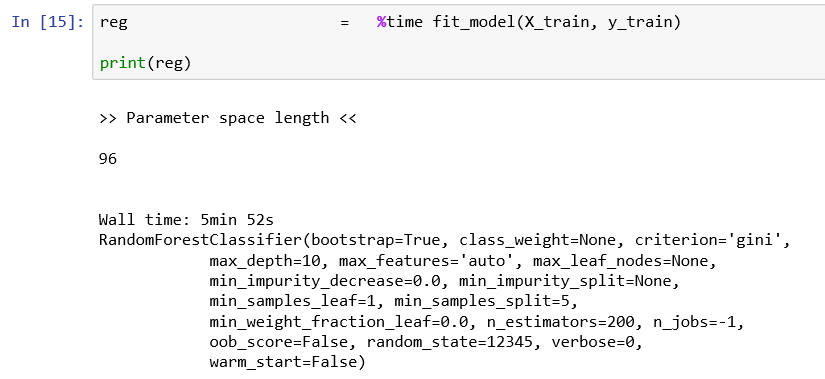
**Step 4: Transform 'performance\_metric' into a scoring function using 'make\_scorer'**

**We have created performance\_metric using accuracy\_score**

**Step 5: Create the grid search object**

**Step 6: Fit the grid search object to the data to compute the optimal model**

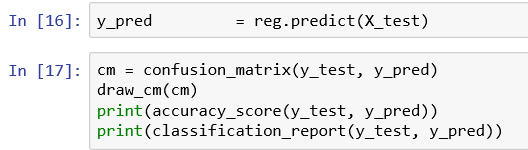


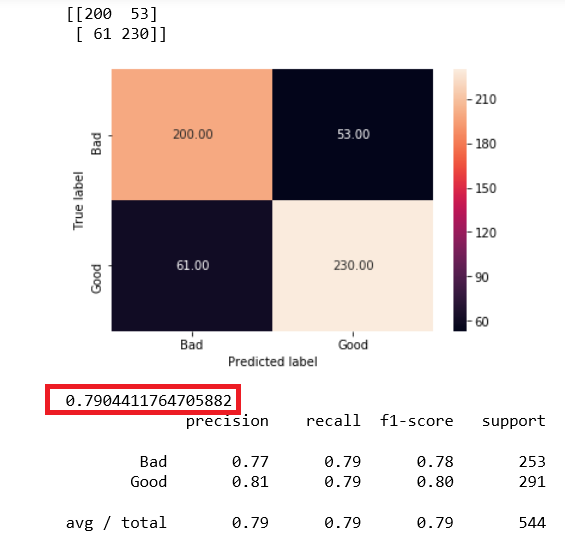


**Observations:**

1. **There are 96 combinations in the parameter space for the grid search to explore.**
2. **Execution time for this grid search algorithm is 5 minutes 52 sec.**
3. **The best values found by the grid search algorithm:**
   * 1. **max\_depth = 10**
     2. **min\_samples\_split = 5**
     3. **n\_estimators = 200**

**Evaluate the model performance using the test data**

****

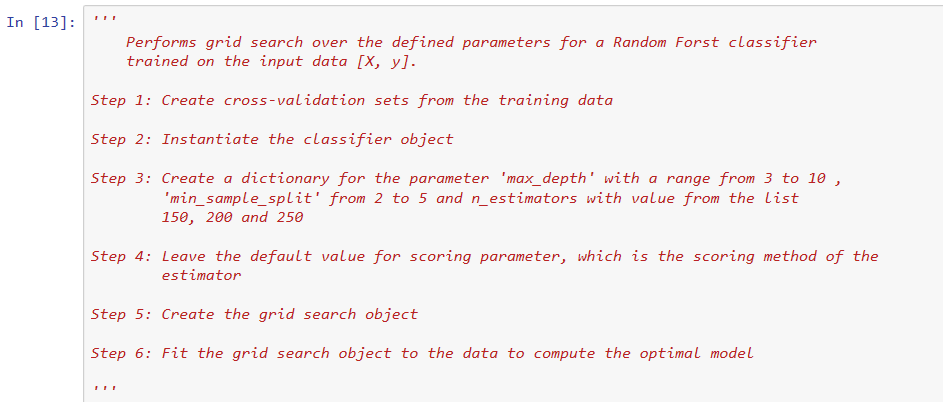


**Observation:**

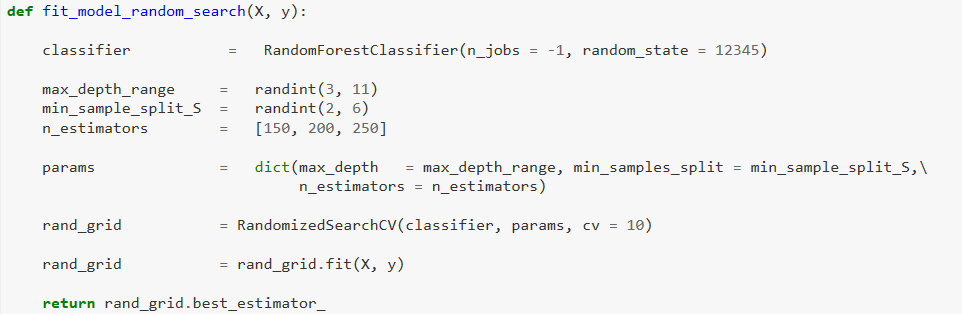
**Accuracy\_ratio = 0.79 and it is good. So are the values of Recall and precision for both classes.**

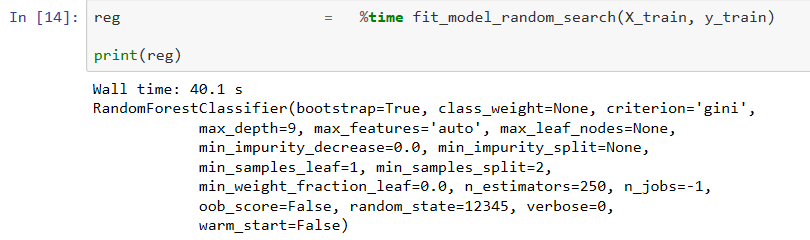
### **Random Search**

* Using Scikit-Learn’s RandomizedSearchCV method, you can define a grid of hyperparameter ranges, and randomly sample from the grid, performing cross validation with each combination of values.
* We shall use random grid search to select the best set of hyperparameters for RandomForest Classifier on the same wine data set.



We have used randint in scipy.stat module to select a random variable sample for the selected parameters.

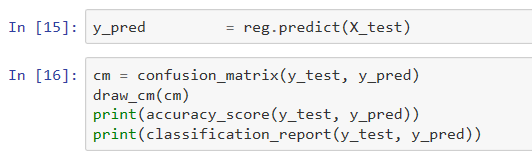


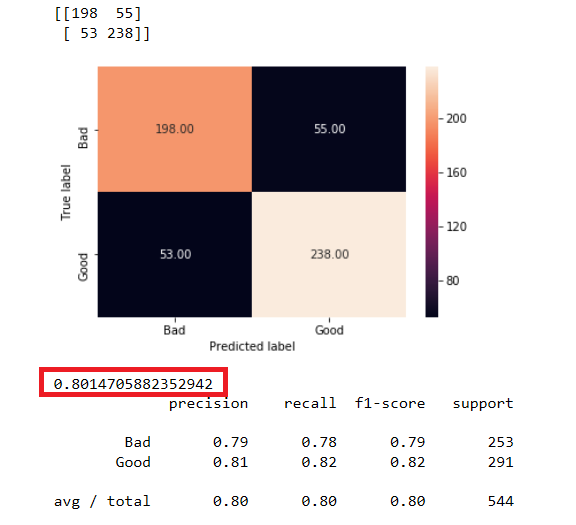


**Observations:**

1. **Execution time for this grid search algorithm is 40.1 seconds only**
2. **The best values found by the grid search algorithm:**
   * 1. **max\_depth = 9**
     2. **min\_samples\_split = 2**
     3. **n\_estimators = 250**

**Evaluate model performance**





**Observation:**

**Accuracy\_ratio = 0.8015 and it is good. So are the values of Recall and precision for both classes.**

## SURROGATE MODEL ALGORITHMS - CLASSIFICATION

* There are five elements of model-based hyperparameter optimization:

1. *A domain of hyperparameters over which to search*
2. *An objective function which takes in hyperparameters and outputs a score that we want to minimize (or maximize)*
3. *The surrogate model of the objective function*
4. *A criteria, called a selection function, for evaluating which hyperparameters to choose next from the surrogate model*
5. *A history consisting of (score, hyperparameter) pairs used by the algorithm to update the surrogate model*

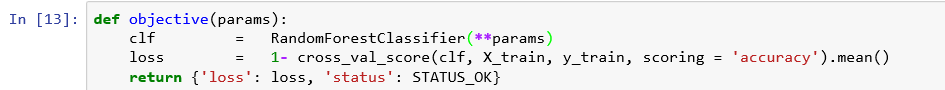
### **Bayesian Optimization**

We shall use Hyperopt (https://hyperopt.github.io/hyperopt/) to put Bayesian optimization into practice.

**Steps in formulating an optimization problem in hyperopt:**

1. Create objective function which takes hyper parameter space as input and returns a loss minimize. We are taking loss as 1 – ACCURACY which gives mis-classification rate . We are making use of ten- fold cross validation and taking its mean to compute the loss for each iteration.



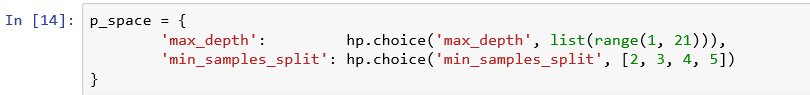


* Note: We need to define a function that return the negative of that metric since our objective function returns a real value that we want to minimize.

1. Domain space which contains the range of input values to evaluate.

We are using both max\_depth and min\_samples\_split as hyper parameters to be tuned.

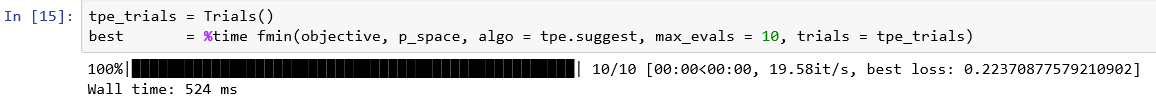
We define the range of values for max\_depth as 1 to 20 and for min\_samples\_split as 2 to 5.



1. Optimization algorithm is the method used to construct the surrogate function and choose the next values to evaluate. We are using Tree structured Parzan Estimator (TPE)\* model and let hyperopt to configure it by using the suggest method.

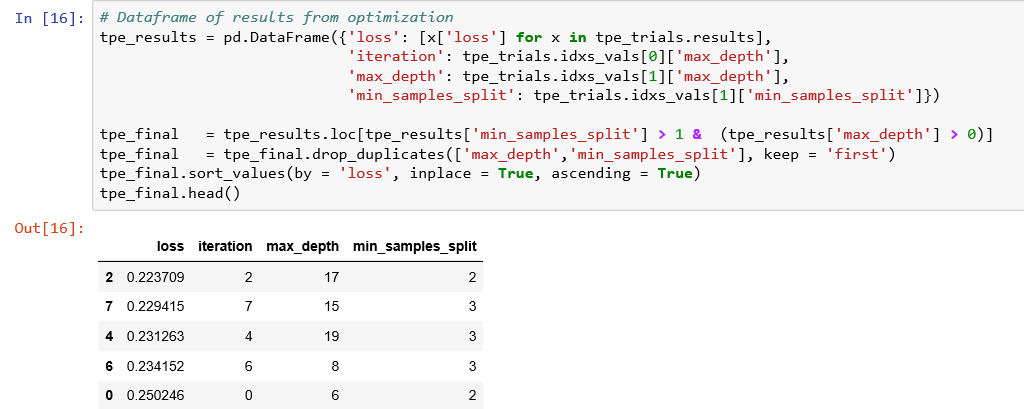
Note: Each iteration TPE collects new observation and at the end of the iteration, the algorithm decides which set of parameters it should try next.

1. Results which contains a score value pair that the algorithm uses to build the model



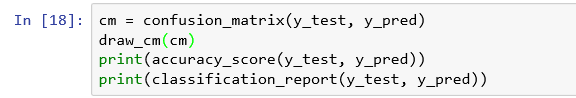
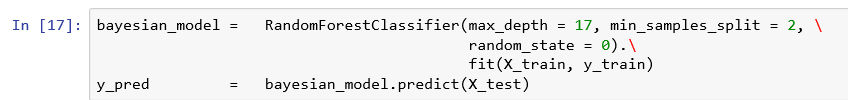
* We have created a Trials object that will record the values and the scores to find out the value - score pair for each trial.
* We have minimized our objective function by using fmin() function that takes four parts above as well as a maximum number of trials (10).
* Hyperopt method took 524 milli seconds while the method GridSearchCV took 5 minutes 52 seconds for execution and RandomizedSearchCV, method took only 40.1 seconds for the same data set.

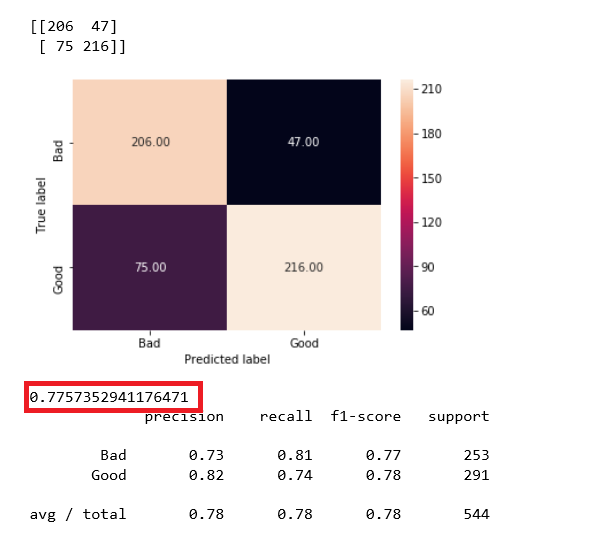
**Find the optimum hyper parameters**

****

* Hyperopt method (a Bayesian method) gives the pair of values (max\_depth = 17, min\_samples\_split = 2).

**Build the model using these hyper parameters**





|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Accuracy Ratio** | **Average F1 score** | **Execution time** |
| **GridSearchCV** | **0.7904** | **0.79** | **5 min 52 sec** |
| **RandomizedSearchCV** | **0.8014** | **0.80** | **40.1 sec** |
| **Bayseian - Hyperopt** | **0.7757** | **0.78** | **524 ms** |

**References:**

<https://medium.com/@ramrajchandradevan/comparison-among-hyper-parameter-optimizers-cd37483cd47>

<https://medium.com/criteo-labs/hyper-parameter-optimization-algorithms-2fe447525903>

<https://blog.nanonets.com/hyperparameter-optimization/>

<https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a>

<https://www.analyticsindiamag.com/what-are-hyperparameters-and-how-do-they-determine-a-models-performance/>