Analytics using Python

Learning outcomes

How to tune hyper-parameters for both classification and regression problemsusing Python

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 13. HYPER TUNING PARAMETERS**

**Table of Contents**

[1. HYPER PARAMETERS TUNING 4](#_Toc19037109)

[1.1. Introduction 4](#_Toc19037110)

[1.2. ComparE HYPERPARAMETERS AND MODEL PARAMETERS 4](#_Toc19037111)

[1.3. DIAGRAM SHOWING HYPERPARAMETERS AND MODEL PARAMETERS 5](#_Toc19037112)

[1.4. Why do we need hyperparameters? 5](#_Toc19037113)

[1.5. CLASSES OF Hyperparameter optimization 6](#_Toc19037114)

[**1.5.1.** **Exhaustive search of the space category** 6](#_Toc19037121)

[**1.5.2.** **Surrogate models** 7](#_Toc19037122)

[**1.5.3.** **Other models** 8](#_Toc19037123)

[1.6. Exhaustive search of the space ALGORITHMS -REGRESSION 8](#_Toc19037124)

[**1.6.1.** **Grid Search** 9](#_Toc19037147)

[**1.6.2.** **Random Search** 21](#_Toc19037148)

[1.7. SURROGATE MODEL ALGORITHMS - REGRESSION 27](#_Toc19037149)

[**1.7.1** **Bayesian Optimization** 27](#_Toc19037194)

[1.8. Exhaustive search of the space ALGORITHMS -CLASSIFICATION 33](#_Toc19037195)

[**1.8.1.** **Grid Search** 38](#_Toc19037205)

[**1.8.2.** **Random Search** 41](#_Toc19037206)

[1.9. SURROGATE MODEL ALGORITHMS - CLASSIFICATION 44](#_Toc19037207)

[**1.9.2** **Bayesian Optimization** 44](#_Toc19037211)

[2. HYPER PARAMETERS – CLASSIFICATION MODELS 48](#_Toc19037212)

[2.1. Important hyper-parameters for classification models 51](#_Toc19037214)

[3. HYPER PARAMETERS – REGRESSION MODELS 56](#_Toc19037218)

[3.1. Important hyper-parameters for REGRESSION models 59](#_Toc19037220)

# HYPER PARAMETERS TUNING

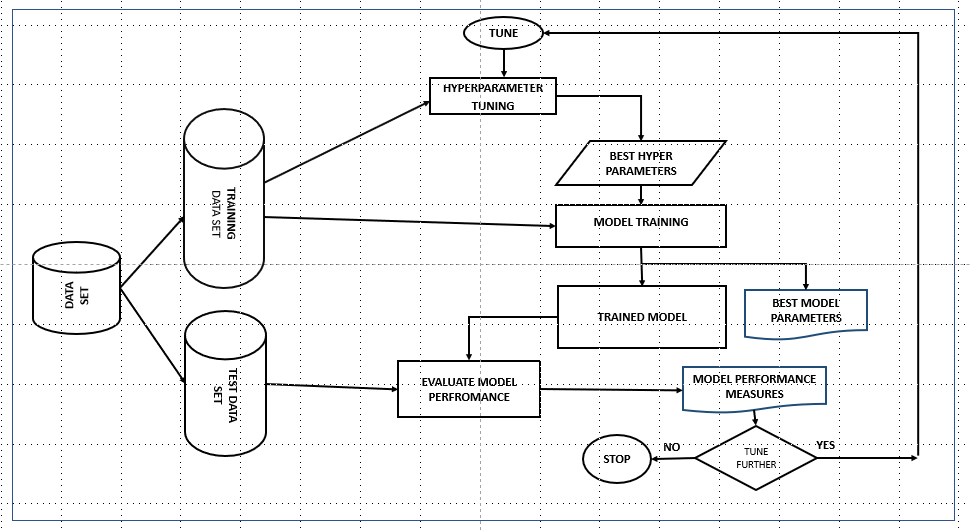
## Introduction

* 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.

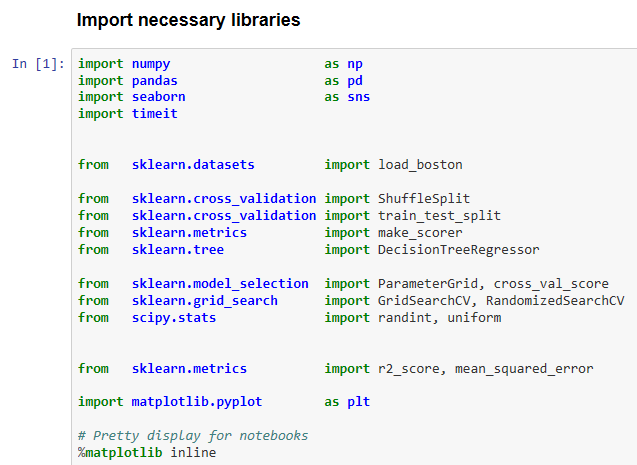
### **Grid Search**

* 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.

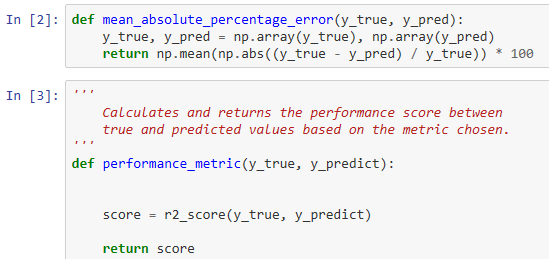
Refer: <https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html>

* 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.

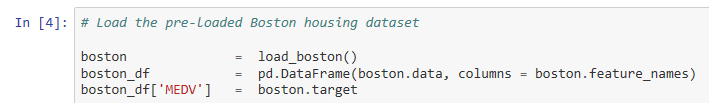
**Import required packages and load data**

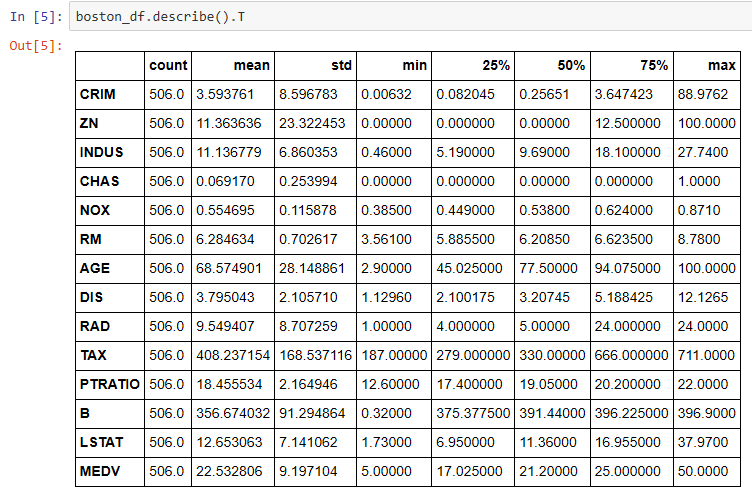
****

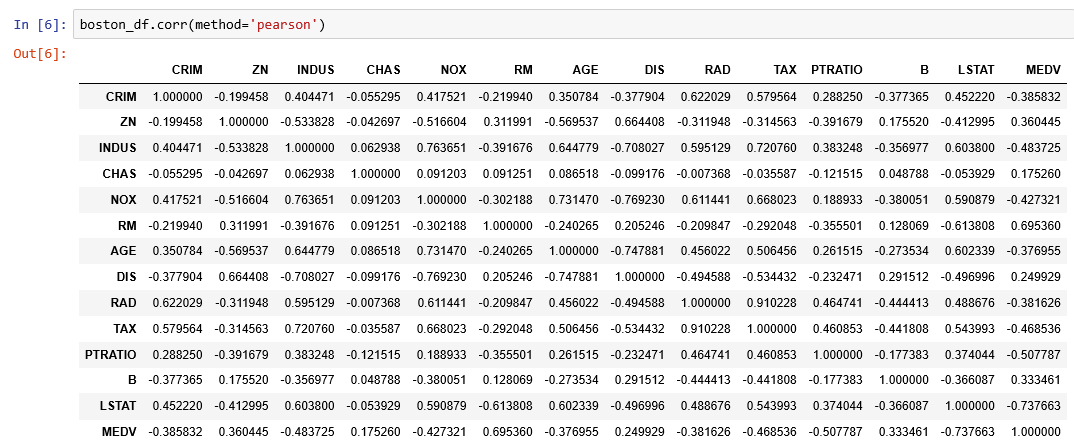
**Define useful functions**

****

**Load the data**

****

**Make a customary investigation into the data **

****

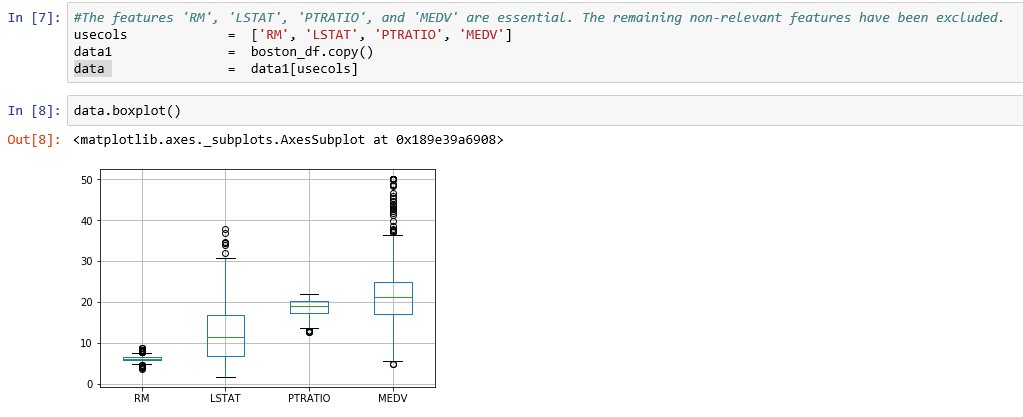
* From the above correlation matrix, we observe that

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:

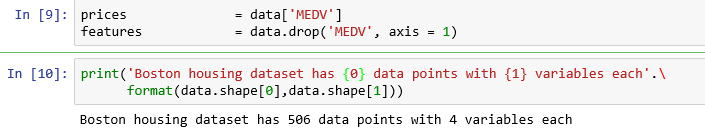
1. RM, which is the average number of rooms among the homes in the neighborhood
2. LSTAT, the percentage of homeowners in the neighborhood considered working poor class
3. PTRATIO, the ratio of students to teachers in primary and secondary schools in the neighborhood

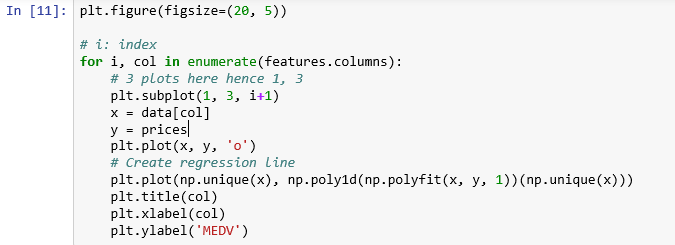
**Check for outliers**

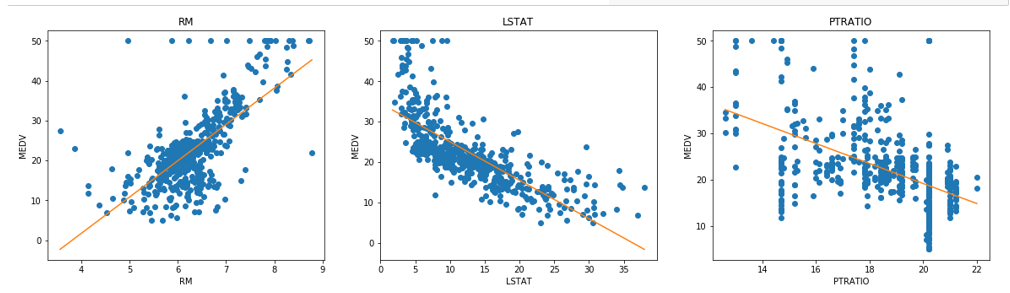
****

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.

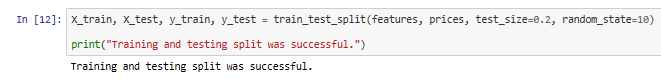
Create predictor variables data set and response variable data set.

****

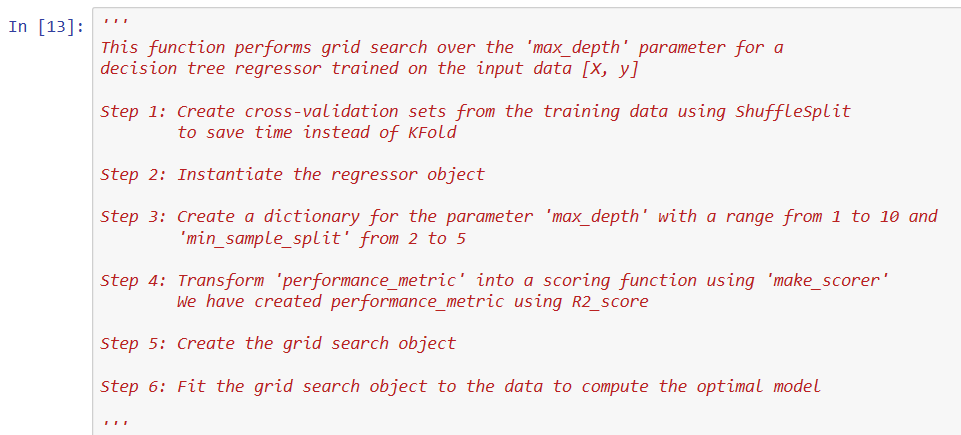
****

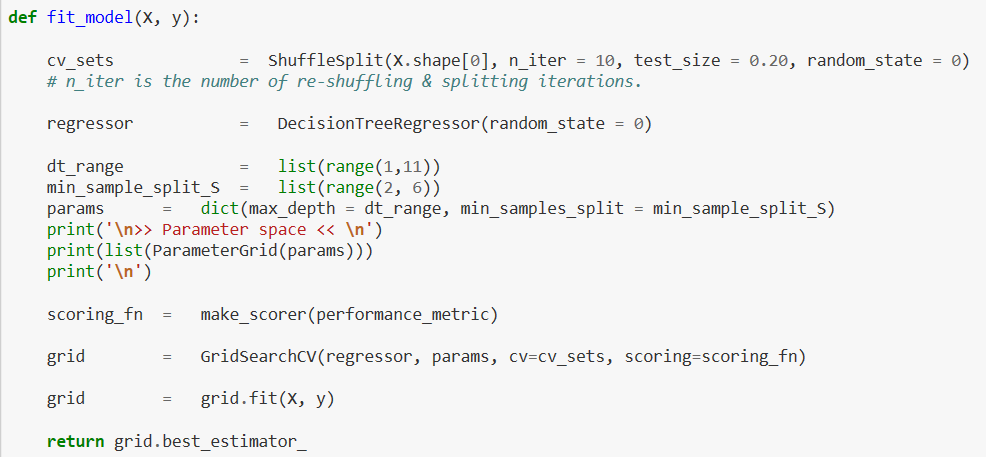
****

**Create Training data and test data**

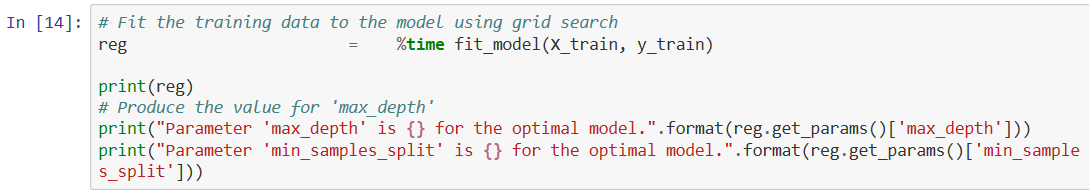
****

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

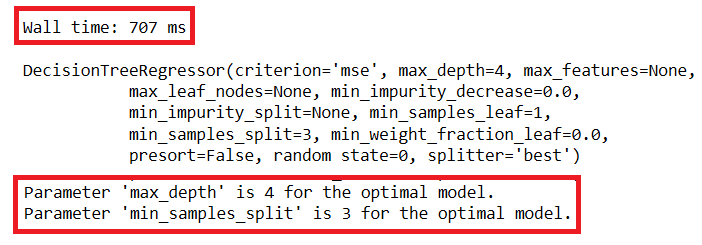
****

****

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

****

****

****

**Here, we observe the following:**

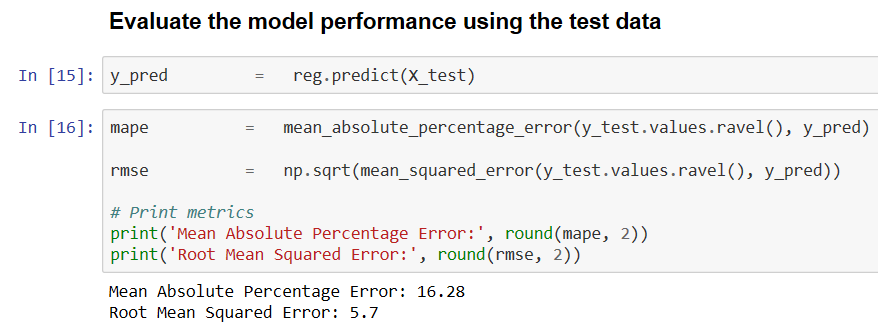
* Our grid search consists of

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.

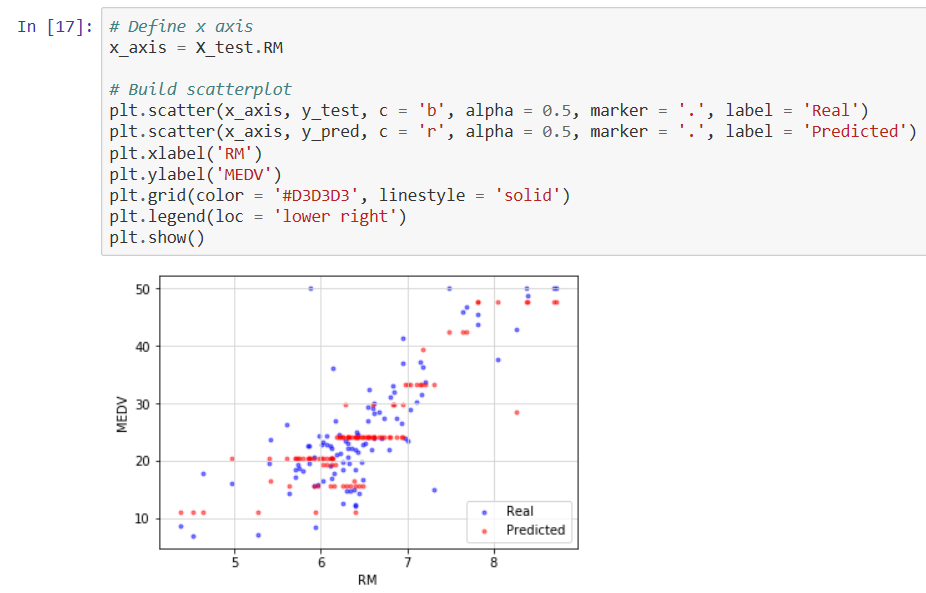
1. A method for searching or sampling candidates: Not Applicable since we need the entire candidates.
2. A cross-validation scheme: ShuffleSplit, a random permutation cross-validator with 10 number of re-shuffling & splitting iterations. This is used instead of KFold
3. A score function: R square as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

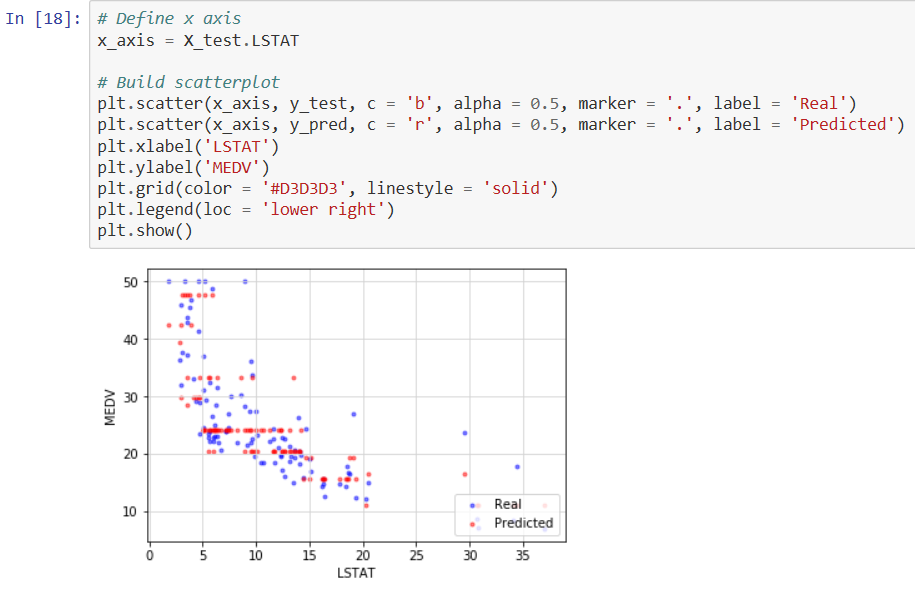
**Correlation coefficient with MEDV is 0.6954**



**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**

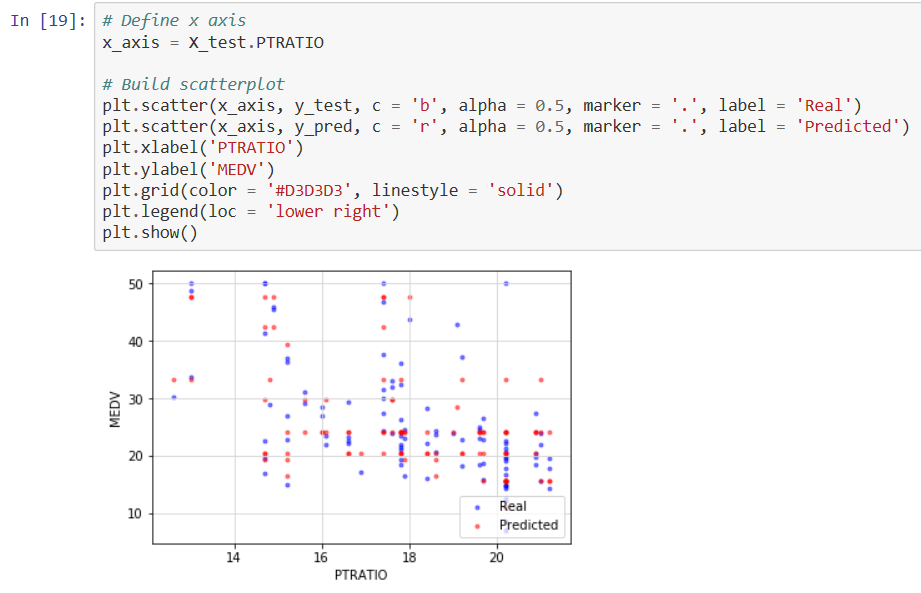
**Correlation coefficient with MEDV is -0.737663**

****

**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**

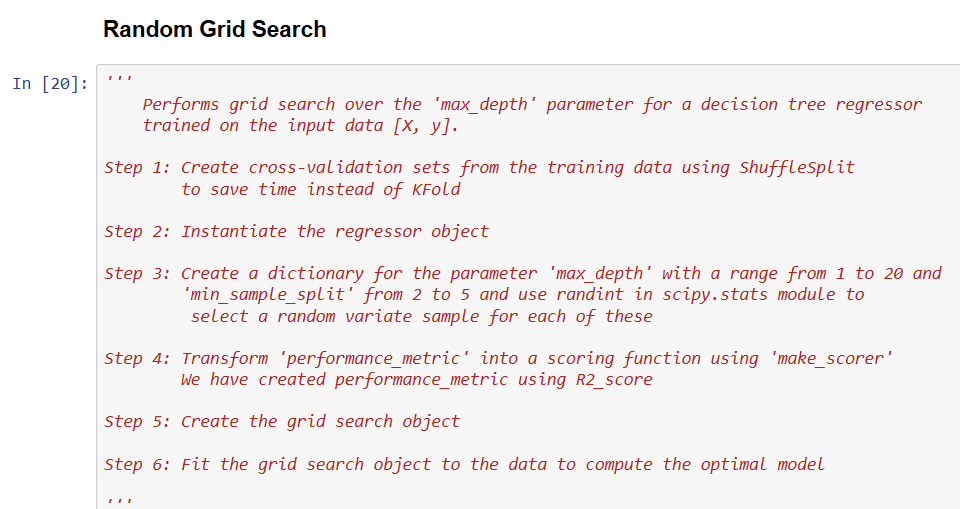
**Correlation coefficient with MEDV is -0.5078**

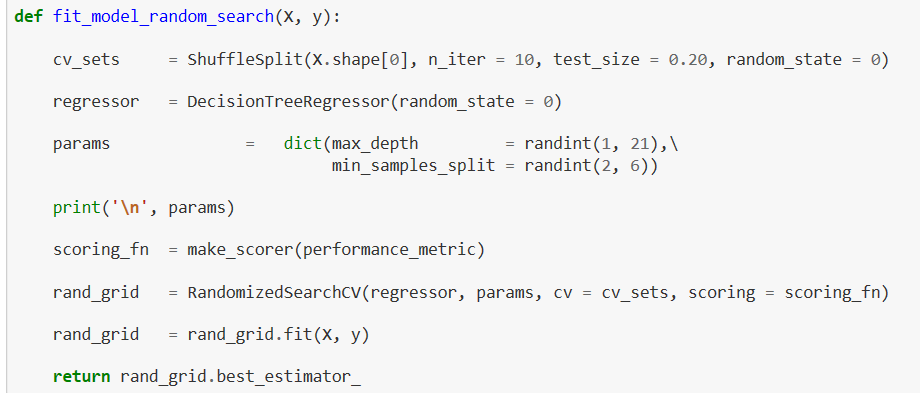
****

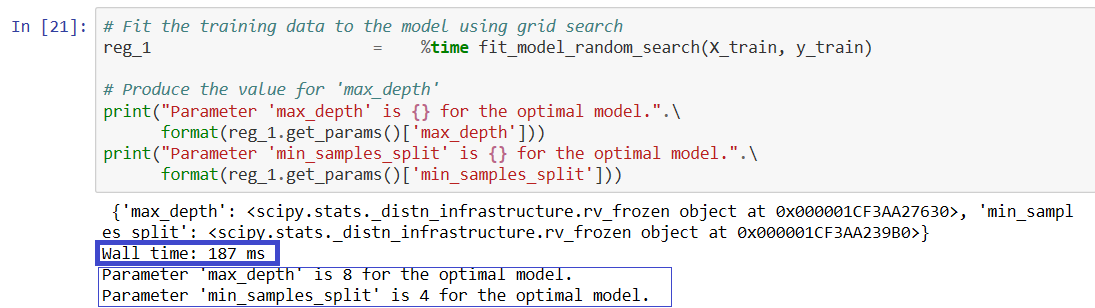
**You can observe how the model predicts MEDV for various values of PTRATIO. Check whether the model predicts MEDV for lower values of PTRATIO or higher values of PTRATIO.**

### **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.





****

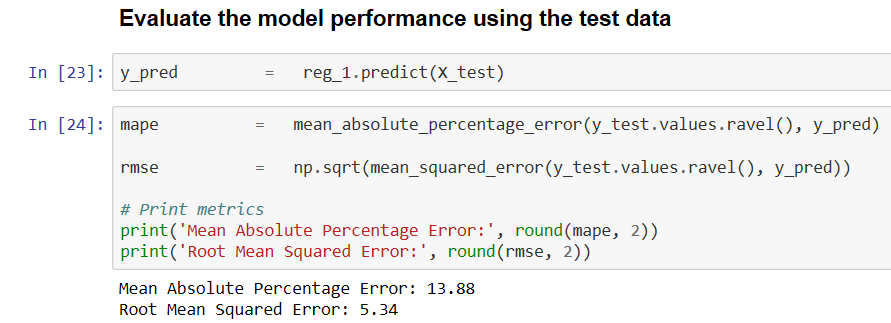
**Observations:**

* 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.

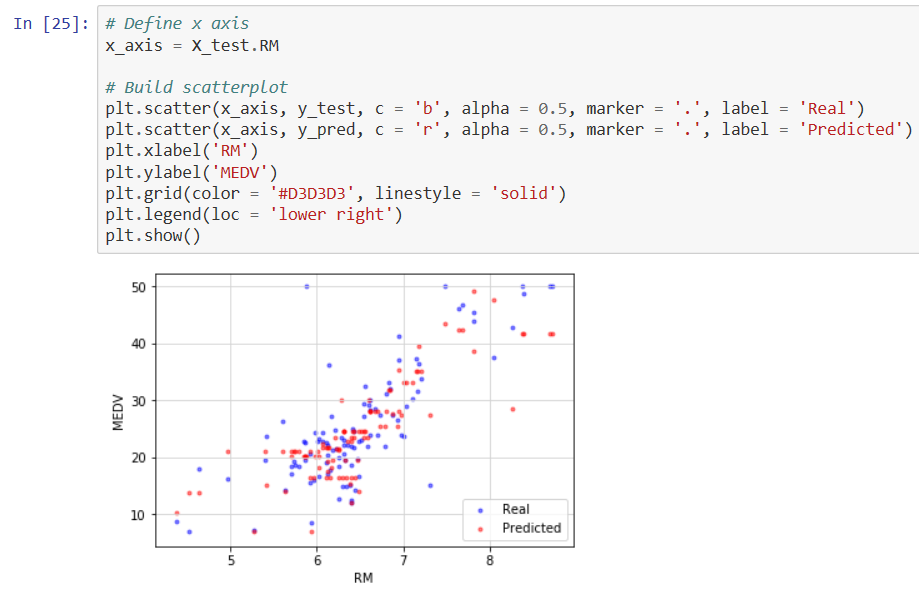


**Observations:**

* 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**

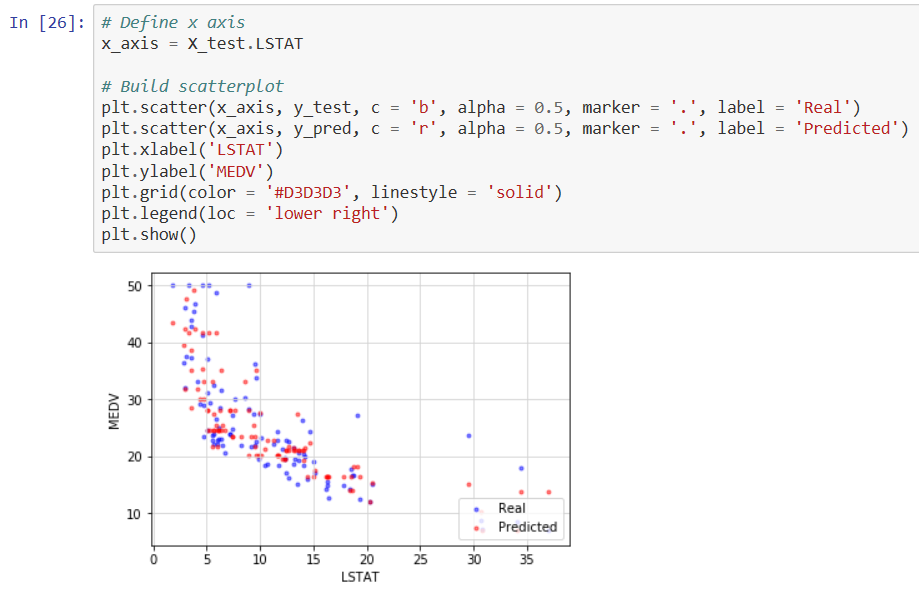
**Correlation coefficient with MEDV is 0.6954.**



**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**

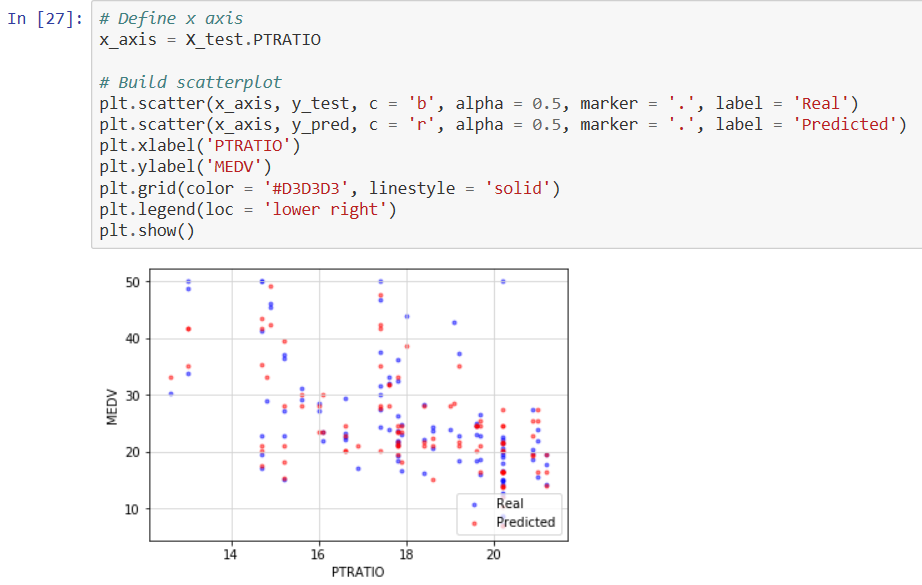
**Correlation coefficient with MEDV is -0.737663**



**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.*

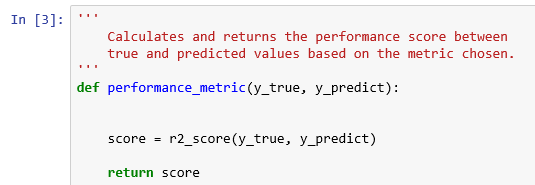


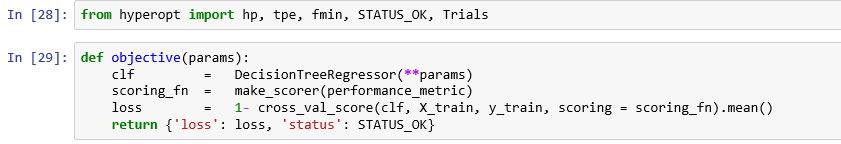
### **Bayesian Optimization**

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.

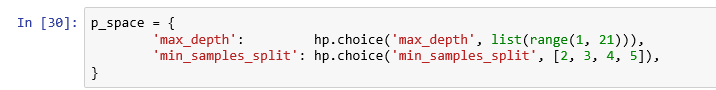




* We need to maximize the R square or minimize 1 – R square.
* Note: We need to define a function that return the negative of that metric since our objective function returns a real value that we want to minimize.
  1. Domain space which contains the range of input values to evaluate.

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

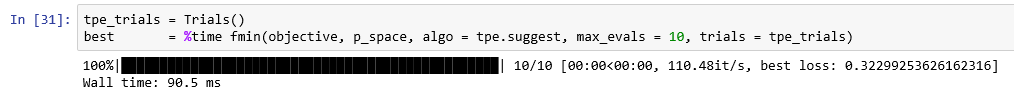
We define the range of values for max\_depth as 1 to 20 and for min\_samples\_split as 2 to 5



* 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.

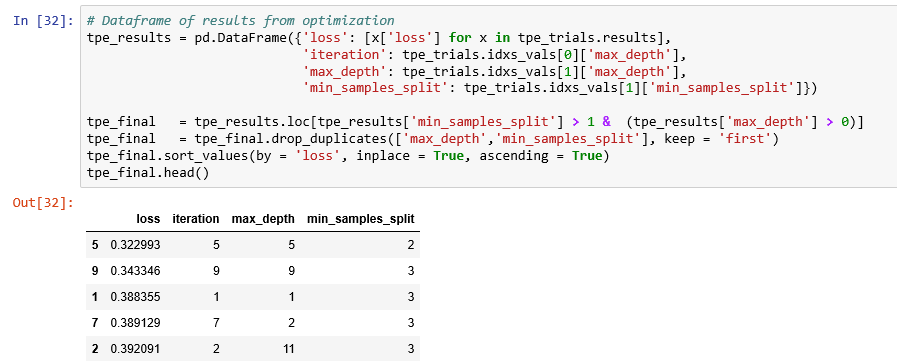
Note: Each iteration TPE collects new observation and at the end of the iteration, the algorithm decides which set of parameters it should try next.

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



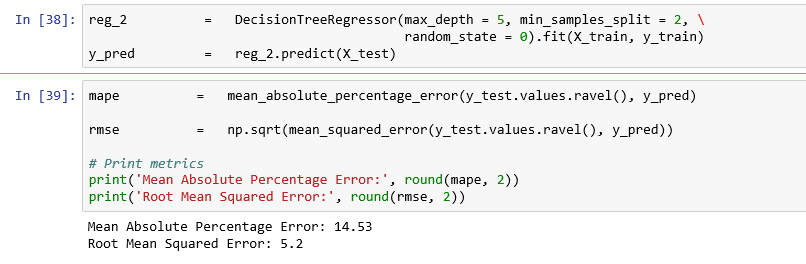
* We have created a Trials object that will record the values and the scores to find out the value - score pair for each trial.
* We have minimized our objective function by using fmin() function that takes four parts above as well as a maximum number of trials (10).
* Hyperopt method took 90.5 milli 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)

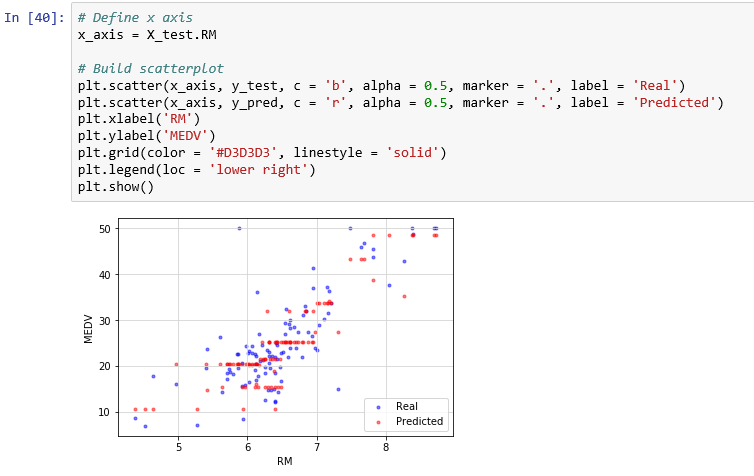
Build the model using these hyper parameters



* 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.

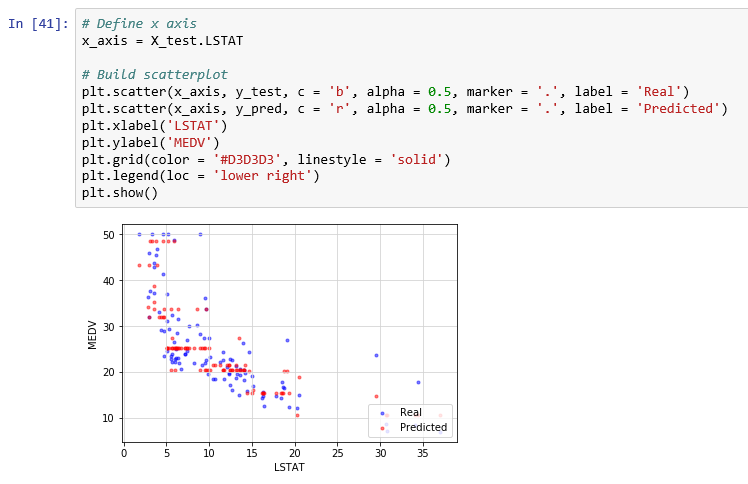
**Visual observed values with predicted values using RM, a predictor variable**

**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.**

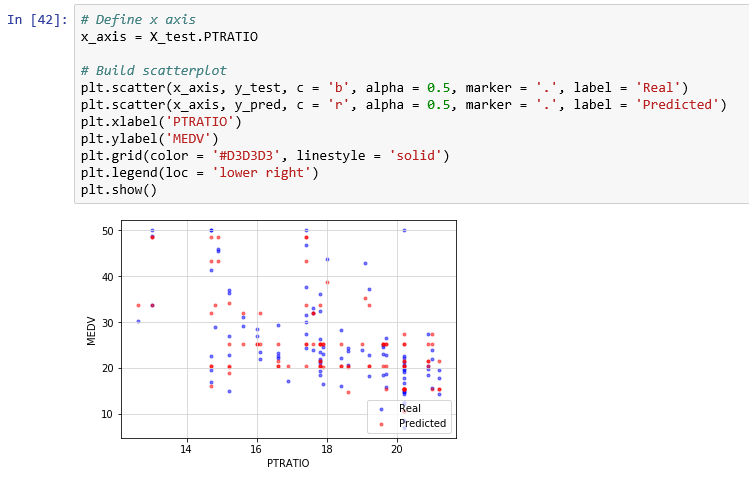
**Visual observed values with predicted values using LSTST, a predictor variable Correlation coefficient with MEDV is -0.737663**



**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)

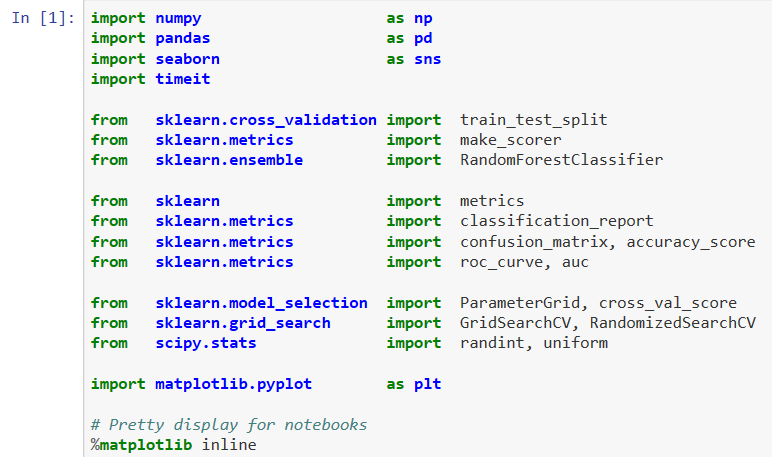
d. A cross-validation scheme. We use 10-fold cross-validation to choose the best combination as the final model

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.

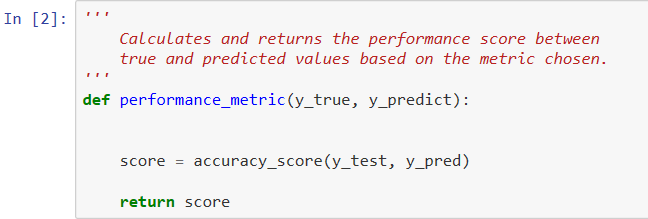
**Data preparation for Random Forest**

* 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>.

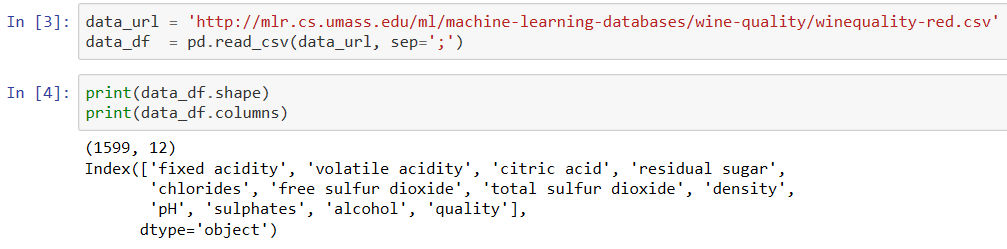
**Import required libraries**

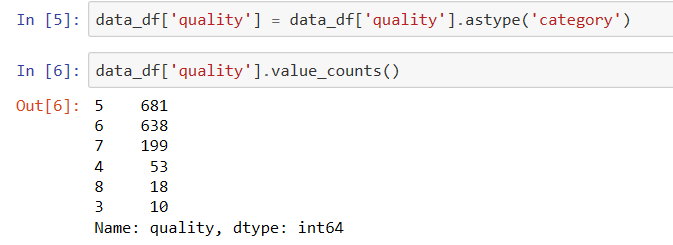


**Define a performance metric**

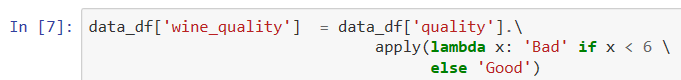


**Read the data**

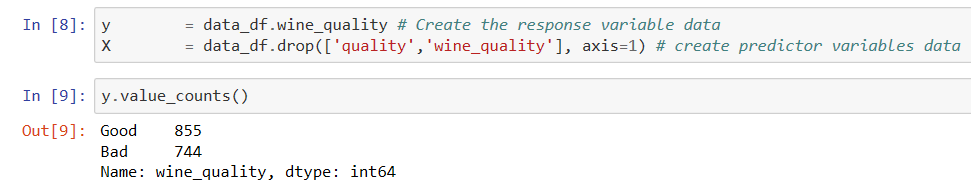
****

****

**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.**

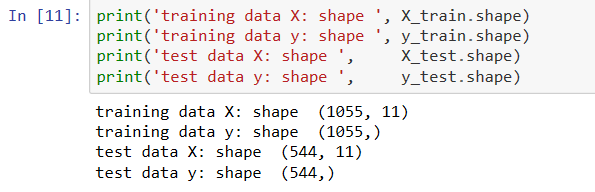
****

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

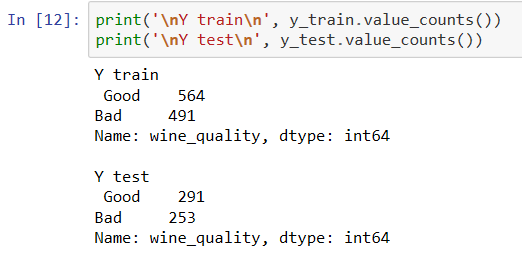
****

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

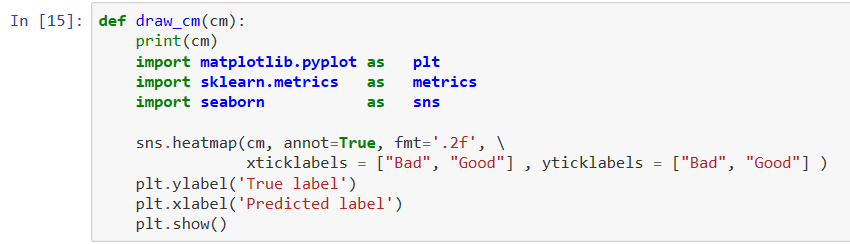
****

****

**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**

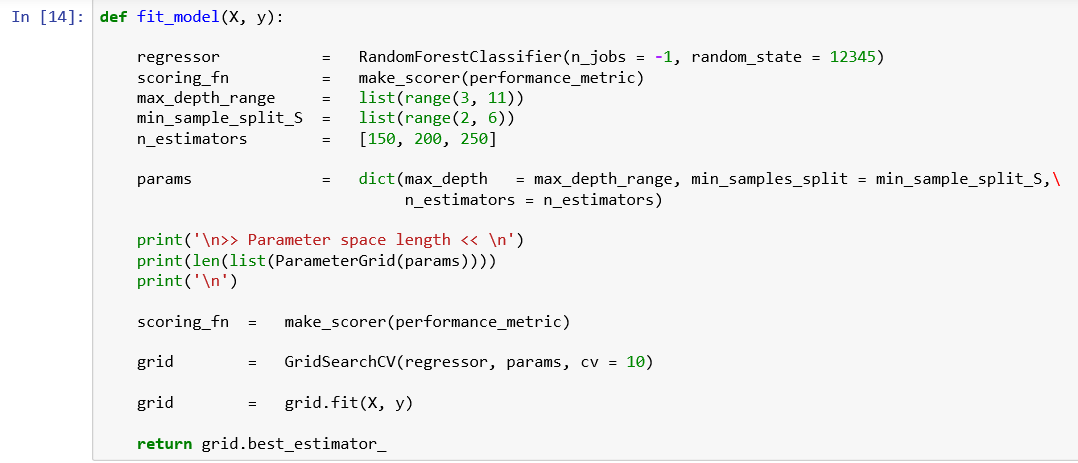
**Step 3: Create a dictionary for the parameter 'max\_depth' with a range from 3 to 10 , 'min\_sample\_split' from 2 to 5 and n\_estimators with value from the list 150, 200 and 250**

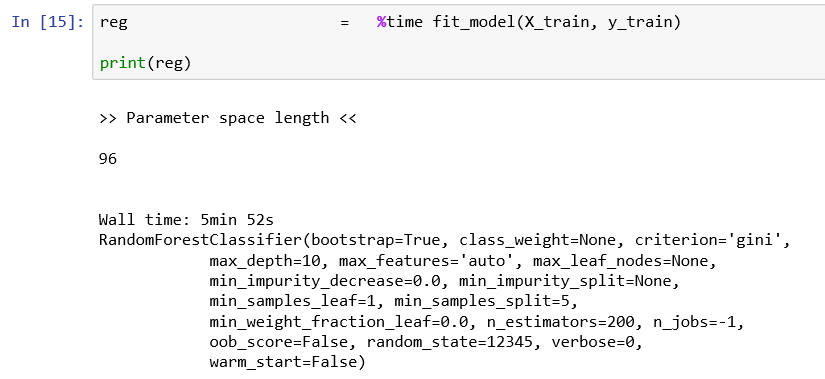
**Step 4: Transform 'performance\_metric' into a scoring function using 'make\_scorer'**

**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**

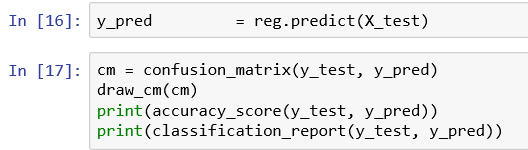


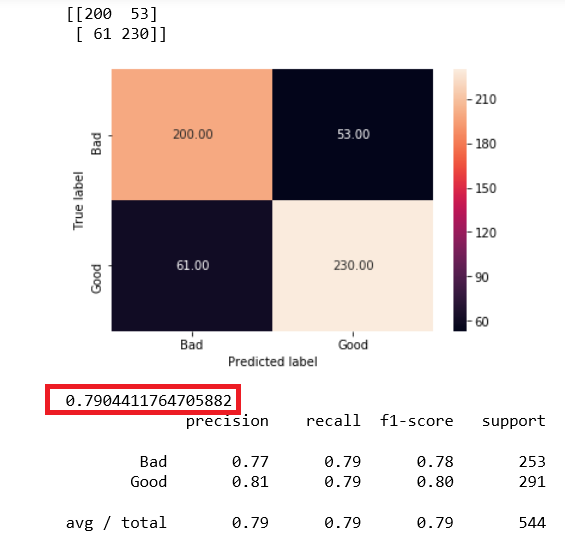


**Observations:**

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**

****

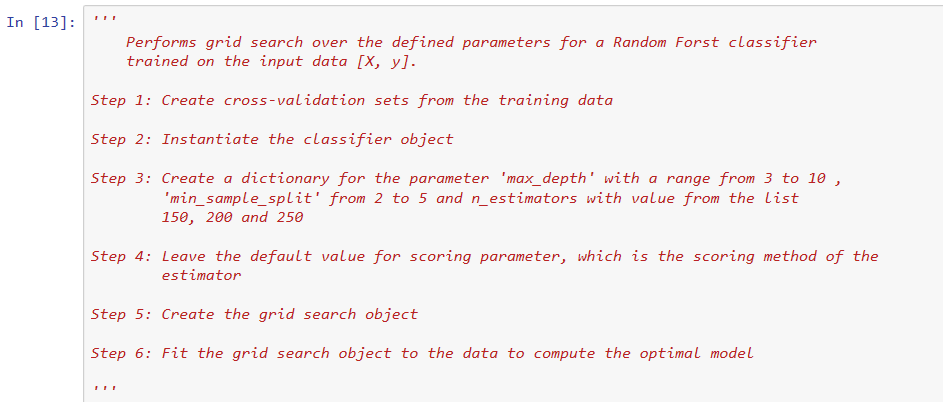


**Observation:**

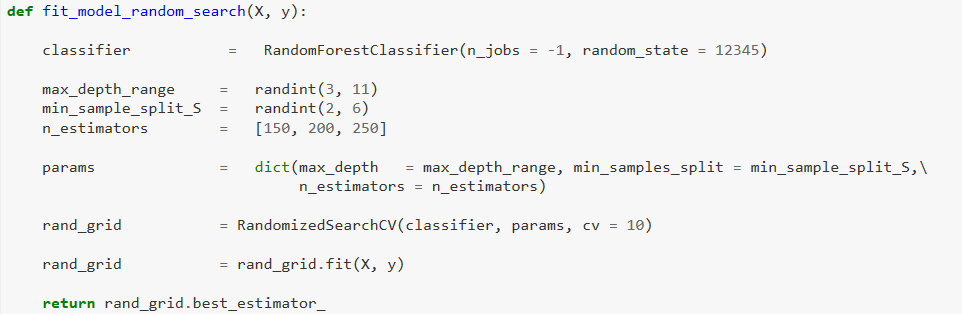
**Accuracy\_ratio = 0.79 and it is good. So are the values of Recall and precision for both classes.**

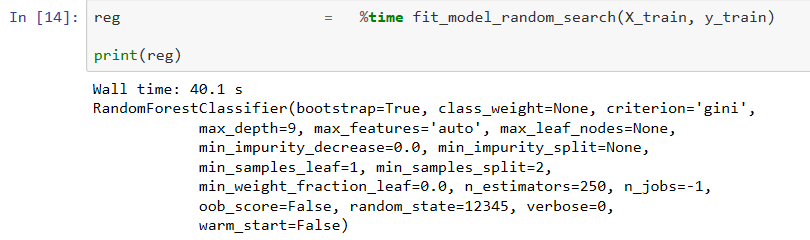
### **Random Search**

* 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.

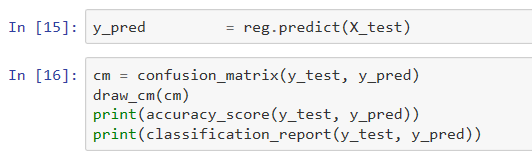


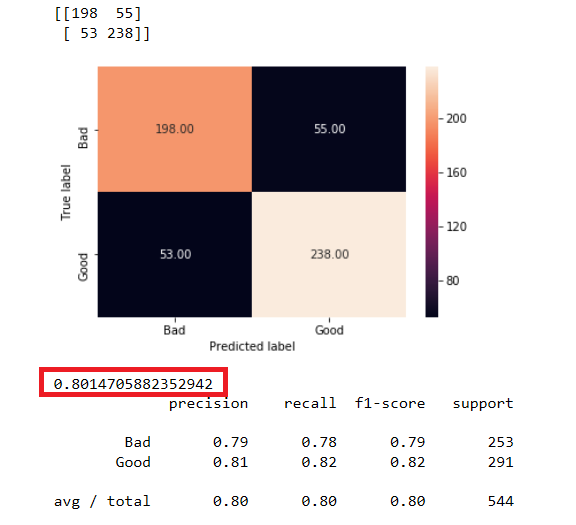


**Observations:**

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*

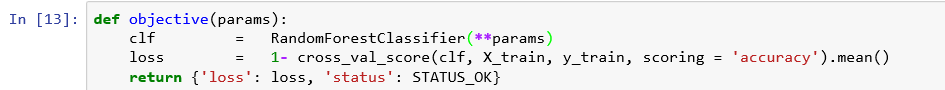
### **Bayesian Optimization**

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.



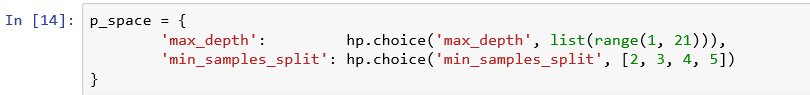


* Note: We need to define a function that return the negative of that metric since our objective function returns a real value that we want to minimize.

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.

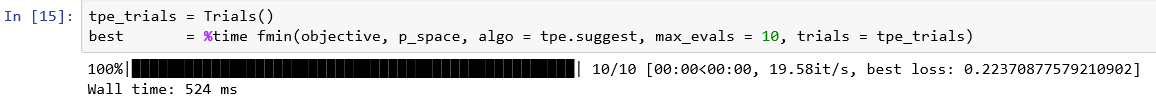
We define the range of values for max\_depth as 1 to 20 and for min\_samples\_split as 2 to 5.



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.

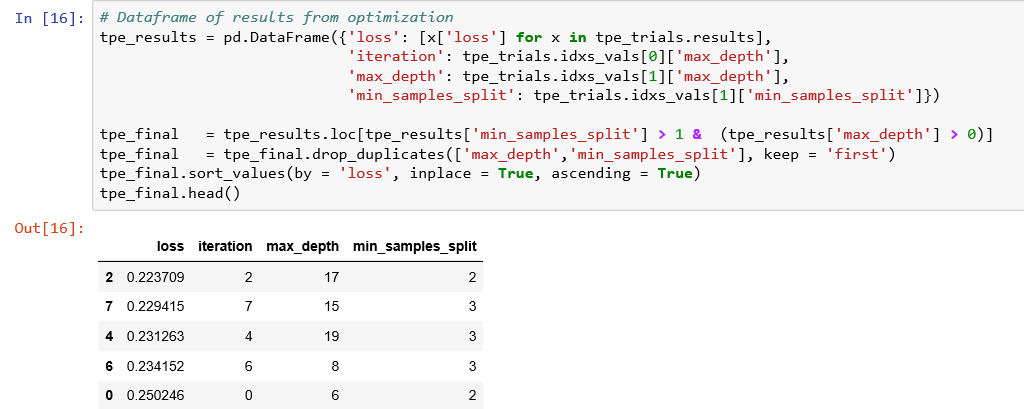
Note: Each iteration TPE collects new observation and at the end of the iteration, the algorithm decides which set of parameters it should try next.

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



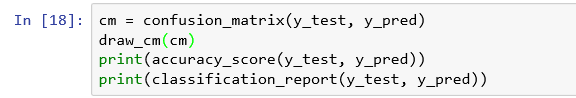
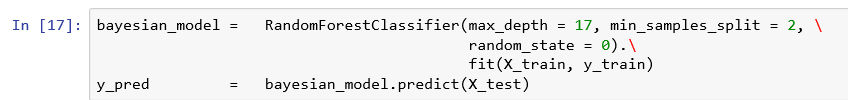
* We have created a Trials object that will record the values and the scores to find out the value - score pair for each trial.
* We have minimized our objective function by using fmin() function that takes four parts above as well as a maximum number of trials (10).
* Hyperopt method took 524 milli 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.

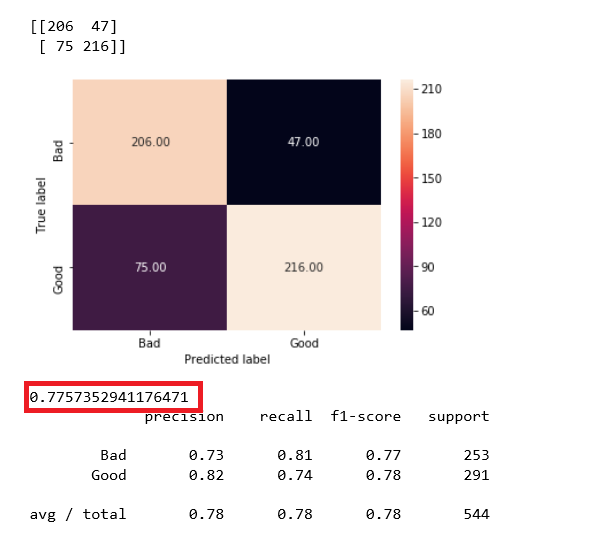
**Find the optimum hyper parameters**

****

* 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>

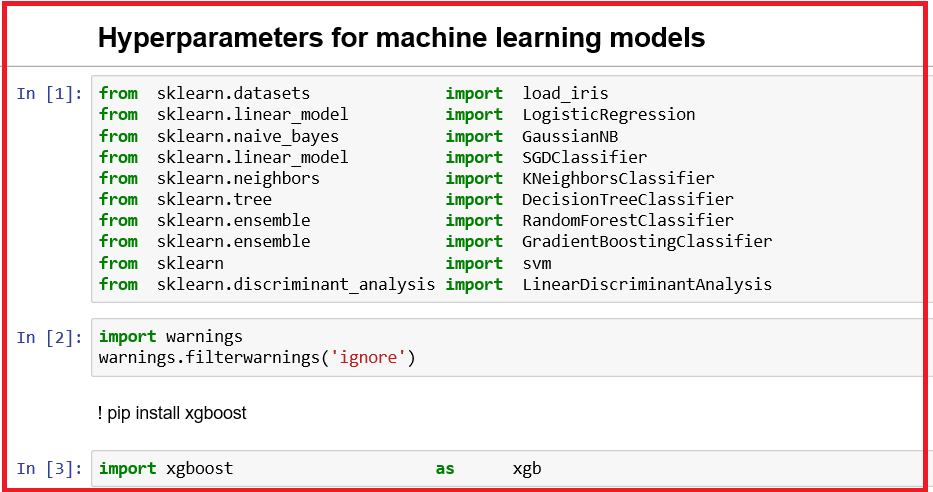
<https://blog.nanonets.com/hyperparameter-optimization/>

<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/>

# HYPER PARAMETERS – CLASSIFICATION MODELS

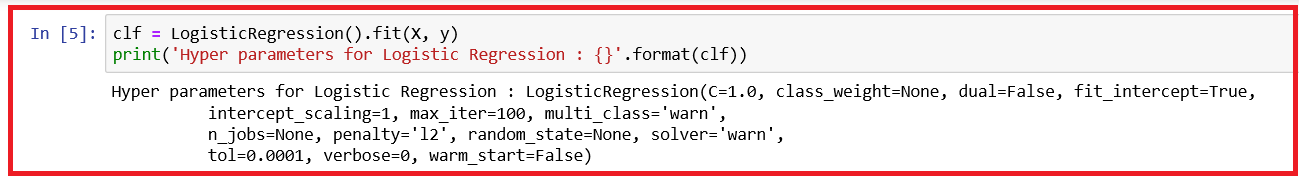
* **How do you find out the hyper-parameters of popular classification models?**
* **See the python code below:**

****

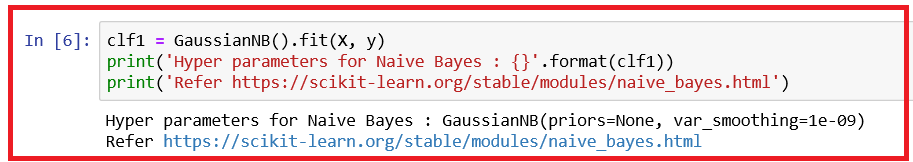
* You have imported the required modules.

****

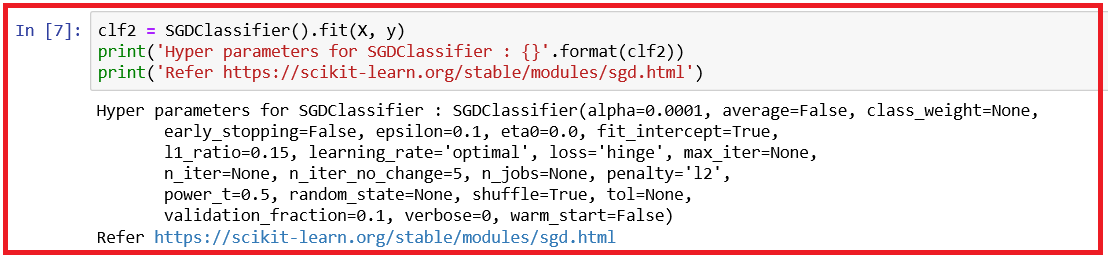
* Create X and y for holding independent variables and dependent variable, respectively.
* Get the hyper-parameters for Logistic Regression



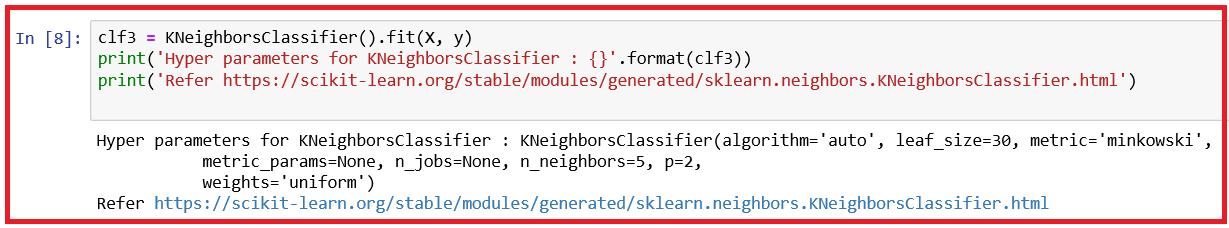
* Get the hyper-parameters for Naïve Bayes Model



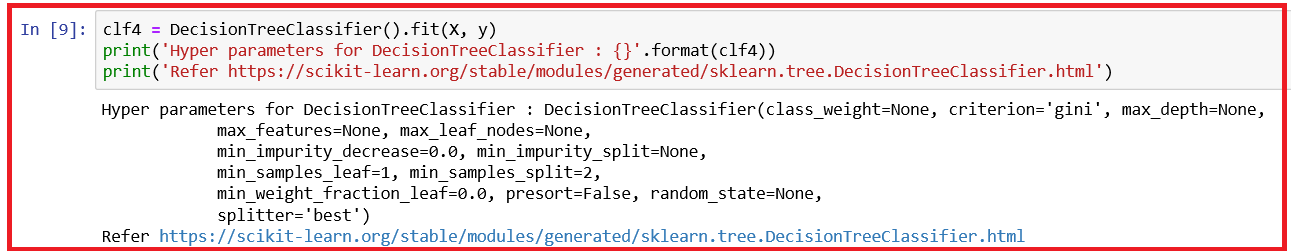
* Get the hyper-parameters for Stochastic Gradient Classifier Model



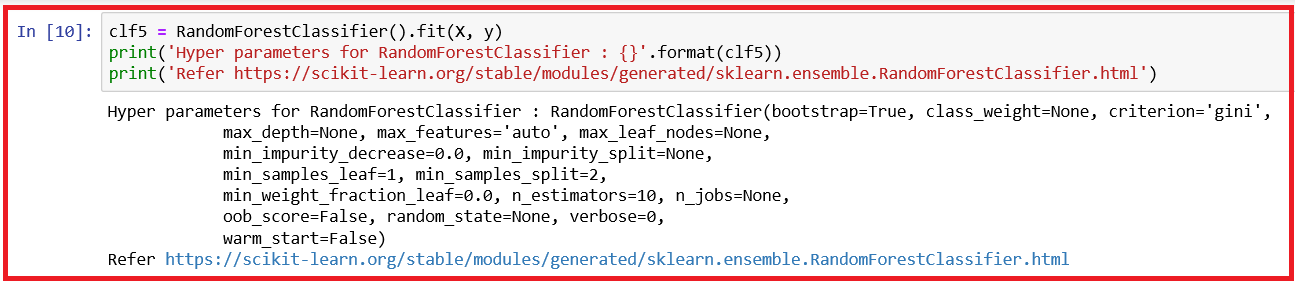
* Get the hyper-parameters for KNN Classifier Model



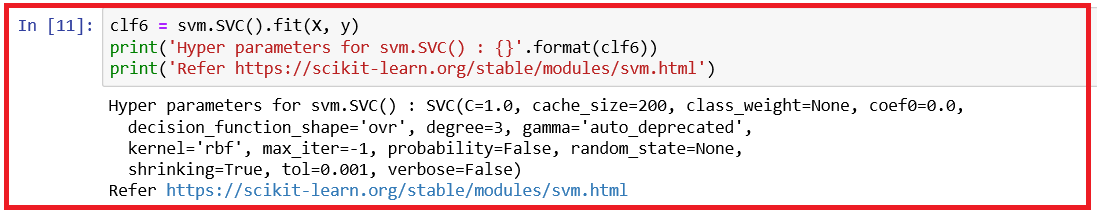
* Get the hyper-parameters for Decision Tree Classifier Model



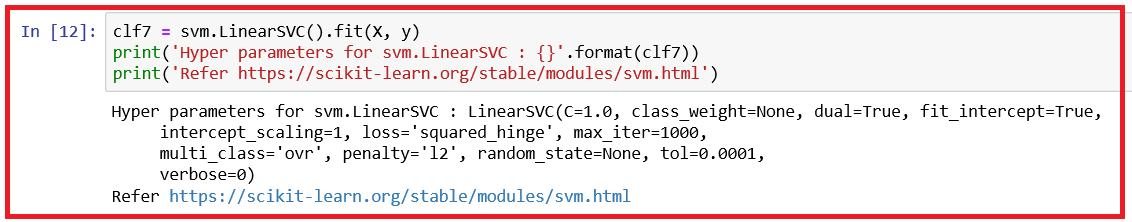
* Get the hyper-parameters for Random Forest Classifier Model



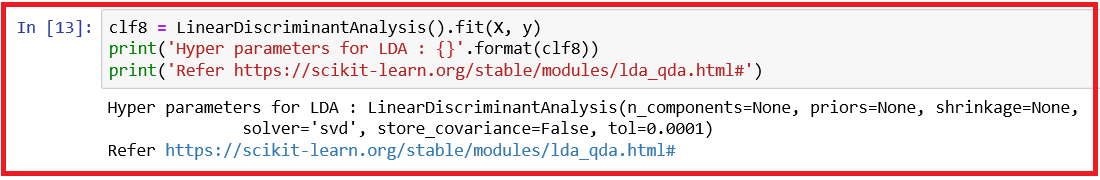
* Get the hyper-parameters for Support Vector Classifier Model



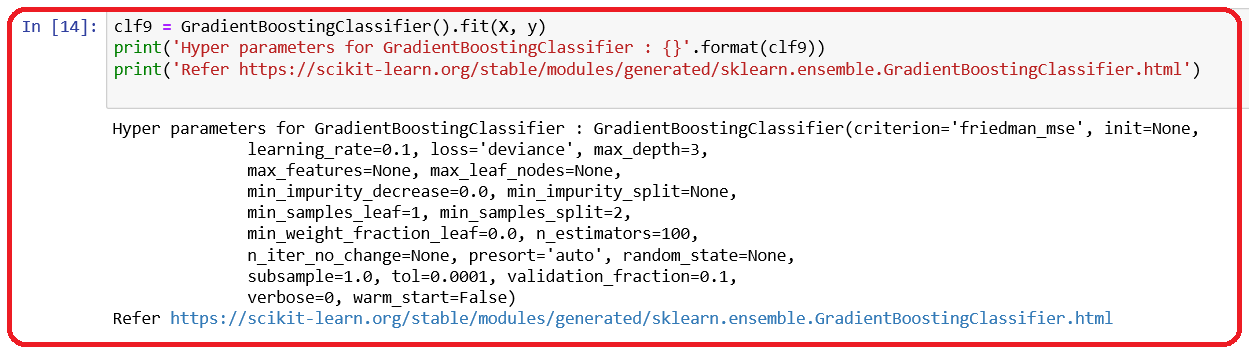
* Get the hyper-parameter for Support Vector Linear Classifier Model



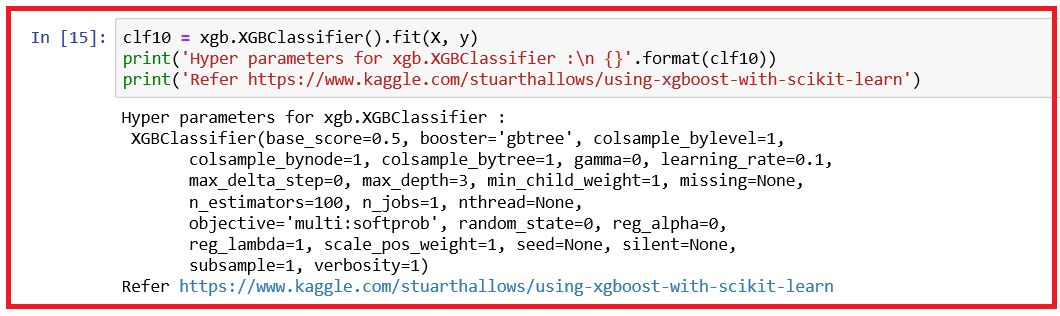
* Get the hyper-parameter for Linear Discriminant Analysis Model



* Get the hyper-parameter for Gradient Boosting Classifier Model



* Get the hyper-parameter for eXtreme Gradient Boosting Classifier Model





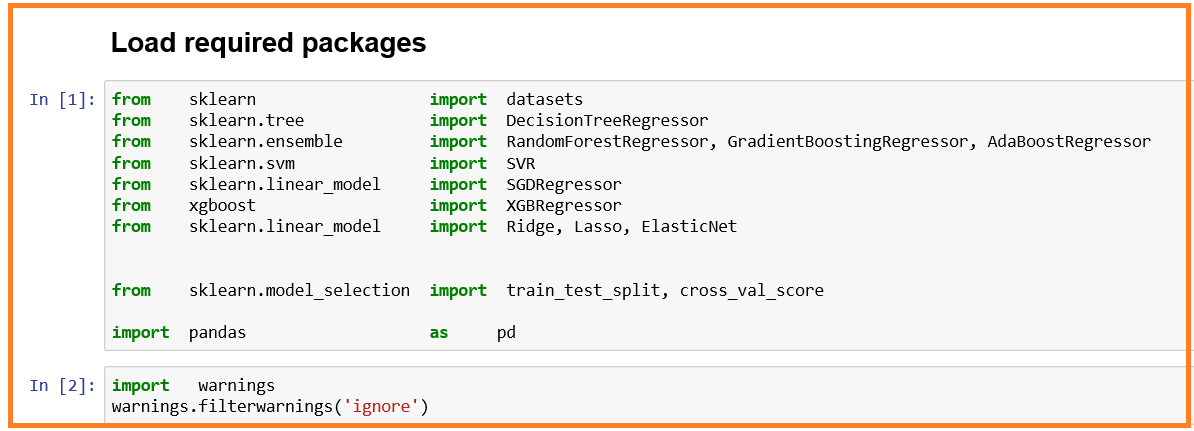
## Important hyper-parameters for classification models

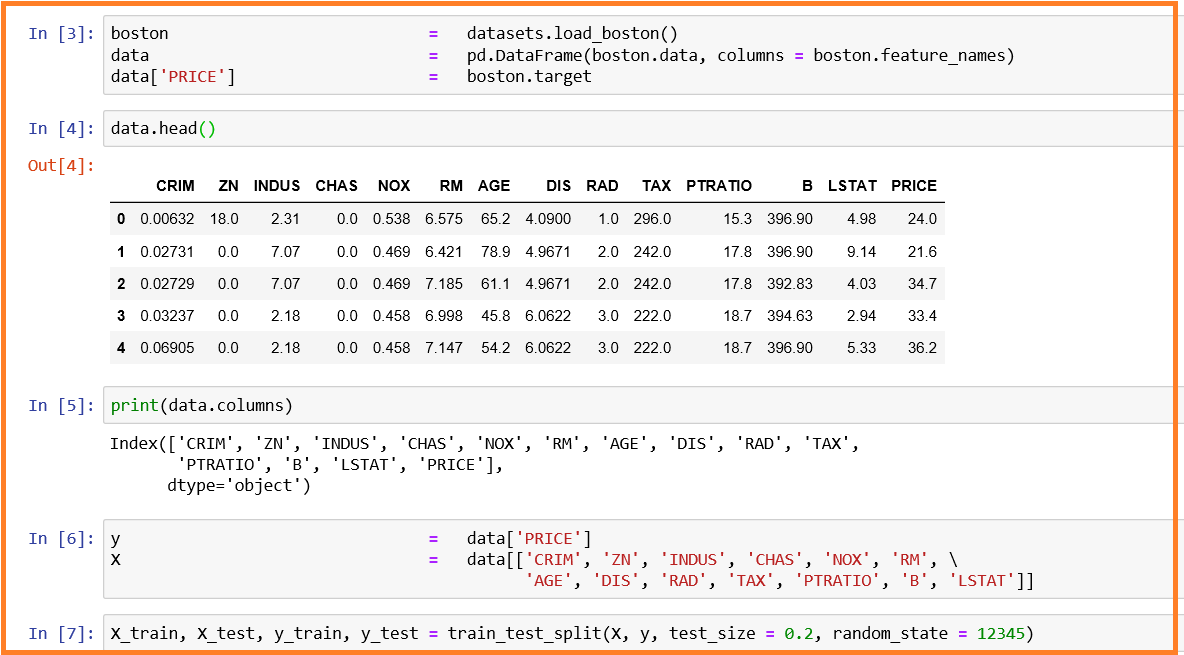
| **Machine Learning Model** | **Important Hyper-parameters** |
| --- | --- |
| 1. Logistic Regression | 1. C describes the inverse of regularization strength. 2. fit\_intercept specifies if a constant (a.k.a bias or intercept) should be added to the decision function 3. class\_weight specifies the weights associated with classes 4. penalty specifies the norm such as l1 or l2 or elasticnet is used in penalization 5. solver specifies the algorithm such as liblinear or sag or newton-cg to use in the optimization problem 6. tol specifies the tolerance for stopping criteria |
| 1. Naïve Bayes | 1. priors describes the prior probabilities of the classes 2. var\_smoothing specifies portion of the largest variance of all features that is added to variances for calculation stability |
| 1. Stochastic Gradient Model | 1. **alpha** specifies the constant that multiplies the regularization term. It defaults to 0.0001. This is also used to compute learning\_rate when set to optimal. 2. **average** when set to True, it computes the averaged SGD weights and stores the result in the coef\_ attribute. If it is set to an integer greater than 1, averaging will begin once the total number of samples seen reaches value set for average. So, average = 10 will begin averaging after seeing 10 samples. 3. **class\_weight** denotes the weights associate with classes. The 'balanced' mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y)) 4. **epsilon** denotes the value in the epsilon-insensitive loss function for epsilon.  * When loss function = huber, epsilon determines the threshold at which it becomes less important to get the prediction exactly right. * When loss function = epsilon\_insenstive or squared\_epsilon\_insensitive, any differences between the current prediction and the current label are ignored if they are less than this threshold.  1. **fit\_intercept** denotes whether the intercept should be estimated or not. If it is set to False, the data is assumed to be already centered. 2. **l1\_ratio** is the Elastic Net mixing parameter, with 0 <= l1\_ratio <= 1. When l1\_ratio = 0, it corresponds to L2 penalty. When l1\_ratio = 1, it corresponds to L1 penalty. Default value is 0.15. 3. **learning\_rate** denotes the learning rate shedule. Valid values: 4. **constant**: eta = eta0 (initial learning rate for the constant, invscaling, adaptive schedules) 5. **optimal**: eta = 1.0 / (alpha \* (t = t0)), where t0 is chosen by heuristic proposed by Leon Bottou. 6. **invscaling**: eta = eta0 / pow(t, power\_t), where t is the learning rate 7. **adaptive**: eta = eta0, as long as the training keeps decreasing. Each time n\_iter\_no\_change consecutive epoches fail to decrease the training loss by tol or fail to increase validation score by tol if early\_stopping is True, the current learning rate is divided by 5. 8. **loss** specifies the loss function to be used. Possible options are:   Classification loss  a. hinge gives linear SVM  b. log gives logistic regression, a probabilistic classifier  c. modified\_huber is another smooth loss that brings tolerance to outliers as well as probability estimates  d. squared\_hinge is like hinge but is quadratically penalized  e. perceptron is the linear loss used by the perceptron algorithm   1. **penalty** specifies the regularization term to be used. Possible values are l1, l2 or elasticnet or none 2. **power\_t** specifies the exponent for inverse scaling learning rate 3. **shuffle** specifies whether or not the training data should be shuffled after each epoch. Default is True. 4. **tol** specifies the stopping criterion 5. **validation\_fraction** specifies the proportion of training data to set aside as validation set for early stopping. This value must be between 0 and 1. This parameter is used only if early\_stopping is set to True. 6. **early\_stopping** specifies whether to use early stopping to terminate training when validation score is not improving   **https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html#sklearn.linear\_model.SGDClassifier** |
| 1. KNN classifier | 1. **algorithm** specifies the algorithm used to compute the nearest neighbors:   1. **ball\_tree** is a binary tree in which every node defines a D-dimensional hypersphere, or ball, containing a subset of the points to be searched. Each internal node of the tree partitions the data points into two disjoint sets which are associated with different balls. https://en.wikipedia.org/wiki/Ball\_tree  2. **kd\_tree** is a binary tree in which every leaf node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces.  https://en.wikipedia.org/wiki/K-d\_tree  3. **brute** to use a brute\_force search  4. **auto** to attempt to decide the most appropriate algorithm based on the values passed to the fit method   1. **leaf\_size** specifies the leaf size passes to the BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. 2. **Metric** specifies the distance metric to use for the tree. 3. **n\_neighbors** specifies the number of neighbors to use by default for kneighbors queries 4. **weights** specify the weight function used in prediction. Possible values:   1. uniform specifies uniform weights  2. distance specifies weight points by the inverse of their distance  3. Callable user defined function which accepts an array of distances, and returns an array of the same shape containing the weights.  *https://scikit-learn.org/stable/modules/neighbors.html* |
| 1. Decision Tree Classifier | 1. **criterion** specifies the function to measure the quality of split. Available options**: gini** for Gini impurity and **entropy** for the information gain 2. **splitter** specifies the strategy used to choose the split at each node, either 'best' to choose the best split or 'random' to choose the the best random split. 3. **max\_depth** denotes the maximum depth of the tree 4. **min\_samples\_split** to specify the ninimum number of samples required to split an internal node 5. **min\_samples\_leaf** to specify the minimum number of samples required to be at a leaf node 6. **max\_features** specifies the number of features to consider when looking for the best split   *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html* |
| 1. Random Forest Classifier | 1. **bootsrap** specifies whether bootstrap samples are used when building trees. If false, the whole dataset is used to build each tree. 2. **class\_weight** specifies the weights associated with classes in the form {class\_label : Weight} 3. n\_estimators specifies the number of trees in the forest 4. **criterion** specifies the function such as gini or entropy to measure the quality of split 5. **max\_depth** denotes the maximum depth of the tree 6. **min\_samples\_split** specifies the minimum number of samples required to split an internal node 7. **min\_samples\_leaf** specifies the minimum number of samples required to be at a leaf node 8. **max\_features** denotes the number of features to consider when looking for the best split   *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html* |
| 1. Support Vector Classifier | 1. **C denotes the Penalty parameter C of the error term.** 2. **kernel specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable.** 3. **degree specifies the degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.** 4. **gamma denotes the Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.** 5. **coef0 specifies the independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.** 6. **shrinking specifies whether to use the shrinking heuristic.** 7. **probability specifies whether to enable probability estimates. This must be enabled prior to calling fit, and will slow down that method.** 8. **tol denotes the tolerance for stopping criterion.**   *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html* |
| 1. Support Vector Linear Classifier | 1. **penalty** specifies the norm used in the penalization. The ‘l2’ penalty is the standard used in SVC. The ‘l1’ leads to coef\_ vectors that are sparse. 2. **loss** specifies the loss function. ‘hinge’ is the standard SVM loss (used e.g. by the SVC class) while ‘squared\_hinge’ is the square of the hinge loss. 3. **dual** specifies whether the algorithm is selected to either solve the dual or primal optimization problem. 4. **tol** specifies tolerance for stopping criteria. 5. **C**  specifies the penalty parameter C of the error term. 6. **multi\_class** determines the multi-class strategy if y contains more than two classes. "ovr" trains n\_classes one-vs-rest classifiers, while "crammer\_singer" optimizes a joint objective over all classes. While crammer\_singer is interesting from a theoretical perspective as it is consistent, it is seldom used in practice as it rarely leads to better accuracy and is more expensive to compute. If "crammer\_singer" is chosen, the options loss, penalty and dual will be ignored. 7. **fit\_intercept specifies whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (i.e. data is expected to be already centered).** 8. **class\_weight sets the parameter C of class i to class\_weight[i]\*C for SVC. If not given, all classes are supposed to have weight one. The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y))**   https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html |
| 1. Linear Discriminant Classifier | 1. **solver** specifies the solver to use, possible values: ‘svd’: Singular value decomposition (default). Does not compute the covariance matrix, therefore this solver is recommended for data with a large number of features. ‘lsqr’: Least squares solution, can be combined with shrinkage. ‘eigen’: Eigenvalue decomposition, can be combined with shrinkage. 2. **shrinkage** specifies the shrinkage parameter, possible values: None: no shrinkage (default). ‘auto’: automatic shrinkage using the Ledoit-Wolf lemma. float between 0 and 1: fixed shrinkage parameter. Note that shrinkage works only with ‘lsqr’ and ‘eigen’ solvers. 3. **priors** : specifies the class priors. 4. **n\_components** denotes the number of components (<= min(n\_classes - 1, n\_features)) for dimensionality reduction. If None, will be set to min(n\_classes - 1, n\_features). 5. **tol** denotes the threshold used for rank estimation in SVD solver.   https://scikit-learn.org/stable/modules/generated/sklearn.discriminant\_analysis.LinearDiscriminantAnalysis.html |
| 1. Gradient Boosting Classifier | 1. **loss** specifies the loss function to be optimized. {‘deviance’, ‘exponential’}, optional (default=’deviance’); ‘deviance’ refers to deviance (= logistic regression) for classification with probabilistic outputs. For loss ‘exponential’ gradient boosting recovers the AdaBoost algorithm. 2. **learning\_rate** shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators. 3. **n\_estimators** denotes the number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance. 4. **criterion** specifies the function to measure the quality of a split. Supported criteria are “friedman\_mse” for the mean squared error with improvement score by Friedman, “mse” for mean squared error, and “mae” for the mean absolute error. The default value of “friedman\_mse” is generally the best as it can provide a better approximation in some cases. 5. **min\_samples\_split** denotes the minimum number of samples required to split an internal node: If int, then consider min\_samples\_split as the minimum number. If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split. 6. **min\_samples\_leaf** denotes the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. If int, then consider min\_samples\_leaf as the minimum number. If float, then min\_samples\_leaf is a fraction and ceil(min\_samples\_leaf \* n\_samples) are the minimum number of samples for each node. 7. **max\_depth** denotes the maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables. 8. **min\_impurity\_decrease** specifies a node will be split if this split induces a decrease of the impurity greater than or equal to this value. 9. **init** specifies an estimator object that is used to compute the initial predictions. init has to provide fit and predict\_proba. If ‘zero’, the initial raw predictions are set to zero. By default, a DummyEstimator predicting the classes priors is used. 10. **max\_features** denotes the number of features to consider when looking for the best split 11. **max\_leaf\_nodes** to Grow trees with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. 12. **presort** specifies whether to presort the data to speed up the finding of best splits in fitting. Auto mode by default will use presorting on dense data and default to normal sorting on sparse data. Setting presort to true on sparse data will raise an error. 13. **tol** specifies the tolerance for the early stopping. When the loss is not improving by at least tol for n\_iter\_no\_change iterations (if set to a number), the training stops.   https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html |
| 1. Xtreme Gradient Boosting Classifier | 1. **max\_depth** (int) denotes the Maximum tree depth for base learners. 2. **learning\_rate** (float) denotes Boosting learning rate (xgb’s “eta”) 3. **n\_estimators** (int) denotes Number of trees to fit. 4. **objective** (string or callable) specifies the learning task and the corresponding learning objective or a custom objective function to be used 5. **booster** (string) – Specify which booster to use: gbtree, gblinear or dart. 6. **n\_jobs** (int) denotes the number of parallel threads used to run xgboost. (replaces nthread) 7. **gamma** (float) denotes the Minimum loss reduction required to make a further partition on a leaf node of the tree. 8. **min\_child\_weight** (int) denotes the Minimum sum of instance weight(hessian) needed in a child. 9. **max\_delta\_step** (int) denotes the Maximum delta step we allow each tree’s weight estimation to be. 10. **missing** (float, optional) denotes the value in the data which needs to be present as a missing value. If None, defaults to np.nan. 11. **importance\_type** (string, default "gain") specifies the feature importance type for the feature\_importances\_ property: either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.   *https://xgboost.readthedocs.io/en/latest/python/python\_api.html#module-xgboost.sklearn* |



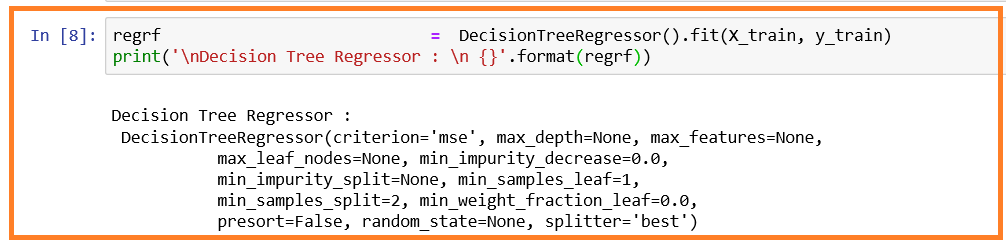
# HYPER PARAMETERS – REGRESSION MODELS

* How do you find the hyper-parameters of popular regression models?
* We shall apply several linear and non-linear models to get the model parameters.
* We shall use the Boston Housing Dataset containing the prices of houses in various places in Boston. Apart from the response variable, PRICE, the dataset also provide information such as Crime (CRIM), areas of non-retail business in the town (INDUS), the age of people who own the house (AGE), and there are many other attributes that available. For more details, refer the Source: <https://archive.ics.uci.edu/ml/datasets/Housing>

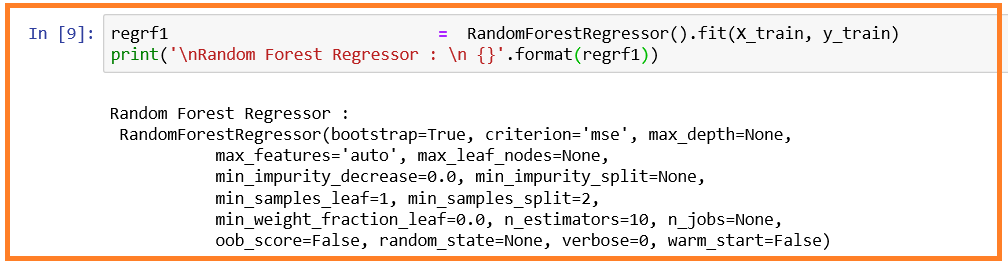




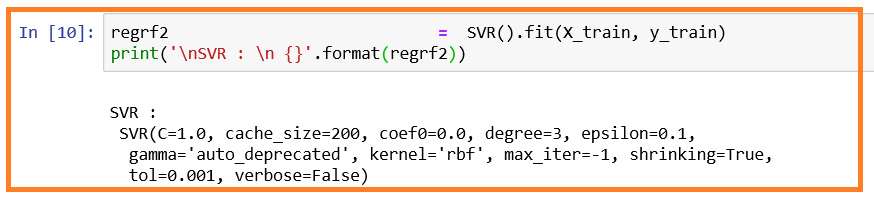
* Decision Tree Regressor



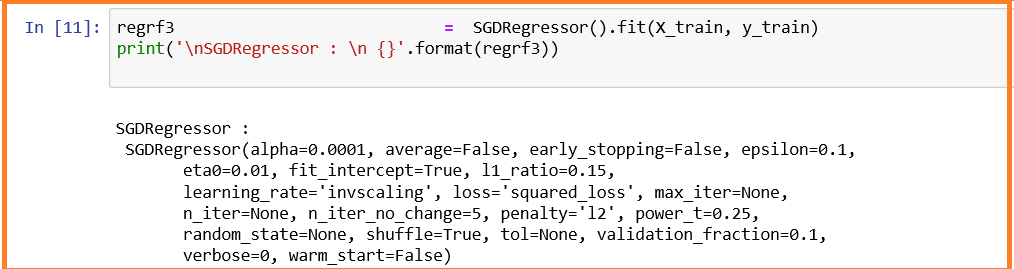
* Random Forest Regressor



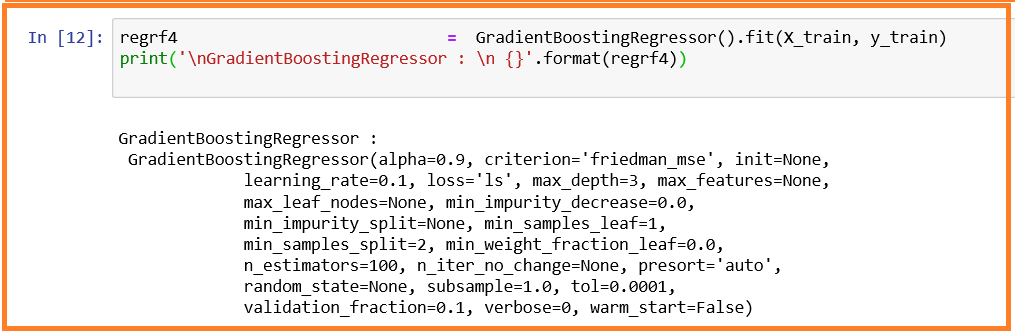
* Support Vector Regressor



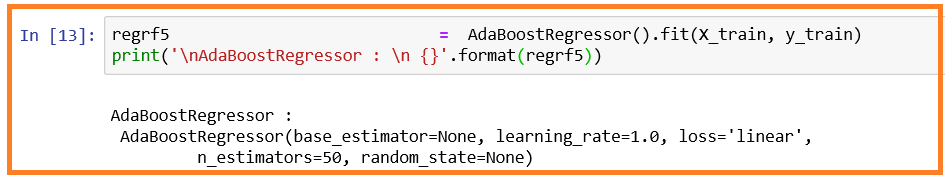
* Stochastic Gradient Regressor



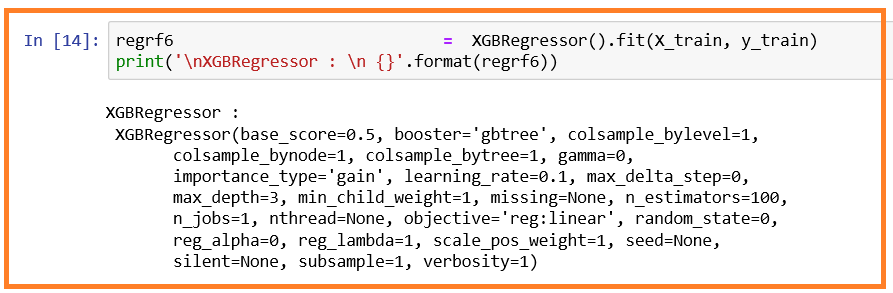
* Gradient Boosting Regressor



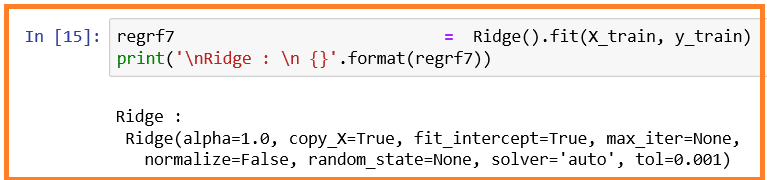
* AdaBoost Regressor



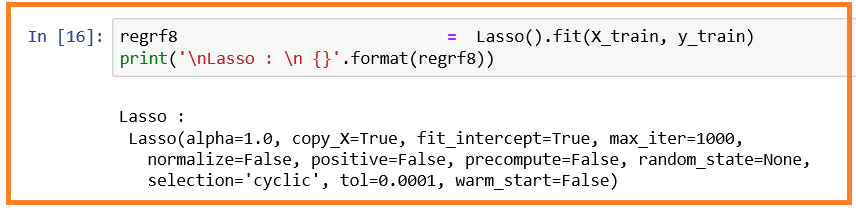
* XG Boosting Regressor



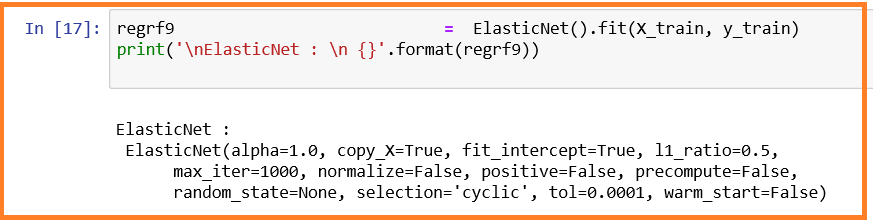
* Ridge Linear Regression



* Lasso Linear Regression



* ElasticNet Linear Regression





## Important hyper-parameters for REGRESSION models

| **Machine Learning Model** | **Important Hyper-parameters** |
| --- | --- |
| Decision Tree Regressor | 1. **criterion** specifies the function to measure the quality of a split. Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node, “friedman\_mse”, which uses mean squared error with Friedman’s improvement score for potential splits, and “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node. 2. **splitter** specifies the strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split. 3. **max\_depth** specifies the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples. 4. **min\_samples\_split** specifies the minimum number of samples required to split an internal node: 5. **min\_samples\_leaf** specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. 6. **max\_features** specifies the number of features to consider when looking for the best split 7. **max\_leaf\_nodes**: Grow a tree with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes. 8. **min\_impurity\_split** specifies threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.   *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html* |

| **Machine Learning Model** | **Important Hyper-parameters** |
| --- | --- |
| Random Forest Regressor | 1. **n\_estimator** specifies the number of trees in the forest. 2. **criterion** specifies the function to measure the quality of a split. Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion, and “mae” for the mean absolute error. 3. **max\_depth** specifies the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples. 4. **min\_samples\_split** specifies the minimum number of samples required to split an internal node: 5. **min\_samples\_leaf** specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. 6. **max\_features** specifies the number of features to consider when looking for the best split: 7. **max\_leaf\_nodes** : Grow trees with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes. 8. **bootstrap** specifies whether bootstrap samples are used when building trees. If False, the whole datset is used to build each tree. 9. **oob\_score** specifies whether to use out-of-bag samples to estimate the R^2 on unseen data. 10. **n\_jobs** specifies the number of jobs to run in parallel for both fit and predict.   https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html |
| Support Vector Regressor | 1. **kernel** specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. If a callable is given it is used to precompute the kernel matrix. 2. **degree** indicates the degree of the polynomial kernel function (‘poly’). Ignored by all other kernels. 3. **gamma** specifies the Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.   Current default is ‘auto’ which uses 1 / n\_features, if gamma='scale' is passed then it uses 1 / (n\_features \* X.var()) as value of gamma. The current default of gamma, ‘auto’, will change to ‘scale’ in version 0.22. ‘auto\_deprecated’, a deprecated version of ‘auto’ is used as a default indicating that no explicit value of gamma was passed.   1. **coef0** specifies the Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’. 2. **tol** specifies the tolerance for stopping criterion. 3. **C** specifies the penalty parameter C of the error term. 4. **epsilon** specifies the in the epsilon-SVR model. It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. 5. **shrinking** specifies whether to use the shrinking heuristic. 6. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html |
| Stochastic Gradient Regressor | 1. **loss** specifies the loss function to be used. The possible values are ‘squared\_loss’, ‘huber’, ‘epsilon\_insensitive’, or ‘squared\_epsilon\_insensitive’   The ‘squared\_loss’ refers to the ordinary least squares fit. ‘huber’ modifies ‘squared\_loss’ to focus less on getting outliers correct by switching from squared to linear loss past a distance of epsilon. ‘epsilon\_insensitive’ ignores errors less than epsilon and is linear past that; this is the loss function used in SVR. ‘squared\_epsilon\_insensitive’ is the same but becomes squared loss past a tolerance of epsilon.   1. **penalty** specifies the penalty (aka regularization term) to be used. Defaults to ‘l2’ which is the standard regularizer for linear SVM models. ‘l1’ and ‘elasticnet’ might bring sparsity to the model (feature selection) not achievable with ‘l2’. Possible values: str, ‘none’, ‘l2’, ‘l1’, or ‘elasticnet’ 2. **alpha** specifies a constant that multiplies the regularization term. Defaults to 0.0001 Also used to compute learning\_rate when set to ‘optimal’. 3. **l1\_ratio** specifies the Elastic Net mixing parameter, with 0 <= l1\_ratio <= 1. l1\_ratio=0 corresponds to L2 penalty, l1\_ratio=1 to L1. Defaults to 0.15. 4. **fit\_intercept** specifies whether the intercept should be estimated or not. If False, the data is assumed to be already centered. Defaults to True. 5. **max\_iter** specifies the maximum number of passes over the training data (aka epochs). It only impacts the behavior in the fit method, and not the partial\_fit. 6. **tol** specifies the stopping criterion. If it is not None, the iterations will stop when (loss > best\_loss - tol) for n\_iter\_no\_change consecutive epochs. 7. **shuffle** specifies whether or not the training data should be shuffled after each epoch. Defaults to True. 8. **epsilon** specifies the Epsilon in the epsilon-insensitive loss functions; only if loss is ‘huber’, ‘epsilon\_insensitive’, or ‘squared\_epsilon\_insensitive’. For ‘huber’, determines the threshold at which it becomes less important to get the prediction exactly right. For epsilon-insensitive, any differences between the current prediction and the correct label are ignored if they are less than this threshold. 9. **learning\_rate** specifies the learning rate schedule:  * ‘constant’: eta = eta0 * ‘optimal’: eta = 1.0 / (alpha \* (t + t0)) where t0 is chosen by a heuristic proposed by Leon Bottou. * ‘invscaling’: [default] eta = eta0 / pow(t, power\_t) * ‘adaptive’: eta = eta0, as long as the training keeps decreasing. Each time n\_iter\_no\_change consecutive epochs fail to decrease the training loss by tol or fail to increase validation score by tol if early\_stopping is True, the current learning rate is divided by 5.  1. **eta0** specifies the initial learning rate for the ‘constant’, ‘invscaling’ or ‘adaptive’ schedules. The default value is 0.01. 2. **power\_t** specifies the exponent for inverse scaling learning rate [default 0.5]. 3. **early\_stopping** indicates whether to use early stopping to terminate training when validation score is not improving. If set to True, it will automatically set aside a fraction of training data as validation and terminate training when validation score is not improving by at least tol for n\_iter\_no\_change consecutive epochs. 4. **average** when set to True, computes the averaged SGD weights and stores the result in the coef\_ attribute. If set to an int greater than 1, averaging will begin once the total number of samples seen reaches average. So average=10 will begin averaging after seeing 10 samples.   https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDRegressor.html#sklearn.linear\_model.SGDRegressor |
| Gradient Boosting Regressor | 1. **loss** *specifies loss function to be optimized. Possible options are: {‘ls’, ‘lad’, ‘huber’, ‘quantile’}, optional (default=’ls’)*   *Note: ‘ls’ refers to least squares regression. ‘lad’ (least absolute deviation) is a highly robust loss function solely based on order information of the input variables. ‘huber’ is a combination of the two. ‘quantile’ allows quantile regression (use alpha to specify the quantile).*   1. **learning\_rate** *shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.* 2. **n\_estimators** *indicates the number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.* 3. **criterion** *specifies the function to measure the quality of a split. Supported criteria are “friedman\_mse” for the mean squared error with improvement score by Friedman, “mse” for mean squared error, and “mae” for the mean absolute error. The default value of “friedman\_mse” is generally the best as it can provide a better approximation in some cases.* 4. **min\_samples\_split** *specifies the minimum number of samples required to split an internal node:* 5. **min\_samples\_leaf** *specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.* 6. **max\_depth** *specifies the maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.* 7. **max\_features** *specifies the number of features to consider when looking for the best split:* 8. **alpha** *specifies the alpha-quantile of the huber loss function and the quantile loss function. Only if loss='huber' or loss='quantile'.* 9. **max\_leaf\_nodes** *denotes the value to grow trees in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.* 10. **tol** *specifies the tolerance for the early stopping. When the loss is not improving by at least tol for n\_iter\_no\_change iterations (if set to a number), the training stops.*   *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html* |
| AdaBoost Regressor | 1. **base\_estimator** specifies the base estimator from which the boosted ensemble is built. Support for sample weighting is required. 2. **n\_estimators** denotes the maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. 3. **learning\_rate** shrinks the contribution of each regressor by learning\_rate. There is a trade-off between learning\_rate and n\_estimators. 4. **loss** specifies the loss function to use when updating the weights after each boosting iteration. Possible values are: {‘linear’, ‘square’, ‘exponential’}, optional (default=’linear’)   *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html* |
| XG Boosting Regressor | 1. **max\_depth** specifies the maximum tree depth for base learners. 2. **learning\_rate** specifies the boosting learning rate (xgb’s “eta”) 3. **n\_estimators** denotes the number of trees to fit. 4. **booster** specifies which booster to use: gbtree, gblinear or dart. 5. **n\_jobs** specifies the number of parallel threads used to run xgboost. 6. **gamma** specifies the minimum loss reduction required to make a further partition on a leaf node of the tree. 7. **missing** indicates the value in the data which needs to be present as a missing value. If None, defaults to np.nan. 8. **importance\_type** specifies the feature importance type for the feature\_importances\_ property: either “gain”, “weight”, “cover”, “total\_gain” or “total\_cover”.   https://xgboost.readthedocs.io/en/latest/python/python\_api.html |
| Ridge Linear Regression | 1. **alpha** specifies the regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to C^-1 in other linear models such as LogisticRegression or LinearSVC. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number. 2. **fit\_intercept** specifies whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (e.g. data is expected to be already centered). 3. **normalize** specifies whether regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm 4. **max\_iter** denotes the maximum number of iterations for conjugate gradient solver. For ‘sparse\_cg’ and ‘lsqr’ solvers, the default value is determined by scipy.sparse.linalg. For ‘sag’ solver, the default value is 1000. 5. **tol** denotes the precision of the solution. 6. **solver** specifies Solver to use in the computational routines:   Possible values are {‘auto’, ‘svd’, ‘cholesky’, ‘lsqr’, ‘sparse\_cg’, ‘sag’, ‘saga’}   * ‘auto’ chooses the solver automatically based on the type of data. * ‘svd’ uses a Singular Value Decomposition of X to compute the Ridge coefficients. More stable for singular matrices than ‘cholesky’. * ‘cholesky’ uses the standard scipy.linalg.solve function to obtain a closed-form solution. * ‘sparse\_cg’ uses the conjugate gradient solver as found in scipy.sparse.linalg.cg. As an iterative algorithm, this solver is more appropriate than ‘cholesky’ for large-scale data (possibility to set tol and max\_iter). * ‘lsqr’ uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative procedure. * ‘sag’ uses a Stochastic Average Gradient descent, and ‘saga’ uses its improved, unbiased version named SAGA. Both methods also use an iterative procedure, and are often faster than other solvers when both n\_samples and n\_features are large. Note that ‘sag’ and ‘saga’ fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing. * All last five solvers support both dense and sparse data. However, only ‘sag’ and ‘sparse\_cg’ supports sparse input when fit\_intercept is True.   https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.Ridge.html#sklearn.linear\_model.Ridge |
| Lasso Linear Regression | 1. **alpha** denotes a constant that multiplies the L1 term. Defaults to 1.0. alpha = 0 is equivalent to an ordinary least square, solved by the LinearRegression object. 2. **fit\_intercept** specifies whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered). 3. **normalize** indicates the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. 4. **max\_iter** denotes the maximum number of iterations 5. **tol** specifies the tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol. 6. **positive** forces the coefficients to be positive, when set to True   https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.Lasso.html#sklearn.linear\_model.Lasso |
| ElasticNet Linear Regression | 1. **alpha** denotes a constant that multiplies the penalty terms. 2. **l1\_ratio** specifies the ElasticNet mixing parameter, with 0 <= l1\_ratio <= 1. For l1\_ratio = 0 the penalty is an L2 penalty. For l1\_ratio = 1 it is an L1 penalty. For 0 < l1\_ratio < 1, the penalty is a combination of L1 and L2. 3. **fit\_intercept** specifies whether the intercept should be estimated or not. If False, the data is assumed to be already centered. 4. **tol** denotes the tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol. 5. **positive** When set to True, forces the coefficients to be positive. 6. **selection** When set to ‘random’, a random coefficient is updated every iteration rather than looping over features sequentially by default.   https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.ElasticNet.html#sklearn.linear\_model.ElasticNet |