Bootstrap Assignmnet

There will be some functions that start with the word "grader" ex: grader_sampples(), grader_30().. etc, we should not change those function definition. Every Grader function has to return True.

In [1]:
```
import numpy as np # importing numpy for numerical computation
from sklearn.datasets import load_boston # here we are using sklearn's boston data
from sklearn.metrics import mean_squared_error # importing mean_squared_error met

import random
from sklearn.tree import DecisionTreeRegressor

boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable

print("x.shape ", x.shape)
print("y.shape ", y.shape)
```

```
x.shape  (506, 13)
y.shape  (506,)
```

Task 1

- **Creating samples**
  Randomly create 30 samples from the whole boston data points
  - Creating each sample: Consider any random 303(60% of 506) data points from whole data set and then replicate any 203 points from the sampled points

    For better understanding of this procedure lets check this examples, assume we have 10 data points [1,2,3,4,5,6,7,8,9,10], first we take 6 data points randomly , consider we have selected [4, 5, 7, 8, 9, 3] now we will replicate 4 points from [4, 5, 7, 8, 9, 3], consder they are [5, 8, 3,7] so our final sample will be [4, 5, 7, 8, 9, 3, 5, 8, 3,7]
- **Create 30 samples**
  - Note that as a part of the Bagging when we are taking the random samples **make sure each of the sample will have different set of columns**
    Ex: Assume we have 10 columns[1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10] for the first sample we will select [3, 4, 5, 9, 1, 2] and for the second sample [7, 9, 1, 4, 5, 6, 2] and so on... Make sure each sample will have atleast 3 feautres/columns/attributes

Step - 1 - Creating samples

Algorithm

Pesudo Code for generating Sample

```
def generating_samples(input_data, target_data):

    Selecting_rows <--- Getting 303 random row indices from the input_data

    Replcaing_rows <--- Extracting 206 random row indices from the "Selecting_rows"

    Selecting_columns<--- Getting from 3 to 13 random column indices

    sample_data<--- input_data[Selecting_rows[:,None],Selecting_columns]

    target_of_sample_data <--- target_data[Selecting_rows]

    #Replicating Data

    Replicated_sample_data <--- sample_data [Replaceing_rows]

    target_of_Replicated_sample_data<--- target_data[Replaceing_rows]

    # Concatinating data

    final_sample_data <---  perform vertical stack on  sample_data, Replicated_sample_data

    final_target_data<--- perform vertical stack on target_of_sample_data.reshape(-1,1), target_of_Replicated_sample_data.reshape(-1,1)

    return final_sample_data,  final_target_data, Selecting_rows, Selecting_columns
```

Generating samples

In [2]:
```
def generating_samples(input_data, target_data):

  # Here we are finding the random 303 row indices without replacement as shown i
  selected_rows = np.random.choice(len(input_data), 303, replace=False)

  #selecting 203 more row indices from the selected rows.
  get_203_from_selected_rows = np.random.choice(selected_rows, 203, replace=False

  # Now get 3 to 13 random column indices from input_data
  random_selected_columns = random.randint(3, 13)
  columns_selected = np.array(random.sample(range(0, 13), random_selected_columns
  #Taken from:: https://stackoverflow.com/questions/22927181/selecting-specific-r
  sample_data = input_data[selected_rows[:, None], columns_selected]

  target_of_sample_data = target_data[selected_rows]

  # Now Replication of Data for 203 data points out of 303 selected points

  replicated_203_sample_data_points = input_data[get_203_from_selected_rows[:, No
#   print(get_203_from_selected_rows)
#   print(type(get_203_from_selected_rows))
#   print("this is printed ",get_203_from_selected_rows[:, None])
#   print("this is end")
#   print(columns_selected)
  target_203_replicated_sample_data = target_data[get_203_from_selected_rows]

  # Concatenating data

  final_sample_data = np.vstack((sample_data, replicated_203_sample_data_points )

  final_target_data = np.vstack((target_of_sample_data.reshape(-1, 1), target_203

  return final_sample_data, final_target_data, selected_rows, columns_selected
```

Grader function - 1

In [3]:
```python
def grader_samples(a,b,c,d):
    length = (len(a)==506  and len(b)==506)
    sampled = (len(a)-len(set([str(i) for i in a]))==203)
    rows_length = (len(c)==303)
    column_length= (len(d)>=3)
    assert(length and sampled and rows_length and column_length)
    return True

a,b,c,d = generating_samples(x, y)
print(a.shape)
print(b.shape)
print(c.shape)
print(d.shape)
grader_samples(a,b,c,d)
```

```
(506, 11)
(506, 1)
(303,)
(11,)
```

Out[3]: True

Creating 30 samples

Run this code 30 times, so that you will 30 samples, and store them in a lists as shown below:

```
list_input_data=[]
list_output_data=[]
list_selected_row=[]
list_selected_columns=[]

for i in range(0,30):
    a,b,c,d=generating_sample(input_data,target_data)
    list_input_data.append(a)
    list_output_data.append(b)
    list_selected_row.append(c)
    list_selected_columns.append(d)
```

In [4]:
```
# getting 30 sample data with their column and row details
list_input_data =[]
list_output_data =[]
list_selected_row= []
list_selected_columns=[]

for i in range (0, 30):
  a, b, c, d = generating_samples(x, y)
  list_input_data.append(a)
  list_output_data.append(b)
  list_selected_row.append(c)
  list_selected_columns.append(d)
```

Grader function - 2

In [5]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

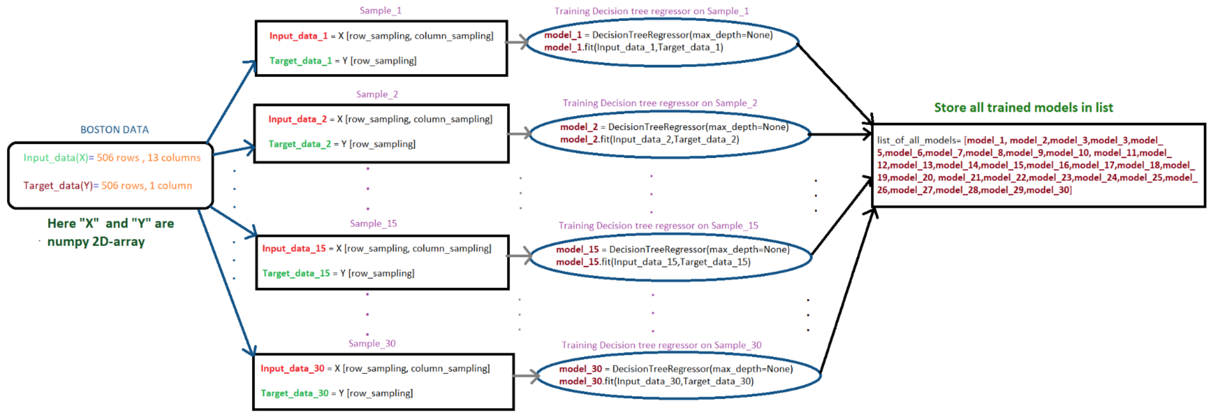
Out[5]:
```
True
```

Step - 2 of Task-1

Building High Variance Models on each of the sample and finding train MSE value

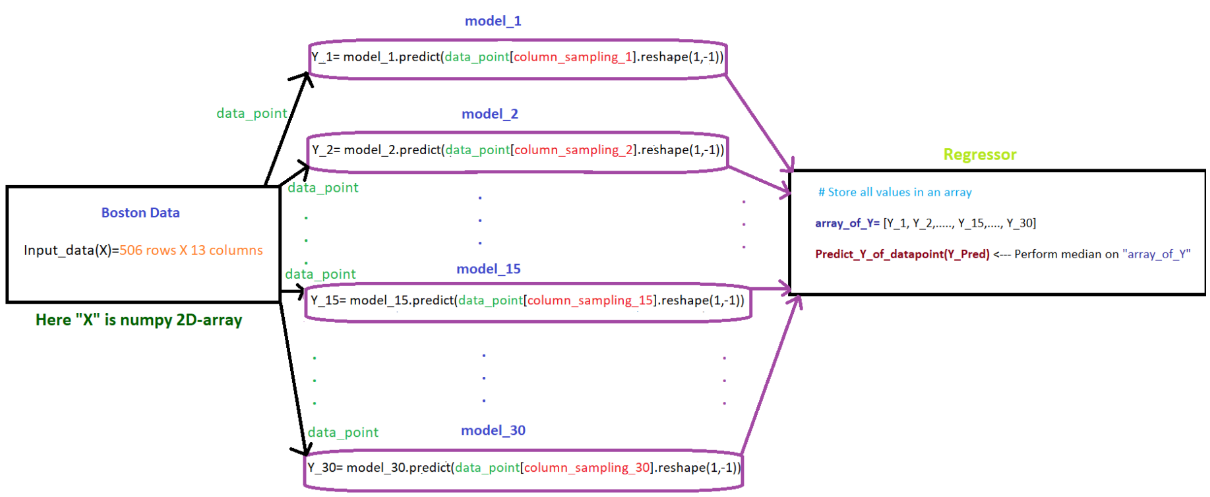
- Build a regression trees on each of 30 samples.
- Computed the predicted values of each data point(506 data points) in our corpus.
- Predicted house price of i^{th} data point
  $y^{i}_{pred} = \frac{1}{30}\sum_{k=1}^{30}(\text{predicted value of } x^{i} \text{ with } k^{th} \text{ model})$
- Now calculate the $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^{i} - y^{i}_{pred})^{2}$

Flowchart for Building regression trees



In [6]:
```python
all_model_list = []
for i in range(0, 30):
  Dtree_i = DecisionTreeRegressor()
  Dtree_i.fit(list_input_data[i], list_output_data[i])
  all_model_list.append(Dtree_i)
```

Flowchart for calculating MSE



After getting predicted_y for each data point, we can use sklearns mean_squared_error to calculate the MSE between predicted_y and actual_y.

Calculating MSE

In [7]:
```
from sklearn.metrics import mean_squared_error
from statistics import median

array_of_Y = []

for i in range(0, 30):
  data_point_i = x[:, list_selected_columns[i]]
  target_Y = all_model_list[i].predict(data_point_i)
  array_of_Y.append(target_Y)


predicted_array_of_target_y = np.array(array_of_Y)
predicted_array_of_target_y = predicted_array_of_target_y.transpose()

# Now to calculate MSE, first calculate the Median of Predicted Y
# passing axis=1 will make sure the medians are computed along axis=1
median_predicted_y = np.median(predicted_array_of_target_y, axis=1)

print("The shape of predicted median ",median_predicted_y.shape)
print("MSE : ", mean_squared_error(y, median_predicted_y))
```

```
The shape of predicted median  (506,)
MSE :  0.0631836023276241
```

Step - 3 of Task-1

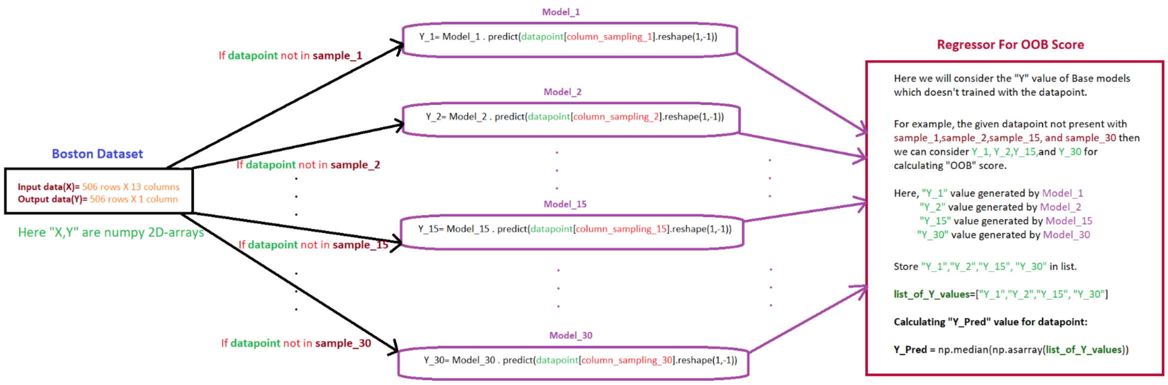
Calculating the OOB score

- Predicted house price of i^{th} data point
  $y^i_{pred} = \frac{1}{k} \sum_{\text{k= model which was buit on samples not included } x^i} \text{(predicted value of } x^i \text{ with } k^{th} \text{ model)}.$
- Now calculate the $OOBScore = \frac{1}{506} \sum_{i=1}^{506} (y^i - y^i_{pred})^2$.

Given a single query point predict the price of house

Consider xq= [0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60] Predict the house price for this point as mentioned in the step 2 of Task 1.

Flowchart for calculating OOB score



Now calculate the $OOBScore = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$.

In [8]:

```
y_predicted_oob_median_list = []

for i in range(0, 506):
  indices_for_oob_models = []

  # For each of i-th row I shall build a list, of sample size 30
  # ONLY condition being that this i-th row should not be part of the list_selecte
  # e.g. say for i = 469 and index_oob in below loop is 10 then
  # list_selected_row[10] (which is an array of row-numbers) should not contain t
  for index_oob in range(0, 30):
    if i not in list_selected_row[index_oob]:
      indices_for_oob_models.append(index_oob)

  y_predicted_oob_list = []

  for oob_Dtree_index in indices_for_oob_models:
    model_oob = all_model_list[oob_Dtree_index]

    row_oob = x[i]
    # print('oob_Dtree_index ', oob_Dtree_index)

    # Now extract ONLY those specific columns/featues that were selected during t
    x_oob_data_point = [row_oob[columns] for columns in list_selected_columns[oob
    # print('np.array(x_oob_data_point) ', np.array(x_oob_data_point))
    x_oob_data_point = np.array(x_oob_data_point).reshape(1, -1)

    y_predicted_oob_data_point = model_oob.predict(x_oob_data_point)
    y_predicted_oob_list.append(y_predicted_oob_data_point)
    #
  y_predicted_oob_list = np.array(y_predicted_oob_list)

  y_predicted_median = np.median(y_predicted_oob_list)
  y_predicted_oob_median_list.append(y_predicted_median)


def calculate_oob_score(num_rows):
  oob_score = 0
  for i in range(0, num_rows):
    oob_score += ((y[i] - y_predicted_oob_median_list[i] ) ** 2)
  final_oob_score = oob_score/506
  return final_oob_score

print("final_oob_score is ", calculate_oob_score(506))
```

```
final_oob_score is  17.738450576416326
```

Further notes on above OOB calculation

The key point is that the OOB sample rows were passed through every Decition Treee that did not contain those specific OOB sample rows during the bootstrapping of training data.

OOB error is simply the error on samples that were not seen during training.

OOB Scoring is very useful when I dont have a large dataset and thereby if I split that dataset into training and validation set - will result in loss of useful data that otherwise could have been used for training the models. Hence in this case, we decide to extract some of the training data as the validation set by using only those data-points that were not used for training a particular sample-set.

Task 2

- Computing CI of OOB Score and Train MSE
- Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB score
- After this we will have 35 Train MSE values and 35 OOB scores
- using these 35 values (assume like a sample) find the confidence intravels of MSE and OOB Score
- we need to report CI of MSE and CI of OOB Score
- Note: Refer the Central_Limit_theorem.ipynb to check how to find the confidence intravel

In [9]:
```
# Function to build the entire bootstrapping steps that we did above and
# Reurning from the function the MSE and oob score
def bootstrapping_and_oob(x, y):

  # Use generating_samples function to create 30 samples
  # store these created samples in a list
  list_input_data =[]
  list_output_data =[]
  list_selected_row= []
  list_selected_columns=[]

  for i in range (0, 30):
    a, b, c, d = generating_samples(x, y)
    list_input_data.append(a)
    list_output_data.append(b)
    list_selected_row.append(c)
    list_selected_columns.append(d)

  # building regression trees
  all_model_list = []
  for i in range(0, 30):
    Dtree_i = DecisionTreeRegressor(max_depth=None)
    Dtree_i.fit(list_input_data[i], list_output_data[i])
    all_model_list.append(Dtree_i)

  # calculating MSE
  array_of_Y = []

  for i in range(0, 30):
    data_point_i = x[:, list_selected_columns[i]]
    target_Y = all_model_list[i].predict(data_point_i)
    array_of_Y.append(target_Y)


  predicted_array_of_target_y = np.array(array_of_Y)
  predicted_array_of_target_y = predicted_array_of_target_y.transpose()

  # print(predicted_array_of_target_y.shape)

  # Now to calculate MSE, first calculate the Median of Predicted Y
  # passing axis=1 will make sure the medians are computed along axis=1
  median_predicted_y = np.median(predicted_array_of_target_y, axis=1)

  # And now the final MSE
  MSE = mean_squared_error(y, median_predicted_y )

  # Calculating OOB Score
  y_predicted_oob_median_list = []

  for i in range(0, 506):
    indices_for_oob_models = []

    # For each of i-th row I shall build a list of sample size 30
    # ONLY condition being that this ith row should not be part of
    # the list_selected_row
    for index_oob in range(0, 30):
```

```
      if i not in list_selected_row[index_oob]:
        indices_for_oob_models.append(index_oob)

    y_predicted_oob_list = []

    for oob_Dtree_index in indices_for_oob_models:
      model_oob = all_model_list[oob_Dtree_index]

      row_oob = x[i]
      # print('oob_Dtree_index ', oob_Dtree_index)

      x_oob_data_point = [row_oob[col] for col in list_selected_columns[oob_Dtree_
      # print('np.array(x_oob_data_point) ', np.array(x_oob_data_point))
      x_oob_data_point = np.array(x_oob_data_point).reshape(1, -1)

      y_predicted_oob_data_point = model_oob.predict(x_oob_data_point)
      y_predicted_oob_list.append(y_predicted_oob_data_point)
      #
    y_predicted_oob_list = np.array(y_predicted_oob_list)

    y_predicted_median = np.median(y_predicted_oob_list)
    y_predicted_oob_median_list.append(y_predicted_median)


  oob_score = 0

  for i in range(0, 506):
    # oob_score = (oob_score + (y[i] - y_predicted_oob_median_list[i] ) ** 2)
    # 13.828377285079045
    oob_score += (y[i] - y_predicted_oob_median_list[i] ) ** 2

  final_oob_score = oob_score/506

  return MSE, final_oob_score

print(bootstrapping_and_oob(x, y))
```

```
(0.013466976528570733, 12.777552353616024)
```

In [10]:
```
import scipy

x=boston.data #independent variables
y=boston.target #target variable

mse_boston_35_times_arr = []
oob_score_boston_35_times_arr = []

# Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB
for i in range(0, 35):
  mse, oob_score = bootstrapping_and_oob(x, y)
  mse_boston_35_times_arr.append(mse)
  oob_score_boston_35_times_arr.append(oob_score)


mse_boston_35_times_arr = np.array(mse_boston_35_times_arr)
oob_score_boston_35_times_arr = np.array(oob_score_boston_35_times_arr)

confidence_level = 0.95
degrees_of_freedom = 34 # sample.size - 1

mean_of_sample_mse_35 = np.mean(mse_boston_35_times_arr)
standard_error_mse = scipy.stats.sem(mse_boston_35_times_arr)


# Per document - https://www.kite.com/python/answers/how-to-compute-the-confidenc
# confidence_interval = scipy.stats.t.interval(confidence_level, degrees_freedom,
confidence_interval_mse_35 = scipy.stats.t.interval(confidence_level, degrees_of_
print("confidence_interval_mse_35 ", confidence_interval_mse_35)

# Now calculate confidence inter for oob score
mean_of_sample_oob_score_35 = np.mean(oob_score_boston_35_times_arr)
standard_error_of_sample_oob_score_35 = scipy.stats.sem(oob_score_boston_35_times_

confidence_interval_oob_score_35 = scipy.stats.t.interval(confidence_level, degre
print("confidence_interval_oob_score_35 ", confidence_interval_oob_score_35)
```

```
confidence_interval_mse_35  (0.07434321719602137, 0.182678895068009)
confidence_interval_oob_score_35  (13.229087513047887, 14.209067839788748)
```

Observation / Interpretation of above Confidence Interval

By definition we know the interpretation of a 95% confidence interval for the population mean as - If repeated random samples were taken and the 95% confidence interval was computed for each sample, 95% of the intervals would contain the population mean.

So in this case

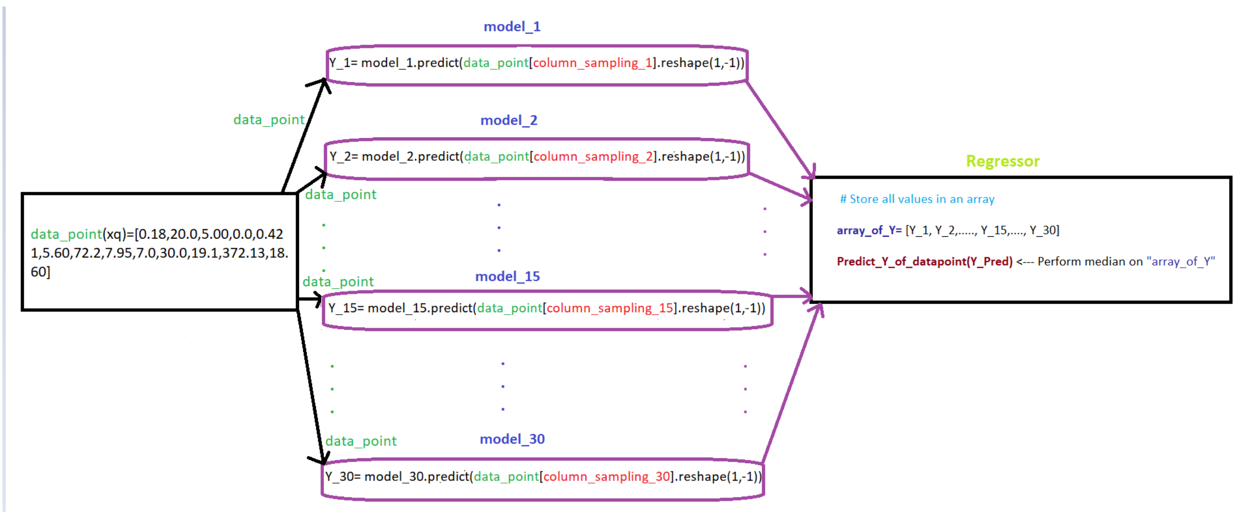
- MSE - There is a 95% chance that the confidence interval of (0.07434321719602137, 0.182678895068009) contains the true population mean of MSE.

- OOB Score - There is a 95% chance that the confidence interval of (13.229087513047887, 14.209067839788748) contains the true population mean of OOB Score.

Task 3 (send query point "xq" to 30 models)

We created 30 models by using 30 samples in TASK-1. Here, we need send query point "xq" to 30 models and perform the regression on the output generated by 30 models

Flowchart for Task 3



In [11]:

```
def predict_y_given_x_bootstrap(x_query):
  y_predicted_array_30_sample = []

  for i in range(0, 30):
    Dtree_i = all_model_list[i]
    # Extract x for ith data point with specific number of featues from list_sele
    x_data_point_i = [x_query[column] for column in list_selected_columns[i]]
    x_data_point_i = np.array(x_data_point_i).reshape(1, -1)
    y_predicted_i = Dtree_i.predict(x_data_point_i)
    y_predicted_array_30_sample.append(y_predicted_i)

  y_predicted_array_30_sample = np.array(y_predicted_array_30_sample)
  y_predicted_median = np.median(y_predicted_array_30_sample)
  return y_predicted_median


xq= [0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60]
y_predicted_for_xq = predict_y_given_x_bootstrap(xq)
y_predicted_for_xq
```

Out[11]: 19.4