Bootstrap assignment

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

Importing packages

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

In [2]:
```python
boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable
```

In [3]:
```python
x.shape
```

Out[3]:
```
(506, 13)
```

In [4]:
```python
x[:5]
```

Out[4]:
```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, 0.0000e+00, 5.3800e-01,
        6.5750e+00, 6.5200e+01, 4.0900e+00, 1.0000e+00, 2.9600e+02,
        1.5300e+01, 3.9690e+02, 4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        6.4210e+00, 7.8900e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9690e+02, 9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
        7.1850e+00, 6.1100e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
        1.7800e+01, 3.9283e+02, 4.0300e+00],
       [3.2370e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 4.5800e-01,
        6.9980e+00, 4.5800e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
        1.8700e+01, 3.9463e+02, 2.9400e+00],
       [6.9050e-02, 0.0000e+00, 2.1800e+00, 0.0000e+00, 4.5800e-01,
        7.1470e+00, 5.4200e+01, 6.0622e+00, 3.0000e+00, 2.2200e+02,
        1.8700e+01, 3.9690e+02, 5.3300e+00]])
```

Task 1

Step - 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 you 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
- **Note - While selecting the random 60% datapoints from the whole data, make sure that the selected datapoints are all exclusive, repetition is not allowed.**

Step - 2

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 your 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 $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$

Step - 3

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

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
  - you 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

Task 3

- **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.

A few key points

- Remember that the datapoints used for calculating MSE score contain some datapoints that were initially used while training the base learners (the 60% sampling). This makes these datapoints partially seen (i.e. the datapoints used for calculating the MSE score are a mixture of seen and unseen data). Whereas, the datapoints used for calculating OOB score have only the unseen data. This makes these datapoints completely unseen and therefore appropriate for testing the model's performance on unseen data.
- Given the information above, if your logic is correct, the calculated MSE score should be less than the OOB score.
- The MSE score must lie between 0 and 10.
- The OOB score must lie between 10 and 35.
- The difference between the left nad right confidence-interval values must not be more than 10. Make sure this is true for both MSE and OOB confidence-interval values.

Task - 1

Step - 1

- **Creating samples**

Algorithm

alt text

- **Write code for generating samples**

In [17]:
```python
def generating_samples(input_data, target_data):

    '''In this function, we will write code for generating 30 samples '''
    # you can use random.choice to generate random indices without replacement
    # Please have a look at this link https://docs.scipy.org/doc/numpy-1.16.0/reference/gene
rated/numpy.random.choice.html for more details
    # Please follow above pseudo code for generating samples
    selected_rows = np.random.choice(input_data.shape[0], size = 303, replace = False)
    replacing_rows = np.random.choice(selected_rows.shape[0],size=203,replace = True)
    selected_columns = np.random.choice(13,np.random.choice(np.arange(3,14)),replace=False)
    sample_data = input_data[selected_rows[:,None],selected_columns]
    target_of_sample_data = target_data[selected_rows]

    #Replicating data
    Replicated_sample_data = sample_data[replacing_rows]
    target_of_replicated_sample_data = target_of_sample_data[replacing_rows]

    #Concatenating data
    final_sample_data = np.vstack((sample_data,Replicated_sample_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_replicated_
sample_data.reshape(-1,1)))

    return final_sample_data , final_target_data, selected_rows, selected_columns
    #note please return as lists
```

Grader function - 1 </fongt>

In [18]:
```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)
grader_samples(a,b,c,d)
```

Out[18]:
```
True
```

- **Create 30 samples**

alt text

In [19]:
```python
# 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)
```

Grader function - 2

In [20]:
```python
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[20]:
```
True
```

Step - 2

Flowchart for building tree

alt text

- **Write code for building regression trees**

In [59]:
```python
from sklearn.tree import DecisionTreeRegressor
from tqdm import tqdm
#Store all the trained models
list_of_all_models = []
for i in tqdm(range(30)):
    model=DecisionTreeRegressor(max_depth=None)
    model.fit(list_input_data[i],list_output_data[i])
    list_of_all_models.append(model)
```

```
100%|██████████████████████████████████████████| 30/30 [00:00<00:00, 326.07it/s]
```

Flowchart for calculating MSE

alt text

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.

- **Write code for calculating MSE**

In [60]:
```python
pred_Y_datapoint=[]     #List to store the final predicted output
for i in range(len(x)):
    array_of_Y=[]
    for j,model in enumerate(list_of_all_models):
        pred=model.predict(x[i][list_selected_columns[j]].reshape(1,-1))
        array_of_Y.append(pred)
    pred_Y_datapoint.append(np.median(array_of_Y))

MSE = mean_squared_error(y,pred_Y_datapoint)
print('Mean Squared Error is :',MSE)
```

```
Mean Squared Error is : 9.678463534702134
```

Step - 3

Flowchart for calculating OOB score

alt text

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

- **Write code for calculating OOB score**

In [61]:
```python
Y_pred=[]                            #List to store the final predicted output
for i,data_point in enumerate(x):
    list_Y_values=[]                 #List to store the datapoints which are not present in the
30 samples
    for j,model in enumerate(list_of_all_models):
        if i not in list_selected_row[j]:
            pred = model.predict(data_point[list_selected_columns[j]].reshape(1,-1))
            list_Y_values.append(pred)
    Y_pred.append(np.median(list_Y_values))

OOB = mean_squared_error(y,Y_pred)
print('OOB score is :',OOB)
```

```
OOB score is : 23.887024138774578
```

Task 2

In [65]:
```python
mse_35_scores=[]
oob_35_scores=[]

for i in range(35):
    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)

# Training 30 DT Regressors from 30 samples
    list_of_all_models = []
    for i in range(30):
        model=DecisionTreeRegressor(max_depth=None)
        model.fit(list_input_data[i],list_output_data[i])
        list_of_all_models.append(model)

#Calculating the MSE score
    pred_Y_datapoint=[]
    for i in range(len(x)):
        array_of_Y=[]
        for j,model in enumerate(list_of_all_models):
            pred=model.predict(x[i][list_selected_columns[j]].reshape(1,-1))
            array_of_Y.append(pred)
        pred_Y_datapoint.append(np.median(array_of_Y))
    MSE = mean_squared_error(y,pred_Y_datapoint)
    mse_35_scores.append(MSE)

#Calculating the OOB score
    Y_pred=[]
    for i,data_point in enumerate(x):
        list_Y_values=[]
        for j,model in enumerate(list_of_all_models):
            if i not in list_selected_row[j]:
                pred = model.predict(data_point[list_selected_columns[j]].reshape(1,-1))
                list_Y_values.append(pred)
        Y_pred.append(np.median(list_Y_values))
    OOB = mean_squared_error(y,Y_pred)
    oob_35_scores.append(OOB)
```

Confidence Interval:

Confidence interval is a range of estimates for an unknown population interval,defined as an interval with a lower bound and upper bound. The 95% confidence interval is the most common and it means that there is 95% probability the true population mean will fall within the confidence interval range calculated using the samples.

Here we are assuming that we dont have the knowlegde on population standard deviation,SE is used is to make confidence intervals of the unknown population mean. If the sampling distribution is normally distributed, the sample mean, the standard error, and the quantiles of the normal distribution can be used to calculate confidence intervals for the true population mean.

95% confidence interval = (sample_mean-2(standard deviation of sample)/sqrt(sample_size)),
(sample_mean+2(standard deviation of sample)/sqrt(sample_size))

In [72]:
```python
#Computing the Confidence Intervals for MSE scores
sample_size = len(mse_35_scores)
sample_mean = np.mean(mse_35_scores)
sample_std = np.std(mse_35_scores)
left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
print('95% Confidence Interval of MSE is', left_limit, 'to', right_limit)
```

```
95% Confidence Interval of MSE is 7.929 to 8.839
```

In [71]:
```python
#Computing the Confidence Intervals for OOB scores
sample_size = len(oob_35_scores)
sample_mean = np.mean(oob_35_scores)
sample_std = np.std(oob_35_scores)
left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
print('95% Confidence Interval of OOB is', left_limit, 'to', right_limit)
```

```
95% Confidence Interval of OOB is 23.254 to 24.6
```

Task 3

Flowchart for Task 3

Hint: 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.

alt text

- **Write code for TASK 3**

In [66]:
```python
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]
xq=np.array(xq)
for i in range(30):
    y_pred=[]
    pred=list_of_all_models[i].predict(xq[list_selected_columns[i]].reshape(1,-1))
    y_pred.append(pred)
Predict_Y_datapoint=np.median(y_pred)
print('The house price for the given query point is :',Predict_Y_datapoint)
```

```
The house price for the given query point is : 18.2
```

Write observations for task 1, task 2, task 3 indetail

Task 1:

As we are performing bootstrap aggregation on the given dataset, training the same set of datapoints in the samples will lead to overfitting. As we want our base learners to have high variance and low bias, but only a subset of features get picked for each model and each model is different from each other. Even if the original dataset changes, very few subsets change as the datapoints are randomnly picked to form the samples.

Finally the aggregation of each subset of model is done to achieve the resultant model which tends to reduce variance without impacting the bias.

Task 2:

Confidence interval is a range of estimates for an unknown population interval,defined as an interval with a lower bound and upper bound. The 95% confidence interval is the most common and it means that there is 95% probability the true population mean will fall within the confidence interval range calculated using the samples.

Here we are assuming that we dont have the knowlegde on population standard deviation,SE is used is to make confidence intervals of the unknown population mean. If the sampling distribution is normally distributed, the sample mean, the standard error, and the quantiles of the normal distribution can be used to calculate confidence intervals for the true population mean.

95% confidence interval = (sample_mean-2(standard deviation of sample)/sqrt(sample_size)),
(sample_mean+2(standard deviation of sample)/sqrt(sample_size))

Task 3:

Given a query point, we are performing bagging plus column sampling to predict the real value and taking the median on all the predictions made to get the final value. The runtime complexity is O(depth * N) where depth = max_depth of each tree and N is the number of base learners.