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
from sklearn.tree import DecisionTreeRegressor
from statistics import median
```

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 $OOB Score = \frac{1}{506}\sum_{i=1}^{506}(y^{i} - y^{i}_{pred})^{2}$.**

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 [5]:
```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/generated/numpy.random.
    # Please follow above pseudo code for generating samples


    # return sampled_input_data , sampled_target_data,selected_rows,selected_columns
    #note please return as lists

    sixty_percent_data_indices = np.random.choice(506, 303, replace=False)
    forty_percent_data_indices = np.random.choice(303, 203, replace=True)


    num = np.random.randint(3,14)
    column_indices = np.random.choice(13,num)

    sixty_percent_data= input_data[sixty_percent_data_indices]
    sixty_percent_data = sixty_percent_data[:,column_indices]
    sixty_percent_data_target = target_data[sixty_percent_data_indices].reshape(-1,1)

    forty_percent_data= sixty_percent_data[forty_percent_data_indices]
    forty_percent_data_target = sixty_percent_data_target[forty_percent_data_indices].reshape(-1,1)

    sample_data = np.vstack((sixty_percent_data,forty_percent_data))

    sample_target = np.vstack((sixty_percent_data_target,forty_percent_data_target))


    return sample_data, sample_target, sixty_percent_data_indices, column_indices
```

Grader function - 1 </fongt>

In [6]:
```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[6]:
```
True
```

In [7]:
```python
len(set([str(i) for i in a]))
```

Out[7]:
```
303
```

- **Create 30 samples**

alt text

In [8]:
```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 [9]:
```python
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[9]:
```
True
```

Step - 2

Flowchart for building tree

alt text

- **Write code for building regression trees**

In [10]:
```python
predictions = []
model= []
for i in range(0,30):
    DT_Regressor = DecisionTreeRegressor(max_depth = None, random_state = 0)
    model.append(DT_Regressor.fit(list_input_data[i],list_output_data[i]))
```

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 [11]:
```python
predictions = []
MSE = []
for i in range(0,30):
    predictions.append(model[i].predict(x[:,list_selected_columns[i]]))
predicted_target = np.median(predictions,axis = 0)
MSE = mean_squared_error(y,predicted_target)
print("MSE:", MSE)
```

```
MSE: 0.09363150765082055
```

Step - 3

Flowchart for calculating OOB score

alt text

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

- **Write code for calculating OOB score**

In [12]:
```python
y_oob_predicted_target = []
y_oob_target = []
for j in range(506):
    oob_val_list = []
    oob_val = 0
    for i in range(30):
        if j not in list_selected_row[i]:

            oob_val_list.append(predictions[i][j])
    if oob_val_list:
        oob_val = np.median(np.asarray(oob_val_list))
        y_oob_predicted_target.append(oob_val)
        y_oob_target.append(y[j])

oob_score = mean_squared_error(y_oob_target,y_oob_predicted_target)
print("OOB Score:",oob_score)
```

```
OOB Score: 17.1809121386495
```

Task 2

In [13]:
```python
def bagging(input_x,target_y):
    list_input_data =[]
    list_output_data =[]
    list_selected_row= []
    list_selected_columns=[]

    predictions = []
    model= []

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

    for i in range(0,30):
        DT_Regressor = DecisionTreeRegressor(max_depth = None, random_state = 0)
        model.append(DT_Regressor.fit(list_input_data[i],list_output_data[i]))


    predictions = []
    for i in range(0,30):
        predictions.append(model[i].predict(input_x[:,list_selected_columns[i]]))
        predicted_target = np.median(predictions, axis = 0)

    MSE = mean_squared_error(target_y,predicted_target)


    y_oob_predicted_target = []
    y_oob_target = []
    for j in range(506):
        oob_val_list = []
        oob_val = 0
        for i in range(30):
            if j not in list_selected_row[i]:
                oob_val_list.append(predictions[i][j])
        if oob_val_list:
            oob_val = np.median(np.asarray(oob_val_list))
            y_oob_predicted_target.append(oob_val)
            y_oob_target.append(y[j])

    oob_score = mean_squared_error(y_oob_target,y_oob_predicted_target)
    return MSE,oob_score
```

In [14]:
```python
mse_scores = []
oob_scores = []
for i in range(35):
    mse_value, oob_value = bagging(x,y)
    mse_scores.append(mse_value)
    oob_scores.append(oob_value)
```

In [15]:
```python
def confidence_interval(sample_data):
    sample_mean = sample_data.mean()
    sample_std = sample_data.std()
    sample_count = len(sample_data)
    left_limit = np.round(sample_mean-(2*(sample_std/np.sqrt(sample_count))),3)
    right_limit = np.round(sample_mean+(2*(sample_std/np.sqrt(sample_count))),3)
    return left_limit, right_limit
```

In [16]:
```python
left_limit_mse, right_limit_mse = confidence_interval(np.asarray(mse_scores))
left_limit_oob, right_limit_oob = confidence_interval(np.asarray(oob_scores))
print("Confidence interval of MSE scores:[{}, {}]".format(left_limit_mse,right_limit_mse))
print("Confidence interval of OOB scores:[{}, {}]".format(left_limit_oob,right_limit_oob))
```

```
Confidence interval of MSE scores:[0.083, 0.19]
Confidence interval of OOB scores:[16.658, 18.028]
```

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 [17]:
```python
xq= np.array([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_column_sampled = []
for i in range(30):
    xq_column_sampled.append(xq[list_selected_columns[i]])
```

In [18]:
```python
xq_pred_list=[]
for i in range(0,30):
    xq_pred_list.append(model[i].predict(xq_column_sampled[i].reshape(1,-1)))
xq_pred = np.median(xq_pred_list)
```

In [19]:
```python
print(xq_pred)
```

```
18.5
```

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

1. **The MSE score is very low as we have fit the model for 60% of our data (all these data will have very low error). Only 40% of our data is unseen and so our error comes from this 40% data.**
2. **The OOB score is very high as we are only testing the model on 40% of data that was unseen during training our model. Simply put, OOB score is the error on samples that were unseen during training.**
3. **OOB scoring is very good method of testing our model as we will have different oob samples for each model and so we will be able to test all of our data without much loss in information (If we removed some data from our training data to be used as test data, we would have lost the information from those test data points.)**
4. **As per central limit theorem, if we do repeated random sampling and find the 95% confidence interval, we can say that there is 95% chance that the MSE scores and oob scores of the popluation lies within that interval. So, we can say that:**
   ```
   a) There is 95% chance that MSE Score of the population lies between 0.083 and 0.19
   b) There is 95% chance that OOB Score of the population lies between 16.658 and 18.028
   ```
5. **In Task 3, as we have already fit our 30 models on the samples, we can easily predict the price of house for any new query point after performing column sampling on the query point.**