Bootstrap assignment

Importing packages

In [71]:

```
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 [72]:

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

In [73]:

```
x.shape
```

Out[73]:

```
(506, 13)
```

In [74]:

```
y.shape
```

Out[74]:

```
(506,)
```

Task - 1

Step - 1

- **Creating samples**

- **Code for generating samples**

In [75]:

```
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/g
enerated/numpy.random.choice.html for more details
    # Please follow above pseudo code for generating samples


    # return sampled_input_data , sampled_target_data,selected_rows,selected_columns
    #note please return as lists
    rows = len(input_data)
    cols = len(input_data[0])
    #https://docs.scipy.org/doc/numpy-1.16.0/reference/generated/numpy.random.choice.html
    row_indicies = np.random.choice(rows,303,replace=False)
    replicated_row_indicies = np.random.choice(row_indicies,203)
    size = np.random.randint(3,cols+1)
    columns = np.random.choice(cols,size,replace=False)
    sample_data = input_data[row_indicies[:,None],columns]
    target_sample_data = target_data[row_indicies]

    #replicating data
    replicated_sample_data = input_data[replicated_row_indicies[:,None],columns]
    target_replicated_sample_data = target_data[replicated_row_indicies]

    #concatinating data
    final_sample_data = np.vstack((sample_data,replicated_sample_data))
    final_target_data = np.vstack((target_sample_data.reshape(-1,1),target_replicated_sa
mple_data.reshape(-1,1)))

    return final_sample_data,final_target_data,row_indicies,columns
```

Grader function - 1

In [76]:

```
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[76]:

```
True
```

- **Create 30 samples**

In [77]:

```
# Use generating_samples function to create 30 samples
# store the 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 [78]:

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

Out[78]:

```
True
```

Step - 2

- **Code for building regression trees**

In [79]:

```
from sklearn.tree import DecisionTreeRegressor
models_list = []

for i in range(0,30):
    model_i = DecisionTreeRegressor()
    model_i.fit(list_input_data[i],list_output_data[i])

    #storing all the trained models in a list
    models_list.append(model_i)
```

- **Code for calculating MSE**

In [80]:

```
def compute_mse():
    y_pred = []
    for i,p in enumerate(x):
        models_y_pred = []
        for j,model in enumerate(models_list):
            pred = model.predict(p[list_selected_columns[j]].reshape(1,-1))
            models_y_pred.append(pred)
        y_pred.append(np.median(models_y_pred))

    mse =  mean_squared_error(y,y_pred)
    return mse
```

In [81]:

```
mse = compute_mse()
print(mse)
```

```
0.02918478260869561
```

Step - 3

- **Code for calculating OOB score**

In [82]:

```
def compute_oob_score():
    y_pred = []
    for i,p in enumerate(x):
        models_y_pred = []
        for j,model in enumerate(models_list):
            if i not in list_selected_row[j]:
                pred = model.predict(p[list_selected_columns[j]].reshape(1,-1))
                models_y_pred.append(pred)
        y_pred.append(np.median(models_y_pred))

    oob_score =  mean_squared_error(y,y_pred)
    return oob_score
```

In [83]:

```
oob_score = compute_oob_score()
print(oob_score)
```

```
10.880513833992095
```

Task 2

In [84]:

```
from sklearn.tree import DecisionTreeRegressor
list_mse = []
list_oob_score = []
for i in range(0,35):
    # Use generating_samples function to create 30 samples
    # store the 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)


    models_list = []

    for i in range(0,30):
        model_i = DecisionTreeRegressor()
        model_i.fit(list_input_data[i],list_output_data[i])

        #storing all the trained models in a list
        models_list.append(model_i)


    mse = compute_mse()
    oob_score = compute_oob_score()
    list_mse.append(mse)
    list_oob_score.append(oob_score)
```

In [87]:

```
#calculating the mean of all the elements in list mse and list oob score
mean_mse = round(sum(list_mse)/len(list_mse),3)
#print(mean_mse)
mean_oob_score =  round(sum(list_oob_score)/len(list_oob_score),3)
#print(mean_oob_score)


#calculating the standard error of mean
sem_mse = round(np.std(list_mse)/len(list_mse)**0.5,3)
#print(sem_mse)
sem_oob_score = round(np.std(list_oob_score)/len(list_oob_score)**0.5,3)
#print(sem_oob_score)


#Calculating the confidence intravel of MSE
print("The confidence intravel of MSE is", (mean_mse - 2*sem_mse,mean_mse + 2*sem_mse))
#Calculating the confidence intravel of oob SCORE
print("The confidence intravel of OOB SCORE is",(mean_oob_score - 2*sem_oob_score,mean_oo
b_score + 2*sem_oob_score))
```

```
The confidence intravel of MSE is (0.051, 0.099)
The confidence intravel of OOB SCORE is (13.146999999999998, 14.471)
```

Task 3

In [86]:

```
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])
models_y_pred = []
for i,model in enumerate(models_list):
    pred = model.predict(xq[list_selected_columns[i]].reshape(1,-1))
    models_y_pred.append(pred)
y_pred = np.median(models_y_pred)
print("The price of the house would be", y_pred)
```

```
The price of the house would be 18.5
```

Observations

Task-1 :

1. **Since we are getting a very low MSE value therefore we can say the model is making near to perfect prediction.**
2. **The oob_score avoids the problem of data leakage hence ensuring a better predictive model. And since there is no leakage, so there is no overfitting of data hence least variance.**

Task-2 :

1. **The confidence intravel for MSE is (0.051, 0.099) which means that we can be 95% confident that the mse values will be within this range.**
2. **The confidence intravel for OOB SCORE is (13.146999999999998, 14.471) which means that we can be 95% confident that the oob_score values will be within this range.**

Task-3 :

1. **Here given the query point we are predicting the price of the house passing the query point through all the models and calculating the median of the all the predicted values from the all the models.**