In [1196]:
```
import numpy as np
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
```

In [1197]:
```
boston = load_boston()
x=boston.data
y=boston.target
```

In [1198]:
```
x.shape
```

Out[1198]:
```
(506, 13)
```

In [1199]:
```
x[:5]
```

Out[1199]:
```
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**

- **Write code for generating samples**

In [1200]:
```
def generating_samples(input_data, target_data):

    selecting_rows=np.random.choice(len(input_data), 303,replace=False)
    rows=selecting_rows.tolist()

    Replacing_rows=np.random.choice(len(selecting_rows), 203,replace=False)

    #selecting_columns
    k=np.random.randint(3,13) #selecting  numbers of colums randomly

    Selecting_columns=np.random.choice(13, k,replace=False)

    columns=Selecting_columns.tolist()

    Sample_data=input_data[selecting_rows[:,None],Selecting_columns]

    target_of_sample_data=target_data[selecting_rows]

    #Replicating data

    Replicated_sample_data=Sample_data[Replacing_rows]

    target_of_Replicated_sample_data=target_of_sample_data[Replacing_rows]

    #Concatinating_data

    final_sample_data=np.row_stack((Sample_data,Replicated_sample_data))
    final_sample_data=final_sample_data.tolist()

    #reshape
    target_of_sample_data=target_of_sample_data.reshape(-1,1)
    target_of_Replicated_sample_data=target_of_Replicated_sample_data.reshape(-1,1)

    final_target_data=np.row_stack((target_of_sample_data,target_of_Replicated_sample_data))
    final_target_data=final_target_data.tolist()

    return final_sample_data,final_target_data,rows,columns


a,b,c,d=generating_samples(x,y)
```

Grader function - 1 </fongt>

In [1201]:
```
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[1201]: True

- **Create 30 samples**

In [1202]:
```
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 [1203]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[1203]: True

Step - 2

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

- **Write code for building regression trees**

In [1204]:
```
models=[]

from sklearn.tree import DecisionTreeRegressor

for i in range(len(list_selected_row)):
    #Building a regression trees on each of 30 samples.
    input_data=x[np.array(list_selected_row[i])[:,None],np.array(list_selected_columns[i])]
    Target_data=np.array(y)[list_selected_row[i]]
    model=DecisionTreeRegressor(max_depth=None)
    model=model.fit(input_data,Target_data)
    models.append(model)

#Computed the predicted values of each data point
y_pred=[]
for i in range(0,506):
    array_of_y=[]
    for j in range(len(models)):
        y1=models[j].predict((x[i,list_selected_columns[j]]).reshape(1,-1))
        array_of_y.append(y1)
    y_pred.append(np.median(array_of_y))
```

- **Write code for calculating MSE**

In [1205]:
```
#Calculating MSE

MSE=MSE=mean_squared_error(y,np.array(y_pred))
print("MSE =",MSE)
```

```
MSE = 0.048296003513394865
```

- **Write code for calculating OOB score**

In [1206]:
```
predicted_y=[]
for i in range(0,506):
    models_y_pred=[]
    for j in range(len(models)):
        if i not in list_selected_row[j]:
            y1=models[j].predict((x[i,list_selected_columns[j]]).reshape(1,-1))
            models_y_pred.append(y1)

    predicted_y.append(np.median(models_y_pred))
print("OOB Score =",mean_squared_error(y,np.array(predicted_y)))
```

```
OOB Score = 11.765008430317637
```

Task 2

- **Computing CI of OOB Score and Train MSE**

- **Repeat Task 1 for 35 times**

In [1213]:
```
TrainMSE=[]
OOB_SCORE=[]

for i in range(0,35):
    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)
    models=[]
    from sklearn.tree import DecisionTreeRegressor
    for i in range(len(list_selected_row)):

        #Building a regression trees on each of 30 samples.
        input_data=x[np.array(list_selected_row[i])[:,None],np.array(list_selected_columns[i])]
        Target_data=np.array(y)[list_selected_row[i]]
        model=DecisionTreeRegressor(max_depth=None)
        model=model.fit(input_data,Target_data)
        models.append(model)
    #Computed the predicted values of each data point
    y_pred=[]
    for i in range(0,506):
        array_of_y=[]
        for j in range(len(models)):
            y1=models[j].predict((x[i,list_selected_columns[j]]).reshape(1,-1))
            array_of_y.append(y1)
        y_pred.append(np.median(array_of_y))
    #Calculating MSE
    MSE=MSE=mean_squared_error(y,np.array(y_pred))
    TrainMSE.append(MSE)

    predicted_y=[]
    for i in range(0,506):
        models_y_pred=[]
```

```
        for j in range(len(models)):
            if i not in list_selected_row[j]:
                y1=models[j].predict((x[i,list_selected_columns[j]]).reshape(1,-1))
                models_y_pred.append(y1)
        predicted_y.append(np.median(models_y_pred))
    OOB_SCORE.append(mean_squared_error(y,np.array(predicted_y)))
```

- **Computing CI of Train MSE**

In [1226]:
```
sample_mean = np.array(TrainMSE).mean()
sample_std =  np.array(TrainMSE).std()
sample_size = len(TrainMSE)
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("C.I. of MSE ",[left_limit,right_limit])
```

```
C.I. of MSE  [0.078, 0.124]
```

- **Computing CI of OOB Score**

In [1233]:
```
sample_mean = np.array(OOB_SCORE).mean()
sample_std =  np.array(OOB_SCORE).std()
sample_size = len(OOB_SCORE)
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("C.I. of OOB_SCORE ",[left_limit,right_limit])
```

```
C.I. of OOB_SCORE  [13.887, 15.179]
```

Task 3

- **Given a single query point predict the price of house.**

In [1159]:
```
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]
```

In [1163]:
```
y_predq=[]
for j in range(len(models)):
    #creating new list with column sampling
    xq1=[]
    for i in list_selected_columns[j]:
        xq1.append(xq[i])
    #predicting price
    y1=models[j].predict((np.array(xq1)).reshape(1,-1))
    y_predq.append(y1)
print("Predicted House Price for given query point = ",np.median(y_predq))
```

```
Predicted House Price for given query point =  18.5
```

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

task 1

- We are taking bootstrap sample with same size as original dataset.
- We are first taking sample1 data from original dataset without replacement and then taking sample2 from sample1 without replament and add both sample1 and sample2
- We are creating 30 such samples and each sample with different set of columns.
- These steps will convert weak model to better perfoming model.
- While training model using DecisionTreeRgressor we are providing max_depth as None which willgrow the tree to the largest
- i.e. nodes are expanded untill leaves are pure
- The MSE is a measure of the quality of an estimator.The lower MSE the closer is predicted value to actual value.
- We are getting lower MSE so we can conclude that our model is better performing model and it is better fitted model.
- OOB score is a method of measuring the prediction error of boosted decision tree.
- We are using datapoints which are out of bag that is not used to train model will be used to test the models.
- We are getting high OOBscore,model is not performing better for unseen data

task 2

- After repeating task one for 35 times created two list (samples) one is TrainMSE and other is OOB_SCORE
- C.I for TrainMSE is [0.078,0.124] i.e. Margin of Error is +-0.023 (smaller margin)
- C.I for TrainMSE is [13.887, 15.179] i.e. Margin of Error is +-0.6459 (slighly larger marging)
- Lower variability will result in a smaller margin of error
- Higher variability will result in a Larger margin of error

task 3

- For given query point first we did column sampling and then predicted value using all 30 models.
- Took median of all 30 predictions which is stored in list and computed house price which is 18.5
- Point xq is out of bag datapoint(not present in original dataset)
- Predicted values from maximum models is between 18.5-20
- For unseen datapoint also different models are predicting similar results.
- For given xq models are performing better.

In [ ]:
```
!jupyter nbconvert --to html Bootstrap_solve.ipynb
```