Applying Bootstrap Aggregated Random Forest using custom functions.

In [1]:
```
import numpy as np
import random as random
from scipy.sparse import vstack
# Here we are using sklearn's boston dataset
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
```

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

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

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

In [4]:
```
x[0]
```

Out[4]:
```
array([6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
       6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
       4.980e+00])
```

Creating samples: Randomly create 30 samples from the whole boston data points.

Creating each sample: Considering any random 303(60% of 506) data points from whole data set and then replicate any 203 points from the sampled points.

For example: 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 this sample, consider they are [5,8,3,7] so our final sample will be [4,5,7,8,9,3,5,8,3,7].

Also, As a part of Bagging when we are taking the random samples we would also make sure that each of our sample will have different set of columns.

For example: 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. Atleast 3 feautres/columns would be there in each sample.

Similarly we will create 30 samples like this.

In [5]:
```
# In this function we will write code for generating 30 samples
def generating_samples(input_data, target_data):

    # Using np.random.choice for genrating random sample.
    # https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html

    selected_rows=np.random.choice(len(input_data), 303, replace=False) # Selecting 303 unique random d
ata points.
    #print(len(selected_rows))
    replicating_rows=np.random.choice(selected_rows, 203) # Replicating 203 points from 303 points whic
h were generated above.
    #print(len(replicating_rows))
    selected_columns=[]
    num=random.randint(3,12) # Selecting random number for columns with minimun 3 columns.
    #print(num)
    selected_columns=random.sample(range(1,13), num) # Selecting random columns.
    #print(selected_columns)
    sample_data=input_data[selected_rows[:,None], selected_columns] # Adding the rows and columns to a
variable.
    #print(sample_data)
    target_sample_data=target_data[selected_rows] # Adding output variable rows similar to as sample da
ta.
    #print(len(target_sample_data))
    replicated_sample_data=input_data[replicating_rows[:,None], selected_columns] # Adding repeating ro
ws and columns.
    #print(replicated_sample_data.shape)
    target_replicate_sample_data=target_data[replicating_rows] # Adding output variable repeating rows
similar to as sample data.
    #print(len(target_replicate_sample_data))
    final_sample=np.vstack((sample_data, replicated_sample_data)) # Combining the data.
    final_target=np.vstack((target_sample_data.reshape(-1,1), target_replicate_sample_data.reshape(-1,1
))) # Combining output data.
    #print(final_sample.shape)
    #print(final_target.shape)

    return final_sample, final_target, selected_rows, selected_columns
```

Creating 30 samples.

In [7]:
```
# Use generating_samples function to create 30 samples.
# Storing these 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)
```

Now we are training 30 Decision Tree's with high variance for our sample data.

In [9]:
```
from sklearn.tree import DecisionTreeRegressor
from tqdm import tqdm
```

In [10]:
```
# Training 30 different high variance models for our sample data.
list_models=[]
for i in tqdm(range(len(list_input_data))):
    model=DecisionTreeRegressor(max_depth=None)
    model.fit(list_input_data[i], list_output_data[i])
    list_models.append(model)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 30/30 [00:00<
00:00, 191.27it/s]
```

Now we are predicting the value of each data point with our 30 models and storing the median of prediction of 30 models. Then calculating the Mean Sqaure Error from median of predictions.

In [11]:
```
predictions=[]

# Iterating over each data point.
for i in tqdm(range(len(x))):
    data=x[i]
    pred=[]

    # Making prediction on each data points with 30 models we created above.
    for j in range(len(list_models)):
        columns=list_selected_columns[j]
        model=list_models[j]
        p = model.predict(data[columns].reshape(1,-1))
        pred.append(p)

    # Taking median of data point from each model
    predictions.append(np.median(pred))

#print(predictions)
mse=mean_squared_error(y,predictions) # Computing Mean squre error
print('Mean Square Error: ',mse)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 506/506 [00:01<
00:00, 259.34it/s]

Mean Square Error:  0.07789946264658806
```

Now we are calculating Out of Bag Score (OOB score).

This is similar to above step, the only difference is that we will not make predition if our model was trained using the same data point for which we are predicting the value.

In [12]:
```
# Performing the same above steps but this time not to include the the points from which model was trai
ned.
predictions=[]
for i in tqdm(range(len(x))):
    data=x[i]
    pred=[]

    for j in range(len(list_models)):
        columns=list_selected_columns[j]
        rows=list_selected_row[j]
        model=list_models[j]
        #Skipping the data points from which models was trained.
        if i not in rows:
            p = model.predict(data[columns].reshape(1,-1))
            pred.append(p)
    #print(pred)
    predictions.append(np.median(pred))

mse=mean_squared_error(y,predictions)
print('OOB Score: ',mse)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 506/506 [00:00<
00:00, 537.75it/s]

OOB Score:  14.080603198656924
```

Repeating all the above steps again to obtain different MSE and OOB scores.

In [13]:
```
# Repeating all above steps 35 times.
list_mse=[]
list_oob=[]

for sample in tqdm(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)

    list_models=[]
    for i in range(len(list_input_data)):
        model=DecisionTreeRegressor(max_depth=None)
        model.fit(list_input_data[i], list_output_data[i])
        list_models.append(model)

    predictions=[]
    for i in range(len(x)):
        data=x[i]
        pred=[]

        for j in range(len(list_models)):
            columns=list_selected_columns[j]
            model=list_models[j]
            p = model.predict(data[columns].reshape(1,-1))
            pred.append(p)


        predictions.append(np.median(pred))

    #print(predictions)
    mse=mean_squared_error(y,predictions)
    list_mse.append(mse)


    predictions=[]
    for i in range(len(x)):
        data=x[i]
        pred=[]

        for j in range(len(list_models)):
            columns=list_selected_columns[j]
            rows=list_selected_row[j]
            model=list_models[j]
            if i not in rows:
                p = model.predict(data[columns].reshape(1,-1))
                pred.append(p)
        #print(pred)
        predictions.append(np.median(pred))

    mse=mean_squared_error(y,predictions)
    list_oob.append(mse)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 35/35 [01:35
<00:00,  2.72s/it]
```

Now we are computing Confidence Intervals of MSE and OOB scores.

In [14]:
```
# Computing Confidence Intervals.
# https://www.statology.org/confidence-intervals-python/
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html

import scipy.stats as st
# Creating 95% confidence interval for MSE
interval_mse=st.norm.interval(alpha=0.95, loc=np.mean(list_mse), scale=st.sem(list_mse))
print('Confidence Interval for Mean Square Errors:',interval_mse)

# Creating 95% confidence interval for MSE
interval_oob=st.norm.interval(alpha=0.95, loc=np.mean(list_oob), scale=st.sem(list_oob))
print('Confidence Interval for OOB scores:',interval_oob)
```

```
Confidence Interval for Mean Square Errors: (0.08597271699380007, 0.16566371321125348)
Confidence Interval for OOB scores: (13.513575196555944, 14.537933193188813)
```

Using our 30 models to predict the price of house from a new data point.

In [15]:
```
# Repeating the same steps as done in Task 1. But this time just to predict for a single query point.
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]
pt=np.array(xq)
ans=[]
for i in range(len(list_models)):
    cols=list_selected_columns[i]
    model=list_models[i]
    pred=model.predict(pt[cols].reshape(1,-1))
    ans.append(pred)

print('Price of the house of new data point:',np.median(ans))
```

```
Price of the house of new data point: 18.9
```