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 [22]:
```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 tqdm import tqdm
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import sys
```

In [23]:
```python
boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will b
e removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

In [24]:
```python
x.shape
y.shape
```

Out[24]:
```
(506,)
```

In [25]:
```python
x[:5]
y[:5]
```

Out[25]:
```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

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 [26]:
```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.choice.html for more details
    # Please follow above pseudo code for generating samples
    selecting_rows = np.random.choice(len(input_data), size = 303, replace = False)
    replacing_rows = np.random.choice(len(selecting_rows), size = 203, replace = False)
    number_of_cols = np.random.randint(3, 13)
    selecting_colmns = np.random.choice(13, size = number_of_cols, replace = False)
    # Sample data
    sample_data = input_data[selecting_rows[:, None], selecting_colmns]
    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]

    # Stacking of 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, selecting_rows,selecting_colmns
    #note please return as lists
```

Grader function - 1 </fongt>

In [27]:
```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[27]:
```
True
```

- **Create 30 samples**

alt text

In [28]:
```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_task1=[]

for i in tqdm(range(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_task1.append(d)
```

```
100%|██████████| 30/30 [00:00<00:00, 3722.76it/s]
```

Grader function - 2

In [29]:
```python
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[29]:
```
True
```

Step - 2

Flowchart for building tree

alt text

- **Write code for building regression trees**

In [30]:
```python
list_of_all_models_task1 = []
for i in tqdm(range(30)):
  clf = DecisionTreeRegressor(max_depth = None, min_samples_split=2)
  train_x = list_input_data[i]
  train_y = list_output_data[i]
  clf.fit(train_x, train_y)
  list_of_all_models_task1.append(clf)
```

```
100%|██████████| 30/30 [00:00<00:00, 368.65it/s]
```

In [31]:
```python
# Predictions on train data
train_predictions = []
for i in tqdm(range(len(x))):
  pred_i = []
  for j in range(30):
    data_pnt = x[i][list_selected_columns_task1[j]].reshape(1, -1)
    pred_i.append(list_of_all_models_task1[j].predict(data_pnt))

  train_predictions.append(np.median(np.asarray(pred_i)))
```

```
100%|██████████| 506/506 [00:01<00:00, 418.74it/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 [32]:
```python
# Calculating mean squared error
train_mean_square_error = mean_squared_error(y, train_predictions)
train_mean_square_error
```

Out[32]:
```
0.023962450592885355
```

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 [33]:
```python
# Predictions on OOB data
oob_predictions = []
for i in tqdm(range(len(x))):
  pred_i = []
  for j in range(30):
    if i not in list_selected_row[j]:
      data_pnt = x[i][list_selected_columns_task1[j]]
      pred_i.append(list_of_all_models_task1[j].predict(data_pnt.reshape(1, -1)))
  oob_predictions.append(np.median(np.asarray(pred_i)))
```

```
100%|██████████| 506/506 [00:00<00:00, 805.28it/s]
```

In [34]:
```python
# Calculating OOB score
oob_score = mean_squared_error(y, oob_predictions)
oob_score
```

Out[34]:
```
14.621926877470354
```

Task 2

Redefining functions

In [16]:
```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.choice.html for more details
    # Please follow above pseudo code for generating samples
    selecting_rows = np.random.choice(len(input_data), size = 303, replace = False)
    replacing_rows = np.random.choice(len(selecting_rows), size = 203, replace = False)
    number_of_cols = np.random.randint(3, 13)
    selecting_colmns = np.random.choice(13, size = number_of_cols, replace = False)
    # Sample data
    sample_data = input_data[selecting_rows[:, None], selecting_colmns]
    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]

    # Stacking of 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, selecting_rows,selecting_colmns
    #note please return as lists

def get_models(list_input_data, list_output_data):
  list_of_all_models = []
  for i in range(30):
    clf = DecisionTreeRegressor(max_depth = None, min_samples_split=2)
    train_x = list_input_data[i]
    train_y = list_output_data[i]
    clf.fit(train_x, train_y)
    list_of_all_models.append(clf)
  return list_of_all_models

def get_train_predictions(x, list_selected_columns, list_of_all_models):
  train_predictions = []
  for i in range(len(x)):
    pred_i = []
    for j in range(30):
      data_pnt = x[i][list_selected_columns[j]].reshape(1, -1)
      pred_i.append(list_of_all_models[j].predict(data_pnt))

    train_predictions.append(np.median(np.asarray(pred_i)))
  return train_predictions

def get_oob_predictions(x, list_selected_columns, list_selected_row, list_of_all_models):
  oob_predictions = []
  for i in range(len(x)):
    pred_i = []
    for j in range(30):
      if i not in list_selected_row[j]:
        data_pnt = x[i][list_selected_columns[j]]
        pred_i.append(list_of_all_models[j].predict(data_pnt.reshape(1, -1)))
    oob_predictions.append(np.median(np.asarray(pred_i)))
  return oob_predictions

def get_mse_score(y_true, y_pred):
  return mean_squared_error(y_true, y_pred)
```

In [17]:
```python
# Repeating task 1 - 35 times
train_scores = []
oob_scores = []
for i in tqdm(range(35)):
  list_input_data =[]
  list_output_data =[]
  list_selected_row= []
  list_selected_columns=[]
  # Generating samples
  for j in range(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)

  # Building models
  list_of_all_models = get_models(list_input_data, list_output_data)
  # Getting train predictions
  train_preds = get_train_predictions(x, list_selected_columns, list_of_all_models)
  # MSE train score
  train_scores.append(get_mse_score(y, train_preds))
  # Getting oob predictions
  oob_preds = get_oob_predictions(x, list_selected_columns, list_selected_row, list_of_all_models)
  # Mean oob score
  oob_scores.append(get_mse_score(y, oob_preds))
```

```
100%|██████████| 35/35 [01:06<00:00,  1.89s/it]
```

In [18]:
```python
print("the length of train scores obtained: ",len(train_scores))

print("the length of oob scores obtained", len(oob_scores) )
```

```
the length of train scores obtained:  35
the length of oob scores obtained 35
```

95% Confidence interval of train scores

In [20]:
```python
train_scores_sample_mean = np.mean(np.asarray(train_scores))
train_scores_std_dev = np.std(np.asarray(train_scores))
sample_size = 35
print('Upper Limit:', train_scores_sample_mean + (train_scores_std_dev/np.sqrt(sample_size)) * 1.96)
print('Lower Limit:', train_scores_sample_mean - (train_scores_std_dev/np.sqrt(sample_size)) * 1.96)
```

```
Upper Limit: 0.20984529010145678
Lower Limit: 0.08443793529827884
```

95% Confidence interval of OOB scores

In [21]:
```python
oob_scores_sample_mean = np.mean(np.asarray(oob_scores))
oob_scores_std_dev = np.std(np.asarray(oob_scores))
sample_size = 35
print('Upper Limit:', oob_scores_sample_mean + (oob_scores_std_dev/np.sqrt(sample_size)) * 1.96)
print('Lower Limit:', oob_scores_sample_mean - (oob_scores_std_dev/np.sqrt(sample_size)) * 1.96)
```

```
Upper Limit: 15.261097143292206
Lower Limit: 13.951614364449648
```

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 [39]:
```python
def predict(list_of_all_models_task1,  list_selected_columns, xq):
  pred = []
  for i in range(30):
    data_point = xq[list_selected_columns[i]]
    pred.extend(list_of_all_models_task1[i].predict([data_point]))
  return np.median(np.asarray(pred))
```

In [40]:
```python
xq = x[0]
predict(list_of_all_models_task1, list_selected_columns_task1, xq)
```

Out[40]:
```
24.0
```

In [41]:
```python
y[0]
```

Out[41]:
```
24.0
```

We see that the predicted value is equal to the actual value.

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

1. **OOB scores is more than train score since it uses models that has not seen that data point for prediction**
2. **Population mean can be calculated when the population std is not given using the sample mean and Standard error.**
3. **Random sampling of columns leads to have more generic models.**

In [ ]: