Bootstrap_assignment_Solution

April 29, 2022

1 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

[1]:
```
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
import pandas as pd
import random
from sklearn.tree import DecisionTreeRegressor
from operator import add
from scipy.sparse import hstack
```

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

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

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

[4]:
```
y.shape
```

[4]:
```
(506,)
```

[5]:
```
x[:2]
```

[5]:
```
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]])
```

[6]:
```
y[:10]
```

[6]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9])
```

1.1 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.

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]
- Write code for generating samples

[7]:
```
def generating_samples(input_data, target_data):

    '''In this function, we will write code for generating 30 samples '''

    index = []
    rows = np.random.choice(len(input_data), size=303, replace=False) #␣
↪generating 60% of points randomly without replacement
    s = np.random.choice(rows, size=203, replace=True) # generating 40% of␣
↪points randomly from rows with replacement
    rs = np.hstack((rows,s))                    # combining them
    x = input_data[rs]
```

```
    no_of_columns = random.randint(3, 13)      # number of columns in each␣
↪sample.
    columns = np.array(random.sample(range(0, 13), no_of_columns)) #randomy␣
↪selecting the columns based on random sample generator
    x_mod = x[:, columns]
    len(x_mod)
    y_mod = target_data[rs]

    return x_mod,y_mod,rows,columns
```

Grader function - 1

[8]:
```
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)
```

[8]:
```
True
```

- 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.

[9]:
```
# 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

[10]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

[10]:
```
True
```

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 $y^i_{pred} = \frac{1}{30}\sum_{k=1}^{30}$(predicted value of x^i with k^{th} model)
- Now calculate the $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$
- Write code for building regression trees

[11]:
```
all_DT_models = []

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

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

[12]:
```
all_pred_y = []

for i in range(0, 30):

    data = x[:, list_selected_columns[i]]
    pred_y = all_DT_models[i].predict(data)
    all_pred_y.append(pred_y)

all_pred_y = np.median(all_pred_y,axis=0)
mse = mean_squared_error(y, all_pred_y)
print("The MSE is : ", mse)
```

```
The MSE is :  0.6018864130434789
```

Step - 3

- Predicted house price of i^{th} data point $y^i_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^i}$(predicted value
- Now calculate the $OOBScore = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$.
- Write code for calculating OOB score

[13]:
```
all_pred_oob = []
score = 0

for i in range(0, 506):      # Total number of rows
    oob_indices = []

    # For each of i build a sample of size 30 which should not be part of the␣
 ↪list_selected_row[i]
    # For eg, say for i=230 and j in the loop is 5 then list_selected_row[5]␣
 ↪should not contain the 230th row

    for j in range(0, 30):
        if i not in list_selected_row[j]:
            oob_indices.append(j)

    pred_oob = []

    for index in oob_indices:
        model_o = all_DT_models[index]

        oob_row = x[i]

        # extracting ONLY the columns that are selected during the bootstrapping
        oob_x = [oob_row[col] for col in list_selected_columns[index]]
        oob_x = np.array(oob_x).reshape(1, -1)

        y_pred_oob = model_o.predict(oob_x)
        pred_oob.append(y_pred_oob)

    pred_oob = np.array(pred_oob)
    pred_oob = np.median(pred_oob)

    all_pred_oob.append(pred_oob)

    score += ((y[i] - all_pred_oob[i] ) ** 2)


oob_score = score/506
print("The OOB score is : ", oob_score)
```

```
The OOB score is :  14.512286723254022
```

3 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

[14]:
```
def calculate_mse_obb(x, y):

    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)

    all_DT_models = []

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

    all_pred_y = []

    for i in range(0, 30):

        data = x[:, list_selected_columns[i]]
        pred_y = all_DT_models[i].predict(data)
        all_pred_y.append(pred_y)

    all_pred_y = np.median(all_pred_y,axis=0)
    mse = mean_squared_error(y, all_pred_y)       # MSE calculation


    # OOB Score calculation

    all_pred_oob = []
```

```
    score = 0

    for i in range(0, 506):      # Total number of rows
        oob_indices = []

        # For each of i build a sample of size 30 which should not be part of␣
↪the list_selected_row[i]
        # For eg, say for i=230 and j in the loop is 5 then␣
↪list_selected_row[5] should not contain the 230th row

        for j in range(0, 30):
            if i not in list_selected_row[j]:
                oob_indices.append(j)

        pred_oob = []

        for index in oob_indices:
            model_o = all_DT_models[index]

            oob_row = x[i]

            # extracting ONLY the columns that are selected during the␣
↪bootstrapping
            oob_x = [oob_row[col] for col in list_selected_columns[index]]
            oob_x = np.array(oob_x).reshape(1, -1)

            y_pred_oob = model_o.predict(oob_x)
            pred_oob.append(y_pred_oob)

        pred_oob = np.array(pred_oob)
        pred_oob = np.median(pred_oob)

        all_pred_oob.append(pred_oob)

        score += ((y[i] - all_pred_oob[i] ) ** 2)


    oob_score = score/506

    return mse,oob_score
```

[15]:
```
mse_list = []
oob_list = []

for i in range(0,35):
    mse,oob = calculate_mse_obb(x, y)
    mse_list.append(mse)
```

```
    oob_list.append(oob)
```

[16]:
```
print("MSE of 35 models : \n",mse_list)
print("\nOOB score of 35 models : \n",oob_list)
```

```
MSE of 35 models :
 [0.05437747035573131, 0.037040513833992074, 0.042792292490118546,
0.050514382960035124, 0.24589856139901428, 0.17584705753184013,
0.21595479249011867, 0.1933448616600789, 0.23908116490996933,
0.030254446640316183, 0.07626736968873526, 0.14677371541501977,
0.02986660079051389, 0.0750944224857269, 0.011828063241106732,
0.1547858418071692, 0.04511363636363635, 0.036931818181818184,
0.02707509881422929, 0.1567037353845066, 0.1408975898161941,
0.009550395256917015, 0.1015374687424376, 0.10088438735177874,
0.028890118577075116, 0.07846343873517794, 0.033116506246035224,
0.026932213438735194, 0.013942138779095333, 0.04849802371541504,
1.0406077075098816, 0.15320707070707063, 0.10567087176108925,
0.16582724473462943, 0.047840909090909094]

OOB score of 35 models :
 [11.57631916996048, 14.491279644268776, 14.355757164031623, 13.200539714421545,
13.533518929004776, 13.291527936534273, 12.942057556336783, 14.06044960474308,
15.118338944280367, 12.874036561264818, 15.743393869721677, 16.863376703173042,
11.496482213438739, 11.02156311041275, 14.886506841821053, 14.05547157899689,
14.623165386072765, 13.67869994408908, 12.425323616600789, 12.322736774322893,
13.310495427114144, 11.475884387351776, 14.394875562068847, 16.21000494071146,
11.475394466403156, 12.436647371931281, 11.885841037707651, 15.928790830699924,
13.85324575856858, 14.935128458498026, 18.601986880324162, 10.920287796442675,
15.313497474747484, 15.92754227439748, 14.610832784788983]
```

[17]:
```
import scipy

con_level = 0.95            # As mentioned in Central_Limit_theorem.ipynb
deg_of_freedom = 34         # deg_of_freedom = sample size-1

# reference = https://www.adamsmith.haus/python/answers/
 ↪how-to-compute-the-confidence-interval-of-a-sample-statistic-in-python
# Confidence interval for MSE

mean = np.mean(mse_list)
std_err = scipy.stats.sem(mse_list)

CI_MSE = scipy.stats.t.interval(con_level, deg_of_freedom, mean, std_err)
print("CI for MSE : ",CI_MSE)

# Confidence interval for OOB
```

```
mean_oob = np.mean(oob_list)
std_err_oob = scipy.stats.sem(oob_list)

CI_OOB = scipy.stats.t.interval(con_level, deg_of_freedom, mean_oob,␣
 ↪std_err_oob)
print("CI for OOB : ",CI_OOB)
```

```
CI for MSE :  (0.058307788771814355, 0.17834432156567812)
CI for OOB :  (13.207911905567054, 14.440145278161618)
```

4 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.

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.

- Write code for TASK 3

[18]:
```
def predict_house_price(x_q):

    y_pred_all = []

    for i in range(0, 30):
        model1 = all_DT_models[i]

        # Extract x for ith data point with specific number of featues from␣
 ↪list_selected_columns

        x = [x_q[col] for col in list_selected_columns[i]]
        x = np.array(x).reshape(1, -1)
        y_pred = model1.predict(x)
        y_pred_all.append(y_pred)

    y_pred_median = np.median(y_pred_all)

    return y_pred_median


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]
y_q = predict_house_price(xq)
print("House Price for query point xq is : ",y_q)

xq1 = [i*2 for i in xq]
y_q1 = predict_house_price(xq1)
```

```
print("House Price for query point xq1 is : ",y_q1)
```

```
House Price for query point xq is :  18.5
House Price for query point xq1 is :  21.35
```

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

Task 1 :

- We do row sampling & column sampling to reduce the variance and to build strong models.
- We created High Variance models of `DecisionTreeRegressor` by keeping `max_depth = None`
- OOB Score is the error on samples that were not seen during the training.
- OOB Scoring is very useful when dataset is small and thereby when splitted into training and validation set - will result in loss of useful data that otherwise could have been used for training the models. Hence in this case, we decide to extract some of the training data as the validation set by using only those data-points that were not used for training a particular sample-set.
- We got the MSE score between 0 & 10 and OOB score between 10 and 35 and also MSE is less than OOB score.

Task 2 :

- By definition we know the interpretation of a 95% confidence interval for the population mean as - If repeated random samples were taken and the 95% confidence interval was computed for each sample, 95% of the intervals would contain the population mean.
- Therefore in our case:
  - MSE - There is a 95% chance that the confidence interval of (0.058307788771814355, 0.17834432156567812) contains the true population mean of MSE.
  - OOB Score - There is a 95% chance that the confidence interval of (13.207911905567054, 14.440145278161618) contains the true population mean of OOB Score.

Task 3 :

- Our goal through Bagging is to reduce the variance of the final model maintaining the same Bias
- As we can see for xq we got the output as 18.5.
- We can see all the elements in xq1 is 1.5 times the xq, since we got a low variance model the output we got is 21.35