parth.pandey13103447@gmail.com_12

August 10, 2020

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

[43]:
```
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 sklearn.tree import DecisionTreeRegressor
```

[44]:
```
boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable
```

[45]:
```
x.shape
```

[45]:
```
(506, 13)
```

1.1 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

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 $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$

Step - 3

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

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

3 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.

4 Task - 1

[46]:
```
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


    # return sampled_input_data ,␣
↪sampled_target_data,selected_rows,selected_columns
```

```
    #note please return as lists

    # Random Rows Generation
    rows, cols = input_data.shape
    selecting_indices = np.random.choice(rows ,int(np.floor(rows *0.6)),␣
↪replace=False)
    replicating_indices = np.random.choice(selecting_indices, rows -␣
↪selecting_indices.shape[0], replace=False)

    # Random Columns Generation
    selecting_cols = np.random.choice(cols, np.random.randint(3,cols))

    sample_data = input_data[selecting_indices[:,None], selecting_cols]
    sample_target_data = target_data[selecting_indices]

    replicated_data = input_data[replicating_indices[:,None], selecting_cols]
    replicated_target_data = target_data[replicating_indices]

    final_sampled_data = np.vstack((sample_data, replicated_data))
    final_target_data  = np.vstack((sample_target_data.reshape(-1,1),␣
↪replicated_target_data.reshape(-1,1)))

    return final_sampled_data, final_target_data, selecting_indices,␣
↪selecting_cols
a,b,c,d=generating_samples(x,y)
```

Grader function - 1

[47]:
```
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)
```

[47]:
```
True
```

Grader function - 2

[48]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

[48]:
```
True
```

Step - 2

[49]:
```
model = {}
for ind, train_data in enumerate(zip(list_input_data,list_output_data)):
    x_train = train_data[0]
    y_train = train_data[1]
    temp_model=DecisionTreeRegressor(max_depth=None)
    temp_model.fit(x_train, y_train)
    model[ind] = temp_model
```

- Write code for calculating MSE

[50]:
```
pred = {}
for key, val in model.items():
    pred[key] = val.predict(x[:,list_selected_columns[key]])

y_pred = np.median(np.array(list(pred.values())).T, axis=1)

mse = mean_squared_error(y, y_pred)
print('MSE =',mse)
```

```
MSE = 0.3072203818972466
```

4.1 Step - 3

- Write code for calculating OOB score

[51]:
```
oob_pred = {}
for i, ele in enumerate(zip(x,y)):
    x_train = ele[0]
    y_train = ele[1]
    temp = {}
    for ind, sample in enumerate(list_input_data):
        if x_train[list_selected_columns[ind]] in sample:
            temp[ind] = model[ind].predict(x_train[list_selected_columns[ind]].
 ↪reshape(1,-1))[0]
    oob_pred[i] = np.median( np.array( list( temp.values())).reshape(-1,1))

oob_score = ( np.sum( np.square( y - np.array(list( oob_pred.values())))))/(x.
 ↪shape[0])
print(oob_score)
```

```
0.30721544118578414
```

5 Task 2

[52]:
```
train_mse = []
oob_score = []

for _ in range(35):
    list_input_data =[]
    list_output_data =[]
    list_selected_row= []
    list_selected_columns=[]
    # Row and column sampling
    for _ 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)
    # Creating Models
    model = {}
    for ind, train_data in enumerate(zip(list_input_data,list_output_data)):
        x_train = train_data[0]
        y_train = train_data[1]
        temp_model=DecisionTreeRegressor(max_depth=None)
        temp_model.fit(x_train, y_train)
        model[ind] = temp_model

    # Calculating MSE
    pred = {}
    for key, val in model.items():
        pred[key] = val.predict(x[:,list_selected_columns[key]])

    y_pred = np.median(np.array(list(pred.values())).T, axis=1)

    train_mse.append(mean_squared_error(y, y_pred))
    # Calculating OOB Score
    oob_pred = {}
    for i, ele in enumerate(zip(x,y)):
        x_train = ele[0]
        y_train = ele[1]
        temp = {}
        for ind, sample in enumerate(list_input_data):
            if x_train[list_selected_columns[ind]] in sample:
                temp[ind] = model[ind].
↪predict(x_train[list_selected_columns[ind]].reshape(1,-1))[0]
        oob_pred[i] = np.median( np.array( list( temp.values())).reshape(-1,1))
```

```
    oob_score.append(( np.sum( np.square( y - np.array(list( oob_pred.
 ↪values())))))/(x.shape[0]))


# Calculating Confidence Interval

sample_size = 30
mse_random_sample = np.random.choice(train_mse, sample_size)
oob_random_sample = np.random.choice(oob_score, sample_size)

mean_mse = np.mean(mse_random_sample)
mean_oob = np.mean(oob_random_sample)

std_mse = np.std(mse_random_sample)
std_oob = np.std(oob_random_sample)

print('Confidence Interval with 95% Confidence')
print('Confidence Interval for MSE')
mse_ci = (mean_mse-(2 * (std_mse))/np.sqrt(sample_size), mean_mse+(2 *␣
 ↪(std_mse))/np.sqrt(sample_size))
print('[{}, {}]'.format(mse_ci[0], mse_ci[1]))
print('Confidence Interval for OOB')
oob_ci = (mean_oob-(2 * (std_oob))/np.sqrt(sample_size), mean_oob+(2 *␣
 ↪(std_oob))/np.sqrt(sample_size))
print('[{}, {}]'.format(oob_ci[0], oob_ci[1]))
```

```
Confidence Interval with 95% Confidence
Confidence Interval for MSE
[0.05825802621418374, 0.16447415855318834]
Confidence Interval for OOB
[0.04679047622597019, 0.11150099825003806]
```

6 Task 3

- Write code for TASK 3

[53]:
```
query_point =  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] ).reshape(1,-1)

list_input_data =[]
list_output_data =[]
list_selected_row= []
list_selected_columns=[]
# Row and column sampling
for _ 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)
# Creating Models
model = {}
for ind, train_data in enumerate(zip(list_input_data,list_output_data)):
    x_train = train_data[0]
    y_train = train_data[1]
    temp_model=DecisionTreeRegressor(max_depth=None)
    temp_model.fit(x_train, y_train)
    model[ind] = temp_model
# Making Predictions
pred = {}
for key, val in model.items():

    pred[key] = val.predict(query_point[:,list_selected_columns[key]])

y_pred = np.median(np.array(list(pred.values())).T, axis=1)
print('For query point {} model gave {} as the regression output'.
 ↪format(query_point[0], y_pred))
```

```
For query point [1.8000e-01 2.0000e+01 5.0000e+00 0.0000e+00 4.2100e-01
5.6000e+00
 7.2200e+01 7.9500e+00 7.0000e+00 3.0000e+01 1.9100e+01 3.7213e+02
 1.8600e+01] model gave [18.5] as the regression output
```

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

- **Task 1**
  - The number of Rows and Columns are kept same of creating the model by randomly sampling some rows and columns and then repeating the sampled columns.
  - The information about the sampled columns are kept and transfered in the algorithm for training and prediction of the model created for each set of sampled data.
  - The OOB Score is like a CV score for the model
- **Task 2**
  - Task 2 is simply repeating the Task 1 35 times.
  - Observations are used to report 95% Confidence Interval
  - Confidence Interval OOB and MSE should overlap as we want them as close as possible
- **Task 3**
  - Simply check how model is performing and create a final ensemble model and get results