DT on boston dataset

April 23, 2021

[1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
import random
from sklearn.tree import DecisionTreeRegressor
from prettytable import PrettyTable
```

[2]:
```
boston = load_boston()
x=boston.data
y=boston.target
```

[3]:
```
df = pd.DataFrame(x, columns=boston.feature_names)
df['PRICE'] = y

no_of_total_columns = len(df.columns) - 1
df.head()
```

[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

[4]:
```
def generating_samples(x, y):

    # random rows 303, keep it
    selecting_rows = sorted(np.random.choice(a=506, size=303, replace=False),␣
↪reverse=False)
```

```
    # randomly choose columns 3<= col <=13
    rnd_number = random.randint(3, 13)
    selecting_cols = sorted(np.random.choice(a=13, size=rnd_number,␣
↪replace=False), reverse=False)


    # Preparing Data

    # selecting_row_ and selecting_cols data
    sample_data = x[selecting_rows,:]
    sample_data = sample_data[:, selecting_cols]
    sample_target = y[selecting_rows]

    # replaced_row data
    # random 203 points from 303
    replacing_rows = sorted(np.random.choice(a=303, size=203, replace=False),␣
↪reverse=False)
    replaced_data = sample_data[replacing_rows]
    replaced_target = sample_target[replacing_rows]

    # stacking selected_data and replaced_data
    final_sample_data = np.vstack((sample_data, replaced_data))
    final_sample_target = np.vstack((sample_target.reshape(-1, 1),␣
↪replaced_target.reshape(-1, 1)))


    # return sampled data_points, sampled target data, sampled_row indexs,␣
↪sampled_column index

    return final_sample_data, final_sample_target, selecting_rows,␣
↪selecting_cols
```

[5]:
```
final_sample_data, final_sample_target, selecting_rows, selecting_cols =␣
↪generating_samples(x, y)
```

[6]:
```
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)
```

[6]: True

1 Task -1 (Creating 30 samples)

[7]:
```
list_of_input_data = []
list_of_output_data = []
list_of_selected_rows = []
list_of_selected_cols = []

for i in range(30):

    final_sample_data, final_sample_target, selecting_rows, selecting_cols =␣
 ↪generating_samples(x, y)

    list_of_input_data.append(final_sample_data)
    list_of_output_data.append(final_sample_target)
    list_of_selected_rows.append(selecting_rows)
    list_of_selected_cols.append(selecting_cols)




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

grader_30(list_of_input_data)
```

[7]: True

2 Storing 30 models

[8]:
```
list_of_models = []

for input_data, output_data in zip(list_of_input_data, list_of_output_data):
  model = DecisionTreeRegressor(max_depth=None)
  model.fit(input_data, output_data)
  list_of_models.append(model)
```

```
print(len(list_of_models))
```

```
30
```

3 Calculating MSE

[9]:
```
y_pred_all_models = []

for col_sampling, model in zip(list_of_selected_cols, list_of_models):

    x_input = x[:, col_sampling]
    y_pred_all_models.append(model.predict(x_input).reshape(-1, 1))


y_pred = np.around(np.divide(np.sum(y_pred_all_models, axis=0), 30), 1)

y = y.reshape(-1, 1)
y_pred.shape, y.shape, list_of_input_data[0].shape
```

[9]:
```
((506, 1), (506, 1), (506, 6))
```

[10]:
```
mse = np.square(np.subtract(y, y_pred)).mean()
mse
```

[10]:
```
2.3622332015810272
```

4 Calculating OOB Score

[20]:
```
oob_y_pred = []

for data_index, data_row in enumerate(x):

    temp_y_pred = []

    for selected_rows_index, selected_col_index, model in␣
 ↪zip(list_of_selected_rows, list_of_selected_cols, list_of_models):

        if data_index not in selected_rows_index:

            # do column sampling
            x_input = np.array(data_row).reshape(1, -1)[:, selected_col_index]


            # predict push to temp_y_pred
            temp_y_pred.append(model.predict(x_input))
```

```
    # do calculataion and push to oob_y_pred
    y_pred = np.median(np.asarray(temp_y_pred))

    oob_y_pred.append(y_pred)



oob_y_pred = np.array(oob_y_pred).reshape(-1, 1)

print(oob_y_pred.shape)
print(y.shape)
print("true: {}, pred: {}".format(y[0], oob_y_pred[0]))
```

```
(506, 1)
(506, 1)
true: [24.], pred: [31.95]
```

[21]:
```
oob_score = np.square(np.subtract(oob_y_pred, y)).mean()
oob_score
```

[21]:
```
14.016645223030086
```

5 Train MSE Scores

[22]:
```
train_mse_scores = []

for i in range(35):

    # sampling

    list_of_input_data = []
    list_of_output_data = []
    list_of_selected_rows = []
    list_of_selected_cols = []

    for i in range(30):

        final_sample_data, final_sample_target, selecting_rows, selecting_cols␣
↪= generating_samples(x, y)

        list_of_input_data.append(final_sample_data)
        list_of_output_data.append(final_sample_target)
        list_of_selected_rows.append(selecting_rows)
        list_of_selected_cols.append(selecting_cols)
```

```
    # List of models

    list_of_models = []

    for input_data, output_data in zip(list_of_input_data, list_of_output_data):

        model = DecisionTreeRegressor(max_depth=None)
        model.fit(input_data, output_data)
        list_of_models.append(model)

    # Predicting and calculating MSE


    y_pred_all_models = []

    for col_sampling, model in zip(list_of_selected_cols, list_of_models):

        x_input = x[:, col_sampling]
        y_pred_all_models.append(model.predict(x_input).reshape(-1, 1))


    y_pred = np.around(np.divide(np.sum(y_pred_all_models, axis=0), 30), 1)

    mse = np.square(np.subtract(y, y_pred)).mean()

    #Stored in a list
    train_mse_scores.append(mse)
```

[14]:
```
print(len(train_mse_scores))
print(train_mse_scores)
```

```
35
[2.8683399209486167, 2.155790513833992, 2.7674901185770744, 2.1426086956521737,
2.1098616600790514, 2.792351778656126, 2.1528063241106716, 2.4678260869565216,
1.9978063241106718, 2.030098814229249, 2.4497628458498024, 2.873458498023715,
2.2759486166007896, 2.1954150197628457, 2.056383399209486, 3.0770750988142295,
2.886324110671936, 2.5661462450592882, 2.1811264822134384, 1.9558102766798422,
1.903102766798419, 1.975039525691699, 2.7585770750988137, 2.058300395256917,
2.393636363636363, 2.2669367588932805, 2.1659486166007906, 2.1801778656126483,
2.181916996047431, 2.1927075098814233, 2.4847430830039525, 2.365415019762845,
2.1672529644268774, 2.9212845849802362, 2.144802371541502]
```

6 OOB Scores

[29]:
```
test_oob_scores = []

for i in range(35):

    # sampling

    list_of_input_data = []
    list_of_output_data = []
    list_of_selected_rows = []
    list_of_selected_cols = []

    for i in range(30):

        final_sample_data, final_sample_target, selecting_rows, selecting_cols␣
↪= generating_samples(x, y)

        list_of_input_data.append(final_sample_data)
        list_of_output_data.append(final_sample_target)
        list_of_selected_rows.append(selecting_rows)
        list_of_selected_cols.append(selecting_cols)

    # List of models

    list_of_models = []

    for input_data, output_data in zip(list_of_input_data, list_of_output_data):

        model = DecisionTreeRegressor(max_depth=None)
        model.fit(input_data, output_data)
        list_of_models.append(model)

    # Predicting and calculating OOB

    oob_y_pred = []

    for data_index, data_row in enumerate(x):

        temp_y_pred = []

        for selected_rows_index, selected_col_index, model in␣
↪zip(list_of_selected_rows, list_of_selected_cols, list_of_models):

            if data_index not in selected_rows_index:

                # do column sampling
```

```
                x_input = np.array(data_row).reshape(1, -1)[:,␣
↪selected_col_index]


                # predict push to temp_y_pred
                temp_y_pred.append(model.predict(x_input))


        # do calculataion and push to oob_y_pred
        y_pred = np.median(np.asarray(temp_y_pred))

        oob_y_pred.append(y_pred)



    oob_y_pred = np.array(oob_y_pred).reshape(-1, 1)

    oob_score = np.square(np.subtract(oob_y_pred, y)).mean()

    #Stored in a list
    test_oob_scores.append(oob_score)
```

[30]:
```
print(len(test_oob_scores))
print(test_oob_scores)
```

```
35
[14.337240146906693, 14.787600217508244, 13.594189723320158, 14.80823740118577,
14.723413916564132, 14.007485704874835, 13.388359683794466, 17.450867988871654,
15.38008166431686, 12.239359295277827, 14.263373164683083, 12.384036807740223,
18.182400221849957, 13.205228782290394, 14.705339117994498, 13.74955533596838,
14.562552170820874, 15.344541749011858, 14.014589787474696, 14.65601909012268,
15.006899436383577, 11.434801129425535, 14.394506779924697, 16.228079161176986,
13.42788098375055, 12.524988123124661, 12.155026124776606, 13.67620937637242,
15.818227903926172, 14.148537989529608, 12.501847277119015, 19.897433064891807,
14.624507707509881, 13.771521739130433, 13.860377327115085]
```

7 Computing CI of OOB Score and Train MSE

[31]:
```
x = PrettyTable()
x = PrettyTable(["#samples", "Sample Size", "MSE", "Left C.I MSE", "Right C.I␣
↪MSE", "MSE Catch", "OOB", "Left C.I OOB", "Right C.I OOB", "OOB Catch"])

count = 0
```

```
for mse, oob in zip(train_mse_scores, test_oob_scores):

    mse = np.round(mse, 3)
    oob = np.round(oob, 3)


    standard_error_mse = np.sqrt(np.divide((2*mse), 506))
    ci_mse_left = np.round((mse - standard_error_mse), 3)
    ci_mse_right = np.round((mse + standard_error_mse), 3)

    standard_error_oob = np.sqrt(np.divide((2*oob), 506))
    ci_oob_left = np.round((oob - standard_error_oob), 3)
    ci_oob_right = np.round((oob + standard_error_oob), 3)

    row = []
    row.append(i+1)
    row.append("506")
    row.append(mse)
    row.append(ci_mse_left)
    row.append(ci_mse_right)
    row.append((mse >= ci_mse_left) and (mse <= ci_mse_right))
    row.append(oob)
    row.append(ci_oob_left)
    row.append(ci_oob_right)
    row.append((oob >= ci_oob_left) and (oob <= ci_oob_right))

    x.add_row(row)

    count += 1


print(x)
```

```
+----------+-------------+-------+-------------+---------------+-----------+---
-----+--------------+---------------+-----------+
| #samples | Sample Size |  MSE  | Left C.I MSE | Right C.I MSE | MSE Catch |
OOB   | Left C.I OOB | Right C.I OOB | OOB Catch |
+----------+-------------+-------+-------------+---------------+-----------+---
-----+--------------+---------------+-----------+
|    30    |     506     | 2.217 |    2.123     |     2.311     |    True   |
14.337 |    14.099    |     14.575    |    True   |
|    30    |     506     |  2.94 |    2.832     |     3.048     |    True   |
14.788 |    14.546    |     15.03     |    True   |
|    30    |     506     | 2.661 |    2.558     |     2.764     |    True   |
13.594 |    13.362    |     13.826    |    True   |
|    30    |     506     | 2.632 |     2.53     |     2.734     |    True   |
14.808 |    14.566    |     15.05     |    True   |
```

| 30 | 506 | 2.54 | 2.44 | 2.64 | True |
14.723 | 14.482 | 14.964 | True |
| 30 | 506 | 2.355 | 2.259 | 2.451 | True |
14.007 | 13.772 | 14.242 | True |
| 30 | 506 | 2.567 | 2.466 | 2.668 | True |
13.388 | 13.158 | 13.618 | True |
| 30 | 506 | 3.351 | 3.236 | 3.466 | True |
17.451 | 17.188 | 17.714 | True |
| 30 | 506 | 3.001 | 2.892 | 3.11 | True |
15.38 | 15.133 | 15.627 | True |
| 30 | 506 | 2.123 | 2.031 | 2.215 | True |
12.239 | 12.019 | 12.459 | True |
| 30 | 506 | 1.99 | 1.901 | 2.079 | True |
14.263 | 14.026 | 14.5 | True |
| 30 | 506 | 2.436 | 2.338 | 2.534 | True |
12.384 | 12.163 | 12.605 | True |
| 30 | 506 | 2.822 | 2.716 | 2.928 | True |
18.182 | 17.914 | 18.45 | True |
| 30 | 506 | 2.489 | 2.39 | 2.588 | True |
13.205 | 12.977 | 13.433 | True |
| 30 | 506 | 2.364 | 2.267 | 2.461 | True |
14.705 | 14.464 | 14.946 | True |
| 30 | 506 | 2.185 | 2.092 | 2.278 | True |
13.75 | 13.517 | 13.983 | True |
| 30 | 506 | 2.517 | 2.417 | 2.617 | True |
14.563 | 14.323 | 14.803 | True |
| 30 | 506 | 2.339 | 2.243 | 2.435 | True |
15.345 | 15.099 | 15.591 | True |
| 30 | 506 | 2.724 | 2.62 | 2.828 | True |
14.015 | 13.78 | 14.25 | True |
| 30 | 506 | 2.311 | 2.215 | 2.407 | True |
14.656 | 14.415 | 14.897 | True |
| 30 | 506 | 2.253 | 2.159 | 2.347 | True |
15.007 | 14.763 | 15.251 | True |
| 30 | 506 | 2.747 | 2.643 | 2.851 | True |
11.435 | 11.222 | 11.648 | True |
| 30 | 506 | 2.246 | 2.152 | 2.34 | True |
14.395 | 14.156 | 14.634 | True |
| 30 | 506 | 2.047 | 1.957 | 2.137 | True |
16.228 | 15.975 | 16.481 | True |
| 30 | 506 | 2.241 | 2.147 | 2.335 | True |
13.428 | 13.198 | 13.658 | True |
| 30 | 506 | 2.277 | 2.182 | 2.372 | True |
12.525 | 12.303 | 12.747 | True |
| 30 | 506 | 2.621 | 2.519 | 2.723 | True |
12.155 | 11.936 | 12.374 | True |
| 30 | 506 | 3.048 | 2.938 | 3.158 | True |
13.676 | 13.444 | 13.908 | True |

```
|    30    |     506     |  2.74 |    2.636     |     2.844     |    True   |
15.818 |    15.568    |     16.068    |    True   |
|    30    |     506     | 3.007 |    2.898     |     3.116     |    True   |
14.149 |    13.913    |     14.385    |    True   |
|    30    |     506     | 2.302 |    2.207     |     2.397     |    True   |
12.502 |    12.28     |     12.724    |    True   |
|    30    |     506     | 2.584 |    2.483     |     2.685     |    True   |
19.897 |    19.617    |     20.177    |    True   |
|    30    |     506     | 2.132 |     2.04     |     2.224     |    True   |
14.625 |    14.385    |     14.865    |    True   |
|    30    |     506     | 2.746 |    2.642     |      2.85     |    True   |
13.772 |    13.539    |     14.005    |    True   |
|    30    |     506     | 2.005 |    1.916     |     2.094     |    True   |
13.86  |    13.626    |     14.094    |    True   |
+----------+-------------+-------+--------------+---------------+-----------+---
-----+--------------+---------------+-----------+
```

8 Task 3 (Predicting a data)

[42]:
```
x_q = [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]
x_q = np.asarray(x_q).reshape(1, -1)
y_pred_all_models = []


for col_sampling, model in zip(list_of_selected_cols, list_of_models):
    x_input = np.array(x_q).reshape(1, -1)[:, col_sampling]
    y_pred_all_models.append(model.predict(x_input))


y_pred = np.around(np.divide(np.sum(y_pred_all_models, axis=0), 30), 1)
```

[43]:
```
y_pred
```

[43]:
```
array([19.5])
```