# DT Regression + Classification

April 4, 2022

## 0.1 Decision Tree Regression Model

[1]:
```
#load the boston dataset
from sklearn.datasets import load_boston
```

[2]:
```
boston = load_boston()
```

```
/home/andropov/anaconda3/envs/imp/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is
deprecated; `load_boston` is deprecated in 1.0 and will be 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)
```

[3]:
```python
import pandas as pd
```

[4]:
```python
data=pd.DataFrame(boston.data, columns=boston.feature_names)
```

[5]:
```python
#column of Median Value is usually the target -
#to be predicted by regression model
data['MEDV']=pd.DataFrame(boston.target)
```

[6]:
```python
data
```

[6]:
```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
..       …    …     …    …     …      …     …       …  …      …
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741   0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       …       …    …     …
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

[7]:
```
#find all correlation values for all features,
#RM, ZN have highest with target MEDV, so select RM and Zone for training
pd.DataFrame(data.corr().round(2))
```

[7]:
```
          CRIM    ZN  INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX  PTRATIO  \
CRIM      1.00 -0.20   0.41 -0.06  0.42 -0.22  0.35 -0.38  0.63  0.58     0.29
ZN       -0.20  1.00  -0.53 -0.04 -0.52  0.31 -0.57  0.66 -0.31 -0.31    -0.39
INDUS     0.41 -0.53   1.00  0.06  0.76 -0.39  0.64 -0.71  0.60  0.72     0.38
CHAS     -0.06 -0.04   0.06  1.00  0.09  0.09  0.09 -0.10 -0.01 -0.04    -0.12
NOX       0.42 -0.52   0.76  0.09  1.00 -0.30  0.73 -0.77  0.61  0.67     0.19
RM       -0.22  0.31  -0.39  0.09 -0.30  1.00 -0.24  0.21 -0.21 -0.29    -0.36
AGE       0.35 -0.57   0.64  0.09  0.73 -0.24  1.00 -0.75  0.46  0.51     0.26
DIS      -0.38  0.66  -0.71 -0.10 -0.77  0.21 -0.75  1.00 -0.49 -0.53    -0.23
RAD       0.63 -0.31   0.60 -0.01  0.61 -0.21  0.46 -0.49  1.00  0.91     0.46
TAX       0.58 -0.31   0.72 -0.04  0.67 -0.29  0.51 -0.53  0.91  1.00     0.46
PTRATIO   0.29 -0.39   0.38 -0.12  0.19 -0.36  0.26 -0.23  0.46  0.46     1.00
B        -0.39  0.18  -0.36  0.05 -0.38  0.13 -0.27  0.29 -0.44 -0.44    -0.18
LSTAT     0.46 -0.41   0.60 -0.05  0.59 -0.61  0.60 -0.50  0.49  0.54     0.37
MEDV     -0.39  0.36  -0.48  0.18 -0.43  0.70 -0.38  0.25 -0.38 -0.47    -0.51

             B  LSTAT  MEDV
CRIM     -0.39   0.46 -0.39
ZN        0.18  -0.41  0.36
INDUS    -0.36   0.60 -0.48
CHAS      0.05  -0.05  0.18
NOX      -0.38   0.59 -0.43
RM        0.13  -0.61  0.70
AGE      -0.27   0.60 -0.38
DIS       0.29  -0.50  0.25
RAD      -0.44   0.49 -0.38
TAX      -0.44   0.54 -0.47
PTRATIO -0.18   0.37 -0.51
B         1.00  -0.37  0.33
LSTAT    -0.37   1.00 -0.74
MEDV      0.33  -0.74  1.00
```

[ ]:
```

```

[8]:
```
# x corresponds to train data, note that multiple
#features are being included now
x=data[['RM','ZN']]
```

[9]:
```
x
```

[9]:
```
        RM    ZN
0    6.575  18.0
1    6.421   0.0
```

```
2    7.185   0.0
3    6.998   0.0
4    7.147   0.0
..     …    …
501  6.593   0.0
502  6.120   0.0
503  6.976   0.0
504  6.794   0.0
505  6.030   0.0

[506 rows x 2 columns]
```

[10]:
```
#y corresponds to labels
y=data['MEDV']
```

[11]:
```
y
```

[11]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       …
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64
```

[12]:
```
#import train_test_split module
from sklearn.model_selection import train_test_split
```

[13]:
```
#convert x and y to pandas Dataframes
x=pd.DataFrame(x)
y=pd.DataFrame(y)
```

[14]:
```
#split the dataset using train_test_split function,
#pass train data, labels, and test data ratio
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
```

[15]:
```
y_train
```

[15]:
```
     MEDV
250  24.4
265  22.8
345  17.5
```

```
55   35.4
488  15.2
..    …
166  50.0
211  19.3
273  35.2
66   19.4
79   20.3

[404 rows x 1 columns]
```

[16]:
```
from sklearn.tree import DecisionTreeRegressor
```

[17]:
```
dt1=DecisionTreeRegressor(max_depth=20)
```

[18]:
```
dt1.fit(x_train,y_train)
```

[18]:
```
DecisionTreeRegressor(max_depth=20)
```

[19]:
```
y_pred1=dt1.predict(x_test)
```

[20]:
```
import numpy as np
```

[21]:
```
#import the mean squared error module
from sklearn.metrics import mean_squared_error
```

[22]:
```
#calculate root mean square error
np.sqrt(mean_squared_error(y_test, y_pred1))
```

[22]:
```
7.318232000137953
```

[23]:
```
from sklearn.ensemble import RandomForestRegressor
```

[24]:
```
rf1=RandomForestRegressor()
```

[25]:
```
rf1.fit(x_train, y_train)
```

```
/home/andropov/anaconda3/envs/imp/lib/python3.7/site-
packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  """Entry point for launching an IPython kernel.
```

[25]:
```
RandomForestRegressor()
```

[26]:
```
rf1.score(x_test,y_test)
```

[26]: 0.47694278373608

[27]:
```
y_pred2=rf1.predict(x_test)
```

[28]:
```
#calculate root mean square error
np.sqrt(mean_squared_error(y_test, y_pred2))
```

[28]: 6.5951712867691805

[ ]:
```

```

## 0.2 Decision Tree Classifier

[29]:
```
from sklearn.tree import DecisionTreeClassifier
```

[30]:
```
from sklearn.datasets import load_digits
```

[31]:
```
digits = load_digits()
```

[ ]:
```

```

[36]:
```
#split the dataset using train_test_split function, pass train data, labels,␣
↪and test data ratio
x_train, x_test, y_train, y_test=train_test_split(
    digits.data,digits.target,test_size=0.25)
```

[37]:
```
dt2=DecisionTreeClassifier(criterion="entropy")
```

[38]:
```
dt2.fit(x_train, y_train)
```

[38]: DecisionTreeClassifier(criterion='entropy')

[39]:
```
dt2.score(x_test,y_test)
```

[39]: 0.8644444444444445

[40]:
```
# max_depth = longest path between root and leaf nodes
dt3=DecisionTreeClassifier(max_depth=30)
```

[41]:
```
dt3.fit(x_train, y_train)
```

[41]: DecisionTreeClassifier(max_depth=30)

[42]:
```
dt3.score(x_test,y_test)
```

[42]: 0.8422222222222222

[ ]:
```
Q. Using iterations, and the classification/regression models,
try to identify the optimum max_depth value
at which the following 3 models give the maximum R2 score.

1. Decision Tree Regressor
2. Random Forest Regressor
3. Decision Tree Classifier
```

[ ]:
```
```