Regression Trees

Estimated time needed: **20** minutes

In this lab you will learn how to implement regression trees using ScikitLearn. We will show what parameters are important, how to train a regression tree, and finally how to determine our regression trees accuracy.

Objectives

After completing this lab you will be able to:

- Train a Regression Tree
- Evaluate a Regression Trees Performance

Setup

For this lab, we are going to be using Python and several Python libraries. Some of these libraries might be installed in your lab environment or in SN Labs. Others may need to be installed by you. The cells below will install these libraries when executed.

In [1]:
```
# Install libraries not already in the environment using pip
#!pip install pandas==1.3.4
#!pip install sklearn==0.20.1
```

In [2]:
```
# Pandas will allow us to create a dataframe of the data so it can be used and manipula
import pandas as pd
# Regression Tree Algorithm
from sklearn.tree import DecisionTreeRegressor
# Split our data into a training and testing data
from sklearn.model_selection import train_test_split
```

About the Dataset

Imagine you are a data scientist working for a real estate company that is planning to invest in Boston real estate. You have collected information about various areas of Boston and are tasked with created a model that can predict the median price of houses for that area so it can be used to make offers.

The dataset had information on areas/towns not individual houses, the features are

CRIM: Crime per capita

ZN: Proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS: Proportion of non-retail business acres per town

CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

NOX: Nitric oxides concentration (parts per 10 million)

RM: Average number of rooms per dwelling

AGE: Proportion of owner-occupied units built prior to 1940

DIS: Weighted distances to five Boston employment centers

RAD: Index of accessibility to radial highways

TAX: Full-value property-tax rate per $10,000

PTRAIO: Pupil-teacher ratio by town

LSTAT: Percent lower status of the population

MEDV: Median value of owner-occupied homes in $1000s

Read the Data

Lets read in the data we have downloaded

In [3]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
```

In [4]:
```
data.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	NaN	36.2

Now lets learn about the size of our data, there are 506 rows and 13 columns

In [5]:
```
data.shape
```

Out[5]:
```
(506, 13)
```

Most of the data is valid, there are rows with missing values which we will deal with in pre-processing

In [6]:
```
data.isna().sum()
```

Out[6]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
LSTAT      20
MEDV        0
dtype: int64
```

Data Pre-Processing

First lets drop the rows with missing values because we have enough data in our dataset

In [7]:
```
data.dropna(inplace=True)
```

Now we can see our dataset has no missing values

In [8]:
```
data.isna().sum()
```

Out[8]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
LSTAT      0
MEDV       0
dtype: int64
```

Lets split the dataset into our features and what we are predicting (target)

In [9]:
```
X = data.drop(columns=["MEDV"])
Y = data["MEDV"]
```

In [10]:
```
X.head()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3	222	18.7	5.21

In [11]:
```
Y.head()
```

Out[11]:
```
0    24.0
1    21.6
2    34.7
3    33.4
5    28.7
Name: MEDV, dtype: float64
```

Finally lets split our data into a training and testing dataset using `train_test_split` from `sklearn.model_selection`

In [12]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.2, random_state=1)
```

Create Regression Tree

Regression Trees are implemented using `DecisionTreeRegressor` from `sklearn.tree`

The important parameters of `DecisionTreeRegressor` are

`criterion` : {"mse", "friedman_mse", "mae", "poisson"} - The function used to measure error

`max_depth` - The max depth the tree can be

`min_samples_split` - The minimum number of samples required to split a node

`min_samples_leaf` - The minimum number of samples that a leaf can contain

`max_features` : {"auto", "sqrt", "log2"} - The number of feature we examine looking for the best one, used to speed up training

First lets start by creating a `DecisionTreeRegressor` object, setting the `criterion` parameter to `mse` for Mean Squared Error

In [13]:
```
regression_tree = DecisionTreeRegressor(criterion = 'mse')
```

Training

Now lets train our model using the `fit` method on the `DecisionTreeRegressor` object providing our training data

In [16]:
```
regression_tree.fit(X_train, Y_train)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/tree/_classes.py:
363: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in versio
n 1.2. Use `criterion='squared_error'` which is equivalent.
  FutureWarning,
```

Out[16]:
```
DecisionTreeRegressor(criterion='mse')
```

Evaluation

To evaluate our dataset we will use the `score` method of the `DecisionTreeRegressor` object providing our testing data, this number is the R^2 value which indicates the coefficient of determination

In [17]:
```
regression_tree.score(X_test, Y_test)
```

Out[17]:
```
0.7687231436993582
```

We can also find the average error in our testing set which is the average error in median home value prediction

In [18]:
```
prediction = regression_tree.predict(X_test)

print("$",(prediction - Y_test).abs().mean()*1000)
```

```
$ 2915.189873417721
```

Excercise

Train a regression tree using the `criterion` `mae` then report its R^2 value and average error

In [20]:
```
regression_tree = DecisionTreeRegressor(criterion = "mae")

regression_tree.fit(X_train, Y_train)

print(regression_tree.score(X_test, Y_test))

prediction = regression_tree.predict(X_test)

print("$",(prediction - Y_test).abs().mean()*1000)
```

```
0.863874691669748
$ 2630.3797468354423
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/tree/_classes.py:
370: FutureWarning: Criterion 'mae' was deprecated in v1.0 and will be removed in versio
n 1.2. Use `criterion='absolute_error'` which is equivalent.
  FutureWarning,
```

▶ Click here for the solution

Authors

Azim Hirjani

<!--

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-07-20	0.2	Azim	Modified Multiple Areas
2020-07-17	0.1	Azim	Created Lab Template

--!>