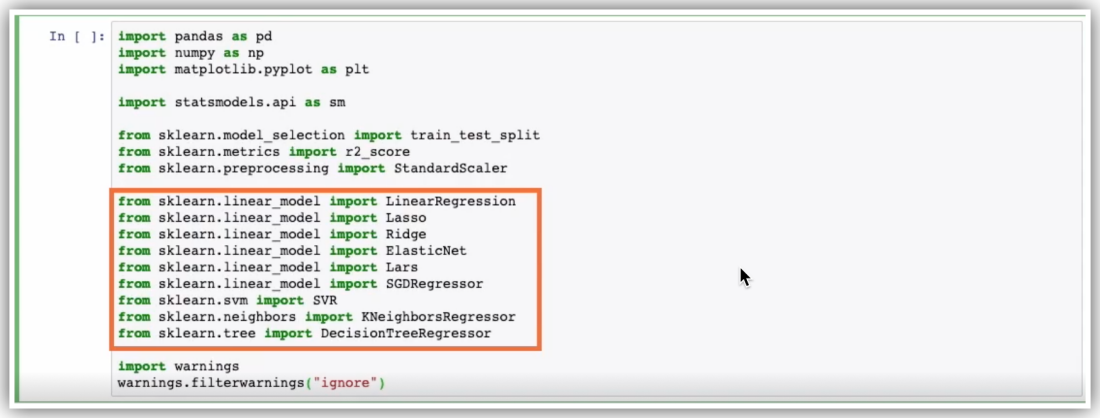
Lasso, Ridge and Elastic Net Regression

October 21, 2021

[3]:
```
import  pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.linear_model import Lars
from sklearn.linear_model import  SGDRegressor

from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

from IPython.display import Image


'''
I've turned off warnings here in this Jupyter Notebook,
'''
import warnings
warnings.filterwarnings("ignore")
```

[4]:
```
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/LinerRegression/Images/2021-10-21_18-38-17.png')
```

[4]:



In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import statsmodels.api as sm

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Lars
from sklearn.linear_model import SGDRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

import warnings
warnings.filterwarnings("ignore")
```

[8]:

```
'''
We'll build all of these models in the same notebook using a few helper
functions that we'll set up first.
'''
'''
Let's go ahead and use pandas to read in our dataset that has been cleaned and␣
 ↪preprocessed earlier.
This is in the auto-mpg- processed.csv file. Here is what the dataset looks␣
 ↪like,

We'll use all of the other features, cylinders, displacement, horsepower, and␣
 ↪so on,
to predict the mileage for the cars.
'''

automobile_df =pd.read_csv('data/auto-mpg-processed.csv')
automobile_df.sample(5)
```

[8]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	age
384	14.0	8	318.0	150	4077	14.0	49
278	22.0	4	140.0	72	2408	19.0	50
71	29.0	4	135.0	84	2525	16.0	39
300	15.0	8	318.0	150	4135	13.5	49
20	17.0	8	302.0	140	3449	10.5	51

[17]:

```
'''
I'm going to instantiate a dictionary here called result_dict that will hold the
training and test scores from the different models that we build and train.

The keys will be meaningful names for the different models that we build and the
values will be their training and test R squares.
```

```
In this way, by simply doing the results stored in this dictionary,
we'll be able to compare different models.
'''
result_dict ={}
```

[18]:
```
'''
I'm going to define a helper function here called build_model that will allow␣
 ↪me to
build and train the different regression models.
'''
'''
:param regression_fn:
    :param name_of_y_col:
    :param names_of_x_cols:
    :param dataset:
    :param test_frac:
    :param preprocess_fn:
    :param show_plot_Y:
    :param show_plot_scatter:

 The first argument here is the regression function. This is a function that␣
 ↪takes in a training
 data and corresponding target values. This will instantiate a particular ML␣
 ↪regression model,
 whether it's a linear regression model, a lasso model, a ridge or an elastic␣
 ↪net model, anything.
 And this function will train the model on our training data.

 The name of y_col input argument specifies the column name in our data frame␣
 ↪for the target
 values that we should use for training.

 The names_of_x_cols is a list of feature columns. These are the columns that␣
 ↪we want to
 include as features when we train our model.

 The dataset is the original data frame that contains the features, as well as␣
 ↪our target values.

 The test_frac specifies how much of our dataset we should hold out to evaluate␣
 ↪or measure our model,
 that is the fraction of our data that will be used as test data.

 If you want the data to be preprocessed in some way, standardized or scaled␣
 ↪before you feed
 it into your regression model, you can specify a preprocessed function.
```

```
    By default, it's set to None.

    Set show_plot_Y to True if you want to display a plot of actual versus predicted Y values,
    and set show_plot_scatter to true if you want to see how your regression line fits on the training data.


    '''
def build_model(regression_fn,
                    name_of_y_col,
                    names_of_x_cols,
                    dataset,
                    test_frac=0.2,
                    preprocess_fn=None,
                    show_plot_Y=False,
                    show_plot_scatter=False):

    '''
    Extract from the dataset the features that you want to train your model into the variable X
    and extract the target value into Y.
    '''
    X=dataset[names_of_x_cols]
    Y=dataset[name_of_y_col]

    '''
       If you've specified a function used to preprocess your model, apply this preprocessing
       function to your X values.
       The preprocessed features are stored once again in the X variable.
    '''
    if preprocess_fn is not None:
        X=preprocess_fn(X)

    '''
    Use scikit-learn's train_test_split function to split up your dataset into training and test data.
    '''
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=test_frac)
    '''
    Once you have your training data, pass in the training data, as well as the
    corresponding labels to the regression function.
```

```
    The regression function is a wrapper that will instantiate a particular␣
↪regression model and
    train on the dataset you've specified.

    The regression function will return the fully trained ML model, which you␣
↪can then use
    for prediction, and store your predicted values in y_pred.
    '''
    model= regression_fn(x_train,y_train)
    y_pred=model.predict(x_test)

    '''
    You can then print out the R square values on the training data, as well as␣
↪the test data
    for your model.
    '''
    print("Training_score : " , model.score(x_train, y_train))
    print("Test_score : ", r2_score(y_test, y_pred))

    '''
    If you've invoked the build model function with show_plot_Y is equal to␣
↪True, plot the
    actual values versus predicted values in the form of a line chart
    '''
    if show_plot_Y == True:
        fig, ax = plt.subplots(figsize=(12, 8))

        plt.plot(y_pred, label='Predicted')
        plt.plot(y_test.values, label='Actual')

        plt.ylabel(name_of_y_col)

        plt.legend()
        plt.show()

    '''
    if you've called it with show_plot_scatter equal to True, display a scatter␣
↪plot in
    matplotlib with the original X and Y values of the test data and the␣
↪predicted line.
    '''
    if show_plot_scatter == True:
        fig, ax = plt.subplots(figsize=(12, 8))

        plt.scatter(x_test, y_test)
        plt.plot(x_test, y_pred, 'r')
```

```
        plt.legend(['Predicted line','Observed data'])
        plt.show()

    '''
    we'll return from this build model function the training score and test R␣
 ↪square
    score for this particular model.
    '''
    return {
        'training_score': model.score(x_train,y_train),
        'test_score': r2_score(y_test,y_pred)
    }
```

[19]:
```
'''
This is the compare_results function. This is the function that will quickly␣
 ↪print out the
training, as well as test scores for all of the regression models that we've␣
 ↪built so far.

This function uses a for loop to iterate through all of the keys in our result␣
 ↪dictionary
and then prints out the kind of regression that was performed, the training␣
 ↪score,
as well as the test score.
'''
def compare_results():
    for key in result_dict:
        print('Regression: ', key)
        print('Training score', result_dict[key]['training_score'])
        print('Test score', result_dict[key]['test_score'])
        print()
```

[ ]:
```

```