**Assignment 4**

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.

**What is linear Regression**

Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e it finds the linear relationship between the dependent and independent variable.

**Linear Regression on Boston Housing Dataset**

The Housing dataset which contains information about different houses in Boston. This data was originally a part of UCI Machine Learning Repository and has been removed now. We can also access this data from the scikit-learn library. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.

The problem that we are going to solve here is that given a set of features that describe a house in Boston, our machine learning model must predict the house price. To train our machine learning model with boston housing data, we will be using scikit-learn’s boston dataset.

In this dataset, each row describes a boston town or suburb. There are 506 rows and 14 attributes (features) with a target column (MEDV).

First, we will import the required libraries.

*# Importing the libraries*

import pandas as pd

import numpy as np

from sklearn import metrics

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

Next, we will load the housing data from the scikit-learn library and understand it.

# Importing the Boston Housing dataset

from sklearn.datasets import load\_boston

boston = load\_boston()

We print the value of the boston\_dataset to understand what it contains. print(boston\_dataset.keys()) gives

dict\_keys(['data', 'target', 'feature\_names', 'DESCR'])

* *data*: contains the information for various houses
* *target*: prices of the house
* *feature\_names*: names of the features
* *DESCR*: describes the dataset

### Data description

The Boston data frame has 506 rows and 14 columns.

This data frame contains the following columns:

**CRIM**: Per capita crime rate by town  
**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 oxide 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  
**PTRATIO**: Pupil-teacher ratio by town  
**B**: 1000(Bk — 0.63)², where Bk is the proportion of [people of African American descent] by town  
**LSTAT**: Percentage of lower status of the population  
**MEDV**: Median value of owner-occupied homes in $1000s

The prices of the house indicated by the variable **MEDV**  is our **target variable** and the remaining are the **feature variables** based on which we will predict the value of a house.

We will now load the data into a pandas dataframe using pd.DataFrame. We then print the first 5 rows of the data using head()

|  |  |
| --- | --- |
|  | |
|  |  |
| *# Initializing the dataframe*  data = pd.DataFrame(boston.data)  *# See head of the dataset*  data.head()  *#Adding the feature names to the dataframe*  data.columns = boston.feature\_names  data.head()  C:\Users\Sir\Desktop\TE DIV3\1_5KmVaL6NijJI3rWZrbGvnA.png  *#Adding target variable to dataframe*  data['MEDV'] = boston.target  *# Median value of owner-occupied homes in $1000s*  *#Check the shape of dataframe*  data.shape  *#Check the data columns*  data.columns  *#Check the data types*  data.dtypes ****Data preprocessing**** After loading the data, it’s a good practice to see if there are any missing values  in the data. We count the number of missing values for each feature using isnull()  # Check for missing values  data.isnull().sum()  # See rows with missing values  data[data.isnull().any(axis=1)]  # Viewing the data statistics  data.describe()  we create a correlation matrix that measures the linear relationships between  the variables. The correlation matrix can be formed by using the  corr function from the pandas dataframe library. We will use the heatmap  function from the seaborn library to plot the correlation matrix.  *# Finding out the correlation between the features*  corr = data.corr()  corr.shape  *# Plotting the heatmap of correlation between features*  plt.figure(figsize=(20,20))  sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot\_kws={'size':15},  cmap='Greens')    *# Spliting target variable and independent variables*  X = data.drop(['MEDV'], axis = 1)  y = data['MEDV']  **Splitting the data into training and testing sets**  we split the data into training and testing sets. We train the model with 80% of the samples  and test with the remaining 20%. We do this to assess the model’s performance on unseen  data. To split the data we use train\_test\_split function provided by  scikit-learn library. We finally print the sizes of our training and test set to verify  if the splitting has occurred properly.  *# Splitting to training and testing data*  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = 0.3, random\_state = 4)  **Linear regression**  We use scikit-learn’s LinearRegression to train our model on both the  training and test sets. Training the model *# Import library for Linear Regression*  from sklearn.linear\_model import LinearRegression  *# Create a Linear regressor*  lm = LinearRegression()  *# Train the model using the training sets*  lm.fit(X\_train, y\_train)  *# Value of y intercept*  lm.intercept\_  *#Converting the coefficient values to a dataframe*  coeffcients = pd.DataFrame([X\_train.columns,lm.coef\_]).T  coeffcients = coeffcients.rename(columns={0: 'Attribute', 1: 'Coefficients'})  coeffcients Model Evaluation *# Model prediction on train data*  y\_pred = lm.predict(X\_train)  *Model Evaluation*  print('R^2:',metrics.r2\_score(y\_train, y\_pred))  print('Adjusted R^2:',1 - (1-metrics.r2\_score(y\_train, y\_pred))\*(len(y\_train)-1)/  (len(y\_train)-X\_train.shape[1]-1))  print('MAE:',metrics.mean\_absolute\_error(y\_train, y\_pred))  print('MSE:',metrics.mean\_squared\_error(y\_train, y\_pred))  print('RMSE:',np.sqrt(metrics.mean\_squared\_error(y\_train, y\_pred)))  𝑅^2 : It is a measure of the linear relationship between X and Y. It is interpreted as  the proportion of the variance in the dependent variable that is predictable from the  independent variable.  Adjusted 𝑅^2 :The adjusted R-squared compares the explanatory power of  regression models that contain different numbers of predictors.  MAE : It is the mean of the absolute value of the errors. It measures the difference  between two continuous variables, here actual and predicted values of y.  MSE: The mean square error (MSE) is just like the MAE, but squares the difference  before summing them all instead of using the absolute value.  RMSE: The mean square error (MSE) is just like the MAE, but squares the difference  before summing them all instead of using the absolute value.  *# Visualizing the differences between actual prices and predicted values*  plt.scatter(y\_train, y\_pred)  plt.xlabel("Prices")  plt.ylabel("Predicted prices")  plt.title("Prices vs Predicted prices")  plt.show()  C:\Users\Sir\Desktop\TE DIV3\__results___30_0.png | |
| *# Checking residuals*  plt.scatter(y\_pred,y\_train-y\_pred)  plt.title("Predicted vs residuals")  plt.xlabel("Predicted")  plt.ylabel("Residuals")  plt.show()  C:\Users\Sir\Desktop\TE DIV3\__results___31_0.png | |
|  | |  |

There is no pattern visible in this plot and values are distributed equally around zero. So Linearity assumption is satisfied

*# Checking Normality of errors*

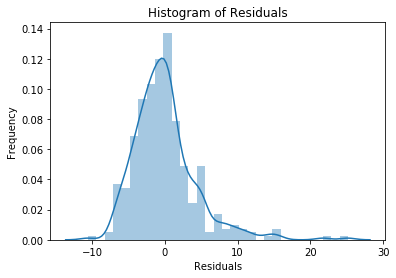
sns.distplot(y\_train-y\_pred)

plt.title("Histogram of Residuals")

plt.xlabel("Residuals")

plt.ylabel("Frequency")

plt.show()



Here the residuals are normally distributed. So normality assumption is satisfied

#### For test data

*# Predicting Test data with the model*

y\_test\_pred = lm.predict(X\_test)

**Model Evaluation**

acc\_linreg = metrics.r2\_score(y\_test, y\_test\_pred)

print('R^2:', acc\_linreg)

print('Adjusted R^2:',1 - (1-metrics.r2\_score(y\_test, y\_test\_pred))\*(len(y\_test)-1)/(len(y\_test)-X\_test.shape[1]-1))

print('MAE:',metrics.mean\_absolute\_error(y\_test, y\_test\_pred))

print('MSE:',metrics.mean\_squared\_error(y\_test, y\_test\_pred))

print('RMSE:',np.sqrt(metrics.mean\_squared\_error(y\_test, y\_test\_pred)))

Here the model evaluations scores are almost matching with that of train

data. So the model is not overfitting.