Boston Housing Price Prediction using Linear Regression

Section 1 : Introduction

1.1 Background

The Boston Housing dataset is a widely used dataset in order to train models to predict the price of the houses based on the attributes/factors provided in the dataset. This is a suitable dataset for regression based tasks. Here, we implement the **Linear Regression Model** on the dataset which was covered as a part of the AI/ML course curriculum.
In order to implement linear regression on the given dataset, multiple Python Libraries are used.

1.2 Goal

The goal of this project is to predict the house prices in Boston correctly with the help of the data given in the Boston Housing Dataset. We need to predict the prices with high accuracy by using **Linear Regression Model**. Application of useful Python Machine Learning and Data Exploration libraries is an added goal of this project.

1.3 Scope

This project shows the application of a Linear Regression Model to real world problems such as predicting the price of a house based on the previously available data.

Section 2 : Dataset Description

2.1 Data Source

The dataset was provided through Kaggle and consisted of a zip file which had 3 csv(comma separated value) files inside.

1. **train.csv :** This file contains the dataset that we will use for training. It consisted of 333 rows and 15 columns.
2. **test.csv :** This file had the data that is to be used for the purpose of testing our model which was trained on the training data. Therefore, this file does not contain the target variable column. This file had 173 rows and 14 columns.
3. **submission.csv:** This file contained the expected format for the submission of the predicted values to kaggle.

2.1 Attributes(Metadata)

The **medv** variable is the target variable.The Boston data frame has 506 rows and 15 columns. (1 for target variable)

This data frame contains the following columns:

1. ***crim***

   per capita crime rate by town.

2. ***zn***

   proportion of residential land zoned for lots over 25,000 sq.ft.

3. ***indus***

   proportion of non-retail business acres per town.

4. ***chas***

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

5. ***nox***

   nitrogen oxides concentration (parts per 10 million).

6. ***rm***

   average number of rooms per dwelling.

7. ***age***

   proportion of owner-occupied units built prior to 1940.

8. ***dis***

   weighted mean of distances to five Boston employment centres.

9. ***rad***

   index of accessibility to radial highways.

10. ***tax***

    full-value property-tax rate per $10,000.

11. ***ptratio***

    pupil-teacher ratio by town.

12. ***black***

    1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.

13. ***lstat***

    lower status of the population (percent).

14. ***medv***

    median value of owner-occupied homes in $1000s.

Section 3 : Data Preprocessing

3.1 Feature Selection

For the purpose of feature selection, it was observed that though the 'ID' column wasn't mentioned as an attribute, it was still present in the training and testing dataset. It is obvious that this attribute would contribute towards the training of the Linear Regression model which may lead to unexpected results or lower accuracy. Therefore, the "ID" column is dropped(removed) from both the files i.e. **train.csv** and t**est.csv.**

3.2 Handling Missing Values

The given Boston Housing Dataset did not consist of any null or redundant values.

3.3 Splitting Data

The scikit-learn API's train_test_split() module is used to split the data into training and testing sets. We make use of 33% of the total data present in the **train.csv** file to train our **Linear Regression Model**. The **test.csv** file does not consist of the target variable, therefore it cannot be used for the purpose of training or for checking the accuracy of our model. The below mentioned order is used to split the data into training and testing set:

1. Load the data from train.csv into a dataframe using Pandas library's read_csv method.
2. Drop the target variable i.e. the ***medv*** column from the dataset and store the new dataframe in a variable say **'X'**.
3. Store the target variable's column in a separate variable say **'y'**.
4. Split both the newly created variables into training and testing sets using the scikit-learn API's train_test_split(). And set the **test_size** argument as 0.33.

This will create 4 variables which will be used for training our model and will contain the splitted dataset values as specified. They will be as follows:

1. **X_train** : Used for training the model.
2. **X_test :** Used for testing the model.
3. **y_train :** Used as the target variable for **X_train**
4. **y_test :** Used as the target variable for **X_test**

Section 4 : Model Implementation

4.1 Model Selection

As the goal of this project is to implement Linear Regression. We will only make use of **Linear Regression** and discuss the further scope for improvement of the project in the discussion section.

4.1.1 Linear Regression

Linear regression is a supervised learning algorithm employed to predict a continuous output variable based on one or more input features, assuming a linear relationship between the input and output variables. The primary objective of linear regression is to determine the line of best fit that minimises the sum of squared errors between the predicted and actual output values.

Given a dataset $x_1, x_2, \ldots, x_n$ where each x_i belongs to $\mathbb{R}^d$, and the corresponding labels $y_1, y_2, \ldots, y_n$ belong to $\mathbb{R}$, the goal of linear regression is to find a mapping between the input and output variables, represented as follows:

$h : \mathbb{R}^d \to \mathbb{R}$

The error for this mapping function can be quantified as:

$\text{error}(h) = \sum_{i=1}^{n} (h(x_i) - y_i)^2$

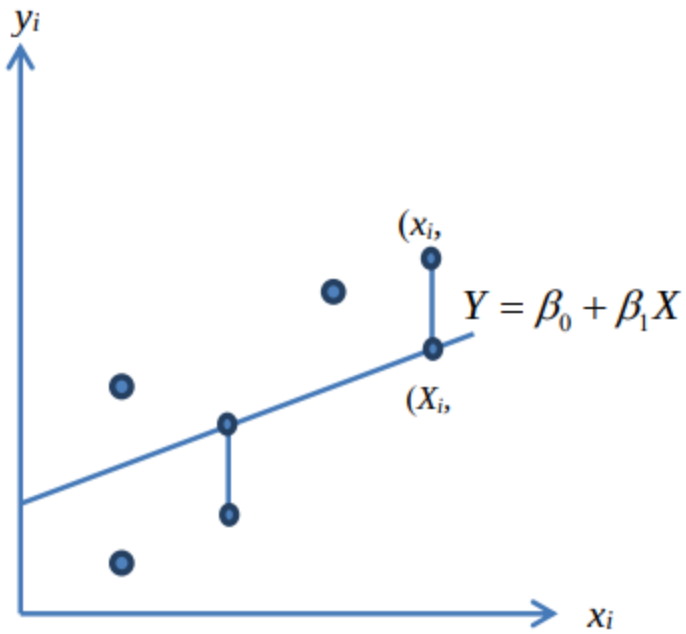
Ideally, this error should be minimised, which occurs when $h(x_i) = y_i$ for all i. However, achieving this may only result in memorising the data and its outputs, which is not a desired outcome.

To mitigate the memorization problem, introducing a structure to the mapping becomes necessary. The simplest and most commonly used structure is linear, which we will adopt as the underlying structure for our data.

Let $\mathcal{H}_{\text{linear}}$ denote the solution space for the mapping in the linear domain:
$\mathcal{H}_{\text{linear}} = \{h_w : \mathbb{R}^d \to \mathbb{R} \text{ s.t. } h_w(x) = w^T x \; \forall w \in \mathbb{R}^d\}$

Thus, our objective is to minimise:

$\min_{h \in \mathcal{H}_{\text{linear}}} \sum_{i=1}^{n} (h(x_i) - y_i)^2$



Direct regression

The above graph represents a simple linear regression. However, in our dataset we will make use of multiple linear regression which will have multiple independent variables trying to predict a single target variable.

4.2 Code Implementation

In order to implement the Linear Regression model we will make use of the Scikit-learn API's Linear Model. Scikit-learn is an ML library for python which comes with a number of useful features such as basic Machine Learning Algorithms, data-preprocessing tools, etc.

The overall flow of code:

1. **Import Necessary Libraries:**
    a. Import `**numpy**`, `**pandas**`, `**matplotlib.pyplot**`, `**train_test_split**` from `sklearn.model_selection`, `**LinearRegression**` from `sklearn.linear_model`, and `**mean_squared_error**`, `**mean_absolute_error**`, and `**r2_score**` from `sklearn.metrics`.
2. **Load the Dataset:**
    a. Load the dataset from a CSV file into a pandas DataFrame.

3. **Inspect and Clean the Data:**
    a. Display the first few rows of the dataset.

b. Check for and handle missing values.

c. Drop any redundant or non-numeric columns, such as IDs.

4. **Prepare Features and Target Variable:**

    a. Define the feature matrix **X** and the target variable **y.**

5. **Split the Dataset into Training and Testing Sets:**

    a. Split the data into training and testing sets using `**train_test_split**`.

6. **Initialize and Train the Model:**

    a. Initialise the `**LinearRegression**` model.

    b. Fit the model to the training data.

7. **Evaluate the Model:**

    a. Predict on the test set.

    b. Calculate and display evaluation metrics, including:

        i. **Mean Squared Error (MSE)**

        ii. **Root Mean Squared Error (RMSE)**

        iii. **Mean Absolute Error (MAE)**

        iv. **R^2 Score**

4.3 Plots

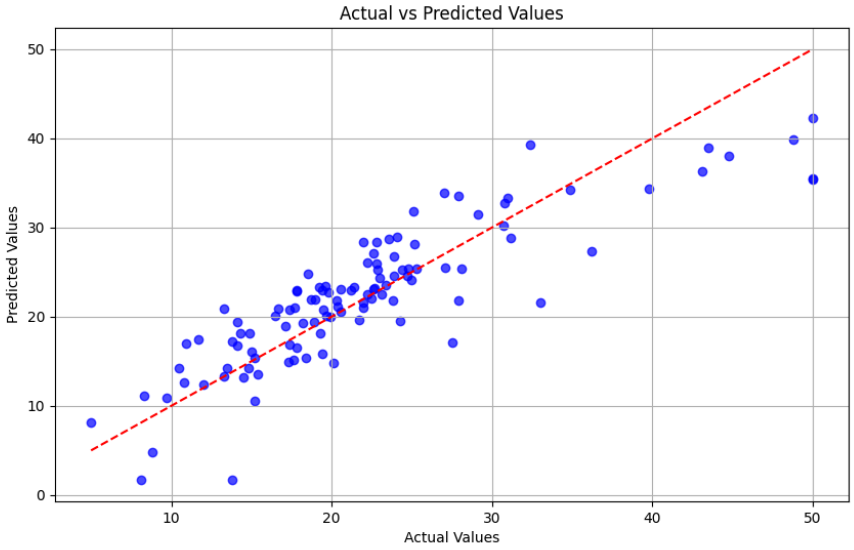


Fig 1: Above is the visual representation of the **actual v/s the predicted** values. The actual values are the blue dots whereas the read dotted line represents the linear regression line that we fit with our model

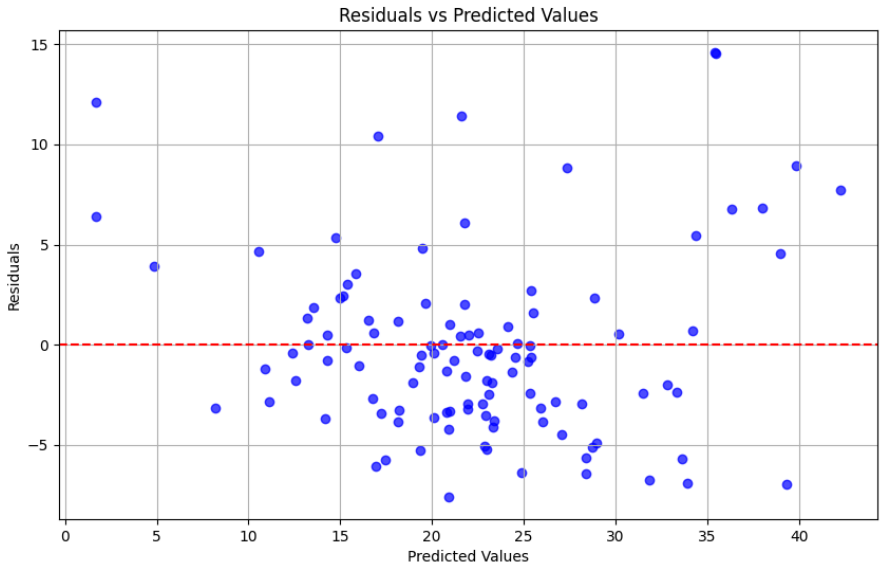


Fig 2: Above figure shows the plot of Residuals (seen in Fig 1) and the predicted values. This plot helps to identify whether the model is appropriately capturing the relationship between input features and target variables.

Section 5 : Results and Analysis

Metric	Result
R_2 Score	0.7451
MSE(Mean Squared Error)	20.6271
RMSE(Root Mean Squared Error)	4.5417
MAE(Mean Absolute Error)	3.4100

From the above table of results based on different metrics, we can conclude that our Linear Regression Model has a decent R_2 Score with all the other metrics being in a similar range as well. This is one of the best possible models with the Linear Regression Algorithm.

Challenges and Limitations

Our model isn't the best due to the restrictions of it being **linear** and also the lack of data reduces the model's effectiveness. Our dataset is restricted to only 333 samples out of which 33% are stored solely for the purpose of testing.

Scope for Improvement and Possible Solution

Other techniques such as Ridge Regression along with Cross Validation have been implemented but did not result in any significant change in the result metric values.

The possible reason for this is the lack of enough samples to train our model on. Linear regression performs very well when the number of samples are large and therefore if we wish to improve the results of our model our first priority should be to increase the sample size of our dataset.

Another possible solution can be to use other machine learning techniques such as Decision Trees or SVM and even Neural Networks could provide a better result on this same dataset.

Conclusion

From this we can conclude that though linear regression may be one of the easiest to implement algorithms with decent accuracy, it still has a lot of constraints due to which its accuracy can be compromised. It is the best to be used with a large sized dataset along with a regularisation technique such as Ridge or Lasso Regularization.