csoc-ig

Sebastian K Sabu

May 2025

1 Introduction

We have use three different approach to build our multi variable linear regression model.

- Using pure Python
- Using Numpy
- Using Scikit-Learn

I also used some preprocessing of the data and used graphs to verify and find better solution for our model

2 Preprocessing

We first checked for any missing values and found that there were 207 missing values in the "total bedrooms" column. This can be completely removed since the missing values only made up 1% of the data or when looked closely there was a strong correlation between number of households and total number of bedrooms, therefore I put in the corresponding missing values with their total household. Later you will see that this will not matter since we are removing this column When plotting a scatter graph of all values with respect to "median house value" only one of the following had a correlation of more than 0.5 that was the feature "median household income" as seen in fig:1

Therefore, I used only that column for my model but the core of my algorithm i build can take in multiple columns. The next thing i did was try and scale the data to similar size. At first i used a scaler but it felt too overwhelming,therefore I just divided the house value column by 10000.

3 Model

The model uses Linear Regression Model Where we try to find the Weights and Bias so that we get a Line

$$y = w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n + C$$

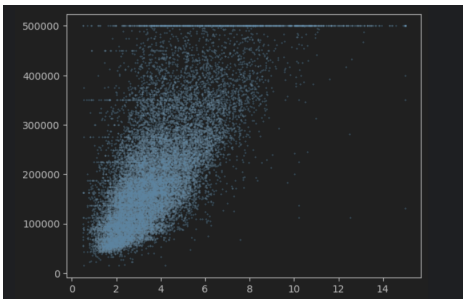

Figure 1: Enter Caption

where

$$w_i$$

are the weights,

$$C$$

is the bias and

$$x_i$$

are values of different feature We then use a cost function $C(x)$ In this case its is the mean square error. Therefore,

$$C(x) = \frac{1}{n}\sum(y_i - y)^2$$

Now we have to minimize this function when the different biases and weights are changed, this can be achieved by setting the partial derivative

$$\frac{\partial C(x)}{\partial w_i} = 0$$

equal to 0 with respect to weights and bias.But we cannot solve the equation, therefore we use something called gradient descent.

In gradient analysis we set some initial value for the weights and bias and then find the derivative of it at that point, then use this formula to find the new weight

$$w_i = w_i + \frac{\partial C(x)}{\partial w_i} * learningrate$$

where the learning factor is the size of the step we should take to reach the minimum. In our case I found $learning\ rate = 0.01$ to be fine. The same formula is used for calculating bias. We repeat this step for a given number of iterations.

4 Evaluation Criteria

4.1 Time

For the part 1 and 2 we use convergence time which is approximately:

1. $45s$
2. $0.077s$

the fitting time in part 3 is $0.004s$

the significant difference between pure python and Numpy is the fact that Numpy is implemented in C and the memory is contiguous. Along with the utilization of vectorized CPU operation.

The difference between Numpy and Scikit is the fact that Scikit use better optimized weights.

4.2 Performance Metrics

The mean absolute error is as follows :

1. 2.33
2. 2.30
3. 6.26

The root mean squared error is as follows :

1. 5.16
2. 5.03
3. 8.37

The R-squared is as follows :

1. 0.446
2. 0.473
3. 0.473

4.3 Cost function Vs Iteration

From the graph it is clear that after around 10 rounds the change in error becomes negligible. And the output of both Part 1 and Part 2 is same since it uses same initial weights and same logic

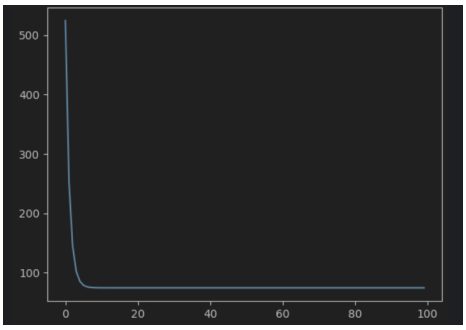

Figure 2: Enter Caption

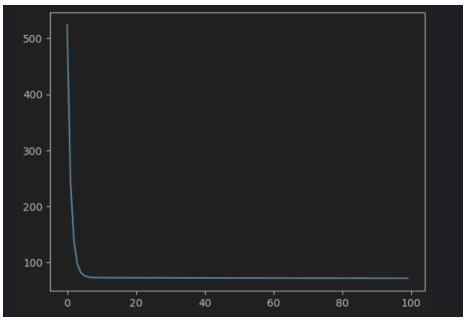

Figure 3: Enter Caption