csoc-ig report

Shambhavi

May 2025

Linear Regression

1 Introduction

The code involves executing linear regression at three levels:

1. Pure Python implementation
2. Optimized NumPy implementation
3. Utilizing the Scikit-learn library.

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. The Ocean Proximity column had some unique values, so i used LabelEencoder to encode the column. Then I built a heatmap to see correlation between "median house value" and other columns. The most correlated column to "median house value" is "median income". So I took "median income" as independent variable or feature colum and "median house value" as dependent variable or target column.

3 Model

The model usese LinearRegression Model Where we try to find the Weights and Bias so that we get a Line

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

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 desecnt we set the some inital value for the weights and bias and then find the derivative of it at that point we then use this formula to find the new weight

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

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

The learning rate for both part 1 and part 2 is 0.001 and number of iterations are 2000. I have taken the initial weights to be based on cross deviation and bias is initialized to 0.The graph for cost function is in code-csoc.ipynb file

4 Convergence Time

The converging time for Pure Python Implementation is 6.841 seconds and for Optimized NumPy Implementation the converging time is 0.1057 second. It is due to NumPy is faster. This significant difference is due to NumPy's superior speed. NumPy's data structures are more efficient in terms of space usage. Thanks to features like array homogeneity and various internal optimizations, memory consumption is minimized, resulting in much faster performance.

The clear contrast in the execution times demonstrates the benefits of employing NumPy for computations involving numbers. Its design features, such as vectorization, memory use, and homogeneity in data types, all contribute to the better performance compared to standard Python implementations.

The fitting time for Scikit-learn library is 0.0089 seconds.

5 Accuracy

The R2 score for Pure Python Implementation is **0.4689** , for Optimized NumPy implementation is **0.8942** and for Scikit-learn is **0.4409**

The key reason for NumPy performing better than Scikit-Learn could be :

NumPy utilizes vectorized operations that allow for element-wise computations directly on arrays without the need for explicit loops.

NumPy arrays are more memory efficient than Python lists or Scikit-learn data structures.

NumPy natively supports multidimensional arrays, enabling efficient manipulation and computation of complex data structures.

Scikit-learn is built on top of NumPy and includes additional layers for machine learning functionalities, which can introduce overhead. NumPy, being a lower-level library, is optimized for raw numerical computations.

The mean Absolute Error for pure Python implementation is 62017.9, for Optimized NumPy it is 62017.9 and for scikit-learn it is 66667.59.

The root mean square error for pure Python implementation is 83962.95, for Optimized NumPy it is 83962.9 and for scikit-learn it is 86526.57.

6 Summary

My NumPy model works better than the Scikit-learn library and pure python implementation in this case.