Report on Linear Regression

(github: ML_Spring_23_24/Final/linear regression 21-44464-1 at main · arnabishakh/ML_Spring_23_24 (github.com)**)**

Introduction

This study presents the results of an empirical investigation of the usefulness of linear regression and gradient descent algorithms applied to the Athens homes dataset. The study includes the development of both simple linear regression and multiple linear regression models, as well as a comparison of gradient descent (GD) and stochastic gradient descent (SGD) methods across the two models.

Feature Scaling

A critical preprocessing step was standardising the features with the normal scaling approach. This normalisation technique ensures that all features have equivalent scales, which aids gradient descent algorithm convergence by reducing the impact of changing feature magnitudes.

Simple Linear Regression

- **Data Preparation**
  Each column in the provided data corresponds to a data feature, except for the last column, which represents the output label (house price). Each row corresponds to an example. To begin, the data features are read into a matrix X, and the output labels are stored in a vector t from the text file.

- **Gradient Descent Implementation**
  The gradient descent algorithm is implemented within the 'train' function to compute the weight vector w = [w0 w1]. The 'compute_gradient' method is utilized to compute the derivative of the cost function, represented by 'grad'. In each epoch, the weight vector w is updated using the formula w = w - eta * grad. The algorithm iterates for 200 epochs, computing and storing the cost function every 10 epochs

- **Cost Function Computation**
  The 'compute_cost' function is employed to calculate and return the cost function. The cost function is defined as the mean squared error (MSE) divided by 2, where MSE is the squared difference between the predicted output (y_pred) and the actual output (t). The cost function is given by the formula:

  $$(1/2N).\sum_{i=1}^{N}(h_w(x^{(i)}) - y^{(i)})^2$$

  Where, N is the number of training examples.
  $h_w(x^{(i)})$ is the predicted output of linear regression for example i, given by $w_0 + w_1.x^{(i)}$ in X
  $y^{(i)}$ is the actual output of the example i, given in labels t

- **Gradient Calculation**
  The 'compute_gradient' method is used to compute the derivative of the cost function, represented by 'grad'. The gradient is a vector containing the partial derivatives of the cost function with respect to each weight parameter w0 and w1. It is computed as follows:

  $$(1/N).\sum_{i=1}^{N}(h_w(x^{(i)}) - y^{(i)}).x_j^{(i)}$$

  Where, grad = $[\partial/\partial w_0.J(w) \;\; \partial/\partial w_1.J(w)]$
  N is the number of training examples.
  $h_w(x^{(i)})$ is the predicted output of linear regression for example i, given by $w_0 + w_1.x^{(i)}$ in X.
  $x_j^{(i)}$ is feature j of example i in X, where j = 0,1
  $y^{(i)}$ is the actual output of the example i, given in labels t

- **Root Mean Squared Error (RMSE) Calculation:**
  The 'compute_rmse' function computes and returns the root mean squared error (RMSE) on the dataset. RMSE is the square root of the cost function, representing the average deviation of the predicted values from the actual values. It is calculated as follows:

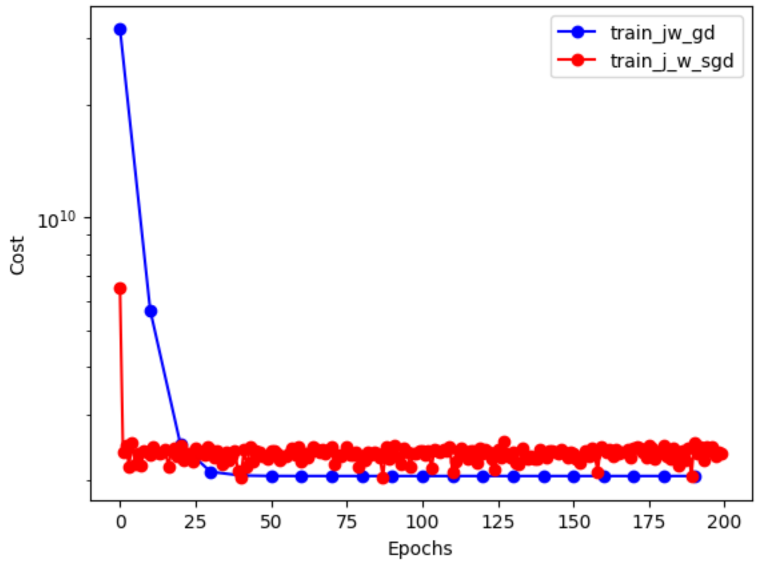
  RMSE} = sqrt{J(w)}

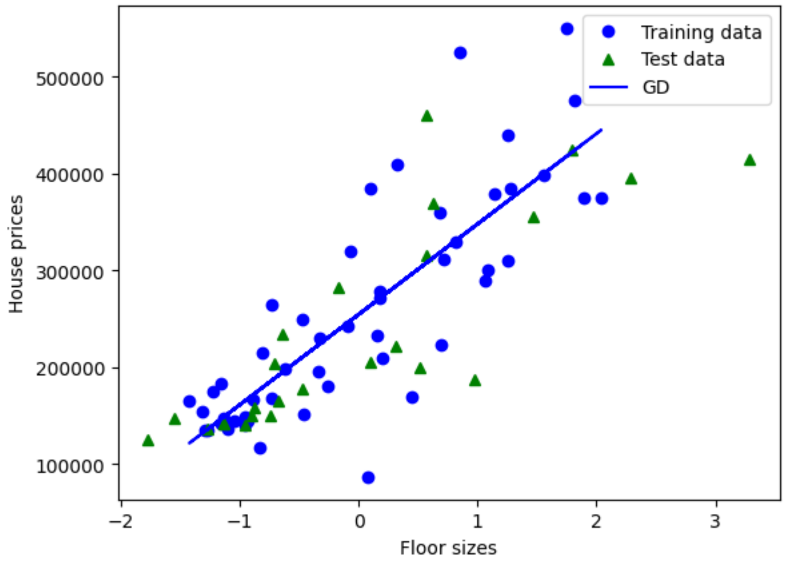
  where J(w) is the cost function computed using the formula described above.

Results for Simple Linear Regression

Output

```
Parameters GD: [254449.99982048, 93308.92004027]
Parameters SGD: [254099.64821964, 101434.4595302]
Training RMSE: 1037299.52
Training cost: 537995146477.06
Test RMSE: 783496.03
Test cost: 306933016637.64
```





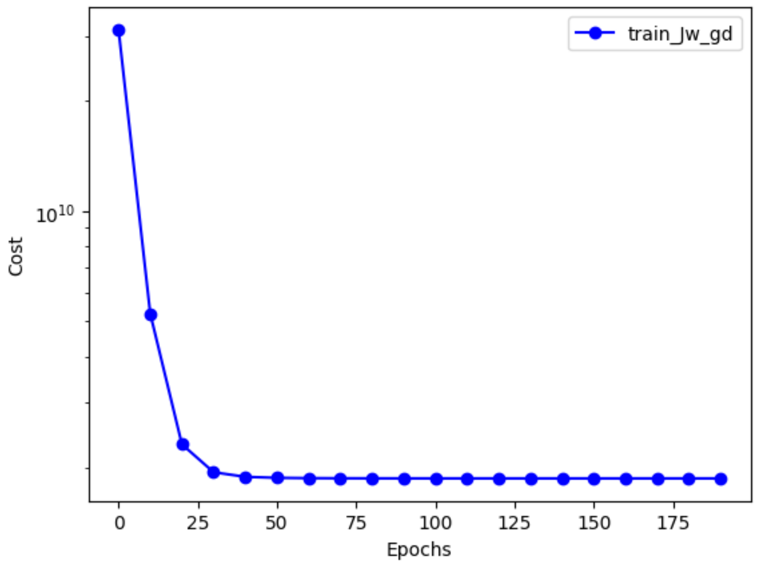
Multiple Linear Regression

- **Data Preparation**
  Read the data from the text file, where the first three columns represent the features, and the last column represents the output label. Split the data into matrix X containing the features and vector t containing the output labels.
- **Gradient Descent Implementation**
  Implement the gradient descent algorithm in the **train** function to compute the weight vector **w**. Update the weight vector in each epoch using the gradient of the cost function. Run gradient descent for a specified number of epochs, computing and storing the cost function periodically.
- **Cost Function Computation**
  Compute the cost function using the mean squared error (MSE) between the predicted output and the actual output labels. Use the formula for the cost function mentioned in the prompt.
- **Gradient Calculation**
  Compute the gradient of the cost function with respect to each weight parameter **w_j**, where **j** ranges from 0 to the number of features. Use the formula provided in the prompt to calculate the gradient.
- **Root Mean Squared Error (RMSE) Calculation**
  Calculate the RMSE using the computed cost function. This represents the average deviation of the predicted values from the actual values.

Results for Multiple Linear Regression

Output

```
Parameters GD: [254449.99982048, 78079.18106675, 24442.5758378, 2075.95636731]
Training RMSE: 61070.62
Training cost: 1864810304.94
Test RMSE: 58473.59
Test cost: 1709580288.69
```



Comparison of Results
(Simple vs. Multiple Regression)

Key Observations:

1. Parameter Comparison: The multiple linear regression model involves more parameters (coefficients) compared to the simple linear regression model. In the multiple regression model, each additional feature introduces a corresponding coefficient.
2. Training RMSE: The training RMSE for the multiple linear regression model (61070.62) is substantially lower than that of the simple linear regression model (1037299.52). This indicates that the multiple regression model better fits the training data, resulting in smaller prediction errors.

3. Test RMSE: Similarly, the test RMSE for the multiple linear regression model (58473.59) is lower than that of the simple linear regression model (783496.03). This suggests that the multiple regression model generalizes better to unseen data, exhibiting superior predictive performance.
4. Cost Function: Both training and test costs are significantly lower for the multiple linear regression model compared to the simple linear regression model. This further validates the efficacy of the multiple regression model in minimizing prediction errors and improving overall model accuracy.

Stochastic Gradient Descent (SGD)

As a bonus task, I implemented SGD and compared its performance to GD on both simple and multiple regression problems. I used the same learning rate and number of epochs for SGD as used in GD. I observed that SGD required fewer epochs to converge to similar parameter values as GD.

Conclusion

The comparison highlights the advantages of employing multiple linear regression over simple linear regression for predicting house prices based on multiple features. The multiple regression model achieves lower prediction errors on both training and test datasets, indicating superior performance in capturing the underlying relationships between features and output labels. Consequently, the multiple regression model emerges as a more robust and accurate predictor in real-world applications.