Linear Regression

Theory

Linear regression is a method used to analyze the relationship between a dependent variable and one or more independent variables. For simple linear regression the goal is to find a "line of best fit" which is a line that minimizes error and can accurately predict the output values in a range.

Formula

The hypothesis function for linear regression is given by:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

for simple linear regression (one variable):

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

where:

- $h_\theta(x)$ is the predicted output.
- $\theta_0, \theta_1, ...$ are parameters of the model.
- $x_1, x_2, ...$ are the input features.

Cost function

The cost function $J(\theta)$ for linear regression is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

where:

- m is the number of training examples.
- $h_\theta(x^{(i)})$ is the predicted output of the i^{th} training example.
- $y^{(i)}$ is the actual output of the i^{th} training example

The goal of linear regression is to find values for θ that minimize our cost function.

Gradient Descent

To minimize the cost function, we use the gradient descent algorithm:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

For all j, where:

- α is the learning rate.
- θ_j is the j^{th} model parameter.
- $x_j^{(i)}$ is the j^{th} feature of the i^{th} training example.

C++ Implementation

In the C++ implementation of this code, I utilized the Eigen library, which streamlines and optimizes linear algebra operations. Given this, I should highlight the modifications I made to the formulas mentioned earlier, which enabled me to leverage linear algebra more effectively.

Linear Algebra in Linear Regression

Benefits of Using Linear Algebra

- Efficency: Linear algebra allows for efficent computations, especially when dealing with large datasets. Operations that would otherwise require loops can be requced to a single line of code using linear algebra.
- Vectorization: Linear algebra capitalizes on the hardware's ability to perform vectorized operations, which significantly speeds up computations. Modern CPUs and GPUs are optimized for these types of operations.
- Clarity: Representing data as matrices and vectors often simplifies the code. Mathematical operations become more intuitive and resemble the mathematical notation more closely, making the code easier to understand and debug.

Matrix for Input Features (X)

In the corresponding C++ code, you may notice that we have opted to use a matrix in the form:

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ 1 & x_1^{(3)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}$$

instead of a vector. This design choice was implemented so that we can use matrix vector multiplication to efficently calculate $h_\theta(x)$ for all values of x.

$$H = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_m \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ 1 & x_1^{(3)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

SquaredNorm of the error

Through vector subtraction, we can efficiently compute the error for each hypothesis outcome. Here, h_i denotes the prediction using the i^{th} value of x

$$E = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_m \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix}$$

With this error vector E, the cost can be succinctly determined as:

$$J(\theta) = \frac{||E||}{2m}$$

This approach leverages Eigen's squaredNorm() method, offering both clarity and computational efficiency.