Multiple linear regression model

A **multiple linear regression model** is a statistical technique used to predict a dependent variable based on multiple independent variables. It extends simple linear regression by considering multiple predictors.

Formulation

The multiple linear regression model with two predictors can be written as:

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon$$

Where:

- $\mathbf{y}$ is the vector of dependent variable observations, with size $n \times 1$, where n is the number of observations (or samples).
- $\mathbf{X}$ is the matrix of independent variables (or regressors), with size $n \times p$, where p is the number of regressors. In this case, with two regressors (including the intercept term), the matrix will have one column for the intercept (1), one for x_1, and one for x_2, so $\mathbf{X}$ will have size $n \times 3$.
- β is the vector of coefficients (parameters to estimate), with size $p \times 1$. In this case, β will be a vector of size 3×1 (including the intercept term).
- ε is the vector of errors (or residuals), with size $n \times 1$, representing the difference between the observed and predicted values of the model.

For the case with two regressors (including the intercept), the matrix $\mathbf{X}$ will look like this:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{bmatrix}_{(n \times 3)}$$

And the vector β will be:

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}_{(3 \times 1)}$$

Thus, the model becomes:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{(n \times 1)} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{bmatrix}_{(n \times 3)} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}_{(3 \times 1)} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}_{(n \times 1)}$$

OLS estimation

The goal is to find the line (or hyperplane in higher dimensions) that best fits the data by minimizing the **sum of squared errors** (cost function).

$$\varepsilon = \mathbf{y} - \mathbf{X}\beta$$

The cost function to minimize is the sum of squared errors:

$$J(\beta) = \varepsilon^\top \varepsilon = (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta)$$

To find the minimum of the cost function, it is necessary to calculate the derivative of $J(\beta)$ with respect to β and set it to zero.

Expand the cost function:

$$J(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) = \mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{X}\beta - \beta^\top \mathbf{X}^\top \mathbf{y} + \beta^\top \mathbf{X}^\top \mathbf{X}\beta$$

Since $\mathbf{y}^\top \mathbf{X}\beta$ is a scalar, it follows that $\mathbf{y}^\top \mathbf{X}\beta = \beta^\top \mathbf{X}^\top \mathbf{y}$. So the cost function becomes:

$$J(\beta) = \mathbf{y}^\top \mathbf{y} - 2\beta^\top \mathbf{X}^\top \mathbf{y} + \beta^\top \mathbf{X}^\top \mathbf{X}\beta$$

Now calculate the derivative of $J(\beta)$ with respect to β:

$$\frac{\partial J(\beta)}{\partial \beta} = -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\beta$$

Set the derivative equal to zero to find the estimated coefficients:

$$-2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\beta = 0$$

Solve for β:

$$\mathbf{X}^\top \mathbf{X}\beta = \mathbf{X}^\top \mathbf{y}$$

$$\hat{\beta}_{\text{OLS}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

This is the **Ordinary Least Squares (OLS)** formula for estimating the coefficients β. This vector contains the estimates of β_0, β_1, and β_2, which are the coefficients of the multiple linear regression model.

Goodness of fit measure

The coefficient of determination R^2 is a measure of how well the model fits the data. It is calculated as:

$$R^2 = 1 - \frac{\text{Sum of Squared Errors (SSE)}}{\text{Total Sum of Squares (SST)}}$$

Where:

- The **Total Sum of Squares** (SST) measures the total variability in the data relative to the mean of y:

$$\text{SST} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

  with $\bar{y}$ being the mean of y.

- The **Sum of Squared Errors** (SSE) measures the variability not explained by the model, i.e., the sum of the squared differences between the observed values and the predicted values:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

  where $\hat{y}_i$ is the predicted value for y_i.

Interpretation of R^2

The value of R^2 ranges from 0 to 1:

- $R^2 = 1$ means the model perfectly explains the variability in the data.
- $R^2 = 0$ means the model explains none of the variability in the data.

A high R^2 value indicates that a large portion of the variability in y is explained by the independent variables in the model, while a low value suggests the model has limited predictive power.

Gauss–Markov assumptions

The Gauss-Markov assumptions are a set of conditions under which the Ordinary Least Squares (OLS) estimator $\hat{\beta}_{\text{OLS}}$ is the Best Linear Unbiased Estimator (BLUE).

Zero conditional mean

The expected value of the residuals conditional on the independent variables should be zero:

$$\mathbb{E}(\varepsilon|\mathbf{X}) = 0$$

This assumption ensures the residuals are random and not linked to the independent variables. In other words, the regressors must be uncorrelated with the residuals i.e.

$$\text{Cov}(x_i, \varepsilon_i) = 0 \quad \forall i$$

Homoscedasticity

The residuals should have constant variance across all observations.

$$\text{Var}(\varepsilon|\mathbf{X}) = \sigma\mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. This implies that the residuals are equally spread out across all levels of the independent variables.

No autocorrelation

The residuals should be uncorrelated with each other. This means that for all $i \neq j$, the covariance between the residuals ε_i and ε_j should be zero:

$$\text{Cov}(\varepsilon_i, \varepsilon_j | \mathbf{X}) = 0 \quad i \neq j$$

Normality of residuals (optional)

The residuals should be normally distributed:

$$\varepsilon \sim N(0, \sigma \mathbf{I})$$

While this assumption is not necessary for the OLS estimator to be BLUE, it is often assumed for the purpose of hypothesis testing.

Feed-Forward Neural Network

A **Feed-Forward Neural Network (FFNN)** consists of layers of neurons where information flows in one direction: from the input layer through hidden layers and finally to the output layer. Each layer processes the input data by applying a weighted sum, followed by an activation function. For this task, we focus on a network with one hidden layer, using **RReLU** as the activation function.

Network structure

1. **Input layer**. The input to the network is a vector $\mathbf{x} \in \mathbb{R}^n$, where n is the number of input features.
2. **Hidden layer**. This layer consists of h neurons.
3. **Output layer**. The output layer contains m neurons, and the network produces an output vector $\mathbf{y} \in \mathbb{R}^m$.

Notation

- $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times n}$ is the weight matrix between the input and hidden layers.
- $\mathbf{b}^{(1)} \in \mathbb{R}^h$ is the bias vector for the hidden layer.
- $\mathbf{W}^{(2)} \in \mathbb{R}^{m \times h}$ is the weight matrix between the hidden and output layers.
- $\mathbf{b}^{(2)} \in \mathbb{R}^m$ is the bias vector for the output layer.
- $\mathbf{x} \in \mathbb{R}^n$ is the input vector.
- $\mathbf{a}^{(1)} \in \mathbb{R}^h$ represents the activations of the hidden layer.

- $\mathbf{a}^{(2)} \in \mathbb{R}^m$ represents the output activations.

Steps

1. Input to Hidden Layer

The input vector $\mathbf{x}$ is multiplied by the weight matrix $\mathbf{W}^{(1)}$ and added to the bias vector $\mathbf{b}^{(1)}$. This gives the pre-activation values for the hidden layer:

$$z_i^{(1)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)}, \quad \text{for each hidden neuron } i = 1, \ldots, h$$

2. Activation with RReLU

The RReLU activation function is applied to the pre-activation values. For each neuron i in the hidden layer, the output is computed as:

$$a_i^{(1)} = \begin{cases} \max(0, z_i^{(1)}) & \text{if } z_i^{(1)} \geq 0 \\ \alpha_i z_i^{(1)} & \text{if } z_i^{(1)} < 0 \end{cases}$$

Here, α_i is a randomly chosen value from a uniform distribution $\alpha_i \sim \text{Uniform}(l, u)$, where l and u are the lower and upper bounds of the distribution.

3. Hidden to Output Layer

The activations from the hidden layer $\mathbf{a}^{(1)}$ are then multiplied by the weight matrix $\mathbf{W}^{(2)}$ and added to the bias vector $\mathbf{b}^{(2)}$. This gives the pre-activation for the output layer:

$$z_j^{(2)} = \sum_{i=1}^{h} W_{ji}^{(2)} a_i^{(1)} + b_j^{(2)}, \quad \text{for each output neuron } j = 1, \ldots, m$$

4. Output Layer (Final Prediction)

The final output of the network is calculated from the pre-activation values of the output layer. If no activation function is applied in the output layer:

$$y_j = z_j^{(2)}, \quad \text{for each output neuron } j = 1, \ldots, m$$

By following these steps, the network processes input data and makes predictions through forward propagation.

Below is the graphical representation of a neural network with 2 input features, 1 hidden layer with 4 neurons, and 1 output.

 Rete Neurale

The derived features of the hidden layer are created from linear combinations of the inputs, and the output is created as a linear combination of these derived features.

Model training and loss tracking

The training process, for a regression model, uses **Mean Squared Error (MSE)** as the loss function and **Stochastic Gradient Descent (SGD)** as the optimizer.

Mean Squared Error (MSE)

The MSE loss function measures the average squared differences between actual y_i and predicted $\hat{y}_i$ values:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Stochastic Gradient Descent (SGD)

The model parameters θ are updated using:

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$$

where:

- η is the learning rate
- $\nabla_\theta J(\theta)$ is the gradient of the loss function with respect to θ.

Learning rate scheduler

The learning rate is adjusted dynamically using `StepLR` scheduler (a powerful tool in PyTorch for adjusting the learning rate during training):

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor \frac{t}{\text{step size}} \rfloor}$$

where:

- η_0 is the initial learning rate
- $\gamma = 0.5$ is the decay factor
- $\lfloor \frac{t}{\text{step size}} \rfloor$ represents the number of completed step intervals.

Training Loop

The model is trained over 200 epochs, performing the following steps in each epoch:

1. **Compute predictions**

$$\hat{Y} = f(X; \theta)$$

2. **Calculate Loss**

$$J(\theta) = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

3. **Compute gradients**

$$\nabla_\theta J(\theta) = \frac{2}{n} \sum (y_i - \hat{y}_i) \cdot \frac{\partial \hat{y}_i}{\partial \theta}$$

4. **Update parameters**

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$$

5. **Adjust learning rate**

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor \frac{t}{\text{step size}} \rfloor}$$

This approach ensures a stable training process by gradually decreasing the learning rate while optimizing model weights using gradient descent.