Machine learning - Linear regression

Botond Koroknai

2025.08.24

1 Introduction

Linear regression is one of the most fundamental and widely used models in the field of statistics and machine learning.

It can be used to model the relationship between input variables (features) and a continuous output variable (target).

At first, we have to assume that this relationship can be approximated by a straight line (if we are talking about a single feature) or a hyperplane (in case of multiple features).

1.1 Why is linear regression considered fundamental in the world of machine learning?

- **Simplicity:** The mathematical formulation of linear regression is straightforward, making it an ideal starting point for understanding more complex models.
- **Interpretability:** The learned parameters (weights) directly indicate the contribution of each input feature to the output prediction, providing clear insights into the data.
- **Practical relevance:** Despite its simplicity, linear regression is highly effective for many real-world problems where relationships are approximately linear or where interpretability is essential.

2 Mathematical background

As previously mentioned, the idea behind using linear regression is to approximate the relationship between the inputs (features) and the output (target) using a linear function.

This way the output can be expressed as a weighted combination of the input features plus a constant, which we call bias.

2.1 Single Feature (1D case):

$$\hat{y} = wx + b$$

where

- w is the slope of the line, determining how strongly x influences the output,
- b represents the value of $\hat{y}$ when $x = 0$,
- $\hat{y}$ is the predicted value for the given input x.

In this case, the model corresponds to finding the best line that passes as closely as possible through the data points in a two-dimensional plane.

2.2 Multiple Features (vector form)

When we have more than one input feature

$$x = (x_1, x_2, \ldots, x_d)$$

the linear model generalizes to:

$$\hat{y} = x_1 w_1 + x_2 w_2 + \cdots + x_d w_d + b$$

Which can be written up in a vector notation as:

$$\hat{y} = \boldsymbol{x}^\top \boldsymbol{w} + b$$

where

- $\boldsymbol{x} \in \mathbb{R}^d$ is the feature vector,
- $\boldsymbol{w} \in \mathbb{R}^d$ is the weight vector,
- $b \in \mathbb{R}$ is the bias term.

Finally, when we consider all n training examples together, it is convenient to organize them into a single data matrix

$$X \in \mathbb{R}^{n \times d}$$

where each row corresponds to one training example and each column corresponds to a particular feature across all examples:

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

The weight vector $\boldsymbol{w} \in \mathbb{R}^d$ contains one parameter for each feature, and $b \in \mathbb{R}$ is the bias term. Using matrix multiplication, we can compute the predicted outputs for all training examples simultaneously as

$$\hat{\boldsymbol{y}} = X\boldsymbol{w} + b\mathbf{1}$$

where $\hat{\boldsymbol{y}} \in \mathbb{R}^n$ is the vector of predictions.

Using such a representation comes with several advantages:

- **Efficiency:** Modern numerical libraries (like NumPy) are optimized for matrix operations, allowing all predictions to be computed at once without explicit loops.
- **Clarity:** The notation clearly expresses the relationship between the data, the weights, and the predictions in a single formula.

In any case we are interested in a specific prediction $\hat{y}^{(i)}$ for a single example i, it can be quickly calculated using the

$$\hat{y}^{(i)} = (\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b$$

formula.

3 Loss Function and Cost Function

In order to train a linear regression model, we must find out how accurately the model predicts the data. This process consists of two steps: first, defining a loss function for a single example. Then, defining a cost function for the entire dataset.

3.1 Loss Function

The **loss function** quantifies the error of the model on a *single training example*. For linear regression, the most common choice is the **squared error**:

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$

where $y^{(i)}$ is the true target value for the i-th example, and $\hat{y}^{(i)}$ is the corresponding prediction.

3.2 Cost Function

The **cost function** aggregates the loss over the entire training dataset. It is essentially the average of all individual losses, providing a single scalar value that represents the model's performance across all examples:

$$J(\boldsymbol{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2$$

3.3 Training Objective

The training objective is to find the parameters $\boldsymbol{w}$ and b that *minimize the cost function*:

$$(\boldsymbol{w}^*, b^*) = \arg\min_{\boldsymbol{w}, b} \underbrace{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2}_{\text{Cost function } J(\boldsymbol{w}, b)}$$

Minimizing the cost function ensures that the model's predictions are as close as possible to the true target values on average across all examples.

4 Numerical Interpretation

Once we have defined the loss function and the training objective, the next step is to determine how to find the optimal parameters $(\boldsymbol{w}^*, b^*)$ that minimize the cost function. There are two standard approaches in linear regression: a closed-form solution and an iterative approach.

4.1 Closed-Form Solution (Normal Equation)

For linear regression with the squared error loss, we can compute the exact optimal parameters directly using linear algebra. This method is called the **normal equation**.

Derivation

The cost function is

$$J(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)}\right)^2$$

To find the minimum, we take the partial derivatives of J with respect to $\boldsymbol{w}$ and b and set them equal to zero:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{2}{n}\sum_{i=1}^{n}\boldsymbol{x}^{(i)}\left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)}\right) = 0$$

$$\frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}\left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)}\right) = 0$$

The second equation gives the bias directly:

$$b^* = \bar{y} - \bar{\boldsymbol{x}}^\top \boldsymbol{w}^*, \quad \text{where } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)}, \quad \bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}^{(i)}.$$

Substituting b^* into the first derivative equation, and using matrix notation, we obtain:

$$X^\top(X\boldsymbol{w}^* + b^*\mathbf{1} - \boldsymbol{y}) = 0$$

Centering the data by subtracting the mean feature vector $\bar{\boldsymbol{x}}$ and the mean target $\bar{y}$ simplifies the equation to:

$$\tilde{X}^\top \tilde{X}\,\boldsymbol{w}^* = \tilde{X}^\top \tilde{\boldsymbol{y}}$$

where $\tilde{X} = X - \mathbf{1}\bar{\boldsymbol{x}}^\top$ and $\tilde{\boldsymbol{y}} = \boldsymbol{y} - \bar{y}\mathbf{1}$. Finally, solving for $\boldsymbol{w}^*$ gives the closed-form solution:

$$\boldsymbol{w}^* = (\tilde{X}^\top \tilde{X})^{-1}\tilde{X}^\top \tilde{\boldsymbol{y}}$$

and the bias is recovered as

$$b^* = \bar{y} - \bar{\boldsymbol{x}}^\top \boldsymbol{w}^*$$

This derivation shows that the optimal weights are found by projecting the target vector y onto the space formed by the features, which gives predictions that are as close as possible to the true values in terms of squared error.

4.2 Iterative Solution (Gradient Descent)

Gradient descent is an iterative method to find the optimal parameters $(\boldsymbol{w}^*, b^*)$ that minimize the cost function. Instead of solving a closed-form equation, we start with an initial guess for $\boldsymbol{w}$ and b, and iteratively adjust them to reduce the prediction error. As a reminder, the cost function is

$$J(\boldsymbol{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)} \right)^2$$

4.2.1 Step 1: Compute gradients

To update the parameters, we compute the partial derivatives of the cost function with respect to $\boldsymbol{w}$ and b. Using calculus:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \left[\frac{1}{n} \sum_{i=1}^{n} \left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)} \right)^2 \right] = \frac{2}{n} \sum_{i=1}^{n} \left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)} \right) \boldsymbol{x}^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{\partial}{\partial b} \left[\frac{1}{n} \sum_{i=1}^{n} \left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)} \right)^2 \right] = \frac{2}{n} \sum_{i=1}^{n} \left((\boldsymbol{x}^{(i)})^\top \boldsymbol{w} + b - y^{(i)} \right)$$

These gradients quantify how much the cost function changes with respect to each parameter, also they indicate the direction in which the parameters should be adjusted to reduce the error.

4.2.2 Step 2: Update rules

We update the parameters by moving in the direction opposite to the gradient:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial J}{\partial \boldsymbol{w}}, \quad b \leftarrow b - \eta \frac{\partial J}{\partial b}$$

where $\eta > 0$ is the learning rate, controlling the size of each step.

4.2.3 Step 3: Matrix representation

For computational efficiency, the gradient can be expressed in matrix notation. Let $\hat{\boldsymbol{y}} = X\boldsymbol{w} + b\mathbf{1}$ be the vector of predictions for all n training examples. Then the gradients become:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{2}{n} X^\top (\hat{\boldsymbol{y}} - \boldsymbol{y}), \quad \frac{\partial J}{\partial b} = \frac{2}{n} \mathbf{1}^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

where $X \in \mathbb{R}^{n \times d}$ is the feature matrix, $\boldsymbol{y} \in \mathbb{R}^n$ is the vector of target values, and $\mathbf{1} \in \mathbb{R}^n$ is a vector of ones.
By substituting back the values, the update rules in matrix form are then:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{2}{n} X^\top (\hat{\boldsymbol{y}} - \boldsymbol{y}), \quad b \leftarrow b - \eta \frac{2}{n} \mathbf{1}^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

The overall goal is to move iteratively in the direction opposite to the gradient, thereby minimising the cost function.