Multiple Linear Regression

▼ Multiple Features (variables)

- The original linear regression model uses a single feature (e.g., size of a house) to predict an outcome (e.g., price).
- By adding more features (e.g., number of bedrooms, floors, and age of the house), the model can make more accurate predictions

Notation:

	Size in feet² x_1	Number of bedrooms x_2	Number of floors x_3	Age of home in years x_4	Price ($) in $1000's
	2104	5	1	45	460
$i=2$	1416	3	2	40	232
	1534	3	2	30	315
	852	2	1	36	178
	...	...	...	...	...

$j=1\ldots4$
$n=4$

$x_j = j^{th}$ feature
$n =$ number of features
$\vec{x}^{(i)} =$ features of i^{th} training example
$x_j{}^{(i)} =$ value of feature j in i^{th} training example

💡 **Examples:**

$\vec{x}^{(2)} = [1416 \quad 3 \quad 2 \quad 40]$

$x_3{}^{(2)} = 2$

Model

Previous single variable model

$f_{w,b}(x) = wx + b$

Updated multivariable model

$f_{w,b}(x) = w_1 x_1 + \ldots + w_n x_n + b$

example

$$f_{w,b}(x) = 0.1 \underset{\text{size}}{x_1} + 4 \underset{\text{\#bedrooms}}{x_2} + 10 \underset{\text{\#floors}}{x_3} + -2 \underset{\text{years}}{x_4} + \underset{\text{base price}}{80}$$

The model can be simplified using vectors of parameters and features:

$$\vec{w} = [w_1 \quad w_2 \quad w_3 \quad \ldots \quad w_n]$$
$$\vec{x} = [x_1 \quad x_2 \quad x_3 \quad \ldots \quad x_n]$$

b, however, is included in the complete model, which is described below.

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

multiple linear regression (not multivariate function)

> 💡 *NOTE:* "·" is *dot product.*

Vectorization

- Vectorization simplifies code by allowing operations on entire arrays or vectors at once, rather than using loops.
- It enhances performance by leveraging modern numerical libraries like NumPy, which can utilize parallel processing capabilities of CPUs and GPUs.

Notice that the linear algebra is 1-based indexing, while the Python code is 0-based indexing.

Vector	Python code

Parameters and features
$\vec{w} = [w_1 \quad w_2 \quad w_3]$
$\vec{x} = [x_1 \quad x_2 \quad x_3]$
$b = 4$
 	
with $w_1 = 1.0,\ w_2 = 2.5,\ w_3 = -3.3$
and $x_1 = 10,\ x_2 = 20,\ x_3 = 30$
and b is a number.
	`w = np.array([1.0, 2.5, -3.3])`
	`x = np.array([10, 20, 30])`
	`b = 4`

Without vectorization
$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$
	`f = w[0] * x[0] +`
	`    w[1] * x[1] +`
	`    w[2] * x[2] + b`

$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j\right) + b$

```
f = 0
for i in range(0, n):
  f = f + w[i] * x[i]
f = f + b
```

Vectorization

$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

```
f = np.dot(w, x) + b
```

▼ Gradient Descent for Multiple Linear Regression

The cost function J is now a function of the vector w and the scalar b.

Pre-derived gradient descent algorithm

repeat until convergence:

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

Final gradient descent algorithm

$w_j = w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^i) - y^i) {x_j}^{(i)}$

$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^i) - y^i)$

simultaneously update

w_j (for $j = 1, \ldots, n$) and b

Normal Equation Method

An alternative to gradient descent

- Normal Equation:
  - Only for Linear Regression.
  - Solves for w and b without iterations.

$$\boxed{\theta = \left(X^T X\right)^{-1} . \left(X^T y\right)}$$

source: GeeksForGeeks 30/05/2025

- Disadvantage:
  - Doesn’t generalize to other learning algorithms.
  - Slow when the number of features is large (> 10,000)
- An advanced linear algebra library is required, so engineers hardly utilize the normal equation directly.
- It is usually used on the backend of machine learning libraries that implement linear regression.
- For most learning algorithms, gradient descent is often the better way to get the job done