Linear Regression

Aaryan Kaushik

July 18, 2023

Abstract

Let's start by talking about the most basic and certainly one of the important algorithms in machine learning: **Linear Regression**.

1 Introduction

We all have heard of the word "Experience". This word, in general, is used to describe our cognitive abilities. *But*, is there any mathematical way to describe or measure experience 🤔?

Interestingly, yes!! This is all what Supervised Learning is about. It is a way to feed a dumb computer with labeled data and allow it to learn from the data. A machine learning algorithm is the way by which the computer is going to learn from the data.

If it is not clear, let's look at an example of property dealers. A property dealer can tell us an almost accurate price of a house just by looking at some of its features. Some of those features might be the land size, the location, number of floors etc. How do they do it? Well, they have "experience" of how much a similar kind of house costs. The more the features of one house match with the other, the closer their prices would be.

2 A Simple Example

Let's take a look at the following data:

Area (sq. feet)	**Price (100 $)**
2104	400
1600	330
2400	369
1416	232
3000	540

Now, we want to predict the price of a house which has an area different from all the areas in above data. The way we are going to do the same is to find a mapping from the area of the house to the price of the house.
In other words, we want to find $f : A \to P$, where A is the area and P is the price of the house.
Note: Our goal is not to find a function that exactly maps the area to the price. Rather, we want to find a function that will tell an appropriate prediction of price.
For example, let's try to fit a straight line through the data.

Here is the plot of the above data:

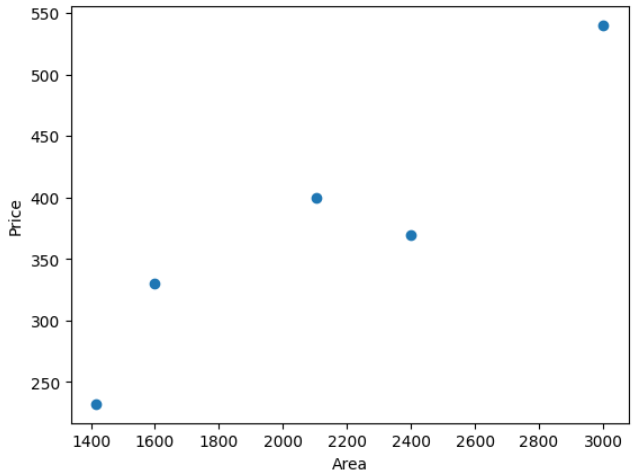


Figure 1: Graph of the data

Of course, this data doesn't lie on a straight line, but as mentioned before, we want to find a straight line that will give us an appropriate prediction of price.

To understand it visually, let's take a look at the following graph:

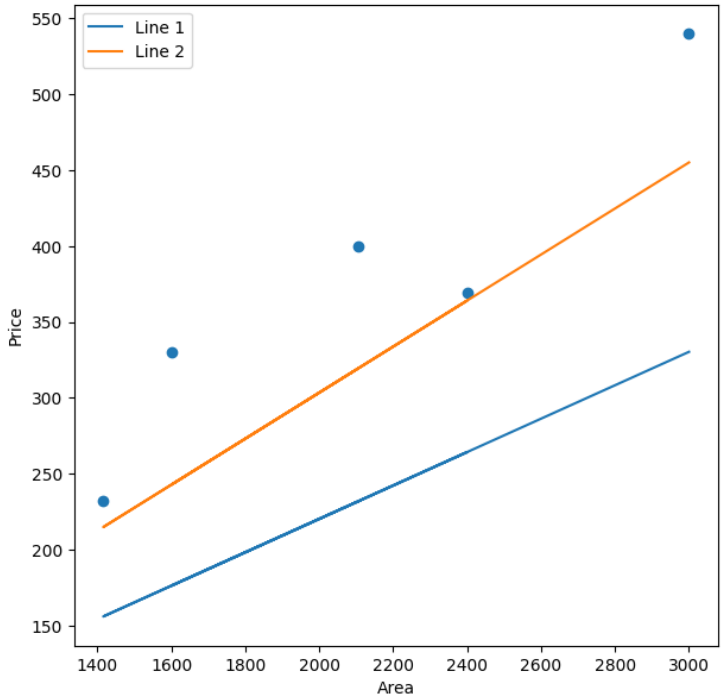


Figure 2: Orange line is a better fit as compared to blue line

Here is the mathematical way to find the most appropriate line:
The general equation for straight line is $y = c + mx$
In this example x denotes the area and y denotes the price of the house.
Following the linear regression literature, we are going to write this as

$$y = \theta_0 + \theta_1 x \tag{1}$$

Since this function is used to predict the output given a new input, it is often called the hypothesis function.

$$h(x) = \theta_0 + \theta_1 x \tag{2}$$

Now, given a data, how do we pick, or learn, the parameters θ_0 and θ_1 ?
One reasonable way is to pick parameters such that for all x in the data, $h(x)$ is closest to y, where y is the price of a house of area x (y is given in the data).
To formalize this, we will define a function that measures, for each value of (θ_0, θ_1), how close the $h(x^{(i)}$'s are to the corresponding $y^{(i)}$'s.
We define the cost function as:

$$J(\theta_0, \theta_1) = \frac{1}{2} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2 \tag{3}$$

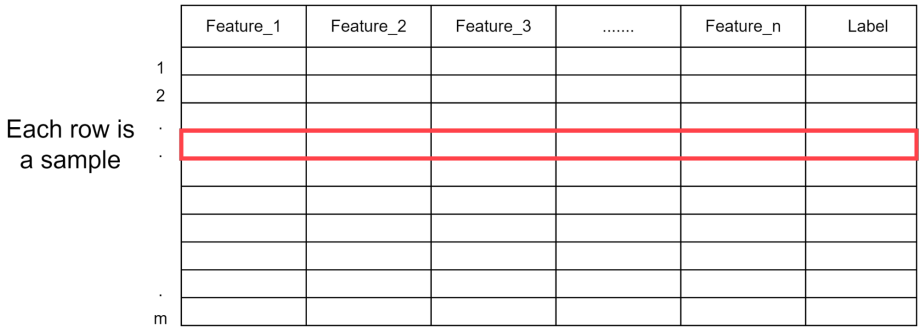
Here m is the total number of samples given in the data. This function is known as the **least square function** or **loss function**. The less the value of loss, the more appropriate the value of (θ_0, θ_1).
To minimize the value of the loss, gradient descent is used which will be discussed in the next section.

3 Generalized Version

In this section, we are going to generalize this method for larger dimensions.

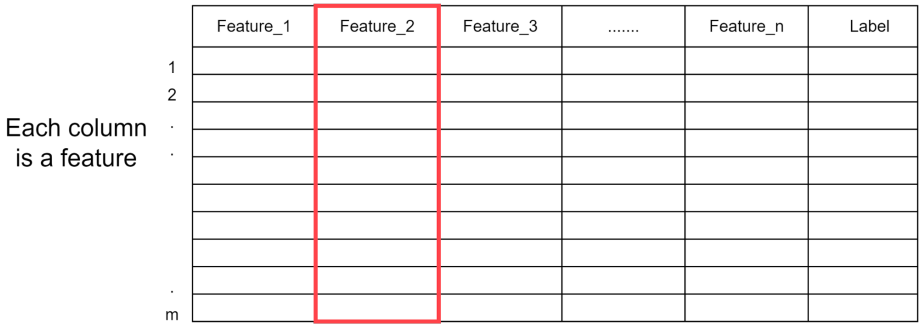
Before that, let's look into an example dataset and understand some terminologies.

- **Sample**



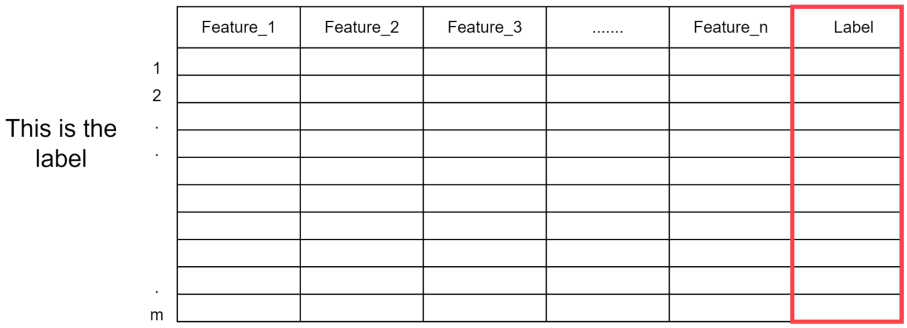
Each row in the dataset represents a sample of the data.

- **Feature**



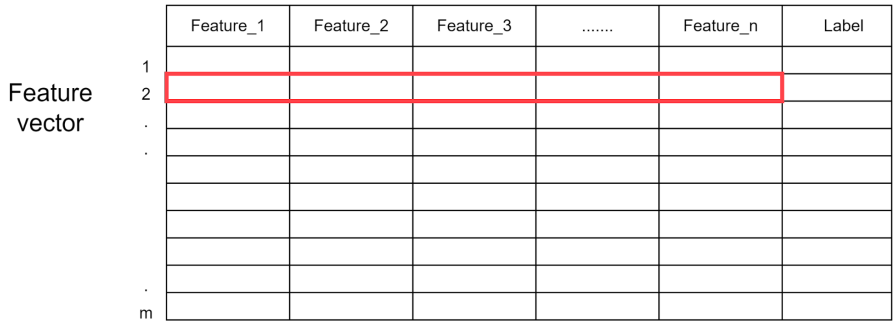
Each column, except the last one, is a feature of the data.

- **Output Label**



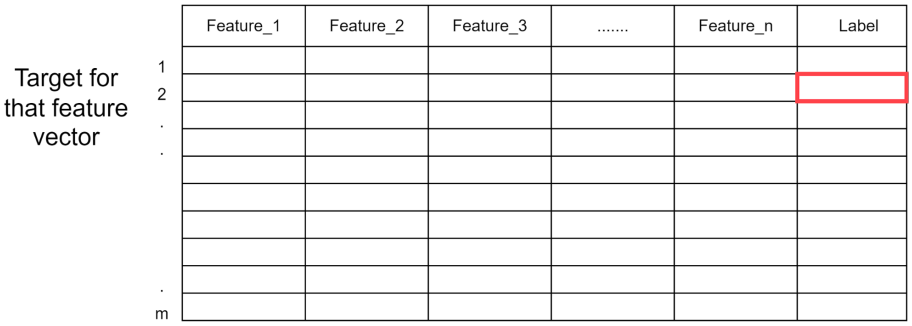
The last column is the output label which we want our model to predict.

- **Feature Vector**



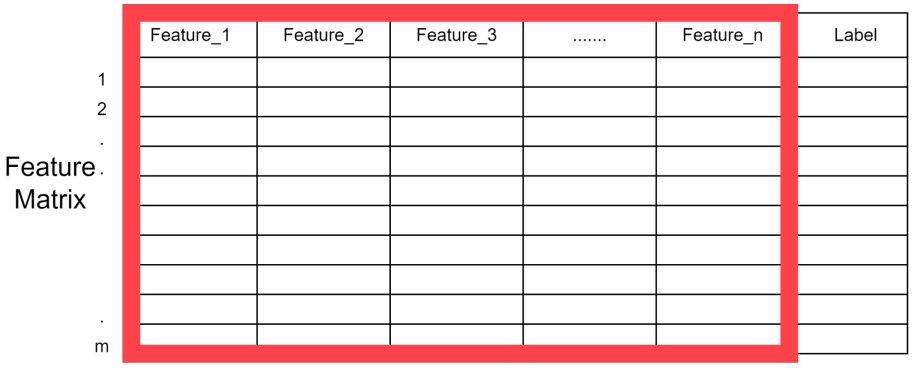
A vector that consists of all the features of a particular sample is called the feature vector of that sample.

- **Target Label**



The label of a sample is the target label of the corresponding feature vector.

- **Feature Matrix** X



The matrix highlighted in the above figure is known as the feature matrix. Its dimensions are (m, n), where m is the number of samples and n is the number of features of the dataset.

- **Target Vector** y

Target vector, y

	Feature_1	Feature_2	Feature_3	.....	Feature_n	Label
1						
2						
.						
.						
.						
m						

The vector highlighted in the above figure is known as the target vector. It is one dimensional with a size of m.

Let's denote the parameters by a vector consisting of $n+1$ values.

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Let's take a sample $X^{(i)} = [X_1, X_2, X_3, ..., X_n]$

To make the dimensions of $X^{(i)}$ and θ equal, let's insert 1 in $X^{(i)}$.
Therefore, $X^{(i)} = [1, X_1, X_2, X_3, ..., X_n]$

We define the hypothesis function as follows:

$$h_\theta(X^{(i)}) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n = X^{(i)}\theta$$

Goal: Calculate the value of which best fits the approximation -

$$h_\theta(X^{(i)}) \approx y^{(i)}$$

To do this, we define the **loss function** as follows:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(X^{(i)}) - y^{(i)})^2 \tag{4}$$

Let's say that J(θ) is minimum for $\theta = \theta^*$, then θ^* is the optimal parameter vector.

There are two approaches to minimize or converge the cost function:

1. Gradient Descent Algorithm

The basic idea behind gradient descent is to adjust the parameters of a model iteratively in the direction of steepest descent of the cost function. The algorithm starts by initializing the model's parameters randomly or with some predefined values. It then computes the gradient of the cost function with respect to these parameters.
The direction of steepest descent is the direction opposite to the gradient. A hyperparameter α known as the learning rate determines the size of the step taken in this direction.

Therefore, the update equation becomes:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{5}$$

Calculating the gradient for i^{th} sample:

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j}(\frac{1}{2}(h_\theta(X^{(i)}) - y^{(i)})^2 \\
&= 2.\frac{1}{2}.(h_\theta(X^{(i)}) - y^{(i)}).\frac{\partial}{\partial \theta_j}(h_\theta(X^{(i)}) - y^{(i)}) \\
&= (h_\theta(X^{(i)}) - y^{(i)}).\frac{\partial}{\partial \theta_j}(\sum_{j=0}^{n}(\theta_j^{(i)} X_j^{(i)} - y_j^{(i)})) \\
&= (h_\theta(X^{(i)} - y^{(i)}).X_j^{(i)}
\end{aligned}$$

The algorithm stops when the gradient approaches 0 or when the specified number of iterations are reached. When gradient approaches 0, the value of the loss function converges.

```
    eps = 1.0e-10  #Convergence limit
    alpha = 0.01       #Learning rate
    theta = 0         # Initialize all parameters by 0
    j0 = J(theta)   #Initial loss
    conv = INFINITY    #Initial convergence
    while |conv| > eps:
        for j = 1 to n:
                update(theta[j])
         j1 = J(theta)
        conv = j1-j0
        j0 = j1
```

2. Normal Equation Method

In this method, we equate the gradient of $J(\theta)$ to 0 and get the optimal θ.

$$\boldsymbol{s_\theta J(\theta)} = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$$

By equating $s_\theta J(\theta) = 0$, we get $\theta^* = (X^T X)^{-1} X^T y$
In this method we have to calculate the inverse of $X^T X$, which is a matrix of dimensions (n, n). This operation takes $O(n^3)$ computations.
Therefore the normal equation method is not suitable for a dataset which has a large number of features.

Link for code implementation