Theory

Erik An

<2025-12-22 Mon>

Contents

1 On This Project

This project focuses on implementing a simple linear regression model. Here, I will delve deeper than the summary provided in README.org. Specifically, we will explore:

- The mathematical intuition behind Linear Regression.
- Defining "error" using the Mean Squared Error (MSE) Cost Function.
- The Gradient Descent algorithm and how it minimizes error.
- The importance of Data Scaling (specifically Standardization).
- Evaluating model accuracy using the coefficient of determination (R^2).

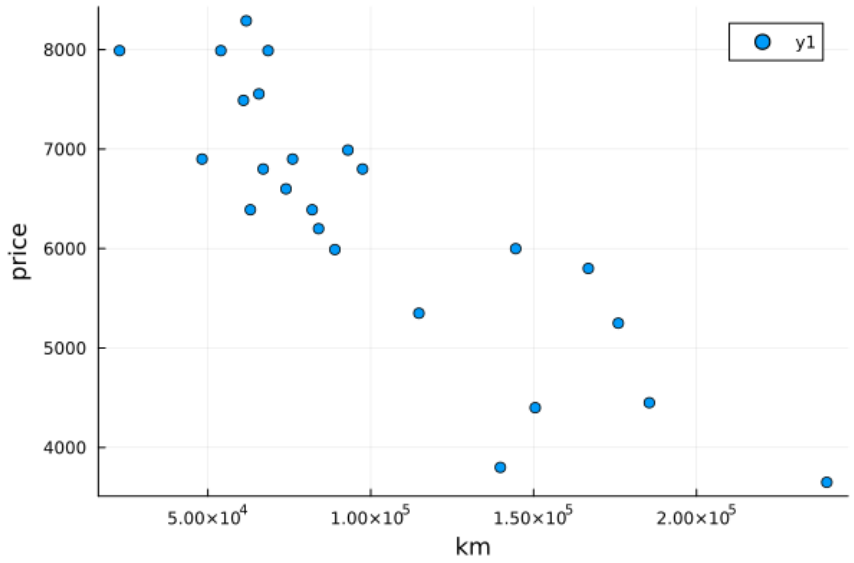
2 Linear Regression

$$y = \theta_1 x + \theta_0$$

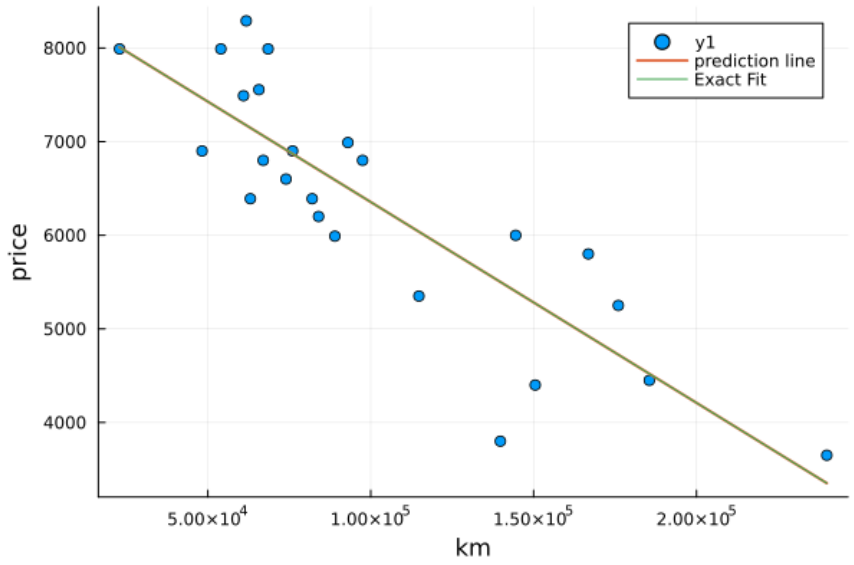
Does this formula look familiar to you?

Indeed, this is the super basic formula for a line that everyone (I hope) studied in middle school. It serves as the cornerstone of this project. It sounds very promising, but this equation is just the tip of the iceberg.

The goal of machine learning is to "teach" a computer to recognize patterns in a set of data. Take a look at the plot of the data used for this project below:



Do you see a pattern between the mileage of a car and its price? Yep! You could technically try to draw a "best fit line" by hand, and it would likely be a pretty fine approximation:



This is exactly the main goal of linear regression: to find the optimal line that best fits the scattered data points. We then use this line's slope (θ_1,

known in ML as the **weight**) and y-intercept (θ_0, known in ML as the **bias**) to estimate the price of a car (y) from a given mileage (x).

3 The Cost Function

There are multiple ways to find this best fit line. But before we can find the "best" line, we first need to define what "best" actually means mathematically.

We need a way to measure how "wrong" our line is at any given moment. To do this, I utilized a **Cost Function**, specifically the Mean Squared Error.

The formula looks like this:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

Where:

- m is the number of samples in the dataset
- $\hat{y}_i$ is the predicted value (the point on our line)
- y_i is the actual value (the real price from the dataset)

In simple terms, we calculate the difference between every real data point and our line, square it (to punish large errors more), and average it out. Our goal for the next step is simple: find the specific θ_0 and θ_1 values that result in the smallest possible error.

4 Gradient Descent Algorithm

The below formulas describe **how** to get to that smallest possible error. We update our values iteratively:

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)x_i$$

Where α is the **learning rate**.

Hint:

- θ_0: Here I slowly move the y-intercept (bias) up or down. I calculate the average difference between my line and the real points, and subtract it from the current θ_0.

  Explaining super simply: if your line is on average 100 units above the actual values, you just need to lower the whole line by 100.

- θ_1: Here I adjust the slope (weight). Notice that we multiply the error by x_i? This is because the input value (x) acts like a "lever".

  Explaining super simply: We are rotating the line. If the slope is wrong, the error will be huge for cars with high mileage (x), but small for cars with 0 mileage. So, we listen more to the high-mileage points to decide how much to rotate the line.

5 Data Preprocessing (Scaling)

Imagine if the x and y values in our dataset were in the hundreds of thousands.

If we plug these huge numbers directly into our gradient descent formulas, the error would be astronomical. This causes the math to become unstable—a phenomenon often called "exploding gradients"—making the model impossible to train.

There are generally two ways to fix this:

1. Drastically decrease the learning rate (α).

2. Decrease the scale of the initial x and y values.

While decreasing the learning rate is a working solution, it forces the computer to take microscopic steps, meaning the training could take forever. Hence, the industry standard is the second option: scaling the data down to a manageable range.

There are multiple ways to scale data, but for this project, I chose the **Standardization** approach (often called Z-score Normalization).

Standardization basically shifts the center of our values (let's consider mileage, which ranges from 20,000 to 200,000) to 0, and scales the spread so that most values fall between -3 and 3.

We don't need to delve too deep into the statistics, but you can think of it as compressing the dataset into a small, neat box centered at zero. This allows us to work safely without any mathematical explosions.

Here is the exact formula used to make this possible:

$$z = \frac{x_i - \mu}{\sigma}$$

Where:

- z: The new standardized value (the "Z-score").
- x_i: The original data point (initial value).
- μ (mu): The mean (average) of the dataset.
- σ (sigma): The standard deviation (a measure of how spread out the data is).

6 Evaluation

Well, during my explanation of Linear Regression at the start, you might have asked yourself: "How exactly do I know if my hand-drawn line is actually the 'best fit' line?"

There are, again, multiple ways to determine this. One of the most common methods is calculating the R^2 **Score** (Coefficient of Determination).

The R^2 Score essentially measures how much better our regression line is at predicting values compared to the dumbest possible model: simply guessing the average price ($\bar{y}$) for every single car.

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$$

Where:

- R^2: Is, surprise surprise, the R^2 score.
- RSS (Residual Sum of Squares): The sum of the squared errors of **our** model. It measures what our model failed to predict.
- TSS (Total Sum of Squares): The sum of the squared errors of the **average line**. It measures the total variance in the dataset.
- $\frac{\text{RSS}}{\text{TSS}}$: This fraction represents the percentage of variance that our model **failed** to explain.

How to interpret the result:

Since we subtract that failure fraction from 1, we get a score that usually ranges from 0 to 1:

- **1.0 (100%)**: A perfect fit. Our model explains all the variation in the data.
- **0.0 (0%)**: Our model is exactly as effective as just guessing the average value for everything.
- **Negative**: Our model is so bad that it is actually worse than just guessing the average (this usually means the learning rate was too high or the math exploded).

7 Conclusion

And there you have it!

We started with a simple line equation from middle school, $y = \theta_1 x + \theta_0$, and transformed it into a fully functional machine learning model. We explored how to measure error using a Cost Function, how to minimize that error using Gradient Descent, how to safely handle large numbers with Standardization, and finally, how to prove our model works using the R^2 score.

While modern machine learning involves massive neural networks and complex architectures, the fundamental principles—optimization, cost functions, and data scaling—remain exactly the same. This simple linear regression is the first real step into that larger world.

I hope this explanation helped demystify the math behind the code!