S21CSE471: Statistical Methods in AI **Spring 2021**

Linear Regression

Prepared by: Kushal Borkar(2020701033), Shiv Sharma(2020201037), Neha Agarwal(2020201095)
Date: 25.01.2021

1 Introduction

We have seen how a Machine Learning Algorithm is formulated. Based on the given data containing features but each example is also associated with a label or target, we try to learn how to predict those label or target. These labels or target might be discrete or continuous.

In the case of regression, we try to predict a target numeric value(which is continuous). For example: predicting the price of the car, given the details such as mileage, age, brand, etc.

2 Definition and Formulation

One of the simple ways to solve this problem is to use Linear Regression. As the name suggests we build a system that takes the input variables X { vector $x \in \mathbb{R}^n$ } as input and predict the target numeric value { scalar $y \in \mathbb{R}$ }. The output of linear regression is a linear function by computing a weighted sum of the input features, plus a constant term called bias(also known as intercept).

Equation of Linear Regression:

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n, \tag{1}$$

where $\hat{y}$ is the predicted value, n is the number of features, x_i is the i^{th} input feature value and w_j is the j^{th} model parameter.

The bias term, w_0 is considered to determine all the potential "lines" including the lines passing through the origin. w_0 can be included in the W vector(which we will see in the vectorized form) when we do an augmentation to the input features, by inserting 1 at the beginning. So, let us consider that our original vector is $[x1, x2]^T$, so by a new vector $[1, x1, x2]^T$. This makes our notation simpler and easy to represent in vector form.

So, Eqn. (1) can be written in the vectorized form as:

$$\hat{y} = W^T \cdot \mathbf{x}, \tag{2}$$

where W^T is the transpose of the model's parameter vector, containing the bias term w_0 and the feature weights w_1 to w_n. x is the input feature vector, containing x_0 to x_n with x_0 always equal to 1.

Hence, the above equation is the Linear Regression Model, but we need to know how to train it. By training a model, we mean determining these weights, the parameter W^T so that the model best fits the training set.

3 Why MSE?

Let us consider that N data points defined as $x_i, y_i \forall\ i = 1, ..., N$ and we need to find a line such that it fits these given data points by using Eqn. (2).

But all the data point will not fit on the line. There could lead to errors, ϵ. This term represents error that comes from sources out of our control, causing the data to deviate slightly from their true position. Hence, we have:

$$y = W^T \cdot \mathbf{x} + \epsilon, \tag{3}$$

And our goal is to minimise this error ϵ. So, we would like to minimise the error across all data points. Our error metrics will be the differences between prediction($W^T \cdot \mathbf{x}$) and actual values(y).

We can use mean absolute error(MAE) for minimising the error. The MAE is also the most intuitive of the metrics since we're just looking at the absolute difference between the prediction($W^T \cdot \mathbf{x}$) and actual values(y), which can be written as follow:

$$\epsilon = (\frac{1}{n})\sum_{i=1}^{n} |y - \hat{y}|\,, \tag{4}$$

We use the absolute value of the error from Eqn. (4), the MAE does not indicate underperformance or overperformance of the model (whether or not the model under or overshoots actual data). Each error term contributes proportionally to the total amount of error, meaning that larger errors will contribute linearly to the overall error. Furthermore, handling the absolute or modulus operator in mathematical equations is not easy.

The mean square error is just like the MAE, but squares the difference before summing them all instead of using the absolute value.

$$\epsilon = (\frac{1}{n})\sum_{i=1}^{n}[\, y - \hat{y}]^{\,2} \tag{5}$$

The MSE will almost always be bigger than the MAE. For this reason, we cannot directly compare the MAE to the MSE. We can only compare our model's error metrics to those of a competing model. The effect of the square term in the MSE equation is most apparent with the presence of outliers in our data. While each error term in MAE contributes proportionally to the total error, the error grows quadratically in MSE. This ultimately means that outliers in our data will contribute to much higher total error in the MSE than they would the MAE.

4 MSE Loss and Cost Function

The mean squared error(MSE) determines the closeness of our regression model to that of given data points. We do this by taking the distances from the points to the regression line (errors) and squaring them. The squaring is necessary to remove any negative signs.

Let us consider, we have N data points defined as $x_i, y_i \forall\ i = 1, ..., N$. If we write the error in matrix format then, we have:

$$\begin{bmatrix}\epsilon_1\\ \epsilon_2\\ \vdots\\ \epsilon_n\end{bmatrix} = \begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix} - \begin{bmatrix}x_1^T\\ x_2^T\\ \vdots\\ x_n^T\end{bmatrix} \cdot W, \tag{6}$$

The loss or mean sum of sqaured error (MSE) can be written in the form(matrix form) as:

$$\begin{bmatrix}\epsilon_1\\ \epsilon_2\\ \vdots\\ \epsilon_n\end{bmatrix} = \begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix} - \begin{bmatrix}x_1^T\\ x_2^T\\ \vdots\\ x_n^T\end{bmatrix} \cdot W, \tag{7}$$

$$\tag{8}$$

In simplified terms:

$$E = Y - X\mathbf{w} \tag{9}$$

where E is the error, Y is actual value and $X\mathbf{w}$ is the predicted value. The loss function can be written as:

$$J = (\frac{1}{n})E^T E, \tag{10}$$

Our objective is to minimize the above loss given in Eqn (10) . We minimize over w. i.e, $J()$ is a function of $\mathbf{w}$ and the minima can be obtained by optimizing

$$J = \frac{1}{n}[Y - X\mathbf{w}]^T[Y - X\mathbf{w}] \tag{11}$$

where Y is a $N \times 1$ vector. X is a $N \times d$ matrix and w is a $d\times$ vector. Let us expand this a little for further use:

$$\begin{aligned} J &= \frac{1}{n}[Y - X\mathbf{w}]^T[Y - X\mathbf{w}] \\ &= \frac{1}{n}[Y^T - \mathbf{w}^T X^T][Y - X\mathbf{w}] \\ &= \frac{1}{n}[Y^T Y - Y^T X\mathbf{w} - \mathbf{w}^T X^T Y + \mathbf{w}^T X^T X\mathbf{w}], \end{aligned} \tag{12}$$

From the above equation, we can write $(Y^T X\mathbf{w})^T = \mathbf{w}^T X^T Y$. Now, this matrix is a 1×1 matrix, hence, $Y^T X\mathbf{w} = \mathbf{w}^T X^T Y$. Thus, Eqn. (12) can be written as:

$$J = \frac{1}{n}[Y^T Y - 2\mathbf{w}^T X^T Y + \mathbf{w}^T X^T X\mathbf{w}], \tag{13}$$

5 Minimizing the Cost Function

5.1 Normal Form

Normal Equation is an analytical approach to Linear Regression with a Least Square Cost Function. To find the value of w that minimizes J, there is a closed-form solution - a mathematical equation that gives the result directly. To get our result, we first find the gradient of the J with respect to w

$$\begin{aligned} \nabla J(w) &= \frac{1}{n}[\nabla Y^T Y - 2\nabla\mathbf{w}^T X^T Y + \nabla\mathbf{w}^T X^T X\mathbf{w}], \\ &= \frac{1}{n}[0 - 2X^T Y + 2X^T X\mathbf{w}], \\ &= \frac{1}{n}[0 - 2X^T Y + 2X^T X\mathbf{w}], \\ &= \frac{2}{n}[X^T X\mathbf{w} - X^T Y] \end{aligned} \tag{14}$$

We now set $\nabla J(w)$ to zero at the optimum,

$$\frac{2}{n}[X^T X\mathbf{w} - X^T Y] = 0, \tag{15}$$

or

$$\mathbf{w} = [X^T X]^{-1} X^T Y \tag{16}$$

This equation has a single solution if $[X^T X]^{-1}$ is invertible(non-singular). However, the solution will not be unique. We can get around the problem of $[X^T X]^{-1}$ being singular by using generalized inverses to solve the problem.

A matrix $X^\dagger$ is a generalized inverse of the matrix A if and only if it satisfies $AX^\dagger A = A$. So, using the definition of a generalized inverse, we can write a solution to the least squares equation as

$$\mathbf{w} = [X^T X]^\dagger X^T Y \tag{17}$$

5.1.1 Computation Complexity

The Normal Form computes the inverse of $[X^T X$, which is a $n \times n$ matrix. The computational complexity of inverting such matrix will $O(n^{2.4})$ to $O(n^3)$ depending on the implementation. This means the Normal form gets very slow when the number of features grows large. Figure 1 shows when the number of features grows the algorithm gets slower.

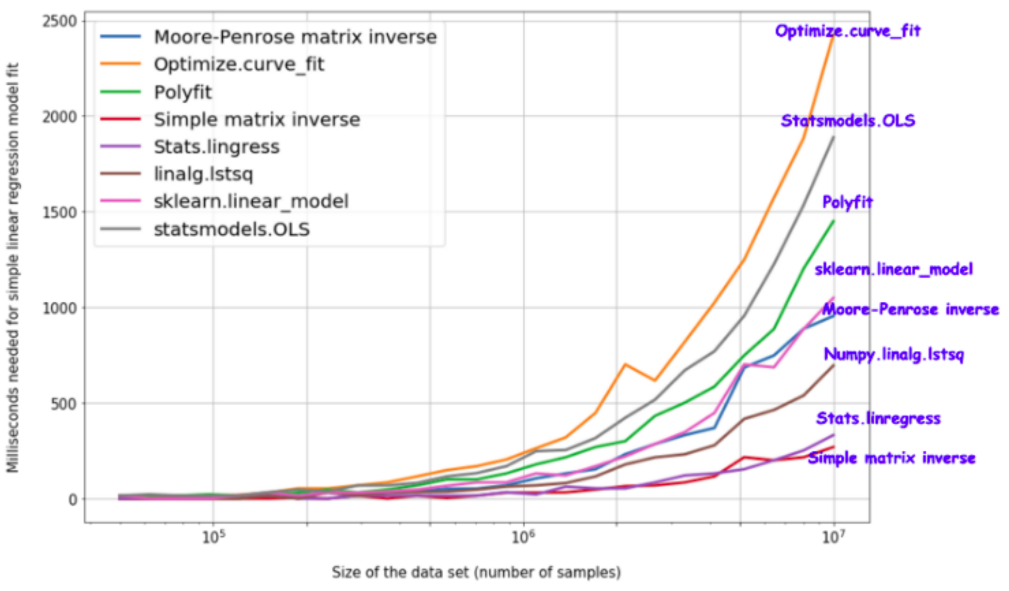


Figure 1: Computational Complexity Measure

5.2 Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). The general idea of Gradient descent is to tweak parameters iteratively in order to minimize J. It measures the local gradient of the cost function of the error function with regards to $\mathbf{w}$, and goes in the direction of the descending gradient. Once the gradient is zero, we have reached a minimum(Figure 2).

Algorithm 1 Gradient descent method.

```
Initialize $\mathbf{w}$ arbitrarily
repeat
   1. Incrementally change the solution by moving in the negative gradient of the cost function.
      $\mathbf{w}_{t+1} := \mathbf{w}_t - \lambda \Delta J(\mathbf{w})$
   2. $t \leftarrow t + 1$
until $\mathbf{w}_{t+1} \approx \mathbf{w}_t$
```

An important parameter in Gradient Descent is the size of the step(λ), which determines the learning rate of hyperparameter. If the learning rate is too small, then the algorithm will go through many iterations to converge, which will take a long time(Figure 3)

On the other hand, if the learning rate is too high, we might jump across the curve and end up on the other side, possibly even higher up than we were before. This might make the algorithm diverge, with larger value, failing to provide a good solution (Figure 3).

The cost function for the Linear Regression is Convex in nature(this means that if we pick two points on the curve, the line segment joining them never crosses the curve. This implies that there is one global minimum. It is also a continuous function).

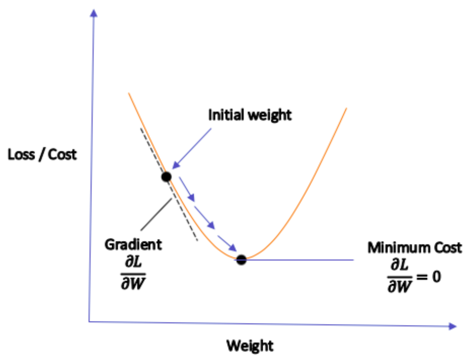

Loss / Cost
Initial weight
Gradient
$\frac{\partial L}{\partial W}$
Minimum Cost
$\frac{\partial L}{\partial W} = 0$
Weight

Figure 2: Gradient Descent

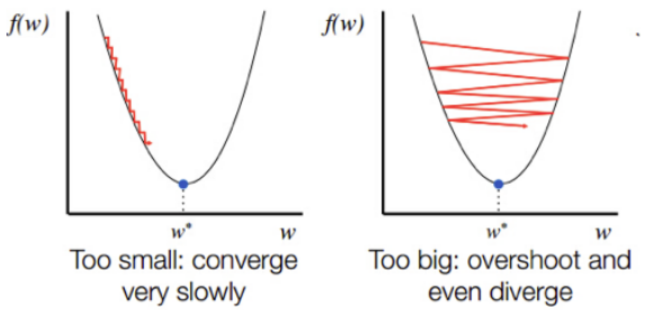

f(w)
w*
w
Too small: converge very slowly
f(w)
w*
w
Too big: overshoot and even diverge

Figure 3: Influence of Learning Rate Parameter

But if the features are not on the same scale, then we might not be able to converge faster. This is illustrated in the Figure 4:

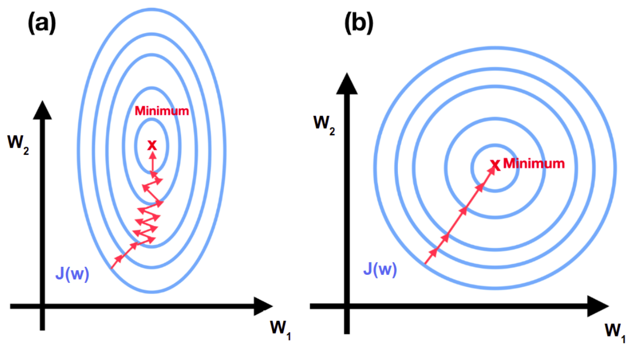


Figure 4: Gradient Descent with and without feature scaling

As we can see, in the Figure 4 (a), the Gradient Descent algorithm would take longer time to converge as the level curves (contours) are narrower and taller, whereas in Figure 4 (b), it converges faster as the level of contours are uniform, thereby reaching it quickly to minima.

5.2.1 Batch Gradient Descent

The batch gradient descent computes the gradient of the cost function by using the complete dataset available. It measures the local gradient of the cost function of the error function with regards to $\mathbf{w}$, and goes in the direction of the descending gradient. Once the gradient is zero, we have reached a minimum

$$W_j := W_j - \lambda \frac{1}{m} \sum_{i=1}^{m} \left(h_W \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \tag{18}$$

As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that don't fit in memory. Batch gradient descent also doesn't allow us to update our model online, i.e. with new examples on-the-fly.

5.2.2 Stochastic gradient descent

The downside of gradient descent is that we have to compute the sum of all the gradients before we update the weights. Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$

$$W_j := W_j - \lambda \left(h_W \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \tag{19}$$

Stochastic gradient descent (SGD) tries to lower the computation per iteration, at the cost of an increased number of iterations necessary for convergence. Instead of computing the sum of all gradients, stochastic gradient descent selects an observation uniformly at random. While this is a noisy estimator, we are able to update the weights much more frequency and therefore hope to converge more rapidly

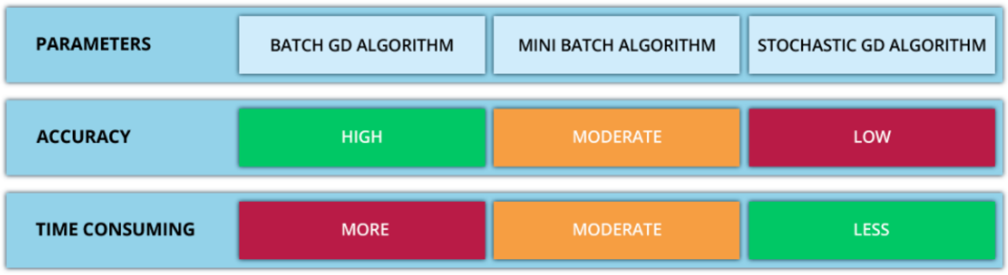
5.2.3 Mini-batch stochastic gradient descent (mini-batch SGD)

Mini-batch stochastic gradient descent is a compromise between full-batch iteration and SGD. A mini-batch is typically between 10 and 1,000 examples, chosen at random. Mini Batch Stochastic gradient descent (SGD) performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$ by computing the mean of gradient of the mini batch of size k

$$W_j := W_j - \lambda \frac{1}{k} \sum_{i=1}^{k} \left(h_W \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \tag{20}$$

Mini-batch SGD reduces the amount of noise in SGD but is still more efficient than full-batch. It reduces the variance of the parameter updates, which can lead to more stable convergence; and can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

Method	cost per iteration
GD	$\mathcal{O}(nd)$
SGD	$\mathcal{O}(d)$
MBSGD	$\mathcal{O}(kd)$



PARAMETERS	BATCH GD ALGORITHM	MINI BATCH ALGORITHM	STOCHASTIC GD ALGORITHM
ACCURACY	HIGH	MODERATE	LOW
TIME CONSUMING	MORE	MODERATE	LESS

Figure 5: Comparisons among different gradient descent algorithm

6 Regularization

It's a technique applied to Cost Function $J(\theta^1)$ in order to avoid Overfitting.

The core idea in Regularization is to keep more important features and ignore unimportant ones. The importance of feature is measured by the value of its parameter θ_j.

In linear regression, we modify its cost function by adding regularization term. The value of θ_j is controlled by regularization parameter λ. Note that m is the number of data and n is the number of features(parameters).

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \overset{\text{Regularization Term}}{\boxed{\underset{\text{Regularization Parameter}}{\boxed{\lambda}}\sum_{\underset{\xleftarrow{\text{start at } \theta_1}}{j=1}}^{n}\theta_j^{\,2}}}$$

For instance, if we want to get a better model instead of the overfitting one. Obviously, we don't need features X^3 and X^4 since they are unimportant. The procedure describes below.

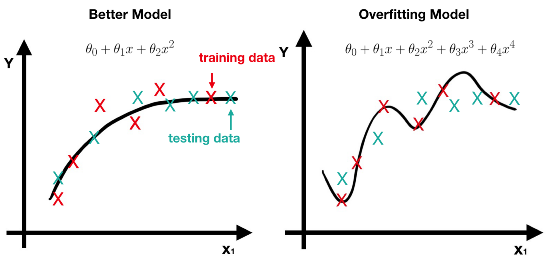


Figure 6:

First, we modify the Cost Function J(θ) by adding regularization. Second, apply gradient descent in order to minimize J(θ) and get the values of θ_3 and θ_4. After the minimize procedure, the values of θ_3 and θ_4 must be near to zero if λ=1000.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \overset{\text{Regularization Term}}{\boxed{1000\theta_3^{\,2} + 1000\theta_4^{\,2}}}$$

$$\min_{\theta} J(\theta), \text{ getting } \theta_3 \approx 0,\ \theta_4 \approx 0$$

Remember, the value of J(θ) represents training error and this value must be positive or equal to 0. The parameter λ=1000 has significant effect on J(θ), therefore, θ_3 and θ_4 must be near to zero (e.g 0.000001) so as to eliminate error value.

Positive

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\boxed{\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2} + 1000\boxed{\theta_3}^2 + 1000\boxed{\theta_4}^2$$

Regularization Parameter λ
If λ is too large, then all the values of θ may be near to zero and this may cause Underfitting. In other words, this model has both large training error and large prediction error. (Note that the regularization term starts from θ_1)
If λ is zero or too small, its effect on parameters θ is little. This may cause Overfitting.

To sum up, there are two advantages of using regularization:

[1]Note that here θ is equal to w.

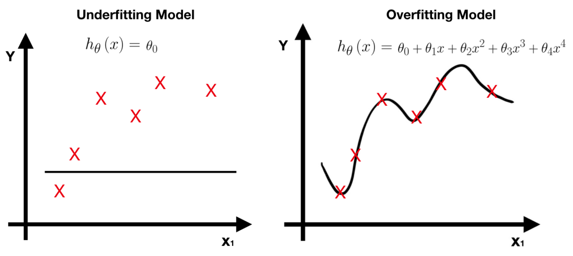

Underfitting Model
Y
$h_\theta(x) = \theta_0$
X₁
Overfitting Model
Y
$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
X₁

Figure 7:

- The prediction error of the regularized model is lesser, that is, it works well in testing data (green points).
- The regularization model is simpler since it has less features (parameters).

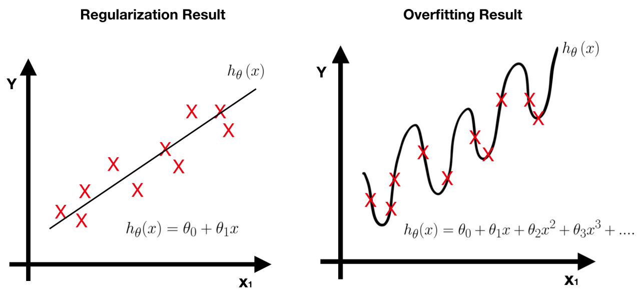

Regularization Result
Y
$h_\theta(x)$
$h_\theta(x) = \theta_0 + \theta_1 x$
X₁
Overfitting Result
Y
$h_\theta(x)$
$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ...$
X₁

Figure 8:

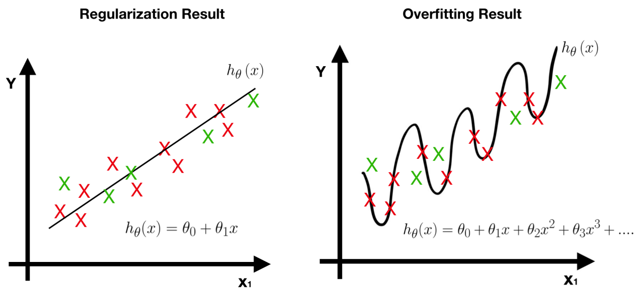

Regularization Result
Y
$h_\theta(x)$
$h_\theta(x) = \theta_0 + \theta_1 x$
X₁
Overfitting Result
Y
$h_\theta(x)$
$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ...$
X₁

Figure 9:

7 Linear Regression in Real Life

Linear regression is one of the most commonly used techniques in statistics. It is used to quantify the relationship between one or more predictor variables and a response variable. The most basic form of linear is regression is known as simple linear regression, which is used to quantify the relationship between one predictor variable and one response variable. If we have more than one predictor variable then we can use multiple linear regression, which is used to quantify the relationship between several predictor variables and a response variable.

Linear Regression Real Life Example 1

Businesses often use linear regression to understand the relationship between advertising spending and revenue. For example, they might fit a simple linear regression model using advertising spending as the predictor variable and revenue as the response variable. The regression model would take the following form:
The coefficient β_0 would represent total expected revenue when ad spending is zero.

$$revenue = \beta_0 + \beta_1 (ad\ spending)$$

The coefficient β_1 would represent the average change in total revenue when ad spending is increased by one

unit (e.g. one dollar).
If β_1 is negative, it would mean that more ad spending is associated with less revenue.
If β_1 is close to zero, it would mean that ad spending has little effect on revenue.
And if β_1 is positive, it would mean more ad spending is associated with more revenue.
Depending on the value of β_1, a company may decide to either decrease or increase their ad spending.

Linear Regression Real Life Example 2

Medical researchers often use linear regression to understand the relationship between drug dosage and blood pressure of patients.
For example, researchers might administer various dosages of a certain drug to patients and observe how their blood pressure responds. They might fit a simple linear regression model using dosage as the predictor variable and blood pressure as the response variable. The regression model would take the following form:
The coefficient β_0 would represent the expected blood pressure when dosage is zero.

$$\textit{blood pressure} = \beta_0 + \beta_1\textit{(dosage)}$$

The coefficient β_1 would represent the average change in blood pressure when dosage is increased by one unit.
If β_1 is negative, it would mean that an increase in dosage is associated with a decrease in blood pressure.
If β_1 is close to zero, it would mean that an increase in dosage is associated with no change in blood pressure.
If β_1 is positive, it would mean that an increase in dosage is associated with an increase in blood pressure.
Depending on the value of β_1, researchers may decide to change the dosage given to a patient.

Linear Regression Real Life Example 3

Agricultural scientists often use linear regression to measure the effect of fertilizer and water on crop yields.
For example, scientists might use different amounts of fertilizer and water on different fields and see how it affects crop yield. They might fit a multiple linear regression model using fertilizer and water as the predictor variables and crop yield as the response variable. The regression model would take the following form:
The coefficient β_0 would represent the expected crop yield with no fertilizer or water.

$$\textit{crop yield} = \beta_0 + \beta_1\textit{(amount of fertilizer)} + \beta_2\textit{(amount of water)}$$

The coefficient β_1 would represent the average change in crop yield when fertilizer is increased by one unit, assuming the amount of water remains unchanged.
The coefficient β_2 would represent the average change in crop yield when water is increased by one unit, assuming the amount of fertilizer remains unchanged.
Depending on the values of β_1 and β_2, the scientists may change the amount of fertilizer and water used to maximize the crop yield.

Linear Regression Real Life Example 4

Data scientists for professional sports teams often use linear regression to measure the effect that different training regimens have on player performance.
For example, data scientists in the NBA might analyze how different amounts of weekly yoga sessions and weightlifting sessions affect the number of points a player scores. They might fit a multiple linear regression model using yoga sessions and weightlifting sessions as the predictor variables and total points scored as the response variable. The regression model would take the following form:
The coefficient β_0 would represent the expected points scored for a player who participates in zero yoga sessions

$$\textit{points scored} = \beta_0 + \beta_1\textit{(yoga sessions)} + \beta_2\textit{(weightlifting sessions)}$$

and zero weightlifting sessions.
The coefficient β_1 would represent the average change in points scored when weekly yoga sessions is increased by one, assuming the number of weekly weightlifting sessions remains unchanged.
The coefficient β_2 would represent the average change in points scored when weekly weightlifting sessions is

increased by one, assuming the number of weekly yoga sessions remains unchanged.
Depending on the values of β_1 and β_2, the data scientists may recommend that a player participates in more or less weekly yoga and weightlifting sessions in order to maximize their points scored.

References

[1] More Math in Latex: https://ctan.math.illinois.edu/info/Math_into_LaTeX-4/Short_Course.pdf

[2] 3 Types of Gradient Descent: https://www.hackerearth.com/blog/developers/3-types-gradient-descent-algorithms-small-large-data-sets

[3] An overview of gradient descent optimization algorithms: https://ruder.io/optimizing-gradient-descent/

[4] Distributed Algorithms and Optimization: https://stanford.edu/ rezab/classes/cme323/S15/notes/lec11.pdf

[5] Overfitting and Regularisation: https://medium.com/@qempsil0914/courseras-machine-learning-notes-week3-overfitting-and-regularization-partii-3e3f3f36a287#:~:text=with%20proper%20features.-,Regularization,value%20of%20its%20parameter%20%CE%B8j.

[6] Real Life Uses of Linear Regression: https://www.statology.org/linear-regression-real-life-examples/