
MANIPAL
INSTITUTE OF TECHNOLOGY
*A Constituent Institute of Manipal University, Manipal*

# Use of Linear Regression in Machine Learning

**SUBMITTED BY**
Siddharth Biswas
130921302

# INDEX

# Introduction

**What is Machine Learning?**

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

A more modern definition of Machine Learning can be stated as follows ‘A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.’

In general, any machine learning problem can be assigned to one of two broad classifications:

- supervised learning OR
- unsupervised learning.

Supervised Learning:

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into **"regression"** and **"classification"** problems.

In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Unsupervised learning:

On the other hand, allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

We can derive this structure by clustering the data based on relationships among the variables in the data.

With unsupervised learning there is **no feedback** based on the prediction results, i.e., there is no teacher to correct.

# Requirements

In regression problems, we take input variables and try to fit the output onto a continuous expected result function

**Univariate Linear Regression(Linear Regression with one variable):**

**Hypothesis function:**

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

**Cost function:**

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

The measure the accuracy of hypothesis function is given by using a **cost function**.

**Gradient Descent:**

Put $\theta_0$ on the x axis and $\theta_1$ on the y axis, with the cost function on the vertical z axis. The points on the graph will be a result of the cost function using the hypothesis with these specific theta parameters.

We will know that we have succeeded when the cost function is at the very bottom of the pits in the graph, i.e. when its value is the minimum.

The way to do this is by taking the derivative (the tangential line to a function) of the cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent, and the size of each step is determined by the parameter α, which is called the learning rate.

The gradient descent algorithm is,

Repeat until convergence:

{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

}

This could be thought of as:

$$\theta_j := \theta_j - \alpha[\text{Slope of tangent aka derivative in j dimension}]$$

**Gradient Descent for Linear Regression:**

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived, given by:

Repeat until convergence, {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

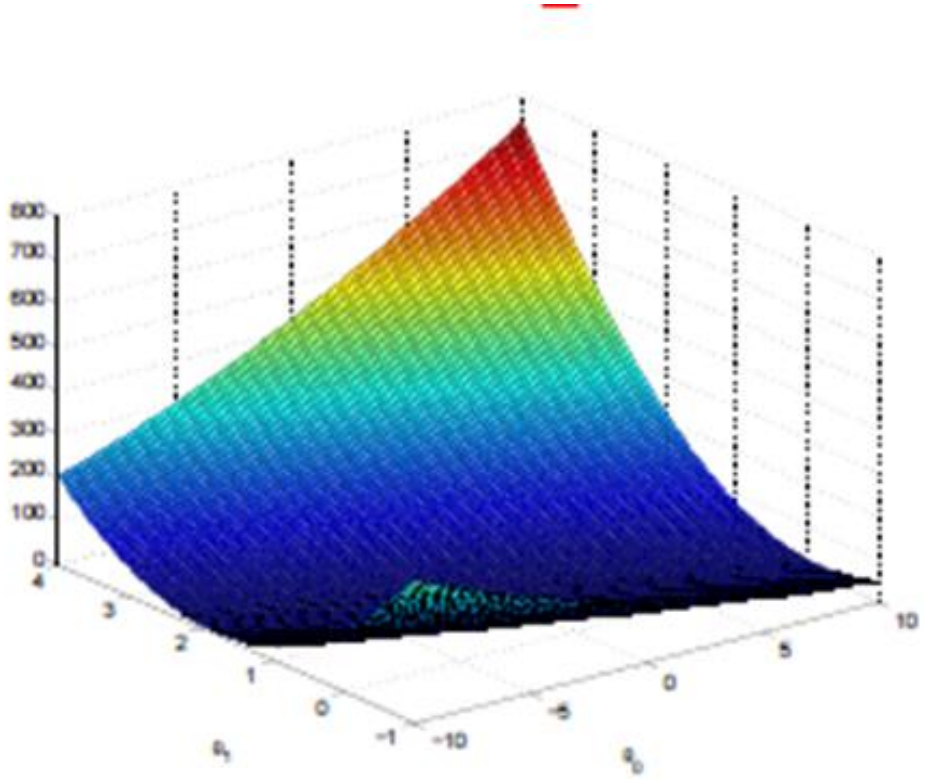
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

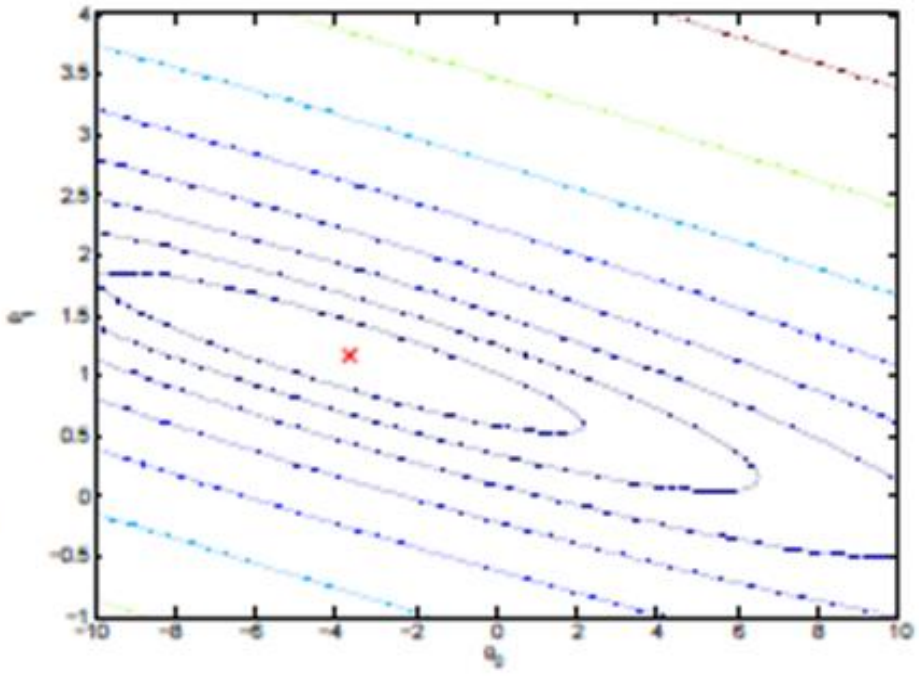
Here m is the number of training examples

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

Example of Gradient Descent Curve:




(a) Surface




(b) Contour, showing minimum

**Multivariate linear regression(Linear Regression with multiple variables):**

For a large number of input variables, the following notations are used:

$$
\begin{aligned}
x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\
x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\
m &= \text{the number of training examples} \\
n &= \left|x^{(i)}\right| \text{ ; (the number of features)}
\end{aligned}
$$

The multivariable form of the hypothesis function is defined as follows:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

In order to develop intuition about this function, we can think about $\theta_0$ as the basic price of a house, $\theta_1$ as the price per square meter, $\theta_2$ as the price per floor, etc. $x_1$ will be the number of square meters in the house, $x_2$ the number of floors, etc.
Using the definition of matrix multiplication, the multivariable hypothesis function can be concisely represented as:

$$h_\theta(x) = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

This is a vectorization of the hypothesis function for one training example

Now, vectorization for multiple training examples with multiple inputs is done in the following way:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$h_\theta(X) = X\theta$$

We calculate the hypothesis as a column vector of size (m x 1)

Cost function remains the same in Multivariate Linear Regression.
i.e,

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Gradient Descent for multiple variables is as follows:

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_2^{(i)}$$

...

}

In other words:

repeat until convergence: {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_j^{(i)} \qquad \text{for j} := 0..\text{n}$$

}

**Feature Normalization:**

It is used to speed up calculations involved in gradient descent.

This involves methods:

1.) Feature Scaling:

Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1.

2.) Mean Normalization:

Mean normalization involves subtracting the average value for an input variable from the values for that input variable, resulting in a new average value for the input variable of just zero.

Implementing both these techniques, adjust the input values as shown in this formula:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Example:

$x_i$ is housing prices with range of 100 to 2000, with a mean value of 1000. Then,

$$x_i := \frac{price - 1000}{1900}$$

**Polynomial Regression:**

The hypothesis function need not be linear (a straight line) if that does not fit the data well.
We can change the behavior or curve of the hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example:

If the hypothesis function is $h_\theta(x)=\theta_0+\theta_1 x_1$ then we can create additional features based on $x_1$, to get the quadratic function $h_\theta(x)=\theta_0+\theta_1 x_1+\theta_2 x^2_1$ or the cubic function $h_\theta(x)=\theta_0+\theta_1 x_1+\theta_2 x^2_1+\theta_3 x^3_1$
In the cubic version, we have created new features $x_2$ and $x_3$ where $x_2=x^2_1$ and $x_3=x^3_1$.
To make it a square root function, we could do $h_\theta(x)=\theta_0+\theta_1 x_1+\theta_2 x_1^{1/2}$

**Normal Equation:**

- The "Normal Equation" is a method of finding the optimum theta **without iteration.**
- There is **no need** to do feature scaling with the normal equation.
- With the normal equation, computing the inversion has complexity. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.
- Normal Equation is given by:

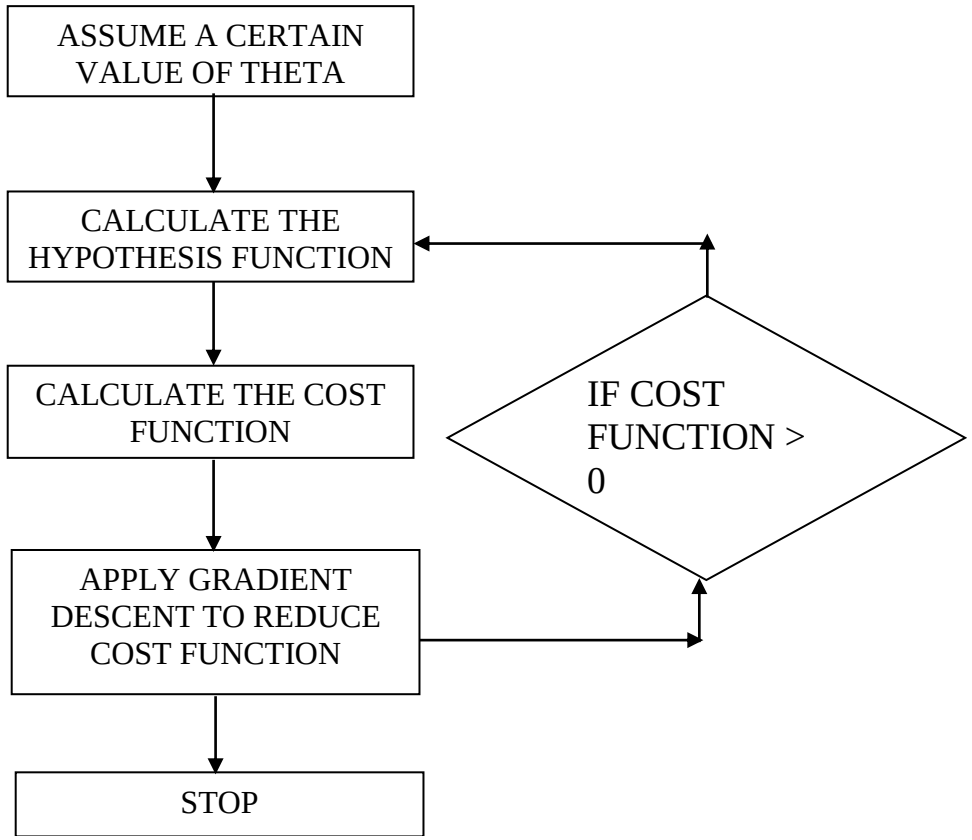
$$\theta = (X^T X)^{-1} X^T y$$

# Comparison

| Gradient Descent | Normal Equation |
| --- | --- |
| Need to choose alpha | No need to choose alpha |
| Needs many iterations | No need to iterate |
| No need to calculate inverse | Need to calculate inverse of $(X^TX)^{-1}$ |
| Works well when n is large | Slow if n is very large |

# Summarizing the steps involved in Linear Regression:

Fig:



# Advanced Optimization Algorithm:

MATLAB allows the use of certain in-built algorithms to find the optimum theta values. These include "Conjugate gradient", "BFGS", and "L-BFGS“ which are more sophisticated, faster ways to optimize $\vartheta$ that can be used instead of gradient descent.

For this we need to provide it with the following:

$$J(\theta)$$

$$\frac{\partial}{\partial \theta_j} J(\theta)$$

# Applications:

- Weather Forecasting
- Groundwater level Prediction
- Business Analytics
- Any application involving data prediction (continuous)

# Advantages

- Relatively easy to implement
- Can be used for large number of training examples with multiple features
- Vectorized implementation is possible on MATLAB which reduces computational speed
- The hypothesis function can be made into a non-linear curve by manipulating parameters theta.

# Disadvantages

- Fails in case of classification problems.
- Since it is implemented in business analytics, which is ever changing, features need to be continuously monitored and manipulated. This increases the complexity over a period of time.

# Conclusion

Understanding of linear regression has been developed from a basic level. A simple real life example of linear regression has also been mentioned.

Implementation of the algorithm has been completely done on MATLAB. This is because MATLAB allows Vectorized computation of iterative codes in a single line using matrix multiplication features, improving the computational speed.

Also we can have non-linear representation of hypothesis function by taking more number of features into consideration. Thus linear regression can be used in the field of engineering as well as analytics.

# Reference

- Course on Machine Learning by Andrew Ng

# THANK YOU!