Day 05 & 06

30 Days of Machine Learning

Multiple Linear Regression

Prepared By

Ahmed Ali

Quaid-e-Awam University of Engineering Science & Technology Nawabshah

Multiple Linear Regression:

In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used. Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable.

We can define it as.

“Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.”

Example: Prediction of CO2 emission based on engine size and number of cylinders in a car.

Some key points about MLR:

- For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.
- Each feature variable must model the linear relationship with the dependent variable.
- MLR tries to fit a regression line through a multidimensional space of data-points.

MLR equation:

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x1, x2, x3, ...,xn. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

Y= b₀+b₁x₁+ b₂x₂+ b₃x₃+..... bnxn (a)

Where,

Y= Output/Response variable

b0, b1, b2, b3 , bn = Coefficients of the model.

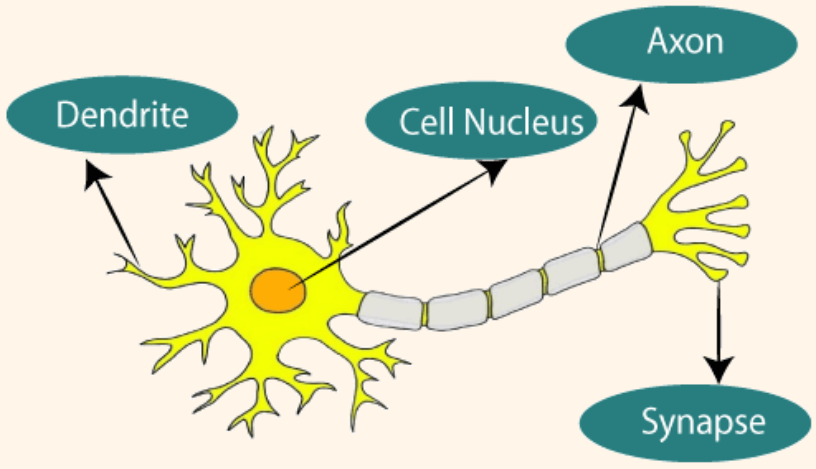
x1, x2, x3, x4, = Various Independent/feature variable

Assumptions for Multiple Linear Regression:

- A linear relationship should exist between the Target and predictor variables.
- The regression residuals must be normally distributed.
- MLR assumes little or no multicollinearity (correlation between the independent variable) in data.

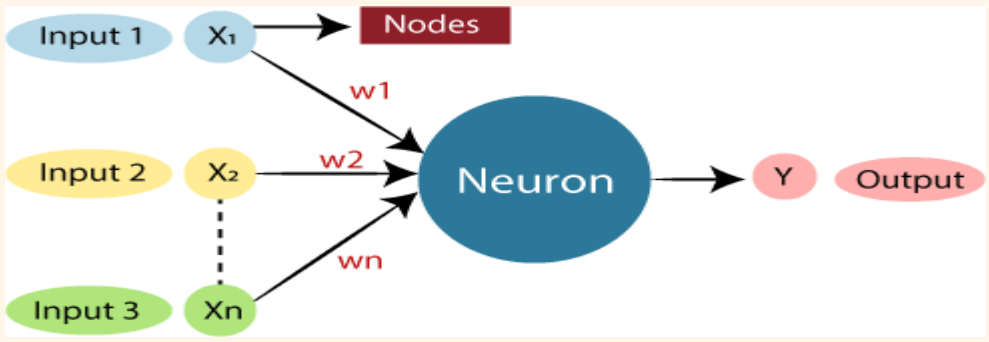
Neural Networks (ANN-Artificial Neural Network)

The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Like the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network.

The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

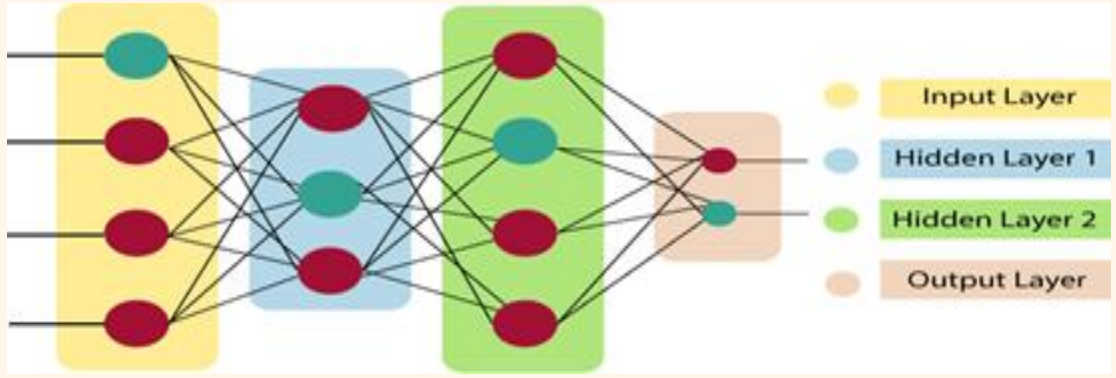
Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

Ahmed Ali

An Artificial Neural Network in the field of Artificial intelligence where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data, when necessary, from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.
We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The architecture of Artificial Neural Networks



Input Layer:
As the name suggests, it accepts inputs in several different formats provided by the programmer.
Hidden Layer:
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.
Output Layer:
The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.
The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing

Advantages of Neural Network:

Parallel processing capability:

Artificial neural networks have a numerical value that can perform more than one task simultaneously.

Storing data on the entire network:

Data that is used in traditional programming is stored on the whole network, not on a database. The disappearance of a couple of pieces of data in one place doesn't prevent the network from working.

Capability to work with incomplete knowledge:

After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.

Having a memory distribution:

For ANN is to be able to adapt, it is important to determine the examples and to encourage the network according to the desired output by demonstrating these examples to the network. The succession of the network is directly proportional to the chosen instances, and if the event can't appear to the network in all its aspects, it can produce false output.

Having fault tolerance:

Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.

Disadvantages of Neural Network:

Assurance of proper network structure:

There is no guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.

Unrecognized behavior of the network:

It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.

Hardware dependence:

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

Difficulty of showing the issue to the network:

ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

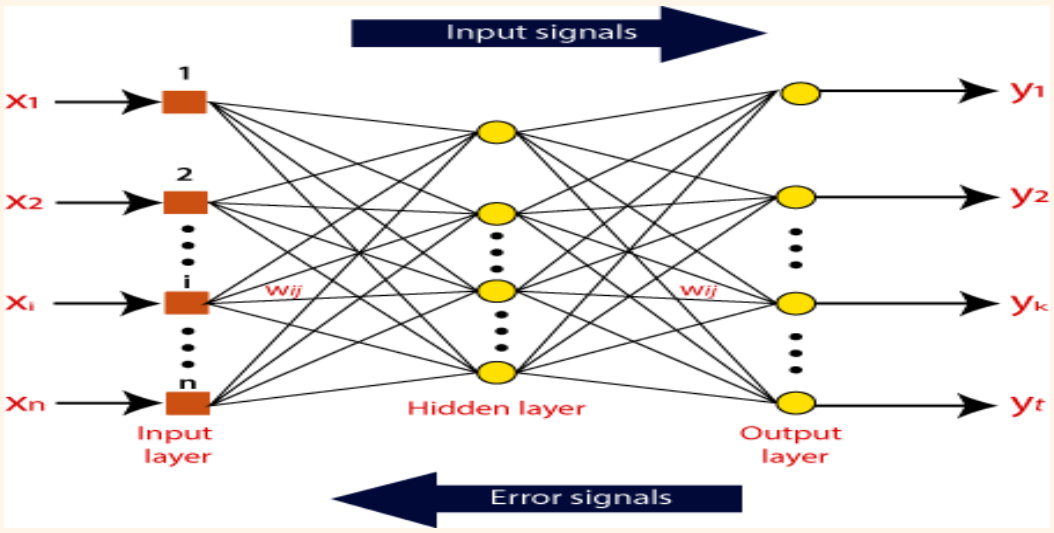
The duration of the network is unknown:

The network is reduced to a specific value of the error, and this value does not give us optimum results. “Science artificial neural networks that have stepped into the world in the mid-20th century are exponentially developing.

In the present time, we have investigated the pros of artificial neural networks and the issues encountered during their utilization. It should not be overlooked that the cons of ANN networks, which are a flourishing science branch, are eliminated individually, and their pros are increasing day by day. It means that artificial neural networks will turn into an irreplaceable part of our lives progressively important."

How do artificial neural network work?

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.



Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.

The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions. Let us look at each of them in details:

Binary:

In binary activation function, the output is either a one or a 0. Here, to accomplish this, there is a threshold value set up. If the net weighted input of neurons is more than 1, then the final output of the activation function is returned as one or else the output is returned as 0.

Sigmoidal Hyperbolic:

The Sigmoidal Hyperbola function is generally seen as an "S" shaped curve. Here the tan hyperbolic function is used to approximate output from the actual net input. The function is defined as:

F(x) = (1/1 + exp(-????x))

Where ???? is considered the Steepness parameter.

Types of Artificial Neural Network:

There are various types of Artificial Neural Networks (ANN) depending upon the human brain neuron and network functions, an artificial neural network similarly performs tasks. Many of the artificial neural networks will have some similarities with a more complex biological partner and are very effective at their expected tasks. For example, segmentation or classification.

Feedback ANN:

In this type of ANN, the output returns into the network to accomplish the best-evolved results internally. As per the University of Massachusetts, Lowell Centre for Atmospheric Research. The feedback networks feed information back into itself and are well suited to solve optimization issues. The Internal system error corrections utilize feedback ANNs.

Feed-Forward ANN:

A feed-forward network is a basic neural network comprising of an input layer, an output layer, and at least one layer of a neuron. Through assessment of its output by reviewing its input, the intensity of the network can be noticed based on group behavior of the associated neurons, and the output is decided. The primary advantage of this network is that it figures out how to evaluate and recognize input patterns.

Prerequisite:

No specific expertise is needed as a prerequisite before starting this tutorial.

Audience:

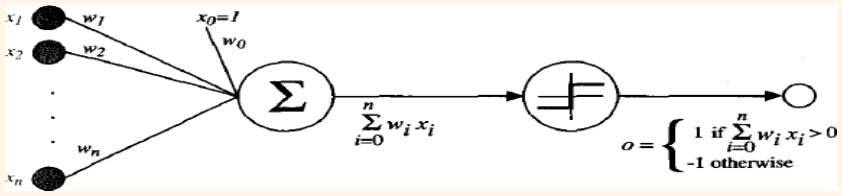
Our Artificial Neural Network Tutorial is developed for beginners as well as professionals, to help them understand the basic concept of ANNs.

PERCEPTRONS:

One type of ANN system is based on a unit called a perceptron, illustrated in below Figure: A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. More precisely, given inputs xl through xn the output o(xl, . . . , xn) computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

Ahmed Ali



where each w_i is a real-valued constant, or weight, that determines the contribution of input xi to the perceptron output. Notice the quantity $(-w_0)$ is a threshold that the weighted combination of inputs $w_1 x_1 + ... + w_n x_n$ must surpass for the perceptron to output a 1.

To simplify notation, we imagine an additional constant input $x_0 = 1$, allowing us to write the above inequality as $w x > 0$, or in vector form as.

For brevity, we will sometimes write the $i=0$ I I w $x>0$ perceptron function as.

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

where

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

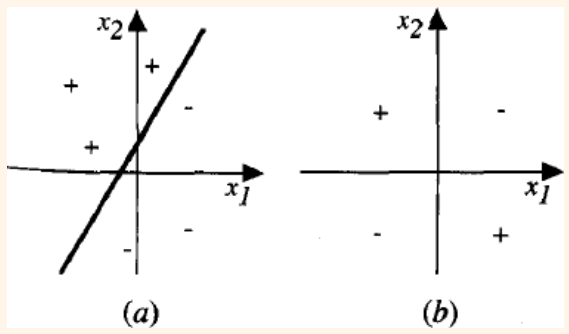
Learning a perceptron involves choosing values for the weights w0 , wN Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors.

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

Representational Power of Perceptron's:

We can view the perceptron as representing a hyperplane decision surface in the n--dimensional space of instances (i.e., points).

The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in Figure below the equation for this decision hyperplane is w. x = 0. Of course, some sets of positive and negative examples cannot be separated by any hyperplane. Those that can be separated are called linearly separable sets of examples.



Ahmed Ali

The decision surface represented by a two-input perceptron. *(a)* A set of training examples and the decision surface of a perceptron that classifies them correctly. *(b)* A set of training examples that is not linearly separable (i.e., that cannot be correctly classified by any straight line). x_1 and x_2 are the Perceptron inputs. Positive examples are indicated by "+", negative by "-". The inputs are fed to multiple units, and the outputs of these units are then input to a second, final stage. One way is to represent the Boolean function in disjunctive normal form (i.e., as the disjunction (OR) of a set of conjunctions (ANDs) of the inputs and their negations). Note that the input to an AND perceptron can be negated simply by changing the sign of the corresponding input weight. Because networks of threshold units can represent a rich variety of functions and because single units alone cannot, we will generally be interested in learning multilayer networks of threshold units.

The Perceptron Training Rule:

Although we are interested in learning networks of many interconnected units, let us begin by understanding how to learn the weights for a single perceptron. Here the precise learning problem is to determine a weight vector that causes the perceptron to produce the correct ·1 output for each of the given training examples.

Several algorithms are known to solve this learning problem. Here we consider two: the perceptron rule and the delta rule. These two algorithms are guaranteed to converge to somewhat different acceptable hypotheses, under somewhat different conditions. They are important to ANNs because they provide the basis for learning networks of many units.

One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the perceptron training rule, which revises the weight wi associated with input xi according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Here ***t*** is the target output for the current training example, ***o*** is the output generated by the perceptron, and is a positive constant called the learning rate. The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

Why should this update rule converge toward successful weight values? To get an intuitive feel, consider some specific cases. Suppose the training example is correctly classified already by the perceptron. In this case, (t - o)) is zero, making ^wi zero, so that no weights are updated. Suppose the perceptron outputs a-1, when the output is +1. To make the perceptron output a+1 instead of -1. in this case, the weights must be altered to increase the value of w,x.

For example, if xi>0, then

increasing wi will bring the perceptron closer to correctly classifying this example. Notice the training rule will increase w, in this case, because (t - o), and xi are all positive. For example, if xi = .8, ・ =1. t = 1, and o = - 1, then the weight update will be wi = (t - o)xi = O.1(1 - (-1))0.8 = 0.16. on the other hand, if t = -1 and o = 1, then weights associated with positive xi will be decreased rather than increased.

In fact, the above learning procedure can be proven to converge within a finite number of applications of the perceptron training rule to a weight vector that correctly classifies all training examples, provided the training examples are linearly separable and provided a sufficiently small ・ is used. If the data are

not linearly separable, convergence is not assured.

Gradient Descent and the Delta Rule

Although the perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable. A second training rule, called the delta rule, is designed to overcome this difficulty. If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept. The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples. This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected units. It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses.

The delta training rule is best understood by considering the task of training a thresholder

perceptron: that is, a linear unit for which the output o is given by.

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.

To derive a weight learning rule for linear units, let us begin by specifying a measure for the training error of a hypothesis (weight vector), relative to the training examples. Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where D is the set of training examples, td is the target output for training example d, and od is the output of the linear unit for training example d. By this definition, E (w) is simply half the squared.

Difference between the target output td and the hear unit output od, summed over all training examples. Here we characterize E as a function of (w) because the linear unit output o depends on this weight vector. Of course, E also depends on the set of training examples, but we assume these are fixed during training, so we do not bother to write E as an explicit function of these. There we show that under certain conditions the hypothesis that minimizes E is also the most probable hypothesis in H given the training data.

Up Next:

Multiple Layer protection.