regression ml
(created by chirag#forkyknight)

A regression problem is when the output variable is a real or continuous value, such as "salary" or "weight". Many different models can be used, the simplest model is the linear regression

Types of regression

- *Linear regression*
- *Logistic regression*
- *Polynomial regression*
- *Stepwise regression*
- *Stepwise regression*
- *Ridge regression*
- *Lasso regression*

In this unit we will deal only with linear regression..

Given a data set of *n* statistical units, a linear regression model assumes that the relationship between the dependent variable *y* and the set of *independent variable* **Xi** is linear.

Simple Linear Regression

IN Simple linear regression we only have a **single feature**.

Example

label=Y

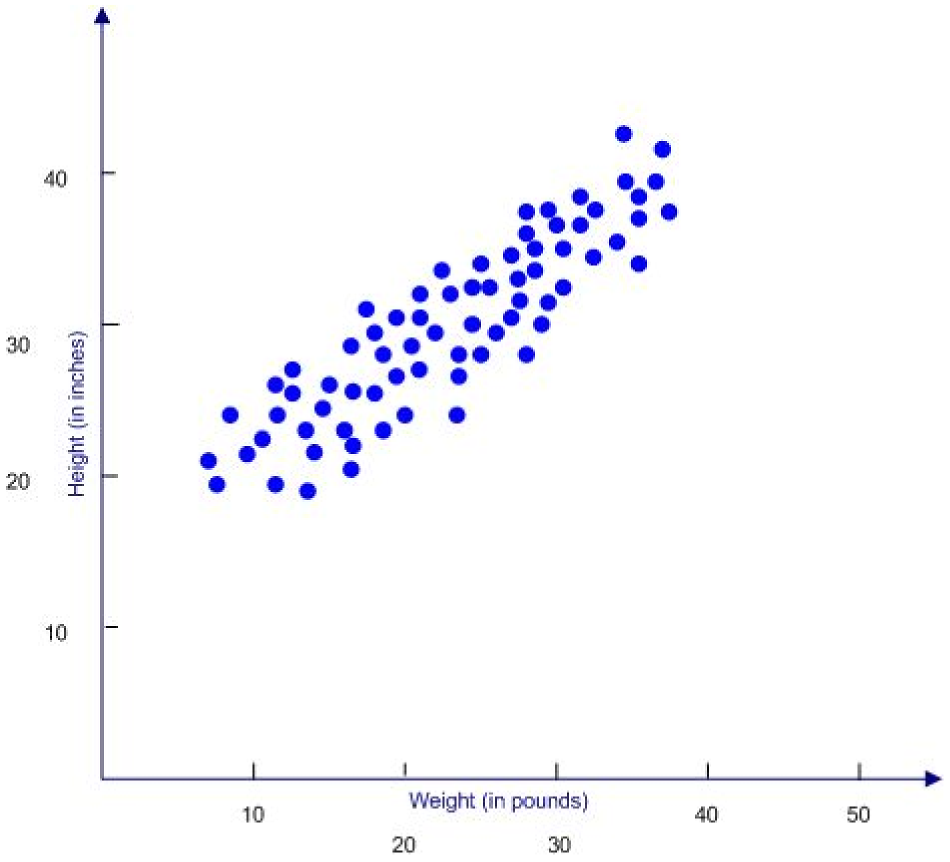
feature=X1

Whant equation we can get = Y=mX1+C

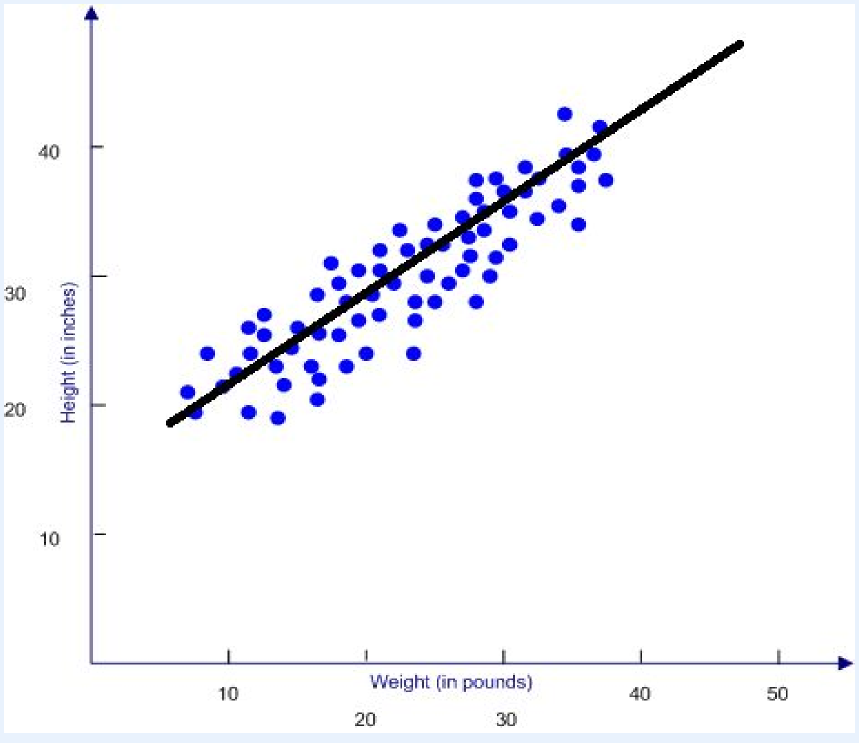
It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x)

Lets us examine this with simple example

Here we take the example of relation of height and weight
Let weight is our feature(i.e independent variable)
And height is label (tht depend on independent variable)



The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.
We will talk about error later…
So here is our graph with best fit line



It is True that the line doesn't pass through every dot, but the line does clearly show the relationship between weight and height. Using the equation for a line, you could write down this relationship as follows:

y=mx+b

Where:

Y is the height in inches-the value we're trying to predict.

m is the slope of the line.

x is the weight in pounds-the value of our input feature.

b is y intercept

By convention in machine learning, you'll write the equation for a model slightly differently:

y′=b+w1x1

where:

y′ is the predicted label (a desired output)

B is the bias (the y-intercept), sometimes referred to as w0 w1 is the weight of feature 1. Weight is the same concept as the "slope"

Now to infer(*pridict) the height(y’) for new weight , just substitute the value of weight in model line.

Although this model uses only one feature, a more sophisticated model might rely on multiple features, each having a separate weight (w1, w2, etc.). For example, a model that relies on three features might look as follows:

y′=b+w1x1+w2x2+w3x3.

Now we will look at training and losses\

Lets us first understand about dataset,

In particular, three datasets are commonly used in different stages of the creation of the model.

1. Training dataset
2. 2. Validation data set
3. Test dataset

Training dataset

The model is initially fit on a training dataset, which is a set of examples used to fit the parameters like weight and bias

The model is generally trained using optimization methods such as gradient descent or stochastic gradient descent. In practice, the training dataset often consists of pairs of an input vector (or scalar)or features and their corresponding output vector (or scalar)or labels.

Val dataset

Successively, the fitted model is used to predict the responses for the observations in a second dataset called the validation dataset.[3] The validation dataset provides an unbiased evaluation of a model fit on the training dataset

And tell us about accuracy and error of our model.

Test dataset

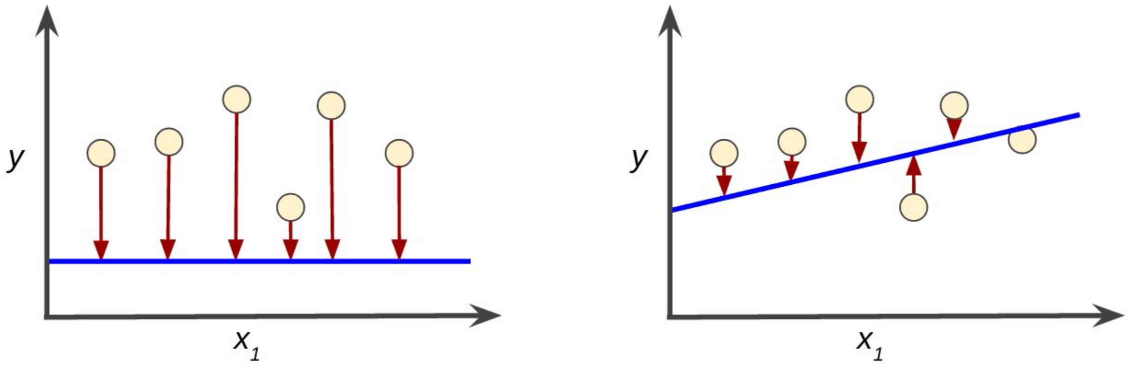
Finally, the test dataset is a dataset used to provide an unbiased evaluation of a *final* model fit on the training dataset. If the data in the test dataset has never been used in training, the test dataset is also called a holdout dataset.

Training

Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization.

Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have *low* loss, on average, across all examples. For example, Figure shows a high loss model on the left and a low loss model on the right. Note the following about the figure:

- The arrows represent loss.
- The blue lines represent predictions.



Clearly, the line in the right plot is a much better predictive model than the line in the left plot.

The linear regression models we'll examine here use a loss function called squared loss (also known as L_2 loss). The squared loss for a single example is as follows:

= the square of the difference between the label and the prediction

= (observation - prediction(x))^{2}

= (y - y')^{2}

Mean square error (MSE) is the average squared loss per example over the whole dataset. To calculate MSE, sum up all the squared losses for individual examples and then divide by the number of examples:

MSE=1/N(summation(y-pridicted(x)^2)

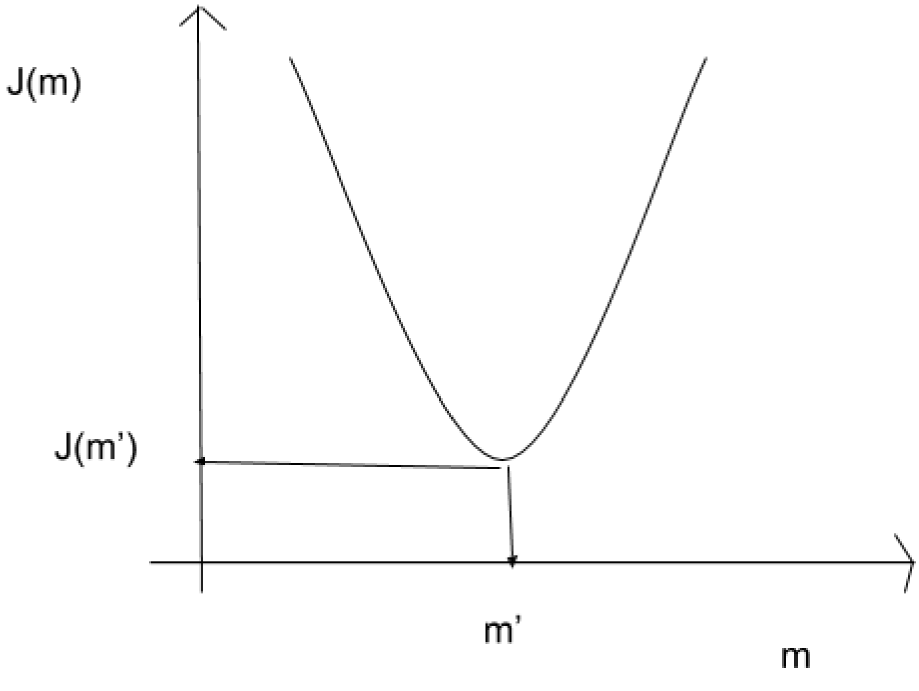
Now we will learn about gradient descent:-

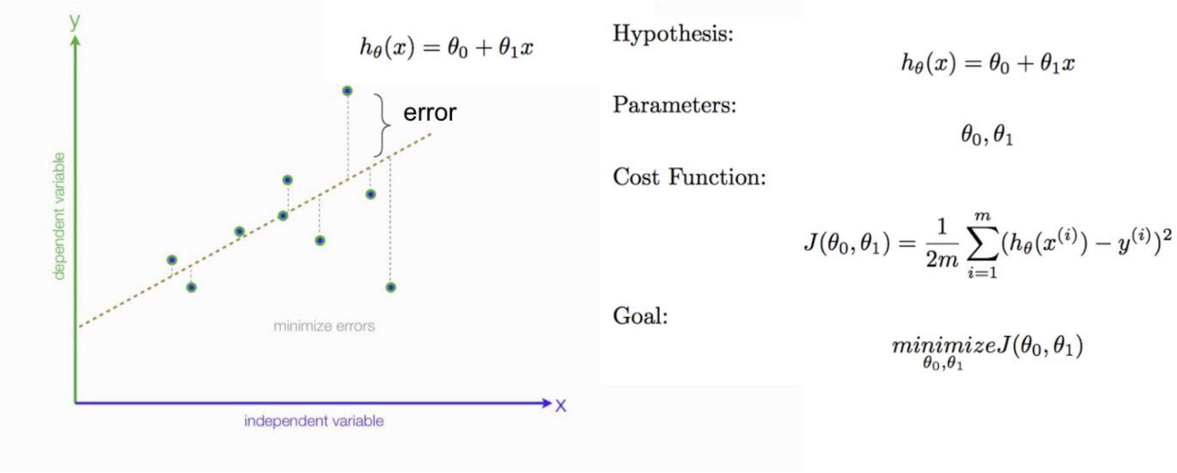
Gradient descent is simply used to find the values of a function's parameters (coefficients) that minimize a cost functionor loss as far as possible.

Previously we get to know about L2 loss or squared loss and MSE.

So we will plot a graph between J(m)(cost function or loss function) and m(slope).

And our task will be to minimize that graph b/w J(m) and m to get the value of parameter m which give lowest possible loss…





Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal:

$$\underset{\theta_0, \theta_1}{minimize} J(\theta_0, \theta_1)$$

How to minimize cost fun.

Calculating gradient descent

Gradient Descent runs iteratively to find the optimal values of the parameters corresponding to the minimum value of the given cost function, using calculus. Mathematically, the technique of the ‘*derivative*’ is extremely important to minimise the cost function because it helps get the minimum point. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad [1.0]$$

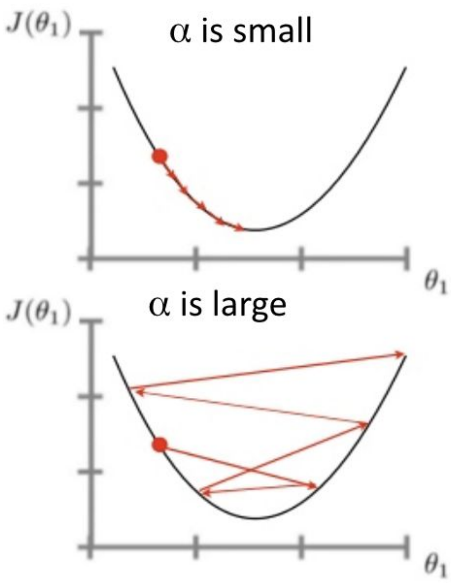
$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_0}\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right) \qquad [1.1]$$

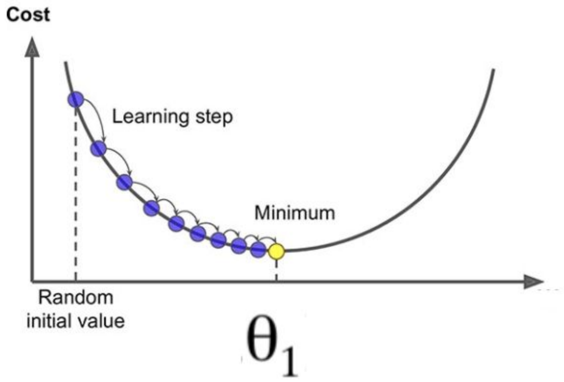
$$= \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad [1.2]$$

$$= \frac{1}{m}\sum_{i=1}^{m}2\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\frac{\partial}{\partial\theta_0}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \qquad [1.3]$$

$$= \frac{2}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \qquad [1.4]$$

```
repeat until convergence {
    $\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)$
        (for $j = 1$ and $j = 0$)
}
```





α used here is learning rate

The learning rate (also known as ‘step size’) is a positive hyper parameter that plays an important role in determining the amount by which a model adapts when the weights are updated. Hence, the value of the network’s learning rate needs to be carefully selected. If the value is too small, it will require more training epochs and the process will take more time whereas, if the value is too big, the network might converge(converge is the instant when there is no significant change in the slope for some iterations) very quickly and won’t be efficient.

This is where we introduce another term called optimal learning rate. Now this can be defined as the learning rate that can precisely move the weight to its minimum (most efficient) value in one step.

Types of gradient descent

1. Batch gradient descent
2. Stochastic gradient descent
3. Mini batch gradient descent

Batch gradient

Batch Gradient Descent is the most straightforward type. It calculates the error for each example within the training set. After it evaluates all training examples, it updates the model parameters. This process is often referred to as a *training epoch*. Advantages of batch gradient descent are that it’s computationally efficient and produces a stable error gradient and a stable convergence. One disadvantage is that the stable error gradient can sometimes result in a state of convergence that isn’t the best the model can achieve. It also requires that the entire training set resides in memory and is available to the algorithm.

Stochastic gradient

Stochastic Gradient Descent updates the parameters according to the gradient of the error with respect to a single training example. This is unlike Batch Gradient Descent, which updates the parameters after all training examples have been evaluated. This can make Stochastic Gradient Descent faster than Batch Gradient Descent depending on the problem. One advantage is that the frequent updates provide a detailed rate of improvement. A disadvantage is that the frequent updates are more computationally expensive than Batch Gradient Descent. The frequency of the updates also can result in noisy gradients, and may cause the error rate to fluctuate instead of slowly decrease

Mini batch

Mini Batch Gradient Descent is an often-preferred method since it uses a combination of Stochastic Gradient Descent and Batch Gradient Descent. It simply separates the training set into small batches and performs an update for each of these batches. It thus creates a balance between the efficiency of Batch Gradient Descent and the robustness of Stochastic Gradient Descent. Common numbers of examples per batch range between 30 and 500. But like for any other machine learning technique, there is no well-defined rule because the optimal number can vary for different problems. Mini Batch Gradient Descent is commonly used for deep learning problems.

Practical example=https://github.com/Forkyknight/linear-regression Check it out……