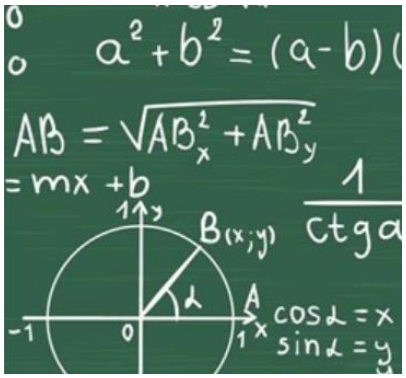


Under the Hood: Normal Equation, Batch GD, Mini-Batch and Stochastic GD for Linear Regression

MACHINE LEARNING – SERIES

This document explains the mathematics behind one the simplest machine learning models, linear regression. We will explore direct "closed-form" equation and a iterative optimization approach, Gradient Descent.

Viraj Vaitha

Contents

Introduction

As seen earlier, a lot can be accomplished without understand the mathematics used to create the algorithms. Our approach has been to treat the entire process as a blackbox.

By understanding the mathematics, we are more likely to select the appropriate model, correct training algorithm, and have good hyperparameters. Understanding what happens under the hood could enable you to solve errors quickly, debug issues and help building and training neural networks.

Two approaches discussed in this report are outlined below:

The direct "close-form" equation to directly compute model parameters that best fit the mode through minimizing the cost function.

Using iterative optimization approach, Gradient Descent (GD). This achieves the same values as that discussed above however the method of reaching the values is iterative.

Linear Regression – Cost Function

Linear regression can be expressed as a vector form (useful for processing large volumes of data and performing complex mathematics across data structures). Then we must obtain the expression that links RMSE to $\boldsymbol{\theta}$. Finally, we must rearrange the equation so that we can find the value of $\boldsymbol{\theta}$ that minimizes the cost function.

If we have several features, the linear regression is as seen below:

$$\hat{y} = \theta_0 + \theta x_1 + \theta_2 x_2 + \theta_3 x_3 \dots \theta_n x_3$$

$$\hat{y} = h_\theta(x) = \theta^T . x \qquad (Vector\ Form)$$

Essentially several linear regressions are added together.

- $\hat{\mathbf{y}}$ is the precited values
- θ_j is the jth model parameter (This includes the bias term $\emptyset_0$ and weightings $\emptyset_{1,2,3..n}$
- x_i is the ith feature
- $m =$ number of training examples

In the vector equation:

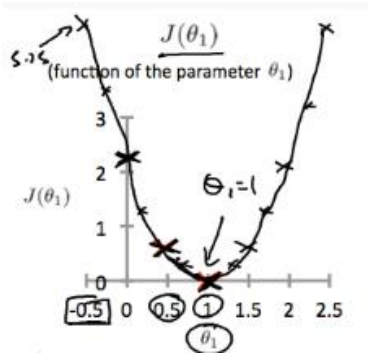
- θ is the model parameters vector, containing the bias term θ_0 and the feature weights θ_0 to θ_n
- θ^T is the transpose of θ converting the column vector into a row vector.
- $\theta^T . x$ is the dot product of θ^T and θ^T
- h_θ is the hypothesis function, using the model parameters θ

We saw that the most common performance metric to analyse regression is accuracy. We want to create a method that finds values of θ that minimize the RMSE. In practice we minimize for MSE, this is because the value that minimises a function also minimizes its square root leading to the same result. Below we obtain an expression derived by subtracting the predicted value formula by actual value formula.

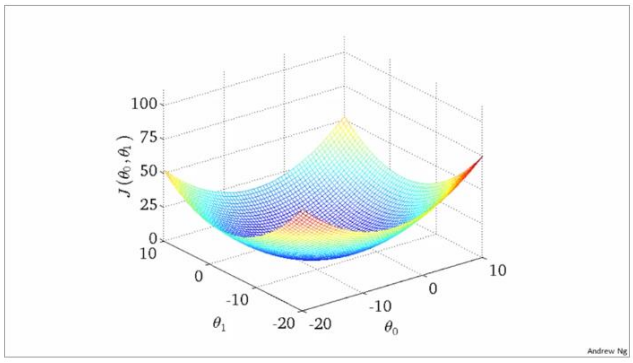
MSE Cost Function for Linear Regression

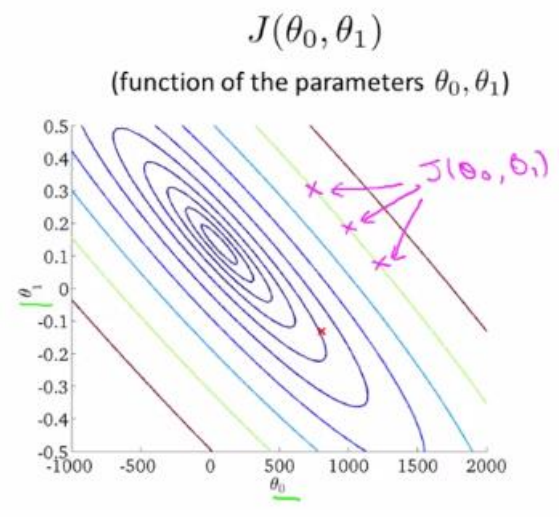
$$MSE(X, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T . x^i - y^i\right)^2$$

The equation states we will take the sum of all differences between the prediction from the actual value (error). The equation is solved to find the values of θ that minimizes the equation.



We can see that when θ =1, the cost (error) is at is lowest. This is key objective of the normal equation. This is simple with one feature. However, if we had several features we must visualise this in a higher dimensional space as seen below.





Each point along the same contour (as seen highlighted above) corresponds to the same cost (error). The centre of the smallest circle is where we approaches our minimum.

Normal Equation

This is the equation used to find the value of θ that minimises the cost function. Closed form solution is a mathematical equation that gives the result directly. This is obtained through manipulation of the previous cost function.

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

- $\theta^{\wedge}$ = the value of θ that minimizes the cost function
- y is the vector of target values containing y^1 to y^n

X^T is a matrix that contains all the features for different training examples.

Suppose you have the training in the table below:

age (x_1)	height in cm (x_2)	weight in kg (y)
4	89	16
9	124	28
5	103	20

You would like to predict a child's weight as a function of his age and height with the model

$\text{weight} = \theta_0 + \theta_1 \text{age} + \theta_2 \text{height}.$

What are X and y?

- $X = \begin{bmatrix} 4 & 89 \\ 9 & 124 \\ 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$
- $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 1 & 16 \\ 1 & 28 \\ 1 & 20 \end{bmatrix}$
- $X = \begin{bmatrix} 4 & 89 & 1 \\ 9 & 124 & 1 \\ 5 & 103 & 1 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$
- $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$

Correct

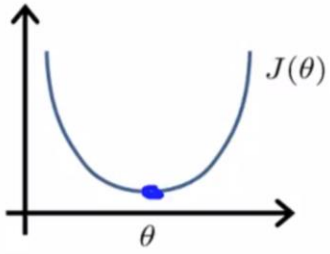
Effectively, the normal equation has taken the derivative of the cost function with respect to θ to equal 0 to solve for each parameter θ.

Intuition: If 1D ($\theta \in \mathbb{R}$)

$\rightarrow J(\theta) = a\theta^2 + b\theta + c$

$\frac{d}{d\theta} J(\theta) = \dots \overset{\text{set}}{=} 0$

Solve for θ



$\theta \in \mathbb{R}^{n+1}$ $\quad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \dots \overset{\text{set}}{=} 0$ (for every j)

Solve for $\theta_0, \theta_1, \dots, \theta_n$

In Scikit Learn, we can use the normal equation by following below:

```
Import Relevant Modules

From sklearn.linear_model import LinearRegression

Initiate Model and Fit

lin_reg= LiniearRegression()

lin_reg.fit(X,y)

lin_reg.intercept_, lin_reg.coef #Code to receive coefficients

Predict Values

lin_reg.predict(X_new)
```

$$X = \begin{matrix} 1 & 34 & 32 \\ 1 & 53 & 75 \\ 1 & 66 & 67 \end{matrix} \qquad y = \begin{matrix} 44 \\ 34 \\ 42 \end{matrix}$$

The first column of the matrix of ones is necessary as during the multiplications the ones are required for the constants to be applied correctly. The remaining columns are the values for your different features. y is your actual values.

Feasibility of Normal Equation

The computational complexity of using the Normal Equation is **O(n^2.4)** to **O(n^3)** this means that if you **double** the number of **features,** you multiply the time taken by 2^2.4= **5.3** to 2^3 = **8**

The positive is that the equation is linear with respect to the size of the training set. Normal equation can be **used with a high number of training examples** but is limited by a **high number of features**.

Gradient Descent

This is a common **optimisation algorithm** capable of finding the **optimal solution to problems.** The nature of algorithm is to **iteratively** change parameters to minimize the cost (error) function.

To personify this algorithm for an example, imagine you are blindfold. You are aware that you need to get to the bottom, but you can't see. You decide that you will always go downhill in the direction of the steepest slope. The direction and size of the steps are the two most important parameters.

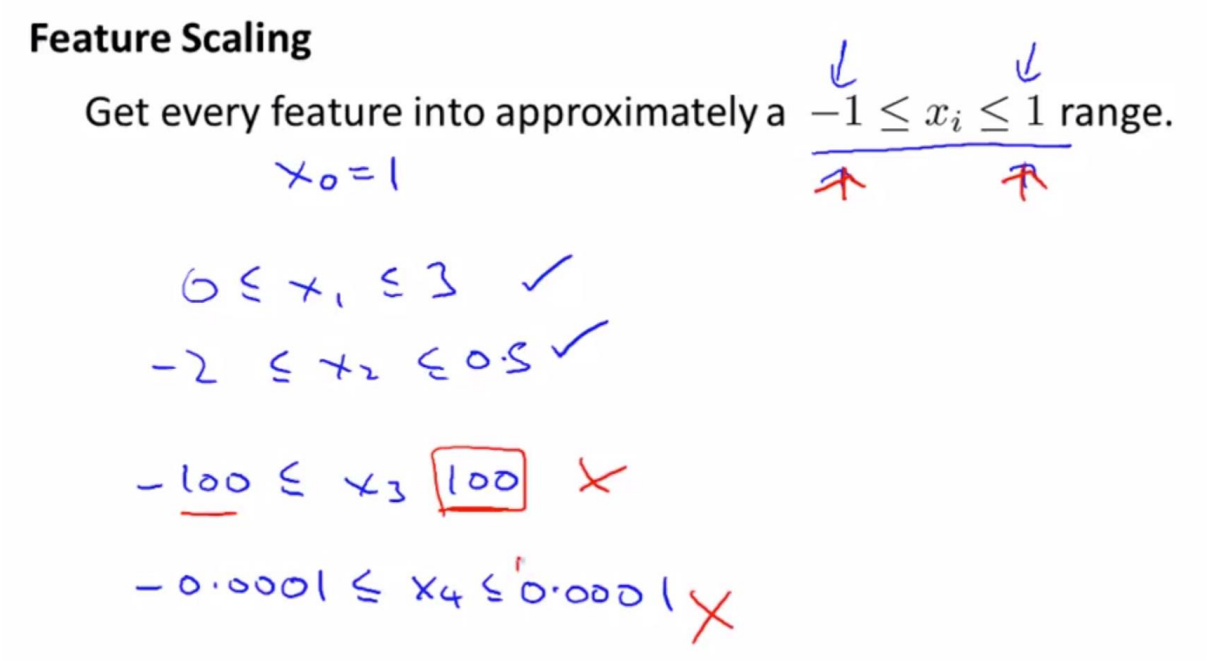
Learning Rate

Mathematically, we initiate the process by using random values for the parameters $\boldsymbol{\theta}$**.** Then they are gradually improved by taking one step at a time. The size of the step is called the **learning rate**

A low learning rate means that the model may take a long time for find the optimum solution. Whereas if the learning rate is too high, we may end up on the other side of the parabola, or higher than we initial started. The algorithm effectively diverges instead of converging.

Scaling

Gradient Descent can converge more effectively the features have been scaled appropriately. Each feature should have similar scales. The Standard Scaler is a good package that can be used to scale the features. The more parameters the higher the dimensional space, leading to the search to be more difficult with a higher number of features. Alternatively, Mean normalisation can also be used to scale data where the mean of each feature is approximately zero. In practice you could try both and see the impact on the gradient descent time and accuracy.

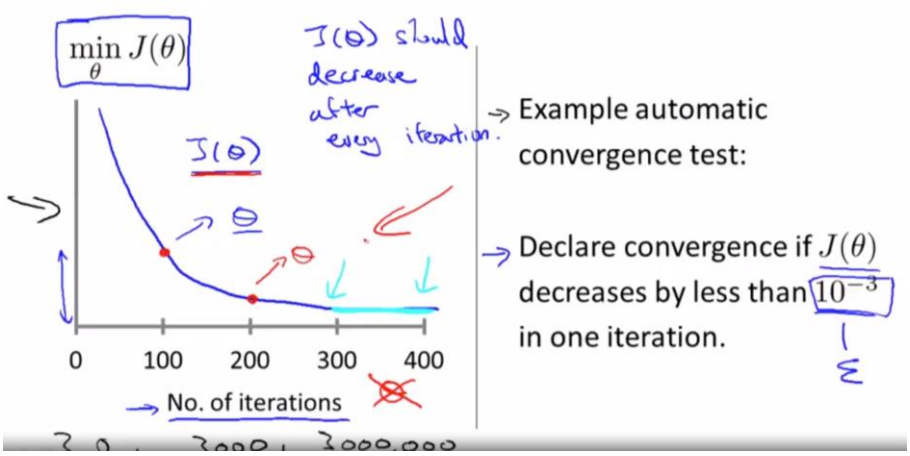


Main Challenges with Gradient Descent

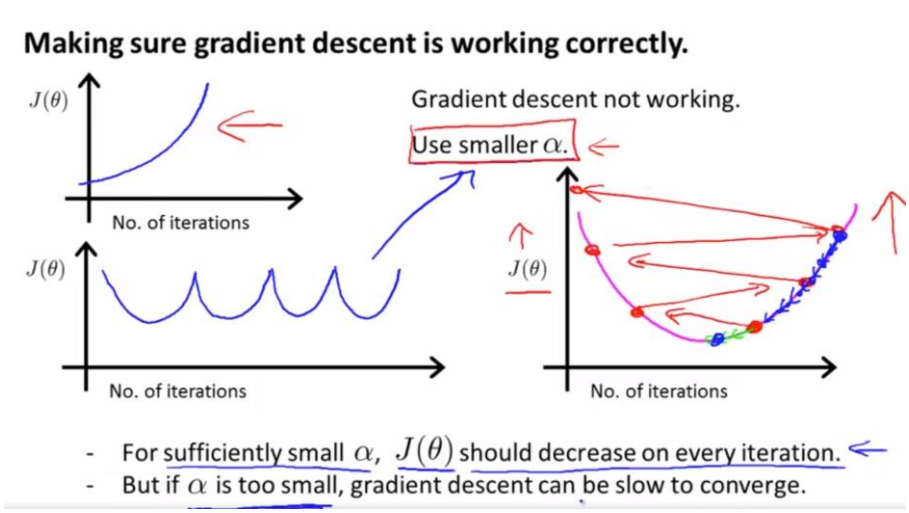
- cost function may not map out to a "convex function", a regular bowl, it may have ridges, holes, plateaus and other irregularities that effect the convergence.
- It may approach a local minimum and get stuck
- it can take to long to cross a plateau.

How to check Gradient Descent is Working Correctly

Plot No. of Iterations against the cost function.



If it is not working correctly you may see something similar to the below.



Batch Gradient Decent

To implement Gradient Descent, we must understand how much the cost function changes with respect to each model parameter. This is the same as saying, what is the slope of the floor under my feet when I face east, or north or south. If we had more dimensions, then we must image there are more directions to face (higher dimensional space) that we must compute the gradient for. In mathematics this is called the partial derivative that is used when an equation has a higher dimension of parameters.

A mathematical technique called the Gradient Vector can be used to compute all of these in one go instead of one by one.

$$\nabla_\theta MSE(\theta) = \begin{matrix} \partial/\partial\theta_o MSE(\theta) \\ \partial/\partial\theta_o MSE(\theta) \\ \partial/\partial\theta_o MSE(\theta) \end{matrix} = \frac{2}{m} X^T.(X.\theta - y)$$

Selecting Learning Rate

Start with 0.001, 0.01, 0.1, 1 ….

Keep increasing with factor of 10's.

Algorithm

<u>In Words</u>

We have the cost equation which we are trying to minimise. We select a random value for all θ.

Now we determine the gradient vector(Gradient with respect to each parameter) for each parameter. The value indicates how much the cost equation (error) changes. If this is positive we must move in the opposite direction as seen by the equation below. The learning rate will determine the size of the step (how much to change theta) at each iteration.

To implement the algorithm please follow below:

```
Eta = 0.1 #learning rate

N_iterations = 100

M = 100 #Training Set Size (no. of rows)

Theta = np.random.randn(2,1) #random initialisation

For iteration in range(n_iterations):

        Gradients = 2/m * X_b.T.dot(X_b.dot(theta)-y)

        Theta = theta – eta*gradients
```

Stochastic Gradient Descent

The main problem with Batch Gradient Descent is that is uses the whole training set to compute the gradient decent at each step. Whereas Stochastic pics a random instance in the training set at each and every step. This may lead the cost function to fluctuate however, the algorithm operates a lot faster since it has less data to manipulate at every iteration. The solution found will be good, but not optimal.

The randomness is good to escape from local optima but bad because it means that the algorithm can never settle at the minimum. One solution to this is to gradually decrease the learning rate. This is called simulated annealing as it resembles the process of annealing in metallurgy. The function that determines this is the learning schedule. If the rate is reduced to quickly you may get stuck in the local minimum or frozen halfway to the minimum. If its reduced to slowly you may jump around the minimum for a long time and end up with a suboptimal solution.

This technique iterates by rounds m iterations' and each round is an epoch. Imagine we are taking a slice of data to run the algorithm, then a new slice, then a new slice each time. We hope to find the optimal through this random nature. We can shuffle the data at each step however this tends to slow down the process. We can implement SGD as seen in Machine learning Series 1. Not feeling confident here

Mini Batch Gradient Descent

The entire training set is broken into equally sized mini batches for the whole training set. This method leads to a performance boost when using GPUs. We can compute say 1000 batches of 5000 training examples in one go as opposed to 5000000 in one go.

Summary

Algorithm	Large m	Out-of-core support	Large n	Hyperparams	Scaling Required	Scikit-Learn?
Normal Equation	Fast	No	Slow	0	No	Linear Regression
Stochastic GD	Fast	Yes	Fast	>1	Yes	N/A
Mini-Batch GD	Fast	Yes	Fast	>1	Yes	SGDRegressor
Batch GD	Slow	No	Fast	2	Yes	SGDRegressor

Fortunately, for linear regression, Gradient Descent is guaranteed to approach arbitrarily close to the global minimum (if you wait long enough and the learning rate is not too high). Linear Regression is a convex function making Gradient Decent work extremely well in this scenario.