Machine Learning Documentation

Enes Agirman

July 2023

1 Introduction

Note: The code used to implement the methods mentioned in this document is in this GitHub repository:

`https://github.com/EnesAgirman/ML_Documentation_Code`

Note: This notes follows the artificial intelligence course taught in Stanford University, CS229 as its syllabus.[1] [2]

The book "A Dictionary of Computing." defines machine learning as: "A branch of artificial intelligence concerned with the construction of programs that learn from experience".[3]

2 LINEAR REGRESSION

Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable. [4]

We use linear regression to find a linear relationship between the inputs and the output. This linear relation can be modeled using the following equation where $h(x)$ is our prediction(also called hypothesis) of the output y given the inputs(x) and parameters(θ)(also called weights):

$$h(x) = \sum_{i=0}^{n} \theta_i x_i$$

We define this linear relation with the input matrix X and weight matrix θ given as:

$$\text{X}= \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}, \quad \theta= \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ . \\ . \\ . \\ \theta_n \end{bmatrix},$$

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T X$$

where x_0 is taken as 0. This is done to be able to shift the function $h(x)$ with adjusting θ_0. x_0 is also called the intercept term.

$h(x)$ now gives us a prediction of y given the inputs X and we can adjust the behavior of this function with adjusting its parameters θ. We want to adjust the parameters so that we are as accurate as possible with our predictions. And to be able to measure how close our prediction is to the actual output y, we use something called a loss function(also called the cost function) denoted by J(θ). We can define different cost functions for different scenarios and for our use case in linear regression, we define the cost function as:

$$J(\theta) = \frac{1}{2} \sum_{i=0}^{n} \left(h(x^{(i)}) - y^{(i)}\right)^2$$

Where we just took the square of the difference between the prediction and the actual output and thus made the cost function always non-negative. We multiply the sum by the constant $\frac{1}{2}$ to make the calculations easier later.(The $\frac{1}{2}$ cancels out later.)

2.1 LEAST MEAN SQUARES(LMS) ALGORITHM

LMS algorithm is an algorithm that uses input output pair(x, y) to adjust θ's in the cost function to get a better hypothesis of the output y using the inputs.

First, we give random values to the parameters. We can give the all 0s, all 1s or give a random number to each of them separately.

After we give our initial values to the parameters, we use the following algorithm to adjust the θs.

$$\theta_i := \theta - \alpha \nabla J(\theta)$$

In here, the sign ":=" denotes the operator. It means that the value on the left is now changed to the number on the right. It is equivalent to the "=" sign in programming. $\nabla J(\theta)$ is the gradient of the cost function.

Note: The gradient of a function gives a vector that has the direction of the steepest increase in any n-dimensional space and has the magnitude of the change in that direction.

$\nabla J(\theta)$ gives us the direction of the steepest increase of the cost function

meaning it is the direction which the direction which our accuracy in our hypothesis decreases the most. We want to increase our accuracy so we move on the opposite direction given by $-\nabla J(\theta)$ which gives us the steepest decrease in the magnitude of the cost function. This gives us the direction in which our prediction's accuracy increase the most.

Our goal in moving in the steepest decrease in the cost function and minimizing our cost is that we want to get to the point where the cost function is the least, in other words the absolute minima. In our discussions about minimizing the cost function below with gradient descent, we always move in the direction of the decrease of the cost function. In order for us to reach the absolute minima, we shouldn't reach any local minima. If we reach one, we cannot leave it because to leave the local minima, we should move in the direction of increase. That is why in our cases, we assume that there are no local minima.

The α is the learning rate. It is the magnitude of how much we should go in the gradient's direction.

We wrote the LMS algorithm in the vector form but let's now write the algorithm for one θ and for 1 training example :

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\frac{\partial}{\partial \theta_j} J(\theta)$ can be manipulated as:

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

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2$$

$$= 2.\frac{1}{2}(h_\theta(x) - y).\frac{\partial}{\partial \theta_j}(h_\theta(x) - y)$$

$$= (h_\theta(x) - y).\frac{\partial}{\partial \theta_j}(\sum_{i=0}^{n} \theta_i x_i - y)$$

Where in the summation, only the j^{th} term is nonzero thus giving us

$$(h_\theta(x) - y).\frac{\partial}{\partial \theta_j}(\sum_{i=0}^{n} \theta_i x_i - y)$$

$$= (h_\theta(x) - y).x_j$$

Thus for a single θ and single x,

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

For n training examples(n of (x, y) pairs), the algorithm is:

Repeat until convergence

$$\theta_j := \theta_j + \alpha(\sum_{i=0}^{n}(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)})$$

And we do this for every j to get the θ vector. In the vector form, the LMS algorithm is:

$$\theta := \theta + \alpha(\sum_{i=0}^{n}(y^{(i)} - h_\theta(x^{(i)}))x^{(i)})$$

This method looks at every example in the entire training set on every step, and is called **batch gradient descent**. See the code in Linear Regression Batch Gradient Descent in the GitHub repository for the implementation of this method on an example.

Batch gradient descent works very well on mapping linear relations from the inputs to accurate output predictions but it is computationally expensive for large data sets with a lot of inputs because we have to go through every input in the training set before updating the parameters one time.

Stochastic Gradient Descent given below is another algorithm to update the parameters:

For i in range(0, n):

$$\theta := \theta + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)})$$

In this algorithm, we move faster towards the absolute minima and it is computationally less expensive. Also we don't have to scan through every input in the input set to update the parameters. Instead we update our parameters for one example i, then update for another example i+1 and go through the examples while updating the parameters along the way.

Using Stochastic Gradient Descent, we actually don't go in the direction of the gradient but we are going in the way that decreases the error for the i^{th} example each time we update the parameters. This usually moves us towards the absolute minima but since we are updating the parameters while going through the parameters, when we get close to the absolute minima, we don't converge to the minima but constantly change the parameters without converging near the absolute minima. For most use cases, getting near the absolute minima is good enough and since stochastic gradient descent moves faster towards the absolute minima, we prefer stochastic gradient descent over batch gradient descent in most cases.

See the code in Linear Regression Stochastic Gradient Descent in the GitHub repository for the implementation of Stochastic Linear Regression on an example.

2.2 THE NORMAL EQUATIONS

The gradient descent algorithm gives us a way to minimize the cost function J(θ) with updating it in a way that decreases J(θ). There is another method

that also can be used to minimize J(θ). We are going to do that by taking the derivative of J(θ) by θ and setting it to zero. Since at the absolute minima of J(θ), the change in θ and thus the derivative should be 0. Using this property, we can take the derivative of J(θ) with respect to(w.r.t.) θ and set it to zero. The θ that ensures that is the θ that minimizes J(θ) .

Before we get into our algorithm, let's look into some linear algebra notation that is used in here and throughout the rest of this documentation. Let A be an m by n matrix where $A_{i,j}$ is the (i, j) entry A as:

$$A = \begin{bmatrix} A_{1,1} & ... & A_{1,n} \\ A_{2,1} & ... & A_{2,n} \\ . & & . \\ . & & . \\ . & & . \\ A_{m,1} & ... & A_{m,n} \end{bmatrix}$$

Let $f : \mathbb{R}^{mxn} \to \mathbb{R}$ be a function that maps matrices to real numbers. We define the gradient of f with respect to matrix A as:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{1,1}} & \cdots & \frac{\partial f}{\partial A_{1,n}} \\ \frac{\partial f}{\partial A_{2,1}} & \cdots & \frac{\partial f}{\partial A_{2,n}} \\ . & & . \\ . & & . \\ . & & . \\ \frac{\partial f}{\partial A_{m,1}} & \cdots & \frac{\partial f}{\partial A_{m,n}} \end{bmatrix}$$

So for $f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}$ and $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 5A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

Now let's try to find the θ that minimizes the loss function J and maximizes our accuracy in our prediction. Let's define the **design matrix** X that to be:

$$X = \begin{bmatrix} x_0^T \\ x_1^T \\ x_2^T \\ . \\ . \\ . \\ x_n^T \end{bmatrix}$$

Where X includes the input values of the training set as its rows and x_0 as the intercept term. Also let Y be the vector containing the target values of the training set as:

$Y = \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ y^{(2)} \\ . \\ . \\ . \\ y^{(3)} \end{bmatrix}$, We can see from these definitions that $h(\theta) = X\theta$. Using $h(\theta)$, we can define $J(\theta)$ as:

$$J(\theta) = \frac{1}{2}(X\theta - Y)^T(X\theta - Y) = \sum_{i=0}^{n}(y^{(i)} - h_\theta(x^{(i)}))^2$$

Where $h_\theta(x^{(i)})$ is the i^{th} element of $h(\theta)$ and $y^{(i)}$ is the i^{th} element of Y.

Now to find the θ that minimize $J(\theta)$, we find the derivative, gradient, of $J(\theta)$ with respect to θ:

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - Y)^T(X\theta - Y)$$

$$= \frac{1}{2}\nabla_\theta((X\theta)^T(X\theta) - (X\theta)^T Y - Y^T(X\theta) - Y^T Y)$$

$$= \frac{1}{2}\nabla_\theta((X\theta)^T(X\theta) - Y^T(X\theta) - Y^T(X\theta) - Y^T Y)$$

$$= \frac{1}{2}\nabla_\theta((X\theta)^T(X\theta) - 2Y^T(X\theta) - Y^T Y)$$

$$= \frac{1}{2}\nabla_\theta(\theta^T X^T X\theta - 2Y^T X\theta - Y^T Y)$$

for a symmetric matrix A, we can say that $\nabla_x b^T x = b$ and $\nabla_x x^T Ax = 2Ax$. Using these facts, we can write:

$$\nabla_\theta J(\theta) = \frac{1}{2}(2(X^T X)\theta - 2X^T Y)$$

$$\nabla_\theta J(\theta) = X^T X\theta - X^T Y$$

When θ minimizes $J(\theta)$, the gradient, derivative, of $J(\theta)$ w.r.t. θ should be 0 so we can equate $\nabla_\theta J(\theta)$ to 0 to find the θ that minimizes $J(\theta)$.

$$\nabla_\theta J(\theta) = X^T X\theta - X^T Y = 0$$

And from here, we find the normal equations as:

$$X^T X\theta = X^T Y$$

Thus the value of θ that minimizes $J(\theta$ is given as:

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

Note: Here, we assume that $(X^T X)$ is an invertible. There might be situations where it is not an invertible but there are some ways to make it an invertible but for simplicity, we assume that it is an invertible.

2.3 THE PROBABILISTIC INTERPRETATION

Throughout this chapter, we used the least-squares function as our cost function $J(\theta)$ to measure how far off we are with our predictions from the actual values. Bit why do we use the least-squares function? We can see clearly that it's magnitude is proportional to the difference between the hypothesis and the target variable but why didn't we use another function like:

$$J(\theta) = \sum_{i=0}^{n} \left(h(x^{(i)}) - y^{(i)}\right)^5$$

In this section, we will try to answer that question. Let us assume that the target variables $y^{(i)}$ and inputs $x^{(i)}$ are related with the following equation:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

Where θ is our parameter vector and $\epsilon^{(i)}$ is the error term in our prediction of the target variable and the actual target variable. $\epsilon^{(i)}$ captures the unmodeled effects that we didn't take into consideration making our prediction and random noise. We can model $\epsilon^{(i)}$ with assuming it is IID (independently and identically distributed). With this assumption, $\epsilon^{(i)}$ can be modeled according to a Gaussian distribution (also called a Normal distribution) with mean zero and some variance σ.[2] We can model the distribution of $\epsilon^{(i)}$ is modeled as:

$$p(\epsilon^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(\epsilon^{(i)})^2}{2\sigma^2})$$

This implies that:

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

Where $p(y^{(i)}|x^{(i)};\theta)$ is indicate the distribution of $y^{(i)}$ given $x^{(i)}$ parameterized by θ. [2]

Using matrix notation, we can express the same distribution over a set of variables as $p(Y|X;\theta)$. We want to view this distribution as a function of of θ so we define the Likelihood function as:

$$L(\theta) = L(\theta; X, Y) = p(Y|X;\theta)$$

The likelihood function $L(\theta)$ expresses the distribution of $y^{(i)}$ given $x^{(i)}$ parameterized by θ for each i in the training set.

Since $\epsilon^{(i)}$ for each i id assumed to be IID (independently and identically distributed), we can write:

$$L(\theta) = \prod_{i=a}^{n} p(y^{(i)}|x^{(i)};\theta)$$

$$= \prod_{i=a}^{n} \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

Given this Likelihood Function relating $y^{(i)}$ and $x^{(i)}$, how can we choose θ to maximize the Likelihood $L(\theta)$. Note that the likelihood function expresses the probability of our prediction defined by $\theta^T x^{(i)}$ be equal to $y^{(i)}$. In other words, it is the probability of our prediction being true. Thus if we find the θ that maximizes $J(\theta)$, we would find the theta that gives us the most accurate results in our prediction of the target variable.

Since the θ that maximize $L(\theta)$ also maximizes the log of $L(\theta)$ and it makes it easier to work with, we will try to maximize $log(L(\theta))$ given as:

$$l(\theta) = log(L(\theta))$$

$$= log(\prod_{i=a}^{n} \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}))$$

$$= \sum_{i=a}^{n} log \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

$$= nlog(\frac{1}{\sigma\sqrt{2\pi}}) - \frac{1}{\sigma^2}\frac{1}{2}\sum_{i=a}^{n}(y^{(i)} - \theta^T x^{(i)})^2$$

Thus maximizing $l(\theta)$ is the same as minimizing:

$$\frac{1}{2}\sum_{i=a}^{n}(y^{(i)} - \theta^T x^{(i)})^2$$

Which is the least-squares function we are familiar with. To answer the question of why are we using the least-squares function as our cost function is that when we assume that the error in our predictions is IID and can be modeled with Gaussian Distribution, minimizing the least-squares function also minimizes the log-Likelihood function $l(\theta)$ and the likelihood function $L(\theta)$ and thus minimize the error in our prediction, making our prediction more accurate.

Notice that the choice of the variance σ doesn't effect our cost function and thus it also doesn't effect our choice of θ. This fact will be useful in the following chapters.

2.4 LOCALLY WEIGHTED LINEAR REGRESSION

Linear regression put a linear relation in the input space. It is a line in 2D, plane in 3D etc. But sometimes our inputs are related with a nonlinear relation. In this case, we cannot use the linear regression we have learnt until now to find a relation between the inputs because the model would be too under-complex to

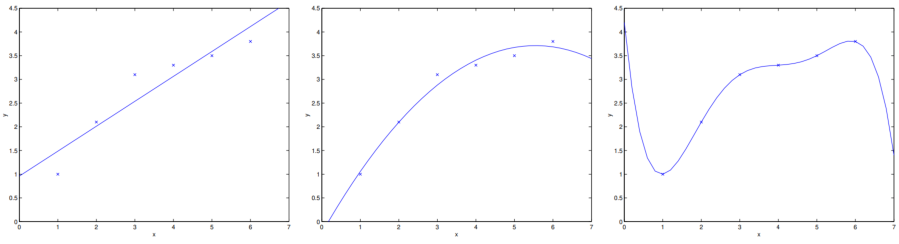

Figure 1: Underfitting and Overfitting[2]

accurately model the data so the model underfits the data. This can be seen in the images above.

Note: When a model cannot capture the underlying trend in the data space because of it being too under-complex, the model is said to be **Underfitting** the data. An example of Underfitting can be seen in the first plot in the figure above. When a model cannot capture the underlying trend in the data space because of it being too over-complex that it catches the noise and inaccurate data, the model is said to be **Overfitting** the data. An example of Overfitting can be seen in the third plot in the figure above. When the model can accurately capture the underlying trend in the data without Underfitting nor Overfitting, the model is said to be a **Good Fit** to the data.

As mentioned above, a linear model like $y = \theta_0 + \theta_1 x$ might underfit the data. Actually, the first plot in the figure above is the fitting of this exact model to a dataset. If we instead add an extra feature x^2 to our model as:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

Now, our model can fit the data better than before without underfitting. This can be seen in the second plot in the figure above. Since adding a feature made our model more accurate, it can be assumed that adding more features always increase a models accuracy but this would be incorrect because adding more features to a model can cause our model to overfit the data. An example of this could be seen in the third plot in the figure above where we use the following model to fit the same data as the other models.

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

Instead of adding features to our model to avoid underfitting, we could've also used the **Locally Weighted Linear Regression**(LWLR) algorithm. LWLR, as the name suggests, puts weights on the features when calculating the hypothesis so instead of fitting θ to minimize:

$$\sum_{i=a}^{n} (y^{(i)} - \theta^T x^{(i)})^2$$

and outputting $\theta^T X$, The LWLR fits θ to minimize

$$\sum_{i=a}^{n} w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$$

And then outputs $\theta^T X$. Here, $w^{(i)}$ is the weight for i^{th} training example. Intuitively, we can see that the greater the weight w^i is, the greater the effect of the $(y^{(i)} - \theta^T x^{(i)})^2$ and thus $x^{(i)}$ on the update of θ. A commonly used choice for the weight is:

$$w^i = exp(-\frac{(x^{(i)} - x)^2}{2\tau^2})$$

When x is vector valued;

$$w^i = exp(-\frac{(x^{(i)} - x)^T(x^{(i)} - x)}{2\tau^2})$$

where x is the input we are using to find the hypothesis for. Note that the weight w^i depends on the particular point x we are trying to evaluate. So for any input x, we need to go through the training set again to calculate the θ particularly for that input x. If $|x^{(i)} - x|$ is small, then w^i is big and $x^{(i)}$ effects θ more and when $|x^{(i)} - x|$ is small, then w^i is small and it effects θ less. So when it makes a prediction for an input x, it regards the input values in the training set($x^{(i)}$) near the input x more than the ones far away from x. The parameter τ controls how quickly the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x. τ is called the bandwidth parameter.[2]

Locally weighted linear regression is the first example we're seeing of a non-parametric algorithm. The (unweighted) linear regression algorithm that we saw earlier is known as a parametric learning algorithm, because it has a fixed, finite number of parameters (the $\theta_i's$), which are fit to the data. Once we have fit the $\theta_i's$ and stored them away, we no longer need to keep the training data around to make future predictions. In contrast, to make predictions using locally weighted linear regression, we need to keep the entire training set around. The term "non-parametric" (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis h grows linearly with the size of the training set.[2]

See the code in "Linear Regression Locally Weighted Stochastic Gradient Descent" in the GitHub repository for the implementation of this method on an example.

References

[1] A. Ng, A. Avati, R. Townshend, and K. Katanforoosh, "Stanford cs229: Machine learning full course taught by andrew ng [playlist]." `"https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU"`, Apr. 2020. Accessed: (31.07.2023).

[2] A. Ng and T. Ma, "Cs229 lecture notes." "https://cs229.stanford.edu/lectures-spring2022/main_notes.pdf", Feb. 2022. Accessed: (27.07.2023).

[3] J. Daintith and E. Wright, *A Dictionary of Computing*. Oxford University Press, 2008.

[4] Deepanshi, "All you need to know about your first machine learning model-linear regression." "https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/#:~:text=In%20the%20most%20simple%20words,the%20dependent%20and%20independent%20variable.", May 2021. Accessed: (27.07.2023).