Basics and Beyond: Linear Regression

Taking apart the linear regression algorithm piece by piece

Kumud Lakara · Follow
Published in Analytics Vidhya · 8 min read · Dec 26, 2020

Photo by Matthias Oberholzer

This post will walk you through linear regression from the very basics. When starting off with machine learning Linear Regression is probably one of the first topics that one comes across and honestly it is one of the most widely and easily implemented algorithms as well. In order to master machine learning it is imperative to have the basics very clear. It may seem exhausting at first but once you have the basics perfect, writing the code will be a cake walk. Most posts try to cover everything in one go and that honestly becomes overwhelming at times. Trying to understand complex equations and then switching over to code then back to mathematics and graphs just makes it difficult to keep up. Well this series aims to guide you through machine learning with a slightly different approach. We will first take apart the algorithm and understand it in absolute detail after which we will move on to implementation in a following post. So let's get started!

Linear Regression comes under the category of supervised machine learning algorithms. In supervised learning when given a data-set, we already know what the correct output should look like, we already have an idea of the

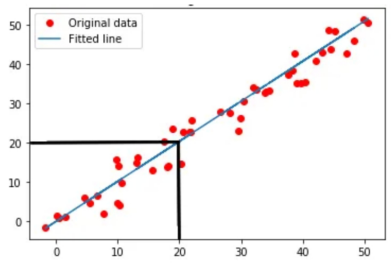
relationship between the input and the output. Supervised learning broadly covers two types of problems:

1. Regression problems (our focus in this post)
2. Classification problems

Now with all that background it is quite obvious that Linear "Regression" does in fact come under the "regression" Problems category. But what exactly are these "regression" problems? Well, in simple words regression problems try to predict results within a continuous output i.e they try to map input variables to some continuous function. The output here is a continuous set. It also helps to remember that when the target variable we are trying to predict is continuous (e.g. in mathematical sense [1,5] is a continuous set where as {1,5} is discrete) then it is a regression problem.

Some examples of regression problems are :

- Predicting price of houses given the area, number of rooms etc.
- Calculating fare for a taxi depending on the distance, traffic etc.

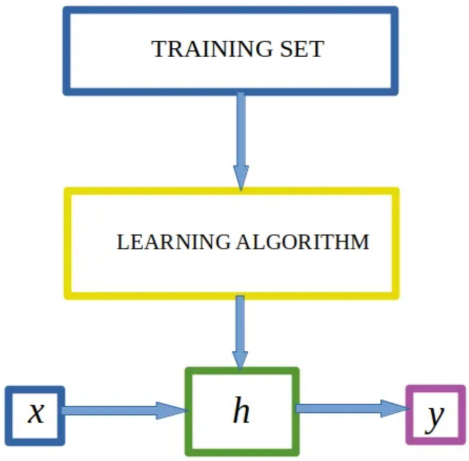


Linear Regression visual representation

In any supervised learning problem, our goal is simple:

"Given a training set, we want to learn a function h: X→Y so that h(x) is a good prediction for the corresponding value of y"

Here *h(x)* is called the hypothesis function and is basically what we are trying to predict through our learning algorithm (Linear Regression in this case).



h maps x to y

Lets start off linear regression with uni-variate linear regression (one variable problems).

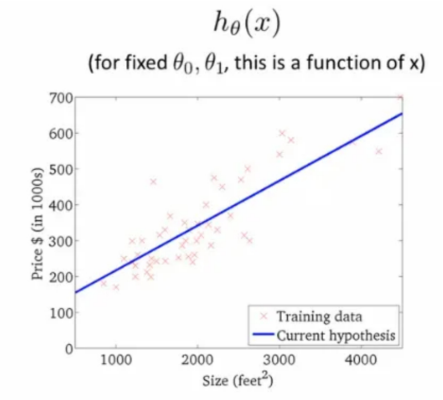
For the case of uni-variate linear regression our hypothesis function is:

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

Equation for h(x)

In the above equation, *θ0* and *θ1* are called the *parameters* of the hypothesis.

A visual representation will be of help here:



Graphical representation of linear regression

Our hypothesis function *h(x)* that we have been talking about is essentially the blue line that runs through the data and the red X's are our data-points. Hence, our aim with linear regression is to find the line that best fits the data. You may have also noticed that our equation for *h(x)* is actually just the mathematical equation for a line in a 2-dimensional plane and now we have seen that our hypothesis *h(x)* is in fact a line in the graphical sense as well hence the term *"linear"* regression.

Cost Function

So far we have a hypothesis that we know will give us our required predictions. But how do we evaluate how well our hypothesis function performs. Its important to know how accurate our predictions are in order to know how well our model performs and if it needs further"training" or more "tuning" (which is basically adjustment of the parameters). This is where the cost function comes into the picture.

The cost function is an expression through which we evaluate the quality of our current hypothesis and proceed to make changes accordingly. In simpler words, our cost function decides the cost we want our model to incur depending on how far off our predictions are from the true value. It is only intuitive to think that the "cost" should in fact be the difference between our prediction and the true value i.e. *h(x)-y.* This actually is a correct intuition and thus we arrive at our cost function for linear regression:

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

Cost function

Now that we have the hypothesis, parameters and the cost function we once again reiterate what our goal is here.

The main goal here is to minimize our cost function $J(\theta)$ so that we get $h(x)$ as the function which passes through maximum points in the plot of X and Y or in other words we want to minimize the cost function so that the predictions of our model are as close as possible to the actual values.

But why minimize the cost?

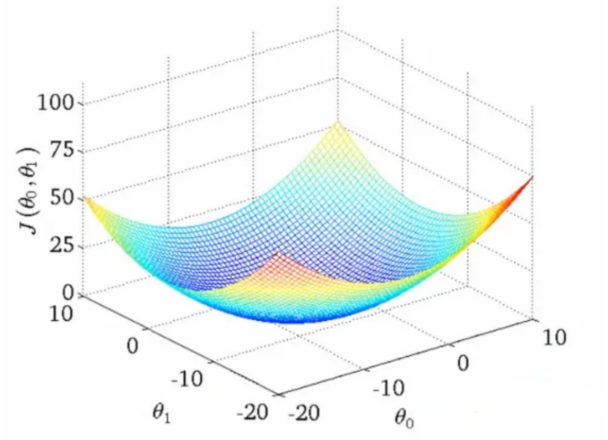
From the equation of the cost function it is quite clear that the cost function $J(\theta)$ is directly proportional to square of the difference between our prediction i.e. $h(x)$ and the true value or the label y. Since, we want our predictions to be very close or equal to the true values we will obviously need the difference between the two to be as small as possible and hence we must minimize the cost function.

So how do we minimize this cost function? Enter Gradient Descent!

Gradient Descent

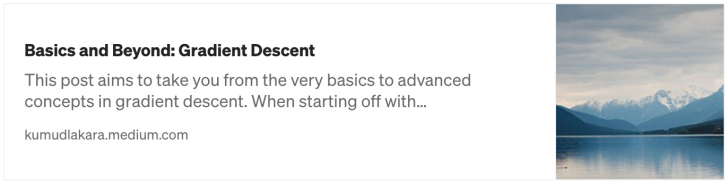
Gradient Descent in our context is an optimization algorithm that aims to adjust the parameters (θ0 and θ1 here) in order to minimize the cost function $J(\theta0,\theta1)$.

Lets think about it this way, imagine a bowl. Any point on this bowl is the current cost and our aim is to reach the bottom of the bowl which is called the “global optimum or minima”. This is exactly what gradient descent tries to achieve. It selects parameters, evaluates the cost and then adjusts these parameters so as to get a lower cost than the previous one hence inching a step closer to the minimum. Once we reach the global minimum or intuitively the bottom of the bowl we will have the best parameters for our hypothesis function and hence be able to make accurate predictions.



Gradient descent

Gradient descent itself is a vast topic. Check out this post in the same series that takes you through the entire concept of gradient descent step by step:



For our purpose we will directly look at the equation for gradient descent:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

Equation for gradient descent

This equation is the main "update" step of gradient descent where after minimizing the cost we attempt to update our parameters in the right direction. α here is the learning rate. I suggest you develop a deeper understanding of gradient descent if you haven't already in order to be able to understand linear regression (and in fact all machine learning algorithms for that matter) better.

Expanding to multiple features

Well that's pretty much all there is to linear regression. Through gradient descent we arrive at the most suitable parameters and hence the most suitable hypothesis. We can save these parameters and use this hypothesis to make predictions on new data outside our data-set.

So far, our discussion in this post has been centered around uni-variate linear regression for the sake of simplicity and easy understanding. But fortunately it is very easy to extend these concepts to multiple linear regression as well. Let's take a quick look!

Multiple Features

Well the only major change we need to make is to our hypothesis function. Instead of *θ0* and *θ1*, we have more parameters simply because we have more features.

so our hypothesis in this case would look something like this:

$h\theta(x) = \theta 0 + \theta 1 x1 + \theta 2 x2 + \theta 3 x3 + \ldots + \theta n x n$

Our cost function also now depends on more than just one or two parameters and will therefore need to be minimized with respect to all the parameters. The cost function will now look something like this:

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

Cost function for linear regression with multiple features

In case of gradient descent, we follow the same update rule but in this case we update simultaneously for all the parameters (θ0....θn) :

New algorithm $(n \geq 1)$:
Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$) $\}$

Gradient descent for linear regression with multiple features

It is also important to note that here this term:

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Derivative or slope of the cost function

is in fact the slope of the cost function hence our core update rule for gradient descent remains the same as:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

The only difference in case of multiple features is that we update all parameters simultaneously and not just $\theta 0$ and $\theta 1$.

That's it!

Congratulations! You now know what Linear Regression is and how it works. It's okay if you didn't understand everything in one go. Its not the easiest thing to grasp but the secret is to keep going over it again and again. Trust me, all this math does start to make sense after a point. Till the implementation post in this series I would suggest you look at some implementations of linear regression and try to take the code apart and understand how everything works. This will help you get ready for implementing this algorithm on your own when we do so in the following post!

. . .

References

1. https://www.coursera.org/learn/machine-learning/home/
2. https://www.holehouse.org

AI | Machine Learning | Deep Learning | Artificial Intelligence | Technology

Kumud Lakara in Towards Data Science

Coding Linear Regression from Scratch

Implementing Linear Regression from absolute scratch using python

9 min read · Jan 5, 2021

343 7

Yogesh Haribhau Kulkarni (Ph... in Analytics Vidh...

Microsoft AutoGen using Open Source Models

That means, without any Open AI API costs

4 min read · Oct 20

265 2

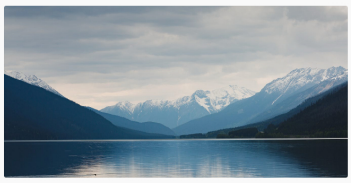
Sajal Rastogi in Analytics Vidhya

Wifi -Hacking using PyWifi 🔐

.

4 min read · Feb 6, 2021

403 3



Kumud Lakara in The Startup

Basics and Beyond: Gradient Descent

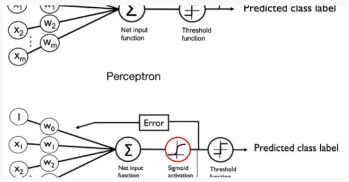
This post aims to take you from the very basics to advanced concepts in gradient...

6 min read · Dec 25, 2020

301 1

See all from Kumud Lakara | See all from Analytics Vidhya

Recommended from Medium



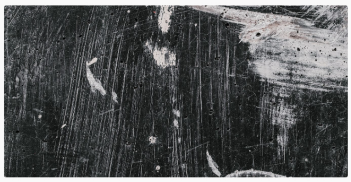
AmyChu

#11 What is a linear classifier (Logistic Regression)

The previous Perceptron can successfully achieve binary classification, but it can only...

5 min read · Aug 20

1



Matteo Possamai in Python in Plain English

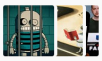
A Comprehensive Guide to Building Linear Regression from...

AI from Scratch in Python

✦ · 3 min read · Aug 5

92 2

Lists

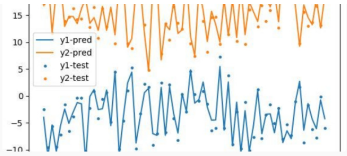


AI Regulation
6 stories · 208 saves

ChatGPT prompts
30 stories · 744 saves

Generative AI Recommended Reading
52 stories · 463 saves

Natural Language Processing
924 stories · 439 saves



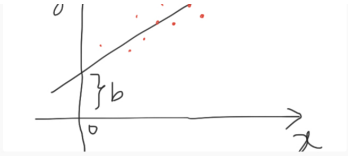
Francesco Franco

Multioutput Regressions with SVMs in Python

Support Vector Machines can be used for performing regression tasks- we know that...

10 min read · Oct 18

15



Dilip Kumar

Linear Regression Model using Gradient Descent algorithm

What is Machine learning?

4 min read · Jul 4

11

Can Benli

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) algorithm is an approach used in regression and...

6 min read · Oct 18

239

Manish Kumar

Understanding Linear Regression Optimization

Gradient Descent and OLS Approaches

7 min read · Sep 11

See more recommendations