All about OLS

“At a basic level, machine learning is about predicting the future based on the past. For instance, you might wish to predict how much a user Alice will like a movie that she hasn’t seen, based on her ratings of movies that she has seen. This prediction could be based on many factors of the movies: their category (drama, documentary, etc.), the language, the director and actors, the production company, etc. In general, this means making informed guesses about some unobserved property of some object, based on observed properties of that object” - Hal Daumé III

This is the first paragraph of the excellent book on machine learning A Course in Machine Learning by Hal Daumé III. The First chapter of the book delves into decision trees which is a great starting point, but we’re choosing a different one. We’re going to start with explaining the Ordinary Least Squares Regression which is also covered but gives in my opinion a less human centric view of creating models.

Let’s start with defining what a model is in the context of machine learning. We will expand on the definition later, but for now a model is a guess about a target given some set of features. We denote a model here as . We are given a set of targets, , that correspond to a set of features given by . Notice we capitalize and because we these variables are generally matrices or vectors. We want to assume that for every where is the true value and is noise or some amount of randomness. We want to find some model, , where , or in other words, we want our guesses to be a close to the value of our target without noise. Notice that could be anything. It does not need to be related to any features at all, we could have for all of our data if we wanted to.

This isn’t explicitly machine learning but let’s try a simple modeling problem.

using Plots, LinearAlgebra, LaTeXStrings

The scatter plot shows our training data generated from the function where our truth is . We attempt to find a model that estimates as closely as possible.

X = -1:.01:1 |> collect  
Y = 2 .\* X .+ randn(length(X)) \* 0.5  
plot(X, Y, seriestype = :scatter, xaxis = ("features"), yaxis = ("target"), legend = false, framestyle = :origin)

If you didn’t already know the true value maybe you assume that it’s .

plot(X, Y, seriestype = :scatter, xaxis = ("features"), yaxis = ("target"), legend = false, framestyle = :origin); plot!(x -> 1.9x)

It looks right, and if you all you needed was an estimate for the general direction of the data it would work, but we’ll see the advantage of having a clearly defined metric later on. For right now lets try to find some metrics for evaluating our guess. One that’s immediately intuitive metric is adding up the errors of the model.

error = 0  
for (xi, yi) in zip(X, Y)  
 error += yi - 1.9xi  
end  
error

-1.6097571579893621

The sum of the errors is around 5.05. Let’s make another guess,

plot!(x -> 2.2x)

error = 0  
for (xi, yi) in zip(X, Y)  
 error += yi - 2.2xi  
end  
error

-1.6097571579893613

Assuming we are only guessing lines with no intercepts of form we can plot the relationship between the sum of errors of any given model at

errors = Dict()  
for w in 0:.01:4  
 error = 0  
 for (xi, yi) in zip(X, Y)  
 error += yi - w\*xi  
 end  
 errors[w] = error  
end  
plot(errors, xaxis = ("w"), yaxis = ("error"), legend = false)

What a messy function. We can see a general trend in that as approaches 2 the error gets closer to 0, but that’s where the utility of this metric ends. We aren’t guaranteed a smaller error as we approach the value of that brings us closest to the true function. How do we even interpret negative error? If a line has both negative errors and positive errors this metric will cancel them out and that’s most likely what’s leading to the relatively small variations in the metric. Let’s try a different approach, taking the sum of the squares of the errors.

errors = Dict()  
for w in 0:.01:4  
 error = 0  
 for (xi, yi) in zip(X, Y)  
 error += (yi - w\*xi)^2  
 end  
 errors[w] = error  
end  
plot(errors, xaxis = ("w"), yaxis = ("error"), legend = false)

Much better. This metric continuously decreases as our model gets closer to the truth and also looks suspiciously like a parabola for which we have methods for finding the value of w that minimizes the function.

I have a confession, we’ve been using the term metric wrong. Metric has a specific connotation within machine learning. The two “metrics” we’ve developed are actually loss functions. We can consider the first one just sum of errors, and it really isn’t a great loss function. The second one you might have seen before, the Sum of Squared Errors and the minimization of this function with respect to different models has been a centuries long endeavor, generally called OLS (Ordinary Least Squares).

You might be wondering why we didn’t just use the sum of absolute values of the errors, that loss function has its own applications that can be explained through the differences in the geometries of these functions.

errors = Dict()  
for w in 0:.01:4  
 error = 0  
 for (xi, yi) in zip(X, Y)  
 error += abs(yi - w\*xi)  
 end  
 errors[w] = error  
end  
plot!(errors, xaxis = ("w"), yaxis = ("error"), legend = false)

For now, we’ll stick to sum of squares. Let’s formally define and then expand SSE (Sum of Squared Error)

For this example we define Let’s take a step back and look at our model, in your general algebra 1 class you were probably asked to find the roots or zeros of a function by manipulating . For our purposes and are constants, we cannot manipulate them. We can only manipulate . We have names for and (features and target), we also have a name for variables like , parameters. In OLS (Ordinary Least Squares) problems, we want to find the parameters (in this case w) of a model that minimize the SSE (Sum of Squared Errors). Let’s replace with our equation.

Expanding the inside of the polynomial gives us

Getting closer to a regular parabola. Split up the summation.

Take out the terms that aren’t being summed

Reorder the terms

And we get the shape we expected all along. SSE with this specific model can be described as quadratic function with respect to . Remember, and are constants in regard to the loss function. Remembering a little more from algebra 1, for this quadratic function,

,

Because is the sum of squares of it’s always going to be positive meaning the vertex is going to be the minimum value. Plot a few parabolas yourself, and you’ll see this true.

plot(x -> x^2 + 2\*x + 1, framestyle = :origin); plot!(x -> x^2 - 2\*x + 1); plot!(x -> x^2 + 2\*x - 1)

The equation for the vertex is

Substituting the values for and we get

sum((yi\*xi) for (xi, yi) in zip(X, Y)) / sum(xi^2 for xi in X)

1.9624178096653309

Not exactly 2.0 but remember the noise in this data set effects our efforts in getting the true value of the function.

Lets consider another approach in minimizing SSE. You only need a little bit of calc 1 to get through this example but going forward you’ll need a strong background in multivariate calculus. For the most part we’ll derive the equation for the vertex.

We’re going to take the derivative of the loss function with respect to w.

It’s important to remember this is the loss of this model not of every model you come across.

Using the power and chain rule we can do this in one step.

Lets simplify this further

Seperate the terms

Take out what can’t be summed

Can you see where this is going? Setting the derivative to 0 gives us the vertex of the polynomial which for reasons we have already established is a minimum, but within the context of calculus you can see that this point should be zero. If we take the second derivative of the equation we get

Which is a positive constant, in other words the second derivative of the turning point of the polynomial is going to be positive, therefore the turning point will be a minimum. This equation also only has one turning point with respect to , therefore this turning point will be the global minimum of the function.

We get the turning point by setting the derivative to 0 and solving for w.

Add the second term to both sides

divide the everything in the first term except on both sides

We get the equation we solved for before, so why did we go through the trouble of proving the minimum of SSE for this model?

Lets say instead of just solving for the slope we want the intercept too, ie. our model is

Lets change the data accordingly.

The new truth () is going to be

Y = 2 .\* X .+ 3 .+ randn(length(X)) \* 0.5  
scatter(X, Y, ylims = (-5, 5), legend = false, framestyle = :origin)

We can plot the potential outputs of our old model

errors = Dict()  
for w in 0:.01:4  
 error = 0  
 for (xi, yi) in zip(X, Y)  
 error += (yi - w\*xi)^2  
 end  
 errors[w] = error  
end  
plot(errors, xaxis = ("w"), yaxis = ("error"), legend = false)

Notice the error is higher than our previous fit.

We have 2 parameters now, ( and ). Our objective is to minimize SSE with respect to both paramaters. We can’t use our algebra 1 method for solving this equation. We’re going to need some multivar going forward.

SSE is

Substituting our new model we get.

Lets visualize the loss function given these 2 parameters

plot(2:.01:4, 1:.01:3, (b, w) -> sum((yi - w\*xi - b)^2 for (xi, yi) in zip(X, Y)), st = :surface, xlabel = "b", ylabel = "w", zlabel = "SSE", legend = false)

The loss is a paraboloid. While their most likely is an equation for the vertex of a paraboloid, the moment we start increasing parameters that equation becomes useless. We want a general way of finding the solution for OLS given a model like this.

Take the derivative with respect to w and b and solve for the parameters while setting the derivative to 0.

This is ugly, let’s fix that. The bar above the variable means we are taking the algebraic mean (average) of that variable, In other words $ = {x}$

We need to solve for w and insert this somewhere.

Plug that back in to

Lets try it out

ybar = sum(Y) / length(Y)  
xbar = sum(X) / length(X)  
w = sum((yi - ybar)\*xi for (xi, yi) in zip(X, Y)) / sum((xi - xbar)\*xi for xi in X)  
b = ybar - w\*xbar  
scatter(X, Y, legend = false, ylims = (-5, 5), framestyle = :origin)  
plot!(x -> w\*x + b)

("w" => w, "b" => b)

("w" => 2.0199647444294064, "b" => 2.971490925147424)

Compare this to the optimal solution found by our previous model

wo = sum((yi\*xi) for (xi, yi) in zip(X, Y)) / sum(xi^2 for xi in X)  
plot!(x -> w\*x)

("w" => w)

"w" => 2.0199647444294064

plot(-1:.01:4, -1:.01:4, (b, w) -> sum((yi - w\*xi - b)^2 for (xi, yi) in zip(X, Y)), st = :surface, xlabel = "b", ylabel = "w", zlabel = "SSE", legend = false)  
scatter!([b], [w], [sum((yi - w\*xi - b)^2 for (xi, yi) in zip(X, Y))], markersize = 3, markerstrokewidth = 1)  
scatter!([0], [wo], [sum((yi - wo\*xi)^2 for (xi, yi) in zip(X, Y))], markersize = 3, markerstrokewidth = 1)

The orange point is our new model while the green is our old model.

Let’s look back at the equations for w and b we got

You might have seen different equations used to represent these parameters. If you haven’t you might be wondering why we chose to use the mean of x and y within the equation. We have one more branch of math (you might find this specific opinion controversial) to introduce. My least favorite visitor, Statistics.

Honestly it shouldn’t be that surprising that we’d eventually encounter statistics, when looking at our original equation for what we expect our data to be

Where is random, when we define the distribution of we get a different interpretation of OLS, SSE, and even SAE, Sum of Absolute Error.

Substituting our model as an approximation of we get

We can reorder this equation to be

Here’s where we define the distribution of , we can choose multiple but for now we’ll use the normal distribution.

Knowing that adding a nonrandom variable to a normal distribution shifts the mean that amount we can also conclude that.

And through the use of maximum likelihood estimation, we can derive equations for these parameters. In a basic sense, maximum likelihood estimation is a method of finding the parameters of distribution that maximize the likelihood (straightforward I know) that some set of data follows said distribution. Given this distribution for we can conclude the probability of given , , and is

I might be losing you, this next part isn’t getting any simpler. If you want a statistical understanding of these concepts you need a strong understanding of statistics. PennState offers course notes for its online versions of STAT 414 (Introduction to Probability Theory) and STAT 415 (Introduction to Mathematical Statistics). I’d recommend you read through all of it but here are some entry points for understanding the concept of maximum likelihood estimation

https://online.stat.psu.edu/stat414/lesson/8

https://online.stat.psu.edu/stat414/lesson/16

https://online.stat.psu.edu/stat415/lesson/1/1.2

Essentially, we want to maximize the likelihood that the data comes from the normal distribution. The likelihood of data is multiplication of the probabilities of each individual data point given that if is from said distribution. We have the probability of given with parameters and . Thus the likelihood is

Which is equivalent to

The likelihood the data comes from a distribution where , , and are 1.5, 2, and 1 respectively

w = 2  
b = 1.5  
σ = 1  
prod([1/sqrt(2\*pi) \* exp(-1\*((y -(w\*x + b)) / σ)^2) for (x, y) in zip(X, Y)])

1.6517281145334849e-292

Looks pretty low but remember probabilities are between 1 and 0, and we are multiplying 201 of them. Fixing and at 1.5 and 1, we can plot the likelihoods of the distribution at different values of as

plot(0:.01:4, [prod([1/sqrt(2\*pi) \* exp(-1\*((y -(w\*x + b)) / σ)^2) for (x, y) in zip(X, Y)]) for w in 0:.01:4])

By finding the combination of and that maximize the value of the likelihood, we get very close to the true value of the parameters.

So how do we go about doing this?

Taking the log of this expression gives us the log likelihood

Expanding this out a little gives us

Thus maximizing the log likelihood with respect to and means minimizing

Which is getting really close to our original SSE minimization problem. We can manipulate the expression a little more.

And when we minimize this function with respect to the derivatives of , , and becomes irrelevant. Thus we get our original problem.

Notice how our model is self-contained thought the entire derivation. There’s no ’s or ’s spilling everywhere. This derivation can be extended to most models of this form. In other words, minimizing SSE for some deterministic model is equivalent to treating the errors as independent and identically distributed variables with a Normal Distribution. Consider another choice for our distribution, the Laplace or double exponential distribution. Given this choice the probability of becomes

In which maximizing the log likelihood becomes minimizing

Or our Sum of Absolute Errors

We have to make one last introduction, from this point on we’ll be using linear algebra to preform a multi variable linear regression using the model

We make two crucial distinctions about our parameters and data. While we previously treated as a vector and as a scaler, we now will treat as a matrix and as row vector of , for example, take the matrix below.

is going to be the vector ,

is

is

Because each row vector has 2 elements, and there are 3 vectors, we consider and .

As is in , in order for to be valid needs to also be in

Let’s go over what is, we refer to this one vector multiplied by the transpose of another vector as the inner product, and sometimes the dot product. The differences between those two concepts is somewhat irrelevant, but it’s important to remember that there is a difference.

We can represent this as , or for our example in

For our example , if then is equal to

We’ve been dealing with a single vector. We haven’t been dealing with multiple features yet, but if we rethink our previous problem we can see an immediate application of viewing our problem like this.

Instead of representing as

We will now refer to this vector as

If we add a column of ones we get

And when we take we get

or simply

which is the same is equivalent to our previous model

If we add another column we get the matrix

which is equivalent to

Which means we can now fit quadratic functions accurately.

We can add more columns to fit data to nearly any polynomial, but we’re getting a little ahead of ourselves, let’s derive the closed form solution for this problem.

Looking at our regular SSE formula we have

is going to be a vector of each of the inner product of the row vectors of the matrix X and w, or

and we already know that corrosponds to

So is equivalent to

and

Is just the sum of the elements in the vector or the inner product of

Where is just a vector of ones or to whatever dimension

Notice we haven’t squared the errors yet.

is equivalent to

Going back to our matrix

is

Thus

is

Which is equivalent to

So

As we go on, you’ll be able to see some similarities between multiplication, inner products, and the properties of transposes. There are alot of properties of matrix multiplication and transposes that can be unintuitive so lets take this step by step.

We distribute the first transpose to the terms of the first parenthetical

We foil the expression like we would an algebraic polynomial

We observe that and both simplify to a variable in and that . Let’s say they both evaluate to some value . The equation becomes where . The transpose of elements in the set is equal to the element. Thus, and

Taking the gradient with respect to gives us

For some intuition on this could be thought of as

Set the gradient to 0

Take out the constants

Solve for w

And we have our general solution for multivariable linear regressions

xv = X  
Xn = hcat(xv, X .^ 0)  
w = inv(Xn'Xn)\*Xn'Y  
scatter(X, Y, framestyle = :origin, legend = false)  
plot!(x -> w[1]\*x + w[2])

"y = $(w[1])x + $(w[2])", "SSE = $((Y - Xn\*w)' \* (Y - Xn\*w))"

("y = 2.0199647444294087x + 2.971490925147423", "SSE = 51.8970131427807")

Using this same method, lets try fitting quadtratic data

xv = X  
Y = 3 .\* xv .^ 2 + 2 .\* xv .+ 1 .+ randn(length(X)) \* 0.3  
Xn = hcat(xv, X .^ 0)  
w = inv(Xn'Xn)\*Xn'Y  
scatter(X, Y, framestyle = :origin, legend = false)  
print("y = $(w[1])x + $(w[2])", " SSE = $((Y - Xn\*w)' \* (Y - Xn\*w))")  
plot!(x -> w[1]\*x + w[2])

y = 1.9775956644033128x + 1.9902318689281158 SSE = 167.09835002975353

Lets add an column to our matrix

xv = X  
Y = 3 .\* xv .^ 2 + 2 .\* xv .+ 1 .+ randn(length(X)) \* 0.3  
Xn = hcat(xv .^2, xv, X .^ 0)  
w = inv(Xn'Xn)\*Xn'Y  
scatter(X, Y, framestyle = :origin, legend = false)  
print("y = $(w[1])x^2 + $(w[2])x + $(w[3])", " SSE = $((Y - Xn\*w)' \* (Y - Xn\*w))")  
plot!(x -> w[1]\*x^2 + w[2]\*x + w[3])

y = 3.047025916094448x^2 + 1.9767385274041585x + 0.989739981181494 SSE = 17.287928758692843

One of the most important realizations you should make about linear regression and other linear methods is that the word “linear” refers to the linear combination of variable. Given the right feature engineering, remember we call the columns of our data matrix features, we can model more than just hyperplanes. Adding a and constant column does not make this nonlinear, it just makes it not a line. Despite their linearity, this family of models is more than capable of producing a large variety of versatile regressions.