SFS Machine Learning Repo

Lasso and Ridge Explained Mathematically

Contents

1 Introduction **2**

2 OLS, conceptually revisited **3**

3 Lasso **4**

3.1 Lasso: an alternative visualization 5

4 Ridge **7**

5 Thanks and Citations **8**

1 Introduction

Yay! You opened this because you like math. In this document, we'll summarize the mathematical concepts needed to understand the basics of what's going on in Lasso and Ridge. Unless I cite things explicitly, assume the knowledge comes from my course with Prof. Ani Silwal in in Spring 2019. I also encourage you to refer to this book for a great theoretical explanation of just about anything: http://faculty.marshall.usc.edu/gareth-james/ISL/index.html.

2 OLS, conceptually revisited

Let's pretend we're students entering our fourth year at Hogwarts School for Witchcraft and Wizardry. We really need a new broomstick, so we plan a trip to Diagon Alley. We're eyeing the Firebolt model but, since we're not familiar with the muggle internet, we can only *estimate* the true cost of the broomstick. Good thing we've taken Arithmancy with Professor Tiongson. He taught us that we can generalize the cost of any broomsticks, y, by creating a mathematical relationship between known variables, or x's. But we don't know what the real y's actually are, so we'll need an estimator called $\hat{y}$ that uses some information from our x's to take a guess at the real y. Now there are of course many variables we may use to estimate the cost of a broomstick, such as x_1, the average cost of last year's top selling broomstick models, x_2, the yearly rate of inflation, and, knowing that pricing patterns often relate to the product release cycle, x_3, the number of days since the last major broomstick model release. We might also want to take into account the number of features the broomstick offers, the broomsticks' length, weight, the number of hours required for the broomstick's production, and whether or not Lucious Malfoy, being among the snobby and pure-blood supremacist families, plans to purchase a set for the Slytherin Quidditch Team. If it peeks Malfoy's interest, it's not likely to be priced for the common folk. See? The number of variables at our disposal quickly adds up.

Good thing we paid attention in class! We learned that we can express $\hat{y}$ by using the "standard" linear equation model, which looks something like this

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_n x_n + \varepsilon$$

where β_0 is our constant, β_i's our coefficients, and ε our error term. So when we `reg y x` in Stata, we're basically just telling the computer to fit the data to this equation the best it can.

But How does Stata actually calculate all the components of the equation? It minimizes the sum of the squared error term, or minimizing

$$(all\, distances\, from\, the\, 'dots'\, on\, the\, scatterplot\, to\, the\, line\, running\, through\, them)^2$$

We can express this idea mathematically by adding up all of the differences between the actual broomstick costs and estimated costs like this

$$\sum(y_i - \hat{y})^2$$

Why do we square the difference between the estimated and actual measurements? Who knows. What it does, in essence, is disproportionately penalize the estimated measurements that are *farther* away from the actual measurement compared to those that quite close. Some old white dude in an Ivy-League school probably decided it was best to do it this way, and so here we are.

Since we know that $\hat{y}$ can be expressed as $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_n x_n + \varepsilon$, we can substitute this later equation into our $\sum$ equation (and distributing the negative to all terms in the expression):

$$\sum(y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_2 - \beta_n x_n)^2$$

Remember, the goal is to minimize the value that results form this equation, which is the all of the error terms added up. Stata does this math in the background, spits out the coefficients that minimize the error value above, and we go on our merry way. If you ever want to know what math Stata is actually doing, it's usually buried out of sight in the command's help file.

3 Lasso

Lasso adds a very slight modification to the OLS equation that makes it super meta. We'll get to this in a moment but first let's walk through what Lasso and Ridge algorithms are trying to do – they're categorically different from OLS. OLS gives you coefficients for the only variables you give it; it does not tell you which variables to include or exclude in your model. You say, "great, I think y_1 the price of a broomstick relates to x_1, the average cost of broomsticks last year, and x_2, the number of features it has", you give those three variables to Stata as `reg price avcost nfeatures`, and Stata runs the equation above and gives you the value of coefficients to the input variables you gave it. Lasso (and Ridge) algorithms, on the other hand, tell you *which* variables to include in your model (and, optionally, what the coefficients are on the variables the algorithm selects). This is a similar idea to stepwise functions. But unlike stepwise functions and OLS, the functions that Lasso and Ridge try to satisfy introduce a λ variable. This lambda is an unknown constant that acts as a penalty applied to each additional coefficient – or variable – you include in the model. Some guy named Tibshirani thought, in this age of computers, we should discourage people from just throwing in any variable they can think of and, instead, make sure we focus only on the few variables that matter the most for predictive power. His goal is to minimize

$$\frac{1}{n}\sum(y_i - x_i'\beta)^2 + \lambda\sum\ |\beta_j|$$

where the β_j's are the coefficients for all the x's and λ is the penalty. Since we want to avoid high penalties, there are two ways we can do that: decrease λ or decrease the number of included coefficients.

One more thing. Notice what happens when $\lambda = 0$?

$$\frac{1}{n}\sum(y_i - x_i'\beta)^2 + 0\sum\ |\beta_j|$$

Right, the final term becomes 0. And this looks a lot like OLS...that's because it is OLS.

$$\frac{1}{n}\sum(y_i - x_i'\beta)^2 + 0$$

So, Lasso algorithms with λ values = 0 is virtually the same equation as OLS. This makes sense, because in this 0-termed Lasso world we are still *minimizing the sum of squared error terms* and there's no penalty for including additional coefficients as lambda is 0.

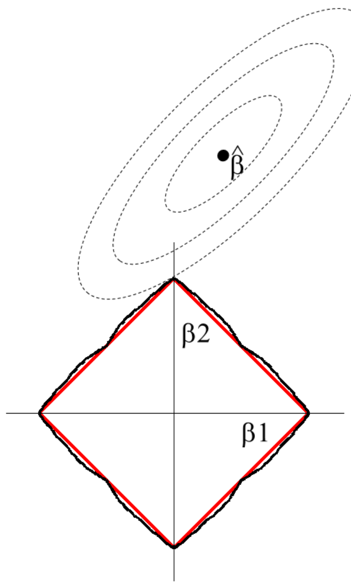
3.1 Lasso: an alternative visualization

We can also think of the lasso expression as solving the following equation. As James et al (2017) put it, Conceptually, for every value of λ, there must be some corresponding s that will both minimize the sum of squared errors (the OLS, left-hand term), that also satisfies the sum of the betas (on the right). In essence, we must perform OLS while satisfying one extra condition.

$$minimize \sum (y_i - \beta_0 - \Sigma \beta_j x_{ij})^2 \ \ such\ that \ \ \Sigma|\beta_j| \ \leq \ s$$

What is s? It's basically a constraint region that limits our ability to changes the values of the betas in order to reduce the Residual Sum of Squares. Notice that the sum of our betas must *less than or equal to s.* Figure 1 helps clarify things.

Figure 1: Solution of Lasso Equation, from Agor153 / CC BY-SA



Here we simplified diagram where the "x" and "y" axes are represented by the value of two betas, β_1 and β_2. Instead of (x, y) our coordinate plane is thus represented by the set of values that represent (β_1, β_2). As such, a point somewhere on the "x" axis would indicate a β_1 of value n and no β_2 – that term is dropped from our equation. Of course, in our models we often have many more explanatory variables than two – so I think to think of the actual math happening in n-dimensions where some betas have values and some are dropped altogether.

The diamond-shape region near the origin is the constraint region determined by s; our solution set must fall inside or on this region. The elliptical circles represent lines of constant error term:

anywhere along the same indicates an array of betas that generate the same value for the Residual Sum of Squares. In traditional OLS, this region is not constrained by s, or the diamond (James et al, 2017; p. 222). In both cases we want to move the "error" ellipses inward as much as possible to minimize the error term. But our solution set must lie on or inside the diamond, so the point at which we minimize the error term and simultaneously satisfy the constraint region is tangential to the constraint region. In lasso, we have a *linear* constraint as indicated mathematically by our $\Sigma|\beta_j| \leq s$ term – or graphically by our diamond-shaped constraint region. Therefore, in lasso *our solution set will almost always reside on an axis.*

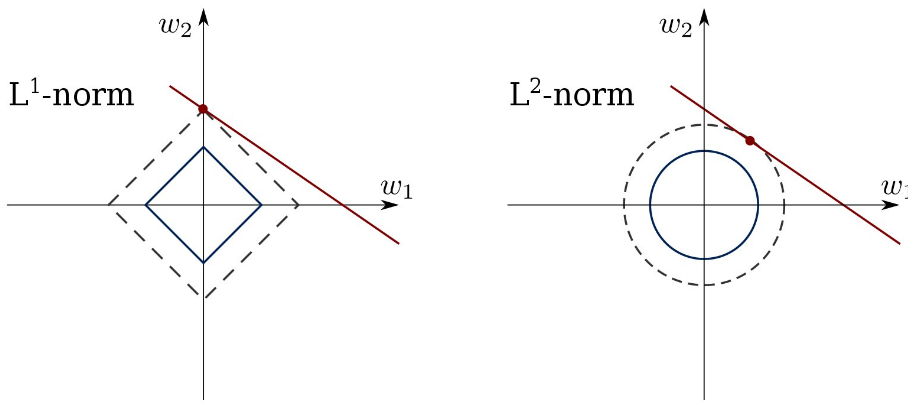
4 Ridge

The general principles of Ridge are the same as Lasso: we determine which coefficients to include in the model by describing the combination coefficients as a function of error. The main difference is the final constraint term: in lasso the term is linear, while in ridge, it's squared:

$$minimize \sum(y_i - \beta_0 - \Sigma\beta_j x_{ij})^2 \;\; such\ that \;\; \Sigma(\beta_j)^2 \leq s$$

This small, one-term difference actually changes things drastically. Let's look at figure 2.

Figure 2: Solution of Lasso and Ridge Compared, adapted from Nicoguaro / CC BY (https://creativecommons.org/licenses/by/4.0)



Because of the squared constraint term $\Sigma(\beta_j)^2 \leq s$, the resulting region is spherical. This also means that our "error" ellipse (here represented by a line, oddly) will never intersect the constraint region on an axis. Thus, we will never drop any coefficients or covariates from the model because the OLS-derived error term will always be minimized at a (β_1, β_2) point where each beta as a non-zero value.

5 Thanks and Citations

James, Gareth; Witten, Daniela; Hastie, Trevor; Tibshirani, Robert. *An Introduction to Statistical Learning with Applications in R*. New York: Springer, 2017.

Silwal, Ani. Advanced Econometrics II. Georgetown University: Spring 2019.