ML Modeling: LASSO, Ridge, Trees & Forests

Brian Quistorff & Matt Goldman

ML != Econometrics (OLS)

Machine Learning is good at selecting from many possible features to predict something.

Econometrics (OLS) is good at carefully measuring a small number of effects that we specify ahead of time.

Intro to Machine Learning

Machine Learning focuses on training models for *predictive accuracy*.

Good for: "How much will I sell next week?"

Different from OLS's strength in *parameter estimation*.

Good for: How much **more** will I **sell if I drop price**? How certain am I about that impact; is it statistically significant?

Key Difference here is whether or not you attribute **causality**.

- We will learn this week, that OLS estimates can sometimes have causal interpretations.
- This is **never** the case with Machine Learning algorithms.
  - Why? Because ML is all about "shrinking" (i.e. biasing) the impact of individual features toward zero to get better prediction.

Should I use ML or OLS?

Two key questions to determine the answer:

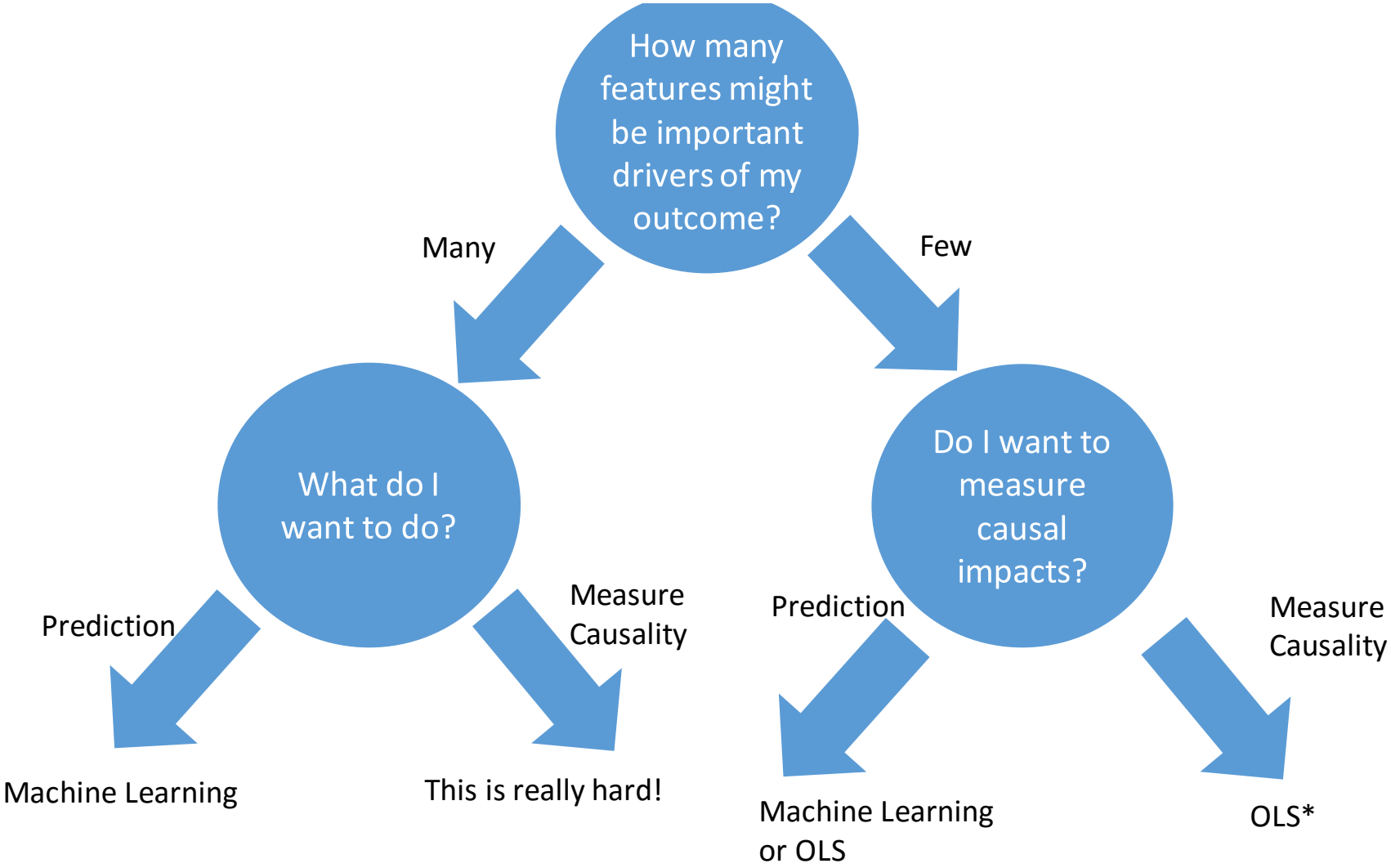
Q1) Do I have in mind (before doing the analysis) a small* number of predictive features that I think drive my outcome?

A) If yes, you can use OLS if not you may need machine learning.

Q2) Do I want to assign a causal interpretation to my estimates?

A) If yes, you need to use OLS.

* Small is relative to the sample size. If you have many data points, you can estimate more complicated models with OLS.



* This does not guarantee you can use OLS to learn about causality. We will learn more about the conditions under which OLS reveals causal parameters, later this week.

Intro to Machine Learning

Machine Learning models are very flexible, but are penalized from adopting *too much* complexity. How does this work?

Idea: what if we make adding additional "stuff" to the model costly?

Same thing as spending $20 at the store

Must carefully evaluate if adding additional features to the model is "worth" it.

What do we mean by "worth it"?

Linear Models: LASSO

Standard OLS Regression

$$\min_\beta \sum_i (y_i - x_i\beta)^2$$

LASSO

$$\min_\beta \sum_i (y_i - x_i\beta)^2 + \lambda \sum_p |\beta_p|$$

Ridge Regression

$$\min_\beta \sum_i (y_i - x_i\beta)^2 + \lambda \sum_p |\beta_p|^2$$

LASSO & Ridge both penalize coefficients towards 0

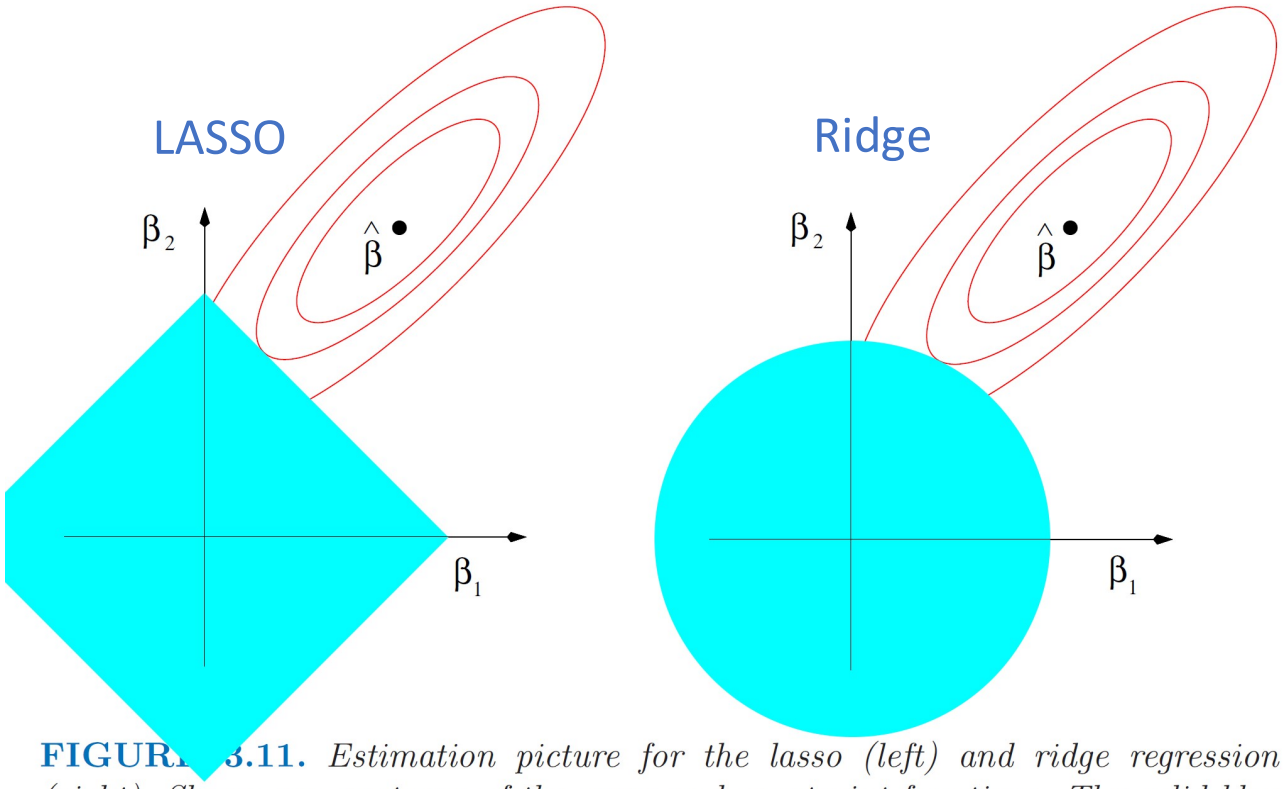
λ controls penalization. Big λ means push most coefficients towards 0 -> bias

Coefficient Budget and Penalization

λ determines (inversely) "budget for coefficients"

"Small λ" = "Big budget" (close to OLS)

Penalty form affects shape (trade-off between params)



FIGUR 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Coefficient Budget and Penalization

Ridge will smoothly make coefficients small

LASSO will push many coefficients to 0 ("model selection")

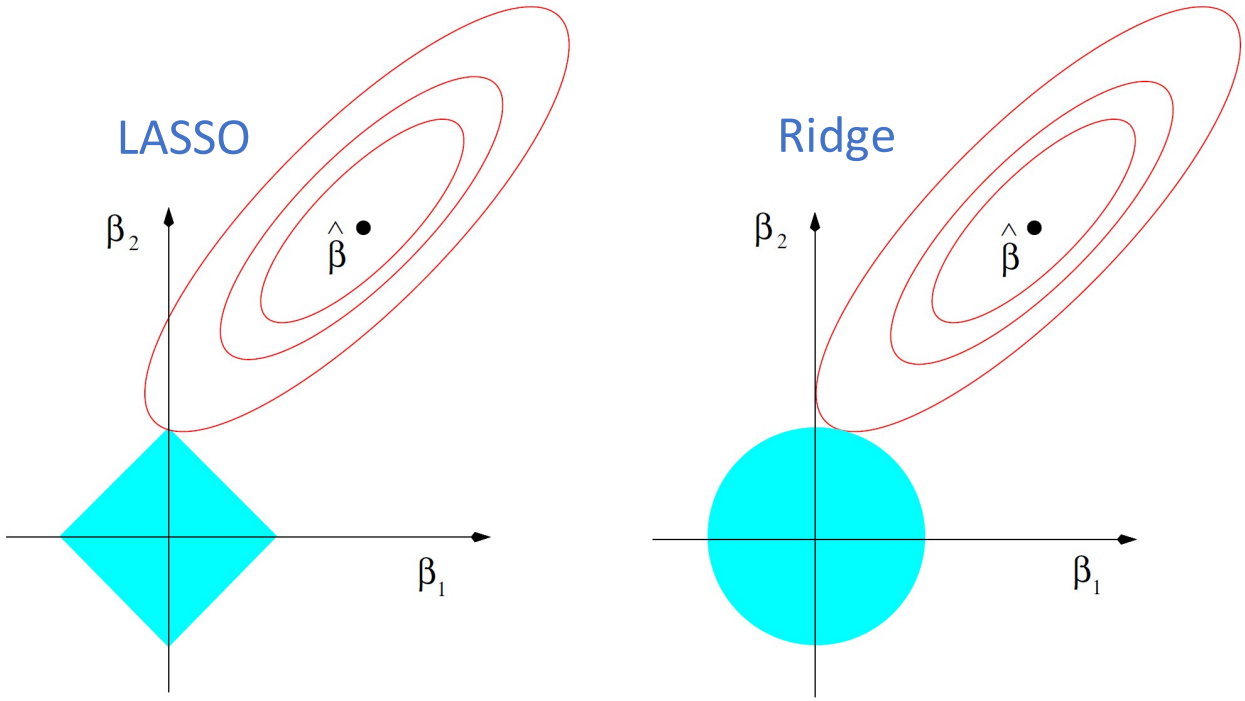
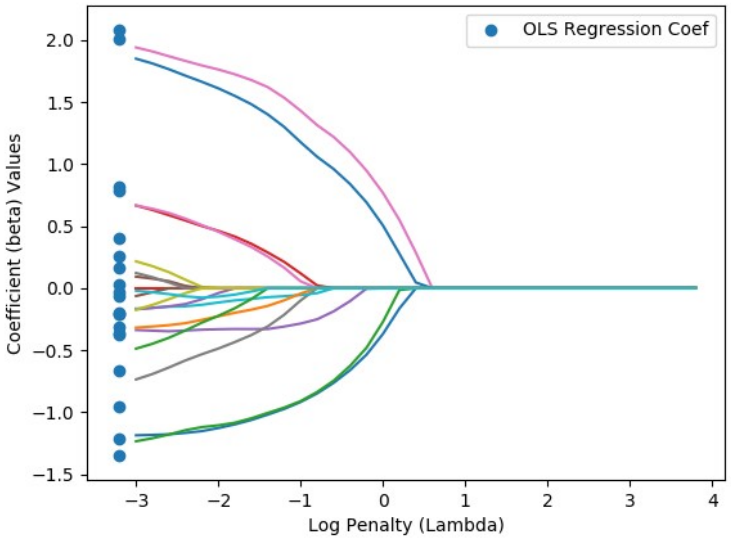


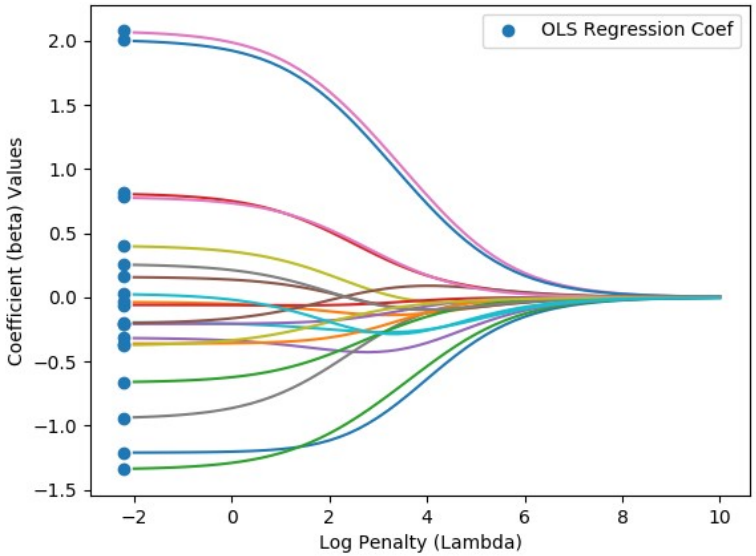
FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Lasso vs. Ridge: Which is Which?

LASSO Regression: $\min_\beta \sum_i (y_i - x_i\beta)^2 + \lambda \sum_p |\beta_p|$

Ridge Regression: $\min_\beta \sum_i (y_i - x_i\beta)^2 + \lambda \sum_p |\beta_p|^2$



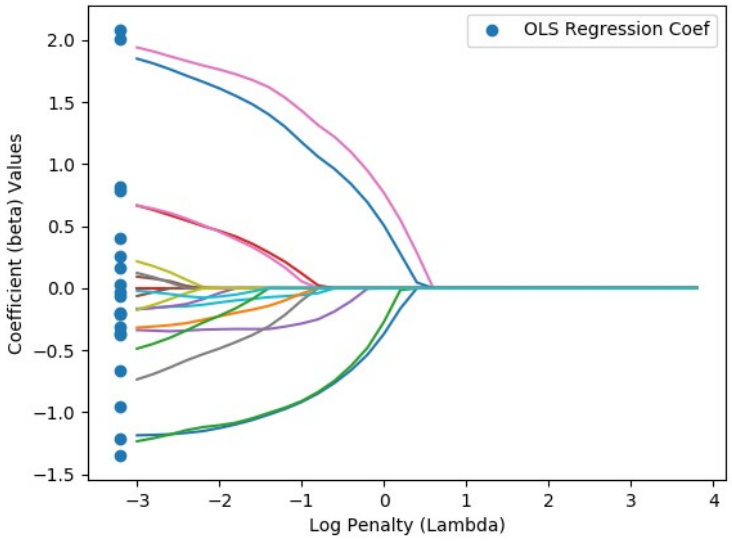


Lasso vs. Ridge: Which is Which?

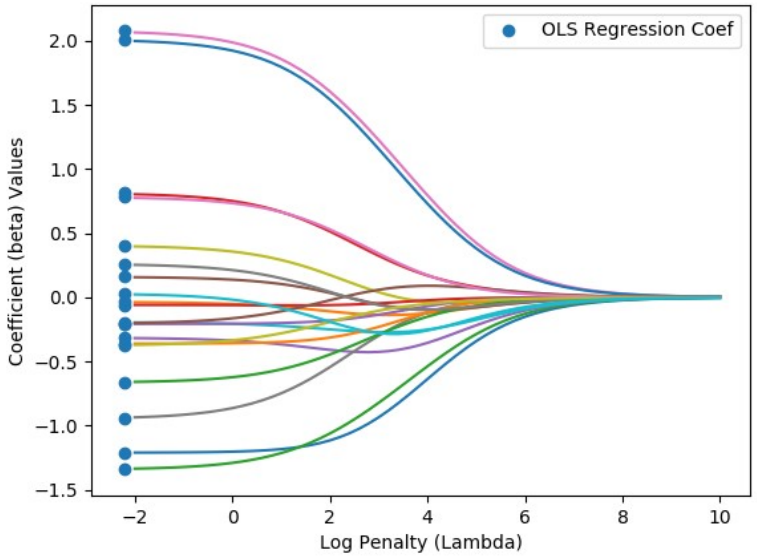
LASSO Regression: $\min_\beta \sum_i (y_i - x_i\beta)^2 + \lambda \sum_p |\beta_p|$

Ridge Regression: $\min_\beta \sum_i (y_i - x_i\beta)^2 + \lambda \sum_p |\beta_p|^2$

Lasso



Ridge



Lasso vs. Ridge: Which is Better?

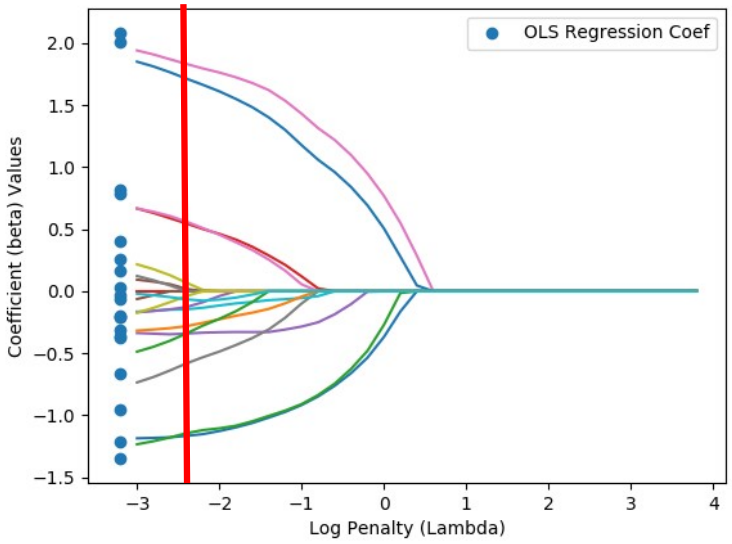
- Ridge penalty scales with the **square** of the size of each coefficient.
  - $\frac{dx^2}{dx}|_{x=0} = 0$. Marginal Cost of moving coefficients away from zero is zero => All coefficients will typically be "selected".
  - Big coefficients are severely penalized (have lots of bias).
  - Will give superior forecasts when the true model is based on many factors that all matter a little.

- Lasso penalty scales **linearly** with the size of each coefficient.
  - $\frac{dx}{dx}|_{x=0} = 1$. Marginal Cost of moving coefficients away from zero is big => A small number of coefficients will typically be "selected" (this is called *sparsity*).
  - Big coefficients are not penalized/biased as much as in Ridge.
  - Will give superior forecasts when the true model is based on a few (unknown) factors that matter a lot.

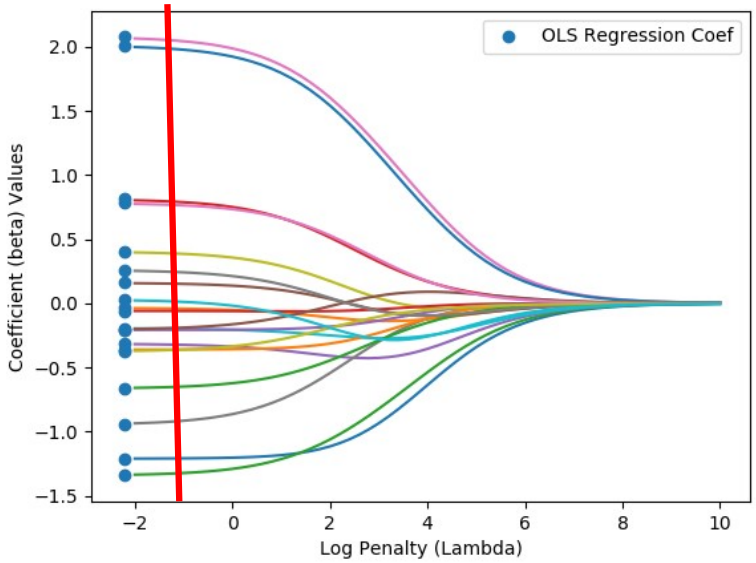
Cross-Validation: Picking the Right Value of Lambda

- How much should we penalize complexity?
- Low penalty = Small Lambda (all coefficients close to OLS values)

Lasso



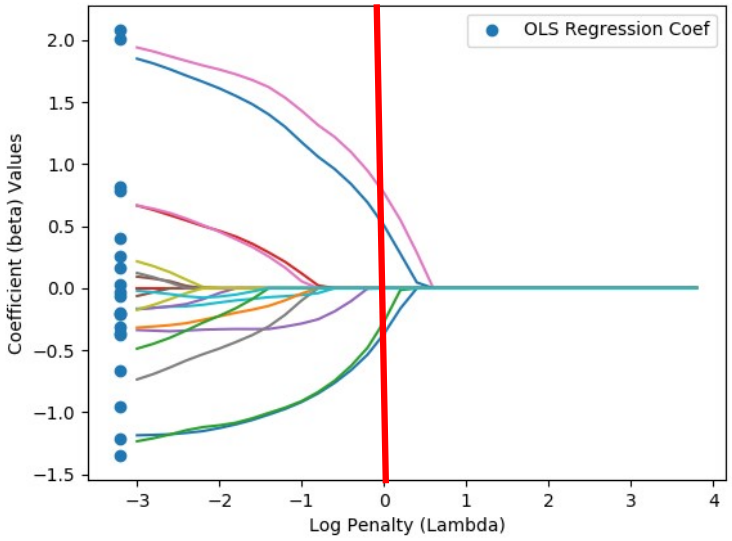
Ridge



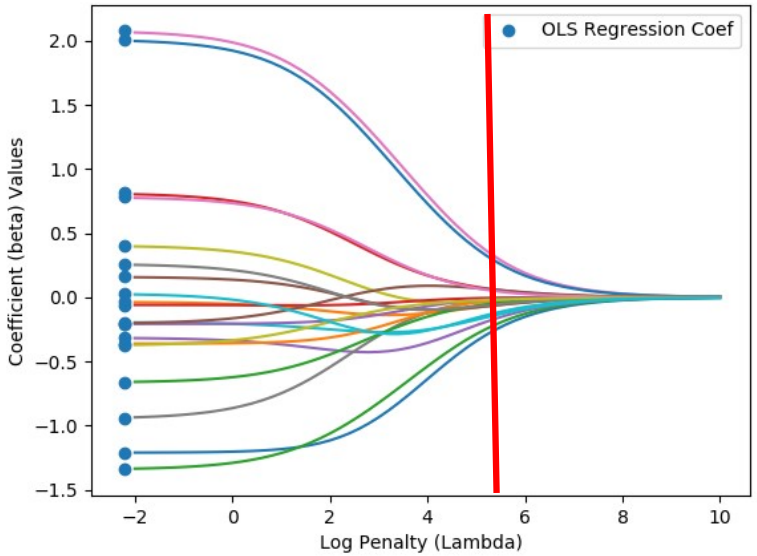
Cross-Validation: Picking the Right Value of Lambda

- How much should we penalize complexity?
- High penalty = Big Lambda (all coefficients close to zero)

Lasso



Ridge



Estimating Prediction Error

Quantify overfitting. Estimate prediction error $\widehat{Error}(\lambda)$

***k*-fold Cross-Validation:**

- Split data randomly into *k* portion (e.g., 5 or 10)
- For each *s* in {1,…,*k*}:
  - Train model on all but section *s* (most of the data)
  - Use trained model to calculate error on section *s*. Call that $Error(\lambda, s)$
- Then $\widehat{Error}(\lambda) = \frac{1}{k}\sum_s Error(\lambda, s)$

NOTE: with our OLS exercise our "λ" indexed unique features!

Overfitting Solution

Pick λ to minimize out of sample error as given by $Error(\lambda)$!

In this way model learns the “optimal level of complexity” to maximize out of sample forecasting.

Lasso + Ridge in R

We will use the `glmnet package. cv.glmnet() allows you to call Ridge and Lasso.`

Ridge vs Lasso: If you pass alpha=0, you will get a Ridge Regression. If you pass alpha=1 (the default) you will get a Lasso Regression. Shifting alpha between 0 and 1 gets you something in between ("elastic net").

$$\min_{\beta} \sum_i (y_i - x_i\beta)^2 + \lambda(\alpha \sum_p |\beta_p| + [1-\alpha] \sum_p |\beta_p|^2)$$

By default, the function will do the estimation for a range of $\lambda's$ (you can specify this range if you like). `my_model$lambda`

We can then use k-fold cross validation on each run of the model in order to select the best one.

LASSO Algorithm

1) Specify the set of candidate features

    In R, must put them in a matrix (not dataframe)

    Still must feature engineer by hand

2) Run a `for` loop through different values of $\lambda_0, \lambda_1, \ldots, \lambda_M$

    R glmnet and cv.glmnet package will do this for you

3) Each λ_m will have a set of parameters associated with it arrived at through numerical optimization.

    The regularization term means there is not an easy way to have a standard error like with OLS

4) Choose model (e.g., coefficients and features associated with a λ_m^* that has the lowest cross validated MSE

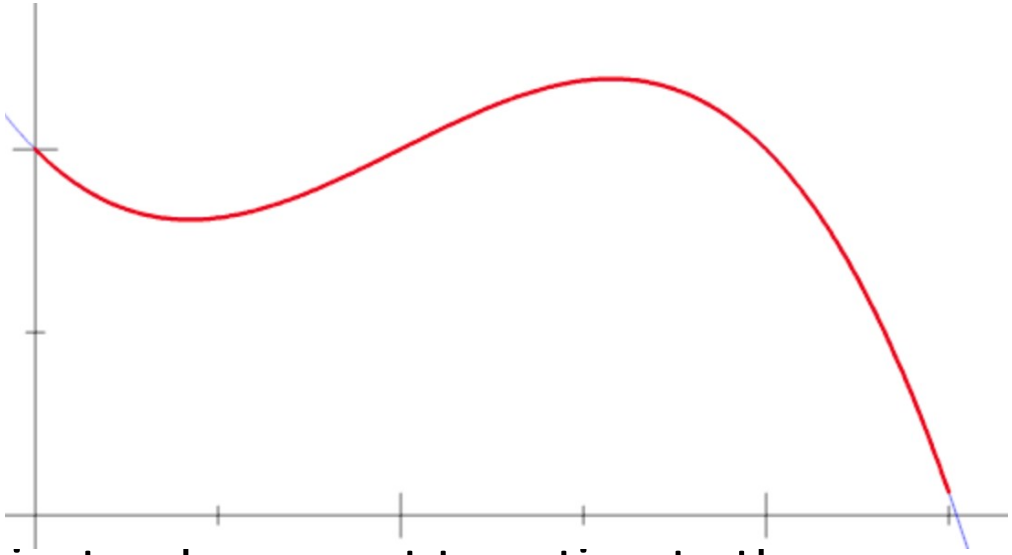
Why trees?

- Suppose we have a set of features $\{x_1, x_2, \dots, x_k\} = X$ and we want to predict outcomes $y = f(X)$.
- How do we construct $f(X)$?
  - Should we add both x_1 and x_1^2? (Non-linear features)
  - Should we add $x_1 \cdot x_2$? (Interaction features)
- How can we do this automatically?

- Trees!

Building to trees intuition

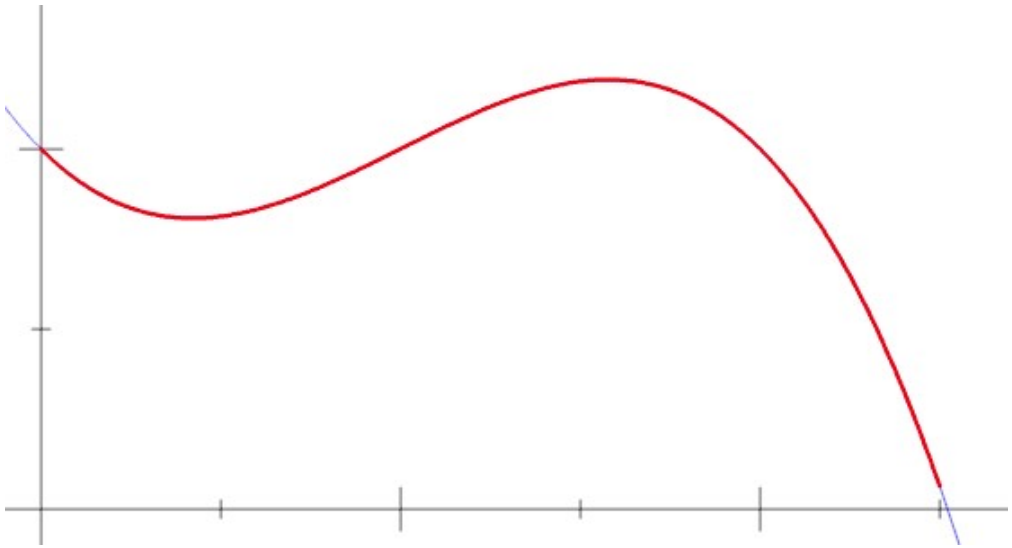
- Suppose you only had one feature x_1 and $f(x_1)$ was funky.



- Suppose instead we want to estimate the area under $f(x_1)$. What's a numerical way to do that?

Building to trees intuition(cont.)

- Split up the graph into sections and work on each individually



Regression trees

- Similar idea if we want to just predict y.
- Split the domain and do something very simple in each side.
  - If the domain has more than one feature then deals with interactions
- Lots of choices:
  - Where to split.
  - What "simple" thing to do in each side.
- What were these rules in the integral example?
- We'll talk about the CART (Classification and Regression Tree) algorithm but there are many

CART Steps

1. Pick a single feature and then a split-point to divide the data into two groups (binary splits). [Detailed next slide]
   - E.g. Split the data by whether a data point has $x_2 > 0$
2. Find the average in either side.
3. Recursively descend into each side.

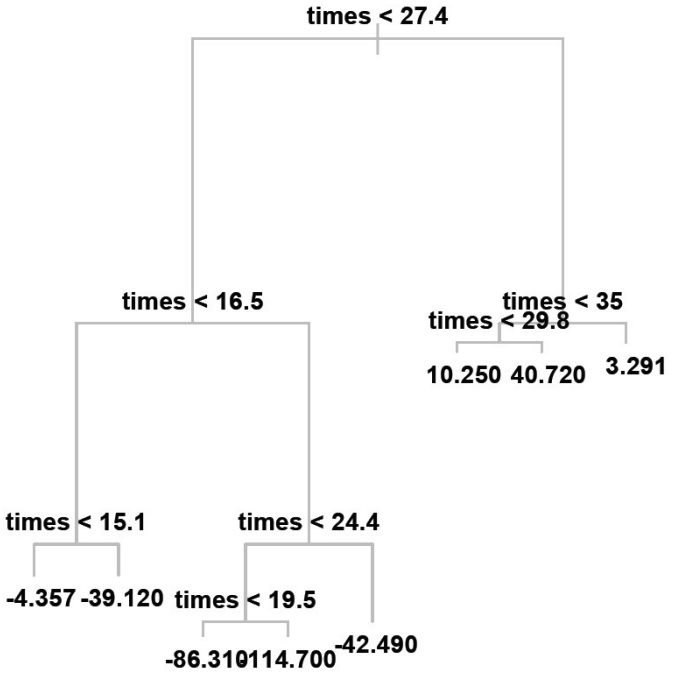
How to pick the split variable & split point?

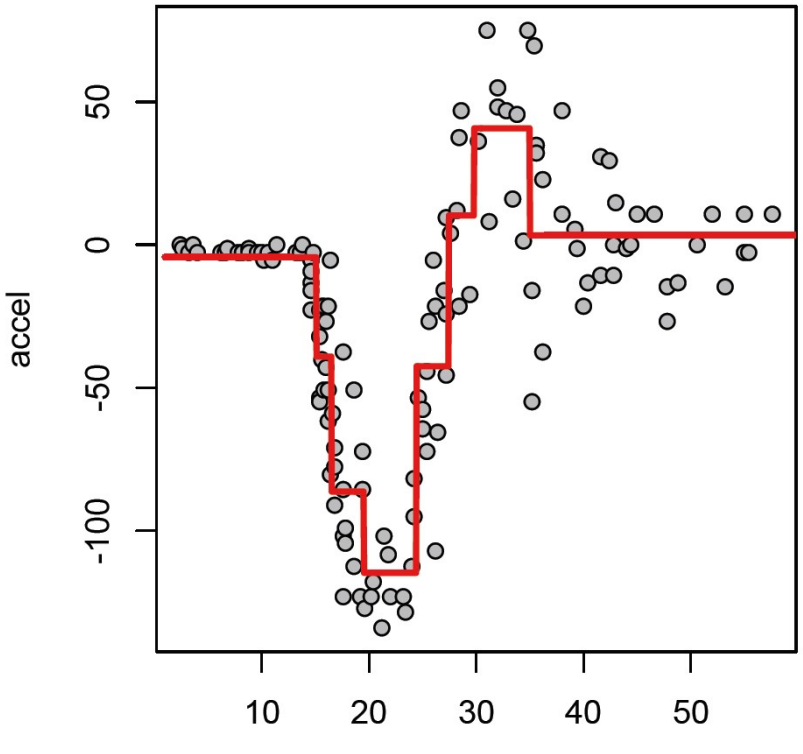
- We are approximating the outcomes of a group by its average.
- If the group has really similar outcomes, this will be good.
- Try to do the split so that this approximation is good.
- We want our approximation error is low.
- Technical: We want the mean squared error to be minimized

$$MSE = \sum_{l \in \{left, right\}} \sum_{i \in l} (y_i - \bar{y}_l)$$

Tree Graphical Example

- Motorcycle crash test data: x it time from impact, y is helmet acceleration





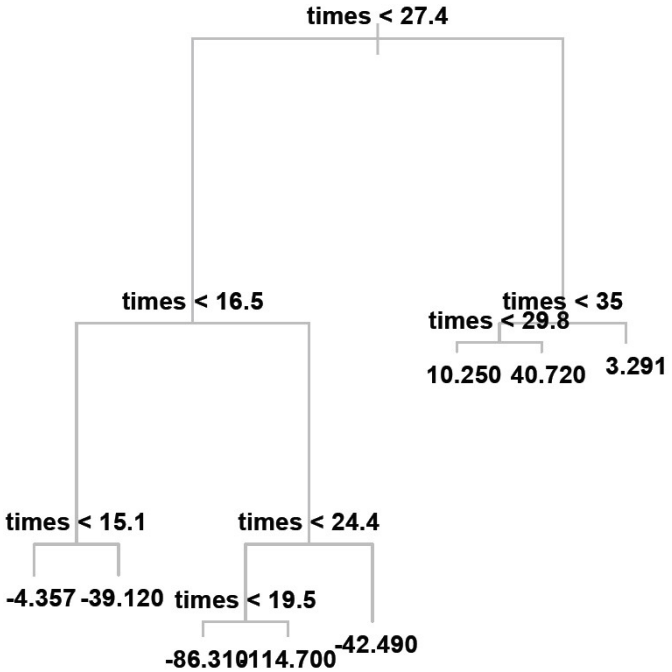
Credit: Matt Taddy

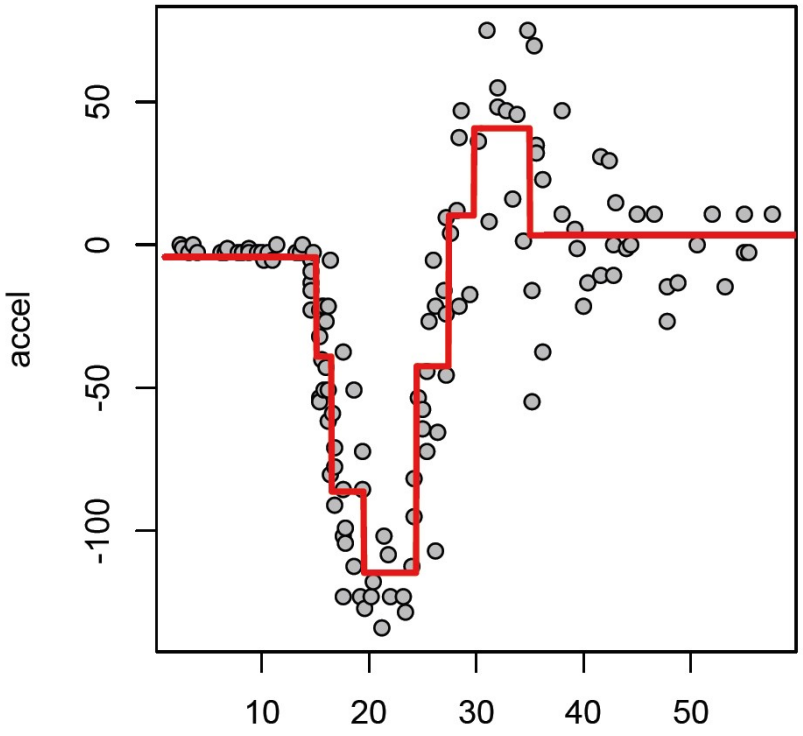
CART Example

- Suppose we wanted to find how people's *height* varied by their characteristics {*gender*, *birth-month*}.
- We want to find $height = f(gender, birthmonth)$

How is tree used for $f(x)$

- How do we put new data x' into tree to get prediction $f(x')$?





Credit: Matt Taddy

When to stop?

- Could keep splitting until you only end up with one node building a very deep/complex tree.
- Is this good?
- No, we would learn the noise of the data rather than the real relation.

CART stopping procedure

- Optimal tree complexity is a (hyper-)parameter α.
- Pick α to minimize out-of-sample prediction error:
  1. Split the data into *train* and *test* samples.
  2. Build trees of varying complex on the *train* sample.
  3. See which one does the best on the *test* sample.

Cross-Validation?

- Find the optimal value of a parameter (e.g. α) by varying it and seeing what value does best in out-of-sample prediction.
- How can we estimate OOS prediction with one set of data?
- Randomly split the sample into *train* & *test*. Estimate model on the *train* sample (using a value of α, say α_1). Estimate SPE of model on *test* sample.
  - Repeat several times and average the SPE from each to get score for α_1.
- Many ways to split the data and repeat.
- In k-fold CV you split into k folds and do this k times with one section being the hold-out, *test* sample.

4-fold Cross-Validation Example