Logit and Regularization

Jacob LaRiviere
(some content from Justin Rao and Brian Quistorff)

Outline

- Classes of explanatory Variables
  - Logit
- Bias-Variance Revisited
- Regularization
  - LASSO

Outcome types

Binary outcomes: equal to 1 or 0. Examples: coin is heads or tails, person is guilty or innocent, default or not on a loan, etc.

Continuous outcomes: can take an real value

Categorical outcomes: can take one of N qualitative values. Example: mapping symptons to a well-defined disease, favorite sports team, etc.

Traditional regression we've done is designed for continuous outcomes.

We can use classification for binary and categorical as outcomes in addition to features!

Why not linear regression?

A mapping of categories to numbers comes with strong implications

$$Y = \begin{cases} 1 & \text{if } \texttt{stroke}; \\ 2 & \text{if } \texttt{drug overdose}; \\ 3 & \text{if } \texttt{epileptic seizure}. \end{cases} \quad \text{Vs.} \quad Y = \begin{cases} 1 & \text{if } \texttt{epileptic seizure}; \\ 2 & \text{if } \texttt{stroke}; \\ 3 & \text{if } \texttt{drug overdose}. \end{cases}$$

Would imply different “differences” across qualitative conditions

Classification with binary outcomes

`Outcome: 0 or 1`

Given features X, we want to say “what is the probability the outcome=1”.

We can write this as $P(outcome = 1|X)$

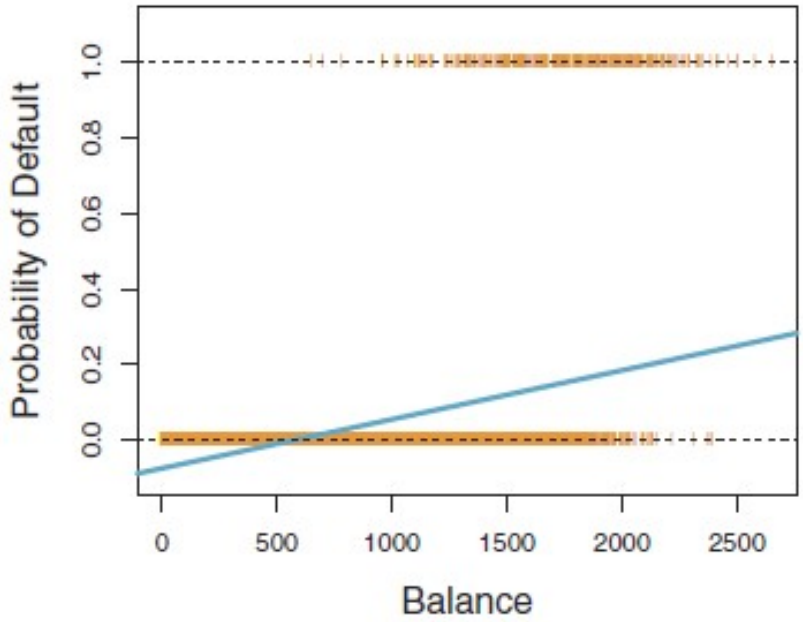
Our model will output a ***probability*** even though real outcomes will always be 0 or 1

Linear models and probabilities

Linear models will tend to make predictions outside [0,1]

Since probabilities should never be outside [0,1], this is a bad feature

We will instead use non-linear models



Logistic regression

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}.$$

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}.$$

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

Odds: ratio of probability outcome=1 to probability outcome=0

Examples

p(x)=.5. Odds=1 (or 1:1)

p(x)=.75. Odds=3 (or 3:1)

p(x)=.9. Odds=9 (or 9:1)

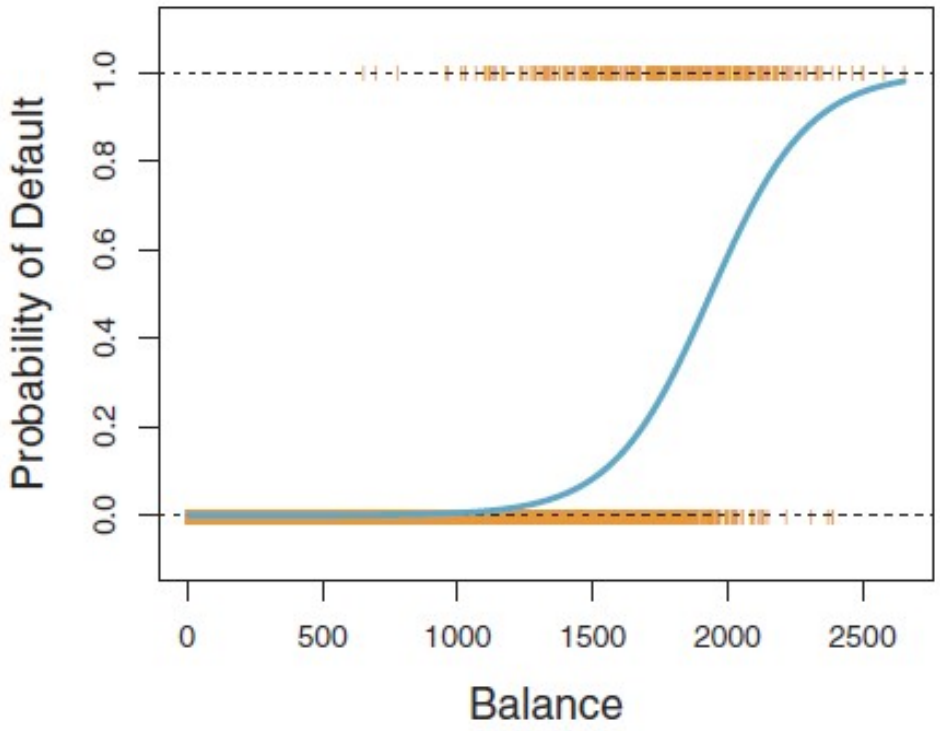
Logit says: the log of the odds is a linear function of the features.

Just like with regular regression, we can add more features

Visualizing a logistic function

The estimate parameters will make the function steeper or flatter (stretch it out) or shift it left or right

Estimation will minimized squared loss



Logistic regression output

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}.$$

	Coefficient	Std. error	Z-statistic	P-value
Intercept	−10.6513	0.3612	−29.5	<0.0001
balance	0.0055	0.0002	24.9	<0.0001

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}.$$

Intercept: -10.65 → $\frac{e^{-10.6}}{1+e^{-10.6}} = \frac{0.000024}{1+0.000024}$

Balance: $1 increase in balance multiplies odds by $e^{0.0055}$ or about 1.0055

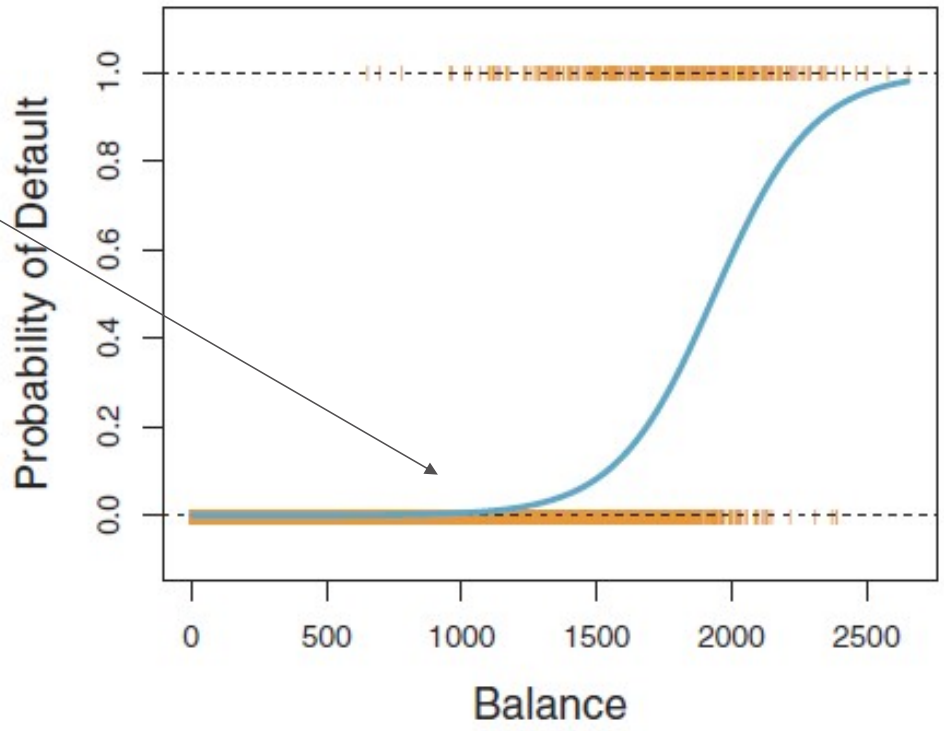
Why multiply? We are effectively multiplying both sides by e^{β_1}

Non-linear response

Increases in balance at the sat the start have a limited impact.

Similarly at the end.

In the middle region, increase are particularly important



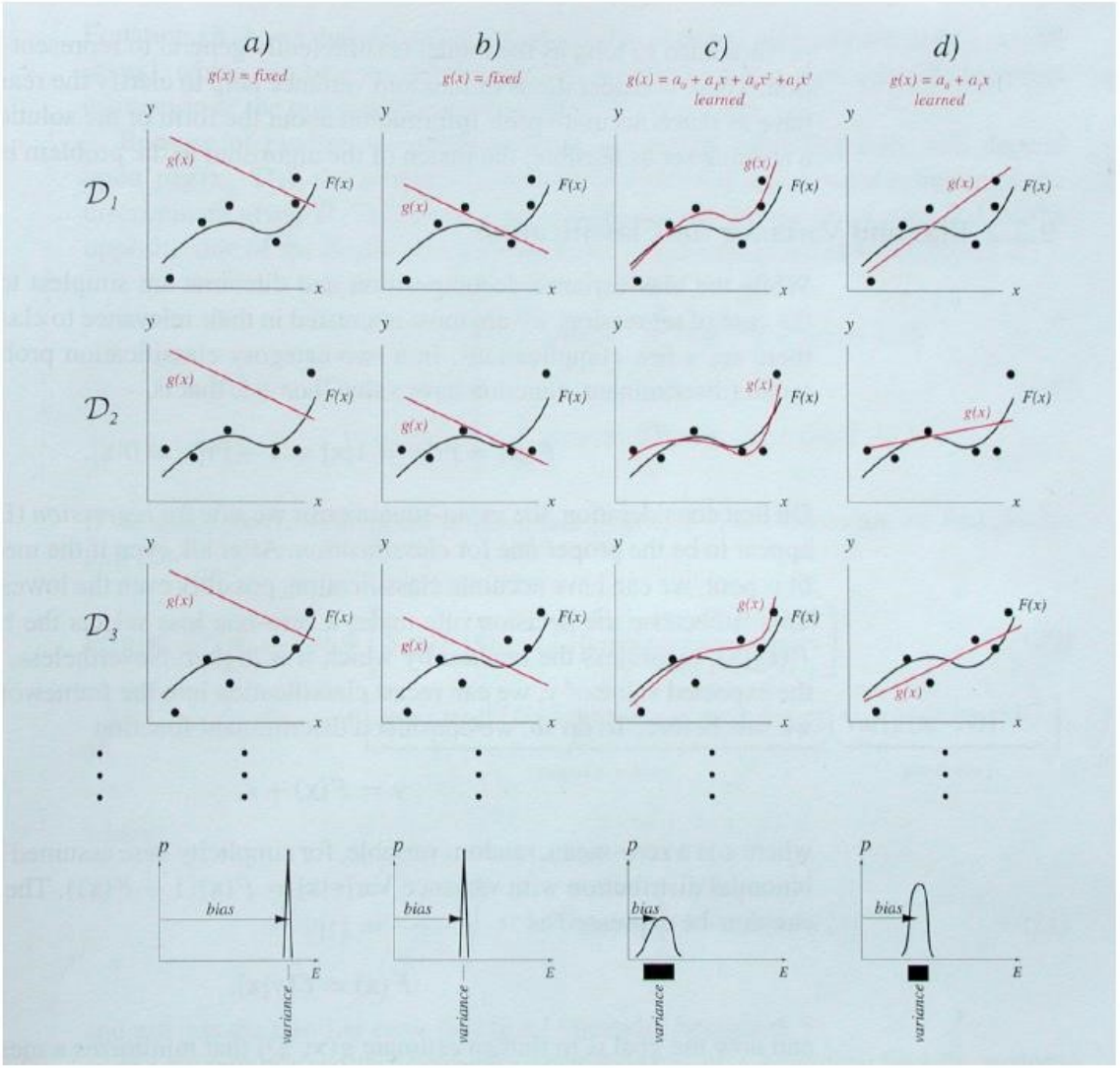
Logistic regression in R

We will use the `glm()` function. "Generalized linear models".
`family=binomial` specifies Logit

`predict(my_model, type="response")` will output the fitted probabilities

To convert this to a 0,1 prediction, we simply set a threshold of when we want to predict 1 (0.5 will minimize prediction error)

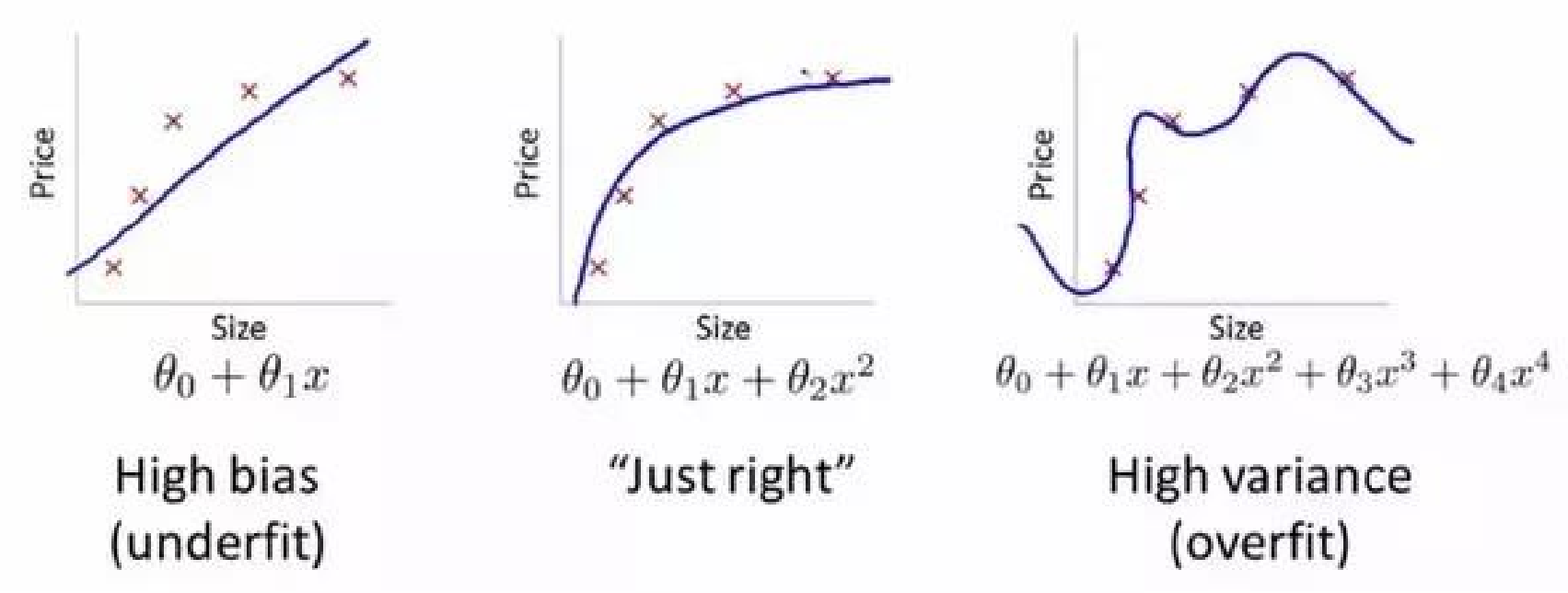
Intro to Machine Learning

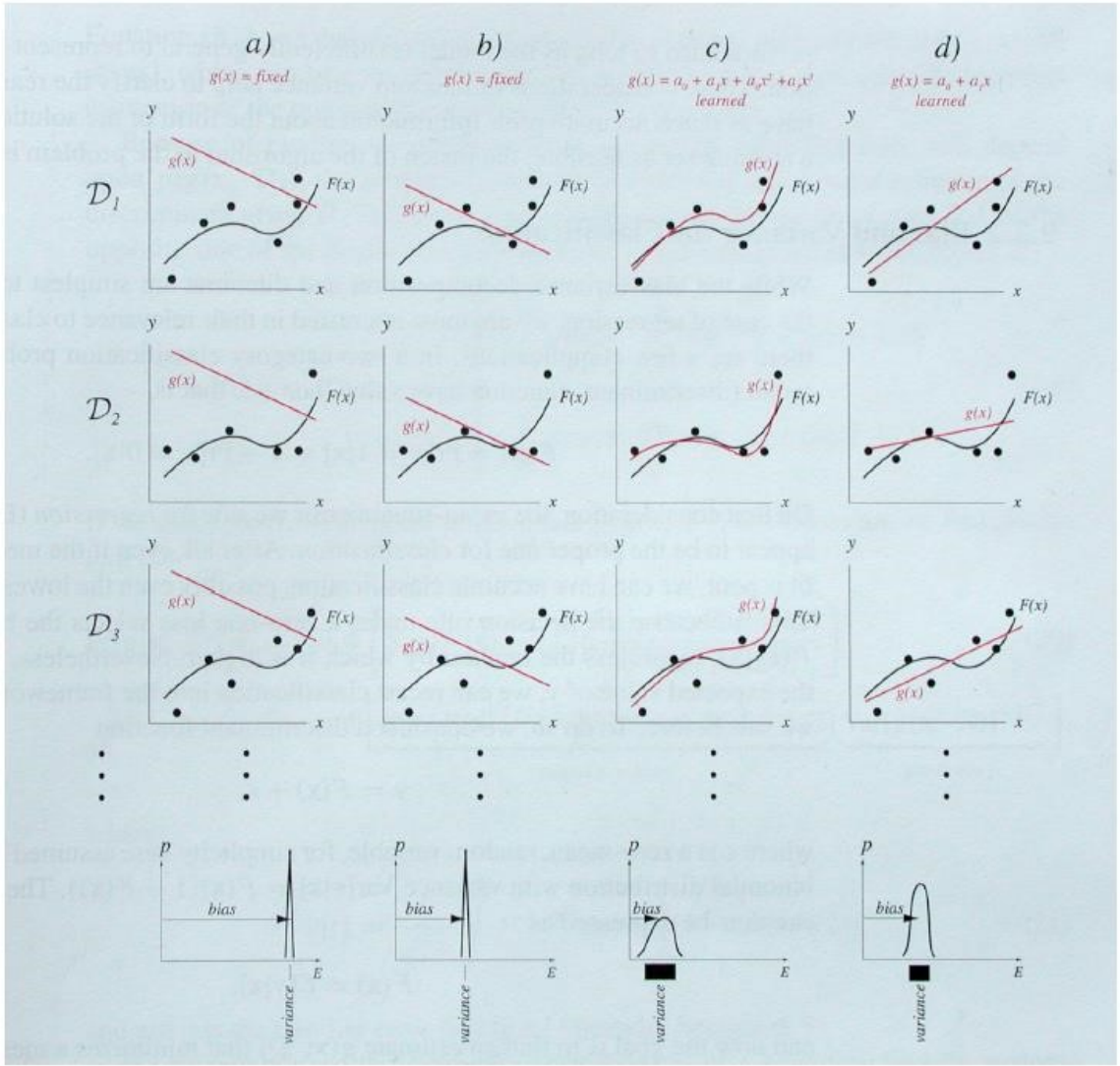


Duda, Hart, Stork "Pattern Classification", 2nd edition, 2001

In ML **bias** refers to a *model* being systematically wrong.
- Econometricians focus on *parameters* which are systematically wrong or "biased"

In ML **variance** refers to a *model* changing a lot as a function of a new sample of data.





Duda, Hart, Stork "Pattern Classification", 2nd edition, 2001

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?

Intro to Machine Learning

Estimates may be systematically “too low” or “too high” (bias) as long as good “out of sample”/”test sample” predictive properties.

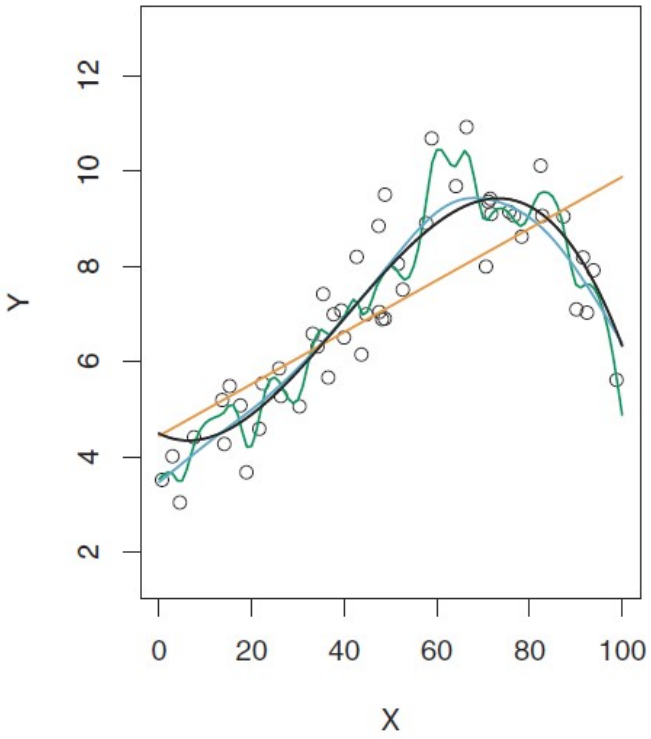
Don’t quantify uncertainty the same way. No classical *standard errors* or *confidence intervals*

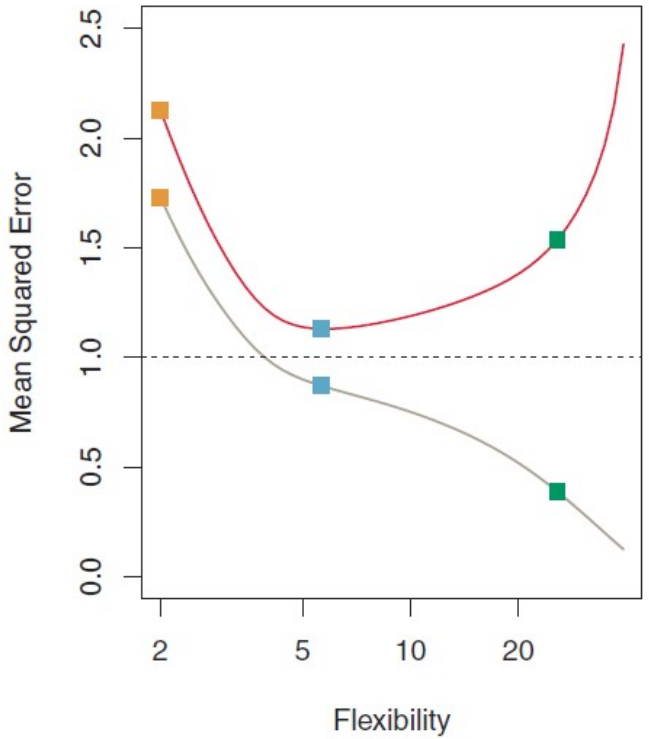
Neither OLS nor ML is “better”- use the right tool for the job!

Regularization

Key insight:

How can we algorithmically prevent overfitting like the green line?





Intro to Machine Learning

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”?

Intro to Machine Learning

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”? If the model fit improves.

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)

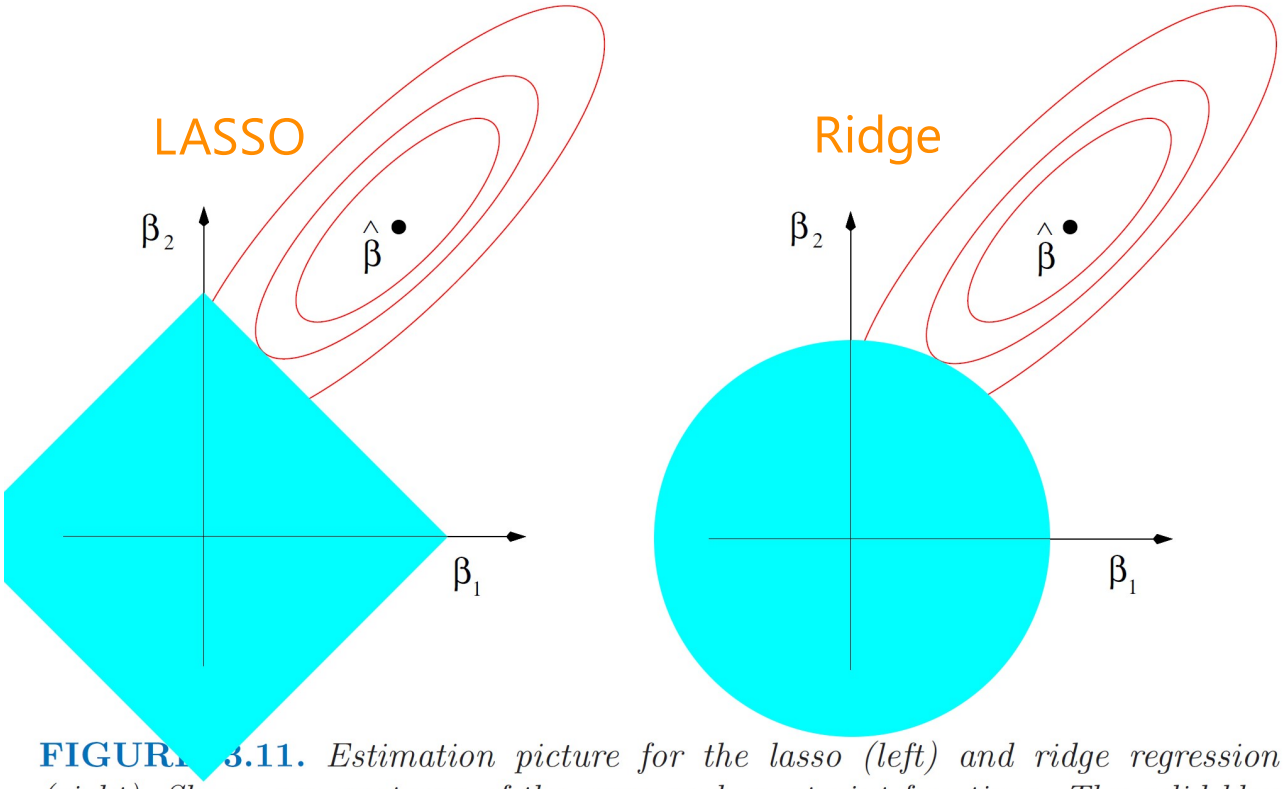


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

Coefficient Budget and Penalization

Ridge will smoothly make coefficients small

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

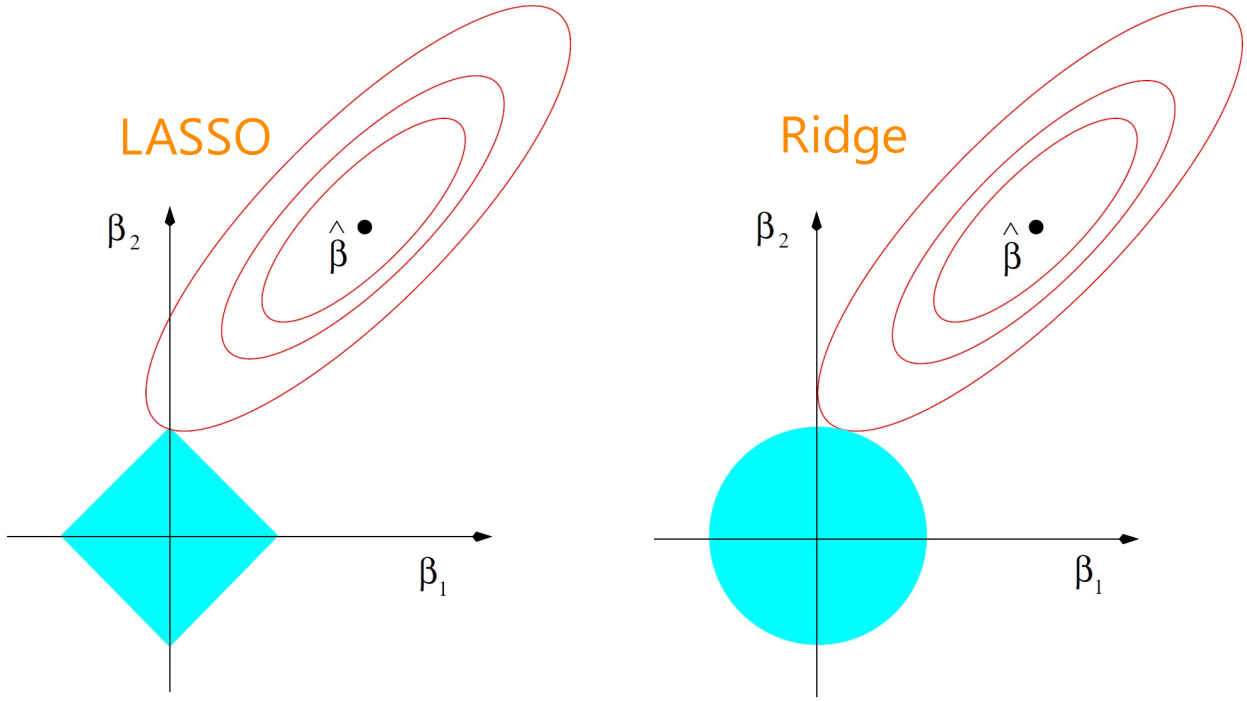


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

Overfitting

ML models are very flexible. Try to fit all patterns

Great when learning the main relationships

Can go too far and fit minor quirks in the data

Always some randomness due to sampling.

World changes so many small things in the past won't generalize to the future.

Overfitting makes model perform worse on new data

- e.g., out-of-sample, test sample, cross validated MSE

We want just the "right" amount of flexibility.

For linear models this is controlled by λ.

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 $Error(\lambda)$!

Same exact thing as our OLS exercise.

Lasso in R

We will use the `glmnet package`

`glmnet()` works for generalized linear models (OLS, Logit, etc.) or `cv.glmnet()` and have more cross validation elements automated.

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