LASSO, Elastic Net, and Cross-Validation

Gov 51: Data Analysis and Politics

Scott Cunningham

Harvard University

Week 10 Tuesday
March 31, 2026

Last week: why OLS breaks down and how to fix it

The problem:

- OLS minimizes *training* error
- More features $\rightarrow$ more overfitting
- Test error $\gg$ training error
- MSE = Bias^2 + Variance + noise

The fix:

- Add a penalty on coefficient size
- Force the model to be skeptical of the data
- Trade a little bias for a lot of stability
- Three methods: Ridge, LASSO, Elastic Net

Penalized regression constrains the coefficients OLS wants to run wild

$$\min_{\beta} \underbrace{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}_{\text{fit the data}} + \underbrace{\lambda \cdot \text{Penalty}(\beta)}_{\text{don't overfit}}$$

- $\lambda = 0$: pure OLS — no constraint, maximum overfitting
- λ large: heavy constraint — shrinks toward zero, underfits
- λ just right: the sweet spot — lowest test error

The penalty defines the method. Different penalties → different behavior.

Ridge shrinks everything toward zero — but never to zero

$$\min_{\beta} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

What it does:

- Shrinks all coefficients proportionally
- Keeps every variable in the model
- Solves instability when predictors are correlated

What it doesn't do:

- Cannot tell you *which* variables matter
- 100 predictors in $\rightarrow$ 100 small coefficients out
- No variable selection

Today: LASSO selects, Elastic Net blends, CV picks λ, and we run it on real data

1. **LASSO** — swap the L2 penalty for L1, get variable selection
2. **Elastic Net** — blend LASSO and Ridge: selection *and* stability
3. **Cross-validation** — let the data pick λ automatically
4. **AmesHousing** — predict sale prices, watch OLS fail, watch Ridge save it

LASSO (1996)

Ridge solved stability — but scientists wanted to know *which variables matter*

- Ridge keeps every variable, just with smaller coefficients
- 100 predictors in $\rightarrow$ 100 small coefficients out
- A doctor wants to know: which 5 genes actually predict this disease? Ridge can't tell you — it never drops anything
- **Robert Tibshirani** (Toronto, later Stanford) asked: what if the penalty could set coefficients to *exactly zero*?
- His insight: swap β_j^2 for $|\beta_j|$ — a tiny change in the math, a huge change in the result

Tibshirani invented LASSO at Toronto in 1996

Least **A**bsolute **S**hrinkage and **S**election **O**perator

- **Author:** Robert Tibshirani, Toronto $\rightarrow$ Stanford
- **Key move:** swap β_j^2 penalty for $|\beta_j|$ penalty
- **Result:** some coefficients shrink to **exactly zero**
- **LASSO selects**, not just shrinks

LASSO shrinks coefficients and sets some exactly to zero

$$\min_{\beta} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Penalty on the **sum of absolute values**
- As λ increases, coefficients shrink
- Some hit exactly zero and drop out
- You get a sparse model: few variables, each one meaningful

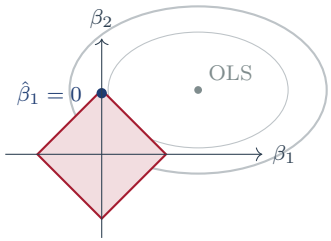
Why "L1"? The penalty measures coefficient size with absolute values

The L1 penalty:

$\lambda \sum_{j=1}^{p} |\beta_j|$

- Take the absolute value of each coefficient, add them up
- "L1" = the absolute distance from zero
- Shrinks *and* can hit exactly zero

Geometrically:



Diamond: corners on axes

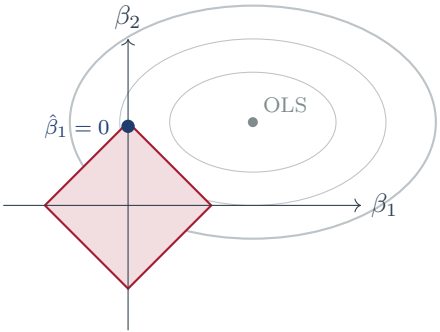
Corners → solutions land on axes → coefficients hit zero

LASSO tells you which variables the data actually needs

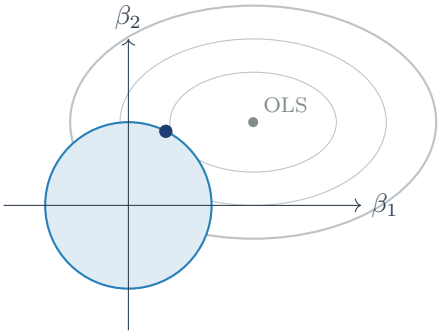
- **Prediction:** fewer noisy variables → less overfitting
- **Interpretation:** a short list of what matters
- **Honesty:** the data picks the model, not the analyst

The penalty selects variables → the analyst doesn't have to

The geometry explains why LASSO zeros out coefficients



LASSO (L1)

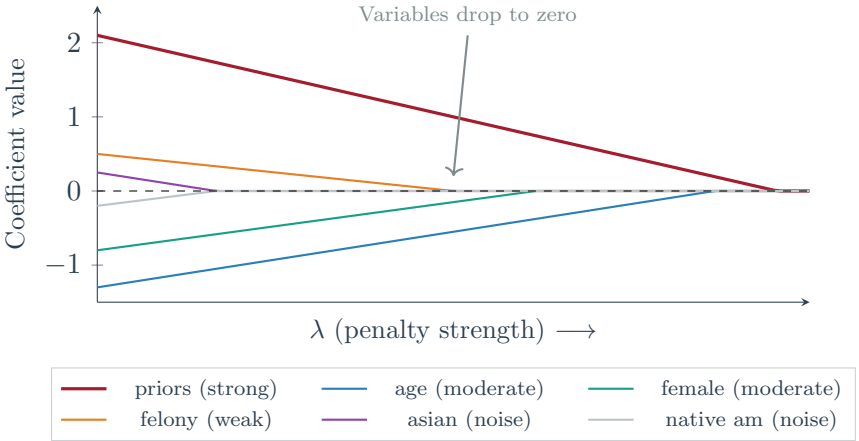


Ridge (L2)

Diamond corners → solutions land on axes → coefficients hit zero.

Circle has no corners → solutions never land exactly on an axis.

Prior record survives longest; noise variables drop out first



Elastic Net and the Full Picture

LASSO has a weakness: it's erratic when predictors are correlated

- If two predictors measure the same thing (e.g., height in inches and height in centimeters), LASSO arbitrarily keeps one and drops the other
- Run it again on a slightly different sample — it might keep the other one
- Ridge handles correlated predictors gracefully (shrinks both equally)
- **Hui Zou** (PhD student) and **Trevor Hastie** (advisor) at Stanford asked: can we get LASSO's variable selection *and* Ridge's stability?
- Their answer: blend both penalties — the "elastic net" stretches between the LASSO diamond and the Ridge circle

Elastic Net blends the LASSO and Ridge penalties

Zou & Hastie (2005)

$$\min_{\beta} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 + \lambda \Big[\alpha \sum_{j=1}^{p} |\beta_j| + (1 - \alpha) \sum_{j=1}^{p} \beta_j^2 \Big]$$

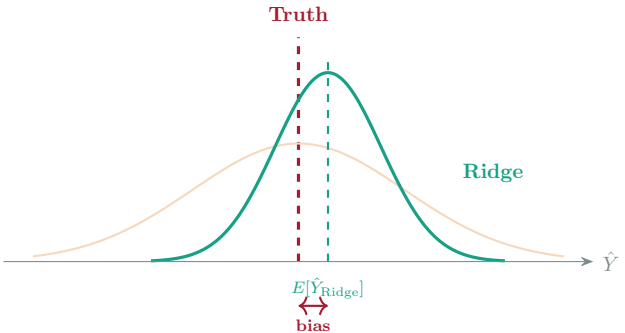
- $\alpha = 1$: pure LASSO
- $\alpha = 0$: pure Ridge
- $0 < \alpha < 1$: blend of both

Ridge, LASSO, and Elastic Net differ in how they shrink

	Ridge	LASSO	Elastic Net
Penalty	$\sum \beta_j^2$	$\sum \lvert\beta_j\rvert$	Both
Shrinks coefficients?	Yes	Yes	Yes
Sets coefficients to zero?	No	Yes	Yes
Variable selection?	No	Yes	Yes
Handles correlation?	Well	Poorly	Well

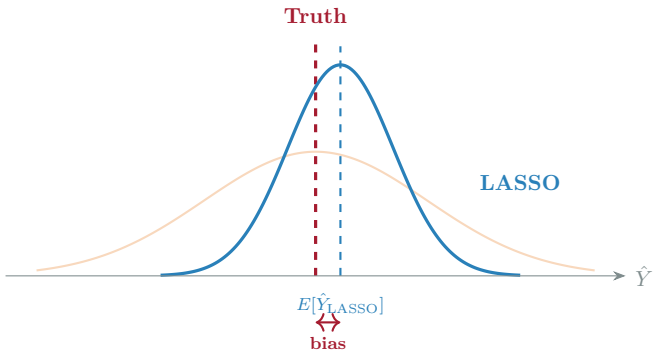
What Do These Methods Do to the Sampling Distribution?

Remember the wide OLS distribution? Here's what Ridge does to it



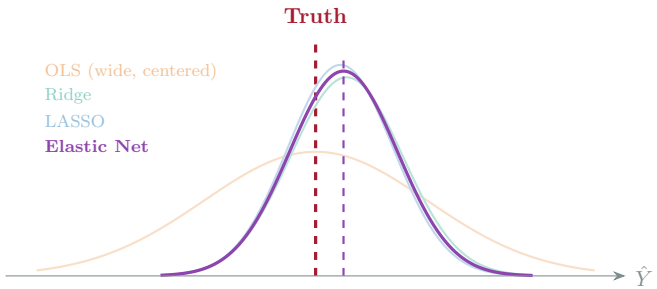
Narrower = lower variance. Small shift = small bias. Total MSE is *lower* than OLS

LASSO does the same thing — and kills noise variables



Same bias-variance tradeoff as Ridge — but LASSO also **selects variables**, setting noise predictors to exactly zero

All three penalized methods: tighter, slightly off-center



Every penalized method: *tighter and slightly off-center.*
The small bias is the price. The low variance is the payoff

To use these methods in R, we need `glmnet`

`glmnet` is the R package for all three penalized methods:

- Same function, one argument changes: `alpha`
- `alpha = 0` → Ridge `alpha = 1` → LASSO `alpha = 0.5` → Elastic Net
- `cv.glmnet()` runs cross-validation automatically to select λ

> But `glmnet` needs your data in a specific form — a numeric matrix, not a formula

That's what the next slide is about: how to build that matrix

`glmnet` needs a matrix, not a formula — here's why

`lm()` accepts a formula:

```
lm(recidivism ~ ., data = train)
```

`glmnet()` requires a numeric matrix:

```
X_train <- model.matrix(f_full, data = train)[, -1]
```

- `model.matrix()` converts the formula into columns of numbers
- `[, -1]` drops the intercept column (`glmnet` adds its own)
- With interactions: 16 variables → ∼141 columns

On PS3, you'll build this matrix once and feed it to Ridge, LASSO, and Elastic Net

What `model.matrix()` actually does — a concrete example

Data with three predictors:

```
df <- data.frame(y=c(10,15,12), x1=c(1,2,3),
                 x2=c(0,1,0),  x3=c(5,8,6))
```

`glmnet()` requires a numeric matrix — no formula allowed:

```
X <- model.matrix(y ~ x1 + x2 + x3, data=df)[, -1]
```

	x1	x2	x3
1	1	0	5
2	2	1	8
3	3	0	6

`[, -1]` drops the intercept column. What remains feeds directly into `glmnet()`

How Do We Choose λ?

Back to crime prediction: which penalty strength works best?

LASSO for rearrest prediction: 30 candidate variables from admin data (age, prior arrests, charge type, time served, . . .)

- $\lambda = 0$: keep all 30 variables $\rightarrow$ OLS, overfits
- λ small: keep 22 variables, drop 8 weak ones
- λ medium: keep 9 variables, drop 21
- λ large: keep 1 variable (prior record), drop 29 $\rightarrow$ underfits

Which λ gives the most honest prediction of who gets rearrested?

The λ dial controls how much the model trusts the data

- λ **too small** — model trusts the data too much

  It memorizes quirks of the training sample. On new defendants, predictions are noisy.

- λ **too large** — model doesn't trust the data enough

  It ignores real patterns. Predictions are stable but systematically off.

- λ **just right** — the sweet spot

We need a method to **test** each candidate λ on data the model hasn't seen

Pick any λ — here's how to honestly score it

Imagine you have 500 defendants. Fix $\lambda = 0.1$ and score it:

1. Set aside the first 100 defendants — don't touch them yet
2. Fit LASSO on the other 400, **using** $\lambda = 0.1$
3. Predict rearrest for the 100 you held out; compute RMSE
4. Put the first 100 back; hold out the *next* 100
5. Repeat: fit on 400, predict on 100, compute RMSE

After 5 rounds: average the 5 RMSEs
$\rightarrow$ one CV-RMSE score for $\lambda \quad = \quad 0.1$

That's it. One λ, one honest out-of-sample score. Now we need to find the *best* λ.

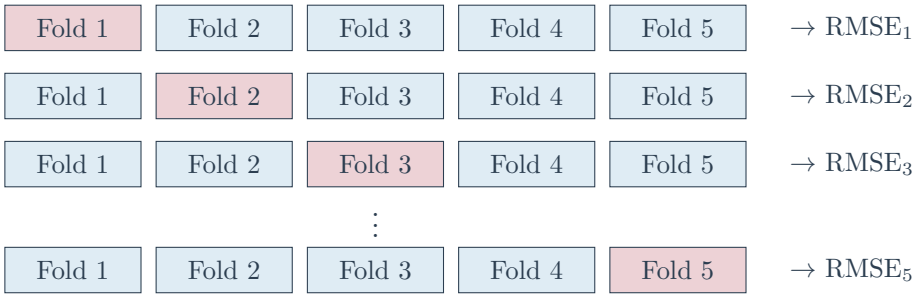
Run that procedure for every λ — the minimum wins

Take the previous slide. Run it 100 times — once per candidate λ. Each run gives one CV-RMSE. Compare:

Results across λ values:

λ	Variables kept	CV-RMSE
0 (OLS)	30	0.48
0.01	22	0.44
0.05	**9**	**0.39 ← minimize this**
0.50	3	0.43
5.00	1	0.51

k-fold cross-validation: each *fold* gets a turn as the test set



$$\text{CV-RMSE} = \tfrac{1}{5}(\text{RMSE}_1 + \cdots + \text{RMSE}_5)$$

Not jackknife: each fold is a *group*, not one observation. LOOCV ($k = n$) is the jackknife. Standard: $k = 5$ or 10.

Minimizing training error always picks $\lambda = 0$

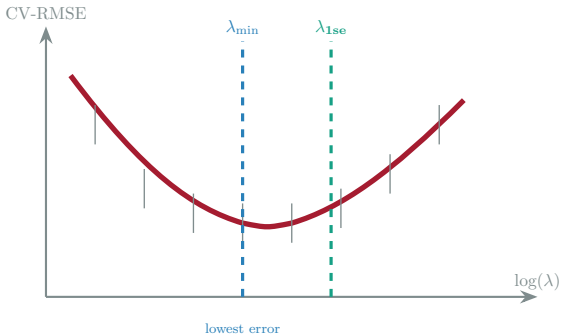
- $\lambda = 0$ means no penalty $\rightarrow$ OLS
- OLS gives the best in-sample fit by definition
- But we already know OLS overfits

Cross-validation estimates **out-of-sample** error $\Rightarrow$ optimal $\lambda > 0$

The CV-RMSE curve is the solution: it tells us which λ wins out-of-sample

The x-axis is $\log(\lambda)$ — the y-axis is out-of-sample error

λ spans many orders of magnitude — we use the log scale.
Left = small penalty (underfits). Right = large penalty (overfits).



Reading the curve:

Left: low λ, high variance
Bottom: best CV-RMSE
Right: high λ, high bias

Vertical lines:

$\lambda_{\min}$: lowest error
λ_{1se}: simpler, close to min

λ_{min} for prediction accuracy; λ_{1se} for interpretability

λ_{min} — lowest CV error

- Best raw prediction accuracy
- Keeps more variables
- Goal: minimize test RMSE

PS3: use `"lambda.min"` throughout

λ_{1se} — sparsest defensible model

- Within 1 SE of minimum
- Drops more variables
- Goal: interpretability, not just accuracy

Baker uses this to identify the smallest model that's defensible

In R: `predict(cv_fit, s = "lambda.min")` or `s = "lambda.1se"`

Application: What Predicts a House's Sale Price?

The Ames housing data: a deliberate replacement for Boston

De Cock (2011) published this dataset as a teaching alternative to the Boston housing data from the 1970s — which used a racially-coded variable as a predictor of home values.

- Covers all residential home sales in Ames, Iowa: 2006–2010
- Now the standard benchmark for regression prediction in data science
- Used in Kaggle competitions, textbooks, and methods courses worldwide
- Outcome: actual sale price (not an index) — real dollars, real decisions

If you Google "house price prediction ML" — this is the dataset you'll find

2,930 houses in Ames, Iowa — one outcome, 80 features

The outcome:

`Sale_Price` — final sale price ($)

A few predictors:

- `Gr_Liv_Area` — living area (sq ft)
- `Overall_Qual` — quality rating (1–10)
- `Year_Built` — construction year
- `Neighborhood` — 28 neighborhoods
- `Garage_Area` — garage size (sq ft)

The question:

Can regularization beat a well-specified OLS model when we let features multiply?

The experiment:

1. OLS with curated predictors
2. OLS with all pairwise interactions
3. Ridge, LASSO, Elastic Net

The goal: let features multiply, then ask regularization to sort them out

We have 34 numeric predictors. We believe the truth is complicated — house price probably depends on interactions, not just main effects.

Strategy:

1. Give OLS all the features plus their pairwise interactions → 629 columns
2. OLS will overfit badly — that's the demonstration
3. Then Ridge, LASSO, EN take the same 629 columns and let the penalty do the work

Step 1 is just mechanical: convert the formula into the numeric matrix that `glmnet` needs

Step 1: `model.matrix()` turns 34 numeric features into 666

Main effects only:

```
ols_simple <- lm(Sale_Price ~ ., data = train_s)
```

Kitchen sink — all pairwise interactions:

```
f_full <- Sale_Price ~ (x1 + x2 + ...  + x34)^2
X_train <- model.matrix(f_full, data = train)[, -1]
X_test   <- model.matrix(f_full, data = test)[, -1]
```

- 34 numeric predictors $\rightarrow \binom{34}{2} + 34 = 595 + 34 = \mathbf{629}$ columns
- Training observations: 2,344 Ratio $n/p \approx 3.5$
- OLS is now asking to estimate 629 coefficients from 2,344 observations

Now we have our matrix — time to penalize

The hard part is done. `X_train` and `X_test` are 629-column numeric matrices.

What we're about to do:

- Feed the same matrix to three penalized methods
- Each uses a different penalty to shrink / eliminate coefficients
- Cross-validation picks the best λ for each method automatically
- Then we compare: which method predicts sale price best out-of-sample?

The code is nearly identical for all three — only `alpha` changes

Steps 2–4: Ridge, LASSO, and Elastic Net take the same matrix

Ridge (`alpha = 0`): shrink all 629 coefficients, keep all features

```
cv_ridge <- cv.glmnet(X_train, y_train, alpha = 0, nfolds = 10)
```

LASSO (`alpha = 1`): shrink and eliminate

```
cv_lasso <- cv.glmnet(X_train, y_train, alpha = 1, nfolds = 10)
```

Elastic Net (`alpha = 0.5`): blend

```
cv_enet  <- cv.glmnet(X_train, y_train, alpha = 0.5, nfolds = 10)
```

Predict and evaluate:

```
predict(cv_ridge, s = "lambda.min", newx = X_test)
coef(cv_lasso, s = "lambda.min")
```

Before we run it: what do you expect?

Which model will have the lowest out-of-sample RMSE?

OLS (main effects)
Small model, well-specified
Neighborhood dummies included
Low variance, possibly high bias

OLS (kitchen sink)
629 features, $n/p = 3.5$
Rank-deficient design matrix
Probably unstable

Ridge
All 629 features, shrunk
Trades bias for variance
Should recover from kitchen sink disaster

LASSO / Elastic Net
Automatic variable selection
Will it find the signal in 629 features?

The results: regularization saves prediction from the kitchen sink

Model	Features	Test RMSE ($)	Regime
OLS (34 main effects)	34	∼34,992	Low variance
OLS (629 kitchen sink)	629	∼951,068	Variance explodes
Ridge	629	∼42,133	Variance controlled
LASSO	∼30	∼37,261	Selection + shrinkage
Elastic Net	∼30	∼36,222	Blend

Kitchen sink OLS:
30× worse than simple OLS

Elastic Net:
uses ∼30 of 629 features, beats all others

The data should pick the model, not the analyst.

A little bias buys a lot of stability.