Section 10: Ridge, LASSO, Elastic Net, and Cross-Validation

Gov 51 — Data Analysis and Politics

George (TF)

Week 10

What we're doing today

Our goal throughout this course — and in PS3 specifically — is **predictive accuracy**: given a new observation, how close can we get to the true outcome? When we shift from explaining to predicting, the bias-variance tradeoff becomes the central tension.

The bias-variance tradeoff in one line

Every model's expected prediction error decomposes as:

$$\text{MSE} = \text{Bias}^2 + \text{Variance} + \text{Irreducible noise}$$

- **Bias**: systematic error from wrong assumptions. A simple model (too few predictors) underfits — it misses real patterns in the data.
- **Variance**: sensitivity to the specific training sample. A complex model (too many predictors) overfits — it memorizes noise instead of learning signal. Out-of-sample, this kills you.

OLS minimizes in-sample residuals. That's not the same as minimizing out-of-sample error. If you throw enough features at OLS, it will eventually memorize the training data — bias goes to zero, but variance explodes.

Regularization is the fix. By penalizing large coefficients, Ridge and LASSO deliberately introduce a small amount of bias in exchange for a large reduction in variance. The net effect is better out-of-sample prediction.

Today we'll *demonstrate* this in action. We'll use the **Boston housing dataset**: 506 neighborhoods, 13 predictors, one outcome — `medv`, the median home value (in $1000s). We'll first show OLS overfitting badly (once we give it enough features), then show Ridge, LASSO, and Elastic Net recovering. Same workflow you'll use on COMPAS. Different data, so you have to do the thinking yourself on PS3.

Step 1: Load the data

```
library(glmnet)
library(MASS)

data(Boston)

# What do we have?
dim(Boston)
```

```
## [1] 506  14
```

```r
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

The outcome is `medv` (column 14). The 13 predictors include crime rate, distance to employment centers, pupil-teacher ratio, and others. One of them — `black` — is a transformation of the proportion of Black residents. We'll come back to that.

```r
# Quick look at the outcome
summary(Boston$medv)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   17.02   21.20   22.53   25.00   50.00
```

Step 2: Train/test split

Before fitting anything, we split the data. This is how we honestly evaluate predictions — fit on training data, evaluate on data the model never saw.

```r
set.seed(51)
n <- nrow(Boston)
train_idx <- sample(1:n, size = floor(0.8 * n))

train <- Boston[train_idx, ]
test  <- Boston[-train_idx, ]

cat("Training:", nrow(train), " Test:", nrow(test), "\n")
```

```
## Training: 404  Test: 102
```

Step 3: OLS baseline

Start simple. Fit OLS on all 13 predictors and see how it does in-sample vs. out-of-sample.

```r
# Fit OLS
ols <- lm(medv ~ ., data = train)

# RMSE function
rmse <- function(actual, predicted) sqrt(mean((actual - predicted)^2))

ols_train_rmse <- rmse(train$medv, predict(ols, train))
ols_test_rmse  <- rmse(test$medv,  predict(ols, test))

cat("OLS in-sample RMSE: ", round(ols_train_rmse, 2), "\n")
```

```
## OLS in-sample RMSE:  4.84
```

```r
cat("OLS out-of-sample RMSE:", round(ols_test_rmse, 2), "\n")
```

```
## OLS out-of-sample RMSE: 4.11
```

```r
cat("Gap:", round(ols_test_rmse - ols_train_rmse, 2), "\n")
```

```
## Gap: -0.73
```

With only 13 main-effect predictors and 404 training observations, OLS doesn't overfit badly here. The gap is small. This is what **low variance** looks like — the model is simple enough that it generalizes. But we haven't given it enough rope to hang itself yet. Keep this baseline; we'll revisit it once we expand the feature space.

Step 4: Deliberately overfit — demonstrating the bias-variance tradeoff

`glmnet` needs a numeric matrix, not a formula. `model.matrix()` converts the formula and handles factor encoding. The `[, -1]` drops the intercept column — `glmnet` adds its own.

But before we regularize, we need to *earn* the motivation. With just 13 predictors, OLS barely overfits. To really see the bias-variance tradeoff blow up, we need to give OLS far more parameters than it can handle.

The strategy: expand from 13 predictors to hundreds by including all three-way interactions — every combination of three predictors multiplied together. Thirteen variables taken three at a time gives us hundreds of features. With ~400 training observations and hundreds of features, the ratio n/p approaches 1. OLS will fit the training data nearly perfectly (bias ≈ 0) but will have exploding variance — it can't generalize because it's essentially memorizing noise.

This is a *controlled* demonstration of the tradeoff. We're choosing to overfit on purpose so we can then show what regularization does to fix it.

```
# Kitchen sink: all pairwise interactions + squared terms for continuous vars
f_kitchen <- medv ~ (crim + zn + indus + chas + nox + rm + age + dis +
                     rad + tax + ptratio + black + lstat)^3

X_train <- model.matrix(f_kitchen, data = train)[, -1]
X_test  <- model.matrix(f_kitchen, data = test)[, -1]

# Align columns between train and test
common_cols <- intersect(colnames(X_train), colnames(X_test))
X_train <- X_train[, common_cols]
X_test  <- X_test[,  common_cols]

y_train <- train$medv
y_test  <- test$medv

cat("Features after interactions + polynomials:", ncol(X_train), "\n")
```

```
## Features after interactions + polynomials: 377
```

```
cat("Training observations:", nrow(X_train), "\n")
```

```
## Training observations: 404
```

```
cat("Ratio n/p:", round(nrow(X_train) / ncol(X_train), 1), "\n")
```

```
## Ratio n/p: 1.1
```

Now fit kitchen sink OLS and watch what happens:

```
ols_kitchen <- lm(f_kitchen, data = train)

kitchen_train_rmse <- rmse(y_train, predict(ols_kitchen, train))
kitchen_test_rmse  <- rmse(y_test,  predict(ols_kitchen, test))

cat("Kitchen sink in-sample RMSE: ", round(kitchen_train_rmse, 2), "\n")
```

```
## Kitchen sink in-sample RMSE:  1.07
```

```
cat("Kitchen sink out-of-sample RMSE:", round(kitchen_test_rmse, 2), "\n")
```

```
## Kitchen sink out-of-sample RMSE: 123.37
```

```
cat("Gap:", round(kitchen_test_rmse - kitchen_train_rmse, 2),
    "<-- this is overfitting\n")
```

```
## Gap: 122.3 <-- this is overfitting
```

This is the bias-variance tradeoff in action:

- **In-sample RMSE near zero**: With $n/p \approx 1$, OLS finds coefficients that fit the training data almost exactly. Bias is crushed to near zero.
- **Out-of-sample RMSE is catastrophic**: All that fitting is noise memorization. The model has seen 404 examples and learned patterns that don't exist. On the 102 held-out neighborhoods, it fails completely.

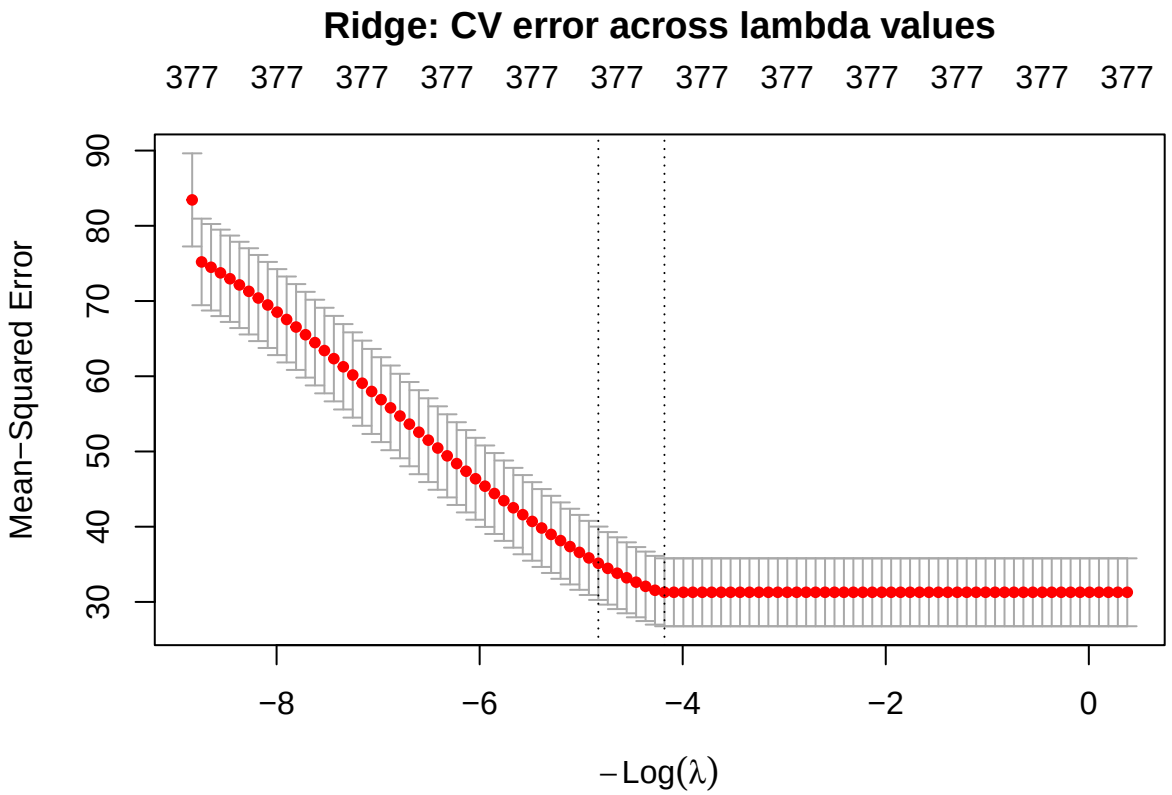
The gap between train and test RMSE *is* the variance. High variance = high sensitivity to which specific observations you happened to train on. A different random seed would give you a completely different set of coefficients, but roughly the same terrible out-of-sample performance.

This is exactly the problem regularization fixes. Instead of letting OLS minimize in-sample error without constraint, we'll add a penalty that shrinks coefficients — deliberately trading a little bias for a lot of variance reduction.

Step 5: Ridge regression

Ridge uses an L2 penalty — it shrinks all coefficients toward zero but never eliminates any. Set `alpha = 0`.

```
# Fit Ridge with cross-validation to pick lambda
set.seed(51)
cv_ridge <- cv.glmnet(X_train, y_train, alpha = 0, nfolds = 10)

# Plot the CV curve
plot(cv_ridge)
title("Ridge: CV error across lambda values", line = 2.5)
```



The x-axis is `log(lambda)`. The top axis shows the number of non-zero coefficients (always 13 for Ridge — it never drops anything). The two vertical lines mark `lambda.min` (lowest CV error) and `lambda.1se` (simplest model within 1 SE of the minimum).

```
cat("lambda.min:", round(cv_ridge$lambda.min, 4), "\n")
```

```
## lambda.min: 65.3497
```

```
cat("log(lambda.min):", round(log(cv_ridge$lambda.min), 4), "\n")
```

```
## log(lambda.min): 4.1798
```

```
# Predict and evaluate
ridge_pred <- predict(cv_ridge, s = "lambda.min", newx = X_test)
ridge_rmse <- rmse(y_test, ridge_pred)
cat("Ridge out-of-sample RMSE:", round(ridge_rmse, 2), "\n")
```

```
## Ridge out-of-sample RMSE: 5.02
```

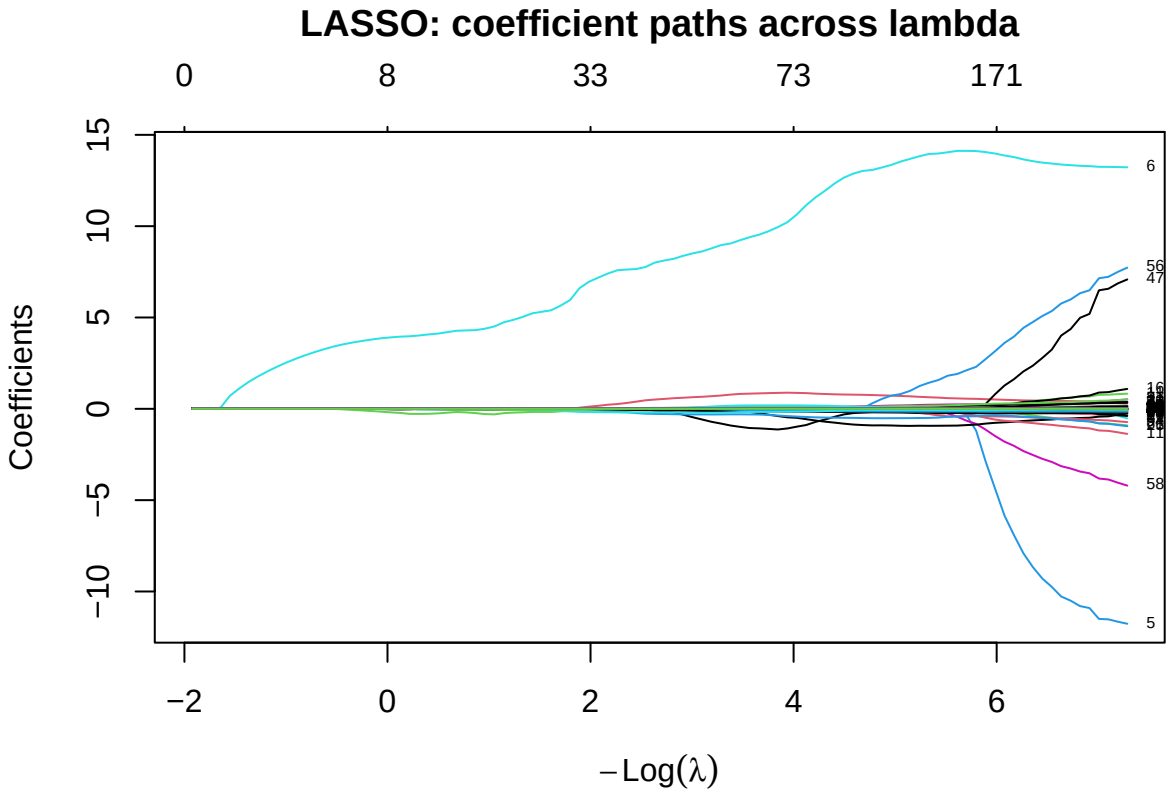
```
# Coefficients - Ridge keeps all 13
ridge_coefs <- coef(cv_ridge, s = "lambda.min")
cat("Non-zero coefficients:", sum(ridge_coefs[-1] != 0), "of", ncol(X_train), "\n")
```

```
## Non-zero coefficients: 377 of 377
```

Step 6: LASSO

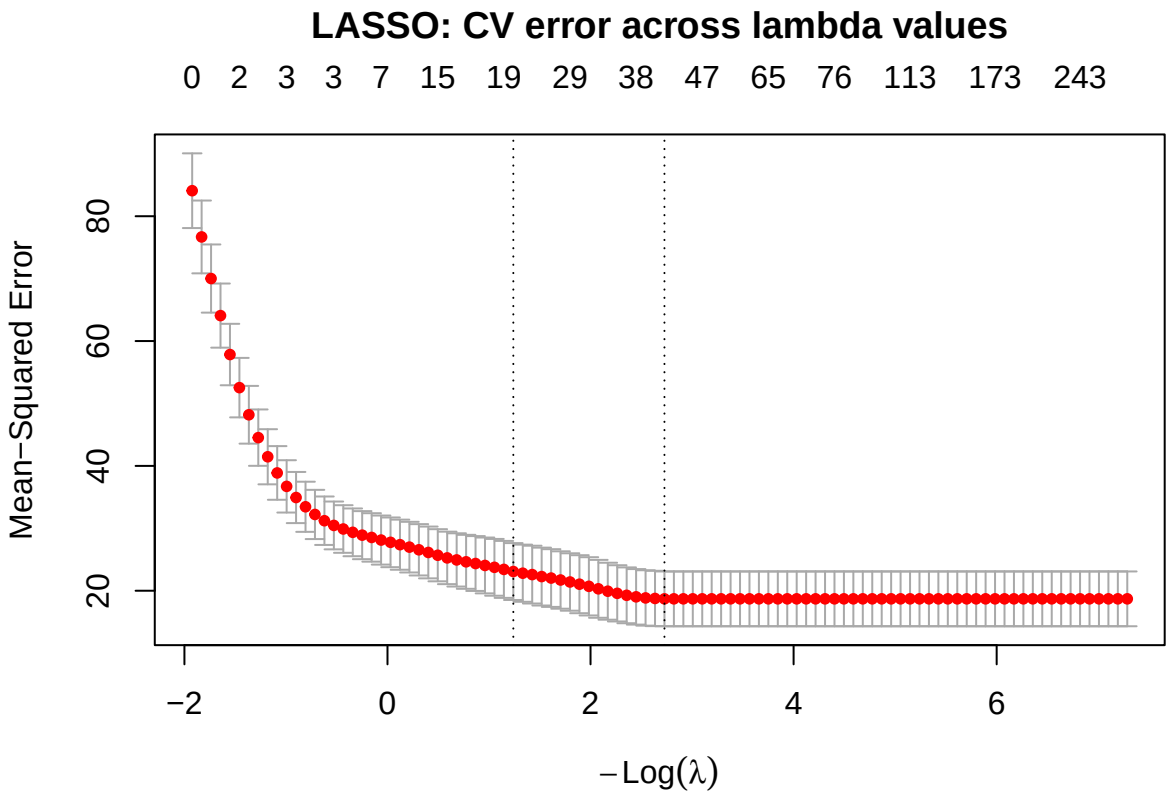
LASSO uses an L1 penalty — it shrinks AND eliminates. Set `alpha = 1`. The key difference from Ridge: some coefficients will be exactly zero.

```r
# Coefficient shrinkage path - watch variables drop to zero
la_path <- glmnet(X_train, y_train, alpha = 1)

plot(la_path, xvar = "lambda", label = TRUE)
title("LASSO: coefficient paths across lambda", line = 2.5)
```



Each line is one predictor. As lambda increases (more penalty), coefficients shrink toward zero. The weakest predictors hit zero first.

```r
# Cross-validate to find optimal lambda
set.seed(51)
cv_lasso <- cv.glmnet(X_train, y_train, alpha = 1, nfolds = 10)

plot(cv_lasso)
title("LASSO: CV error across lambda values", line = 2.5)
```




```r
cat("lambda.min:", round(cv_lasso$lambda.min, 4), "\n")
```

```
## lambda.min: 0.0653
```

```r
cat("log(lambda.min):", round(log(cv_lasso$lambda.min), 4), "\n")
```

```
## log(lambda.min): -2.728
```

```r
# Predict and evaluate
lasso_pred <- predict(cv_lasso, s = "lambda.min", newx = X_test)
lasso_rmse <- rmse(y_test, lasso_pred)
cat("LASSO out-of-sample RMSE:", round(lasso_rmse, 2), "\n")
```

```
## LASSO out-of-sample RMSE: 3.97
```

```r
# Which variables survived?
lasso_coefs <- coef(cv_lasso, s = "lambda.min")
n_kept <- sum(lasso_coefs[-1] != 0)
cat("Variables kept:", n_kept, "of", ncol(X_train), "\n\n")
```

```
## Variables kept: 41 of 377
```

```r
survivors <- rownames(lasso_coefs)[lasso_coefs[, 1] != 0]
survivors <- survivors[survivors != "(Intercept)"]
cat("Surviving variables:\n")
```

```
## Surviving variables:
```

```r
print(round(lasso_coefs[survivors, 1], 3))
```

```
##                rm               dis               rad       crim:lstat
##             8.122            -0.130             0.543             0.001
##          zn:lstat         indus:age         nox:lstat        rm:ptratio
```

```
##             0.000             0.001             0.000            -0.052
##          rm:black          rm:lstat          dis:tax     ptratio:lstat
##             0.002            -0.002            -0.001             0.007
##       crim:zn:dis  crim:indus:lstat     crim:chas:age     crim:chas:rad
##             0.054             0.000             0.028             0.047
##       crim:nox:rm       crim:rm:dis     crim:rm:black      zn:indus:rad
##            -0.020            -0.007             0.000             0.000
##  zn:indus:ptratio     zn:chas:lstat         zn:nox:rm      zn:rad:lstat
##             0.000             0.000             0.006             0.000
##    zn:black:lstat      indus:rm:dis     indus:age:tax indus:dis:ptratio
##             0.000            -0.014             0.000            -0.001
##   indus:dis:lstat   indus:tax:black    chas:nox:lstat       chas:rm:rad
##             0.002             0.000            -0.286            -0.094
##    chas:dis:black    chas:dis:lstat    nox:rm:ptratio      rm:rad:lstat
##             0.002             0.029            -0.191            -0.002
##    rm:tax:ptratio      rm:tax:lstat    rm:black:lstat       age:dis:tax
##             0.000             0.000             0.000             0.000
##     age:dis:lstat
##             0.000
```

Notice: LASSO doesn't just shrink — it selects. The variables that survive tell you what the data says actually predicts home values.

Discussion: Does `black` (racial composition) survive LASSO selection? What does it mean for a prediction algorithm if it does?

Step 7: Elastic Net

Elastic Net blends both penalties. `alpha = 0.5` means equal weight on L1 and L2. It handles correlated predictors better than pure LASSO.

```
set.seed(51)
cv_enet <- cv.glmnet(X_train, y_train, alpha = 0.5, nfolds = 10)

enet_pred <- predict(cv_enet, s = "lambda.min", newx = X_test)
enet_rmse <- rmse(y_test, enet_pred)

enet_coefs <- coef(cv_enet, s = "lambda.min")
enet_kept <- sum(enet_coefs[-1] != 0)

cat("Elastic Net RMSE:", round(enet_rmse, 2), "\n")
```

```
## Elastic Net RMSE: 4.04
```

```
cat("Variables kept:", enet_kept, "of", ncol(X_train), "\n")
```

```
## Variables kept: 55 of 377
```

Step 8: Comparison table

```
results <- data.frame(
  Model        = c("OLS", "Ridge", "LASSO", "Elastic Net"),
  Predictors   = c(ncol(X_train),
```

```
                  ncol(X_train),
                  n_kept,
                  enet_kept),
  Test_RMSE    = round(c(ols_test_rmse,
                         ridge_rmse,
                         lasso_rmse,
                         enet_rmse), 3)
)

print(results)
```

```
##         Model Predictors Test_RMSE
## 1         OLS        377     4.112
## 2       Ridge        377     5.021
## 3       LASSO         41     3.968
## 4 Elastic Net         55     4.042
```

What this means for PS3

On PS3 you'll do exactly this workflow — but on the COMPAS dataset predicting recidivism, and with a much larger feature matrix (main effects plus interactions). A few things to keep in mind:

- **Your lambda values will be different.** With many more features, `lambda.min` will be smaller, and `log(lambda.min)` will be negative. That's normal — it just means the optimal penalty is less than 1.
- **Your CV plot will look similar** — same U-shape, same two vertical lines, same interpretation. Don't panic if the x-axis is entirely negative.
- **LASSO will eliminate many more variables** — with 141 features, it might keep only 10-30. That's the point.
- **Use `s = "lambda.min"` throughout** for PS3 unless the question says otherwise.

The `black` variable — a preview of Thursday

One of the 13 Boston predictors is `black`, a transformation of the proportion of Black residents in the neighborhood. It was included in the original 1978 paper because the authors believed racial composition affected home values.

Whether LASSO keeps or drops it isn't an answer to whether it *should* be in the model. This is the same tension you'll examine on PS3 Q6.5 with COMPAS: the algorithm doesn't know or care about fairness — it just optimizes prediction. Who decides what goes in, and who bears the cost when the algorithm is wrong?