Lecture 12

Lasso regression

Julian Reif
Fall 2025

RStudio setup for this lecture

- Log into RStudio on your Amazon EC2 instance
  - Use AMI `FIN550-RStudio` with IAM role `BigDataEC2Role`

```
# Enter this command via RStudio Terminal
aws s3 cp --recursive s3://bigdata-fin550-reif/lecture-12 ~/fin550/lecture-12
```

Lasso regression theory

Recall: multiple linear regression

Ordinary least squares regression finds $\hat{\beta}$ parameters that minimize RSS:

$$
\begin{aligned}
RSS &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\
&= \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_j \right)^2
\end{aligned}
$$

Lasso regression includes an additional penalty term

Lasso regression finds $\hat{\beta}$ parameters that minimize:

$$\underbrace{\sum_{i=1}^{n}\left(y_i - \hat{\beta}_0 - \sum_{j=1}^{p}\hat{\beta}_j x_j\right)^2}_{RSS} + \underbrace{\lambda\sum_{j=1}^{p}|\hat{\beta}_j|}_{\text{penalty term}}$$

where $|\hat{\beta}_j|$ is the absolute value of β_j

- $\lambda \geq 0$ is called the "tuning parameter" (or "penalty parameter")
  - When $\lambda = 0$, lasso is identical to ordinary linear regression
  - As $\lambda \to \infty$, lasso estimates of $\hat{\beta}$ approach 0 (why?)
  - λ is chosen using cross-validation

Lasso algorithm

1. Consider a large set of different λ's: $\lambda_1, \lambda_2, \ldots \lambda_k$

2. For each possible λ, estimate $\hat{\beta}$ by minimizing $RSS + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|$

3. For each model (λ) from step 2, calculate the **cross-validated** mean-squared error

4. Select the model with the smallest cross-validated mean-squared error

- Note: Step 2 requires numerical methods, because there is no closed-form solution

Lasso is a shrinkage estimator

- Lasso regression finds $\hat{\beta}$ parameters that minimize $RSS + \lambda \sum_{j=1}^{p} |\hat{\beta}_j|$

- The penalty term encourages lasso to "shrink" $\hat{\beta}$ towards zero

- Lasso feature: coefficients for variables with no predictive power are shrunk to **exactly** 0
  - Thus, lasso can be used as a variable selection method!

What is lasso used for?

1. Predictive modeling
    - Lasso may outperform ordinary regression in terms of mean-squared error

2. Variable selection
    - Use lasso to identify relevant variables (i.e., those with non-zero coefficients)
    - Include those variables in whatever model you are using

Selecting lambda involves a bias-variance tradeoff

$$MSE = \underbrace{E\left[\epsilon^2\right]}_{\text{Noise}} + \underbrace{\left(f - E[\hat{f}\,]\right)^2}_{\text{Bias squared}} + \underbrace{Var(\hat{f}\,)}_{\text{Variance}}$$

- Ordinary linear regression $(\lambda = 0)$ generally has low bias
- Larger values of λ increase bias, but reduce variance
  - (Why? Consider the extreme case of $\lambda \to \infty$)
- Optimal value of λ balances these two effects, producing smallest possible MSE

Other shrinkage estimators

- Changing the form of the penalty term will produce different shrinkage estimators

- For example, ridge regression uses a squared penalty instead of absolute value:

$$RSS + \lambda \sum_{j=1}^{p} \left(\hat{\beta}_j{}^2 \right)$$

- However, only lasso can shrink estimates of $\hat{\beta}_j$ all the way to exactly zero
  - Thus, lasso is the only shrinkage estimator used for variable selection

Lasso regression in R

Try it: simulated data

```r
library(tidyverse)

set.seed(1)
nrows <- 100

# Create 50 random variables (X1...X50), each distributed Normal(0,1)
sim_data <- data.frame(replicate(50, rnorm(nrows,0,1) ))

# Simulated model: only two X vars are predictive of Y
# Y = 12 + 3*X1 + 4*X2 + error
sim_data$y <- 12 + 3*sim_data$X1 + 4*sim_data$X2 + rnorm(nrows, mean=0, sd=1)
```

Inspect dataset

```
head(sim_data)
```

```
#           X1          X2         X3         X4         X5          X6
# 1 -0.6264538 -0.62036668  0.4094018  0.8936737  1.0744410  0.07730312
# 2  0.1836433  0.04211587  1.6888733 -1.0472981  1.8956548 -0.29686864
# 3 -0.8356286 -0.91092165  1.5865884  1.9713374 -0.6029973 -1.18324224
# 4  1.5952808  0.15802877 -0.3309078 -0.3836321 -0.3908678  0.01129269
# 5  0.3295078 -0.65458464 -2.2852355  1.6541453 -0.4162220  0.99160104
# 6 -0.8204684  1.76728727  2.4976616  1.5122127 -0.3756574  1.59396745
#           X7          X8          X9        X10         X11        X12
# 1 -0.3410670 -0.70756823 -1.08690882 -1.5414026  1.13496509  0.2418959
# 2  1.5024245  1.97157201 -1.82608301  0.1943211  1.11193185 -1.1327594
# 3  0.5283077 -0.08999868  0.99528181  0.2644225 -0.87077763  1.4899074
# 4  0.5421914 -0.01401725 -0.01186178 -1.1187352  0.21073159 -0.2482471
# 5 -0.1366734 -1.12345694 -0.59962839  0.6509530  0.06939565  0.1835837
# 6 -1.1367339 -1.34413012 -0.17794799 -1.0329002 -1.66264885  0.4048710
#          X13        X14        X15        X16         X17        X18         X19
# 1 -1.5570357  0.3412484  1.5468813  0.8500435  0.34419403  1.6212029  0.7140855
# 2  1.9231637  1.3161672  0.1789210 -0.9253130  0.01271984 -0.3291028  0.5813846
```

What model should we choose?

- Suppose we did not know the form of $f(X) = 12 + 3X_1 + 4X_2$
  - How should we choose the right model?

- We could search across models and select variables using cross-validation

- Example: consider three models
  1. $Y = \beta_1 X_1 + \epsilon$
  2. $Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon$
  3. $Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots + \beta_{50} X_{50} + \epsilon$

```
lm1  <- glm(y ~ X1, data = sim_data)
lm2  <- glm(y ~ X1 + X2, data = sim_data)
lm50  <- glm(y ~ ., data = sim_data)
```

Evaluate model performance

- Which model has the lowest cross-validated mean-squared error?

```
library(boot)

cv.glm(data = sim_data, glmfit = lm1)$delta[1]
cv.glm(data = sim_data, glmfit = lm2)$delta[1]
cv.glm(data = sim_data, glmfit = lm50)$delta[1]

lm2$formula
```

```
# [1] 16.64053
# [1] 1.028916
# [1] 1.590093
# y ~ X1 + X2
```

How do we know if we found the right model?

- With 50 predictors, there are many other models we could consider (how many?)
- We could try an intelligent search procedure, e.g., forward stepwise selection
- Or, we could use lasso regression

Estimate lasso using the glmnet package

- `glmnet` uses vectors and matrices, not data frames or tibbles
  - Outcome variable, Y, must be a vector
  - X variables must be a matrix

- Key `glmnet` functions:
  - `mylasso <- cv.glmnet()`: estimate lasso model
  - `mylasso$lambda`: vector of λ values
  - `mylasso$cvm`: CV MSE corresponding to each λ
  - `coef(mylasso, s = #)`: coefficients corresponding to $\lambda = \#$
  - `mylasso$lambda.min`: optimal λ value (smallest CV MSE)

Try it: prepare data for lasso regression

```
library(glmnet)

# y vector
y <- sim_data$y

# x matrix with all X variables
f <- formula(y ~ 0 + .) # exclude intercept (`glmnet` will include one for us)
x <- model.matrix(f, data = sim_data)
head(x)
```

Try it: run lasso regression and display lambdas

```
# Estimate lasso regression
cvfit <- cv.glmnet(x=x, y=y)

# Different lambdas considered by `cv.glmnet()` are stored in vector `cvfit$lambda`
length(cvfit$lambda)
cat("\n")
cvfit$lambda
```

```
# [1] 82
#
#  [1] 3.883250077 3.538272843 3.223942435 2.937536275 2.676573648 2.438794222
#  [7] 2.222138465 2.024729808 1.844858393 1.680966258 1.531633848 1.395567719
# [13] 1.271589329 1.158624837 1.055695800 0.961910695 0.876457200 0.798595158
# [19] 0.727650165 0.663007730 0.604107951 0.550440665 0.501541034 0.456985510
# [25] 0.416388177 0.379397399 0.345692780 0.314982386 0.287000219 0.261503911
# [31] 0.238272624 0.217105141 0.197818118 0.180244502 0.164232077 0.149642152
```

Try it: inspect the optimal lambda

```
# Two definitions of optimal lambda: "min" and "1se". We will always use "min"
cvfit
cat("\n")
cvfit$lambda.min
```

```
#
# Call:  cv.glmnet(x = x, y = y)
#
# Measure: Mean-Squared Error
#
#     Lambda Index Measure     SE Nonzero
# min 0.1802    34   1.131 0.1152       4
# 1se 0.3150    28   1.231 0.1122       2
#
# [1] 0.1802445
```

Try it: consider three extremes for lambda

- Usually we are only interested in the optimal lambda
- To learn about lasso, we will also inspect other lambdas

```
lambda.largest <- max(cvfit$lambda)
lambda.smallest <- min(cvfit$lambda)
lambda.best <- cvfit$lambda.min # lambda with the smallest cross-validated MSE

lambda.largest
lambda.smallest
lambda.best
```

```
# [1] 3.88325
# [1] 0.002072374
# [1] 0.1802445
```

Try it: coefficient estimates for largest lambda

```
# How many coefficients are non-zero? Why?
coef.largest <- coef(cvfit,
                     s = lambda.largest)

cat("Number of non-zero coefficients:",
    sum(coef.largest != 0), "\n\n")
coef.largest
```

```
# Number of non-zero coefficients: 1
#
# 51 x 1 sparse Matrix of class "dgCMatrix"
#              s=3.88325
# (Intercept)  12.06483
# X1            .
# X2            .
# X3            .
# X4            .
# X5            .
# X6            .
# X7            .
# X8            .
# X9            .
# X10           .
```

Try it: coefficient estimates for smallest lambda

```
# How many coefficients are non-zero? Why?
coef.smallest <-


cat("Number of non-zero coefficients:",
    sum(coef.smallest != 0), "\n\n")
coef.smallest
```

Coefficient estimates for smallest lambda

```
# How many coefficients are non-zero? Why?
coef.smallest <- coef(cvfit,
                      s = lambda.smallest)

cat("Number of non-zero coefficients:",
    sum(coef.smallest != 0), "\n\n")
coef.smallest
```

```
# Number of non-zero coefficients: 49
#
# 51 x 1 sparse Matrix of class "dgCMatrix"
#                s=0.002072374
# (Intercept)  11.919761223
# X1            2.964691881
# X2            4.056783688
# X3           -0.205273505
# X4            0.173650992
# X5           -0.025635757
# X6            0.218079545
# X7           -0.054009672
# X8            0.203237828
# X9           -0.035735662
# X10          -0.284276837
```

Try it: coefficient estimates for optimal lambda

```
# Which coefficients are non-zero?
coef.best <-

cat("Number of non-zero coefficients:",
  sum(coef.best != 0), "\n\n")
coef.best
```

Coefficient estimates for optimal lambda

```r
# Which coefficients are non-zero?
coef.best <- coef(cvfit, s = lambda.best)

cat("Number of non-zero coefficients:",
  sum(coef.best != 0), "\n\n")
print(coef.best[coef.best@i + 1,
                , drop = FALSE])
```

```
# Number of non-zero coefficients: 5
#
# 5 x 1 sparse Matrix of class "dgCMatrix"
#                s=0.1802445
# (Intercept) 11.92193420
# X1           2.69459325
# X2           3.88638756
# X15         -0.01073199
# X50          0.02640677
```

Display cross-validated MSE (cvm) for each lambda

```
cvfit$cvm[which.max(cvfit$lambda)]     # MSE for largest lambda
cvfit$cvm[which.min(cvfit$lambda)]     # MSE for smallest lambda
cat("\n")

min(cvfit$cvm)                         # Minimum MSE
cvfit$cvm[cvfit$lambda == lambda.best] # Minimum MSE corresponds to the best lambda
```

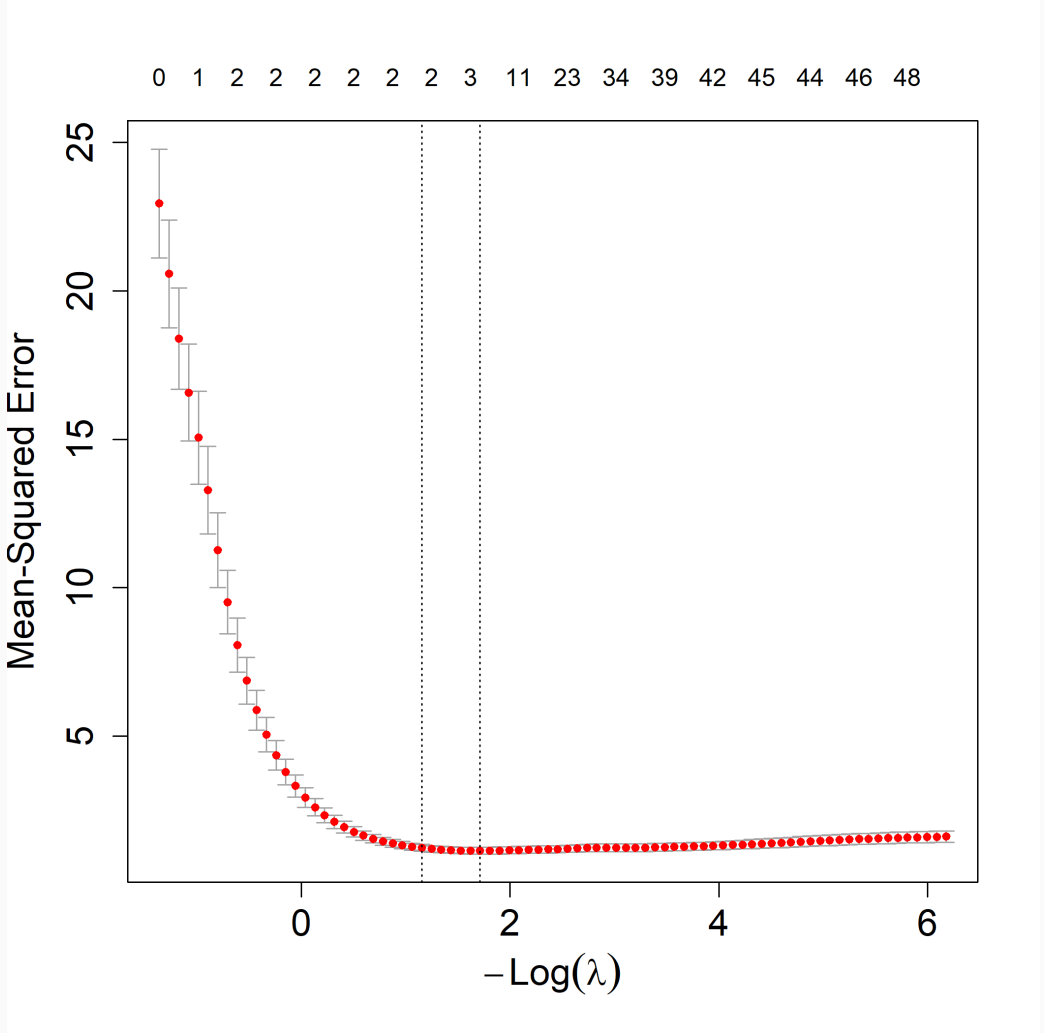
```
# [1] 22.94783
# [1] 1.607909
#
# [1] 1.131168
# [1] 1.131168
```

Try it: optimal lambda was selected using cross-validation

```
# Plot cv mse for each lambda (log scale)
plot(cvfit, cex.lab=1.5, cex.axis=1.5)
```

```
# First vertical line is log(best lambda)
lambda.best
log(lambda.best)
```

```
# [1] 0.1802445
# [1] -1.713441
```



How does lasso MSE compare to ordinary regression?

- Create a test dataset to compare lasso vs regression

```
set.seed(11) # Different seed
nrows <- 100

# Create 50 random variables (X1...X50)
sim_data_test <- data.frame(replicate(50, rnorm(nrows,0,1) ))

# Simulated model: only two X vars are predictive of Y
# Y = 12 + 3*X1 + 4*X2 + error
sim_data_test$y <- 12+3*sim_data_test$X1+4*sim_data_test$X2 + rnorm(nrows, mean=0, sd=1)

y_test <- sim_data_test$y
f      <- formula(y ~ 0 + .) # exclude intercept
x_test <- model.matrix(f, data = sim_data_test)
```

How does lasso MSE compare to ordinary regression?

```
sim_data_test$yhat_reg   <- predict(lm50, sim_data_test, type = "response")
sim_data_test$yhat_lasso <- predict(cvfit, newx=x_test, s = "lambda.min")

mean((sim_data_test$y - sim_data_test$yhat_reg)^2)
mean((sim_data_test$y - sim_data_test$yhat_lasso)^2)
```

```
# [1] 3.371214
# [1] 0.8333494
```

You can estimate a lasso model even when p>n!

- Consider a model with 1000 predictors and 100 observations

```
set.seed(75)
nrows <- 100

# Create 1000 random variables (X1...X1000)
sim_data_1000 <- data.frame(replicate(1000, rnorm(nrows,0,1) ))

# Simulated model: only two X vars are predictive of Y
# Y = 12 + 3*X1 + 4*X2 + error
sim_data_1000$y <- 12+3*sim_data_1000$X1+4*sim_data_1000$X2 + rnorm(nrows, mean=0, sd=1)

y_1000 <- sim_data_1000$y
f      <- formula(y ~ 0 + .) # exclude intercept
x_1000 <- model.matrix(f, data = sim_data_1000)
```

You can estimate a lasso model even when p>n!

```
# Out of 1000 predictors, which ones does lasso set to non-zero?
cvfit_1000 <- cv.glmnet(x=x_1000, y=y_1000)
coef(cvfit_1000)[which(coef(cvfit_1000)!=0),]
```

```
# (Intercept)          X1          X2
#   12.048464    2.699534    3.568011
```

Summary

- Lasso regression shrinks parameter estimates towards zero
- Lasso may produce better predictions than ordinary regression
- Lasso can be used for variable selection

- Lab-12 due Sunday at 11:59pm
- Next class: decision trees (Chapter 8.1)
- Midterm is **Wednesday, October 29**