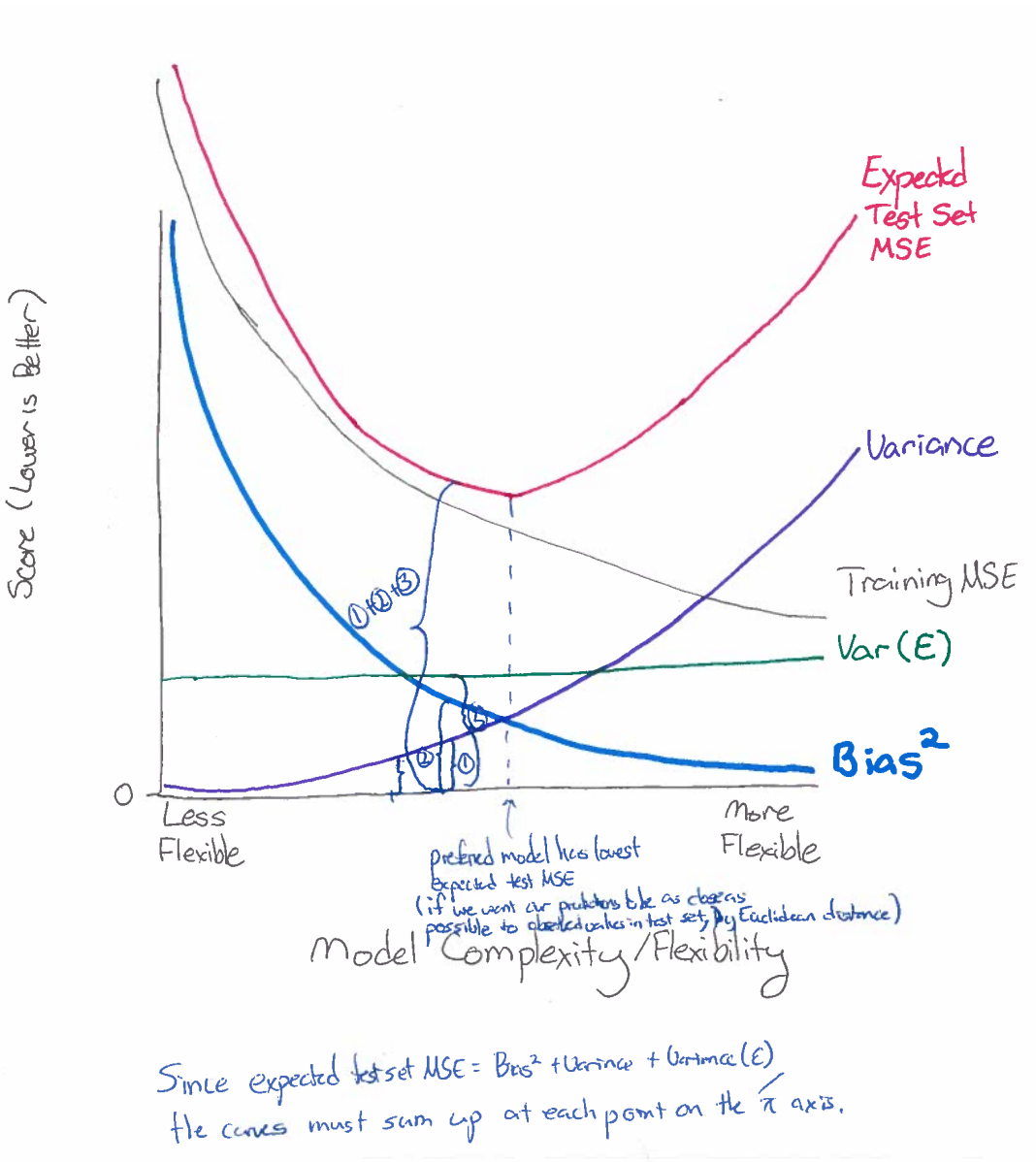
Penalized Estimation and Shrinkage

Challenge:

Training set error (training MSE or training classification error) **always go down** if we add more explanatory variables, higher degree polynomial terms, interactions, ...

Test set error goes down for a while, then back up.



We care about test set error.

Two Running Examples

Example 1: Polynomial Regression

Let's consider fitting polynomials of degree 10 to data generated from the following model:

$$
\begin{aligned}
X_i &\sim \text{Uniform}(-3, 3) \\
Y_i &= 2 + 2X_i - 2X_i^2 + X_i^3 + \varepsilon_i \\
\varepsilon_i &\sim \text{Normal}(0, 20) \\
i &= 1, \ldots, 100
\end{aligned}
$$

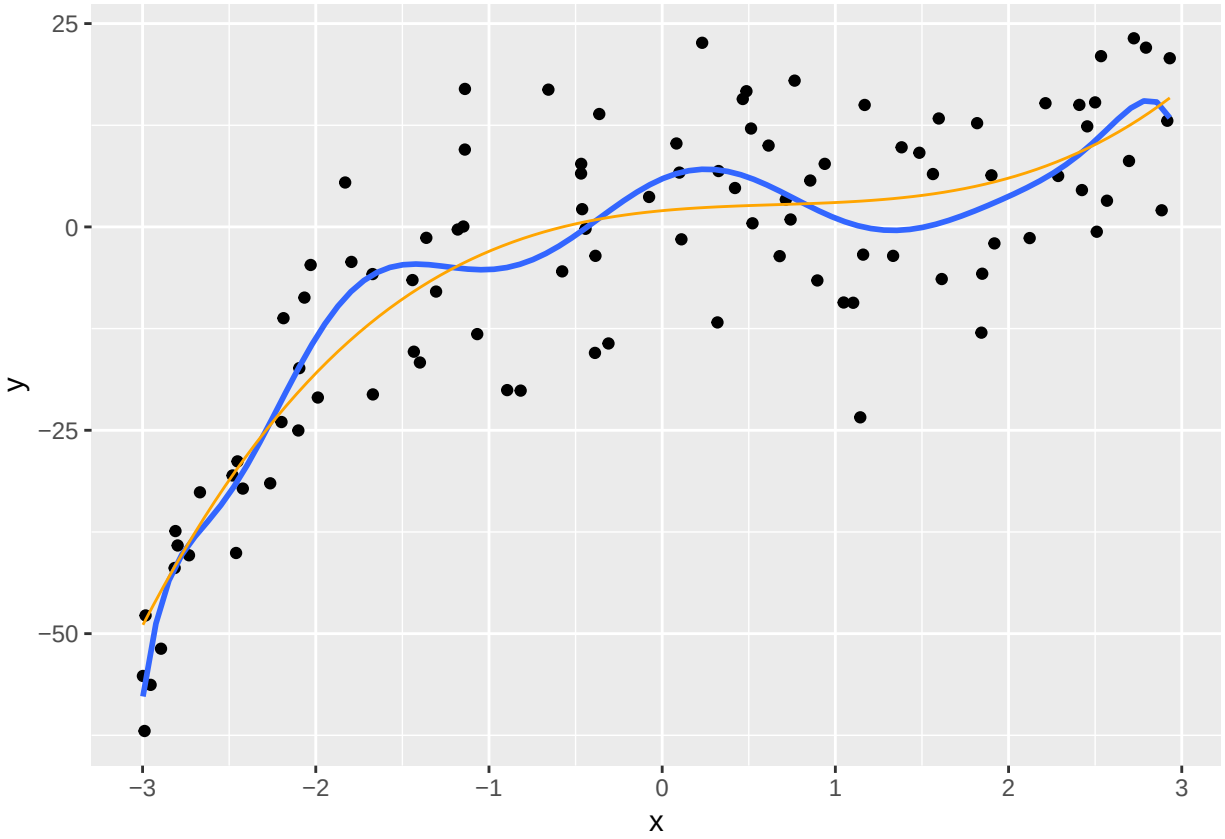
```
true_f_poly <- function(x, beta0, beta1, beta2, beta3) {
  beta0 + beta1 * x + beta2 * x^2 + beta3 * x^3
}

simulate_train_data_poly <- function(n, beta0, beta1, beta2, beta3, sigma) {
  data.frame(
    x = runif(n, min = -3, max = 3)
  ) %>%
    mutate(
      y = true_f_poly(x, beta0, beta1, beta2, beta3) + rnorm(n, mean = 0, sd = sigma)
    )
}

set.seed(974246)
train_data <- simulate_train_data_poly(
  n = 100,
  beta0 =  2, beta1 = 2, beta2 = -2, beta3 = 1, sigma = 10)
head(train_data)
```

```
##            x          y
## 1  2.4542876  12.363632
## 2  2.1219489  -1.370620
## 3  0.8551751   5.703446
## 4  0.7647634  17.984398
## 5 -2.9827294 -47.763883
## 6  1.1438582 -23.408360
```

```
dim(train_data)
```

```
## [1] 100   2
```

```
ggplot(data = train_data, mapping = aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ poly(x, 10), se = FALSE) +
  stat_function(fun = true_f_poly,
    args = list(beta0 =  2, beta1 = 2, beta2 = -2, beta3 = 1),
    color = "orange")
```




```
lm_fit <- lm(y ~ poly(x, 10), data = train_data)
coef(lm_fit)
```

```
##   (Intercept)  poly(x, 10)1  poly(x, 10)2  poly(x, 10)3  poly(x, 10)4
##     -5.327173    143.973446    -70.584437     57.095163     -1.945988
##  poly(x, 10)5  poly(x, 10)6  poly(x, 10)7  poly(x, 10)8  poly(x, 10)9
##      9.234072    -10.096321    -13.167566      1.011708     11.278393
## poly(x, 10)10
##    -12.077299
```

What's the problem?

Those coefficient estimates are too big!!

Example 2: Linear Regression with $p = 100$ explanatory variables

```
true_f_p10 <- function(X, beta) {
  X %*% beta
}

simulate_train_data_p10 <- function(n, beta, sigma) {
  p <- 10
  X <- matrix(runif(n * p, min = -3, max = 3), nrow = n, ncol = p)
  result <- as.data.frame(X)
  colnames(result) <- paste0("X", 1:p)
  result <- result %>%
    mutate(
      y = true_f_p10(X, beta) + rnorm(n, mean = 0, sd = sigma)
    )
}

train_data <- simulate_train_data_p10(
  n = 100,
  beta =  c(1, 2, 3, rep(0, 7)),
  sigma = 20)

lm_fit <- lm(y ~ ., data = train_data)
coef(lm_fit)
```

```
## (Intercept)          X1          X2          X3          X4          X5
##   3.3172496   2.4199628   3.2782373   2.9279032   1.3142178  -1.3582212
##          X6          X7          X8          X9         X10
##  -0.6889279   1.5074253  -0.2138175   0.4920307   0.8193166
```

What's the problem?

(Most of) those coefficient estimates are too big!!

Solution: Shrinkage, a.k.a. Penalized Estimation, a.k.a. Regularized Estimation

Ordinary Least Squares (the standard procedure)

Recall: The standard approach to linear model estimation is Ordinary Least Squares. (Equivalent to maximum likelihood estimation if residuals are normally distributed.)

The coefficients $\beta_0, \ldots, \beta_p$ are estimated by minimizing the Residual Sum of Squares on the training data:

$$\min_{\beta} RSS = \min_{\beta} \sum_{i=1}^{n} \left[y_i - \hat{f}(x_i) \right]^2 = \min_{\beta} \sum_{i=1}^{n} \left[y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \right]^2$$

The RSS measures how well $\widehat{f}$ describes the training data.

In many cases, minimizing the RSS results in overfitting the training data.

A sign/symptom of this is that **coefficient estimates are too large**.

Penalized Estimation

Add a penalty to the RSS that encourages small coefficient estimates

LASSO

Penalty is sum of absolute values of coefficients (other than intercept).

$$\min_{\beta} RSS + \lambda \sum_{j=1}^{p} |\beta_j| = \min_{\beta} \sum_{i=1}^{n} \left[y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \right]^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

We estimate the β's by minimizing the sum of these terms:

- As before, the first term (RSS) says the β coefficients should be chosen so that $\widehat{f}$ is near the training data.
- The second term is larger for coefficients with larger magnitude. Since we are minimizing this, this encourages small coefficient estimates.

λ controls how much large values of β are penalized.

- $\lambda = 0$ means no penalty (effectively, we are doing OLS)
- λ very large means we definitely want to choose small β's to minimize the two terms combined.

Ridge Regression

Penalty is sum of squared values of coefficients (other than intercept). All the intuition is the same.

$$\min_{\beta} RSS + \lambda \sum_{j=1}^{p} \beta_j^2 = \min_{\beta} \sum_{i=1}^{n} \left[y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \right]^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Connection to Bias-Variance Trade-Off

OLS:

- If the linear model is correctly specified (or includes extra stuff), the OLS estimates are **unbiased** (bias = 0).
- But they can have **large variance**.

Penalized Estimation:

- Introduces **some bias** to coefficient estimates and predictions
- Ideally, **reduces variance**
- Exactly where it falls is determined by the value of λ
  - λ near 0: low bias, higher variance
  - λ large: higher bias, lower variance

In practice, we use cross-validation to select the value of λ to make a good bias-variance trade-off.

Connection to Bayesian Inference

The coefficient estimates obtained from LASSO are also the posterior mode in a Bayesian analysis if a double exponential prior distribution is used for the β's.

The coefficient estimates obtained from Ridge Regression are also the posterior mode and the posterior mean in a Bayesian analysis if a normal prior distribution is used for the β's.

Why Shrink Towards 0?

- One motivation above:
  - often, large coefficient estimates are a sign of overfitting
  - a "soft" version of variable selection (coefficients of 0 mean we drop the variable from the model)
- But we could do better!
  - If we have some information from another source about possible values of the coefficients, shrinking towards those values causes less bias, while still reducing variance!
  - This is one of the ideas of Bayesian statistics, but it can also be done in a Frequentist setting.

Application to polynomial regression example

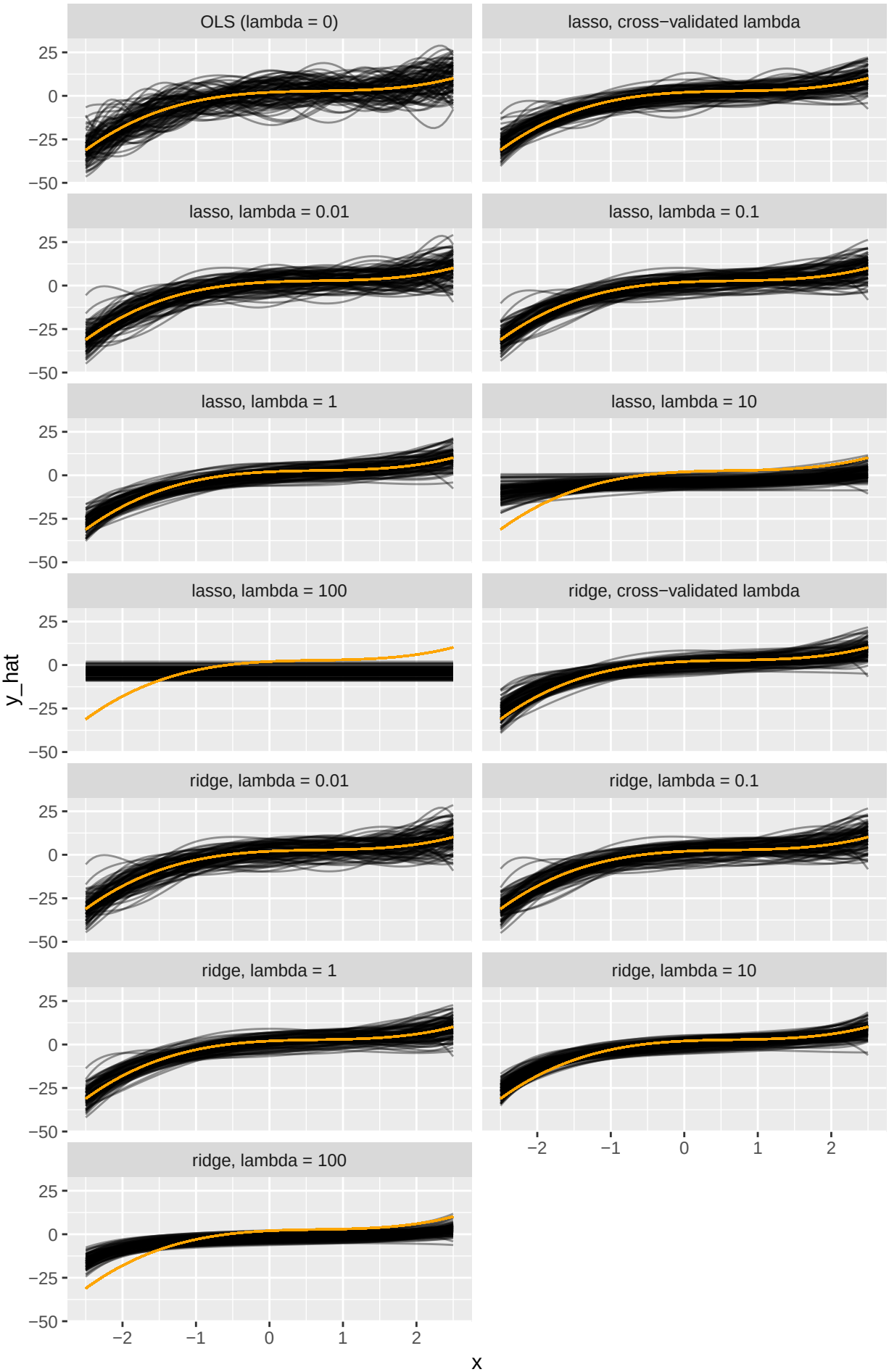
Recall that bias and variance are about behavior of estimators across all possible training sets. To understand behavior, we will look at estimates from each method across 100 randomly generated training sets (each of size $n = 100$) from the model

$$\begin{aligned} X_i &\sim \text{Uniform}(-3, 3) \\ Y_i &= 2 + 2X_i - 2X_i^2 + X_i^3 + \varepsilon_i \\ \varepsilon_i &\sim \text{Normal}(0, 20) \end{aligned}$$

Code suppressed to avoid distraction, but here is what I did:

- Simulate 100 different data sets of size 100 from this model
- For each simulated data set, fit a degree 10 polynomial using each of the following procedures:
  - OLS ($\lambda = 0$)
  - LASSO with the following choices of λ:
    - 0.01, 0.1, 1, 10, 100
    - Cross-validated choice of λ (the selected value was generally a little less than 1)
  - Ridge Regression with the following choices of λ:
    - 0.01, 0.1, 1, 10, 100
    - Cross-validated choice of λ (the selected value was generally a little less than 1)

Plot the resulting estimates $\hat{f}(x)$, with the true function $f(x)$ overlaid in orange.

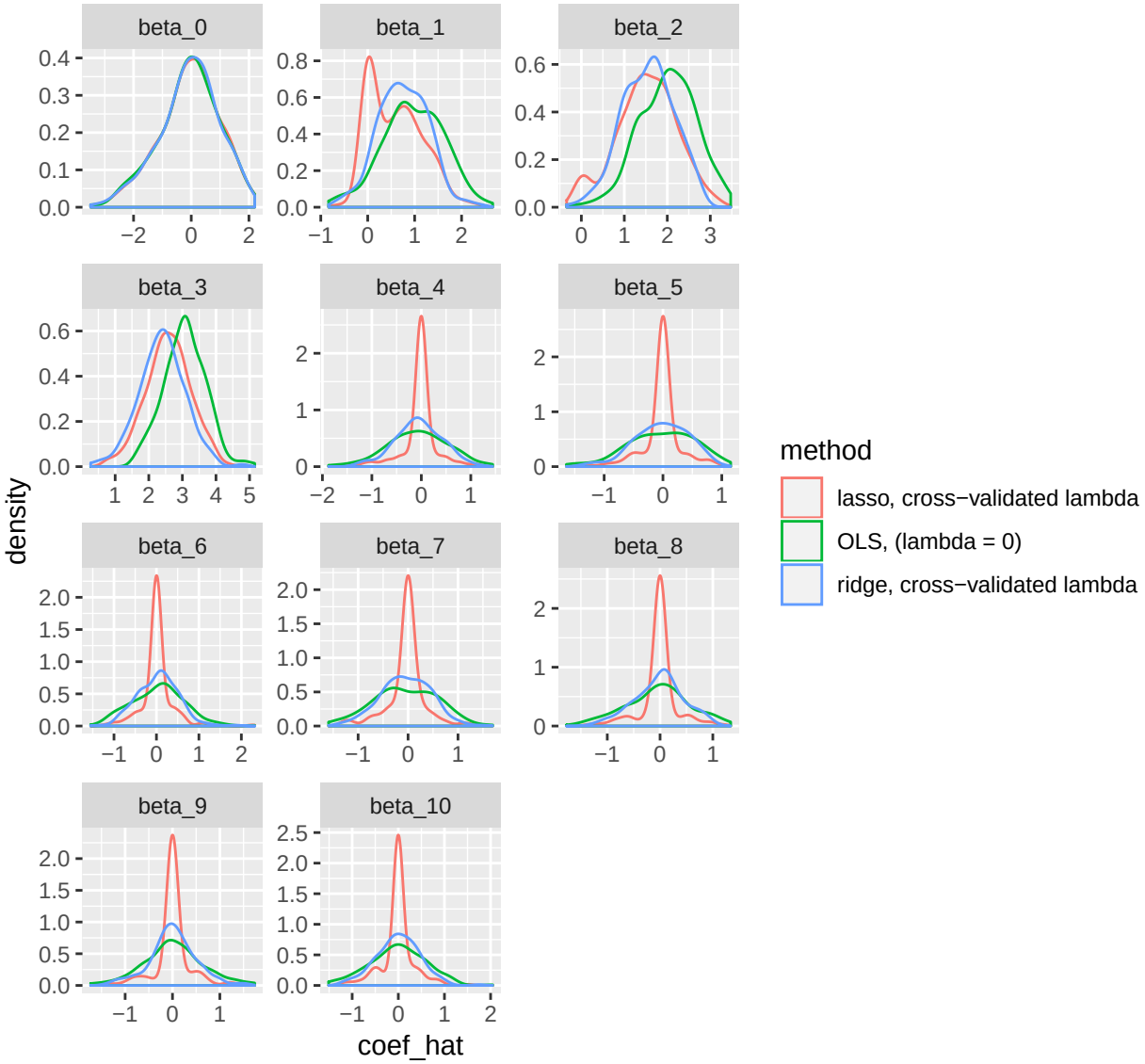

OLS (lambda = 0)
lasso, cross−validated lambda
lasso, lambda = 0.01
lasso, lambda = 0.1
lasso, lambda = 1
lasso, lambda = 10
lasso, lambda = 100
ridge, cross−validated lambda
ridge, lambda = 0.01
ridge, lambda = 0.1
ridge, lambda = 1
ridge, lambda = 10
ridge, lambda = 100
y_hat
x
25
0
−25
−50
−2
−1
0
1
2

Example with large p

I simulated 200 data sets from the model with $p = 10$ explanatory variables.

For each simulated data sets, I estimated the model with OLS, Ridge regression, and LASSO.

Here are density plots summarizing the resulting coefficient estimates.



- We introduced a small amount of bias in estimates of the three non-zero coefficients, in exchange for a large reduction in variance of estimates of the remaining coefficients.
- This bias-variance trade-off in coefficient estimates translates into a bias-variance trade-off in prediction errors.
- LASSO is more aggressive in setting coefficient estimates to 0 than Ridge regression
  - LASSO is preferred if many β's are close to 0
  - Ridge is preferred if not

Example: Prostrate Cancer

This example comes from Chapter 3 of Elements of Statistical Learning. Here's a quote from that book describing the setting:

> The data for this example come from a study by Stamey et al. (1989). They examined the correlation between the level of prostate-specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy. The variables are log cancer volume (lcavol), log prostate weight (lweight), age, log of the amount of benign prostatic hyperplasia (lbph), seminal vesicle invasion (svi), log of capsular penetration (lcp), Gleason score (gleason), and percent of Gleason scores 4 or 5 (pgg45).

Our goal is to understand the relationship between these explanatory variables and the level of prostrate-specific antigen.

```
library(readr)
library(dplyr)
library(ggplot2)
library(caret)
library(leaps) # functionality for best subsets regression

prostrate <- read_table2("http://www.evanlray.com/data/ESL/prostate.data")
prostrate <- prostrate %>%
  select(-ID, -train)
head(prostrate)
```

```
## # A tibble: 6 x 9
##   lcavol lweight   age  lbph   svi   lcp gleason pgg45   lpsa
##    <dbl>   <dbl> <int> <dbl> <int> <dbl>   <int> <int>  <dbl>
## 1 -0.580    2.77    50 -1.39     0 -1.39       6     0 -0.431
## 2 -0.994    3.32    58 -1.39     0 -1.39       6     0 -0.163
## 3 -0.511    2.69    74 -1.39     0 -1.39       7    20 -0.163
## 4 -1.20     3.28    58 -1.39     0 -1.39       6     0 -0.163
## 5  0.751    3.43    62 -1.39     0 -1.39       6     0  0.372
## 6 -1.05     3.23    50 -1.39     0 -1.39       6     0  0.765
```

```
dim(prostrate)
```

```
## [1] 97  9
```

```
set.seed(76471)

# perform train/test split
train_test_split_inds <- caret::createDataPartition(prostrate$lpsa)

prostrate_train <- prostrate %>% slice(train_test_split_inds[[1]])
prostrate_test <- prostrate %>% slice(-train_test_split_inds[[1]])

# standardize all covariates
# this is not really necessary for best subsets regression, but I want to set up
# an ability to compare results to other methods later on.
train_sds <- map_dbl(prostrate, sd)
prostrate_train <- scale(prostrate_train, scale = train_sds) %>%
  as.data.frame()
prostrate_test <- scale(prostrate_test, scale = train_sds) %>%
  as.data.frame()

# A function to calculate mean squared error
calc_mse <- function(observed, predicted) {
  return(mean((observed - predicted)^2))
}
```

Ordinary least squares

```
# ordinary least squares
lm_fit <- lm(lpsa ~ ., data = prostrate_train)
coef(lm_fit)
```

```
##   (Intercept)        lcavol       lweight           age          lbph
## -5.187197e-17  5.237652e-01  3.642817e-01 -1.241648e-01  5.913240e-02
##           svi           lcp       gleason         pgg45
##  2.055260e-01 -1.869933e-01 -8.582756e-02  3.194025e-01
```

```
calc_mse(prostrate_test$lpsa,
  predict(lm_fit, newdata = prostrate_test))
```

```
## [1] 0.3835322
```

```
summary(lm_fit)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = prostrate_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2887 -0.3142  0.0863  0.3893  1.1510
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.187e-17  8.919e-02   0.000  1.00000
## lcavol       5.238e-01  1.443e-01   3.629  0.00078 ***
## lweight      3.643e-01  1.414e-01   2.576  0.01371 *
## age         -1.242e-01  1.108e-01  -1.121  0.26890
## lbph         5.913e-02  1.167e-01   0.507  0.61522
## svi          2.055e-01  1.159e-01   1.773  0.08360 .
## lcp         -1.870e-01  1.686e-01  -1.109  0.27388
## gleason     -8.583e-02  1.372e-01  -0.626  0.53508
## pgg45        3.194e-01  1.643e-01   1.944  0.05878 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6306 on 41 degrees of freedom
## Multiple R-squared:  0.6509, Adjusted R-squared:  0.5827
## F-statistic: 9.554 on 8 and 41 DF,  p-value: 2.411e-07
```

LASSO

```
# first get response and covariates in a format to be used in a call to get the fit
y <- prostrate_train$lpsa
x_train <- model.matrix(lpsa ~ 0 + ., data = prostrate_train)

# cv.glmnet fits the lasso model and
# does cross-validation to evaluate performance of a grid of values for lambda
# alpha = 1 says to do lasso
lasso_fit <- cv.glmnet(x_train, y, alpha = 1)
best_lambda_lasso <- lasso_fit$lambda.min
best_lambda_lasso
```

```
## [1] 0.002473917
```

```
# print out estimated coefficients from lasso
coef(lasso_fit, s = best_lambda_lasso)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -2.236067e-17
## lcavol       5.138602e-01
## lweight      3.611421e-01
## age         -1.202035e-01
## lbph         5.726499e-02
## svi          2.022083e-01
## lcp         -1.698960e-01
## gleason     -7.545335e-02
## pgg45        3.027324e-01
```

```
# get test set mse from lasso
x_test <- model.matrix(lpsa ~ 0 + ., data = prostrate_test)
calc_mse(prostrate_test$lpsa,
  predict(lasso_fit, s = best_lambda_lasso, newx = x_test))
```

```
## [1] 0.3781991
```

Ridge Regression

```
# first get response and covariates in a format to be used in a call to get the fit
# (we don't really need to do this again since it was done above - including for completeness)
y <- prostrate_train$lpsa
x_train <- model.matrix(lpsa ~ 0 + ., data = prostrate_train)

# cv.glmnet fits the ridge model and
# does cross-validation to evaluate performance of a grid of values for lambda
# alpha = 0 says to do ridge regression (even though the penalty involves squaring things)
ridge_fit <- cv.glmnet(x_train, y, alpha = 0)
best_lambda_ridge <- lasso_fit$lambda.min
best_lambda_ridge
```

```
## [1] 0.002473917
```

```
# print out estimated coefficients from ridge
coef(ridge_fit, s = best_lambda_ridge)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -6.619475e-18
## lcavol       4.571737e-01
## lweight      3.334182e-01
## age         -1.057310e-01
## lbph         6.776263e-02
## svi          1.983144e-01
## lcp         -9.557174e-02
## gleason     -4.071260e-02
## pgg45        2.347169e-01
```

```
# get test set mse from ridge
x_test <- model.matrix(lpsa ~ 0 + ., data = prostrate_test)
calc_mse(prostrate_test$lpsa,
  predict(ridge_fit, s = best_lambda_ridge, newx = x_test))
```

```
## [1] 0.3598427
```