Penalized Regression

Prof Wells

STA 295: Stat Learning

March 12th, 2024

Outline

- Investigate the relationship between coefficient size and variance in linear models
- Discuss penalized regression models as means of improving MSE of linear models
- Implement Ridge Regression in R

Section 1

Penalized Regression

Motivation

- Recall, for SLR, $\hat{\beta}_0, \hat{\beta}_1$ are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

Motivation

- Recall, for SLR, $\hat{\beta}_0, \hat{\beta}_1$ are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

- Under the standard assumptions, the coefficients produced by least squares regression are unbiased.

Motivation

- Recall, for SLR, $\hat{\beta}_0, \hat{\beta}_1$ are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

- Under the standard assumptions, the coefficients produced by least squares regression are unbiased.
- That is, if the true relationship between Y and X is linear $Y = \beta_0 + \beta_1 X + \epsilon$, then

$$E[\hat{\beta}_0] = \beta_0 \qquad E[\hat{\beta}_1] = \beta_1$$

Motivation

- Recall, for SLR, $\hat{\beta}_0, \hat{\beta}_1$ are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

- Under the standard assumptions, the coefficients produced by least squares regression are unbiased.
- That is, if the true relationship between Y and X is linear $Y = \beta_0 + \beta_1 X + \epsilon$, then

$$E[\hat{\beta}_0] = \beta_0 \qquad E[\hat{\beta}_1] = \beta_1$$

- Moreover, among all **unbiased** linear models, the least squares model has the lowest variance.

Motivation

- Recall, for SLR, $\hat{\beta}_0, \hat{\beta}_1$ are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

- Under the standard assumptions, the coefficients produced by least squares regression are unbiased.
- That is, if the true relationship between Y and X is linear $Y = \beta_0 + \beta_1 X + \epsilon$, then

$$E[\hat{\beta}_0] = \beta_0 \qquad E[\hat{\beta}_1] = \beta_1$$

- Moreover, among all **unbiased** linear models, the least squares model has the lowest variance.
- Does this mean that the least squares model has the lowest MSE among all linear models?

Motivation

- Recall, for SLR, $\hat{\beta}_0, \hat{\beta}_1$ are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

- Under the standard assumptions, the coefficients produced by least squares regression are unbiased.
- That is, if the true relationship between Y and X is linear $Y = \beta_0 + \beta_1 X + \epsilon$, then

$$E[\hat{\beta}_0] = \beta_0 \qquad E[\hat{\beta}_1] = \beta_1$$

- Moreover, among all **unbiased** linear models, the least squares model has the lowest variance.
- Does this mean that the least squares model has the lowest MSE among all linear models?
    - No! MSE is a combination of bias and variance.
    - It is possible that a small *increase* in bias can correspond to large *decrease* in variance.

Shrinking Coefficients

- Suppose the true relationship between Y and X_1, X_2 is given by

$$Y = 1 + X_1 + 5X_2 + \epsilon \quad \epsilon \sim N(0, 1).$$

- Let $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ be the model coefficient estimates given by least squares regression. Which of the following models has higher variance in predictor estimates? Higher bias?

Shrinking Coefficients

- Suppose the true relationship between Y and X_1, X_2 is given by

$$Y = 1 + X_1 + 5X_2 + \epsilon \quad \epsilon \sim N(0,1).$$

- Let $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ be the model coefficient estimates given by least squares regression. Which of the following models has higher variance in predictor estimates? Higher bias?

$$\begin{aligned} \text{Model 1:} \quad \hat{y} =& \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 \\ \text{Model 2:} \quad \hat{y} =& \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2 \end{aligned}$$

Shrinking Coefficients

- Suppose the true relationship between Y and X_1, X_2 is given by

$$Y = 1 + X_1 + 5X_2 + \epsilon \quad \epsilon \sim N(0,1).$$

- Let $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ be the model coefficient estimates given by least squares regression. Which of the following models has higher variance in predictor estimates? Higher bias?

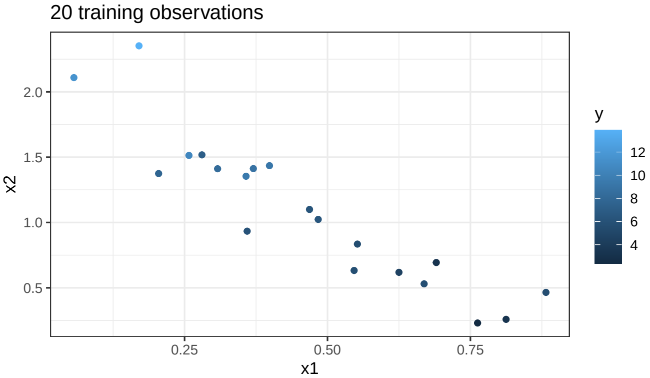
$$\text{Model 1:} \quad \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$$

$$\text{Model 2:} \quad \hat{y} = \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2$$

- Model 2 has higher bias, but lower variance.

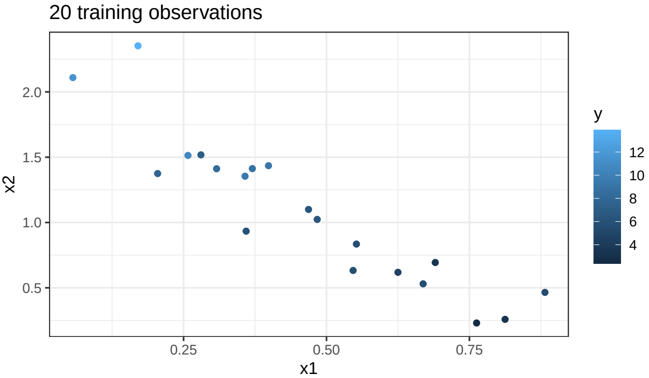
A Linear Model

- Consider the following training data for the model:
  $Y = 1 + X_1 + 5X_2 + \epsilon \quad \epsilon \sim N(0,1)$



A Linear Model

- Consider the following training data for the model: $Y = 1 + X_1 + 5X_2 + \epsilon \quad \epsilon \sim N(0,1)$



- What are some likely problems with the MLR model?

Bias-Variance in Least Squares

- Using least squares, the model estimates are

$$\hat{Y} = -0.5 + 2.8X_1 + 5.8X_2$$

Bias-Variance in Least Squares

- Using least squares, the model estimates are

$$\hat{Y} = -0.5 + 2.8X_1 + 5.8X_2$$

- Let's consider variance and bias for estimate Y when $X_1 = 0.25$ and $X_2 = .5$.

Bias-Variance in Least Squares

- Using least squares, the model estimates are

$$\hat{Y} = -0.5 + 2.8X_1 + 5.8X_2$$

- Let's consider variance and bias for estimate Y when $X_1 = 0.25$ and $X_2 = .5$.
  - Using the true model, the expected value of Y is

$$Y = 1 + X_1 + 5 \cdot X_2 = 1 + 0.25 + 5 \cdot 0.5 = 3.75$$

Bias-Variance in Least Squares

- Using least squares, the model estimates are

$$\hat{Y} = -0.5 + 2.8X_1 + 5.8X_2$$

- Let's consider variance and bias for estimate Y when $X_1 = 0.25$ and $X_2 = .5$.
  - Using the true model, the expected value of Y is

$$Y = 1 + X_1 + 5 \cdot X_2 = 1 + 0.25 + 5 \cdot 0.5 = 3.75$$

  - Using the least squares model from training data, the predicted value of Y is

$$Y = -0.5 + 2.8X_1 + 5.8X_2 = -0.5 + 2.8 \cdot 0.25 + 5.8 \cdot 0.5 = 3.1$$

Bias-Variance in Least Squares

- Using least squares, the model estimates are

$$\hat{Y} = -0.5 + 2.8X_1 + 5.8X_2$$

- Let's consider variance and bias for estimate Y when $X_1 = 0.25$ and $X_2 = .5$.
  - Using the true model, the expected value of Y is

$$Y = 1 + X_1 + 5 \cdot X_2 = 1 + 0.25 + 5 \cdot 0.5 = 3.75$$

  - Using the least squares model from training data, the predicted value of Y is

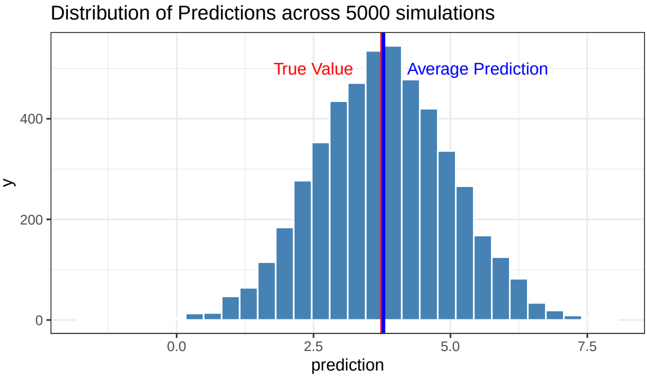
$$Y = -0.5 + 2.8X_1 + 5.8X_2 = -0.5 + 2.8 \cdot 0.25 + 5.8 \cdot 0.5 = 3.1$$

- But how will the predicted value change if we repeat across 5000 simulations from the model?

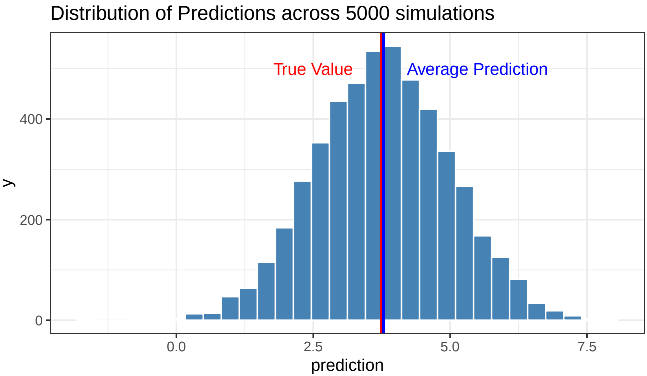
Simulation

```
set.seed(1011)
test_point <- data.frame(x1 = 0.25, x2 = .5)

trials<-5000
prediction <- rep(NA, trials)
for (i in 1:trials){
  e<- rnorm(20,0,1)
  y<- 1 + x1 + 5*x2 + e
  sim_data <- data.frame(x1,x2,y)
  mod <- lm(y ~ x1 + x2, data = sim_data)
  prediction[i] <- predict(mod, test_point)
}

simulation <- data.frame(trial_num = 1:trials, prediction)
```

Prediction Distribution



Prediction Distribution




```
simulation %>% summarize(
  mean = mean(prediction), variance = var(prediction))
```

```
##       mean variance
## 1 3.772056 1.480935
```

A Shrunken Model

- Now suppose we use the model algorithm

$$\hat{y} = \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2$$

- Since $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ are unbiased, then the expected prediction for Y when $X_1 = 0.25$ and $X_2 = 0.5$ is

$$E[\hat{y}] = \beta_1 + 0.97 \cdot \beta_1 x_1 + 0.98 \cdot \beta_2 x_2 = 1 + 0.97 \cdot 0.25 + 0.98 \cdot 5 \cdot 0.5 = 3.69$$

A Shrunken Model

- Now suppose we use the model algorithm

$$\hat{y} = \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2$$

- Since $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ are unbiased, then the expected prediction for Y when $X_1 = 0.25$ and $X_2 = 0.5$ is

$$E[\hat{y}] = \beta_1 + 0.97 \cdot \beta_1 x_1 + 0.98 \cdot \beta_2 x_2 = 1 + 0.97 \cdot 0.25 + 0.98 \cdot 5 \cdot 0.5 = 3.69$$

- Based on the first simulation, the model estimate is

$$\hat{Y} = -0.5 + 0.97 \cdot 2.8 X_1 + 0.98 \cdot 5.8 X_2 = -0.5 + 2.71 X_1 + 5.68 X_2$$

A Shrunken Model

- Now suppose we use the model algorithm

$$\hat{y} = \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2$$

- Since $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ are unbiased, then the expected prediction for Y when $X_1 = 0.25$ and $X_2 = 0.5$ is

$$E[\hat{y}] = \beta_1 + 0.97 \cdot \beta_1 x_1 + 0.98 \cdot \beta_2 x_2 = 1 + 0.97 \cdot 0.25 + 0.98 \cdot 5 \cdot 0.5 = 3.69$$

- Based on the first simulation, the model estimate is

$$\hat{Y} = -0.5 + 0.97 \cdot 2.8 X_1 + 0.98 \cdot 5.8 X_2 = -0.5 + 2.71 X_1 + 5.68 X_2$$

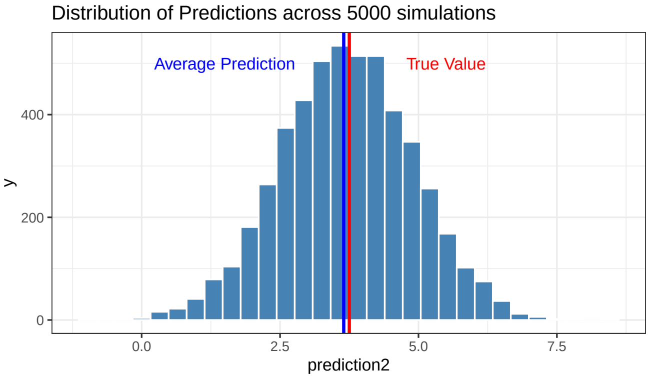
- And the prediction when $X_1 = 0.25$ and $X_2 = 0.5$ is

$$\hat{y} = -0.5 + 2.71 X_1 + 5.68 X_2 = -0.5 + 2.71 \cdot 0.25 + 5.68 \cdot 0.5 = 3.525$$

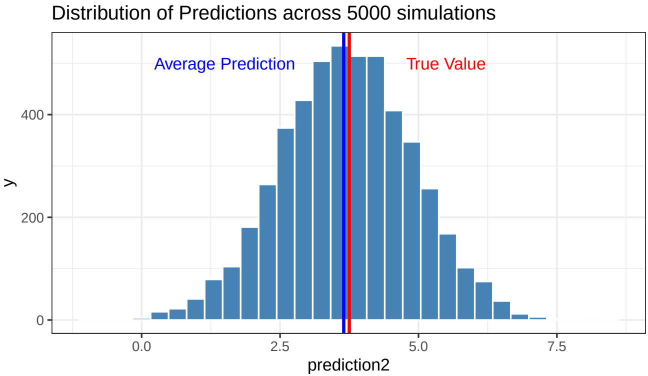
Simulation II

```
set.seed(1001)

trials<-5000
prediction2 <- rep(NA, trials)
for (i in 1:trials){
  e<- rnorm(20,0,1)
  y<- 1 + x1 + 5*x2 + e
  sim_data <- data.frame(x1,x2,y)
  mod <- lm(y ~ x1 + x2, data = sim_data)
  b0 <- 1*coef(mod)[1]
  b1 <- .97*coef(mod)[2]
  b2 <- .98*coef(mod)[3]
  prediction2[i] <- b0 + b1*0.25 + b2*0.5
}

simulation2 <- data.frame(trial_num = 1:trials, prediction2)
```

Prediction Distribution



Prediction Distribution




```
simulation2 %>% summarize(
  mean = mean(prediction2), variance = var(prediction2))
```

```
##       mean variance
## 1 3.70387 1.434099
```

Model Comparison

- True relationship: $Y = 1 + X_1 + 5X_2 + \epsilon$

Model Comparison

- True relationship: $Y = 1 + X_1 + 5X_2 + \epsilon$
- Model 1: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$

```
##       mean variance avg_error
## 1 3.772056 1.480935  1.481125
```

Model Comparison

- True relationship: $Y = 1 + X_1 + 5X_2 + \epsilon$
- Model 1: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$

```
##       mean variance avg_error
## 1 3.772056 1.480935  1.481125
```

- Model 2: $\hat{y} = \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2$

```
##      mean variance avg_error
## 1 3.70387 1.434099  1.435941
```

Model Comparison

- True relationship: $Y = 1 + X_1 + 5X_2 + \epsilon$
- Model 1: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$

```
##       mean variance avg_error
## 1 3.772056 1.480935  1.481125
```

- Model 2: $\hat{y} = \hat{\beta}_0 + 0.97 \cdot \hat{\beta}_1 x_1 + 0.98 \cdot \hat{\beta}_2 x_2$

```
##      mean variance avg_error
## 1 3.70387 1.434099  1.435941
```

- It looks like the model with smaller coefficients actually performed better!

Section 2

Ridge Regression

Shrinkage Penalty

- There are some situations in which multiple linear regression has high MSE:

Shrinkage Penalty

- There are some situations in which multiple linear regression has high MSE:
  - Predictors are strongly correlated (high variance)
  - Many predictors relative to data size (high variance)
  - Model form is non-linear (high bias)

Shrinkage Penalty

- There are some situations in which multiple linear regression has high MSE:
  - Predictors are strongly correlated (high variance)
  - Many predictors relative to data size (high variance)
  - Model form is non-linear (high bias)
- To improve models in the first two cases, we reduce MSE by reducing variance at the cost slight increase in bias.

Shrinkage Penalty

- There are some situations in which multiple linear regression has high MSE:
  - Predictors are strongly correlated (high variance)
  - Many predictors relative to data size (high variance)
  - Model form is non-linear (high bias)
- To improve models in the first two cases, we reduce MSE by reducing variance at the cost slight increase in bias.
- In the presence of multicollinearity or over-fitting, least squares estimates tend to be too large.

Shrinkage Penalty

- There are some situations in which multiple linear regression has high MSE:
  - Predictors are strongly correlated (high variance)
  - Many predictors relative to data size (high variance)
  - Model form is non-linear (high bias)
- To improve models in the first two cases, we reduce MSE by reducing variance at the cost slight increase in bias.
- In the presence of multicollinearity or over-fitting, least squares estimates tend to be too large.
- To build a better model, we reduce the size of coefficients relative to least squares regression.

Ridge Regression

- Recall that least squares regression estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ for

$$\hat{y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

are obtained by finding the values of β that minimize

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

Ridge Regression

- Recall that least squares regression estimates $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ for
$$\hat{y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$
are obtained by finding the values of β that minimize
$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$
- To perform **Ridge Regression**, we instead find coefficients β that minimize
$$\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2 \qquad \text{where } \lambda \geq 0 \text{ is tuning parameter}$$

Ridge Regression

- Recall that least squares regression estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ for

$$\hat{y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

  are obtained by finding the values of β that minimize

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- To perform **Ridge Regression**, we instead find coefficients β that minimize

$$\text{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2 \qquad \text{where } \lambda \geq 0 \text{ is tuning parameter}$$

Why?

Ridge Regression

- Recall that least squares regression estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ for

$$\hat{y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

are obtained by finding the values of β that minimize

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- To perform **Ridge Regression**, we instead find coefficients β that minimize

$$\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2 \qquad \text{where } \lambda \geq 0 \text{ is tuning parameter}$$

Why?

- The term $\lambda \sum_{i=1}^{p} \beta_i^2$ is the **shrinkage penalty**, and is small when the β are small.

Ridge Regression

- Recall that least squares regression estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ for

$$\hat{y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

are obtained by finding the values of β that minimize

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- To perform **Ridge Regression**, we instead find coefficients β that minimize

$$\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2 \qquad \text{where } \lambda \geq 0 \text{ is tuning parameter}$$

Why?

- The term $\lambda \sum_{i=1}^{p} \beta_i^2$ is the **shrinkage penalty**, and is small when the β are small.
- With a shrinkage penalty, the algorithm prefers models with lower coefficients.

Ridge Regression

- Recall that least squares regression estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ for

$$\hat{y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

are obtained by finding the values of β that minimize

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- To perform **Ridge Regression**, we instead find coefficients β that minimize

$$\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2 \qquad \text{where } \lambda \geq 0 \text{ is tuning parameter}$$

Why?

- The term $\lambda \sum_{i=1}^{p} \beta_i^2$ is the **shrinkage penalty**, and is small when the β are small.
- With a shrinkage penalty, the algorithm prefers models with lower coefficients.
- This tends to reduce variance, at the cost of increased bias.

Effects of the Tuning Parameter

- **Goal:** Find β which minimize $\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2$

Effects of the Tuning Parameter

- **Goal:** Find β which minimize $\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2$
- What will happen to β_i as $\lambda \to \infty$? As $\lambda \to 0$?

Effects of the Tuning Parameter

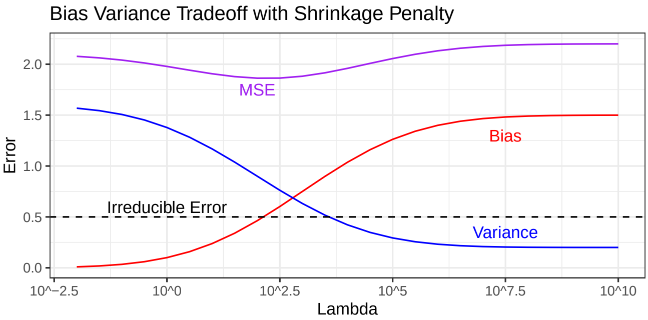
- **Goal:** Find β which minimize $\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2$
- What will happen to β_i as $\lambda \to \infty$? As $\lambda \to 0$?
- What will happen to β_0 as $\lambda \to \infty$? As $\lambda \to 0$?

Effects of the Tuning Parameter

- **Goal:** Find β which minimize $\text{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2$
- What will happen to β_i as $\lambda \to \infty$? As $\lambda \to 0$?
- What will happen to β_0 as $\lambda \to \infty$? As $\lambda \to 0$?
- What happens to MSE as $\lambda \to 0$ or $\lambda \to \infty$?

Effects of the Tuning Parameter

- **Goal:** Find β which minimize $\mathrm{RSS} + \lambda \sum_{i=1}^{p} \beta_i^2$
- What will happen to β_i as $\lambda \to \infty$? As $\lambda \to 0$?
- What will happen to β_0 as $\lambda \to \infty$? As $\lambda \to 0$?
- What happens to MSE as $\lambda \to 0$ or $\lambda \to \infty$?



Simulation

- Consider a linear model with 9 predictors and 100 observations.

$$y = 10 + 1x_1 + 2x_2 \cdots + 8x_8 + 9x_9 + \epsilon \quad \epsilon \sim N(0, 4)$$

Simulation

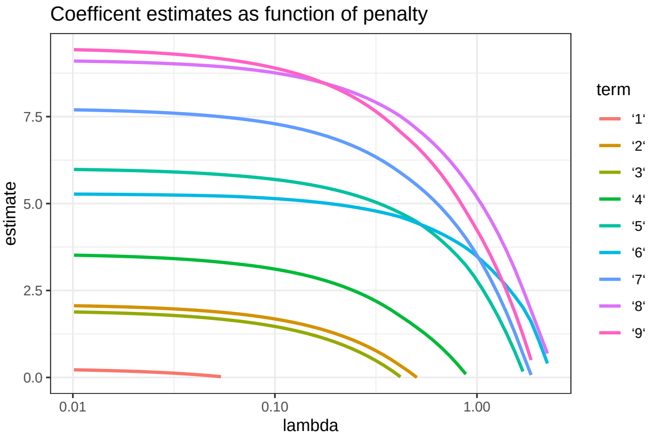
- Consider a linear model with 9 predictors and 100 observations.

$$y = 10 + 1x_1 + 2x_2 \cdots + 8x_8 + 9x_9 + \epsilon \quad \epsilon \sim N(0, 4)$$

```
##
## Call:
## lm(formula = y ~ ., data = sim_data2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.5148 -1.5155 -0.0932  1.8054  5.1007
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6034     1.3023   0.463   0.6443
## x1            0.2653     0.8831   0.300   0.7645
## x2            2.1047     0.8005   2.629   0.0101 *
## x3            1.9316     0.7766   2.487   0.0147 *
## x4            3.5635     0.8133   4.382 3.18e-05 ***
## x5            6.0143     0.7925   7.589 2.84e-11 ***
## x6            5.2844     0.7810   6.766 1.30e-09 ***
## x7            7.7421     0.8657   8.944 4.51e-14 ***
## x8            9.1352     0.7466  12.236  < 2e-16 ***
## x9            9.4859     0.8046  11.789  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.244 on 90 degrees of freedom
## Multiple R-squared:  0.8437, Adjusted R-squared:  0.828
## F-statistic: 53.97 on 9 and 90 DF,  p-value: < 2.2e-16
```

Simulation

- What happens to the size of coefficients as λ gets larger?



Scale

The coefficients in the least squares regression equation are **scale-equivalent**

Scale

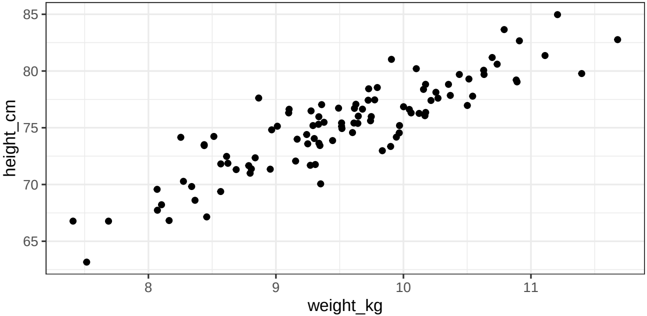
The coefficients in the least squares regression equation are **scale-equivalent**

- That is, scaling a predictor x_i by a value c will lead rescaling slope estimate $\hat{\beta}_i$ by $1/c$.
  - The predicted value is the same, regardless of scale.

Scale

The coefficients in the least squares regression equation are **scale-equivalent**

- That is, scaling a predictor x_i by a value c will lead rescaling slope estimate $\hat{\beta}_i$ by $1/c$.
  - The predicted value is the same, regardless of scale.
- Suppose we model a toddler's height (in cm) based on their weight (in kg)



Scale

- Suppose we model a toddler's height (in cm) based on their weight (in kg)

```
lm1 <- lm(height_cm ~ weight_kg, data = toddler)
summary(lm1)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)    36.61     2.0709   17.68 3.017e-32
## weight_kg       4.05     0.2166   18.70 4.322e-34
```

- For every 1 kg increase in weight, the model predicts a 4.05 cm increase in height.

Scale

- Suppose we model a toddler's height (in cm) based on their weight (in kg)

```
lm1 <- lm(height_cm ~ weight_kg, data = toddler)
summary(lm1)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)    36.61     2.0709   17.68 3.017e-32
## weight_kg       4.05     0.2166   18.70 4.322e-34
```

- For every 1 kg increase in weight, the model predicts a 4.05 cm increase in height.

```
predict(lm1, newdata = data.frame(weight_kg = 10))
```

```
##     1
## 77.11
```

- The predicted height for a 10 kg toddler is 77.11 cm.

Scale

- If we instead measured weight in grams

```
toddler <- toddler %>% mutate(weight_g = 1000*weight_kg)
```

Scale

- If we instead measured weight in grams

```
toddler <- toddler %>% mutate(weight_g = 1000*weight_kg)
```

```
lm2 <- lm(height_cm ~ weight_g, data = toddler)
summary(lm2)$coefficients
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 36.60964  2.0708779   17.68 3.017e-32
## weight_g     0.00405  0.0002166   18.70 4.322e-34
```

- For every 1 g increase in weight, the model predicts a 0.00405 cm increase in height.

Scale

- If we instead measured weight in grams

```
toddler <- toddler %>% mutate(weight_g = 1000*weight_kg)
```

```
lm2 <- lm(height_cm ~ weight_g, data = toddler)
summary(lm2)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 36.60964  2.0708779   17.68 3.017e-32
## weight_g     0.00405  0.0002166   18.70 4.322e-34
```

- For every 1 g increase in weight, the model predicts a 0.00405 cm increase in height.

```
predict(lm2, newdata = data.frame(weight_g = 10*1000))
```

```
##     1
## 77.11
```

- The predicted height for a 10 kg toddler is still 77.11 cm.

Scale

- If we instead measured weight in grams

```
toddler <- toddler %>% mutate(weight_g = 1000*weight_kg)
```

```
lm2 <- lm(height_cm ~ weight_g, data = toddler)
summary(lm2)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 36.60964  2.0708779   17.68 3.017e-32
## weight_g     0.00405  0.0002166   18.70 4.322e-34
```

- For every 1 g increase in weight, the model predicts a 0.00405 cm increase in height.

```
predict(lm2, newdata = data.frame(weight_g = 10*1000))
```

```
##     1
## 77.11
```

- The predicted height for a 10 kg toddler is still 77.11 cm.
- Rescaling predictors in a least squares model *does not* change the model accuracy (predictions and RSS do not change)

Scale

- However, for Ridge Regression, the optimal model depends on the relative scale of the predictors. Changing the scale of one predictor will lead to a different optimal model.

Scale

- However, for Ridge Regression, the optimal model depends on the relative scale of the predictors. Changing the scale of one predictor will lead to a different optimal model.
    - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2$

Scale

- However, for Ridge Regression, the optimal model depends on the relative scale of the predictors. Changing the scale of one predictor will lead to a different optimal model.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2$
- Consider Y as a function of predictors X_1 and X_2

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2'(X_2/1000)$$

- In the second case, we rescaled X_2 by a factor of $1/1000$. Comparable predictions will be made for $\beta_2' \approx 1000 \cdot \beta_2$.

Scale

- However, for Ridge Regression, the optimal model depends on the relative scale of the predictors. Changing the scale of one predictor will lead to a different optimal model.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2$
- Consider Y as a function of predictors X_1 and X_2

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2'(X_2/1000)$$

- In the second case, we rescaled X_2 by a factor of $1/1000$. Comparable predictions will be made for $\beta_2' \approx 1000 \cdot \beta_2$.
- In the second case, ridge regression will prefer models with very small β_2'; and therefore, will select models which make predictions using only minimal contributions of X_2.

Scale

- However, for Ridge Regression, the optimal model depends on the relative scale of the predictors. Changing the scale of one predictor will lead to a different optimal model.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2$
- Consider Y as a function of predictors X_1 and X_2

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2'(X_2/1000)$$

- In the second case, we rescaled X_2 by a factor of $1/1000$. Comparable predictions will be made for $\beta_2' \approx 1000 \cdot \beta_2$.
- In the second case, ridge regression will prefer models with very small β_2'; and therefore, will select models which make predictions using only minimal contributions of X_2.
  - In the first case, ridge regression may prefer models where β is relatively large, and so selects models which do include contributions from X_2.

Scale

- However, for Ridge Regression, the optimal model depends on the relative scale of the predictors. Changing the scale of one predictor will lead to a different optimal model.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2$
- Consider Y as a function of predictors X_1 and X_2

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \qquad Y = \beta_0 + \beta_1 X_1 + \beta_2'(X_2/1000)$$

- In the second case, we rescaled X_2 by a factor of $1/1000$. Comparable predictions will be made for $\beta_2' \approx 1000 \cdot \beta_2$.
- In the second case, ridge regression will prefer models with very small β_2'; and therefore, will select models which make predictions using only minimal contributions of X_2.
  - In the first case, ridge regression may prefer models where β is relatively large, and so selects models which do include contributions from X_2.
- Ridge regression is most effective if predictors are standardized first.

Section 3

Ridge Regression in R

Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

- For this demonstration, we'll work with just a subset of 30% of the available observations.

Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

- For this demonstration, we'll work with just a subset of 30% of the available observations.
- This subsetted data has been split into a training set `solTrain` and a testing set `solTest`.

Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

- For this demonstration, we'll work with just a subset of 30% of the available observations.
- This subsetted data has been split into a training set `solTrain` and a testing set `solTest`.

Solubility

The `solubility` data set from the `AppliedPredictiveModeling` package contains solubility and chemical structure for a sample of 1,267 different compounds.

- For this demonstration, we'll work with just a subset of 30% of the available observations.
- This subsetted data has been split into a training set `solTrain` and a testing set `solTest`.

```
nrow(solTrain)
```

```
## [1] 285
```

```
ncol(solTrain)
```

```
## [1] 21
```

```
nrow(solTest)
```

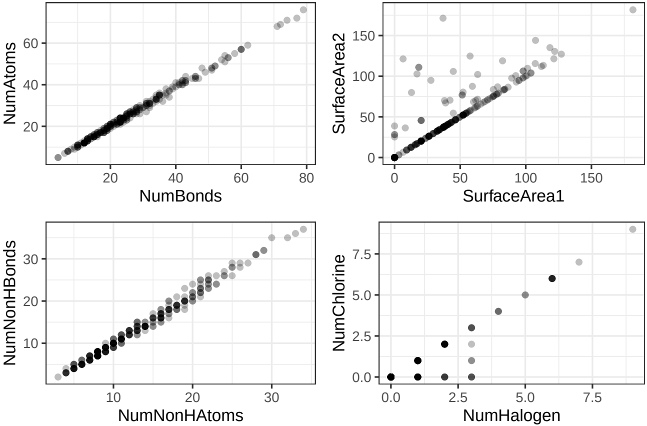
```
## [1] 95
```

```
ncol(solTest)
```

```
## [1] 21
```

Multicollinearity

- Recall that several predictors were very strongly correlated
  - We even removed several from our linear model because of they were completed determined by the values of other variables (`NumNonHBonds` `NumHydrogen` `NumRings`)



Feature Selection

- Previously, we used `regsubsets` from the `leaps` package to choose the best model:

```
best15 <-lm(Solubility ~.-NumNonHBonds -NumHydrogen -NumRings
            -NumNitrogen -NumOxygen,
            data = solTrain)
```

Feature Selection

- Previously, we used `regsubsets` from the `leaps` package to choose the best model:

```
best15 <-lm(Solubility ~.-NumNonHBonds -NumHydrogen -NumRings
            -NumNitrogen -NumOxygen,
            data = solTrain)
```

- And computed the MSE of the model on test data

```
preds <- predict(best15, solTest)
data.frame(
  mse = mean((solTest$Solubility - preds)^2)
  )
```

```
##      mse
## 1 0.7549
```

Variable Importance

- The summary table suggests most variables have very significant p-value.

```
##
## Call:
## lm(formula = Solubility ~ . - NumNonHBonds - NumHydrogen - NumRings -
##     NumNitrogen - NumOxygen, data = solTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9349 -0.5748  0.0814  0.6091  1.8835
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.31384    0.29783    1.05  0.29295
## MolWeight         -0.00826    0.00276   -2.99  0.00301 **
## NumAtoms           0.22441    0.14905    1.51  0.13336
## NumNonHAtoms       1.21912    0.20542    5.93  9.0e-09 ***
## NumBonds          -0.54781    0.17740   -3.09  0.00223 **
## NumMultBonds      -1.36634    0.38003   -3.60  0.00039 ***
## NumRotBonds       -0.08849    0.05353   -1.65  0.09947 .
## NumDblBonds        0.47275    0.31674    1.49  0.13673
## NumAromaticBonds   0.99386    0.34750    2.86  0.00457 **
## NumCarbon         -0.40511    0.12471   -3.25  0.00131 **
## NumSulfer          0.35662    0.44543    0.80  0.42405
## NumChlorine       -0.28807    0.16132   -1.79  0.07528 .
## NumHalogen        -1.32653    0.28033   -4.73  3.6e-06 ***
## HydrophilicFactor  0.20762    0.15463    1.34  0.18050
## SurfaceArea1       0.03301    0.01460    2.26  0.02462 *
## SurfaceArea2      -0.05094    0.01692   -3.01  0.00285 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.927 on 269 degrees of freedom
## Multiple R-squared:  0.791,  Adjusted R-squared:  0.779
## F-statistic: 67.9 on 15 and 269 DF,  p-value: <2e-16
```

Rescaling a Data Frame

- We can use the `scale` function in R to standardize every column of a data frame:

```
std_solTrain <- scale(solTrain) %>% as.data.frame()
```

Rescaling a Data Frame

- We can use the `scale` function in R to standardize every column of a data frame:

```
std_solTrain <- scale(solTrain) %>% as.data.frame()
```

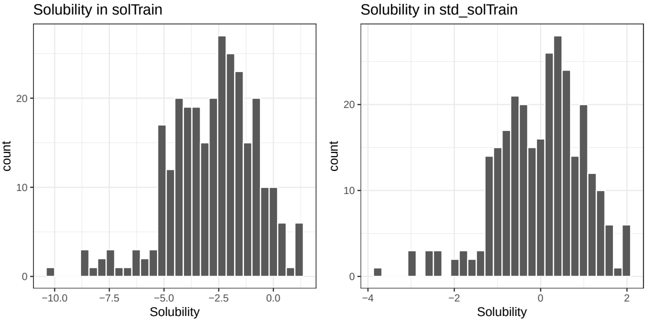
- A quick verification:

Rescaling a Data Frame

- We can use the `scale` function in R to standardize every column of a data frame:

```
std_solTrain <- scale(solTrain) %>% as.data.frame()
```

- A quick verification:




```
##              df mean_sol sd_sol
## 1      solTrain   -2.775  1.974
## 2 std_solTrain    0.000  1.000
```

Scaled Model Coefficients

- Some coefficients are still relatively large (possibly because of collinearity)

```
##
## Call:
## lm(formula = Solubility ~ . - NumNonHBonds - NumHydrogen - NumRings -
##     NumNitrogen - NumOxygen, data = std_solTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4871 -0.2912  0.0412  0.3086  0.9544
##
## Coefficients:
##                    Estimate Std. Error t value    Pr(>|t|)
## (Intercept)       -2.12e-15   2.78e-02    0.00     1.00000
## MolWeight         -4.10e-01   1.37e-01   -2.99     0.00301 **
## NumAtoms           1.44e+00   9.58e-01    1.51     0.13336
## NumNonHAtoms       3.88e+00   6.53e-01    5.93 0.000000009 ***
## NumBonds          -3.76e+00   1.22e+00   -3.09     0.00223 **
## NumMultBonds      -3.39e+00   9.44e-01   -3.60     0.00039 ***
## NumRotBonds       -1.08e-01   6.52e-02   -1.65     0.09947 .
## NumDblBonds        2.79e-01   1.87e-01    1.49     0.13673
## NumAromaticBonds   2.51e+00   8.77e-01    2.86     0.00457 **
## NumCarbon         -1.08e+00   3.33e-01   -3.25     0.00131 **
## NumSulfer          1.09e-01   1.36e-01    0.80     0.42405
## NumChlorine       -1.98e-01   1.11e-01   -1.79     0.07528 .
## NumHalogen        -9.48e-01   2.00e-01   -4.73 0.000003595 ***
## HydrophilicFactor  1.03e-01   7.69e-02    1.34     0.18050
## SurfaceArea1       5.31e-01   2.35e-01    2.26     0.02462 *
## SurfaceArea2      -9.31e-01   3.09e-01   -3.01     0.00285 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.47 on 269 degrees of freedom
## Multiple R-squared:  0.791,  Adjusted R-squared:  0.779
## F-statistic: 67.9 on 15 and 269 DF,  p-value: <2e-16
```

Ridge Regression Preparation

- In order to use ridge regression, we need to separate our training data into a predictor matrix and a response vector:

Ridge Regression Preparation

- In order to use ridge regression, we need to separate our training data into a predictor matrix and a response vector:

```
x<-model.matrix(Solubility ~., data = solTrain)[,-1]
y<-solTrain$Solubility
```

Ridge Regression Preparation

- In order to use ridge regression, we need to separate our training data into a predictor matrix and a response vector:

```
x<-model.matrix(Solubility ~., data = solTrain)[,-1]
y<-solTrain$Solubility
```

- The `model.matrix` function creates a `matrix` of predictors and converts all categorical variables to dummy variables
- The `[,-1]` code selects all columns of the model matrix except the 1st (which corresponds to the intercept)

Ridge Regression Preparation

- In order to use ridge regression, we need to separate our training data into a predictor matrix and a response vector:

```
x<-model.matrix(Solubility ~., data = solTrain)[,-1]
y<-solTrain$Solubility
```

- The `model.matrix` function creates a `matrix` of predictors and converts all categorical variables to dummy variables
- The `[,-1]` code selects all columns of the model matrix except the 1st (which corresponds to the intercept)
- We also create vector `grid` of suitable tuning parameters λ.

```
grid = 10^(seq( -5, 5, length = 100))
head(grid)
```

```
## [1] 0.00001000 0.00001262 0.00001592 0.00002009 0.00002535 0.00003199
```

Ridge Regression Preparation

- In order to use ridge regression, we need to separate our training data into a predictor matrix and a response vector:

```
x<-model.matrix(Solubility ~., data = solTrain)[,-1]
y<-solTrain$Solubility
```

- The `model.matrix` function creates a `matrix` of predictors and converts all categorical variables to dummy variables
- The `[,-1]` code selects all columns of the model matrix except the 1st (which corresponds to the intercept)
- We also create vector `grid` of suitable tuning parameters λ.

```
grid = 10^(seq( -5, 5, length = 100))
head(grid)
```

```
## [1] 0.00001000 0.00001262 0.00001592 0.00002009 0.00002535 0.00003199
```

- The grid of values should be changed depending on the problem at hand.

The `glmnet` package

- We use the `glmnet` function in the `glmnet` package in order to perform Ridge Regression for a variety of values of the tuning parameter λ.

The glmnet package

- We use the `glmnet` function in the `glmnet` package in order to perform Ridge Regression for a variety of values of the tuning parameter λ.

```
library(glmnet)
ridge_mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

The glmnet package

- We use the glmnet function in the glmnet package in order to perform Ridge Regression for a variety of values of the tuning parameter λ.

```
library(glmnet)
ridge_mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

- The alpha argument in glmnet determines the type of penalty
  - alpha = 0 corresponds to Ridge Regression. alpha = 1 corresponds to LASSO (to be discussed next class)

The glmnet package

- We use the glmnet function in the glmnet package in order to perform Ridge Regression for a variety of values of the tuning parameter λ.

```
library(glmnet)
ridge_mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

- The alpha argument in glmnet determines the type of penalty
  - alpha = 0 corresponds to Ridge Regression. alpha = 1 corresponds to LASSO (to be discussed next class)
- By default, glmnet standardizes observations. To use unstandardized observations, add standardize = FALSE

The `glmnet` package

- We use the `glmnet` function in the `glmnet` package in order to perform Ridge Regression for a variety of values of the tuning parameter λ.

```
library(glmnet)
ridge_mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

- The `alpha` argument in `glmnet` determines the type of penalty
  - `alpha = 0` corresponds to Ridge Regression. `alpha = 1` corresponds to LASSO (to be discussed next class)
- By default, `glmnet` standardizes observations. To use unstandardized observations, add `standardize = FALSE`
- Here, we gave a specific range of values for the tuning parameter. But if no `lambda` value is supplied, the function will automatically select a range.

The glmnet package

- We use the glmnet function in the glmnet package in order to perform Ridge Regression for a variety of values of the tuning parameter λ.

```
library(glmnet)
ridge_mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

- The alpha argument in glmnet determines the type of penalty
  - alpha = 0 corresponds to Ridge Regression. alpha = 1 corresponds to LASSO (to be discussed next class)
- By default, glmnet standardizes observations. To use unstandardized observations, add standardize = FALSE
- Here, we gave a specific range of values for the tuning parameter. But if no lambda value is supplied, the function will automatically select a range.
- Remember! x needs to be the model matrix and y needs to be the response vector. glmnet does not use the formula syntax of lm.

Understanding output of `glmnet`

- Applying `coef` to the `glmnet` object gives a matrix of regression coefficients
  - one column for each value of lambda and one row for each predictor (and intercept)

Understanding output of glmnet

- Applying coef to the glmnet object gives a matrix of regression coefficients
  - one column for each value of lambda and one row for each predictor (and intercept)
- An example of several rows and columns:

```
coef(ridge_mod)[1:5,1:6]
```

```
## 5 x 6 sparse Matrix of class "dgCMatrix"
##                      s0          s1           s2           s3           s4
## (Intercept)  -2.77506730 -2.774979291 -2.774868231 -2.774728103 -2.774551305
## MolWeight    -0.00000026 -0.000000328 -0.000000414 -0.000000522 -0.000000659
## NumAtoms     -0.00000134 -0.000001691 -0.000002133 -0.000002692 -0.000003397
## NumNonHAtoms -0.00000354 -0.000004467 -0.000005636 -0.000007112 -0.000008974
## NumBonds     -0.00000131 -0.000001656 -0.000002090 -0.000002637 -0.000003327
##                       s5
## (Intercept)  -2.774328246
## MolWeight    -0.000000831
## NumAtoms     -0.000004286
## NumNonHAtoms -0.000011323
## NumBonds     -0.000004198
```

```
coef(ridge_mod)[1:5,95:100]
```

```
## 5 x 6 sparse Matrix of class "dgCMatrix"
##                   s94      s95      s96      s97      s98      s99
## (Intercept)   0.64413  0.64726  0.64976  0.65181  0.65347  0.65478
## MolWeight    -0.00806 -0.00806 -0.00806 -0.00806 -0.00806 -0.00806
## NumAtoms      0.01618  0.01758  0.01872  0.01969  0.02048  0.02110
## NumNonHAtoms  0.15747  0.15971  0.16150  0.16299  0.16419  0.16514
## NumBonds     -0.05314 -0.05411 -0.05491 -0.05557 -0.05612 -0.05655
```

Understanding output of glmnet

- In coef, columns are labeled by index of lambda (i.e. s_0, s_1, s_2). The actual values of lambda are stored in ridge_mod$lambda

```
ridge_mod$lambda
```

```
##  [1] 100000  79248  62803  49770  39442  31257  24771  19630  15557  12328
## [11]   9770   7743   6136   4863   3854   3054   2420   1918   1520   1205
```

Understanding output of glmnet

- In coef, columns are labeled by index of lambda (i.e. s_0, s_1, s_2). The actual values of lambda are stored in ridge_mod$lambda

```
ridge_mod$lambda
```

```
##  [1] 100000  79248  62803  49770  39442  31257  24771  19630  15557  12328
## [11]   9770   7743   6136   4863   3854   3054   2420   1918   1520   1205
```

- To find a particular value of lambda (i.e. s_{17}), subset the vector:

Understanding output of `glmnet`

- In `coef`, columns are labeled by index of lambda (i.e. s_0, s_1, s_2). The actual values of lambda are stored in `ridge_mod$lambda`

```
ridge_mod$lambda
```

```
##  [1] 100000  79248  62803  49770  39442  31257  24771  19630  15557  12328
## [11]   9770   7743   6136   4863   3854   3054   2420   1918   1520   1205
```

- To find a particular value of lambda (i.e. s_{17}), subset the vector:

```
ridge_mod$lambda[17]
```

```
## [1] 2420
```

Understanding output of glmnet

- In coef, columns are labeled by index of lambda (i.e. s_0, s_1, s_2). The actual values of lambda are stored in ridge_mod$lambda

```
ridge_mod$lambda
```

```
##  [1] 100000  79248  62803  49770  39442  31257  24771  19630  15557  12328
## [11]   9770   7743   6136   4863   3854   3054   2420   1918   1520   1205
```

- To find a particular value of lambda (i.e. s_{17}), subset the vector:

```
ridge_mod$lambda[17]
```

```
## [1] 2420
```

- And to get the corresponding model, subset columns of the coef matrix:

Understanding output of glmnet

- In `coef`, columns are labeled by index of lambda (i.e. s_0, s_1, s_2). The actual values of lambda are stored in `ridge_mod$lambda`

```
ridge_mod$lambda
```

```
##  [1] 100000  79248  62803  49770  39442  31257  24771  19630  15557  12328
## [11]   9770   7743   6136   4863   3854   3054   2420   1918   1520   1205
```

- To find a particular value of lambda (i.e. s_{17}), subset the vector:

```
ridge_mod$lambda[17]
```

```
## [1] 2420
```

- And to get the corresponding model, subset columns of the coef matrix:

```
coef(ridge_mod)[,17]
```

```
##       (Intercept)         MolWeight          NumAtoms      NumNonHAtoms
##       -2.76158736       -0.00001070       -0.00005508       -0.00014558
##          NumBonds       NumNonHBonds      NumMultBonds       NumRotBonds
##       -0.00005394       -0.00012659       -0.00014768       -0.00009046
##       NumDblBonds  NumAromaticBonds       NumHydrogen         NumCarbon
##       -0.00000461       -0.00014217       -0.00005565       -0.00017694
##       NumNitrogen         NumOxygen         NumSulfer       NumChlorine
##        0.00018026        0.00009812       -0.00038716       -0.00053786
##        NumHalogen          NumRings HydrophilicFactor      SurfaceArea1
##       -0.00054195       -0.00064385        0.00045126        0.00000904
##      SurfaceArea2
##        0.00000370
```

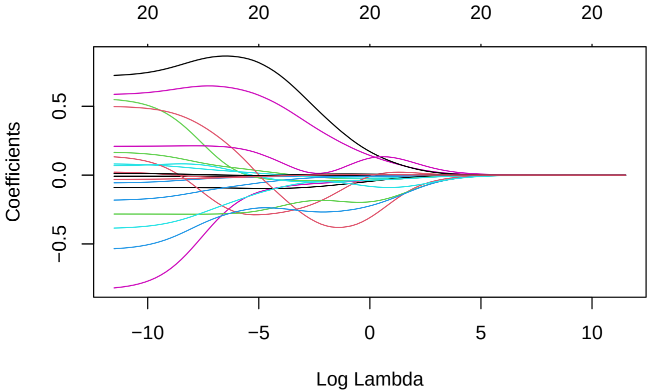
Coefficient Size

- What happens to coefficient size as λ changes?

Coefficient Size

- What happens to coefficient size as λ changes?

```
plot(ridge_mod, xvar = "lambda")
```



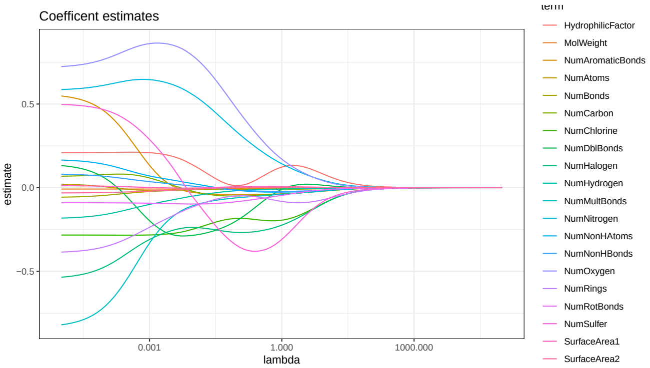
ggplot2 for glmnet

- A better plot using the broom package to tidy the output of glmnet for ggplot2:

ggplot2 for glmnet

- A better plot using the broom package to tidy the output of glmnet for ggplot2:

```
library(broom)
tidied <- tidy(ridge_mod) %>% filter(term != "(Intercept)")
ggplot(tidied, aes(lambda, estimate, group = term, color = term)) +
    geom_line() + scale_x_log10()+ theme_bw()+labs(title = "Coefficent estimates")
```



Penalized Regression Performance

- Which values of lambda produce best model among $\lambda = 0.001, 1, 1000$?

Penalized Regression Performance

- Which values of lambda produce best model among $\lambda = 0.001, 1, 1000$?
- The `glmnet` function already fit models, so we just need to make predictions:

Penalized Regression Performance

- Which values of lambda produce best model among $\lambda = 0.001, 1, 1000$?
- The `glmnet` function already fit models, so we just need to make predictions:

```
x_tst <- model.matrix(Solubility ~., data = solTest)[,-1]
preds<- predict(ridge_mod, s = c(0.001, 1, 1000), newx = x_tst) %>% as.data.frame()
head(preds)
```

```
##       s1     s2    s3
## 1 -2.164 -2.540 -2.78
## 2 -3.609 -3.983 -2.78
## 3 -2.171 -2.353 -2.78
## 4  0.318 -0.456 -2.75
## 5  0.519  0.182 -2.75
## 6 -3.856 -3.548 -2.78
```

Penalized Regression Performance

- Which values of lambda produce best model among $\lambda = 0.001, 1, 1000$?
- The glmnet function already fit models, so we just need to make predictions:

```
x_tst <- model.matrix(Solubility ~., data = solTest)[,-1]
preds<- predict(ridge_mod, s = c(0.001, 1, 1000), newx = x_tst) %>% as.data.frame()
head(preds)
```

```
##         s1     s2    s3
## 1 -2.164 -2.540 -2.78
## 2 -3.609 -3.983 -2.78
## 3 -2.171 -2.353 -2.78
## 4  0.318 -0.456 -2.75
## 5  0.519  0.182 -2.75
## 6 -3.856 -3.548 -2.78
```

```
get_rmse <- function(x){sqrt(mean((solTest$Solubility-x)^2))}
preds %>% summarize(across(everything(), get_rmse) )
```

```
##      s1    s2   s3
## 1 0.856 0.909 1.95
```

Penalized Regression Performance

- Which values of lambda produce best model among $\lambda = 0.001, 1, 1000$?
- The glmnet function already fit models, so we just need to make predictions:

```
x_tst <- model.matrix(Solubility ~., data = solTest)[,-1]
preds<- predict(ridge_mod, s = c(0.001, 1, 1000), newx = x_tst) %>% as.data.frame()
head(preds)
```

```
##        s1     s2    s3
## 1 -2.164 -2.540 -2.78
## 2 -3.609 -3.983 -2.78
## 3 -2.171 -2.353 -2.78
## 4  0.318 -0.456 -2.75
## 5  0.519  0.182 -2.75
## 6 -3.856 -3.548 -2.78
```

```
get_rmse <- function(x){sqrt(mean((solTest$Solubility-x)^2))}
preds %>% summarize(across(everything(), get_rmse) )
```

```
##      s1    s2   s3
## 1 0.856 0.909 1.95
```

- But how do we find the **best** value of λ?

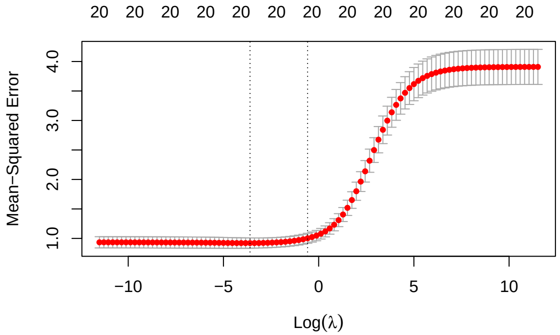
Cross Validation and `glmnet`

- We use the `cv.glmnet` function to perform cross-validation to compare MSE across all values of λ

Cross Validation and glmnet

- We use the cv.glmnet function to perform cross-validation to compare MSE across all values of λ

```
set.seed(1010)
my_cv<-cv.glmnet(x, y, alpha = 0, lambda = grid, nfolds = 10)
plot(my_cv)
```



Best Lambda

- The `cv.glmnet` object records the value of lambda that...
  - Has minimum error (`lambda.min`)
  - Is largest with error within 1 st. dev of minimum error (`lambda.1se`)

Best Lambda

- The `cv.glmnet` object records the value of lambda that...
  - Has minimum error (`lambda.min`)
  - Is largest with error within 1 st. dev of minimum error (`lambda.1se`)
  - Why is `lambda.1se` useful?

Best Lambda

- The `cv.glmnet` object records the value of lambda that...
  - Has minimum error (`lambda.min`)
  - Is largest with error within 1 st. dev of minimum error (`lambda.1se`)
  - Why is `lambda.1se` useful?

```
best_L<-my_cv$lambda.min
best_L
```

```
## [1] 0.0272
```

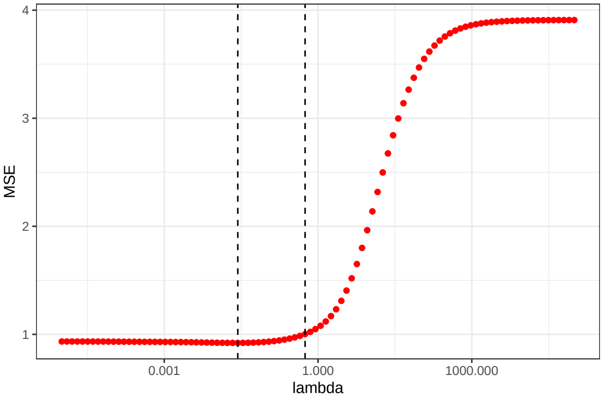
```
reg_L <-my_cv$lambda.1se
reg_L
```

```
## [1] 0.559
```

Better Plots

- As before, we can obtain a better plot using broom

```
tidied <- tidy(my_cv)
ggplot(tidied, aes(x = lambda, y = estimate))+geom_point( color = "red")+
  scale_x_log10()+theme_bw()+labs(y = "MSE")+
  geom_vline(xintercept = best_L, linetype = "dashed" )+
  geom_vline(xintercept = reg_L, linetype = "dashed")
```



Overall Performance

- Let's compare performance for: the full model, the best 15 model, ridge regression with $\lambda = 0.027$, and ridge regression with $\lambda = 0.559$.

Overall Performance

- Let's compare performance for: the full model, the best 15 model, ridge regression with $\lambda = 0.027$, and ridge regression with $\lambda = 0.559$.

```
full_mod <- lm(Solubility ~ ., data = solTrain)
preds <- data.frame(
  full = predict(full_mod, solTest),
  best_15 = predict(best15, solTest),
  rr_min = c(predict(ridge_mod, s = best_L, newx = x_tst)),
  rr_1se = c(predict(ridge_mod, s = reg_L, newx = x_tst))
)
preds %>% summarize(across(everything(),get_rmse))
```

```
##    full best_15 rr_min rr_1se
## 1 0.868   0.869  0.859  0.883
```

- Ridge Regression wins!