Ridge Regression in R

Nate Wells

Math 243: Stat Learning

October 13th, 2021

Outline

In today's class, we will...

- Implement Ridge Regression in R

Section 1

Ridge Regression in R

Ridge Regression

- To perform **Ridge Regression**, we find coefficients β in the linear model that minimize

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

Ridge Regression

- To perform **Ridge Regression**, we find coefficients β in the linear model that minimize

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

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

Ridge Regression

- To perform **Ridge Regression**, we find coefficients β in the linear model that minimize

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

- 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

- To perform **Ridge Regression**, we find coefficients β in the linear model that minimize

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

- 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.

Effect of Scale

- Suppose $\hat{y} = 1 + 0.01x_1 + 20x_2$ is the best fitting linear model for Y using X_1 and X_2, and that both are statistically significant.

Effect of Scale

- Suppose $\hat{y} = 1 + 0.01x_1 + 20x_2$ is the best fitting linear model for Y using X_1 and X_2, and that both are statistically significant.
  - Are we justified in saying that X_2 is a more important predictor than X_1?

Effect of Scale

- Suppose $\hat{y} = 1 + 0.01x_1 + 20x_2$ is the best fitting linear model for Y using X_1 and X_2, and that both are statistically significant.
  - Are we justified in saying that X_2 is a more important predictor than X_1?
  - What if $\mathrm{sd}(x_1) = 10000$ and $\mathrm{sd}(x_2) = .1$?
- Suppose we first standardize X_1 and X_2 by subtracting off their means and dividing by their standard deviations:

$$Z_1 = \frac{X_1 - \mu_1}{\sigma_1} \qquad Z_2 = \frac{X_2 - \mu_2}{\sigma_2}$$

Effect of Scale

- Suppose $\hat{y} = 1 + 0.01x_1 + 20x_2$ is the best fitting linear model for Y using X_1 and X_2, and that both are statistically significant.
  - Are we justified in saying that X_2 is a more important predictor than X_1?
  - What if $\text{sd}(x_1) = 10000$ and $\text{sd}(x_2) = .1$?
- Suppose we first standardize X_1 and X_2 by subtracting off their means and dividing by their standard deviations:

$$Z_1 = \frac{X_1 - \mu_1}{\sigma_1} \qquad Z_2 = \frac{X_2 - \mu_2}{\sigma_2}$$

- If we build a model and find $\hat{y} = 1 + 0.01z_1 + 20z_2$, where Z_1 and Z_2 are standardized, are we now justified in saying that Z_2 is more important than Z_1?

Effect of Scale

- Suppose $\hat{y} = 1 + 0.01x_1 + 20x_2$ is the best fitting linear model for Y using X_1 and X_2, and that both are statistically significant.
  - Are we justified in saying that X_2 is a more important predictor than X_1?
  - What if $\text{sd}(x_1) = 10000$ and $\text{sd}(x_2) = .1$?
- Suppose we first standardize X_1 and X_2 by subtracting off their means and dividing by their standard deviations:

$$Z_1 = \frac{X_1 - \mu_1}{\sigma_1} \qquad Z_2 = \frac{X_2 - \mu_2}{\sigma_2}$$

- If we build a model and find $\hat{y} = 1 + 0.01z_1 + 20z_2$, where Z_1 and Z_2 are standardized, are we now justified in saying that Z_2 is more important than Z_1?
  - Assuming both are statistically significant, we are probably justified.

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 by a value c just leads to scaling the estimate by $1/c$.

Scale

- The coefficients in the least squares regression equation are **scale-equivalent**
  - That is, scaling a predictor by a value c just leads to scaling the estimate 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 by a value c just leads to scaling the estimate by $1/c$.
  - The predicted value is the same, regardless of scale.
  - Therefore, rescaling predictors *does not* change the fit of the model (RSS is the same)

Scale

- The coefficients in the least squares regression equation are **scale-equivalent**
  - That is, scaling a predictor by a value c just leads to scaling the estimate by $1/c$.
  - The predicted value is the same, regardless of scale.
  - Therefore, rescaling predictors *does not* change the fit of the model (RSS is the same)
  - Suppose $y = 1 + 0.01x_1 + 20x_2$, $\sigma_1 = 10000, \sigma_2 = 0.1$, and both x_1, x_2 have mean 0.
  - After rescaling, $z_1 = \frac{x_1}{10000}, z_2 = \frac{x_2}{0.1}$ and the linear model is
$$y = 100z_1 + 2z_2$$
- However, for Ridge Regression, coefficient estimates can change depending on scale.

Scale

- The coefficients in the least squares regression equation are **scale-equivalent**
  - That is, scaling a predictor by a value c just leads to scaling the estimate by $1/c$.
  - The predicted value is the same, regardless of scale.
  - Therefore, rescaling predictors *does not* change the fit of the model (RSS is the same)
  - Suppose $y = 1 + 0.01x_1 + 20x_2$, $\sigma_1 = 10000, \sigma_2 = 0.1$, and both x_1, x_2 have mean 0.
  - After rescaling, $z_1 = \frac{x_1}{10000}, z_2 = \frac{x_2}{0.1}$ and the linear model is
$$y = 100z_1 + 2z_2$$
- However, for Ridge Regression, coefficient estimates can change depending on scale.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2 = \lambda(0.01^2 + 20^2)$

Scale

- The coefficients in the least squares regression equation are **scale-equivalent**
  - That is, scaling a predictor by a value c just leads to scaling the estimate by $1/c$.
  - The predicted value is the same, regardless of scale.
  - Therefore, rescaling predictors *does not* change the fit of the model (RSS is the same)
  - Suppose $y = 1 + 0.01x_1 + 20x_2$, $\sigma_1 = 10000, \sigma_2 = 0.1$, and both x_1, x_2 have mean 0.
  - After rescaling, $z_1 = \frac{x_1}{10000}, z_2 = \frac{x_2}{0.1}$ and the linear model is
$$y = 100z_1 + 2z_2$$
- However, for Ridge Regression, coefficient estimates can change depending on scale.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2 = \lambda(0.01^2 + 20^2)$
  - Which models will ridge regression favor?

Scale

- The coefficients in the least squares regression equation are **scale-equivalent**
  - That is, scaling a predictor by a value c just leads to scaling the estimate by $1/c$.
  - The predicted value is the same, regardless of scale.
  - Therefore, rescaling predictors *does not* change the fit of the model (RSS is the same)
  - Suppose $y = 1 + 0.01x_1 + 20x_2$, $\sigma_1 = 10000, \sigma_2 = 0.1$, and both x_1, x_2 have mean 0.
  - After rescaling, $z_1 = \frac{x_1}{10000}, z_2 = \frac{x_2}{0.1}$ and the linear model is
$$y = 100z_1 + 2z_2$$
- However, for Ridge Regression, coefficient estimates can change depending on scale.
  - Recall the shrinkage penalty is $\lambda \sum_{i=1}^{2} \beta_i^2 = \lambda(0.01^2 + 20^2)$
  - Which models will ridge regression favor?
- Ridge regression is most effective if predictors are standardized first.

Solubility

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

- But suppose we only have a fraction of the data to work with. . .

Solubility

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

- But suppose we only have a fraction of the data to work with. . .

```
set.seed(1013)
library(AppliedPredictiveModeling)
data(solubility)
solTest <- data.frame(solTestX, Solubility = solTestY) %>% sample_frac(.3)
solTrain <- data.frame(solTrainX, Solubility =  solTrainY) %>% sample_frac(.3)
solTest <- solTest %>% dplyr::select(!starts_with("FP"))
solTrain <- solTrain %>%  dplyr::select(!starts_with("FP"))
```

Solubility

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

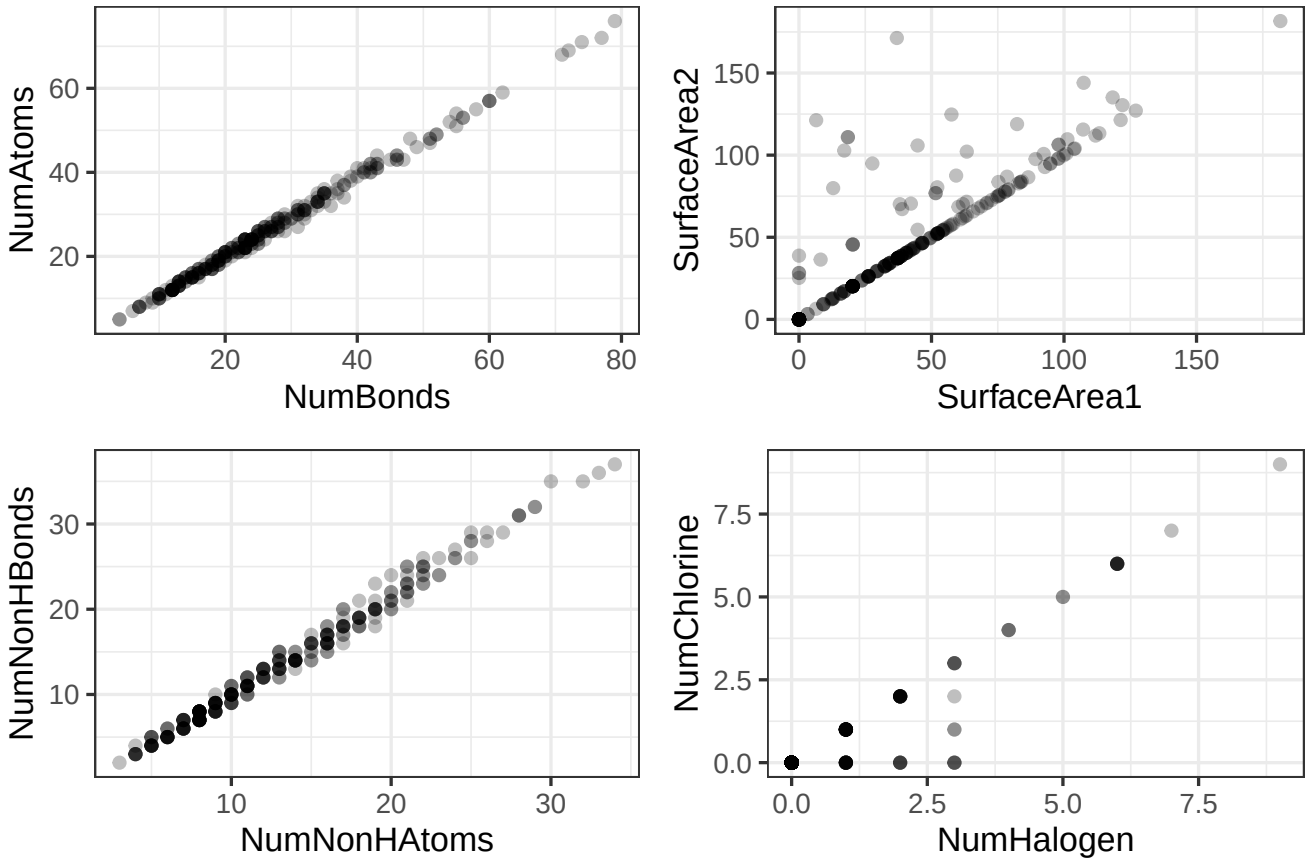
- But suppose we only have a fraction of the data to work with. . .

```
set.seed(1013)
library(AppliedPredictiveModeling)
data(solubility)
solTest <- data.frame(solTestX, Solubility = solTestY) %>% sample_frac(.3)
solTrain <- data.frame(solTrainX, Solubility =  solTrainY) %>% sample_frac(.3)
solTest <- solTest %>% dplyr::select(!starts_with("FP"))
solTrain <- solTrain %>%  dplyr::select(!starts_with("FP"))
```

- Our goal is to predict solubility using the 20 chemical structure attributes.

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 0.754869
```

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
## -2.93489 -0.57479  0.08137  0.60908  1.88354
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.313835   0.297830   1.054 0.292947
## MolWeight         -0.008262   0.002760  -2.994 0.003010 **
## NumAtoms           0.224406   0.149054   1.506 0.133360
## NumNonHAtoms       1.219121   0.205416   5.935 9.03e-09 ***
## NumBonds          -0.547814   0.177397  -3.088 0.002225 **
## NumMultBonds      -1.366339   0.380031  -3.595 0.000385 ***
## NumRotBonds       -0.088494   0.053531  -1.653 0.099471 .
## NumDblBonds        0.472754   0.316741   1.493 0.136725
## NumAromaticBonds   0.993862   0.347495   2.860 0.004567 **
## NumCarbon         -0.405111   0.124706  -3.249 0.001307 **
## NumSulfer          0.356621   0.445427   0.801 0.424053
## NumChlorine       -0.288069   0.161321  -1.786 0.075276 .
## NumHalogen        -1.326534   0.280328  -4.732 3.59e-06 ***
## HydrophilicFactor  0.207625   0.154632   1.343 0.180501
## SurfaceArea1       0.033006   0.014604   2.260 0.024616 *
## SurfaceArea2      -0.050940   0.016919  -3.011 0.002853 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9271 on 269 degrees of freedom
## Multiple R-squared:  0.791,  Adjusted R-squared:  0.7794
## F-statistic: 67.88 on 15 and 269 DF,  p-value: < 2.2e-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()
```

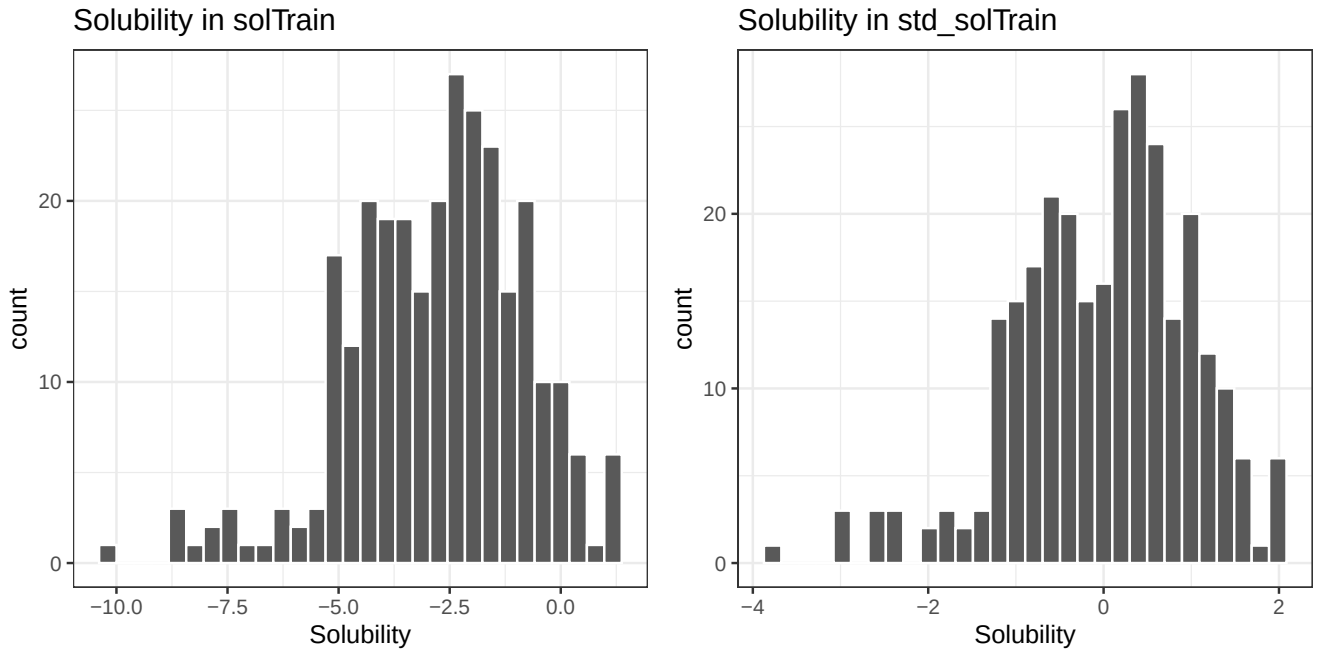
- A quick verification:

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
## -1.48705 -0.29123  0.04123  0.30861  0.95435
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -1.697e-15  2.782e-02   0.000 1.000000
## MolWeight         -4.102e-01  1.370e-01  -2.994 0.003010 **
## NumAtoms           1.442e+00  9.581e-01   1.506 0.133360
## NumNonHAtoms       3.877e+00  6.532e-01   5.935 9.03e-09 ***
## NumBonds          -3.765e+00  1.219e+00  -3.088 0.002225 **
## NumMultBonds      -3.394e+00  9.439e-01  -3.595 0.000385 ***
## NumRotBonds       -1.078e-01  6.523e-02  -1.653 0.099471 .
## NumDblBonds        2.788e-01  1.868e-01   1.493 0.136725
## NumAromaticBonds   2.508e+00  8.767e-01   2.860 0.004567 **
## NumCarbon         -1.083e+00  3.334e-01  -3.249 0.001307 **
## NumSulfer          1.087e-01  1.358e-01   0.801 0.424053
## NumChlorine       -1.977e-01  1.107e-01  -1.786 0.075276 .
## NumHalogen        -9.479e-01  2.003e-01  -4.732 3.59e-06 ***
## HydrophilicFactor  1.032e-01  7.689e-02   1.343 0.180501
## SurfaceArea1       5.306e-01  2.348e-01   2.260 0.024616 *
## SurfaceArea2      -9.311e-01  3.092e-01  -3.011 0.002853 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4697 on 269 degrees of freedom
## Multiple R-squared:  0.791,  Adjusted R-squared:  0.7794
## F-statistic: 67.88 on 15 and 269 DF,  p-value: < 2.2e-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] 1.000000e-05 1.261857e-05 1.592283e-05 2.009233e-05 2.535364e-05
## [6] 3.199267e-05
```

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] 1.000000e-05 1.261857e-05 1.592283e-05 2.009233e-05 2.535364e-05
## [6] 3.199267e-05
```

- 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 (Friday's 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 (Friday's 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 (Friday's 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 (Friday's 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        s5
## (Intercept)  -2.78e+00 -2.77e+00 -2.77e+00 -2.77e+00 -2.77e+00 -2.77e+00
## MolWeight    -2.60e-07 -3.28e-07 -4.14e-07 -5.22e-07 -6.59e-07 -8.31e-07
## NumAtoms     -1.34e-06 -1.69e-06 -2.13e-06 -2.69e-06 -3.40e-06 -4.29e-06
## NumNonHAtoms -3.54e-06 -4.47e-06 -5.64e-06 -7.11e-06 -8.97e-06 -1.13e-05
## NumBonds     -1.31e-06 -1.66e-06 -2.09e-06 -2.64e-06 -3.33e-06 -4.20e-06
```

```
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
##         -2.76e+00         -1.07e-05         -5.51e-05         -1.46e-04
##          NumBonds      NumNonHBonds      NumMultBonds       NumRotBonds
##         -5.39e-05         -1.27e-04         -1.48e-04         -9.05e-05
##       NumDblBonds  NumAromaticBonds       NumHydrogen         NumCarbon
##         -4.61e-06         -1.42e-04         -5.57e-05         -1.77e-04
##       NumNitrogen         NumOxygen         NumSulfer       NumChlorine
##          1.80e-04          9.81e-05         -3.87e-04         -5.38e-04
##        NumHalogen          NumRings HydrophilicFactor      SurfaceArea1
##         -5.42e-04         -6.44e-04          4.51e-04          9.04e-06
##      SurfaceArea2
##          3.70e-06
```

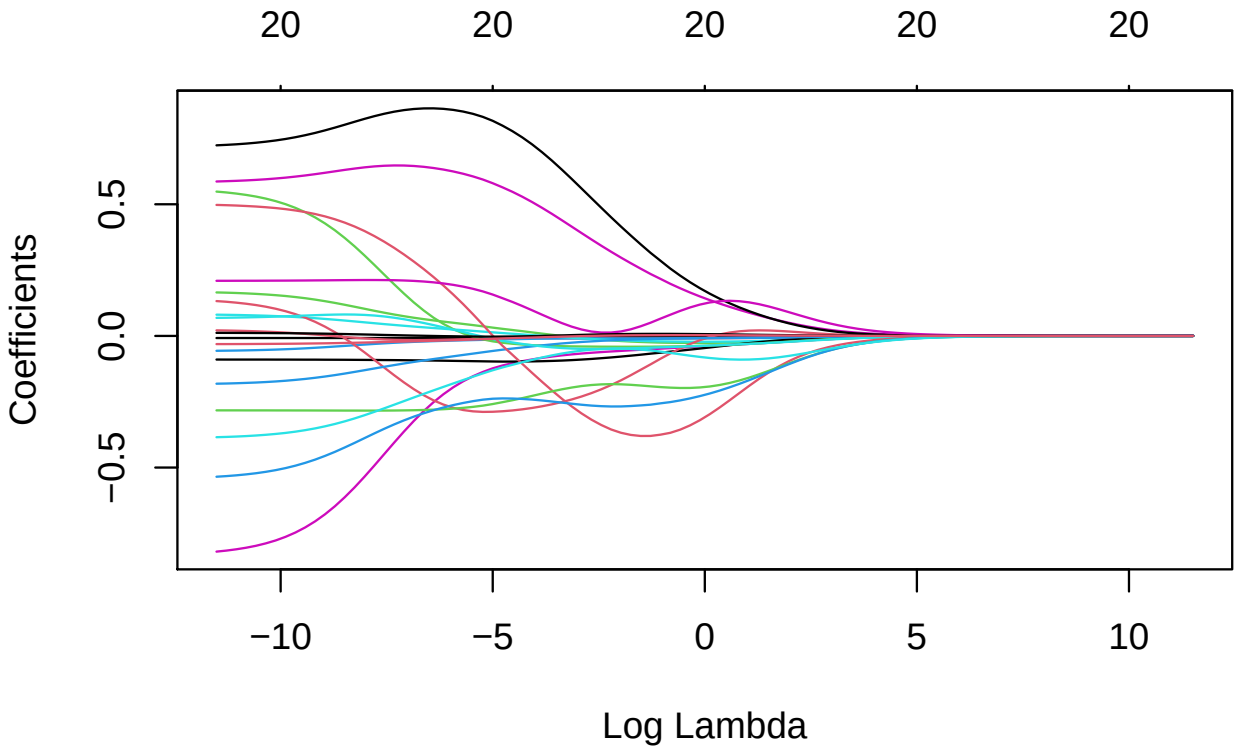
Coefficient Size

- What happens to coefficient size as λ changes?

Coefficient Size

- What happens to coefficient size as λ changes?

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



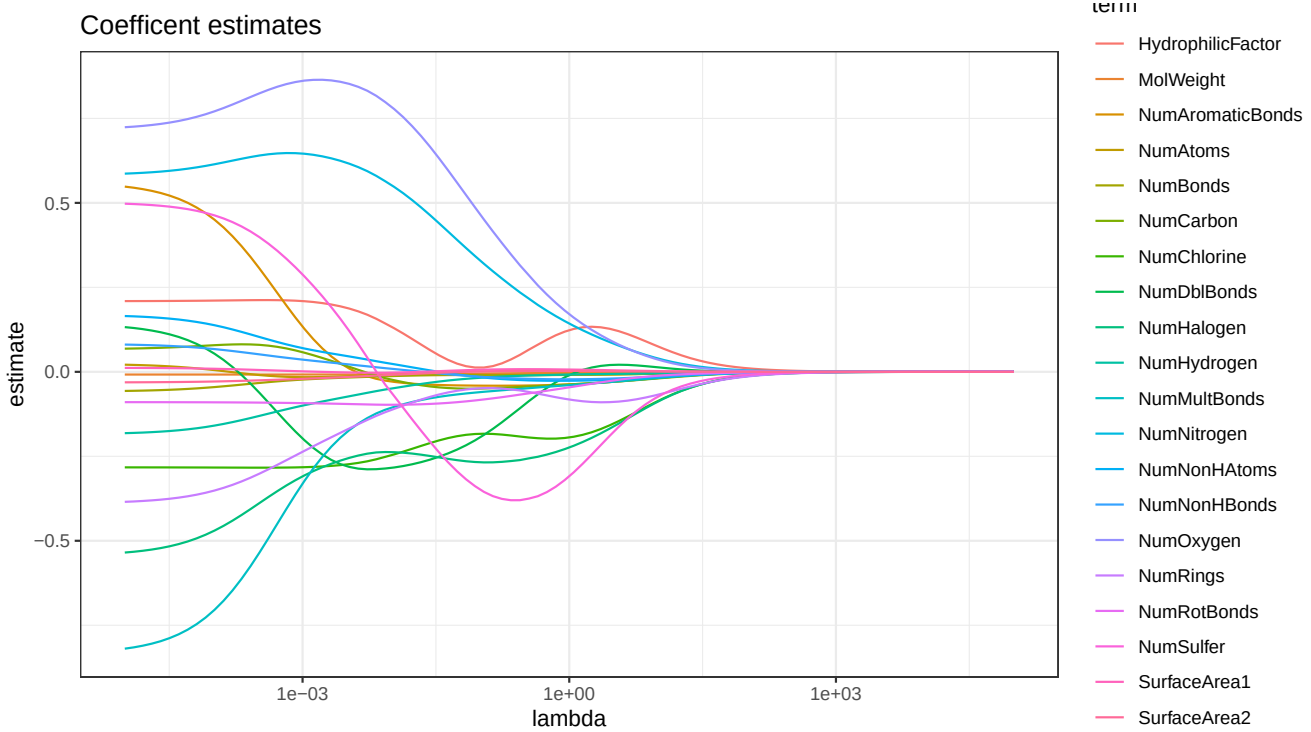
ggplot2 for glmnet

- 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? $\lambda = 0.001, 1, 1000$?

Penalized Regression Performance

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

Penalized Regression Performance

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

```
##         1      2     3
## 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? $\lambda = 0.001, 1, 1000$?
- 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)
```

```
##        1      2     3
## 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_mse <- function(x){mean((solTest$Solubility-x)^2)}
preds %>% summarize(across(everything(), get_mse) )
```

```
##       1     2    3
## 1 0.733 0.827 3.79
```

Penalized Regression Performance

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

```
##        1      2     3
## 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_mse <- function(x){mean((solTest$Solubility-x)^2)}
preds %>% summarize(across(everything(), get_mse) )
```

```
##       1     2    3
## 1 0.733 0.827 3.79
```

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

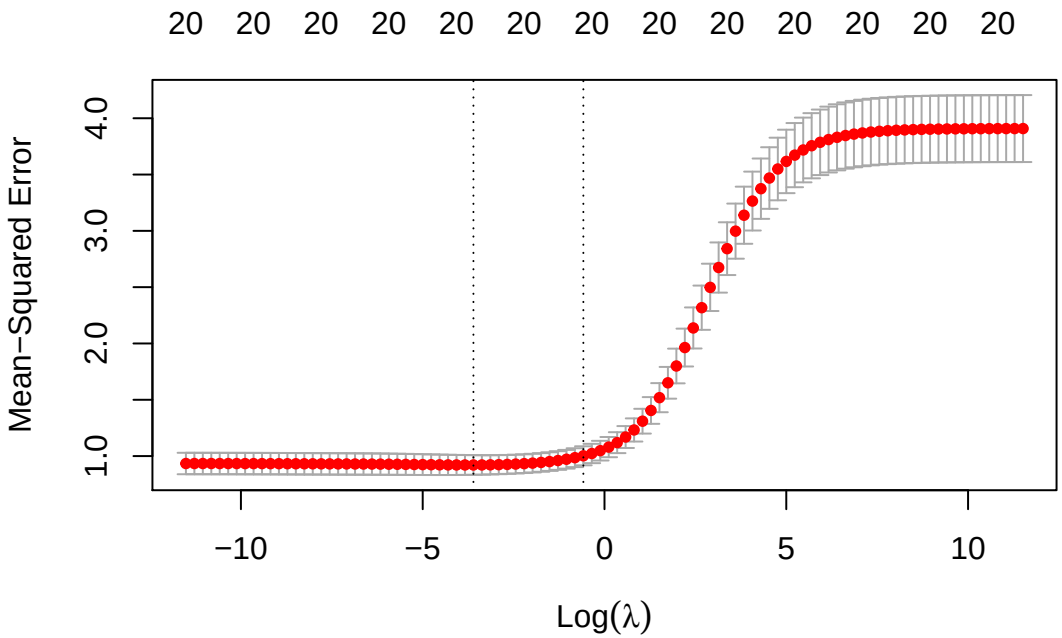
Cross Validation and `glmnet`

- 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
```

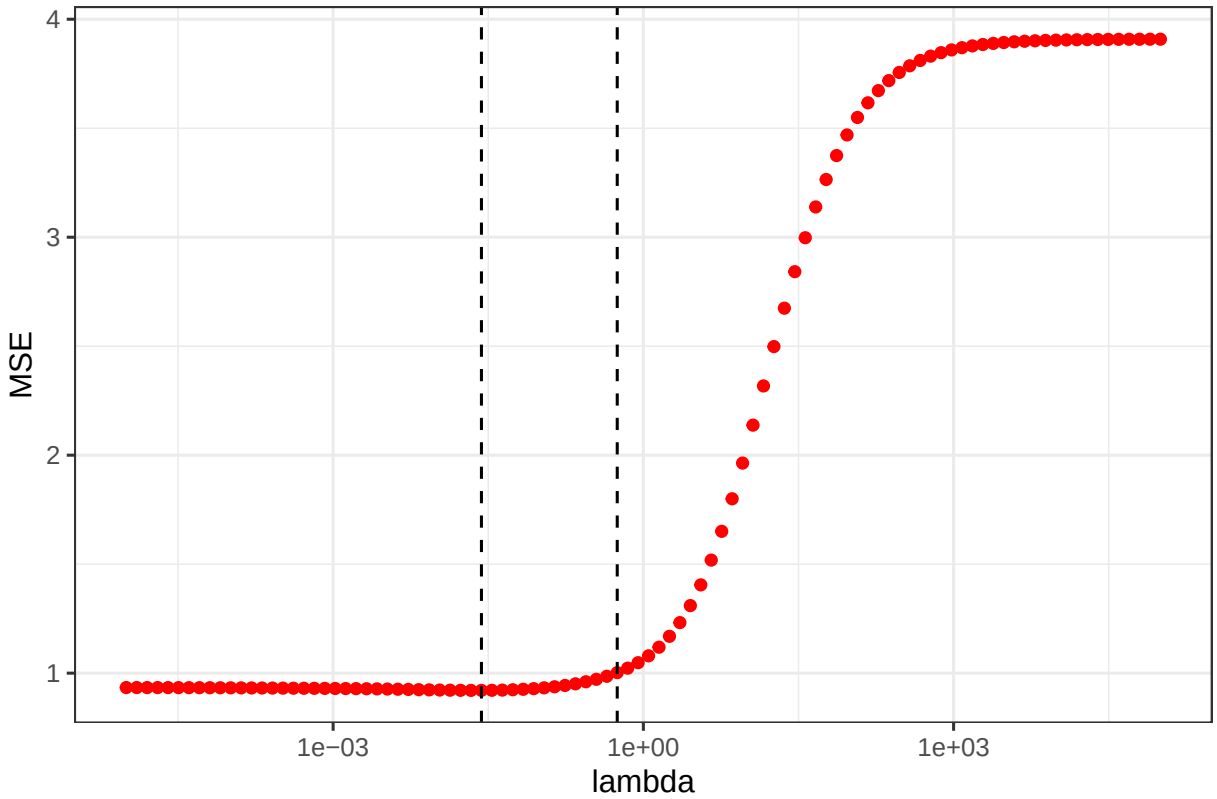
```
reg_L <-my_cv$lambda.1se
reg_L
```

```
## [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_mse))
```

```
##    full best_15 rr_min rr_1se
## 1 0.753   0.755  0.739   0.78
```

- Ridge Regression wins!