Ridge regression

Gerko Vink

Data Science and Predictive Machine Learning

Packages used in this lecture

```
library(magrittr) # pipes
library(dplyr)    # data manipulation
library(ggplot2)  # plotting
library(GGally)   # ggplot's friend
library(mice)     # boys data
library(mvtnorm)  # multivariate normal fun
library(glmnet)   # penalized regression
library(plotmo)   # informative plots
```

This session

Goal: learning to apply ridge regression

To reach the goal we need to

1. Refresh our memory
2. Understand how regression works
3. Understand the shortcomings of OLS
4. PLug in the ridge penalty
5. Apply it in software

So far

At this point we have covered the following models for outcome Y and predictor matrix X

- Linear regression

$$Y = \alpha + \beta X + \varepsilon$$

- Logistic regression

$$Y = \mathbb{E}[Y] + \varepsilon, \quad \text{where} \quad \mathbb{E}[Y] = \alpha + \beta X$$

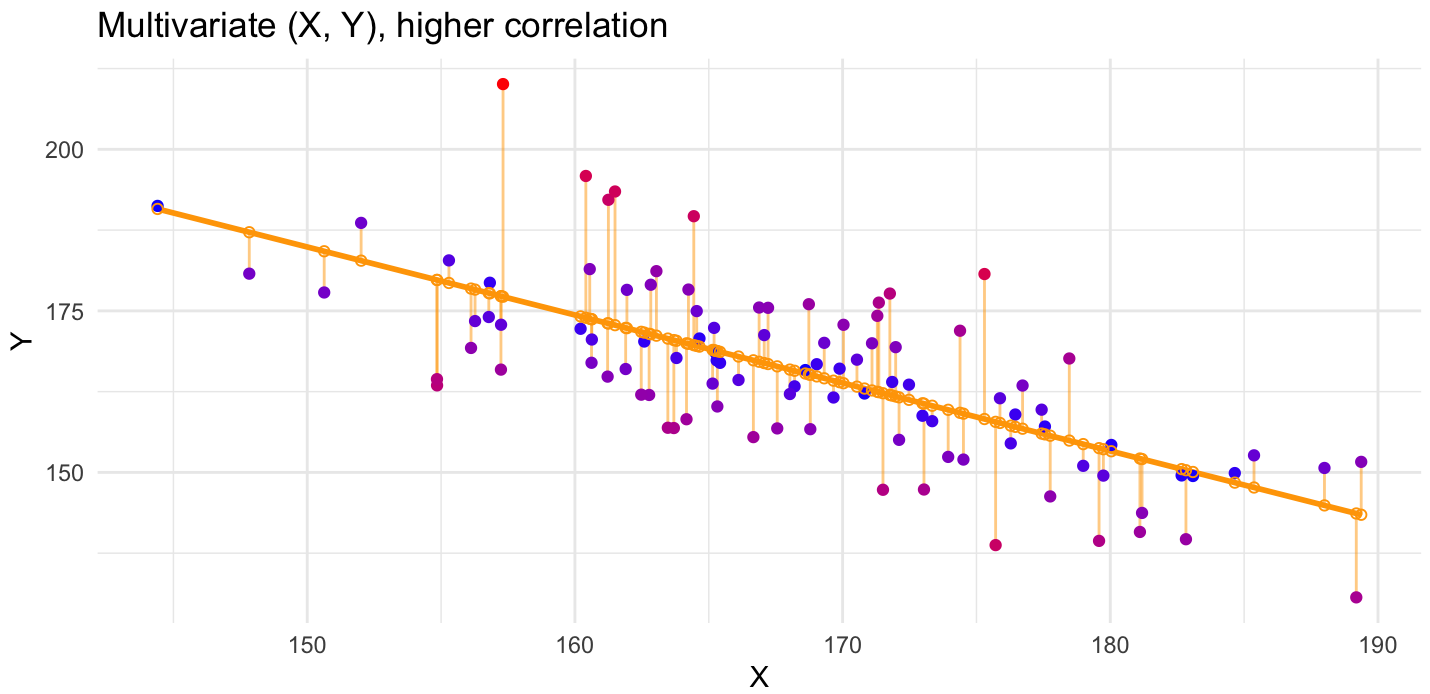
This would enabled us to change $\mathbb{E}[Y]$ such that $f(\mathbb{E}[Y])$ had a linear relation with X.

We used the `logit` link function to relate the binary outcome to the linear predictor space.

We did not

- formally define it all
- discuss the relation between X and Y under assumed linearity
- discuss how to mathematically regress X on Y
- discuss how vector β can be obtained, given Y and X
- discuss what happens when X does not behave

Least squares



So far we have been focusing on minimizing the squared deviations. This is equivalent to maximizing the likelihood.

Thus, for regression it holds that OLS = ML

Notation

Consider an scenario in which p features are measured for n cases.
The resulting $(n \times p)$-dimensional data matrix X matrix X is

$$X = (X_{*,1}| \dots |X_{*,p}) = \begin{bmatrix} X_{1,*} \\ \vdots \\ X_{n,*} \end{bmatrix} = \begin{bmatrix} X_{1,1} & \cdots & X_{1,p} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,p} \end{bmatrix}.$$

This matrix is called the *design matrix*.

```
##      (Intercept)   hgt    wgt   bmi   hc regeast regwest regsouth regcity
## 4752           1 158.9 49.100 19.44 59.0       0       0        0       1
## 5247           1 159.2 42.700 16.84 53.6       0       0        1       0
## 1882           1  81.8 10.535 15.74 46.7       0       0        0       0
## 5806           1 172.0 52.300 17.67 56.9       0       0        1       0
## 2125           1  90.2 13.000 15.97 49.5       0       1        0       0
## 1122           1  75.0  9.840 17.49 46.8       1       0        0       0
```

This is not the only information that we have. We also have another source of information about the cases and features in X: the *response variable* Y.

Aim

The aim is now to explain Y in **terms** of X through the functional relationship

$$Y_i = f(\mathbf{X}_{i,*}), \quad \text{where} \quad i = 1, \dots, n.$$

When we know nothing about the nature of the form of $f(.)$, a reasonable assumption would be to assume that the nature is linear.

Why is that reasonable?

When the nature is linear:

When X changes, Y also changes with a constant coefficient

The linear model

When X and Y are assumed to be linearly related, then

$$Y_i = \mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i = \beta_1 X_{i,1} + \cdots + \beta_p X_{i,p} + \varepsilon_i.$$

In the above additive model, the vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)'$ represents the *regression parameter* when regressing $\mathbf{X}$ on Y.

The nice thing is that each $\beta_j, j = 1, \ldots, p$ denotes the effect size of the jth column in $\mathbf{X}$ on the response Y.

- for each unit change in X_j, the realized change in the modeled response $\hat{Y}$ is equal to β.

The term ε_i represent the part of the response Y that is not explained by the functional model $\beta\mathbf{X}_{i,*}$ and is assumed to be random

- the functional relation $Y_i = f(\mathbf{X}_{i,*})$ is not assumed to be random
- the probability distribution of $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$
- all ε are independent (i.e. $\mathrm{Cov}(\varepsilon_i, \varepsilon_{i'}) = 0$ when $i \neq i'$)

Importance of ε

The random nature of ε_i has a strict implication: because ε_i is random, Y_i is also a random variable.

- Y_i is normally distributed because $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ and $\mathbf{X}_{i,*}\beta$ is a non-random scalar.
- in practice $\mathbf{Y}_i$ may not be exactly normally distributed -> but the above does imply normality for Y_i
- without ε_i all observed $\mathbf{Y}_i$ would need to be exactly on the regression line.
- without ε_i, our model needs to be perfect

Moments of $\mathbf{Y}_i$

The expectation for $\mathbf{Y}_i$ equals

$$\mathbb{E}[\mathbf{Y}_i] = \mathbb{E}[\mathbf{X}_{i,*}\beta] + \mathbb{E}[\varepsilon_i].$$

This is equivalent to

$$\mathbb{E}[\mathbf{Y}_i] = \mathbf{X}_{i,*}\beta,$$

because $\mathbb{E}[\varepsilon_i] = 0$.

The variance of $\mathbf{Y}_i$ is

$$\begin{aligned}
\mathrm{Var}[\mathbf{Y_i}] &= \mathbb{E}[(\mathbf{Y}_i - \mathbb{E}[\mathbf{Y}_i])^2] \\
&= \mathbb{E}[(\mathbf{X}_{i,*}\beta)^2 + 2\varepsilon_i\mathbf{X}_{i,*} + \varepsilon_i^2] - (\mathbf{X}_{i,*}\beta) \\
&= \mathbb{E}[\varepsilon_i^2] \\
&= \sigma_\varepsilon^2.
\end{aligned}$$

Hence, $Y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\beta, \sigma_\varepsilon^2)$.

Estimating

Remember the linear regression model

$$Y_i = \mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i = \beta_1 X_{i,1} + \cdots + \beta_1 X_{i,p} + \varepsilon_i.$$

In this model the values for Y_i and $\mathbf{X}_{i,*}$ are known. What remains unknown are the parameters β and σ_ε^2.

We know that $Y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\beta, \sigma_\varepsilon^2)$, so the density is

$$f_{Y_i}(y_i) = (2\pi\sigma_\varepsilon^2)^{-1/2}\exp[-(y_i - \mathbf{X}_{i,*}\beta)^2/2\sigma_\varepsilon^2].$$

So far we have expressed everything in terms of the least-squares (i.e. minimizing σ_ε^2), but we can also express the linear model in terms of maximizing the likelihood. The likelihood is:

$$L(\mathbf{Y}, \mathbf{X}; \beta, \sigma_\varepsilon^2) = \prod_{i=1}^{n}(\sigma_\varepsilon\sqrt{2\pi})^{-1}\exp[\mathbf{Y}_i - \mathbf{X}_{i,*}\beta)^2/2\sigma_\varepsilon^2].$$

The log-likelihood

The maximum of the likelihood coincides with the maximum of the logarithm of the likelihood. In Maximum Likelihood (ML) estimation we therefore aim to maximize the log-likelihood:

$$\mathcal{L}(\mathbf{Y}, \mathbf{X}; \beta, \sigma_\varepsilon^2) = \log[\mathbf{Y}, \mathbf{X}; \beta, \sigma_\varepsilon^2] = -n\log(\sqrt{2\pi\sigma_\varepsilon}) - \frac{1}{2}\sigma_\varepsilon^{-2}\Sigma_{i=1}^{n}(y_i - \mathbf{X}_{i,*}\beta)^2.$$

By equating $\sigma_\varepsilon^{-2}\Sigma_{i=1}^{n}(y_i - \mathbf{X}_{i,*}\beta)^2 = (\mathbf{Y} - \mathbf{X}\beta)'(\mathbf{Y} - \mathbf{X}\beta)$ we can find the maximum of the log-likelihood by taking its derivative with respect to β:

$$\frac{\partial}{\partial\beta}\mathcal{L}(\mathbf{Y}, \mathbf{X}; \beta, \sigma_\varepsilon^2) = \frac{1}{2}\sigma_\varepsilon^{-2}\mathbf{X}'(\mathbf{Y} - \mathbf{X}\beta).$$

Equating this derivative to zero yields

$$\mathbf{X}'\mathbf{X}\beta = \mathbf{X}'\mathbf{Y},$$

which gives the ML estimator of the regression parameter as:

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \quad \text{with} \quad \text{Var}(\hat{\beta}) = \sigma_\varepsilon^2(\mathbf{X}'\mathbf{X})^{-1}.$$

What is the problem?

Everything we have discussed thus far assumed that $(\mathbf{X}'\mathbf{X})^{-1}$ is well-defined. But that is not always the case!

This means that

$$\begin{aligned}\hat{\beta} &= (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \\ \text{Var}(\hat{\beta}) &= \sigma_\varepsilon^2(\mathbf{X}'\mathbf{X})^{-1},\end{aligned}$$

can only be defined when $(\mathbf{X}'\mathbf{X})^{-1}$ exists.

$(\mathbf{X}'\mathbf{X})^{-1}$ is problematic when

- predictors are highly linearly related. We call this *multicollinearity*
- predictors are fully linearly dependent. We call this *supercollinearity*

Create some data

```
set.seed(123) # to allow reproduction
data <- rmvnorm(100,
                mean = c(50, 100),
                sigma = matrix(c(14, 21, 21, 32), nrow = 2, ncol = 2)) %>%
  set_colnames(c("X1", "X2")) %>%
  as_tibble() %>%
  mutate(epsilon = rnorm(100, mean = 0, sd = 15),
         Y =  2 * X1 + 4* X2 + epsilon)

var(data)
```

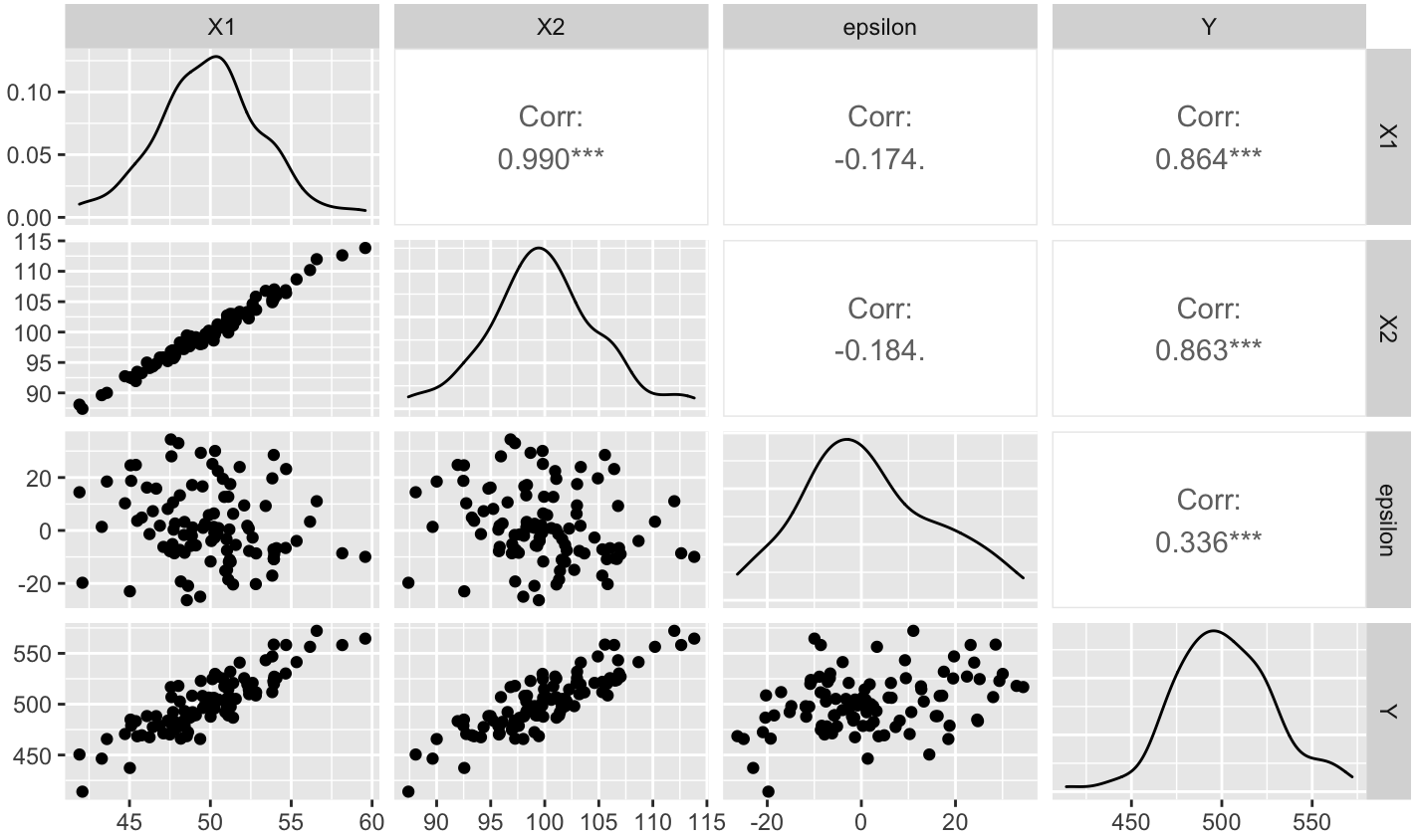
```
##                X1        X2    epsilon         Y
## X1      10.735808  16.41276  -8.112252  79.01041
## X2      16.412762  25.59313 -13.249393 121.94865
## epsilon -8.112252 -13.24939 203.010760 133.78868
## Y       79.010411 121.94865 133.788683 779.60411
```

```
colMeans(data)
```

```
##         X1         X2    epsilon          Y
##  49.946209  99.906016   1.806977 501.323459
```

Inspect our data

```
data %>%
  ggpairs() # from GGally
```



By ourselves

```
Y <- data$Y %>% as.matrix
X <- data %>% select(X1, X2) %>% as.matrix %>% cbind(1, .)
solve(t(X) %*% X) %*% t(X) %*% Y
```

```
##         [,1]
##    79.341826
## X1  3.827800
## X2  2.310146
```

```
Y <- data$Y %>% as.matrix
X <- data %>% select(X1) %>% as.matrix %>% cbind(1, .)
solve(t(X) %*% X) %*% t(X) %*% Y
```

```
##          [,1]
##    133.743257
## X1   7.359522
```

A linear model

```
data %$% lm(Y ~ X1 + X2) %>% summary
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.968  -8.412  -1.734   9.369  31.769
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   79.342     51.944   1.527    0.130
## X1             3.828      3.095   1.237    0.219
## X2             2.310      2.004   1.153    0.252
##
## Residual standard error: 14.12 on 97 degrees of freedom
## Multiple R-squared:  0.7493, Adjusted R-squared:  0.7441
## F-statistic:   145 on 2 and 97 DF,  p-value: < 2.2e-16
```

As a generalized linear model

```
data %$% glm(Y ~ X1 + X2, family = gaussian(link = "identity")) %>% summary
```

```
##
## Call:
## glm(formula = Y ~ X1 + X2, family = gaussian(link = "identity"))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -28.968   -8.412   -1.734    9.369   31.769
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   79.342     51.944   1.527    0.130
## X1             3.828      3.095   1.237    0.219
## X2             2.310      2.004   1.153    0.252
##
## (Dispersion parameter for gaussian family taken to be 199.4787)
##
##     Null deviance: 77181  on 99  degrees of freedom
## Residual deviance: 19349  on 97  degrees of freedom
## AIC: 818.31
##
## Number of Fisher Scoring iterations: 2
```

Without multicollinearity

```
data %$% lm(Y ~ X1 + X2) %>% summary %>% list(coef = .$coefficients, R2 = .$r.squared) %>% .[-1]
```

```
## $coef
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 79.341826  51.944290 1.527441 0.1299058
## X1           3.827800   3.094799 1.236849 0.2191292
## X2           2.310146   2.004417 1.152528 0.2519368
##
## $R2
## [1] 0.7492974
```

```
data %$% lm(Y ~ X1) %>% summary %>% list(coef = .$coefficients, R2 = .$r.squared) %>% .[-1]
```

```
## $coef
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 133.743257 21.7202725  6.157531 1.628550e-08
## X1            7.359522  0.4339498 16.959383 6.541008e-31
##
## $R2
## [1] 0.7458642
```

Supercollinearity

```
mammalsleep %>% select(sws, ps, ts) %>%
  tail()
```

```
##     sws  ps   ts
## 57 11.0 2.3 13.3
## 58  4.9 0.5  5.4
## 59 13.2 2.6 15.8
## 60  9.7 0.6 10.3
## 61 12.8 6.6 19.4
## 62   NA  NA   NA
```

For predictors `sws`, `ps` and `ts`, the crossproduct $\mathbf{X}'\mathbf{X}$ is singular

```
X <- mammalsleep %>% select(sws, ps, ts) %>% na.omit() %>% as.matrix()
solve(t(X) %*% X)
```

```
## Error in solve.default(t(X) %*% X): system is computationally singular: reciprocal condition number = 5.4813
```

Singularity

If we remove the third column, the linear dependency is gone and all is well.

```
solve(t(X[, -3]) %*% X[, -3])
```

```
##                sws           ps
## sws  0.0009579194 -0.003152412
## ps  -0.0031524119  0.013760299
```

- A square matrix with no inverse we call singular
- A matrix $\mathbf{M}$ is said to be singular if its determinant $\det(\mathbf{M}) = 0$
- The determinant is the product of the eigenvalues

```
matrix(c(2, 3, 5, 7.5), 2, 2) %>% solve()
```

```
## Error in solve.default(.): Lapack routine dgesv: system is exactly singular: U[2,2] = 0
```

```
matrix(c(2, 3, 5, 7.5), 2, 2) %>% eigen() %>% .$values
```

```
## [1] 9.5 0.0
```

So?

Multicollinearity and supercollinearity may impact any data set.
Supercollinearity will occur in high-dimensional data, especially when $n \ll p$.

In general; the more columns you are introducing to the estimation procedure, the higher the likelihood that something may impact the estimation.

Why would we avoid this?

- If paramater estimates cannot be trusted, then model selection becomes challenging
- If the assumptions of the (linear) model are not met, then estimation becomes unreliable
- If $(\mathbf{X}'\mathbf{X})^{-1}$ does not exist, then it all falls apart.

Solution
Mess up $(\mathbf{X}'\mathbf{X})^{-1}$ such that it exists.

Ridge regression

How it works

Because the least squares estimator

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y},$$

is not defined, we add a small *ridge parameter* to avoid the problem:

$$\hat{\beta} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I_{pp}})^{-1}\mathbf{X}'\mathbf{Y}$$

Remember the simulated data:

```
Y <- data$Y %>% as.matrix
X <- data %>% select(X1, X2) %>% as.matrix %>% cbind(1, .)
```

Estimating $\hat{\beta}$

$$\hat{\beta} = (\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}_{\mathbf{pp}})^{-1}\mathbf{X}'\mathbf{Y}$$

When $\lambda = 0$, the result is simple and all three methods yield the same result

```
lm(Y ~ X1 + X2, data = data) %>% coef
```

```
## (Intercept)          X1          X2
##   79.341826    3.827800    2.310146
```

```
glmnet(x=X[,-1], y=Y, family = gaussian, alpha = 0, lambda = 0, thresh=1e-17) %>% coef %>% t()
```

```
## 1 x 3 sparse Matrix of class "dgCMatrix"
##    (Intercept)     X1       X2
## s0    79.34183 3.8278 2.310146
```

```
MASS::lm.ridge(Y ~ X1 + X2, data = data, lambda = 0)
```

```
##                   X1        X2
## 79.341826  3.827800  2.310146
```

Estimating $\hat{\beta}$

$$\hat{\beta} = (\mathbf{X}'\mathbf{X} + \lambda \mathbf{I_{pp}})^{-1}\mathbf{X}'\mathbf{Y}$$

When $\lambda = 1$, the results are different

```
lm(Y ~ X1 + X2, data = data) %>% coef
```

```
## (Intercept)          X1          X2
##   79.341826    3.827800    2.310146
```

```
glmnet(x=X[,-1], y=Y, family = gaussian, alpha = 0, lambda = 1) %>% coef %>% t()
```

```
## 1 x 3 sparse Matrix of class "dgCMatrix"
##    (Intercept)       X1       X2
## s0    219.2025 2.462095 1.592984
```

Why are results different?

The results differ because of the regularization.

- The total residual sum of squares is still minnimized in ridge regression
- However, with λ we aim to shrink the estimates towards zero
- You can see it as keeping the sum of squares of the coefficient below a certain value. λ governs which value.

With $\lambda = 0$, shrinkage is absent and ridge regression yields the same solution as least-squares estimation. But if $\lambda \to \infty$, the effect of shrinkage becomes much more prominent and the ridge parameters will move to zero.

Different λ will thus produce different estimates.

Penalization visualized

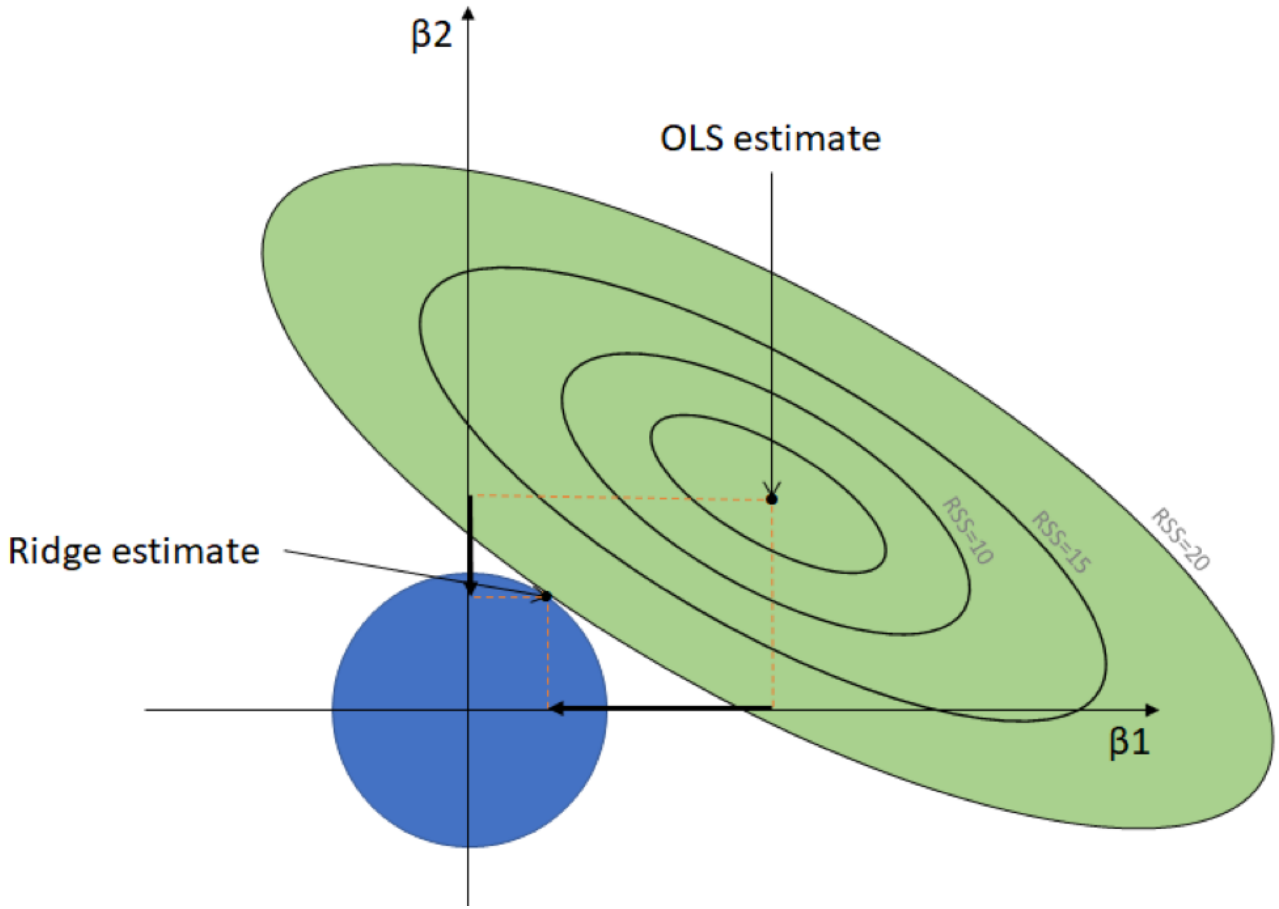


image from Bruce Hansen

How to choose λ

```
fit <- cv.glmnet(X, Y, alpha = 0, nfolds = 5)
fit
```

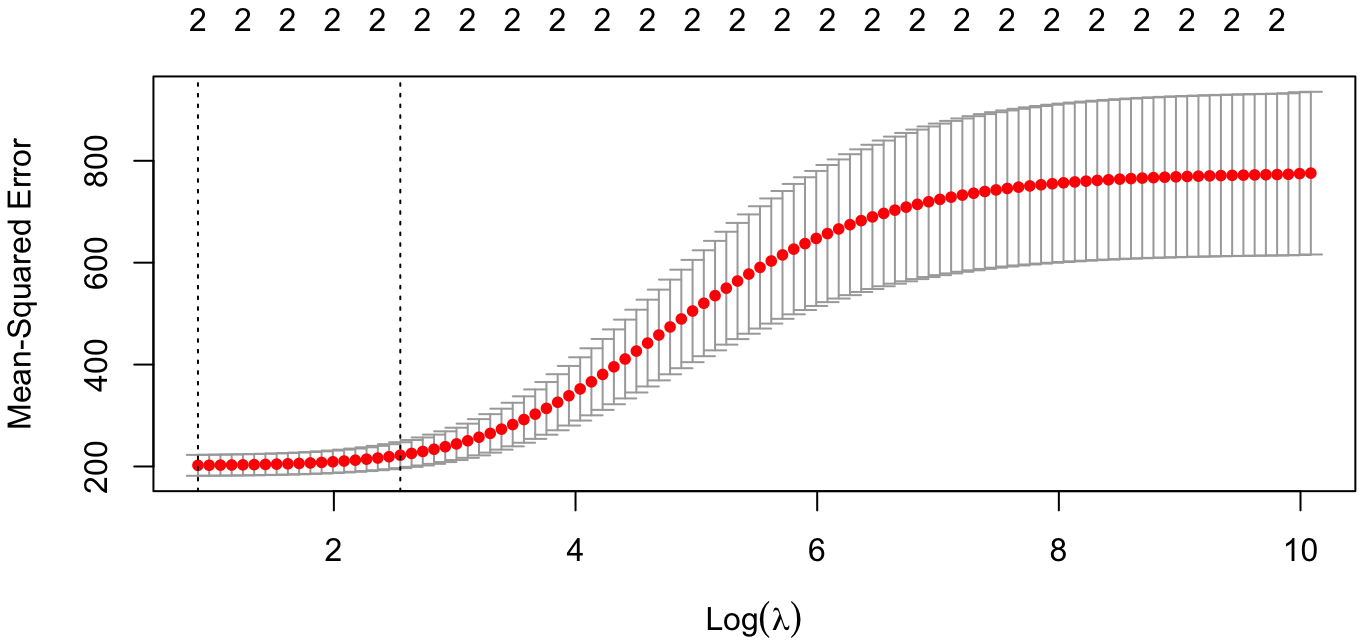
```
##
## Call:  cv.glmnet(x = X, y = Y, nfolds = 5, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min  2.399   100   202.2 20.54       2
## 1se 12.804    82   222.2 25.41       2
```

```
fit %>% coef(s = c(2.399, 12.804))
```

```
## 4 x 2 sparse Matrix of class "dgCMatrix"
##                    s1         s2
## (Intercept) 95.327477 157.161508
##               .          .
## X1           3.566194   3.005572
## X2           2.280925   1.942276
```

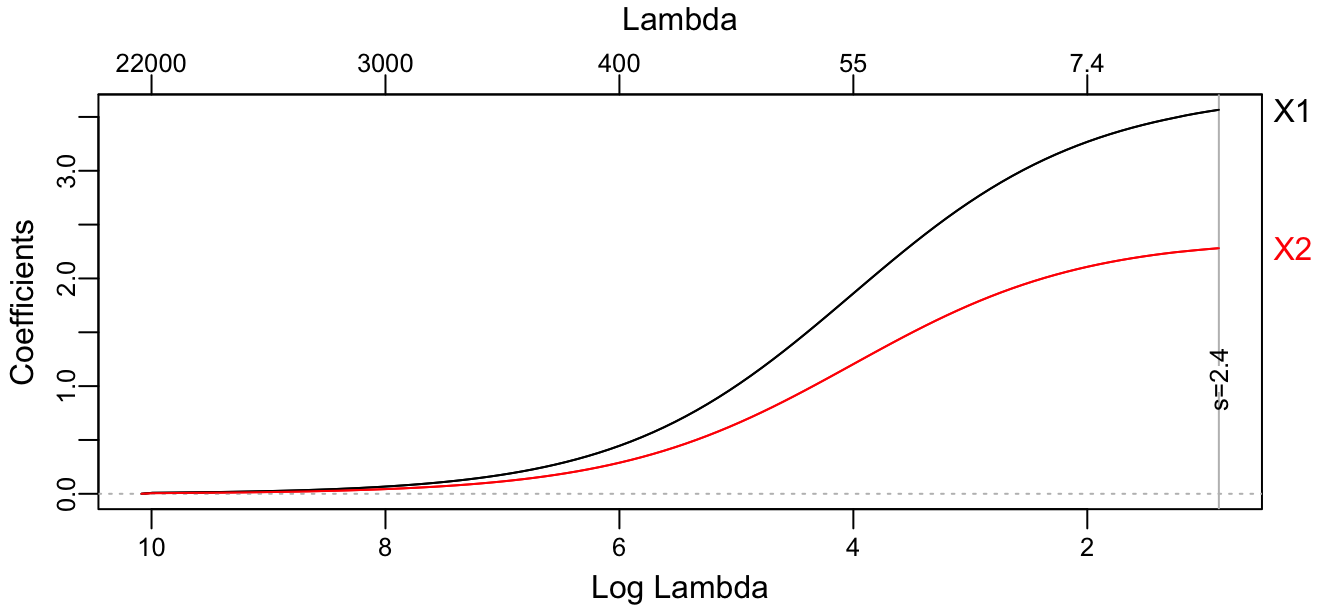
Visual inspection

```
plot(fit)
```



Visual inspection

```
glmnet(X, Y, alpha = 0) %>% plot_glmnet(s = fit$lambda.min) # from plotmo
```



Last comments

Historically, ridge regression has been developed to

1. solve multicollinearity
2. solve singularity
3. stabilize the estimator

More recently, in contemporary data science it is mainly aimed at

a penalizing least squares **b** regularizing least squares