

Class 13: Ridge

MGSC 310

Prof. Jonathan Hersh

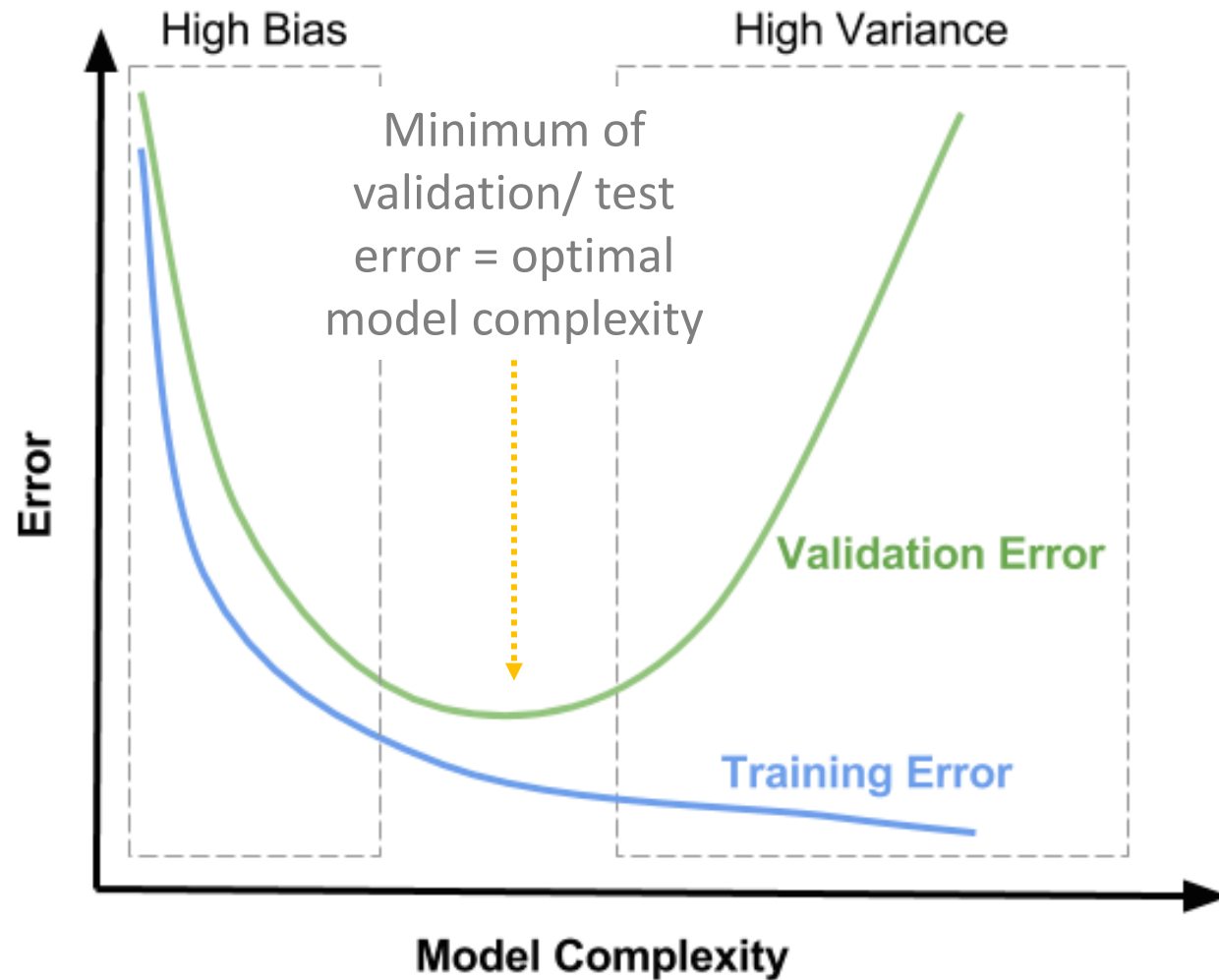
Class 13: Announcements

1. Quiz 5 posted, due Thursday @ midnight
2. Problem Set 4 will post later this week

Class 13: Outline

1. Review: K-Fold Cross-Validation
2. What is Regularization?
 - Choosing to Reduce Model Complexity To Increase Out of Sample Performance
3. Why Regularize?
4. Ridge Regression
5. Ridge Regression in R
6. Ridge Regression Lab

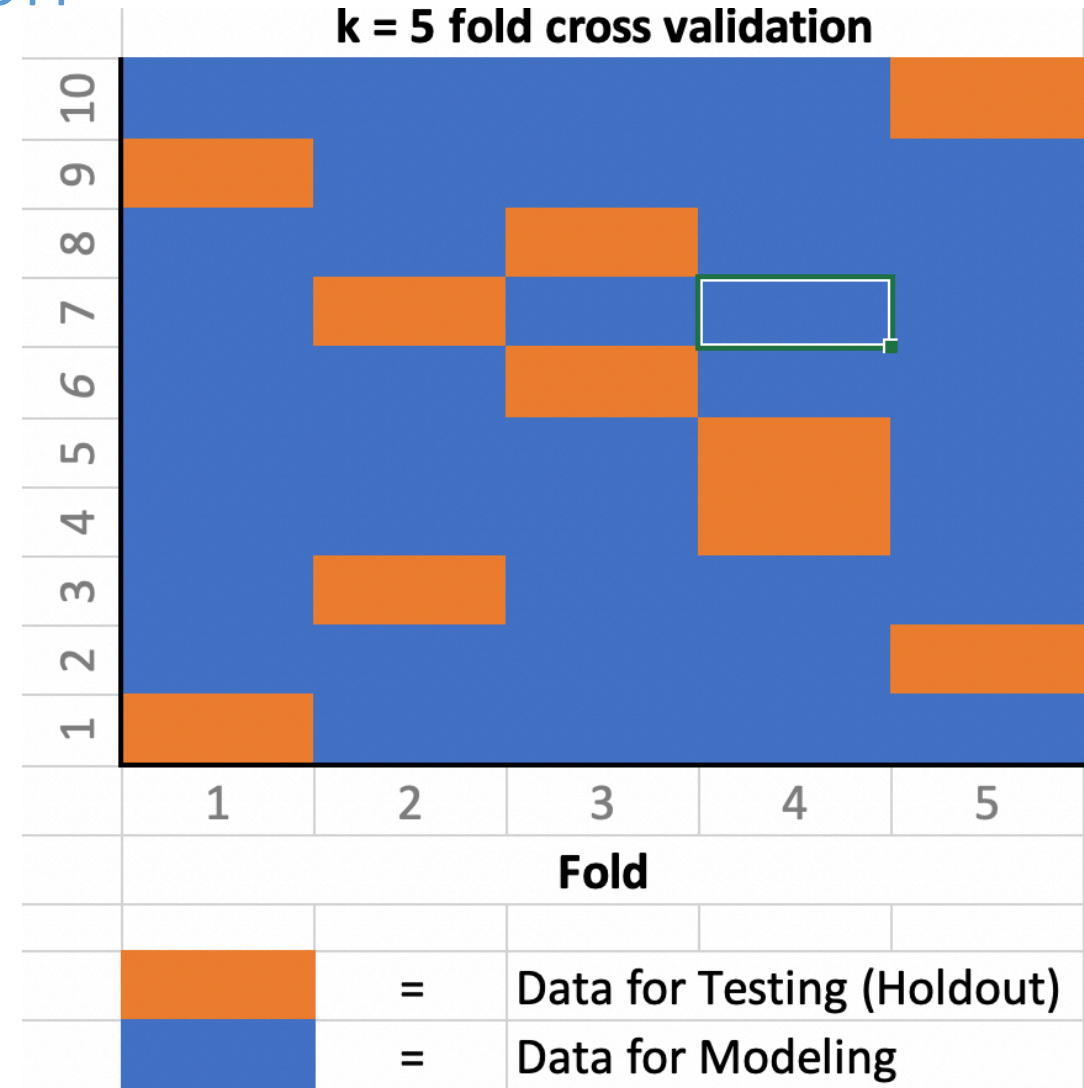
Recall Bias-Variance Tradeoff



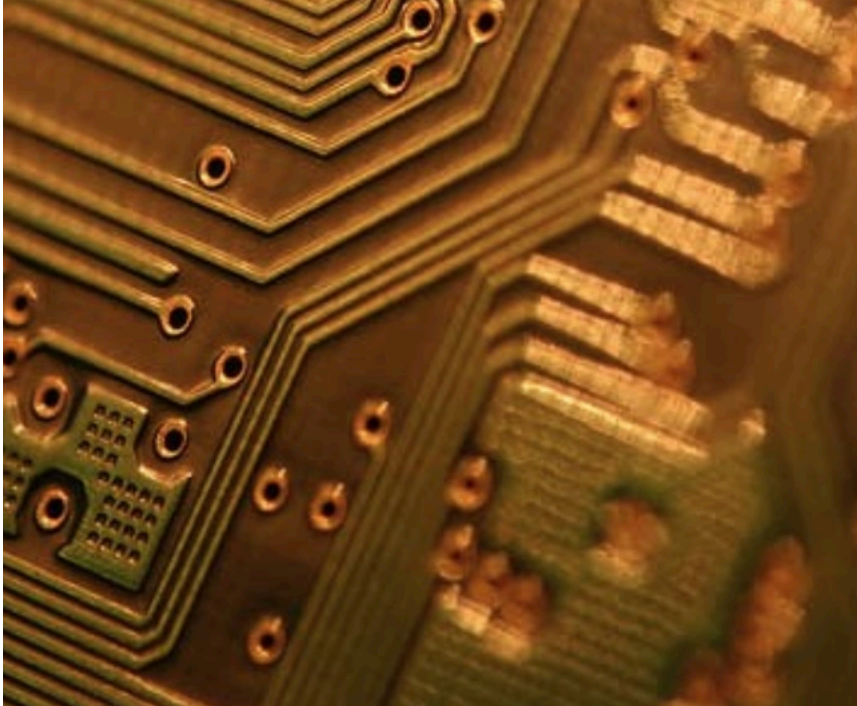
- More model complexity test performance, but beyond a certain point it can increase test/validation error
- Note that training error always increases with model complexity!
- Key is determining optimal model complexity (in linear models, more complexity = more variables)

Recall: K-Fold Cross-Validation

- In K-Fold Cross Validation we partition (divide) data into K distinct groups
- Fit a model using data excluding group 1, use that model to predict into group 1.
- Fit a model using data excluding group 2, etc
- **Cross-validation is used to approximate out of sample fit as we tune/optimize model parameters**



Example: Semiconductor Manufacturing Data



- As an example, let's use the semiconductor.csv data.
- This data has **X data on 200** sensors during a semiconductor manufacturing process
- Y (outcome) data is 0 = if chip has a failure, 1 = if chip has a failure
- We can model probability of failure (Y) based on sensor readings as:

$$pr(Y = failure|X) = \frac{\exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_{200} x_{200})}{1 + \exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_{200} x_{200})}$$

Semiconductor Data

semi		regularization.R				
		Filter		Cols: « < 1 - 50 > »		
	FAIL	SIG1	SIG2	SIG3	SIG4	SIG5
1	0	2.22377902	-3.490181e-01	-1.985797334	-2.60099344	0.45109185
2	1	-0.13037789	-3.813179e-01	-1.285171583	-0.76630168	-3.09830188
3	0	-0.87113777	-8.253003e-01	-1.615008020	-0.44880323	-2.57078190
4	0	-0.87734089	-1.905598e+00	-2.306113839	-1.21076117	-4.03747080
5	0	-2.11345560	-2.313845e+00	-2.917152994	-3.33346700	-4.43934574
6	0	1.80488091	-4.560380e+00	-2.506962628	-2.61460475	-1.01839946
7	0	-1.30743168	-1.161162e+00	-4.139834032	-4.43371350	-2.23762303
8	0	0.40400604	-2.036166e+01	0.459195317	0.84722747	-1.90408832
9	0	-1.30589584	-2.924771e+00	-5.101385030	-5.62690442	-2.93436240
10	1	2.05038660	-2.426871e+01	2.978795014	4.22481789	3.96080164
11	1	-0.77749202	-2.883279e+00	-4.253569166	-4.65590226	-2.61283119
12	0	-0.42034703	-6.701494e-01	-2.276992521	-0.33334995	-3.45007905

- This is big data!
 - $N = 1477$
 - $K = 200$
- Specifically **wide** given that we have many predictors to possibly model chip failure

Fit a Logistic Model Using All Sensors

```
semi <- read_csv('https://raw.githubusercontent.com/
write_csv(semi, here::here("datasets", "semicond

head(semi)

semi_split <- initial_split(semi, 0.9)
semi_train <- training(semi_split)
semi_test <- testing(semi_split)
```

```
# -----
# Estimate Wildly Overfit Model
# -----
full_mod <- glm(FAIL ~ .,
               data = semi_train,
               family = binomial)

summary(full_mod)
|
```

- Split data into testing and training sets
- Estimate a logistic model predicting failure as a function of all 200 sensor readings
- Note “~ .” means use every other variable in the matrix to predict the outcome

Model Output

```
Call:
glm(formula = FAIL ~ ., family = binomial, data = semi_train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7348  -0.0553  -0.0039  -0.0001   3.7994

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -12.202987   1.957725  -6.233 4.57e-10 ***
SIG1         -1.425270   2.922034  -0.488 0.625715
SIG2         -0.490279   0.159093  -3.082 0.002058 **
SIG3          0.084154   0.719326   0.117 0.906868
SIG4          0.170293   1.493732   0.114 0.909234
SIG5          1.991300   4.062134   0.490 0.623985
SIG6          2.751601   4.293379   0.641 0.521591
SIG7          2.470717   3.527188   0.700 0.483629
SIG8          0.065707   0.607741   0.108 0.913903
SIG9          1.527822   1.298849   1.176 0.239479
SIG10         -1.599538   1.887191  -0.848 0.396674
SIG11          0.693596   1.028660   0.674 0.500139
SIG12         -0.716754   1.197967  -0.598 0.549634
SIG13          0.531440   0.882866   0.602 0.547209
SIG14          0.594248   0.507533   1.171 0.241656
SIG15         -0.630230   0.661081  -0.953 0.340421
SIG16          1.862401   0.851868   2.186 0.028797 *
SIG17         -1.244271   1.220076  -1.020 0.307809
```

- Model seems “okay”
- But two problems
 1. Do we have too many variables?
 2. If so, which are the right variables?
- For #1) in-sample error measures will always tell us more variables improve model fit
- For #2) With a p-value heuristic like 0.05, we expect 10 variables to be canonically significant even if they are truly equal to zero!

However, We Are Really Overfit

```
> semi_test_preds <- predict(full_mod,
+                             newdata = semi_test,
+                             type = "response")
> semi_train_preds <- predict(full_mod,
+                              newdata = semi_train,
+                              type = "response")
>
> dev_test <- deviance(semi_test$FAIL,
+                       semi_test_preds)
> dev0_test <- deviance(semi_test$FAIL,
+                        mean(semi_test$FAIL))
>
> dev_train <- deviance(semi_train$FAIL,
+                       semi_train_preds)
> dev0_train <- deviance(semi_train$FAIL,
+                        mean(semi_train$FAIL))
>
> print(paste0("Pseudo In-Sample R2: ",
+              round(1 - dev_train/dev0_train,4)))
[1] "Pseudo In-Sample R2: 0.6"
>
> print(paste0("Pseudo Out-Sample R2: ",
+              round(1 - dev_test/dev0_test,3)))
[1] "Pseudo Out-Sample R2: -0.922"
```

- In-sample R2 is 0.6, Out-of-sample R2 is negative! - 0.922
- Why is it negative? Because a model with just an intercept does better than our really overfit model!
- We are fitting sensor reading noise and not the true sensor reading indicating possible failure!
- Why? N is too small relative to K

How do we make it "costly" to increase K? We add a penalty to increasing model complexity. That's the idea of **regularization**

- **Lasso, Ridge, and ElasticNet** are all examples of regularized regression models, that penalize magnitude of coefficients to balance the overfitting cost

Even IMF Overfits Forecasting Models!

Overfitting in Judgment-based Economic Forecasts: The Case of IMF Growth Projections

Author/Editor: [Klaus-Peter Hellwig](#)

Publication Date: December 7, 2018

Electronic Access: [Free Download](#) Use the free [Adobe Acrobat Reader](#) to view this PDF file

Disclaimer: IMF Working Papers describe research in progress by the author(s) and are published to elicit comments and to encourage debate. The views expressed in IMF Working Papers are those of the author(s) and do not necessarily represent the views of the IMF, its Executive Board, or IMF management.

- TIL IMF forecasts are no better than a model with just an intercept

Series:

Working Paper No. 18/260

Subject:

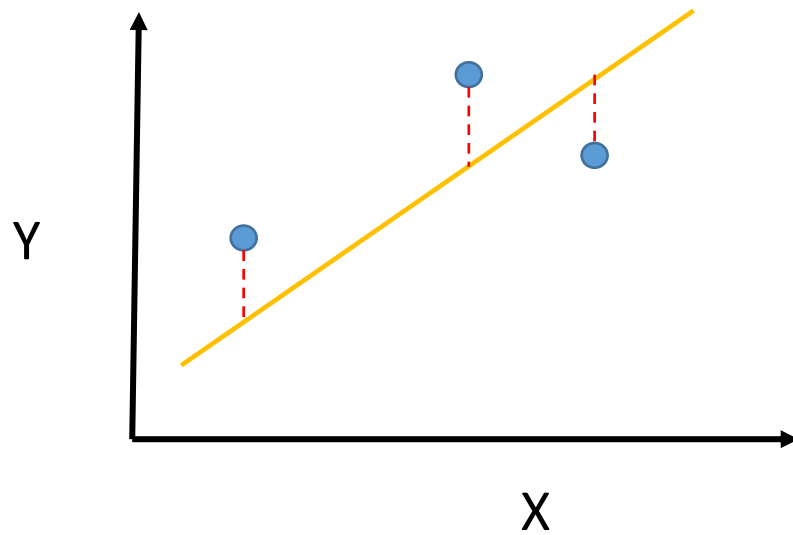
[Econometric models](#) | [Economic forecasting](#) | [Economic growth](#) | [Gross domestic product](#)

I regress real GDP growth rates on the IMF's growth forecasts and find that IMF forecasts behave similarly to those generated by overfitted models, placing too much weight on observable predictors and underestimating the forces of mean reversion. I identify several such variables that explain forecasts well but are not predictors of actual growth. I show that, at long horizons, IMF forecasts are little better than a forecasting rule that uses no information other than the historical global sample average growth rate (i.e., a constant). Given the large noise component in forecasts, particularly at longer horizons, the paper calls into question the usefulness of judgment-based medium and long-run forecasts for policy analysis, including for debt sustainability assessments, and points to statistical methods to improve forecast accuracy by taking into account the risk of overfitting.

Least Squares (OLS Estimator)

$$\hat{\beta} \text{ minimizes: } \sum_{i=1}^N (y_i - \beta_0 - x_{i1}\beta_1)^2$$

Least squares minimizes the sum of squared residuals (e.g. $y_i - \hat{y}$)

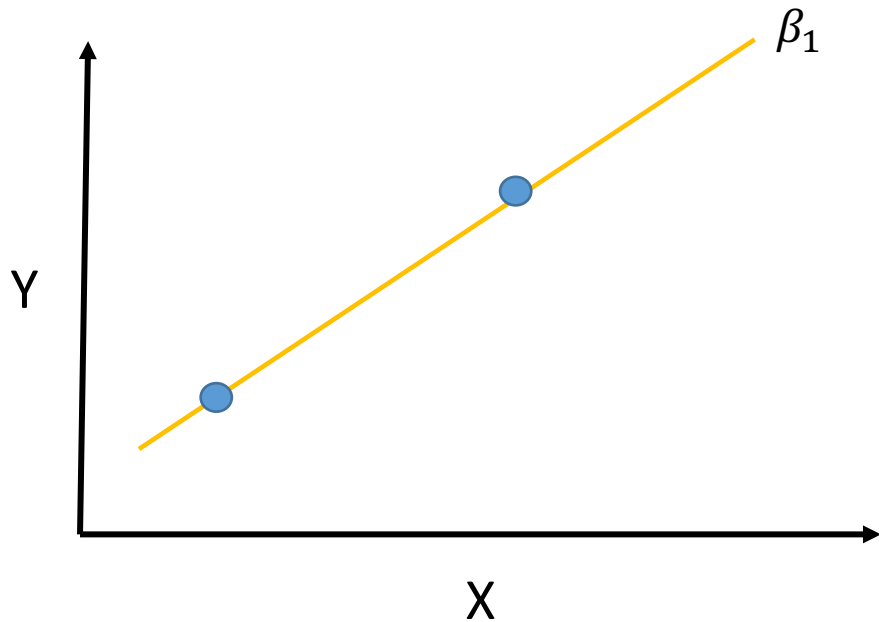


Let's set $K = 1$ (i.e. one explanatory variable to make this easier)

Visually, the slope (β_1) minimizes the difference between the points and the yellow line (red lines)

Ridge Regression Idea

$\hat{\beta}_{ridge}$ minimizes: *residuals* + $\lambda \cdot (\beta_1)^2$

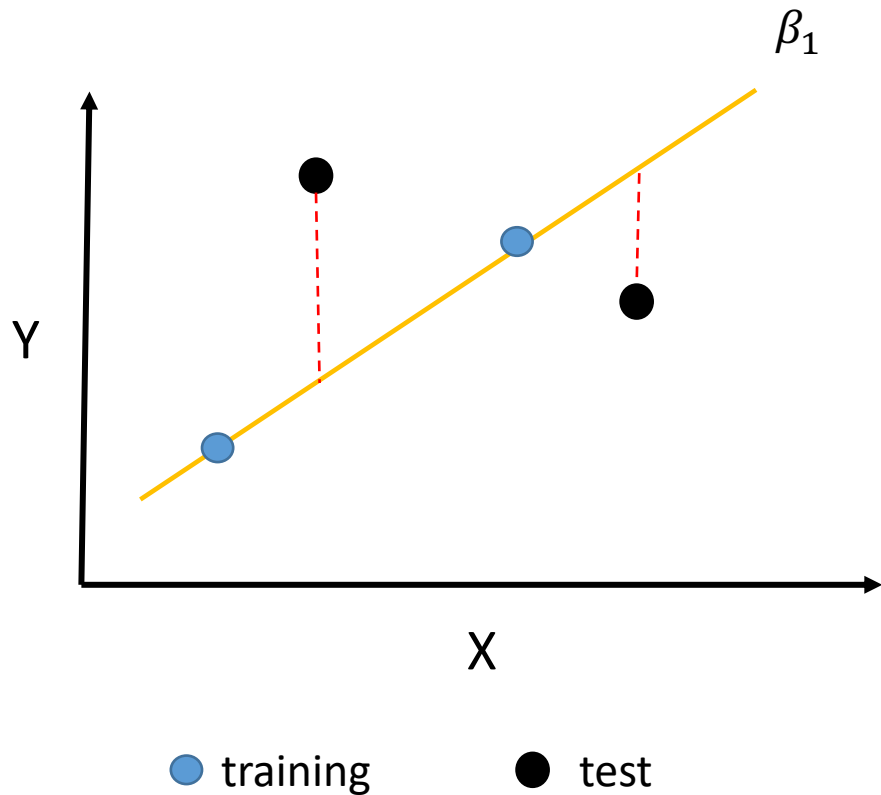


Ridge regression in contrast minimizes the OLS residuals plus the squared slope of beta times λ

Why is this smart? Let's take the example where we only have two data points.

Our best fit line describes the points perfectly and our residuals = 0. Our bias is zero!

OLS Bias and Variance



Bias = 0, which is good

What about our variance?

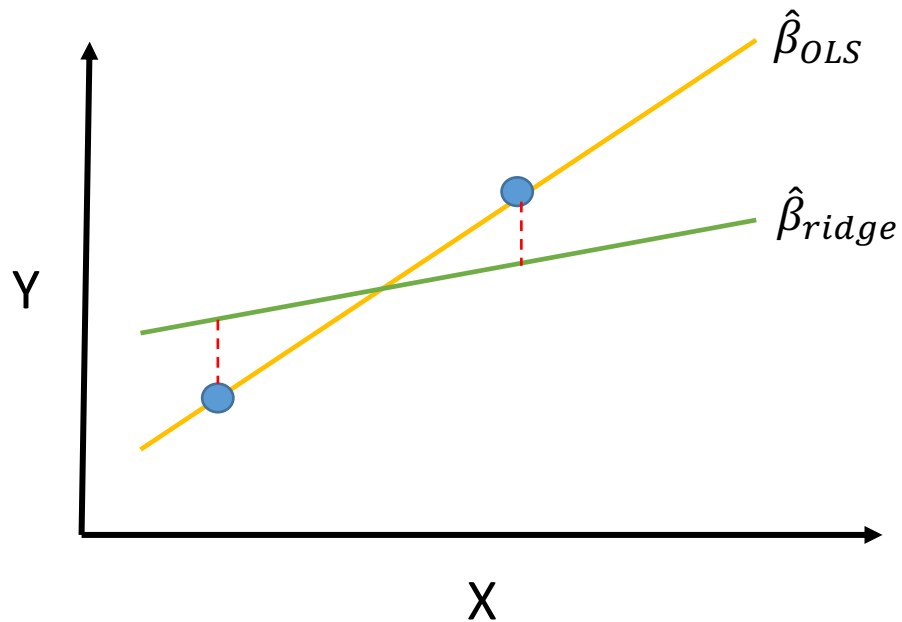
Imaging we had the following data in our test set

Then our test error would be the lines in red, and our variance would be high

Intuition: increasing bias can often reduce variance

Ridge Regression Idea

$\hat{\beta}_{ridge}$ minimizes: $residuals + \lambda \cdot (slope)^2$



Now let $\lambda = 1$. $\hat{\beta}_{ridge}$ minimizes:

$$\begin{aligned} & residuals + 1 \cdot (\beta_1)^2 \\ &= residuals + 1 \cdot (\beta_1)^2 \end{aligned}$$

Suppose the OLS slope = 2

then $\hat{\beta}_{ridge}$ would choose to have some positive residuals to because

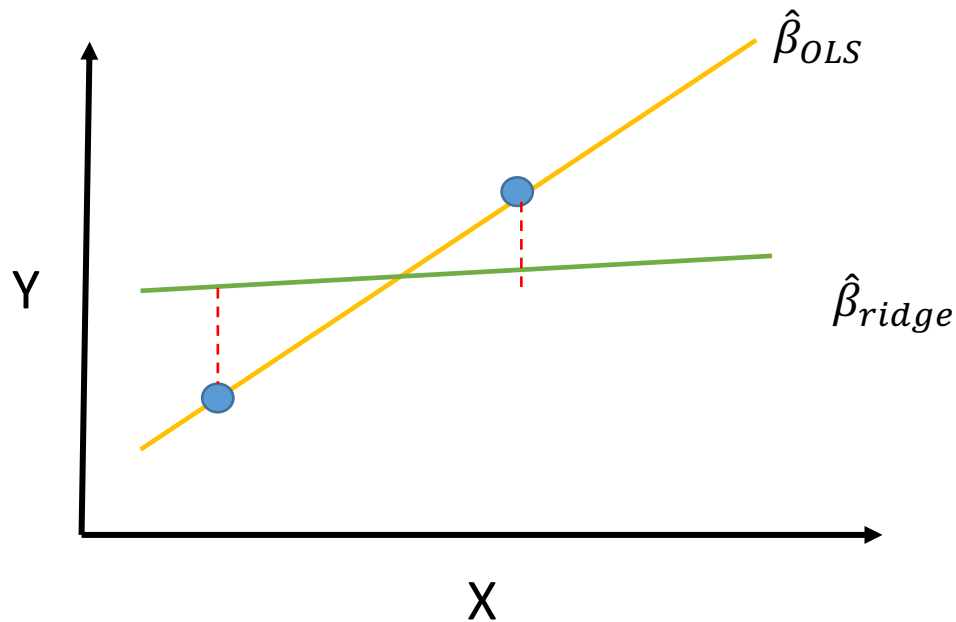
$$\text{residuals_ridge} + 1 \cdot (1)^2$$

is likely less than $0 + (2)^2 = 4$ (residuals with OLS plus lambda penalty)

Aka: we accept a little bias (higher residuals) for less variance (better test performance)

Ridge Regression and Lambda

$\hat{\beta}_{ridge}$ minimizes: $residuals + \lambda \cdot (slope)^2$



What if we set $\lambda = 100$?

$\hat{\beta}_{ridge}$ minimizes:

$$residuals + 100 \cdot (\beta_1)^2$$

Here ridge will have to accept very high residuals in order to avoid high slope penalty

$\hat{\beta}_{ridge}$ will set slope to a very small amount (ex = 0.001) and we get:

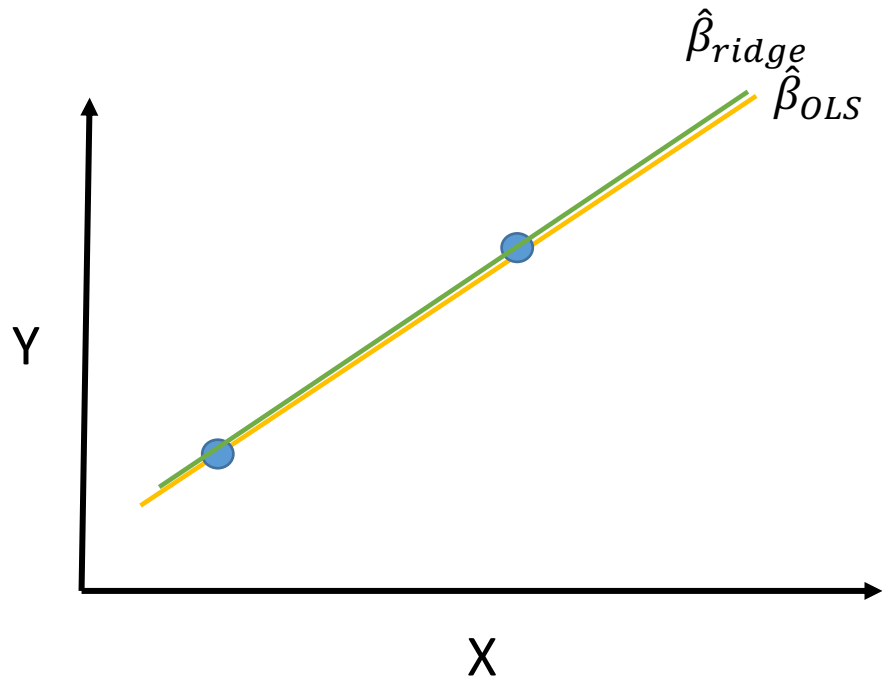
$$residuals + 100 \cdot (0.001)^2$$

$$= residuals + 0.0001$$

E.g. we accept a lot of bias but low variance

Ridge Regression and Lambda

$\hat{\beta}_{ridge}$ minimizes: $residuals + \lambda \cdot (slope)^2$



What if we set $\lambda = 0$?

$\hat{\beta}_{ridge}$ minimizes:

$$residuals + 0 \cdot (\beta_1)^2$$

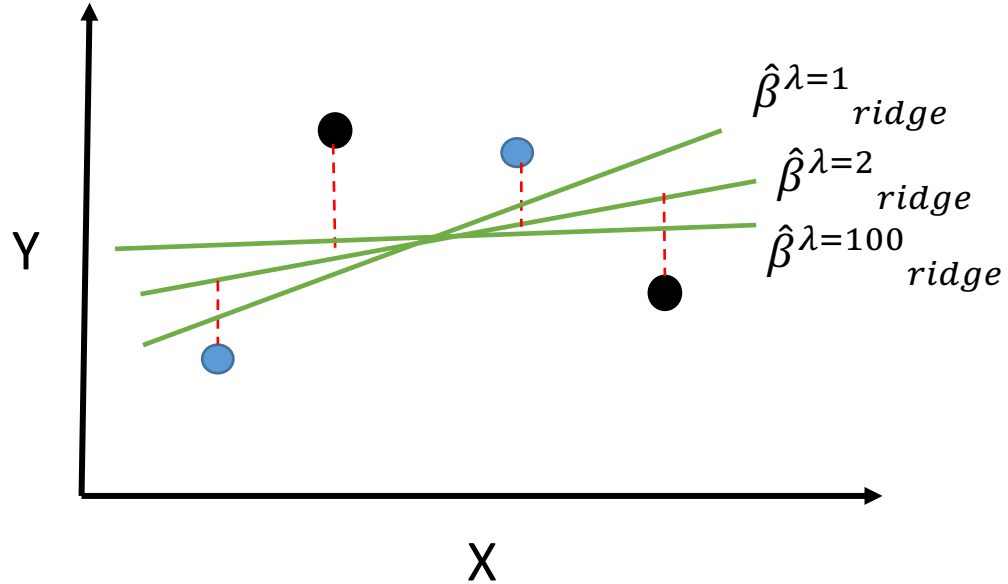
$$= residuals + 0$$

That's just the OLS estimator and $\hat{\beta}_{ridge} = \hat{\beta}_{OLS}$

E.g. lower the penalty on lambda the closer ridge is to OLS.

Larger $\lambda \Rightarrow$ More Penalization, Smaller Coefficients

$\hat{\beta}_{ridge}$ minimizes: *residuals* + $\lambda \cdot (\text{slope})^2$



So how do we choose λ ?


In practice we estimate a many models with many different values of λ

We pick a min and max lambda (say 0 and 100), then choose some points in-between

Optimal λ^* minimizes cross-validated error

Ridge Regression in R: glmnet and glmnetUtils


glmnet 4.0.2

 Get started Reference Articles ▾ Changelog

Lasso and Elastic-Net Regularized Generalized Linear Models

We provide extremely efficient procedures for fitting the entire lasso or elastic-net regularization path for linear regression (gaussian), multi-task gaussian, logistic and multinomial regression models (grouped or not), Poisson regression and the Cox model. The algorithm uses cyclical coordinate descent in a path-wise fashion. Details may be found in Friedman, Hastie, and Tibshirani (2010), Simon et al. (2011), Tibshirani et al. (2012), Simon, Friedman, and Hastie (2013).

Version 3.0 is a major release with several new features, including:



Introduction to glmnetUtils

The [glmnetUtils package](#) provides a collection of tools to streamline the process of fitting elastic net models with [glmnet](#). I wrote the package after a couple of projects where I found myself writing the same boilerplate code to convert a data frame into a predictor matrix and a response vector. In addition to providing a formula interface, it also features a function `cva.glmnet` to do crossvalidation for both α and λ , as well as some utility functions.

The formula interface

The interface that glmnetUtils provides is very much the same as for most modelling functions in R. To fit a model, you provide a formula and data frame. You can also provide any arguments that glmnet will accept. Here are some simple examples for different types of data:

```
# least squares regression
(mtcarsMod <- glmnet(mpg ~ cyl + disp + hp, data=mtcars))
```

- glmnet quickly estimates Ridge and Lasso models
- It's one of the best package in R or any language (ported to python in 2015) but can be difficult to work with
- glmnetUtils is a helper package that makes our lives much easier
- Make sure to install both glmnet and glmnetUtils!

Ridge Model with glmnetUtils

```
# estimate a Ridge model using glmnet
# note if you get an error make sure you
# have loaded glmnetUtils
ridge_mod <- cv.glmnet(hwy ~ .,
                      data = mpg_clean,
                      # note alpha = 0 sets ridge!
                      alpha = 0)
```

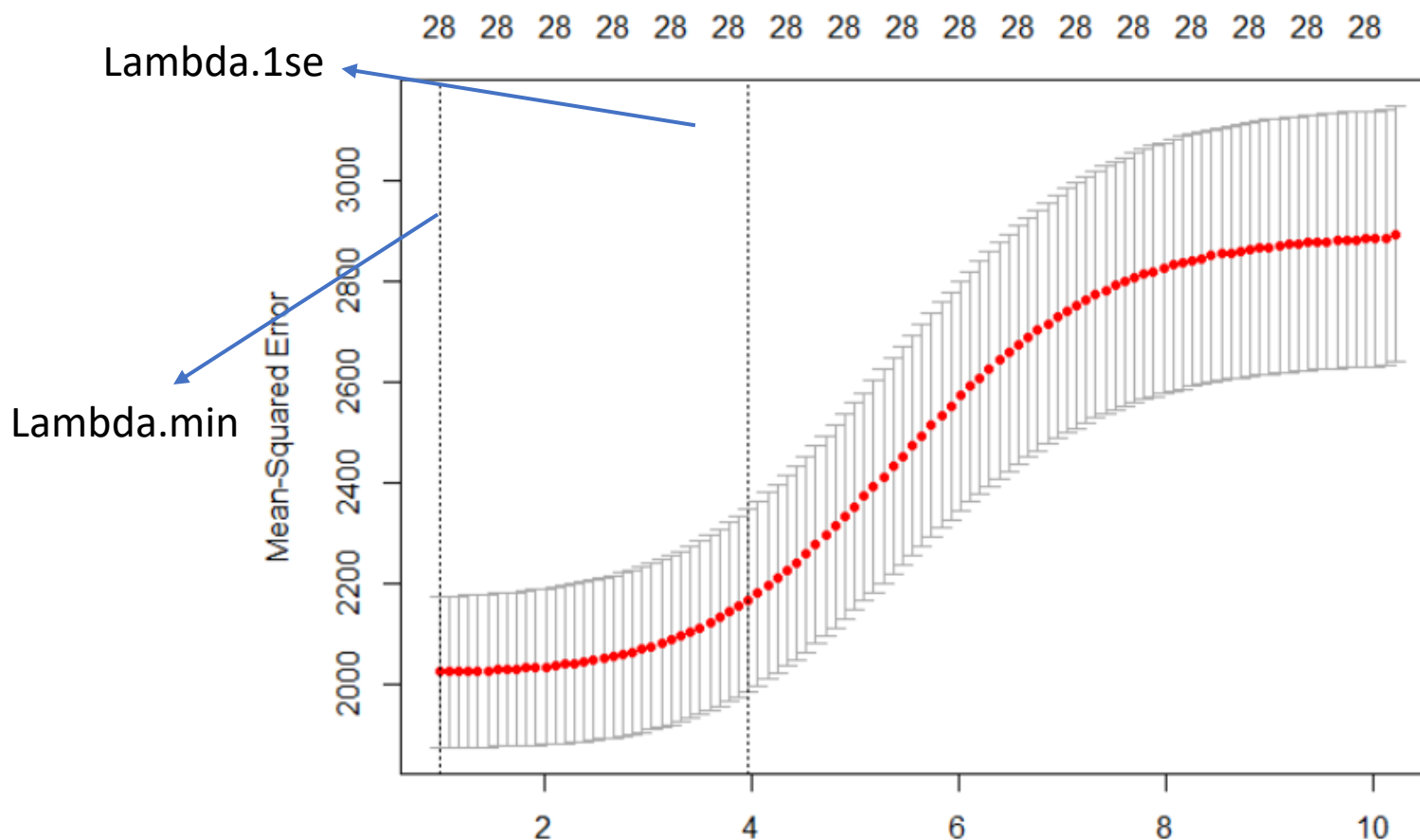
- cv.glmnet estimates a lasso or ridge model. Automatically performs cross-validation to select optimal lambda!
- We must set $\alpha = 0$ to signify ridge model

```
> print(ridge_mod$lambda.min)
[1] 0.7247465
```

```
> #
> print(ridge_mod$lambda.1se)
[1] 2.665893
```

- lambda.min stores the value of lambda that minimizes cross-validated error
- lambda.1se stores the value of lambda that minimizes cross-validated error plus one estimated standard error
- Why the difference? Lambda.min gives the best performing value, lambda.1se add extra penalization for more parsimony

Cross-Validated MSE Plot As A Function of Lambda



- `plot(model_object)` calls the MSE plot
- This shows how the cross-validated MSE (y-axis) varies as we increase lambda (penalization)
- Model defaults to lambda.1se but either can be appropriate

Smaller λ
Less penalization
Coefficients more
Similar to OLS

Larger λ
More penalization
Smaller coefficients

Printing Ridge Coefficient Vector

```
# print coefficient using lambda.min
coef(ridge_mod, s = ridge_mod$lambda.min) %>%
  round(3)

# print coefficient using lambda.1se
coef(ridge_mod, s = ridge_mod$lambda.1se) %>%
  round(3)
```

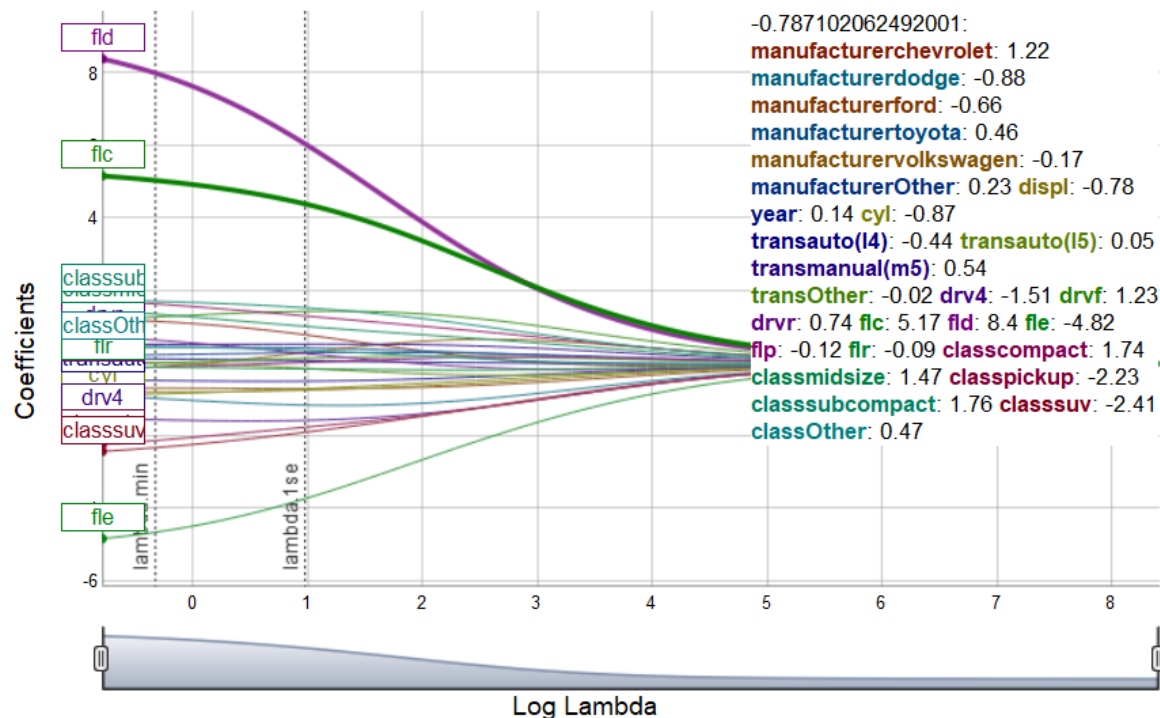
```
> ridge_coefs <- data.frame(
+   `ridge_min` = coef(ridge_mod, s = ridge_mod$lambda.min) %>%
+     round(3) %>% as.matrix() %>% as.data.frame(),
+   `ridge_1se` = coef(ridge_mod, s = ridge_mod$lambda.1se) %>%
+     round(3) %>% as.matrix() %>% as.data.frame()
+ ) %>% rename(`ridge_min` = 1, `ridge_1se` = 2)
> ridge_coefs
```

	ridge_min	ridge_1se
(Intercept)	-236.585	-144.878
manufacturerchevrolet	1.167	0.798
manufacturerdodge	-0.957	-1.139
manufacturerford	-0.669	-0.688
manufacturertoyota	0.461	0.437
manufacturervolkswagen	-0.091	0.309
manufacturerOther	0.245	0.302
displ	-0.783	-0.727
year	0.134	0.087
cyl	-0.835	-0.689
transauto(14)	-0.461	-0.480
transauto(15)	0.023	-0.126
transmanual(m5)	0.535	0.541
transOther	0.020	0.150
drv4	-1.543	-1.561

- Remember ridge estimates a model for every value of lambda, so we actually have dozens of coefficient vectors
- We must specify the lambda to use that indexes the coefficients desired
- We can collect and print the coefficients for every lambda value desired

Coefpath() in coefplot to Examine Shrinkage Path of Coefficients

```
### examine coefficient shrinkage path
# note may need to install devtools first
# install.packages('devtools')
devtools::install_github("jaredlander/coefplot")
library('coefplot')
coefpath(ridge_mod)
```



- The function `coefpath` will allow us to interactively examine the shrinkage path for coefficients
- Coefficients are standardized (mean zero, std dev =1) and coefficient magnitudes are plotted as we vary λ

Ridge Lab

```
# -----  
# Lab Exercises  
# -----  
# 1. Load the semiconductor dataset and split into testing and  
# training sets  
semi <- read_csv('https://raw.githubusercontent.com/TaddyLab/MBACourse/master/semiconductor.csv')  
semi_split <- initial_split(semi, 0.9)  
semi_train <- training(semi_split)  
semi_test <- testing(semi_split)  
  
# 2. Using the function cv.glmnet estimate a ridge model using  
# the training data and call this ridge_mod2.  
# Note we must set family = "binomial" to specify  
# the binary y variable  
ridge_mod2 <- cv.glmnet(semi_train, semi_test[, "y"], family = "binomial")  
  
# 3. What does the "cv" in cv.glmnet stand for? Why is it used here?  
  
# 4. Call the plot function against the ridge_mod2 and describe the plot.  
  
# 5. Print the coefficient vector using lambda.1se. Store this vector as  
# ridge_coef_1se.  
ridge_coef_1se <- coef(ridge_mod2, lambda = 1se)
```


Ridge Lab Continued

```
# 5. Print the coefficient vector using lambda.1se. Store this vector as
#   ridge_coef_1se.

# 6. Print the coefficient vector using lambda.min. Store this vector as
#   ridge_coef_min

# 7. Use the code below to change the ridge matrices from sparse matrices (efficient for
#   large number of rows) to regular matrices. And then combine the coefficient vectors
#   into one coefficient vector for both ridge 1se and ridge min.

ridge_coef_min <- ridge_coef_min %>% as.matrix() %>%
  as.data.frame() %>%
  round(3)

ridge_coef_1se <- ridge_coef_1se %>% as.matrix() %>%
  as.data.frame() %>%
  round(3)

ridge_coefs <- bind_cols(ridge_coef_min, ridge_coef_1se) %>%
  rename(`Ridge_min` = 1, `Ridge_1se` = 2)

# 8. Print ridge_coefs and describe the difference between
#   the coefficients for ridge_mod2 with lambda.min versus lambda.1se?
#   How are they different and why?

# 9. If you have time, load the library 'coefplot' and run the function
#   'coefpath' against the ridge model.
```