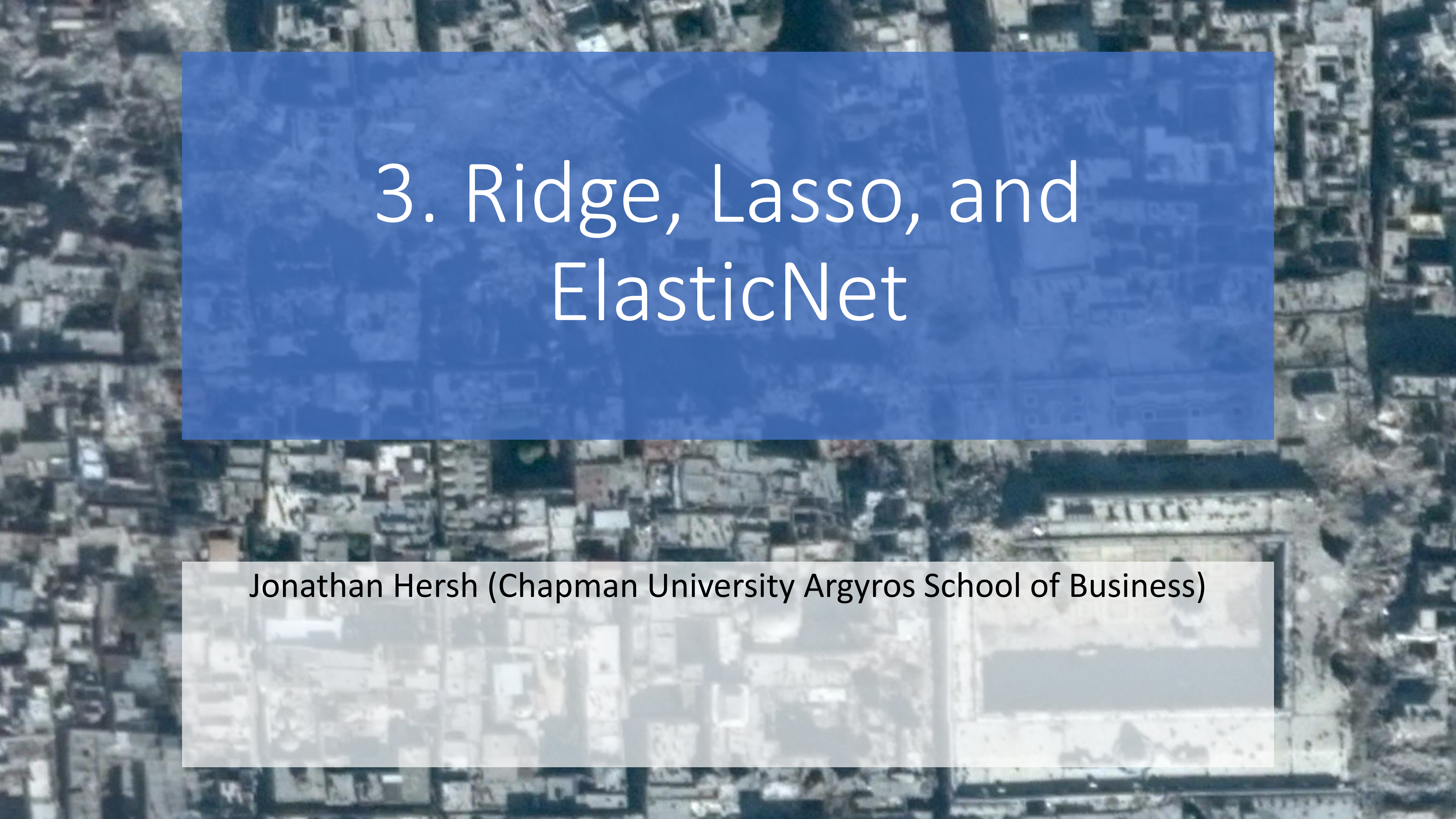


3. Ridge, Lasso, and ElasticNet

Jonathan Hersh (Chapman University Argyros School of Business)

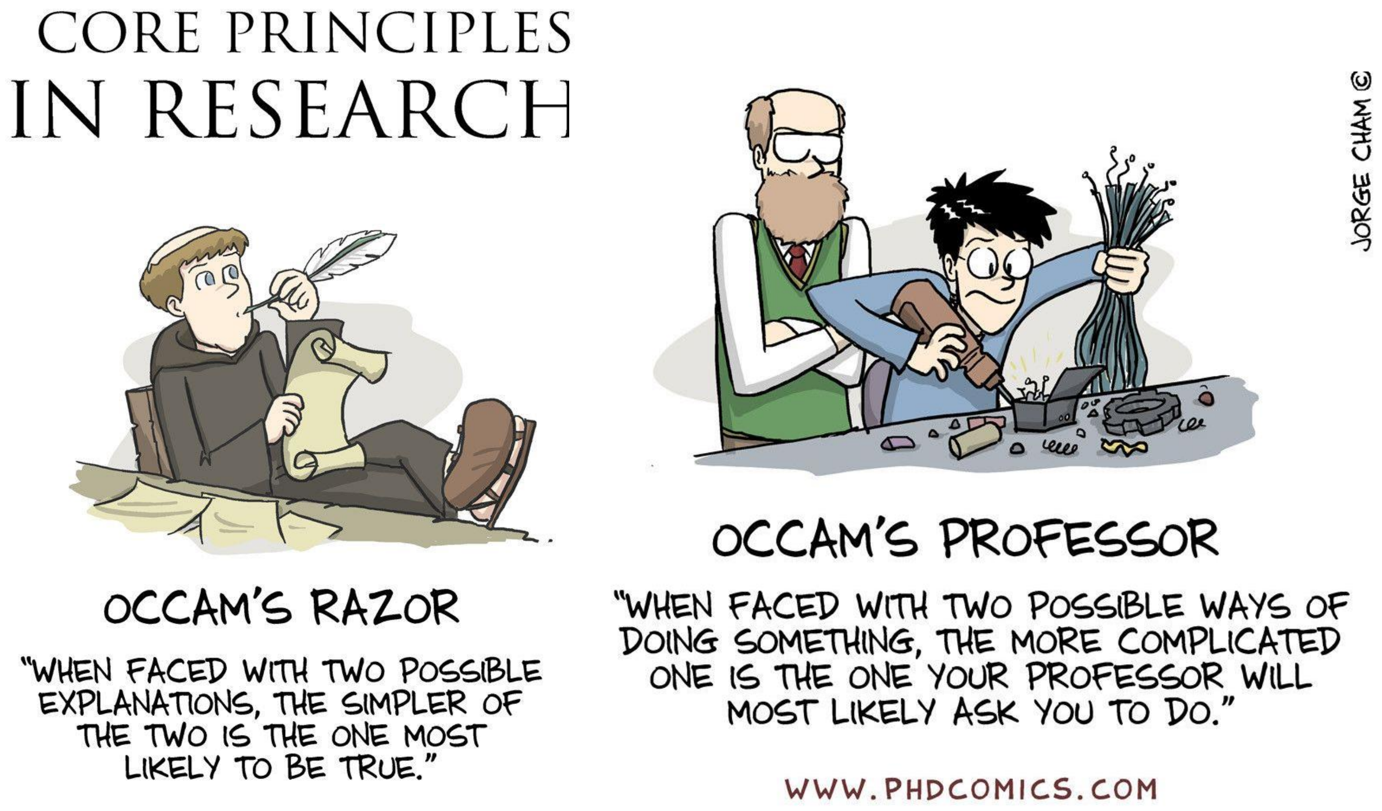
Outline

1. **Ridge Regression (Regularized Regression)**
2. **Cross-Validation**
3. **Lasso regression**
4. **ElasticNet!**

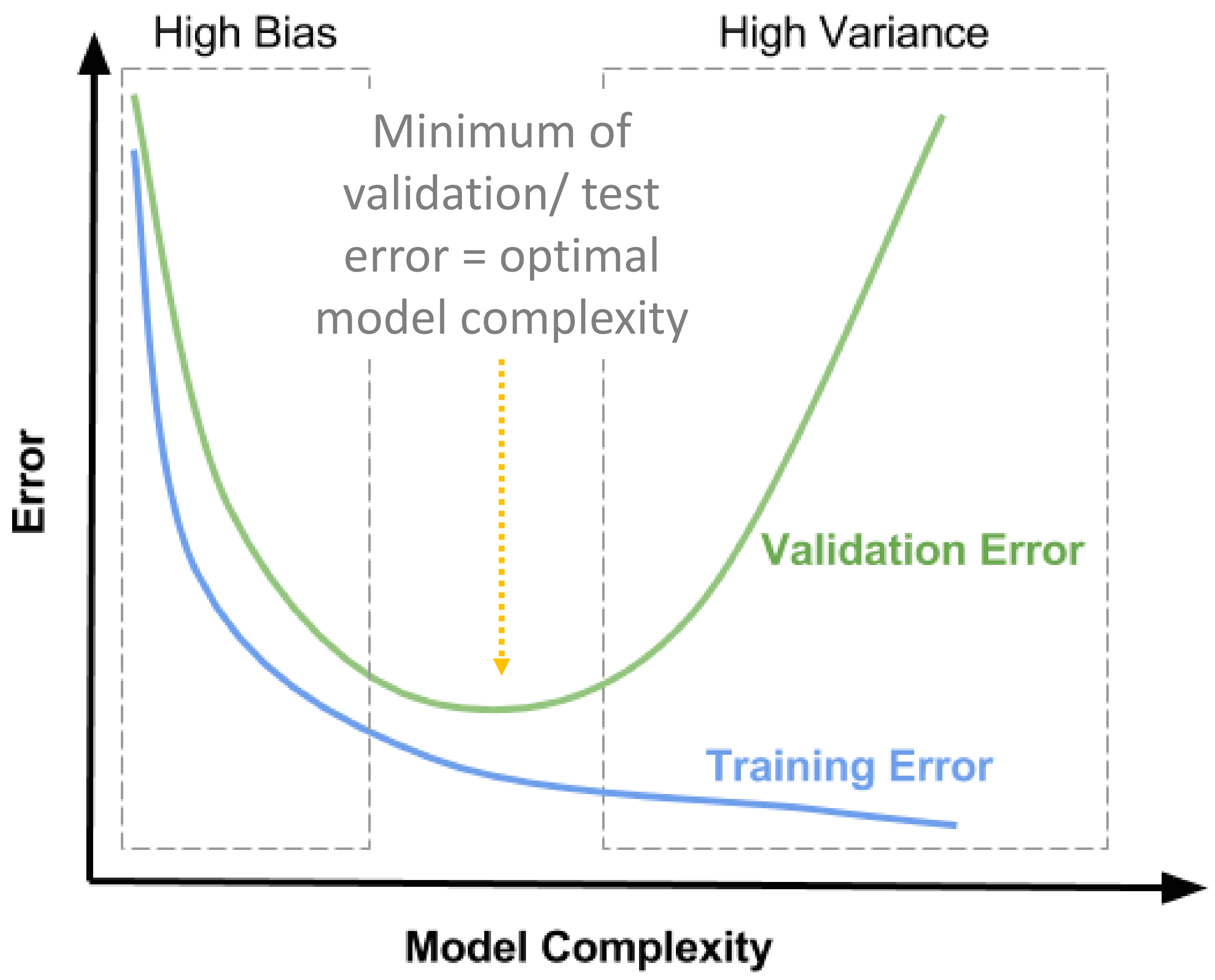
Linear Regression Review

Occam's Razor and Statistics

- **In statistics Parsimony is the analogue of Occam's razor**
- Between two equal models (performance-wise) we prefer the simpler one
- Simpler = fewer variables
- But which variables are superfluous?! (unnecessary)
- Lasso and Ridge are key methods to derive parsimonious linear models


CORE PRINCIPLES IN RESEARCH
OCCAM'S RAZOR
"WHEN FACED WITH TWO POSSIBLE EXPLANATIONS, THE SIMPLER OF THE TWO IS THE ONE MOST LIKELY TO BE TRUE."
OCCAM'S PROFESSOR
"WHEN FACED WITH TWO POSSIBLE WAYS OF DOING SOMETHING, THE MORE COMPLICATED ONE IS THE ONE YOUR PROFESSOR WILL MOST LIKELY ASK YOU TO DO."
WWW.PHDCOMICS.COM
JORGE CHAM ©

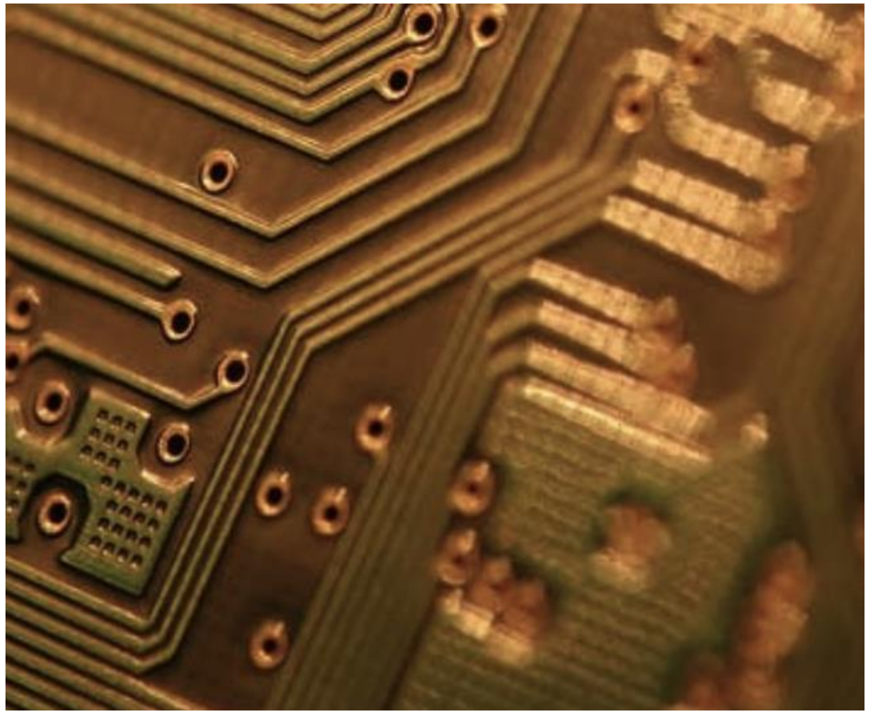
Recall Bias-Variance Tradeoff



- 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)

Ridge Regression

Example: Semiconductor Manufacturing Data

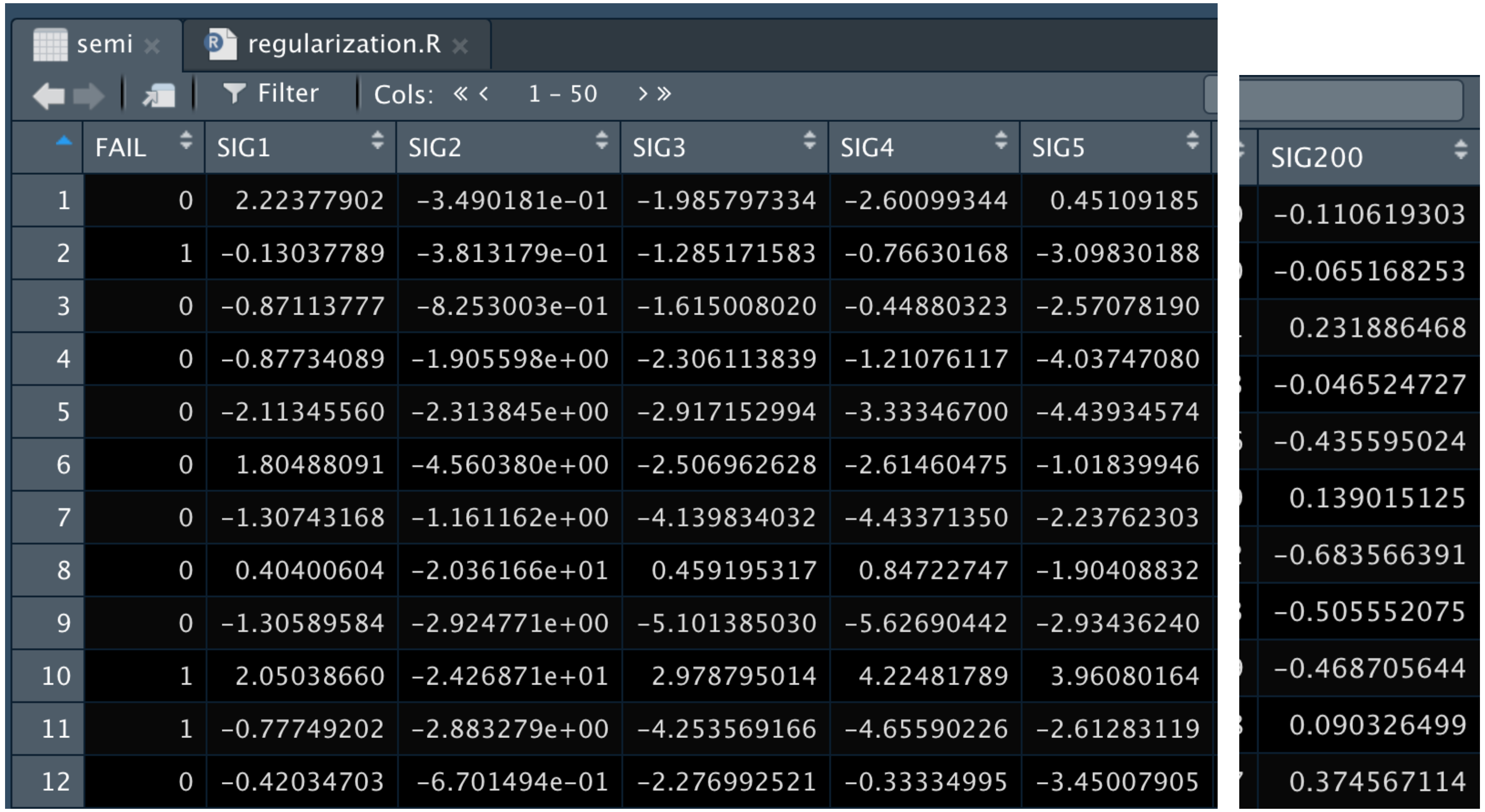


- As an example, let’s use the semiconductor.csv data.
- This data is has **X data on 200** sensors during a semiconductor manufacturing process
- Y (outcome) data is 0 = if chip has a failure, 1 = if chip has a failure

- We can model probability of failure (Y) based on sensor readings as:

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

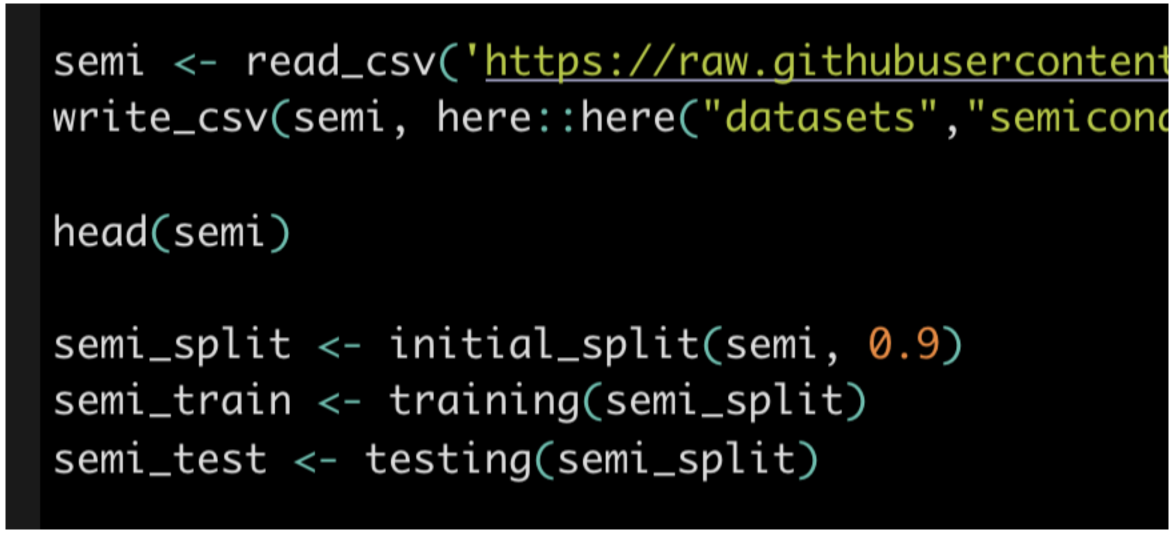
Semiconductor Data



	FAIL	SIG1	SIG2	SIG3	SIG4	SIG5	SIG200
1	0	2.22377902	-3.490181e-01	-1.985797334	-2.60099344	0.45109185	-0.110619303
2	1	-0.13037789	-3.813179e-01	-1.285171583	-0.76630168	-3.09830188	-0.065168253
3	0	-0.87113777	-8.253003e-01	-1.615008020	-0.44880323	-2.57078190	0.231886468
4	0	-0.87734089	-1.905598e+00	-2.306113839	-1.21076117	-4.03747080	-0.046524727
5	0	-2.11345560	-2.313845e+00	-2.917152994	-3.33346700	-4.43934574	-0.435595024
6	0	1.80488091	-4.560380e+00	-2.506962628	-2.61460475	-1.01839946	0.139015125
7	0	-1.30743168	-1.161162e+00	-4.139834032	-4.43371350	-2.23762303	-0.683566391
8	0	0.40400604	-2.036166e+01	0.459195317	0.84722747	-1.90408832	-0.505552075
9	0	-1.30589584	-2.924771e+00	-5.101385030	-5.62690442	-2.93436240	-0.468705644
10	1	2.05038660	-2.426871e+01	2.978795014	4.22481789	3.96080164	0.090326499
11	1	-0.77749202	-2.883279e+00	-4.253569166	-4.65590226	-2.61283119	0.374567114
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

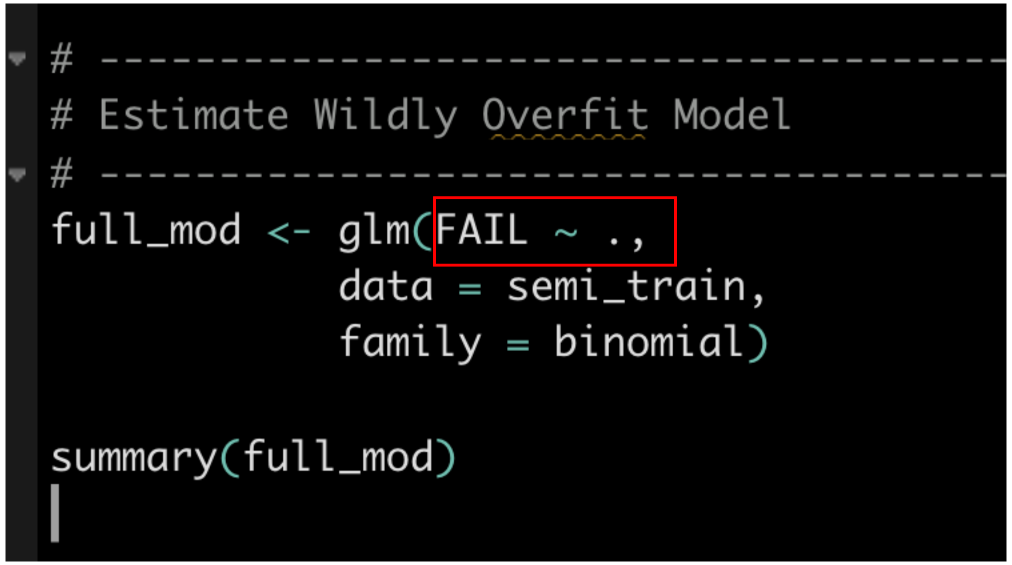
Fit a Logistic Model Using All Sensors



```
semi <- read_csv('https://raw.githubusercontent
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)
```

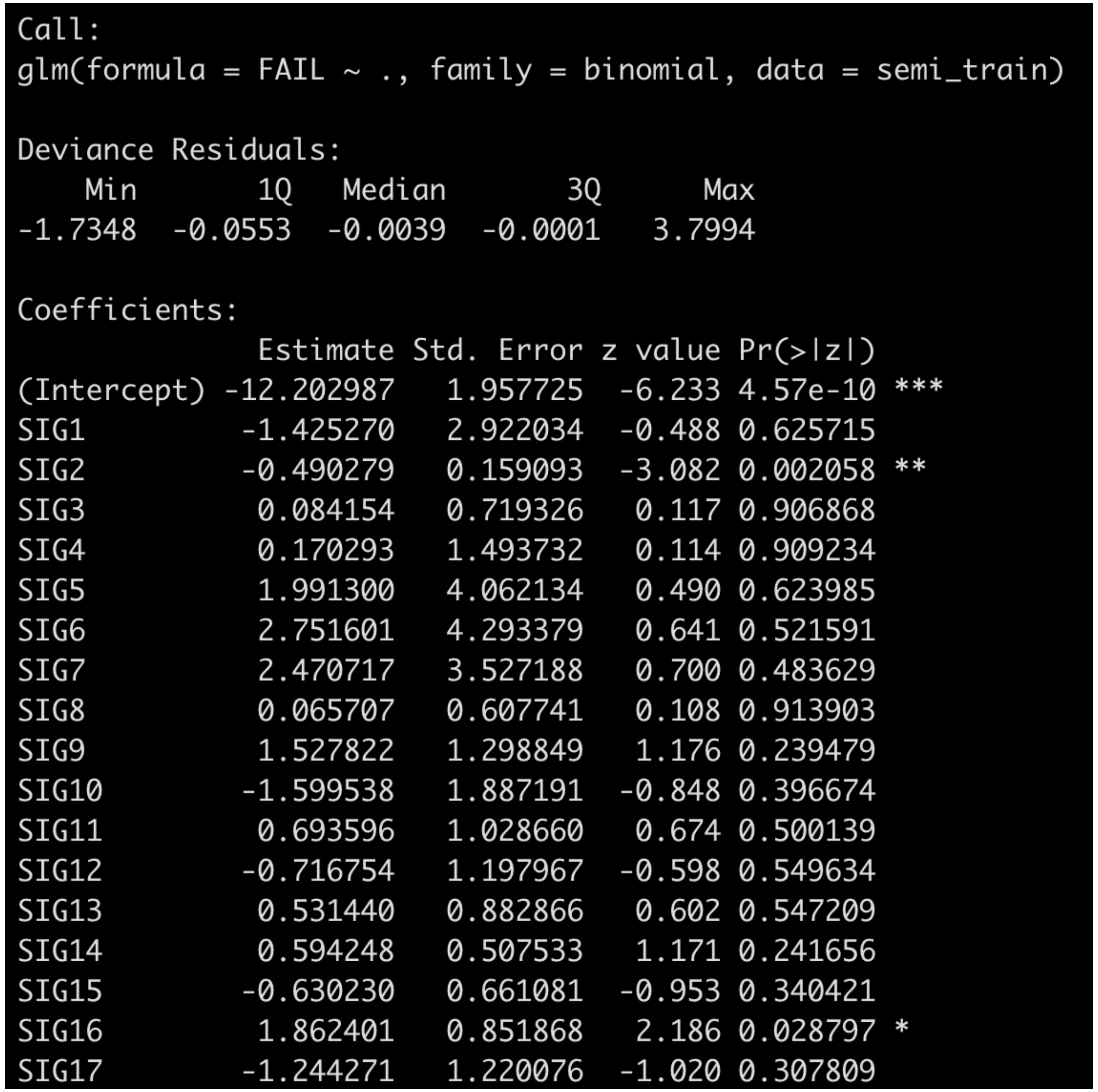


```
# ----------------------------------------
# Estimate Wildly Overfit Model
# ----------------------------------------
full_mod <- glm(FAIL ~ .,
                data = semi_train,
                family = binomial)

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



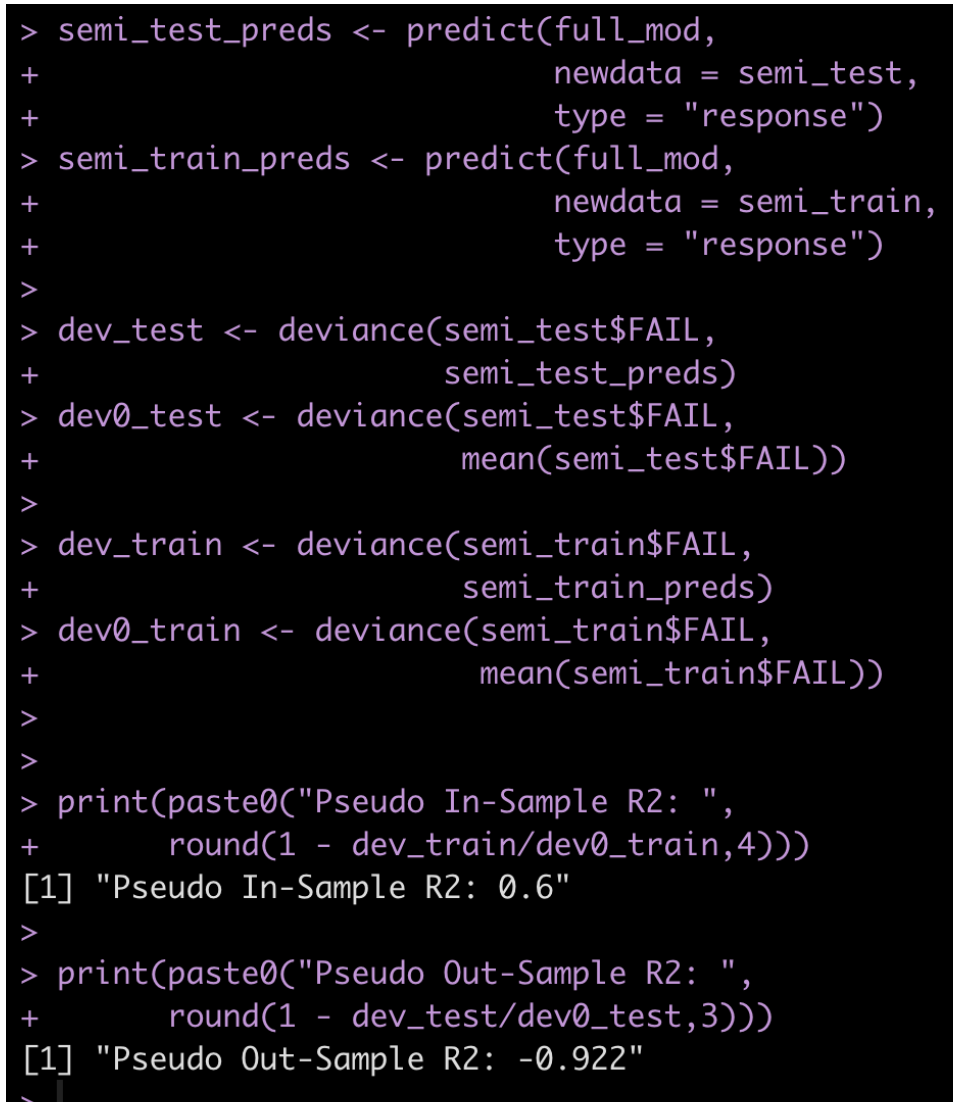
```
Call:
glm(formula = FAIL ~ ., family = binomial, data = semi_train)

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 except 10 variables to be canonically significant even if they are truly equal to zero!

However, We Are Really Overfit



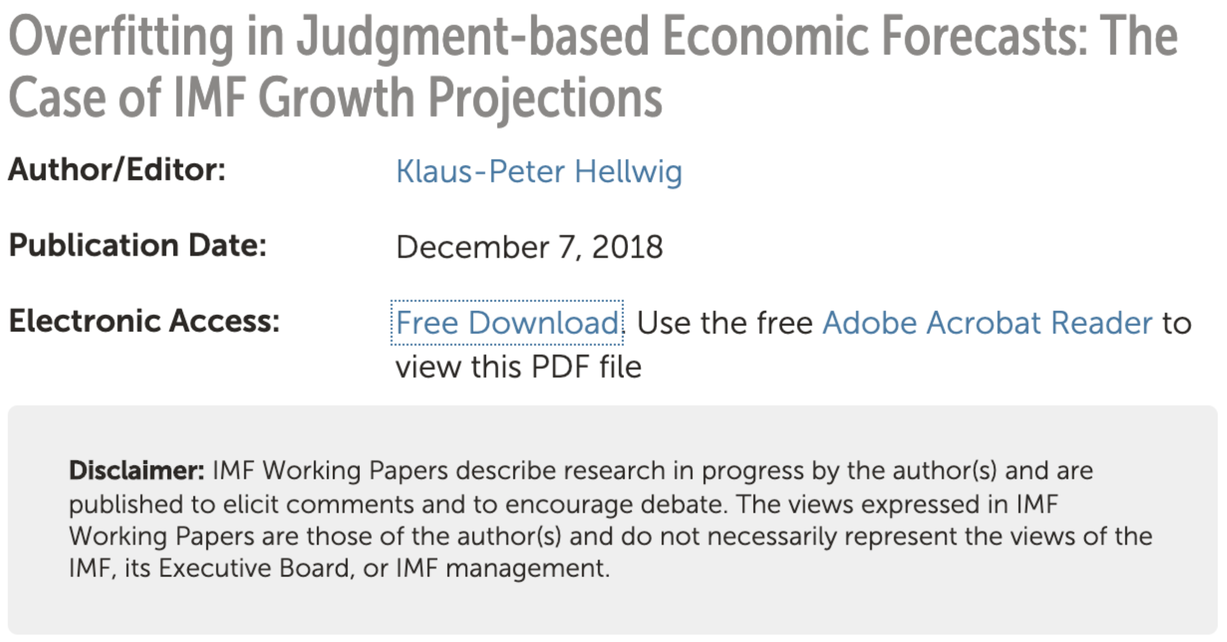
```
> semi_test_preds <- predict(full_mod,
+                            newdata = semi_test,
+                            type = "response")
> semi_train_preds <- predict(full_mod,
+                            newdata = semi_train,
+                            type = "response")
>
> dev_test <- deviance(semi_test$FAIL,
+                      semi_test_preds)
> dev0_test <- deviance(semi_test$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

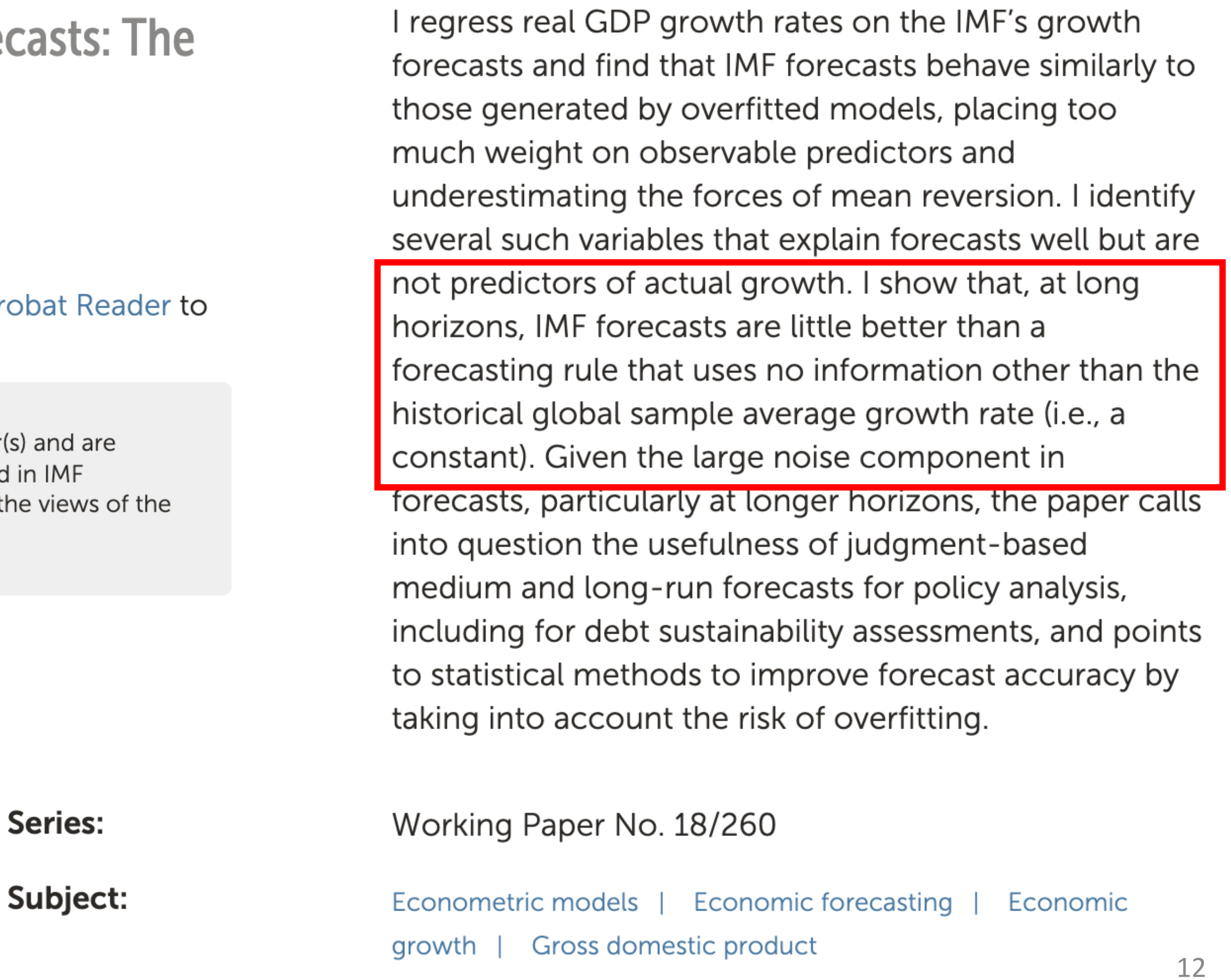
Even IMF Overfits Forecasting Models!



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.



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.

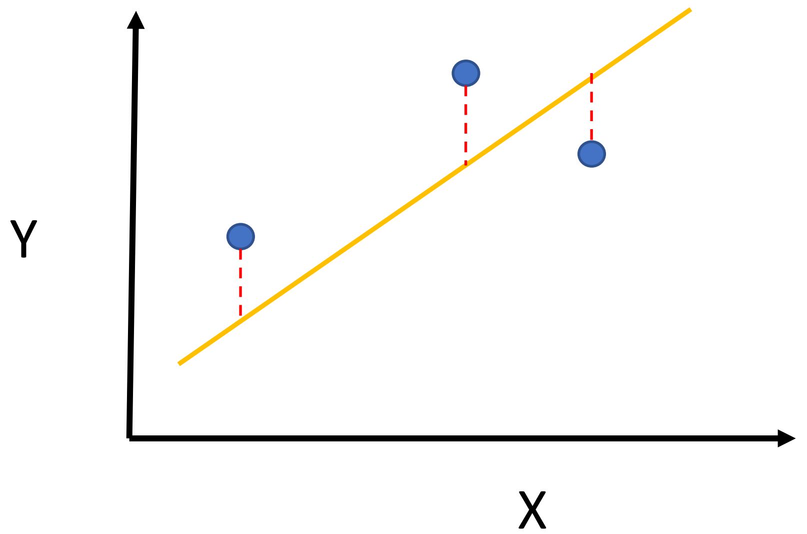
Series:	Working Paper No. 18/260
Subject:	Econometric models | Economic forecasting | Economic growth | Gross domestic product

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

Least Squares (OLS Estimator)

$$\hat{\beta}\; minimizes: \underbrace{\sum_{i=1}^{N}(y_i - \beta_0 - x_{i1}\beta_1)^2}_{\text{Least squares minimizes the sum of squared residuals (e.g. } y_i - \hat{y}\text{)}}$$

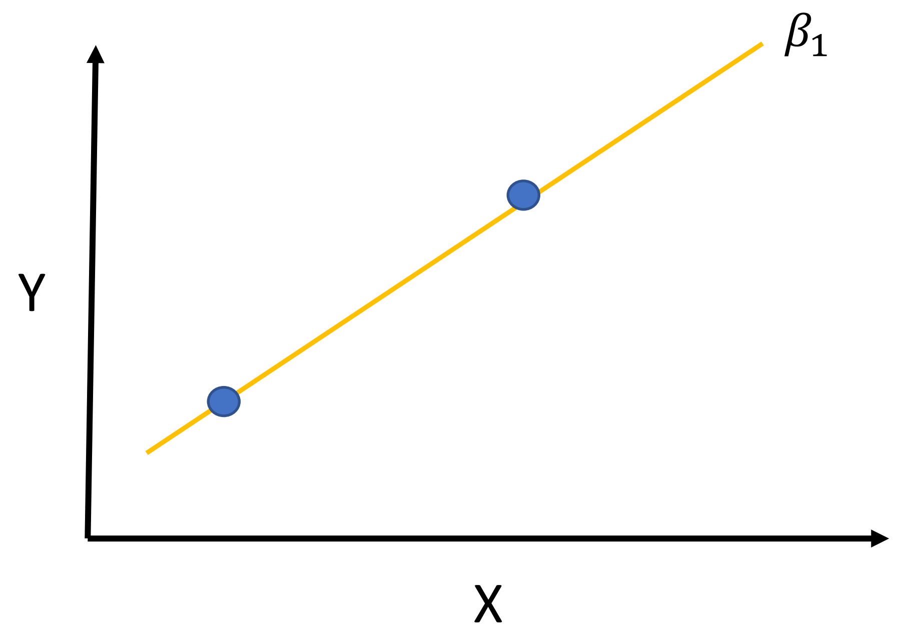
Let’s set K = 1 (i.e. one explanatory variable to make this easier)



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

Ridge Regression Idea

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

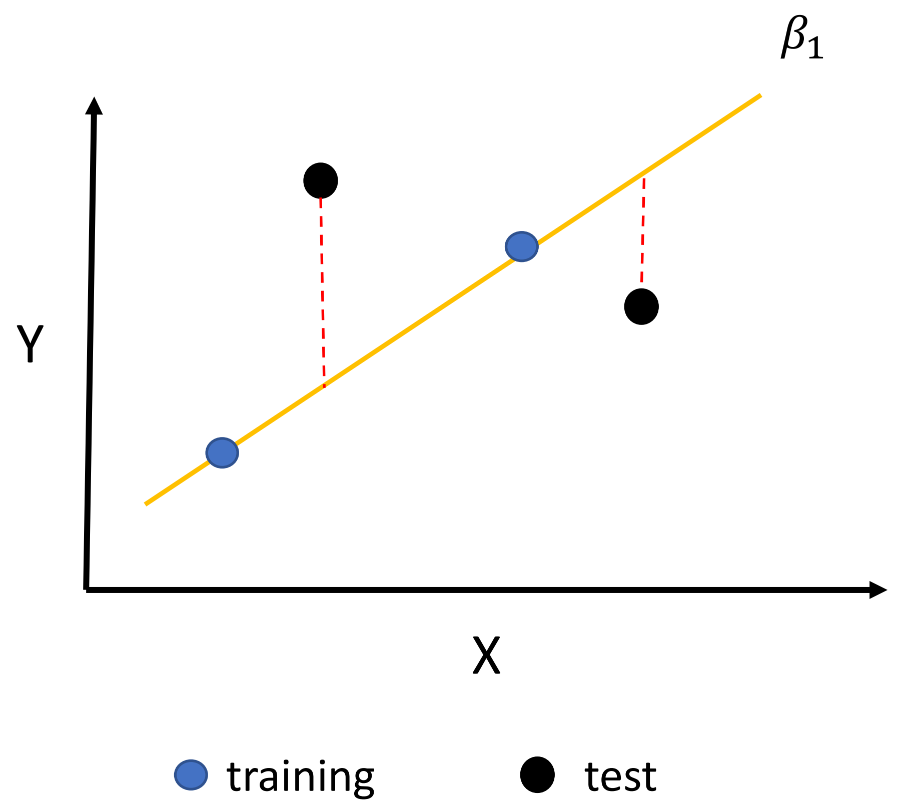


Ridge regression in contrast minimizes the OLS residuals plus the squared slope of beta times λ

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?

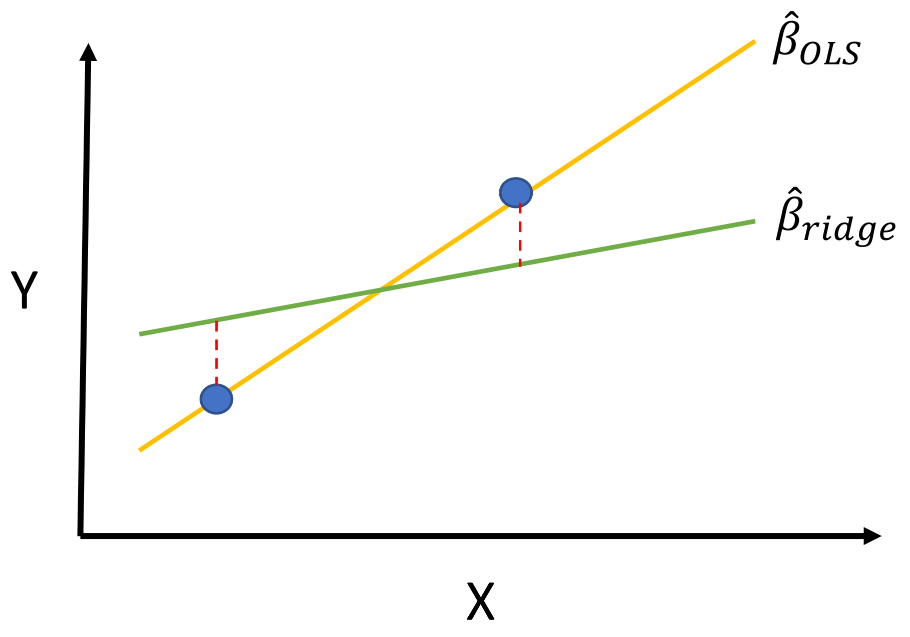
Imaging we had the following data in our test set

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:

residuals + $1 \cdot (\beta_1)^2$

= residuals + $1 \cdot (\beta_1)^2$

Suppose the OLS slope = 2

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

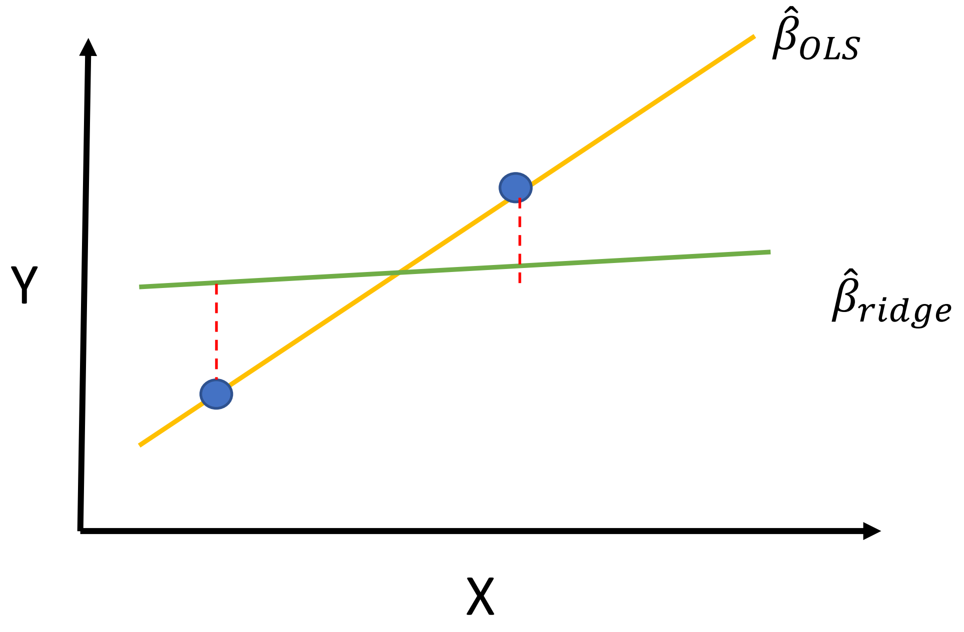
residuals_ridge + $1 \cdot (1)^2$

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:

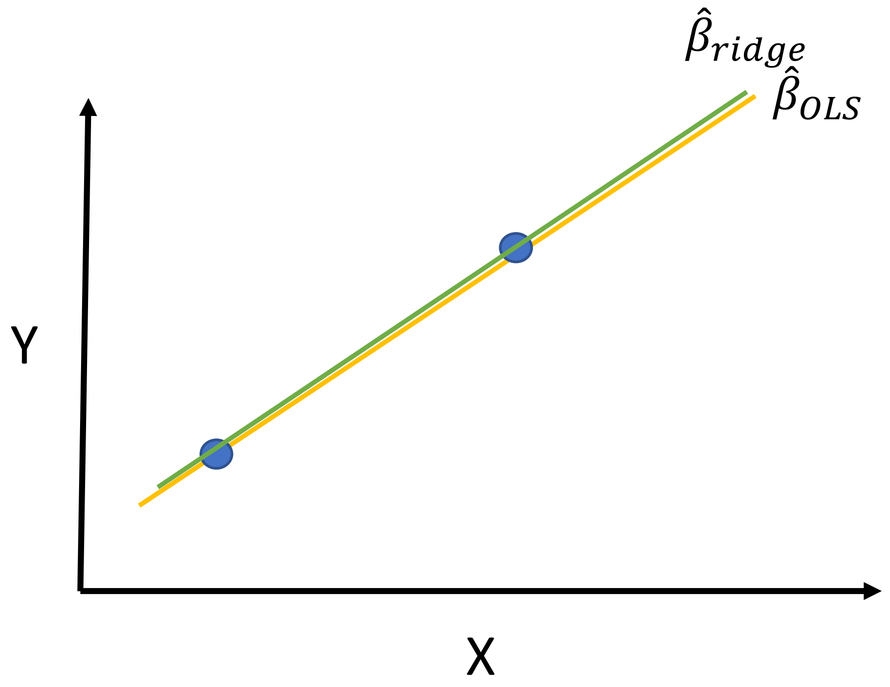
residuals + $100 \cdot (0.001)^2$

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

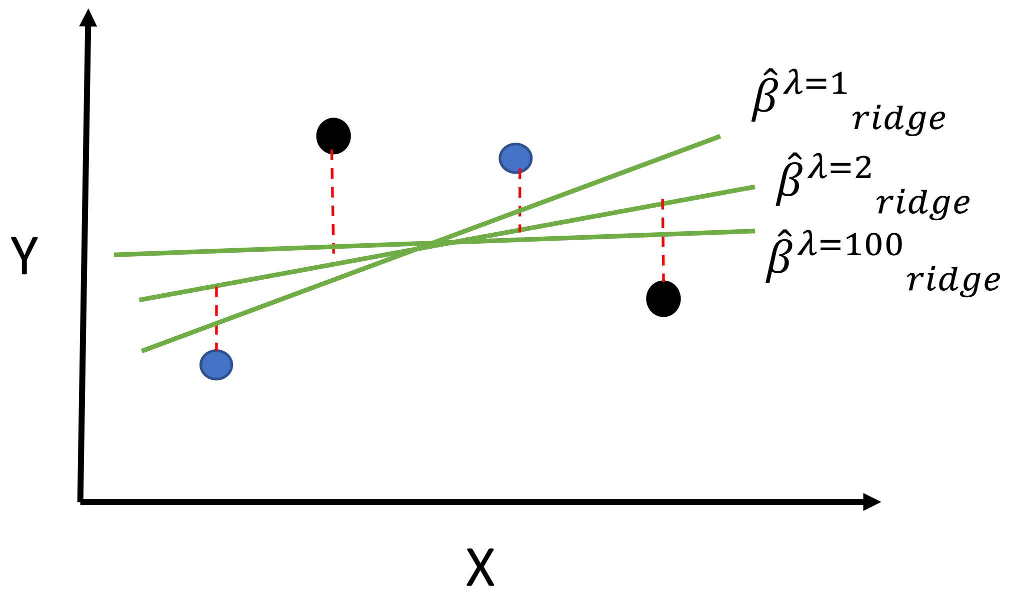
= residuals + 0

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 λ => More Penalization, Smaller Coefficients

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



$\hat{\beta}^{\lambda=1}{}_{ridge}$
$\hat{\beta}^{\lambda=2}{}_{ridge}$
$\hat{\beta}^{\lambda=100}{}_{ridge}$
Y
X

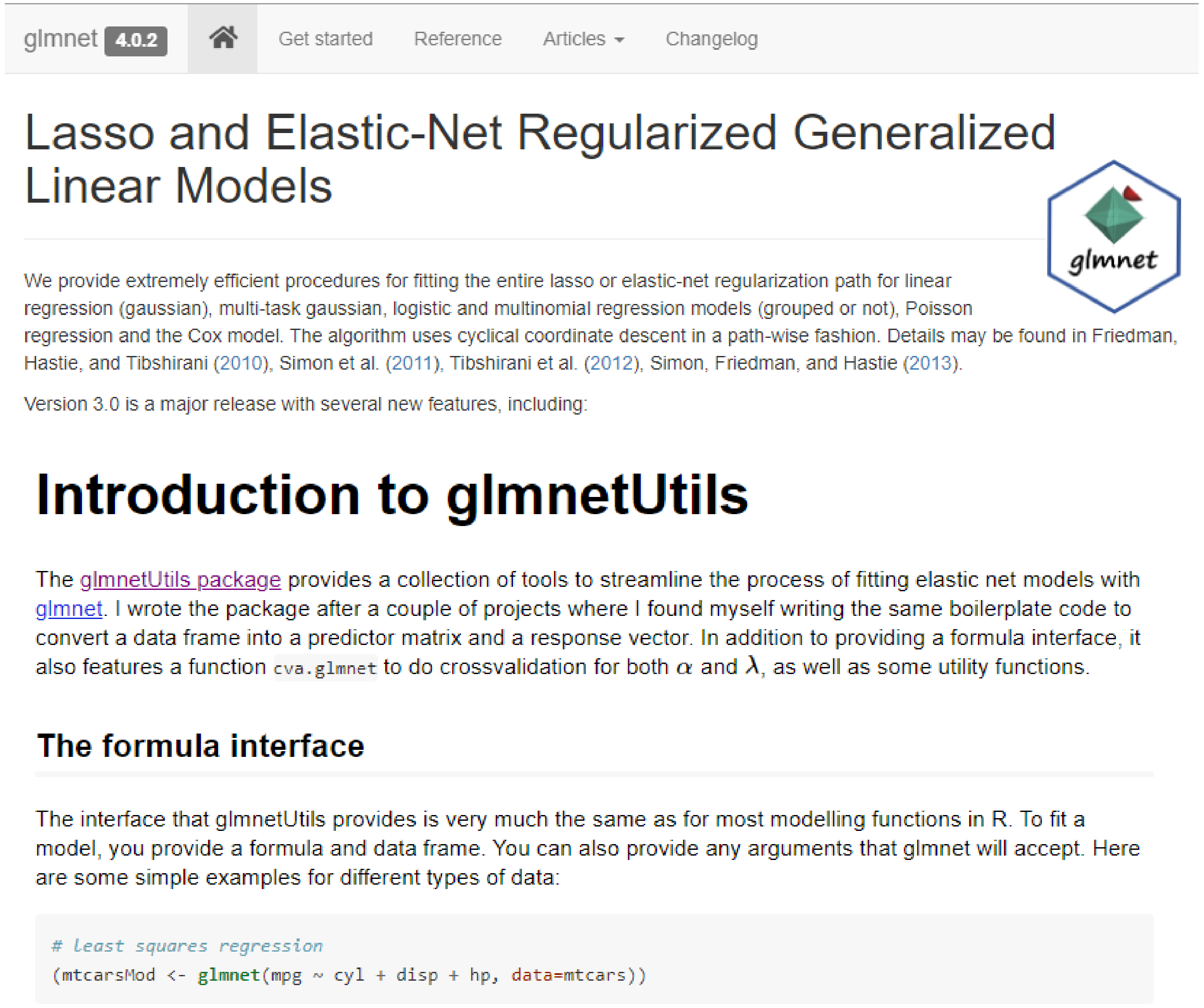
So how do we choose λ?

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



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.

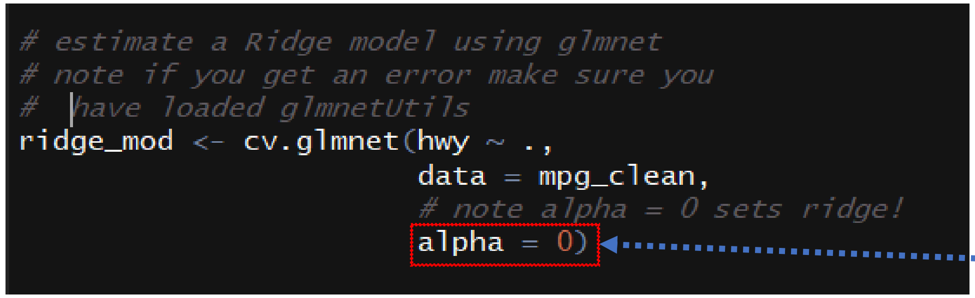
The formula interface

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))
```

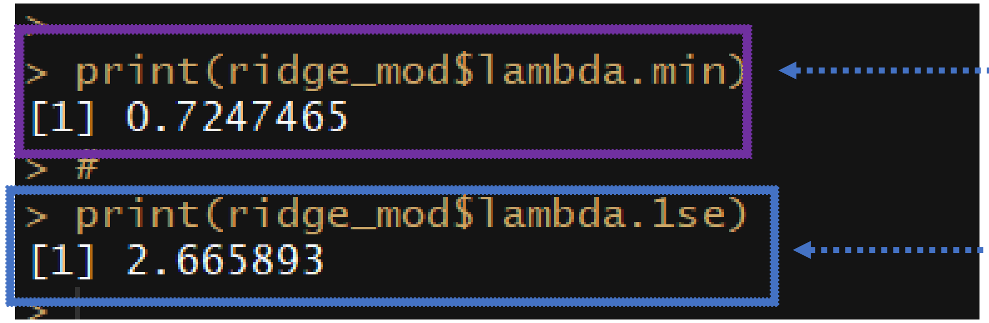
- glmnet quickly estimates Ridge and Lasso models
- It's one of the best package in R or any language (ported to python in 2015) but can be difficult to work with
- glmnetUtils is a helper package that makes our lives much easier
- Make sure to install both glmnet and glmnetUtils!

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)
```

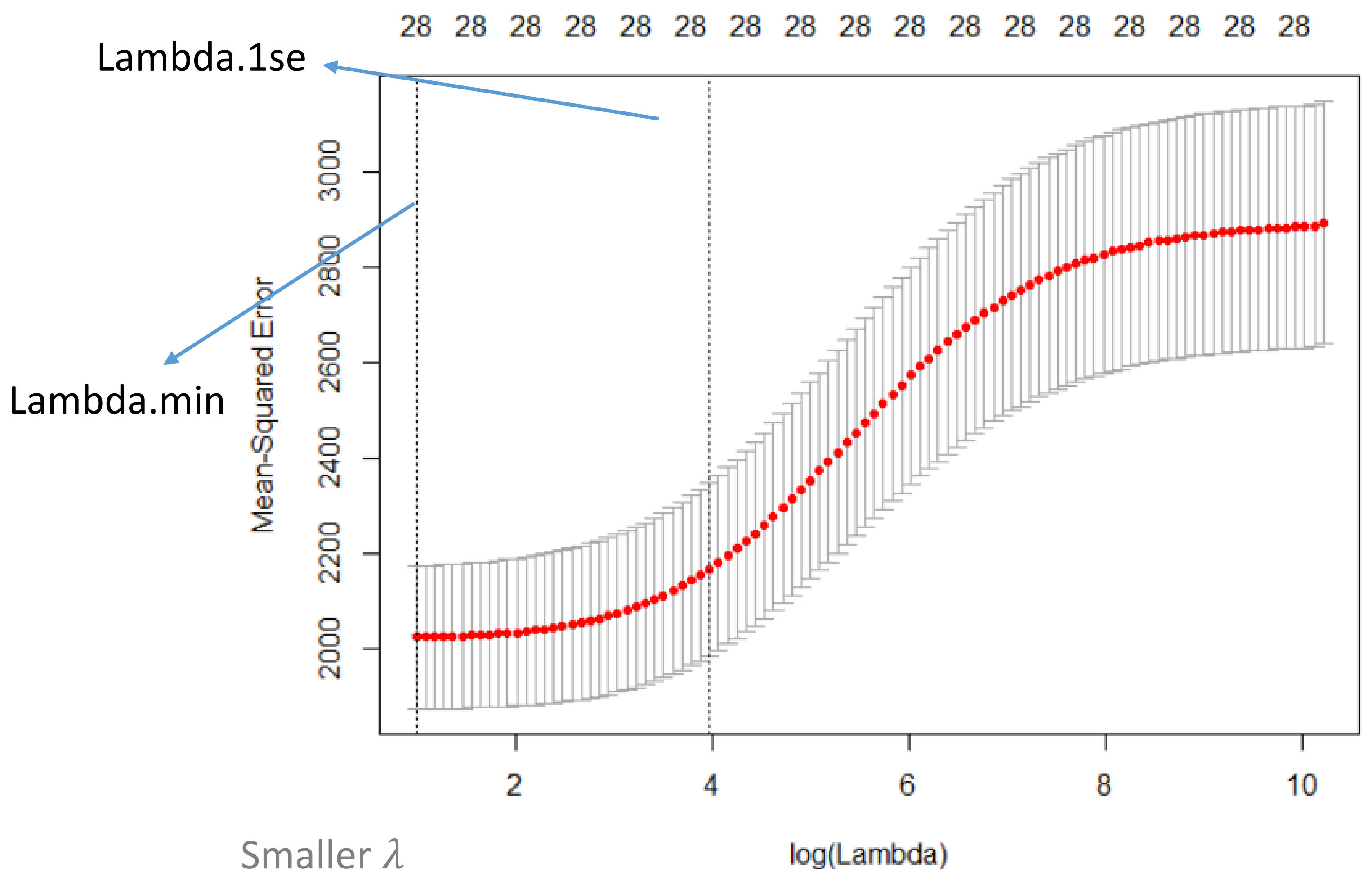
- cv.glmnet estimates a lasso or ridge model. Automatically performs cross-validation to select optimal lambda!
- We must set alpha = 0 to signify ridge model



```
> 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

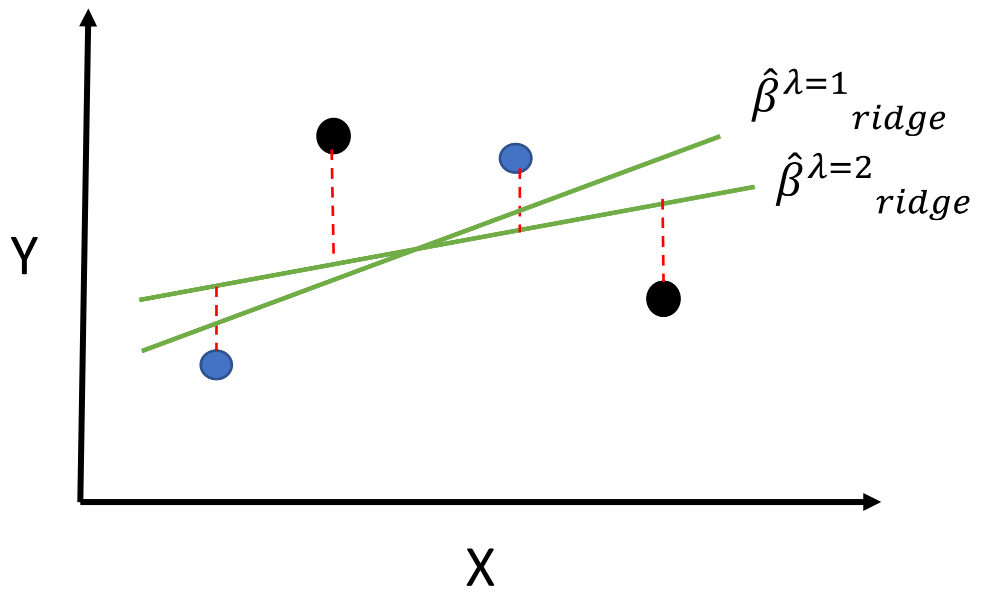
K-Fold Cross-Validation

How to choose λ?

- λ determines how parsimonious our final model is
  - Higher λ -> more parsimonious = fewer variables
  - Lower λ -> less parsimonious = more variables
- In practice, we will estimate several values of λ and see which best approximates out of sample-fit.
- **Cross-validation** is a clever technique that approximates out of sample fit.
- Why not just estimate against the test set? We want to save our test set to estimate final “tuned” model performance after we’ve chosen λ

Choosing λ for Ridge?

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


$\hat{\beta}^{\lambda=1}{}_{ridge}$
$\hat{\beta}^{\lambda=2}{}_{ridge}$
Y
X

So how do we choose λ?

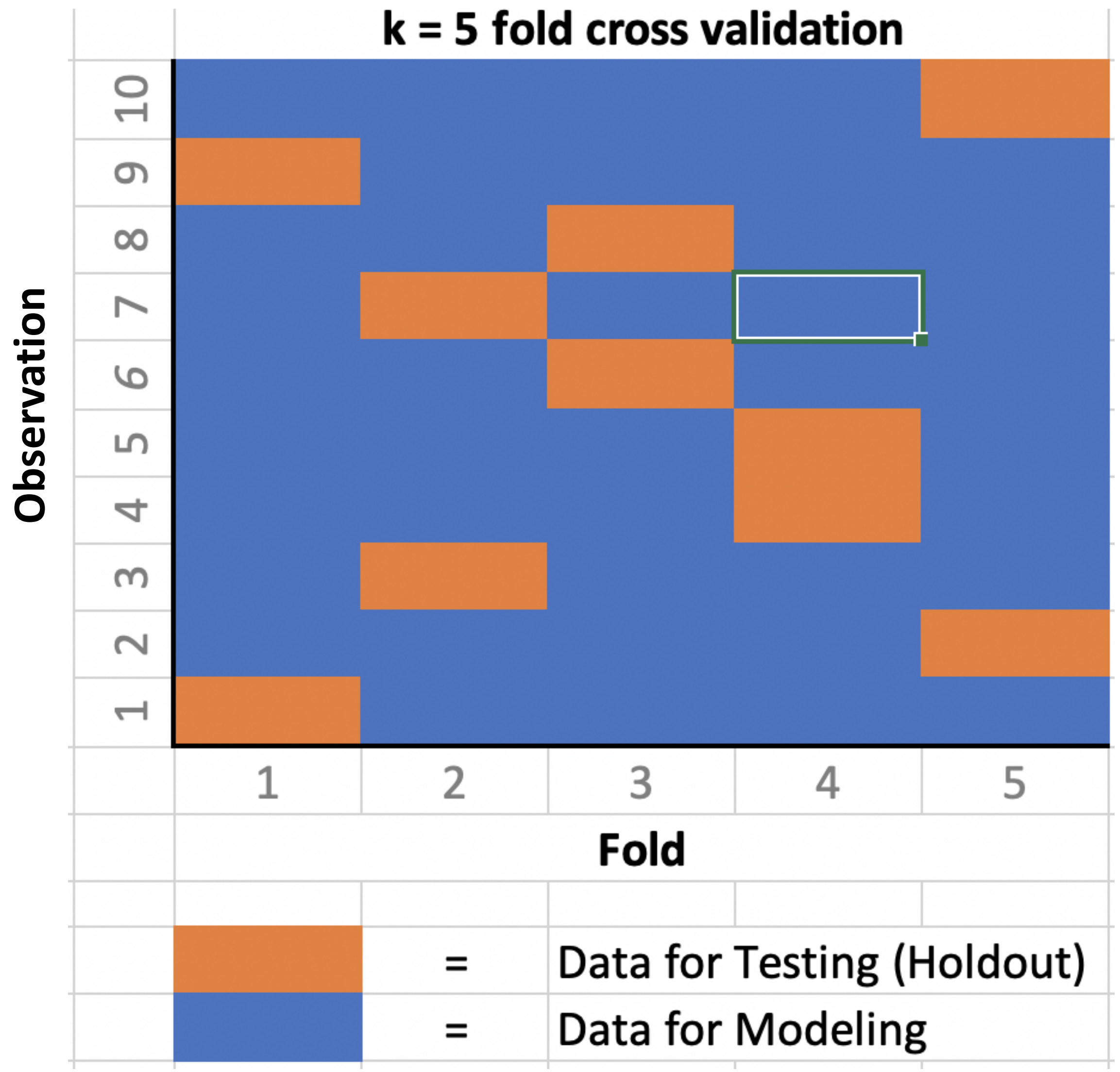
In practice we estimate a many models with many different values of λ

We pick a min and max lambda (say 0 and 10), then choose some points in-between

Our **optimal** λ^* is the lambda that minimizes cross-validated error

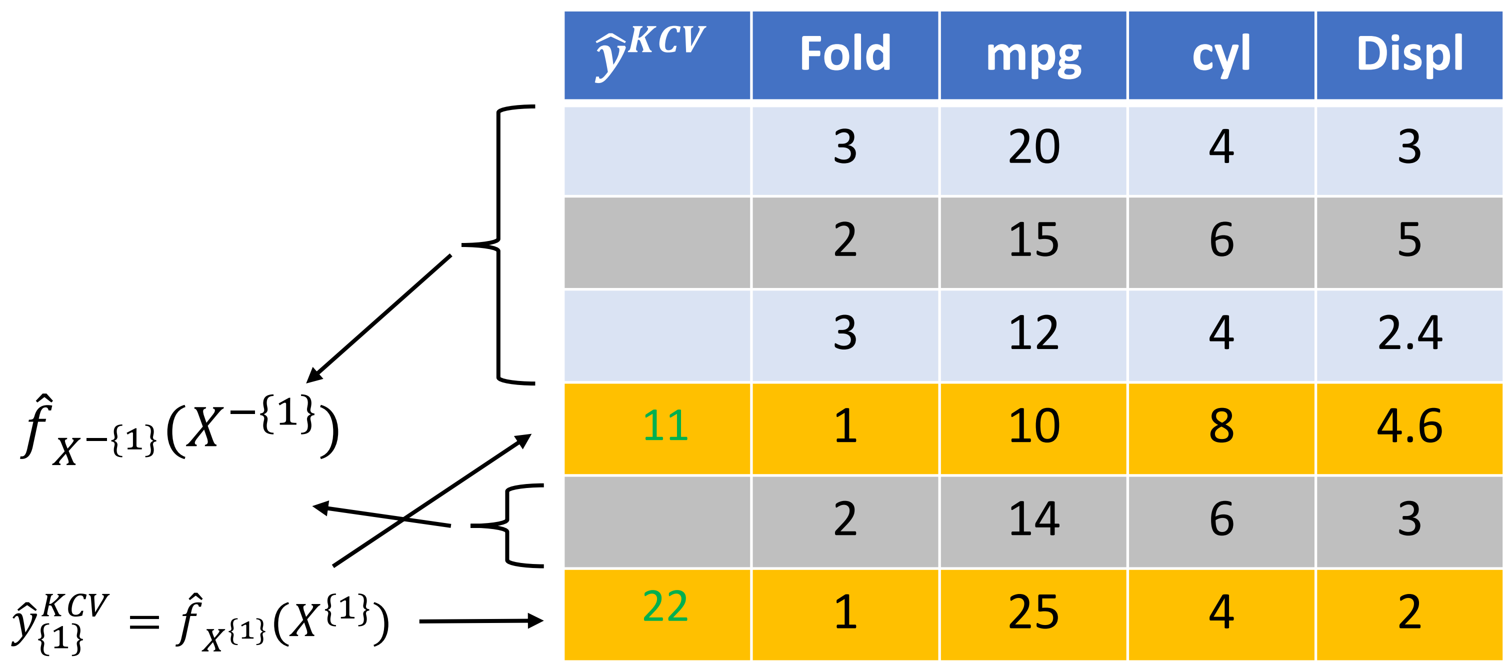
K-Fold Cross-Validation

- With k-fold **cross-validation** we first 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
- Proceed until we have yhats for every group



Resampling: K-Fold Cross-Validation

- We start by randomly assigning each data point to one of k folds
- Here we are setting k = 3
- We fit a model excluding data from fold 1
- That model is used to predict into fold 1



$\hat{f}_{X^{-\{1\}}}(X^{-\{1\}})$

$\hat{y}^{KCV}_{\{1\}} = \hat{f}_{X^{\{1\}}}(X^{\{1\}})$

$\hat{y}^{KCV}$	Fold	mpg	cyl	Displ
	3	20	4	3
	2	15	6	5
	3	12	4	2.4
11	1	10	8	4.6
	2	14	6	3
22	1	25	4	2

$X^{-\{1\}}$: X excluding fold 1

Resampling: K-Fold Cross-Validation

- Here we are setting k = 3
- Next we fit a model excluding observations in fold 2
- That model is used to predict into fold 2

$$\hat{f}_{X^{-\{2\}}}(X^{-\{2\}})$$

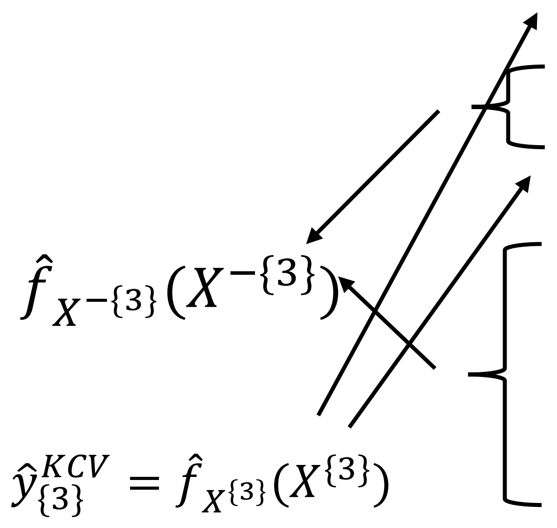
$$\hat{y}^{KCV}_{\{2\}} = \hat{f}_{X^{\{2\}}}(X^{\{2\}})$$

$\hat{y}^{KCV}$	Fold	mpg	cyl	Displ
	3	20	4	3
18	2	15	6	5
	3	12	4	2.4
11	1	10	8	4.6
15	2	14	6	3
22	1	25	4	2

$X^{-\{2\}}$: X excluding fold 2

Resampling: K-Fold Cross-Validation

- Here we are setting k = 3
- Next we fit a model excluding observations in fold 3
- That model is used to predict into fold 3



$\hat{f}_{X^{-\{3\}}}(X^{-\{3\}})$
$\hat{y}^{KCV}_{\{3\}} = \hat{f}_{X^{\{3\}}}(X^{\{3\}})$

$\hat{y}^{KCV}$	Fold	mpg	cyl	Displ
22	3	20	4	3
18	2	15	6	5
12	3	12	4	2.4
11	1	10	8	4.6
15	2	14	6	3
22	1	25	4	2

$X^{-\{3\}}$: X excluding fold 3

K-Fold CV Versus LOOCV

- Advantages of K-Fold CV over LOOCV
  - Only need to estimate K models
- Disadvantages
  - Higher variance (more uncertainty in y_hats)
- Because of computational cost K-Fold CV more commonly used

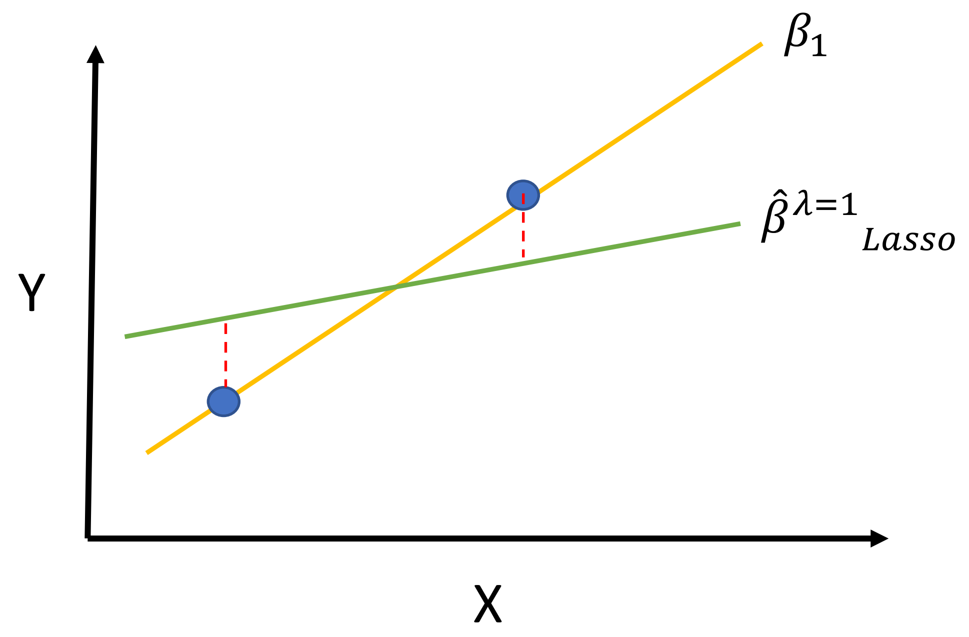
$\hat{y}^{KCV}$	mpg	cyl	Displ
18	20	4	3
16	15	6	5
12	12	4	2.4
11	10	8	4.6
11	14	6	3
22	25	4	2

$$MSE_{KCV} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i^{KCV}\right)^2$$

Lasso Regression

Lasso Regression Idea

$$\hat{\beta}_{Lasso}\ minimizes:\ residuals + \lambda \cdot (\,|\beta_1| + |\beta_2| + \cdots + |\beta_k|\,)$$



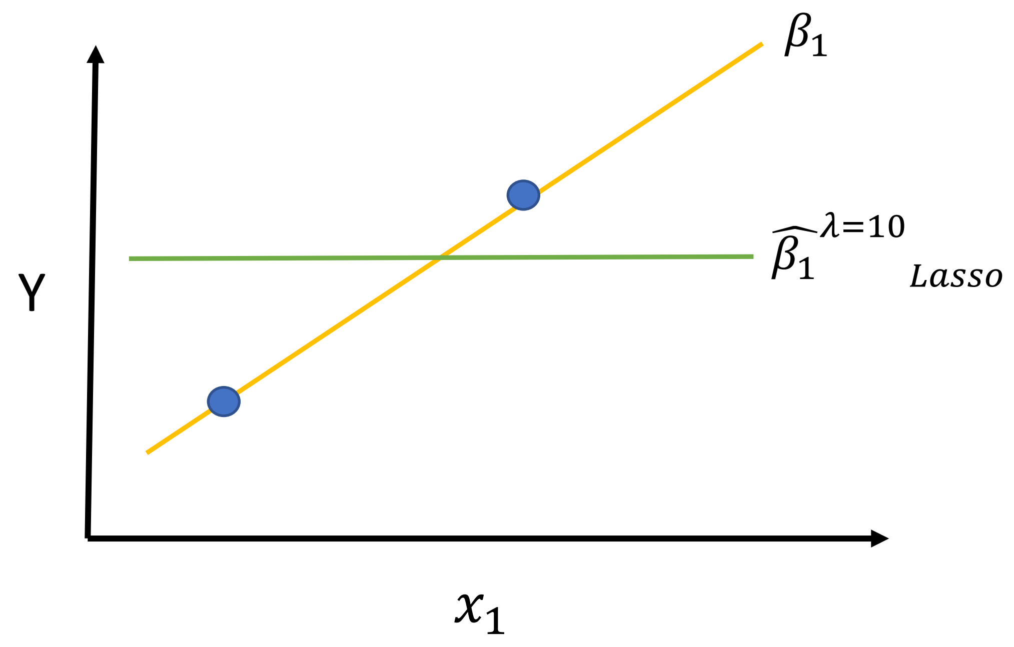
Lasso minimizes the residuals plus lambda times the absolute value of the slope coefficients

Lasso coefficients are still smaller than OLS coefficients

Lasso still accepts a little bias for (hopefully) less variance

Key Lasso Property: Variable Selection

$$\hat{\beta}_{Lasso}\ minimizes:\ residuals + \lambda \cdot (\ |\cancel{\beta_1}| + |\beta_2|)$$

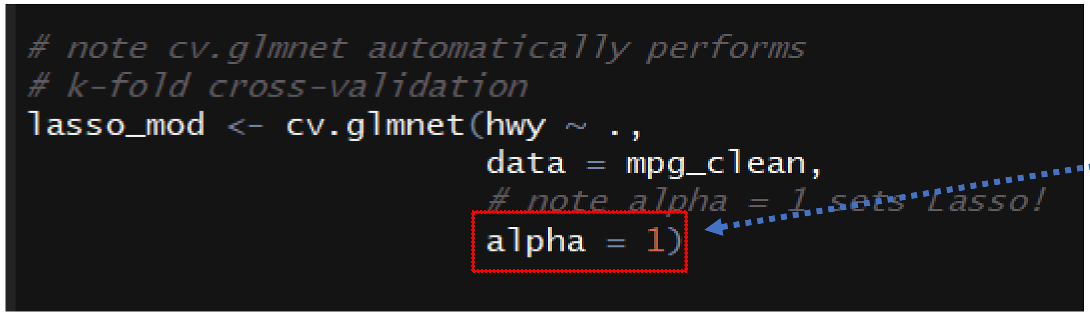


For large values of λ, some slope coefficients will be chosen to be exactly zero

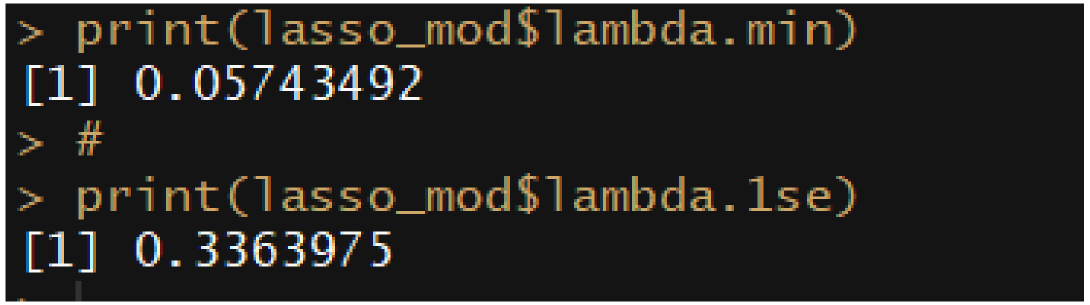
E.g. if we set $\lambda = 10$, maybe $\beta_1^{lasso} = 0$ but $\beta_1^{lasso} != 0$

If that happens we effectively remove β_1 from the equation, and we have a variable selection algorithm

Lasso Model with glmnetUtils



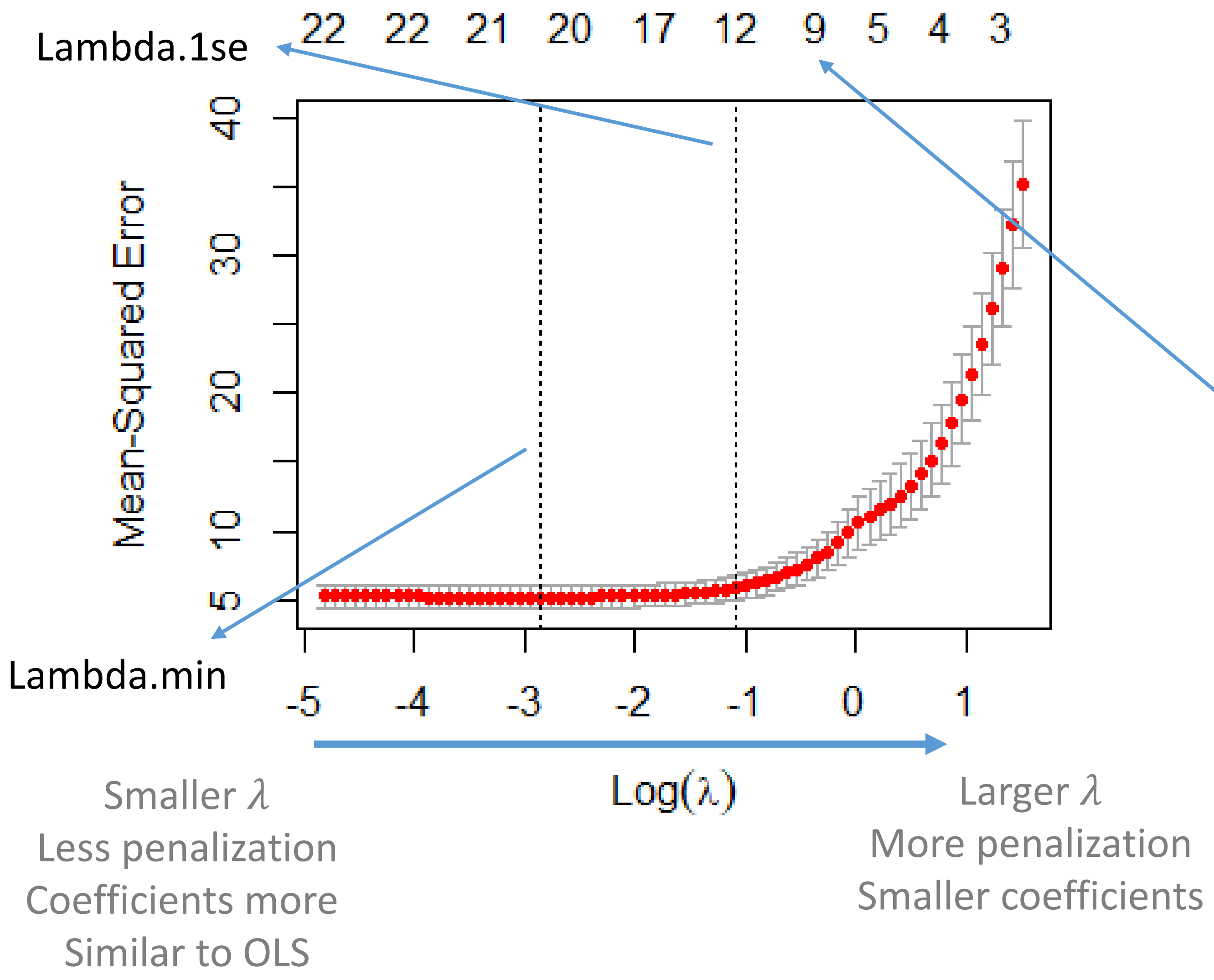
```
# note cv.glmnet automatically performs
# k-fold cross-validation
lasso_mod <- cv.glmnet(hwy ~ .,
                       data = mpg_clean,
                       # note alpha = 1 sets Lasso!
                       alpha = 1)
```



```
> print(lasso_mod$lambda.min)
[1] 0.05743492
> #
> print(lasso_mod$lambda.1se)
[1] 0.3363975
```

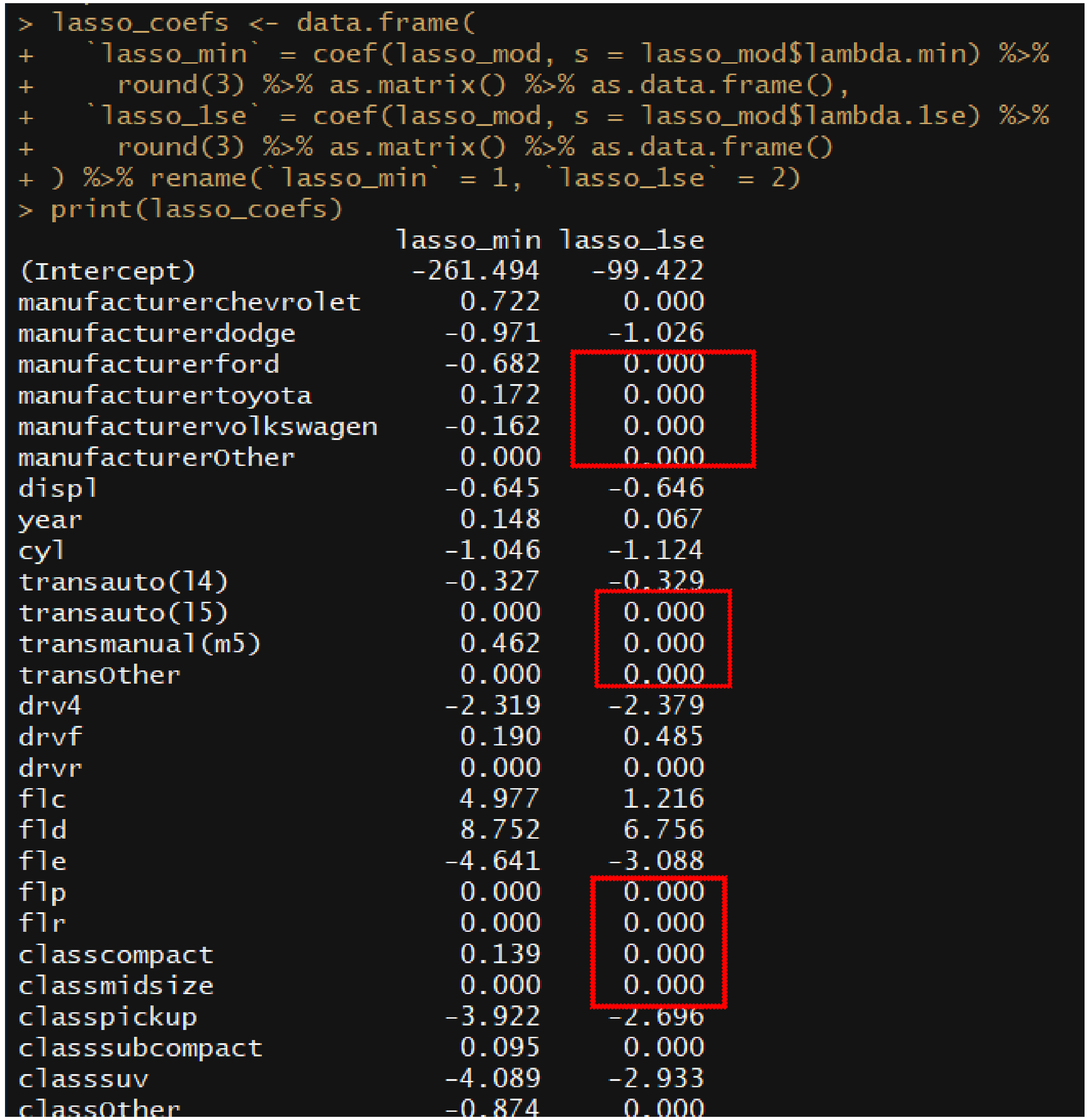
- We estimate lasso using cv.glmnet
- Here we must set alpha = 1 to estimate Lasso
- Again we get values of lambda.min (minimizes cross-validated MSE) and lambda.1se (minimum plus 1 SE)

Lasso Cross-Validated MSE Plot



- Lasso MSE plot is very similar
- Top number indicates number of non-zero coefficients for each value of lambda!
- E.g. at this value of lambda, 9 variables are non-zero
- We still have lambda.1se and lambda.min vertical dashed lines but lambda.1se generally shrinks more variables to exactly zero!

Lasso Coefficient Vector



```
> lasso_coefs <- data.frame(
+   `lasso_min` = coef(lasso_mod, s = lasso_mod$lambda.min) %>%
+     round(3) %>% as.matrix() %>% as.data.frame(),
+   `lasso_1se` = coef(lasso_mod, s = lasso_mod$lambda.1se) %>%
+     round(3) %>% as.matrix() %>% as.data.frame()
+ ) %>% rename(`lasso_min` = 1, `lasso_1se` = 2)
> print(lasso_coefs)
                     lasso_min lasso_1se
(Intercept)           -261.494   -99.422
manufacturerchevrolet    0.722     0.000
manufacturerdodge       -0.971    -1.026
manufacturerford        -0.682     0.000
manufacturertoyota       0.172     0.000
manufacturervolkswagen  -0.162     0.000
manufacturerOther        0.000     0.000
displ                   -0.645    -0.646
year                     0.148     0.067
cyl                     -1.046    -1.124
transauto(l4)           -0.327    -0.329
transauto(l5)            0.000     0.000
transmanual(m5)          0.462     0.000
transOther               0.000     0.000
drv4                    -2.319    -2.379
drvf                     0.190     0.485
drvr                     0.000     0.000
flc                      4.977     1.216
fld                      8.752     6.756
fle                     -4.641    -3.088
flp                      0.000     0.000
flr                      0.000     0.000
classcompact             0.139     0.000
classmidsize             0.000     0.000
classpickup             -3.922    -2.696
classsubcompact          0.095     0.000
classsuv                -4.089    -2.933
classOther              -0.874     0.000
```

- We can build the lasso coefficient vector as we did for Ridge

- Note the higher the lambda (lambda.1se > lambda.min) the more variables that are “shrunk” to zero

- Lasso sets coefficients = 0 if they do not improve the cross-validated MSE

- Ridge will just shrink these coefficients towards zero but will never set them exactly = 0

Another way to write Lasso

Lasso

$$\min_{\beta} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \le s$$

Lasso with two variables

$$\min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2})^2 \quad \text{subject to} \quad |\beta_1| + |\beta_2| \le s$$

In other words: I give you s as a budget (like setting some lambda)

You can increase your coefficients but the sum of the absolute value of them must be less than s

Another way to write Ridge

Ridge $$\min_{\beta} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \left(\beta_j\right)^2 \leq s$$

Ridge with two variables $$\min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2})^2 \quad \text{subject to} \quad (\beta_1)^2 + (\beta_2)^2 \leq s$$

In other words: I give you s as a budget (like setting some lambda)

You can increase your coefficients but the sum of
the absolute value of them must be less than s

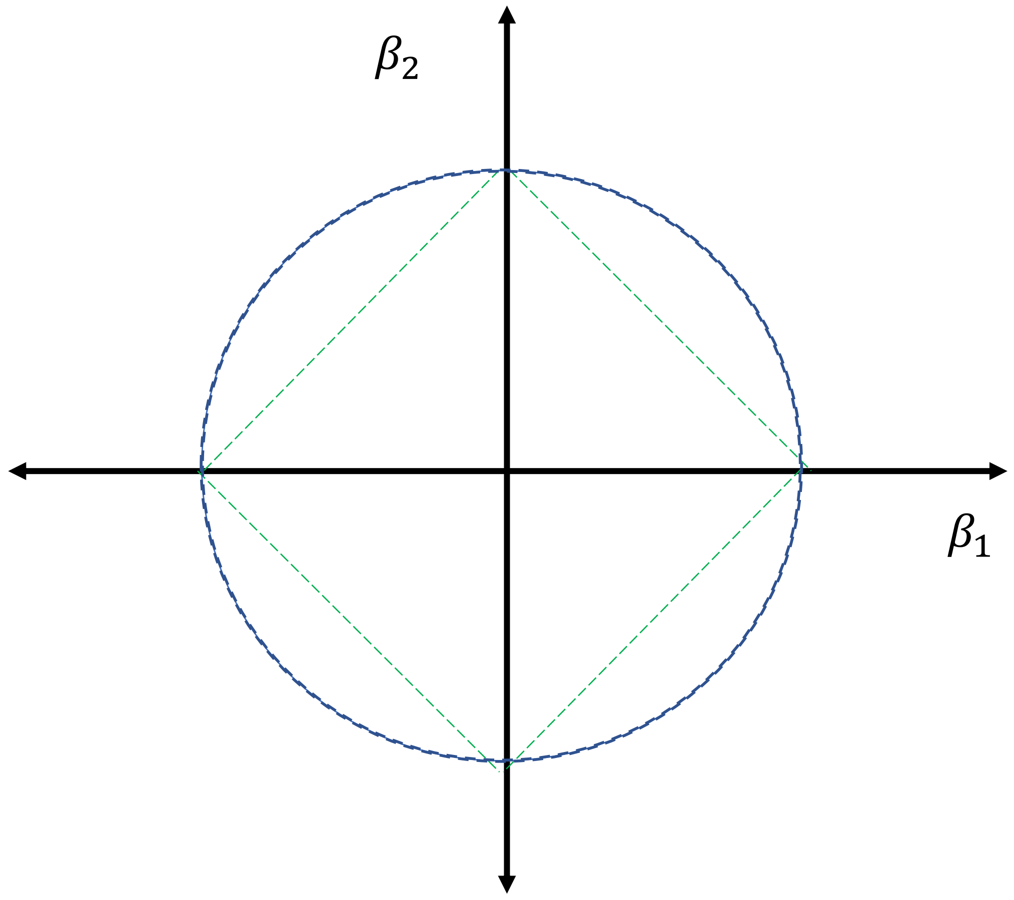
Ridge Versus Lasso Penalty

Ridge penalty $(\beta_1)^2 + (\beta_2)^2 \leq 1$

Lasso penalty $|\beta_1| + |\beta_2| \leq 1$

Let's pick an arbitrary value of s = 1

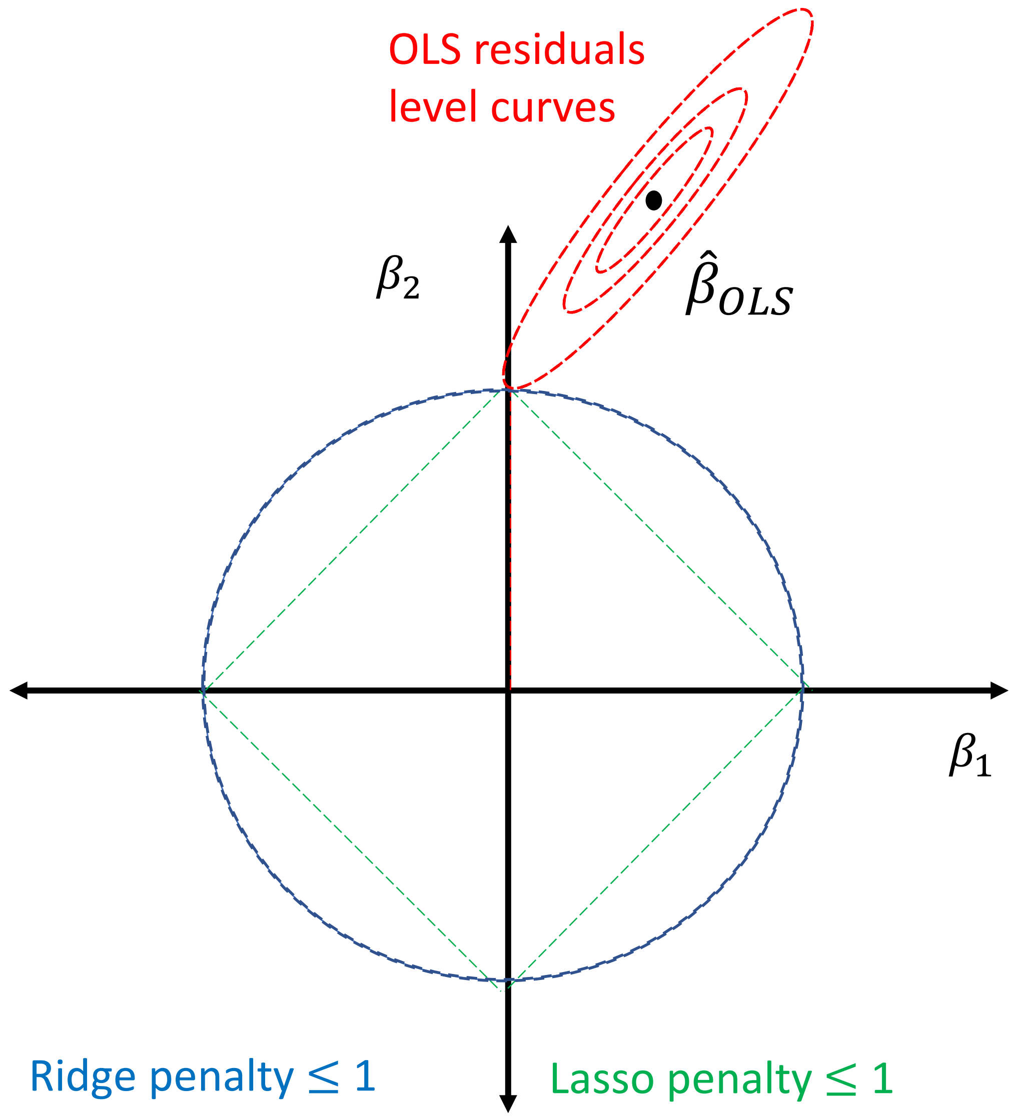
What do these look like graphically?



Ridge penalty ≤ 1

Lasso penalty ≤ 1

Ridge and Lasso Equations Redux



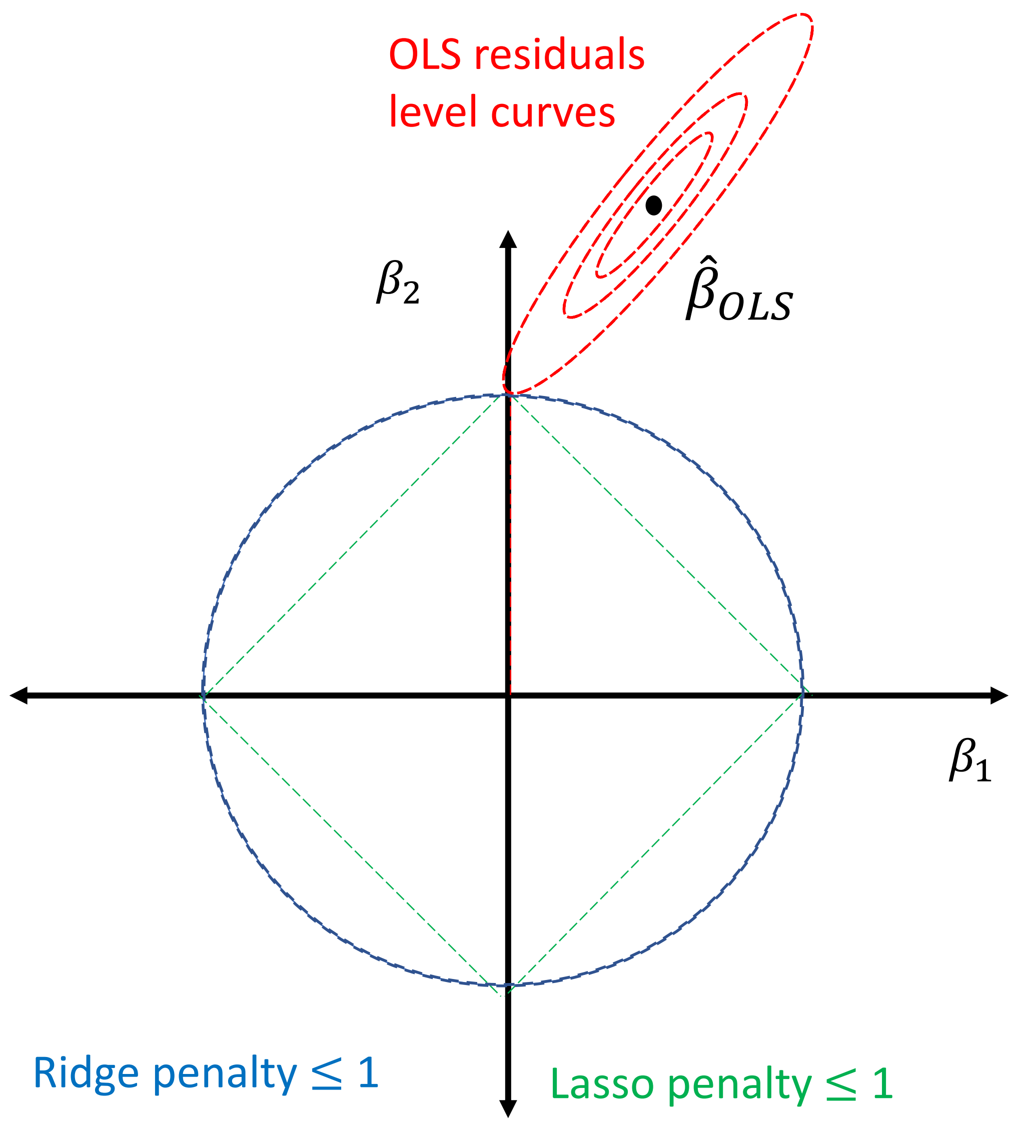
Suppose the optimal OLS beta is this point in black

Meaning, without constraints this point achieves a minimum of the residuals

We can represent that graphically as a series of contour lines where the black dot (OLS beta) is the minimum

Level curves farther from the OLS point are higher residuals

Ridge and Lasso Equations Redux



Graphically what the ridge equation is asking is: “find the lowest residual level curve while staying within the blue circle”

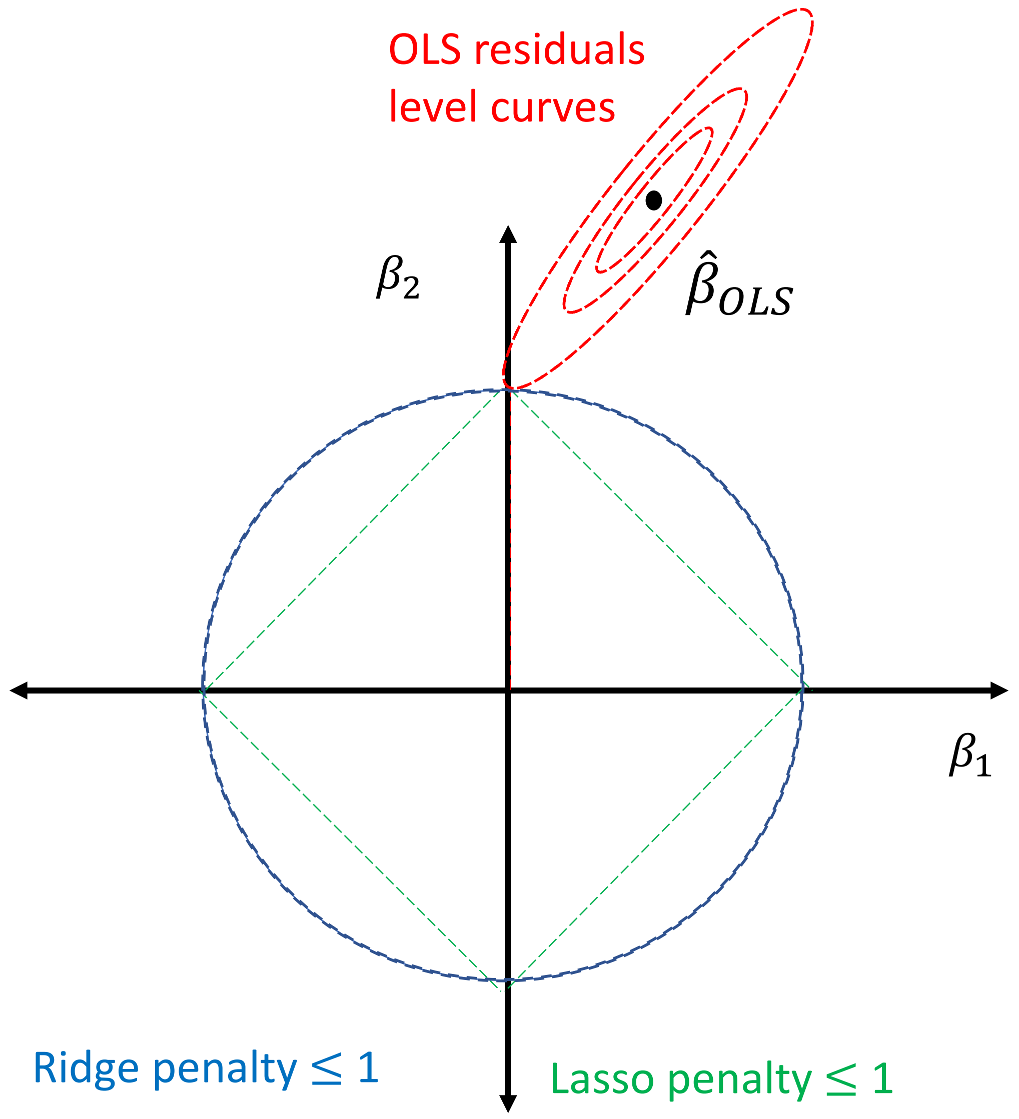
That is the level curve tangent to the blue line

Ridge

$$\min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2})^2$$

subject to $(\beta_1)^2 + (\beta_2)^2 \leq s$

Ridge and Lasso Equations Redux



Graphically what the Lasso equation is asking is: “find the lowest residual level curve while staying within the green diamond”

That is the level curve tangent to the green line

Lasso $$\min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2})^2$$

subject to $$|\beta_1| + |\beta_2| \leq s$$

Ridge and Lasso Equations Redux

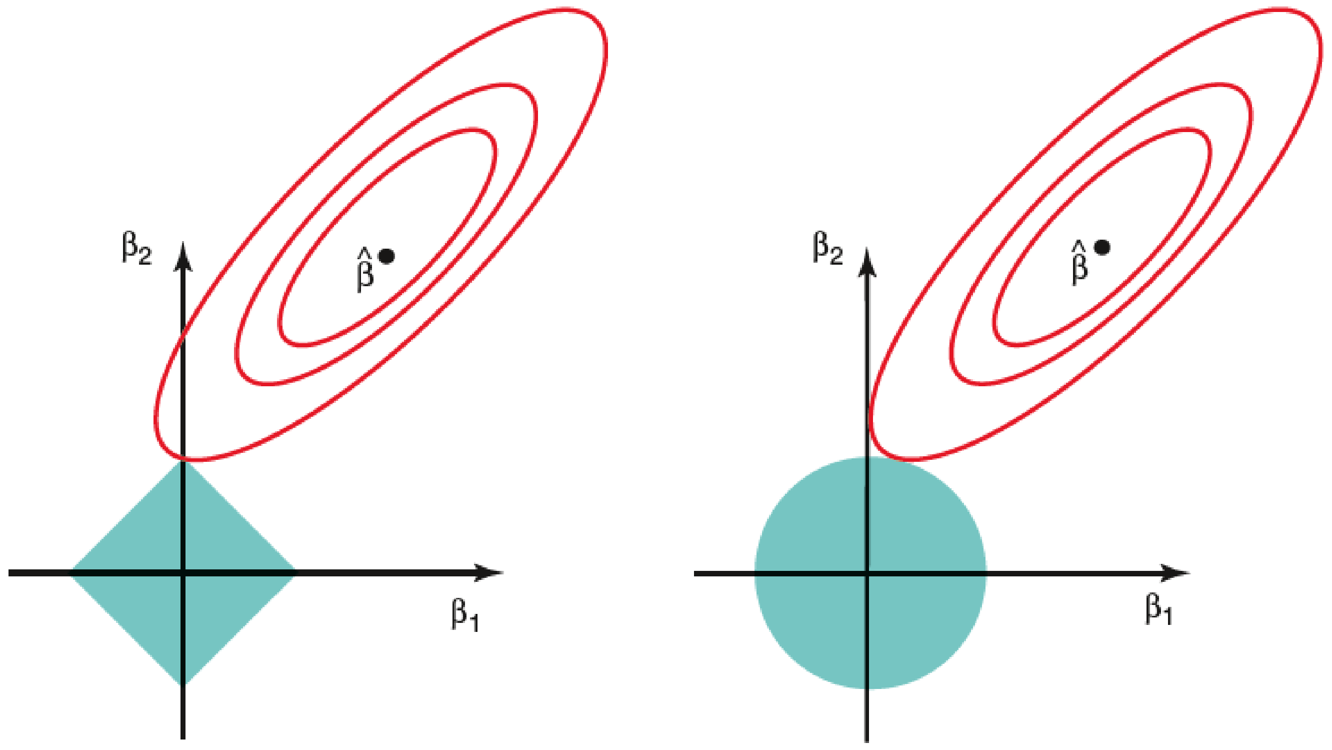


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.*

Lasso acts as a **variable selector** because the point of tangency for Lasso is often such that one of the variables (here beta_1) is zero

Ridge does not have this property, and we see there's still some small value for beta_1 in the right plot

Ridge versus Lasso

- Use Lasso when the "data generating process" (DGP, how the data is really formed) is **sparse**
- What is a sparse DGP?
    - Only a few variables really matter!
    - True model is parsimonious
- Ridge should be used when many variables matter a little

ElasticNet

Why Choose? ElasticNet Uses Both Ridge and Lasso Penalty

$$\beta_{ENet} = \min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2})^2$$

$$+ \lambda[\underbrace{\alpha(|\beta_1| + |\beta_2|)}_{\text{Lasso penalty}} + \underbrace{(1-\alpha)(\beta_1^2 + \beta_2^2)}_{\text{Ridge penalty}}]$$

- $\alpha \in [0,1]$ controls the amount of ridge versus lasso penalty
- λ functions as before -> controlling total amount of shrinkage penalty

How to choose λ and α? Grid Search

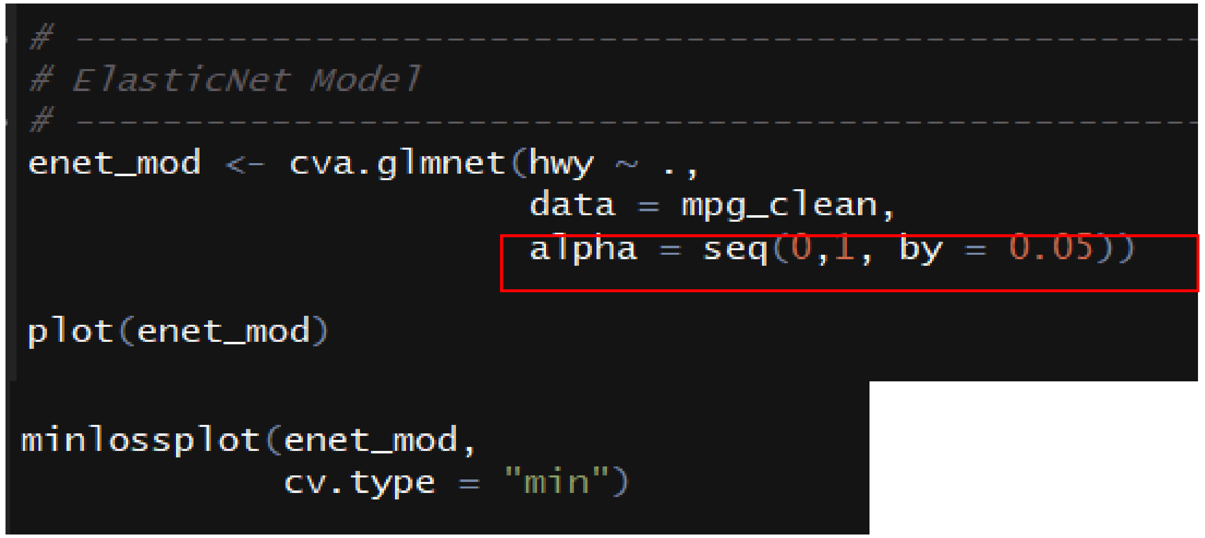
$$\beta_{ENet} = \min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_1 x_{i2})^2$$

$$+ \lambda[\alpha(|\beta_1| + |\beta_2|) + (1-\alpha)(\beta_1^2 + \beta_2^2)]$$

λ	$\boxed{\alpha = 0}$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	$\alpha = 1$
$\boxed{0.5}$	0.3	0.6	1.3	1.7	4.0
1.0	2.6	3.1	2.1	3.2	4.3
1.5	3.1	3.9	3.2	4.3	5.3
2	3.8	4.6	5.4	6.0	7.4

- We try out a number of different combinations of hyper-parameters
- For each hyper-parameter combination we calculate cross-validate MSE
- Optimal combination has lowest cross-validated MSE

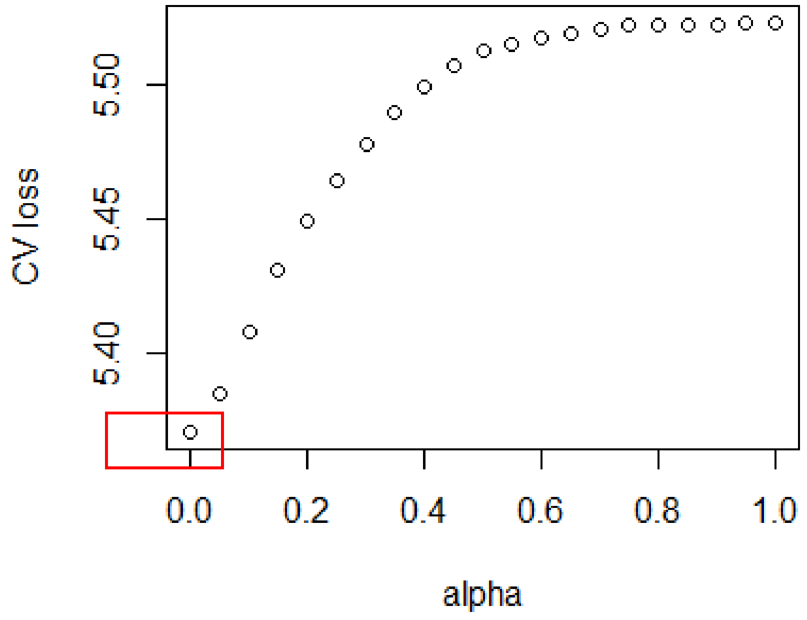
Estimating ElasticNet with cva.glmnet



```
# ------------------------------------------------
# ElasticNet Model
# ------------------------------------------------
enet_mod <- cva.glmnet(hwy ~ .,
                       data = mpg_clean,
                       alpha = seq(0,1, by = 0.05))

plot(enet_mod)

minlossplot(enet_mod,
            cv.type = "min")
```



- cva.glmnet will estimate a variety of elasticNet models varying alpha from 0 (all ridge) to 1 (all lasso)
- We must specify a sequence of alphas (between zero and 1) to estimate
- The function minlossplot() shows us how cross-validated MSE varies as we change alpha
- This plot reveals the minimum alpha value is at alpha = 0 or full lasso

Summary – Some Machine Learning Models

- **Parsimony means a model explains equally well using fewer variables**
- **Ridge and Lasso penalize magnitude and number of variables to obtain parsimony and avoid overfitting**
- Ridge uses ^2 penalization, Lasso uses absolute value penalization
- Cross-validation approximates out of sample fit
- λ key parameter and controls how much parsimony or shrinkage is used
    - Higher lambda -> more shrinkage
- Lasso acts as a variable selector
- Use lasso when you believe the true model is sparse/parsimonious/few variables matter
- Use ridge when you believe the true model is not sparse