Lecture 04 – Model Assessment and Validation

AMOD-5440

Kat Clark
Trent University

2026-10-02

Model Assessment and Validation

- So far, we've been fitting models, and aside from a brief discussion of the F statistic for overall model fit, we haven't really had a good way to discuss the performance of a model in any rigorous fashion.
- Today we will explore this topic, from the point of view of assessing a model's performance, and also for selecting the level of flexibility of a model.
- These explorations will involve two topics: the concept of **cross-validation**, and the concept of **the bootstrap**.

Validation

Validation

- We briefly discussed **test** and **train** data sets
  - Chapter 2 of ISLR2
- We used this so far, just fitting all of the data available to us in a regression framework
- Moving forward, we will not “just” be fitting the model
- if we want to determine the performance of a model, we have to lean on the test/train paradigm

We will start with an exploration of the idea of **validation**, then proceed to develop the full method of **cross-validation**.

Errors: Test and Training

- **Test error** is the average error that results from using a statistical learning method to *predict* the response on a new observation.
- Given a data set, the use of a particular statistical learning method is warranted if it results in a *low* test error.
- The test error can be easily calculated if a designated test set is available
  - this is usually not the case.
- **Training error** can be easily calculated by applying the statistical learning method to the observations used in its training.
- The training error rate often is *quite different* from the test error rate
  - in particular the former can dramatically underestimate the latter

The Validation Set Approach

- We consider a class of methods that estimate the test error rate by:
  - holding out a subset of the training observations from the fitting process
  - we do **not** estimate the model on these observations
  - then, apply the statistical learning method to those held out observations
- We do this by:
  - randomly dividing the available set of observations into two parts: the **training set** and the **validation set** (also called the **hold-out set**)
  - fit the model on the training set, and
  - validate the fitted model against the validation set (typically using the mean-squared error (MSE))

do not touch test set

An Example to Demonstrate

Consider the `Auto` data set from the `ISLR2` package.
It consists of a series of observations of automobiles, with variables `mpg`, `horsepower` and others.

If we want to model

$$y \sim \beta_0 + \beta_1 x$$

$$\text{mpg} \sim \beta_0 + \beta_1 \text{horsepower}$$

as a base model, we have 392 observations, and can do so.

> **Note**
>
> This example was discussed in Chapter 3 of ISLR2 (not in the lectures), and it was discovered in that case that there was a possible quadratic relationship between the variables: a model using horsepower and horsepower^2 did significantly better.

Examining the Model

```
1 library(ISLR2)
2 data(Auto)
3 mod1 <- lm(mpg ~ horsepower, data = Auto)
4 mod1 |> summary()
```

full data

```
Call:
lm(formula = mpg ~ horsepower, data = Auto)

Residuals:
     Min       1Q   Median       3Q      Max 
-13.5710  -3.2592  -0.3435   2.7630  16.9240 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 39.935861   0.717499   55.66   <2e-16 ***
horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.906 on 390 degrees of freedom
Multiple R-squared:  0.6059,    Adjusted R-squared:  0.6049 
F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16
```

Examining the Model

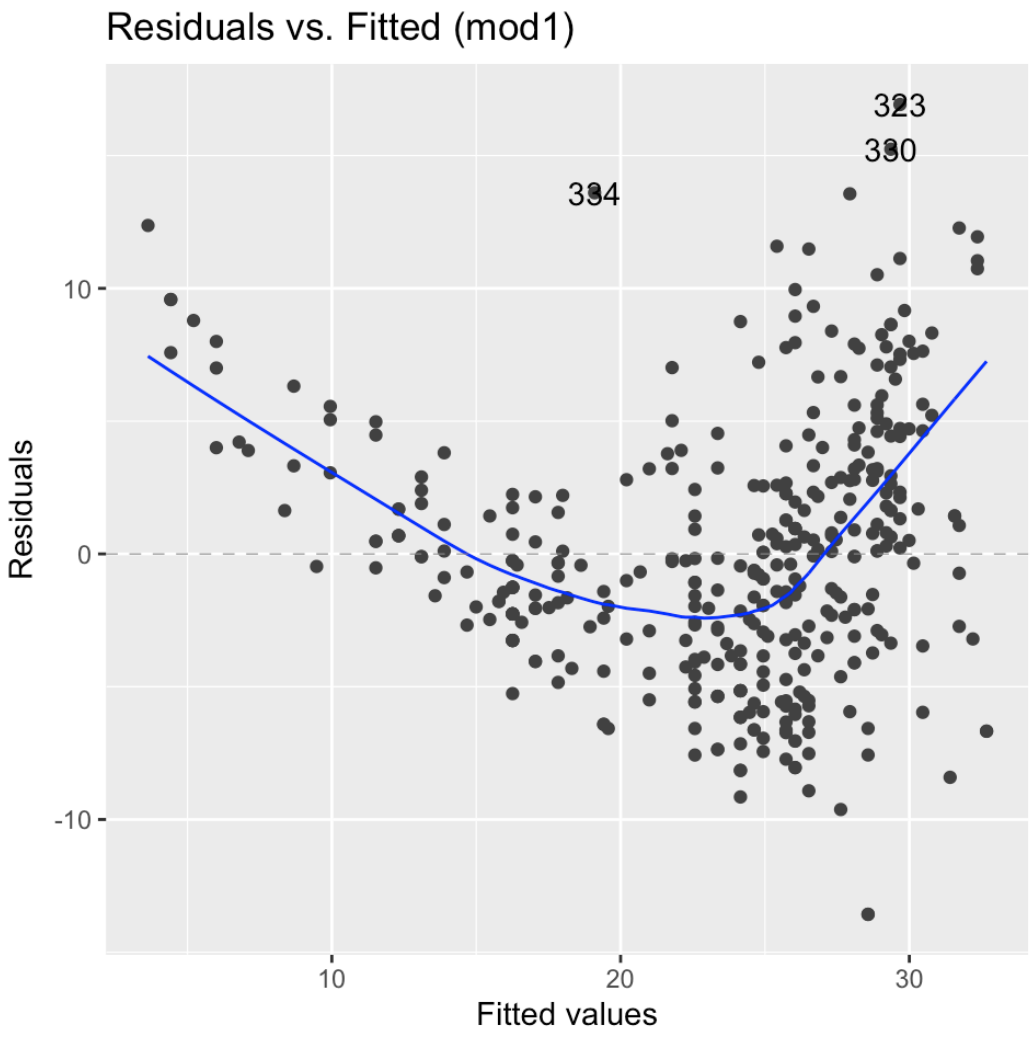
```
1 mod2 <- lm(mpg ~ horsepower + I(horsepower^2), data = Auto)
2 mod2 |> summary()
```

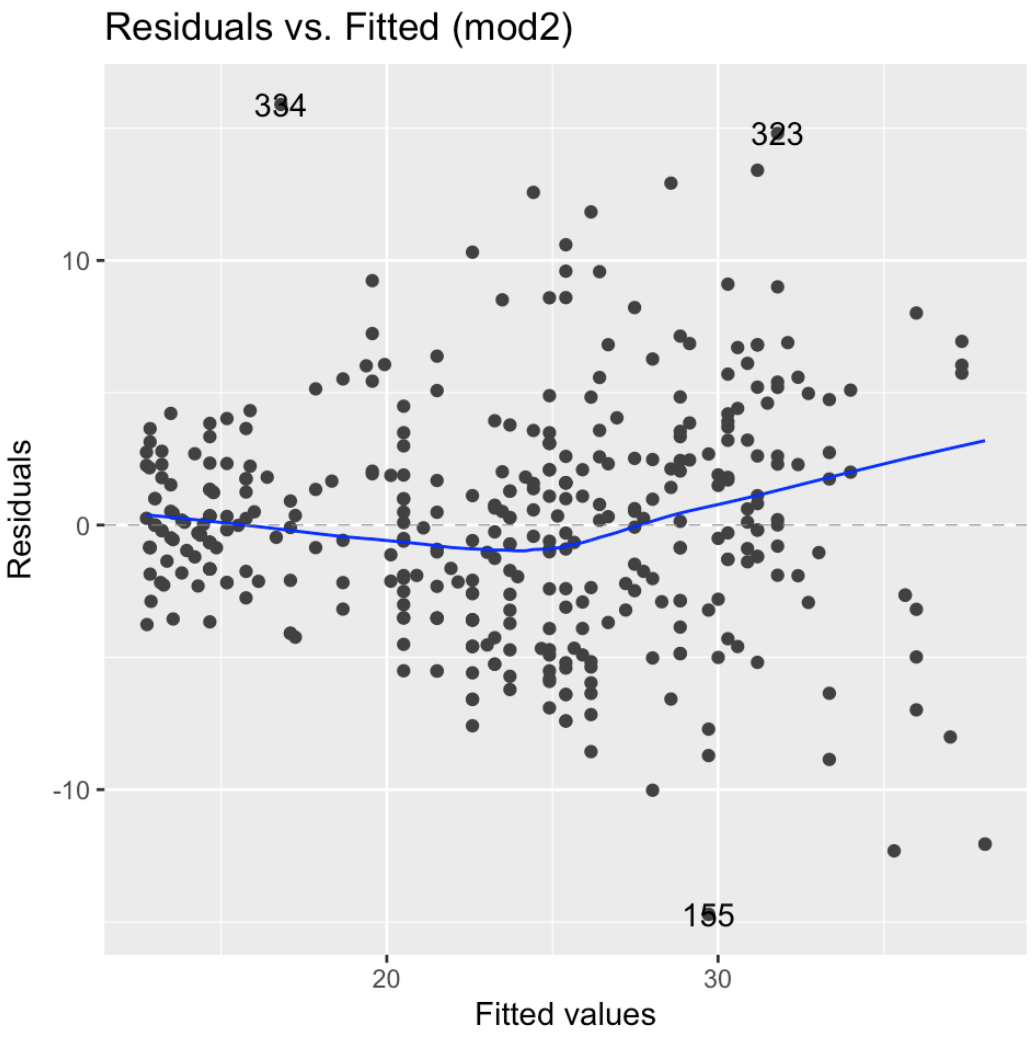
```
Call:
lm(formula = mpg ~ horsepower + I(horsepower^2), data = Auto)

Residuals:
     Min       1Q   Median       3Q      Max 
-14.7135  -2.5943  -0.0859   2.2868  15.8961 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)     56.9000997  1.8004268   31.60   <2e-16 ***
horsepower      -0.4661896  0.0311246  -14.98   <2e-16 ***
I(horsepower^2)  0.0012305  0.0001221   10.08   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.374 on 389 degrees of freedom
Multiple R-squared:  0.6876,    Adjusted R-squared:  0.686 
F-statistic:   428 on 2 and 389 DF,  p-value: < 2.2e-16
```

Examining the Model

- These models have 60.6 and 68.8 percent variation explained.
- The adjusted R-squared increases from 60.5 to 68.6
- So the higher order polynomial inclusion of horsepower does seem to be modelling something in a “better” way than only the linear term.

Diagnostic: Fitted vs. Residuals





Can We Do More?

- It is natural to wonder whether a cubic or higher-order fit might provide an even better fit.
- We could keep adding terms (e.g., a cubic) and looking at p-values, or we could use validation!
- Divide the 392 observations **randomly** into two sets:
  - a training set containing 196 data points, and
  - a validation set containing the remaining 196 observations

0.8
0.7

- Fit the model on the training set, and evaluate its performance (MSE)
- This lets us compare two variations of models while being somewhat more certain we are not overfitting.

↳ tailoring to much to the training set.

Demonstration for Degree of Polynomial

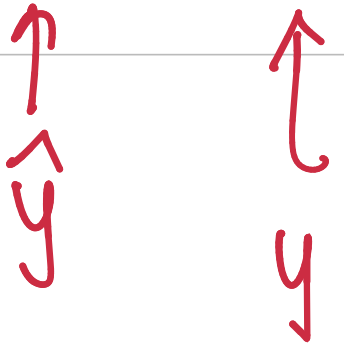
- Assume we want to know what degree polynomial will provide the “best” fit
- Fit polynomials from degree 1 to 10
- For each degree, fit the model on the training set, and evaluate the MSE on the testing (validation) set

```
1 set.seed(19)
2 split <- sample(x = 1:392, size = 196, replace = FALSE)
3 test <- Auto[split, ]
4 train <- Auto[-split, ]
5 nrow(test)
```

```
[1] 196
```

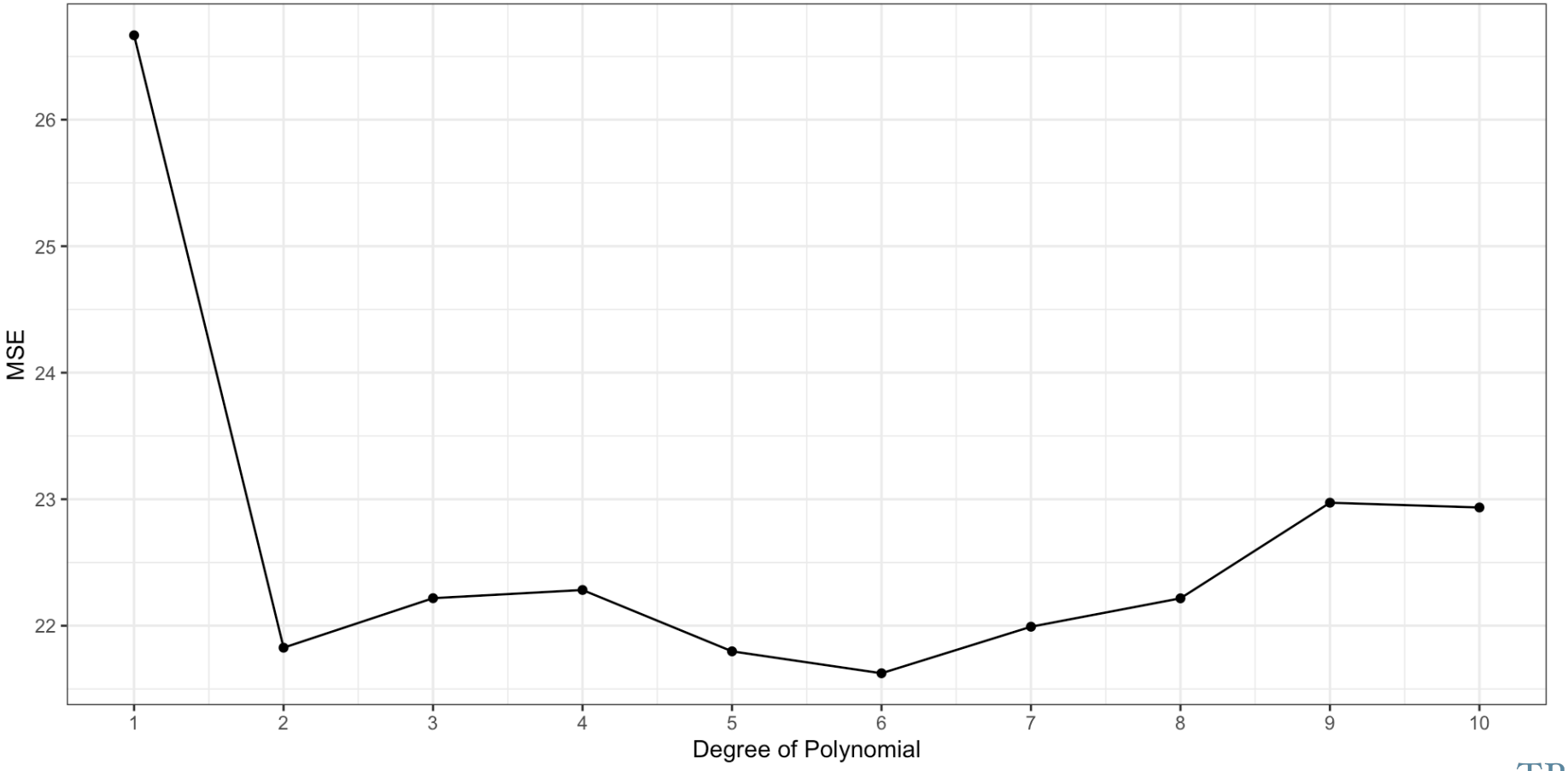
Demonstration for Degree of Poly

```
MSE <- vector(length = 10)
for(j in 1:10) {
  mod <- lm(formula = mpg ~ poly(horsepower, degree = j), data = train)
  pred <- predict.lm(mod, newdata = test, type = "response")
  MSE[j] <- mean((pred - test$mpg)^2)
}
```



Demonstration for Degree of Poly

```
library(scales)
ggplot(data = NULL, aes(x = 1:10, y = MSE)) + geom_line() +
  geom_point()+
  scale_x_continuous(breaks = breaks_pretty(n = 10)) +
  xlab("Degree of Polynomial") + theme_bw()
```

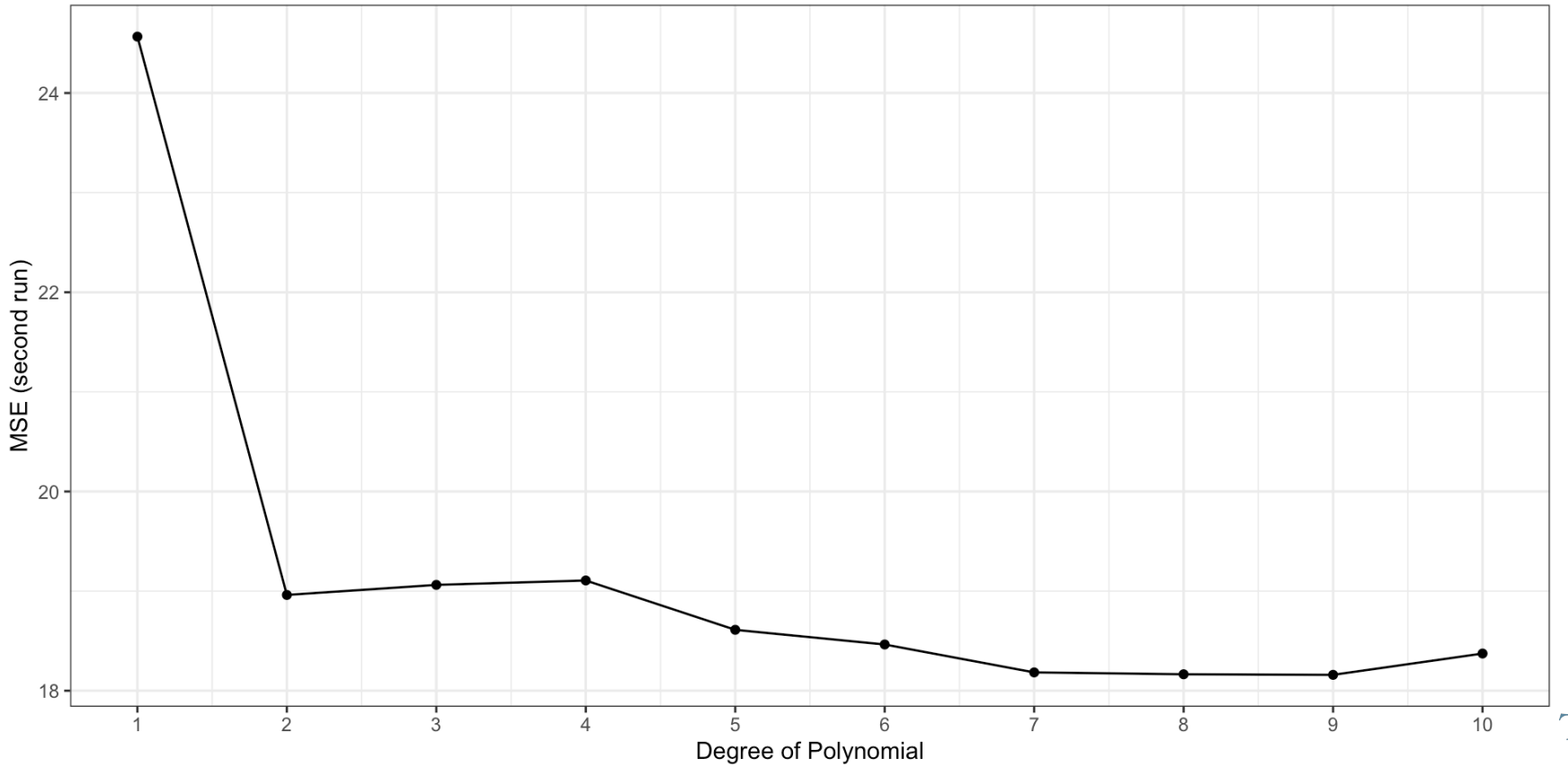


Demonstration for Degree of Poly

- Clearly the MSE improves tremendously between degree 1 (linear) and degree 2 (quadratic)!
- It looks as though the MSE goes down very slightly from degree 2 to 6, but not so much that we can justify the added complexity.
- All that we can conclude with any confidence is that the linear fit is not adequate for this data.

Drawbacks

1. The validation estimate of the test error rate can be highly variable
    - this will depend on which observations are included in the training set and testing sets
    - we only have **a single sample** of the test error rate!
    - we set a seed above, but otherwise, the results could be very different each time the code is run



Drawbacks

2. In the validation approach, only a subset of the observations - those that are included in the training set rather than in the validation set — are used to fit the model
    - we are cutting our training data in half, which is not ideal
    - smaller sample sizes tend to lead to worse models
    - the validation set error rate may tend to overestimate the test error rate compared to the model fit on the *entire data set*

Cross-Validation

LOOCV: Leave-One-Out Cross-Validation

- **LOOCV** involves splitting the set of validation observations into two parts (like the validation set approach)
- instead of creating two subsets of comparable size, a single observation is used for the validation set, and the remaining observations make up the training set
- the statistical learning method is fit on the $n - 1$ training observations, and a prediction $\hat{y}_1$ is made for the excluded observation, using its value x_1

Repeat this, leaving each element of the data set out, and obtaining n MSEs. Then, average these to get the LOOCV MSE for the model.

Benefits and Drawbacks

Advantages

1. Less bias - the MSE you measure should, on average, be a good representation of the true MSE.
2. Performing LOOCV multiple times will always give the same results
   - there is no randomness in the split between test/train
3. Maximizes the use of our data!

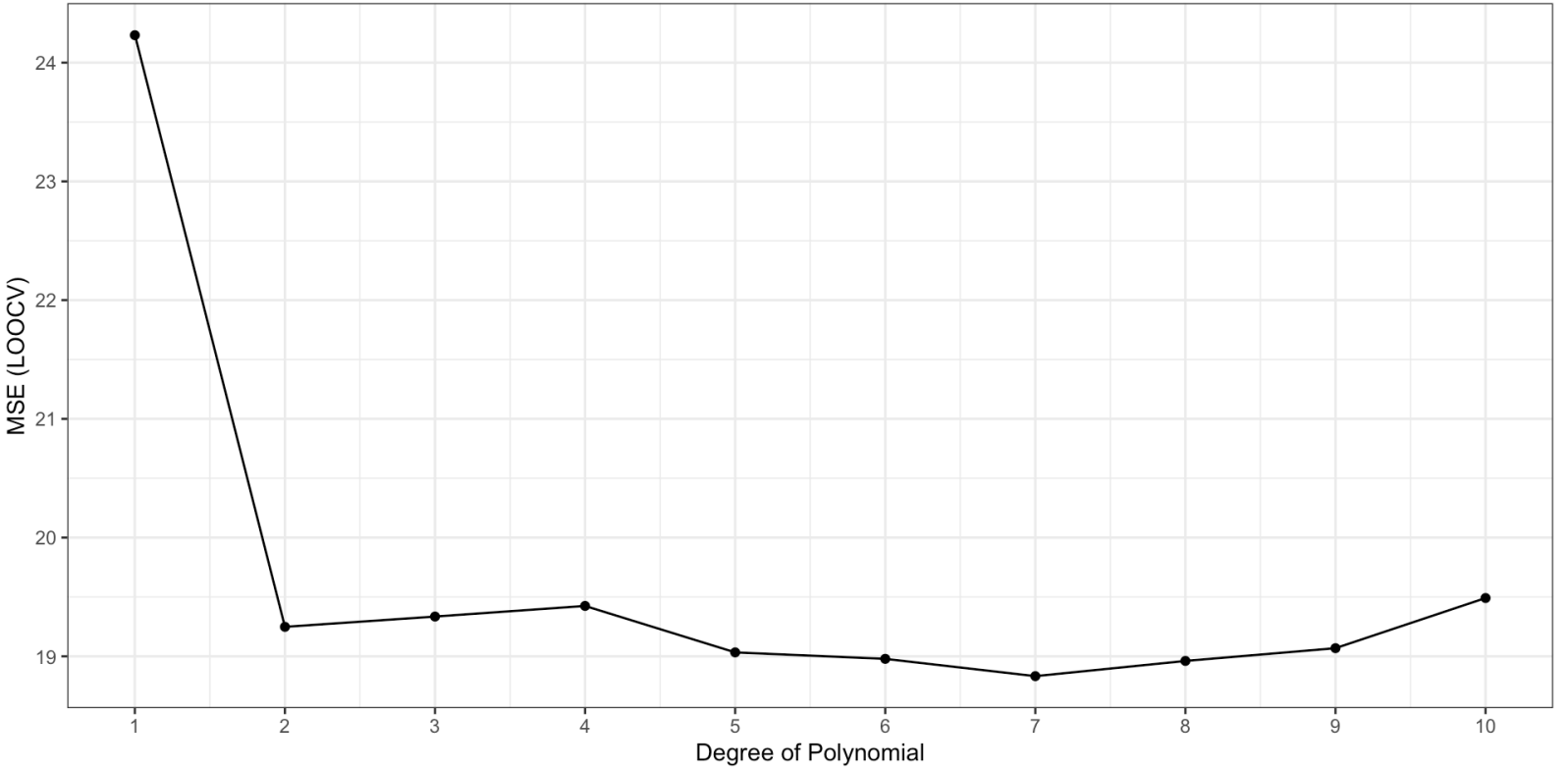
Disadvantages

1. Computationally expensive (have to fit the model n times!)
2. Higher variance estimates of the MSE
   - the MSEs from each leave-one-out is highly correlated with the others

Better Way to do CV and do LOOCV

```
1  library(boot)
2
3  cv_error <- vector(length = 10)
4  for(i in 1:10) {
5    glm_fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
6    cv_error[i] <- cv.glm(Auto, glm_fit )$delta[1]
7  }
```

K=



More on LOOCV

- This cross-validation function is part of the boot package.
- K represents the number of *folds* (more on that next)
- If you do not specify the argument K, it defaults to LOOCV: that is, to $K = n$, the number of observations.
- The use of LOOCV is very general, and can be used with any kind of predictive modelling, including logistic regression and Linear Discriminant Analysis already covered.
- It will also be very useful for methods discussed in coming weeks!

Alternative to LOOCV: k-Fold CV

- An alternative to Leave-One-Out CV is **k-fold CV**, which involves randomly dividing the set of observations into k groups (or **folds**), of approximately equal size.
- Then, taking one group at a time as the "left out validation set", the method is fit on the remaining $k-1$ folds, and the MSE computed on the left-out validation set.
- This is repeated a total of k times, and then the results averaged.
- Thus, LOOCV is a special case of k-fold CV, where $k = n$.

Advantage to k-Fold

- The most obvious advantage is computational.
- LOOCV requires fitting the statistical learning method n times.
- This has the potential to be computationally expensive.
- But cross-validation is a very general approach that can be applied to almost any statistical learning method.
- Some statistical learning methods have computationally intensive fitting procedures, and we can't afford to do the procedure n times.

All of this done on training set
↳ most of the time.

Cross-Validation for Classification Problems

- Cross-validation can also be a very useful approach in the classification setting when Y is qualitative.
- In this setting, cross-validation works just as described earlier in this chapter, except that rather than using MSE to quantify test error, we instead use the number of misclassified observations.
- Then, e.g., the LOOCV takes the form of the average of the Error Rate for each of the n validation runs.

The Bootstrap

Introduction

SE

- The **bootstrap** is a widely applicable and extremely powerful tool that lets us quantify uncertainty **without** knowing the distribution of things.
- That helps a lot in cases where the distributions aren't automatic, or where we don't trust that the distributions suggested/given are actually correct.

How Does it Work?

- You may have computed the **SE** of various statistics in previous statistics classes.
- You might even be comfortable enough with the concept to remember that regression returns a column of SEs, which are actually what are used to compute the p-values for each of the coefficients in a model.
- The bootstrap lets us use only the data, with no assumptions about the distribution of it, to estimate the variability (the SE) of a statistic of interest.
- It does this by sampling **with replacement** from the data to produce a **replicate** set; this set has the statistic applied to it.
- We can show that if we do this many, many times, the behaviour of the resulting set of replicate statistics acts a lot like the **sampling distribution**.

How Complicated Is This to Do?

- If you do it by hand? Incredibly.
- If you write your own code? Moderately.
- If you use the **boot** package? Trivial.

An Example of the Bootstrap

- Performing a bootstrap analysis in R entails only two steps:
- First, we must create a function that computes the statistic of interest.
- Second,we use the `boot()` function, which is part of the boot library, to perform the bootstrap by repeatedly sampling observations from the data set with replacement.

Example using Portfolio

```
1 library(ISLR2)
2 data(Portfolio)
```

- Suppose that we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y, respectively, where X and Y are random quantities.
- We will invest a fraction α of our money in X, and will invest the remaining $1 - \alpha$ in Y.
- Since there is variability associated with the returns on these two assets, we wish to choose α to minimize the total risk, or variance, of our investment.
- In other words, we want to minimize

$$\mathbf{Var}[\alpha X + (1 - \alpha)Y].$$

Example using Portfolios

The value that minimizes this is formed from a quotient of the variances of X and Y:

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

- We can estimate this using the data: estimate $\hat{\sigma}_Y$, $\hat{\sigma}_X$ and $\hat{\sigma}_{XY}$, and plug it in.
- But if we want an estimate of the variability … that's tough.

Using the Bootstrap

To use `boot()`, we need a defined function that does what we want to some data:

```
library(boot)
alpha_fn <- function(data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  (var(Y) - cov(X, Y)) / (var(X) + var(Y) - 2 * cov(X, Y))
}
alpha_fn(Portfolio, 1:100)
```

```
[1] 0.5758321
```

So this function computes the $\hat{\alpha}$ above, using whatever indices we specify.

Using boot()

```
1 boot(Portfolio, alpha_fn , R = 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Portfolio, statistic = alpha_fn, R = 1000)


Bootstrap Statistics :
     original     bias    std. error
t1* 0.5758321 0.00587001  0.09190174
```

So we have an estimated SE of 0.0919, and the estimate for $\hat{\alpha}$ is 0.5758. From this, we could write down a confidence interval, or even perform a hypothesis test, if we wanted to.

Connecting the Bootstrap to Regression

- The bootstrap approach can be used to assess the variability of the coefficient estimates and predictions from a statistical learning method.
- Let's apply it to the regression of horsepower on mpg from Auto, from earlier.
- We will compare the estimates obtained using the bootstrap to those obtained using the formulas for SE that are provided in the standard output.
- To do this, we need to create a simple function, `boot_fn()`, which takes in the Auto data and indices, then computes a regression model and returns the intercept and slope.

Example: Bootstrapping the Errors for Auto

```
boot_fn <- function(data, index) {
  coef(lm(mpg ~ horsepower, data = data, subset = index))
}
```

Our original results were:

```
summary(lm(mpg ~ horsepower, data = Auto))$coefficients[, 1:2]
```

```
              Estimate  Std. Error
(Intercept) 39.9358610 0.717498656
horsepower  -0.1578447 0.006445501
```

Then, apply boot to this, and estimate those SEs using the bootstrap instead!

Example: Bootstrapping the Errors for Auto

```
1 boot(Auto, boot_fn, 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Auto, statistic = boot_fn, R = 1000)


Bootstrap Statistics :
      original       bias    std. error
t1* 39.9358610 -0.033486392 0.846404847
t2* -0.1578447  0.000117534 0.007419546
```

The Power of the Bootstrap

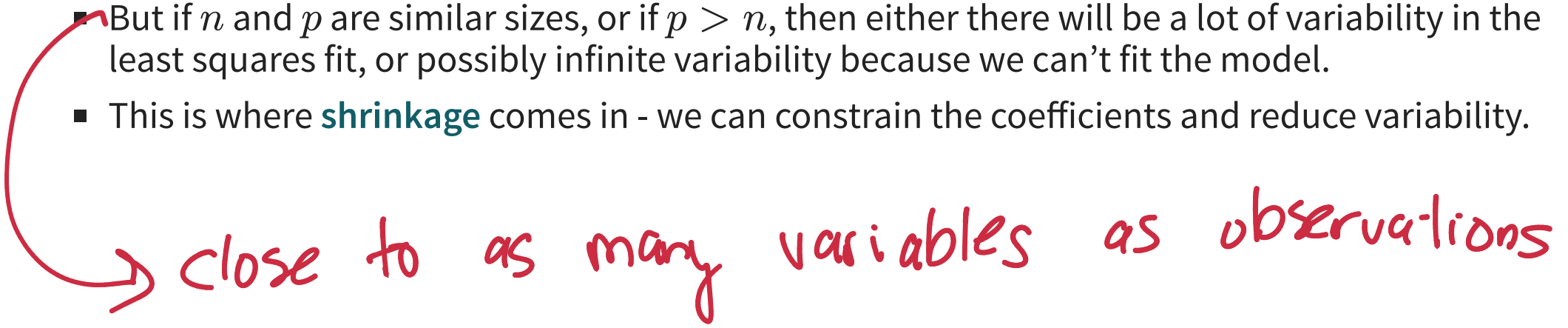
- The bootstrap seems a little gimmicky when you first see it.
- Why bother doing it, when there are parametric results just sitting there?
- But the advantage of the bootstrap is that it works on all kinds of cases where we **can't** do the parametric approach!
- Additionally, it's a useful way of checking assumptions.
- If the bootstrap and parametric approaches match up, it's a good sign the assumptions of the parametric approach were met.
- If they disagree, it's a warning sign.

Variable Selection and Regularization

Improvements to the Linear Model

Before we move on to some non-linear options for modelling in coming weeks, we should try to see what we can do to improve the linear model, aiming to yield better prediction accuracy and model interpretability.

- **Prediction Accuracy**:
  - Provided the true relationship between predictors and response is linear, least squares has low bias.
  - But if n and p are similar sizes, or if $p > n$, then either there will be a lot of variability in the least squares fit, or possibly infinite variability because we can't fit the model.
  - This is where **shrinkage** comes in - we can constrain the coefficients and reduce variability.



Improvements to the Linear Model

- **Model Interpretability**:
  - It is often the case that some/many of the variables used in a multiple regression model are not associated at all.
  - Including these leads to unnecessary complexity.
  - Removing them makes the model simpler, and easier to understand.
  - We already did this with the `step()` function.
  - We will now discuss some automatic methods for performing **variable selection**.

Subset Selection

The **best subset selection** procedure involves fitting least squares regressions for every possible combination of the p predictors. Fit all models with 1 predictor; fit all models with any 2 predictors; and so on. This results in 2^p possible models!

1. Let $\mathcal{M}_0$ be the *null model*, with no predictors.
2. For $k = 1, 2, \ldots, p$:
   - fit all possible models that have exactly k predictors
   - pick the best (where best is either highest R^2 or smallest RSS), label this model as $\mathcal{M}_k$
3. From the set $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$, choose the single "best" model using cross-validated prediction error, AIC, or adjusted R^2, depending on the kind of models.

Disadvantages

- This is super, super computational.
- And annoying to do.
- If $p = 10$ (10 predictors), there are ~ 1000 models; if $p = 20$, there are over $1,000,000$!

Subset Selection Example

- We will use the **Hitters** database from the ISLR2 package, which is a Major League Baseball data set from 1986 with a variety of relevant variables.
- We will try to predict the Salary of the players in the data set using the variables available.

```
library(ISLR2)
data(Hitters)
```

```
library(leaps)
regfit<-regsubsets(Salary ~ . ,data=Hitters, nvmax = 19,
                   method="exhaustive")
reg.summary <- regfit |> summary()
```

Subset Selection Example

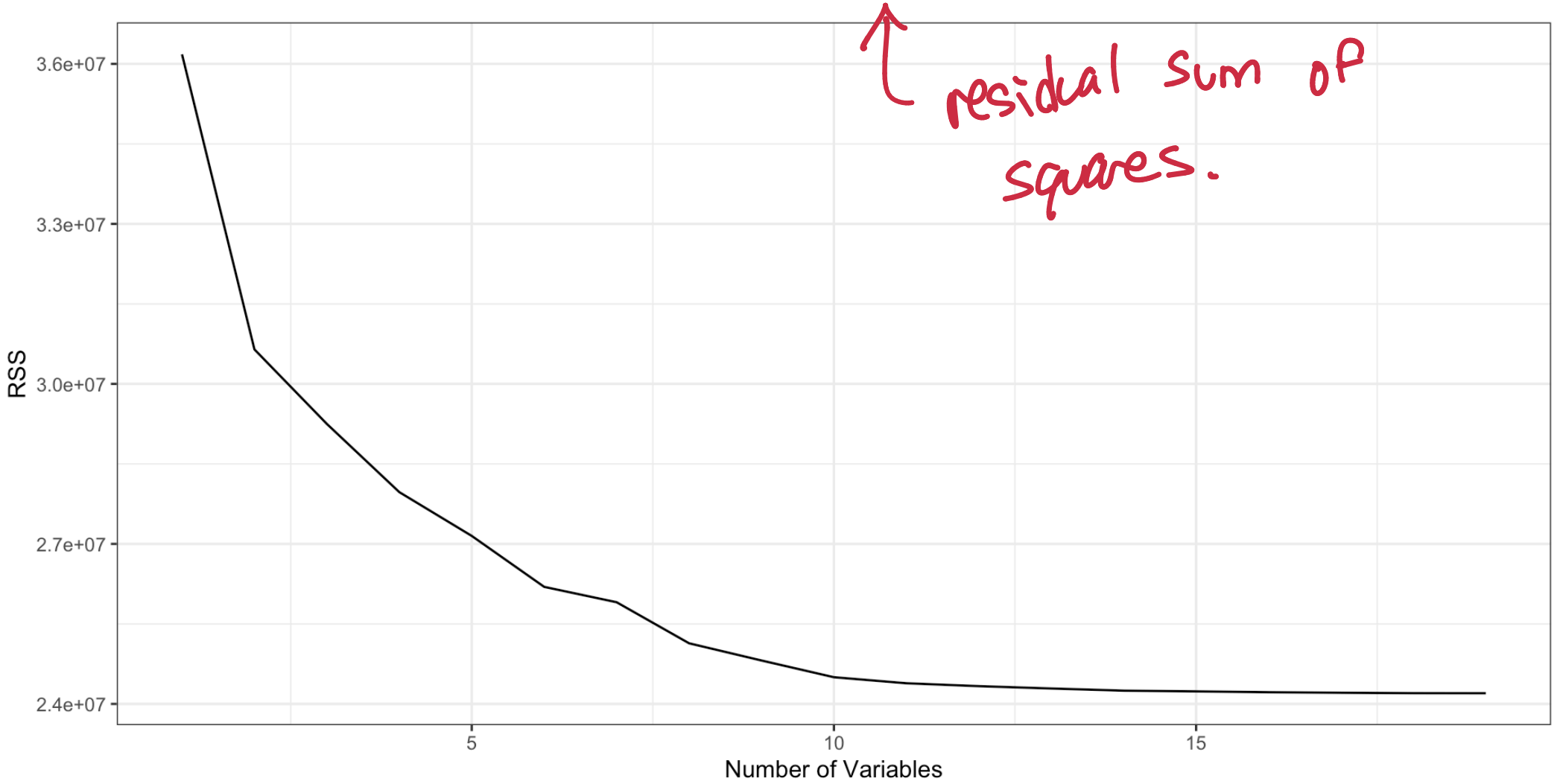
```
1 reg.summary$outmat
```

```
          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
1  ( 1 )  " "   " "  " "   " "  " " " "   " "  " "   " "  " "   " "   "*"
2  ( 1 )  " "   "*"  " "   " "  " " " "   " "  " "   " "  " "   " "   "*"
3  ( 1 )  " "   "*"  " "   " "  " " " "   " "  " "   " "  " "   " "   "*"
4  ( 1 )  " "   "*"  " "   " "  " " " "   " "  " "   " "  " "   " "   "*"
5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "  " "   " "  " "   " "   "*"
6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "  " "   " "  " "   " "   "*"
7  ( 1 )  " "   "*"  " "   " "  " " "*"   " "  "*"   "*"  "*"   " "   " "
8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "  " "   " "  "*"   "*"   " "
9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "  "*"   " "  " "   "*"   "*"
10 ( 1 )  "*"   "*"  " "   " "  " " "*"   " "  "*"   " "  " "   "*"   "*"
11 ( 1 )  "*"   "*"  " "   " "  " " "*"   " "  "*"   " "  " "   "*"   "*"
12 ( 1 )  "*"   "*"  " "   "*"  " " "*"   " "  "*"   " "  " "   "*"   "*"
13 ( 1 )  "*"   "*"  " "   "*"  " " "*"   " "  "*"   " "  " "   "*"   "*"
14 ( 1 )  "*"   "*"  "*"   "*"  " " "*"   " "  "*"   " "  " "   "*"   "*"
15 ( 1 )  "*"   "*"  "*"   "*"  " " "*"   " "  "*"   "*"  " "   "*"   "*"
16 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   " "  "*"   "*"  " "   "*"   "*"
17 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   " "  "*"   "*"  " "   "*"   "*"
18 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   "*"  "*"   "*"  " "   "*"   "*"
19 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   "*"  "*"   "*"  "*"   "*"   "*"
          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
```

```
1  ( 1 )  " "    " "    " "    " "    " "    " "    " "
2  ( 1 )  " "    " "    " "    " "    " "    " "    " "
3  ( 1 )  " "    " "    " "    "*"    " "    " "    " "
4  ( 1 )  " "    " "    "*"    "*"    " "    " "    " "
5  ( 1 )  " "    " "    "*"    "*"    " "    " "    " "
6  ( 1 )  " "    " "    "*"    "*"    " "    " "    " "
7  ( 1 )  " "    " "    "*"    "*"    " "    " "    " "
8  ( 1 )  "*"    " "    "*"    "*"    " "    " "    " "
9  ( 1 )  "*"    " "    "*"    "*"    " "    " "    " "
10  ( 1 ) "*"    " "    "*"    "*"    "*"    " "    " "
11  ( 1 ) "*"    "*"    "*"    "*"    "*"    " "    " "
12  ( 1 ) "*"    "*"    "*"    "*"    "*"    " "    " "
13  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "
14  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "
15  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "
16  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "
17  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"
18  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"
19  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"
```

Subset Selection Example

```
1 ggplot(data = NULL,aes(x = 1: 19, y = reg.summary$rss)) + geom_line() +
2   labs(x = "Number of Variables",y = "RSS") + theme_bw()
```



Stepwise Selection

- This is the method we already talked about: backwards selection is quite computationally efficient, and easy to think about.
- Just iterate until you're happy with the overall model fit.

Choosing Optimal Models

F-statistic

- We've talked about using RSS, adjusted R^2, and overall model fit to select a model.
- But there are more things we can use as metrics for comparing one model to another in order to determine which to select.
- But when we start thinking about test data and training data, even if a model has the lowest RSS on training data does not make it the best or most useful model.
- Additionally, the model with the most predictors will always have the lowest RSS, if the predictors are a superset of another model's.
- So, in order to select the best model with respect to test error, we need to estimate the testing error.
- We can either **adjust** the training error to try to compensate for the issues (e.g., as we do with "Adjusted R^2"), or we can use a validation or cross-validation approach.

Alternatives to adjusted R^2

- Mallow's C_p
- Akaike information criterion (AIC)
- Bayesian information criterion (BIC)

All three of these equations use the Residual Sum of Squares (RSS), but modify this by adding in a penalty which compensates for the test/train issue.

How to Interpret

- C_p: takes on small values for models with low test error. Choose the model with the lowest C_p.
- AIC: low values for low test errors, so lower is better
- BIC: low values for low test errors, so lower is better

We will see examples of this in the lab. It's not terribly complicated.

Shrinkage Methods

regularization

Ridge Regression

- We can fit a model containing all p predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.
- The first such method we will consider is called **ridge regression**.

Ridge Regression

- Instead of fitting our model using least squares, which just minimizes the sum of the squares errors (hence the name), **ridge regression** uses a **penalized** version of the same thing, that adds a second term to the minimization.
- We want to minimize the RSS, and also, at the same time, minimize the sum of the squares of the coefficients of the model.
- This forces the minimization to push the β_j coefficients down toward 0.
- This idea is called **shrinkage**, and the extra term in the minimization is called a **penalty**.
- Unlike least squares, which produces only one result, ridge regression has a parameter (λ), so ridge regression will produce a different set of coefficients for each choice of λ.
- Selecting this coefficient is important!

Ridge Regression: Formula

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

RSS - residual sum of squares

- The first term is just the RSS that is optimized over for least squares (regression).
- The second is the new bit, the penalty.

Why does Constraining Help?

- Ridge regression's advantage over least squares is rooted in the bias-variance trade-off.
- As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias.
- And when $\lambda = 0$, we just get least squares.
- The increasing λ can lead to quite large decreases in variance, though, at the cost of only minor increases in bias, so it is often worthwhile.
- Additionally, ridge regression has substantial computational advantages: it is much, much faster than best subset selection. It's barely slower than least squares, which is very fast.
- We will do an example in a few minutes, after discussing the last method of the lecture: the Lasso.

The Lasso

- Ridge regression does have one obvious disadvantage.
- Unlike best subset and backward stepwise selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model.
- The penalty term may shrink the coefficients toward 0, but it will not set them **exactly** equal to 0.

The Lasso

- A lasso is a rope tool used by cowhands to “reach out” and “select” a specific animal from a herd, made iconic by Western movies in the 1940s and 1950s.
- In the context of modelling, we use the lasso to “reach out” and “select” relevant variables from a candidate set of variables.

The Lasso: Formula

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

- In math terms, this is known as a l_1 penalty instead of a l_2 penalty: we have absolute values instead of squares.
- The effect is to shrink the coefficients toward zero, but in the case of the lasso, and the l_1 penalty, we actually force some of the coefficients to **be** zero.
- This effectively performs variable selection.

Comparing Ridge to Lasso

- Neither ridge regression nor the lasso will universally dominate the other.
- In general, one might expect the lasso to perform better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero.
- Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.
- However, the number of predictors that is related to the response is never known *a priori* for real data sets.
- A technique such as cross-validation can be used in order to determine which approach is better on a particular data set.

Tuning Parameter

- Both ridge regression and the lasso have a tuning parameter, λ, included in their formulation.
- Both require some method of **choosing** λ.
- It turns out that Cross-Validation is an ideal tool for helping us do this.
- Start by choosing a grid of λ values, and compute the CV error for each of the λ values.
- Then, select the λ for which the CV error is smallest.
- Then, re-fit the model, using all available observations, and use the selected value of the parameter.

Example of Ridge and Lasso on a Real Data Set

We need a new package to do this, called **glmnet**.

```
1 install.packages("glmnet")
2 library(glmnet)
```

- We will revisit the **Hitters** database from the ISLR2 package.

Hitters Data Set

To do the work of cross-validation to find λ, we have to set some things up a bit differently than normal:

```
data(Hitters)
Hitters <- Hitters[complete.cases(Hitters), ]  # some missing data
x <- model.matrix(Salary ~ ., data = Hitters)[, -1]
y <- Hitters$Salary
grid <- 10^seq(from = 10, to = -2, length.out = 101)
mod_ridge <- glmnet(x, y, alpha = 0, lambda = grid)
```

- Here we load the data, delete the observations that are not complete, and manually form the x/y we need for the finding of λ.
- Then we specify a grid of possible λ values running from 10^{10} down to 10^{-2}, with fairly granular steps.
- The function **glmnet()** takes an alpha argument - 0 is ridge regression.

Results

The results of running glmnet() are complicated, but one of the things contained in them are a matrix of coefficients for the different choices of λ.

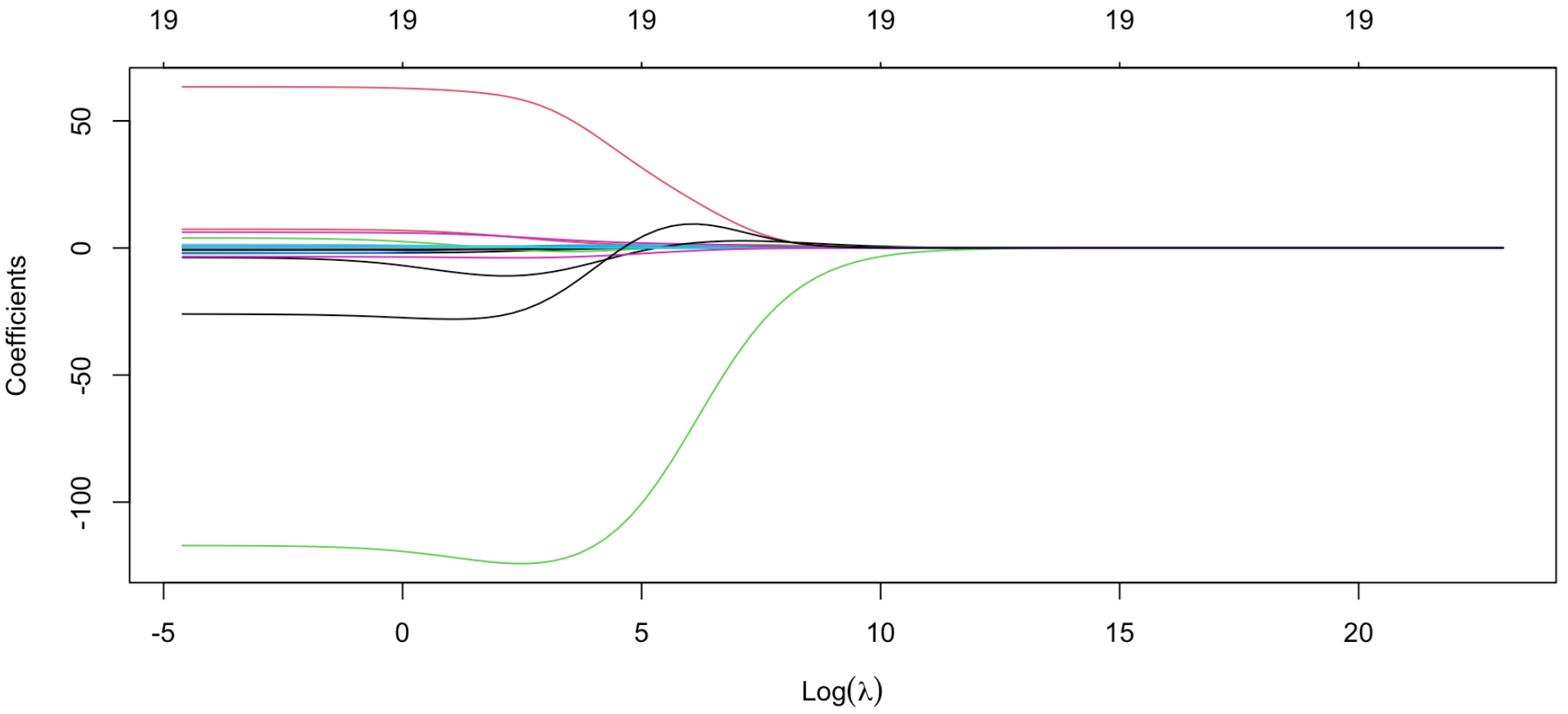
```
1 mod_ridge$lambda[c(1,50,101)]
2 coef(mod_ridge)[, c(1,50,101)]
```

```
[1] 1.000000e+10 1.318257e+04 1.000000e-02
```

```
20 x 3 sparse Matrix of class "dgCMatrix"
                      s0           s49          s100
(Intercept)  5.359257e+02 420.607020391  164.11225278
AtBat        5.443467e-08   0.033171848   -1.97389578
Hits         1.974589e-07   0.123551735    7.37814683
HmRun        7.956523e-07   0.472570943    3.93812459
Runs         3.339178e-07   0.206608010   -2.19943830
RBI          3.527222e-07   0.215205646   -0.91671104
Walks        4.151323e-07   0.259045076    6.20057597
Years        1.697711e-06   0.998795705   -3.71130288
CAtBat       4.673743e-09   0.002814475   -0.17513909
CHits        1.720071e-08   0.010458222    0.21124213
CHmRun       1.297171e-07   0.078603947    0.05574747
CRuns        3.450846e-08   0.020981724    1.36636951
CRBI         3.561348e-08   0.021661167    0.70990662
```

A Plot

```
1 plot(mod_ridge,sign.lambda = 1)
```



Test/Train Paradigm

- We now split the samples into a training set and a test set in order to estimate the test error of ridge regression (and the lasso).
- There are two common ways to randomly split a data set.
- The first is to produce a random vector of TRUE, FALSE elements and select the observations corresponding to TRUE for the training data.
- The second is to randomly choose a subset of numbers between 1 and n; these can then be used as the indices for the training observations.

```
set.seed(1)
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
```

Using CV

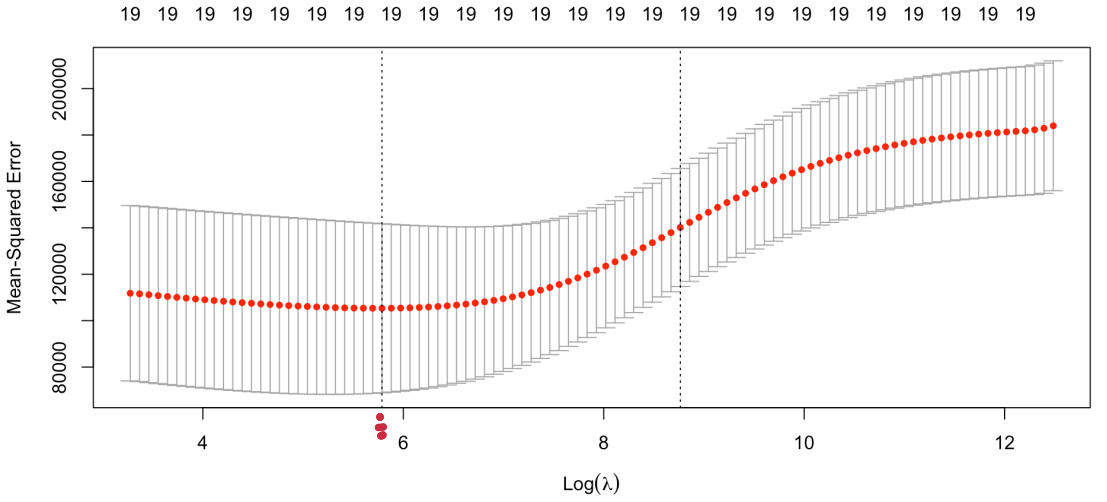
We can use cross-validation to choose the tuning parameter.

```
set.seed(1)
cv_out <- cv.glmnet(x[train, ],
                    y[train],
                    alpha = 0)
plot(cv_out,sign.lambda = 1)
```

ridge

```
lambda <- cv_out$lambda.min
lambda
```

```
[1] 326.0828
```



Comparing MSEs

Using the best $\lambda = 326.1$, what's our MSE?

```
mod_ridge2 <- glmnet(x[train, ], y[train], alpha = 0,
                     lambda = grid, thresh = 1e-12)
best_pred <- predict(mod_ridge2, s = lambda, newx = x[test, ])
mean((best_pred - y[test])^2)
```

```
[1] 139855.5
```

Compare this to an arbitrary λ, say $\lambda = 4$:

```
other_pred <- predict(mod_ridge2, s = 4, newx = x[test, ])
mean((other_pred - y[test])^2)
```

```
[1] 142197.3
```

Final Ridge

Finally, we refit the model with all data (test and train) and examine the coefficient estimates for the value of λ chosen by cross-validation.

```
1 out <- glmnet(x, y, alpha = 0, lambda = grid)
2 predict(out,s=lambda,type="coefficients")[,1]
```

```
(Intercept)         AtBat          Hits         HmRun          Runs           RBI
15.65475615   0.07597376   0.86426334   0.59358826   1.06370867   0.87791607
      Walks         Years        CAtBat         CHits        CHmRun         CRuns
 1.62660328   1.33036623   0.01128510   0.05737513   0.40605889   0.11485706
       CRBI        CWalks       LeagueN    DivisionW      PutOuts      Assists
 0.12191069   0.05269470 22.20530854 -79.23643685   0.16660030   0.02973165
     Errors   NewLeagueN
-1.37336501   9.03463145
```

None of the coefficients are 0, as expected, but many are quite small.

Summary of the CV Procedure

1. **Split the data into a training set and a test set.** The test set is set aside and will not be used during model building.
2. **Use only the training set to select the model**. Cross-validation is performed within the training set, repeatedly splitting it into training and validation folds. The test set remains untouched.
3. **Choose the optimal tuning parameter** based on the cross-validation results and refit the model using the **entire training set** with that parameter. The test set has still not been used.
4. **Evaluate predictive performance** by applying the fitted model to the test set and calculating the prediction error. This provides an estimate of out-of-sample prediction accuracy.
5. **Interpret the model** by refitting it using **all available data** and examining the resulting coefficients. This model is used for interpretation, not for assessing predictive performance.

What about lm()?

We can also fit the Hitters data using lm():

```
mod_lm <- lm(Salary ~ ., data = Hitters, subset = train)
pred_y<-predict(mod_lm, type = "response", newdata = Hitters[test, ])
mean((pred_y- y[test]) ^2)
```

```
[1] 168593.3
```

Just because we use a “fancy” method doesn’t mean we necessarily get a better test-set MSE, but in this case, the basic multiple regression with absolutely no variable selection is quite a bit worse than most of the ridge.

lm(), with variable selection

If instead we did some variable selection and used `step()`, we'd get some improvement, but still nowhere near the performance of the ridge.

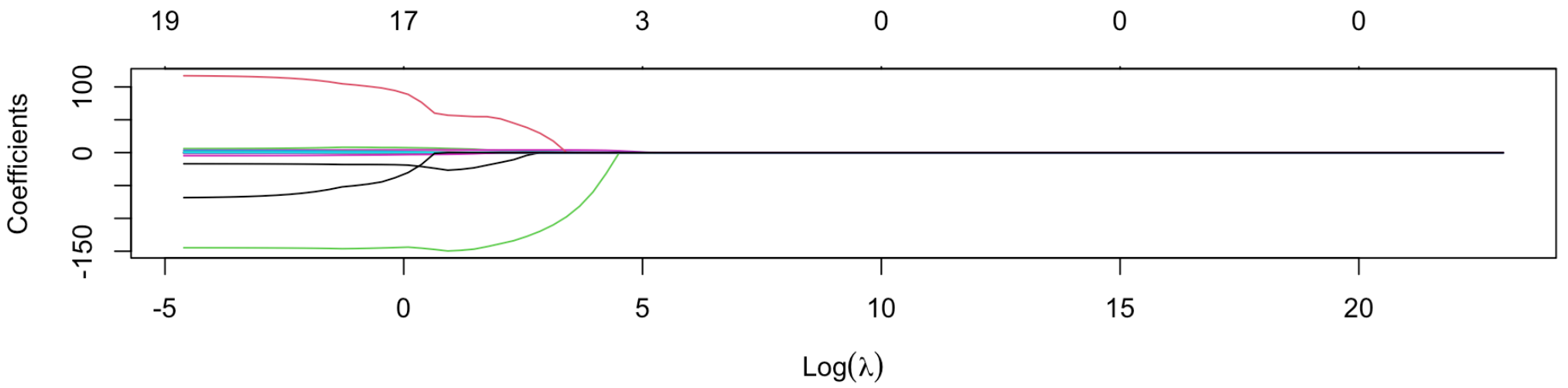
```
smallest_mod <- lm(Salary ~ 1, data = Hitters, subset = train)
mod_red <- step(mod_lm, scope = list(lower = smallest_mod,
                                     upper = mod_lm),
                direction = "backward", trace = 0)

pred_y<-predict(mod_red, type = "response", newdata = Hitters[test, ])
mean((pred_y- y[test]) ^2)
```

```
[1] 156090.9
```

Lasso, Instead

We now ask whether the lasso can yield either a more accurate or a more interpretable model than ridge regression.

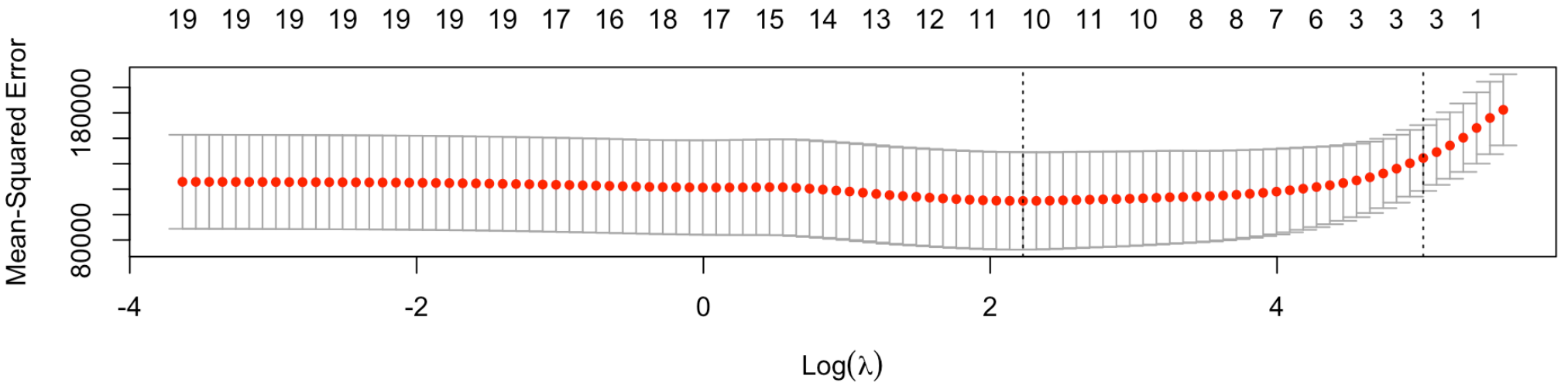
```
1 mod_lasso <- glmnet(x[train, ], y[train], alpha = 1,lambda = grid)
2 plot(mod_lasso,sign.lambda=1)
```



Lasso, ctd.

From the above plot, we see that depending on how the tuning happens, some of the variables (the coloured lines) can be exactly equal to zero. Let's do CV and find the test error:

```
set.seed(1)
cv_out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv_out,sign.lambda = 1)
```



Lasso, ctd.

```
lambda_lasso <- cv_out$lambda.min
lambda_lasso
```

```
[1] 9.286955
```

```
pred_lasso <- predict(mod_lasso, s = lambda_lasso,
                      newx = x[test, ])
mean((pred_lasso - y[test])^2)
```

```
[1] 143674.1
```

In this specific case, lasso performs worse than ridge ridge regression (MSE of ~138k).

Checking coefficients for lasso

```
out <- glmnet(x, y, alpha = 1, lambda = grid)
predict(out, type = "coefficients",
        s = lambda_lasso)[,1 ]
```

```
  (Intercept)         AtBat          Hits         HmRun          Runs 
  2.75817275  -0.06419778   2.20296652   0.00000000   0.00000000 
          RBI         Walks         Years        CAtBat         CHits 
  0.00000000   2.29692195  -0.53267352   0.00000000   0.00000000 
       CHmRun         CRuns          CRBI        CWalks       LeagueN 
  0.02296020   0.21730302   0.42003184   0.00000000  20.16304171 
    DivisionW       PutOuts       Assists        Errors    NewLeagueN 
-116.12487580   0.23750321   0.00000000  -0.84475264   0.00000000 
```

- However, lasso has taken care of zeroing out some of the variables for us, without having to do the backwards selection procedure.
- It also performs better than `lm()` with variable selection.
- This is why using lasso is preferred over doing stepwise variable selection, if at all possible!

Note: PCR and PLS

- There are two very powerful techniques for dimension reduction, called Principal Components Regression (PCR), connected to Principal Components Analysis (PCA), which is very popular in applied sciences; and
- Partial Least Squares, which is less so.
- We did not have time to go over them, but if you're curious, Chapters 6.3 and the examples in 6.5.3 are worth exploring to expose yourself to the ideas.

Summary

- Resampling methods and regularization are extremely powerful techniques, and would still be considered quite modern in terms of statistical theory.
- Many of the key pieces of theory for these areas are still being explored as active areas of research by statistical researchers.

Next Topic

Our next topic relaxes the assumption of linearity, giving us a significantly more flexible modelling structure, at the cost of interpretability. We're starting into the great unknown now!

Readings

The material of this week is Chapters 5.1, 5.2 and 6.1-6.2 of ISLR2, discussing model validation and variable selection.

Next week, we will continue into Chapter 7, moving beyond linearity and discussing the use of polynomials, splines, and Generalized Additive Models. Readings are taken from Chapters 7.1-7.5 and 7.7 of ISLR2.