Day 7: Resampling Methods and Model Selection

ME314: Introduction to Data Science and Machine Learning

Jack Blumenau

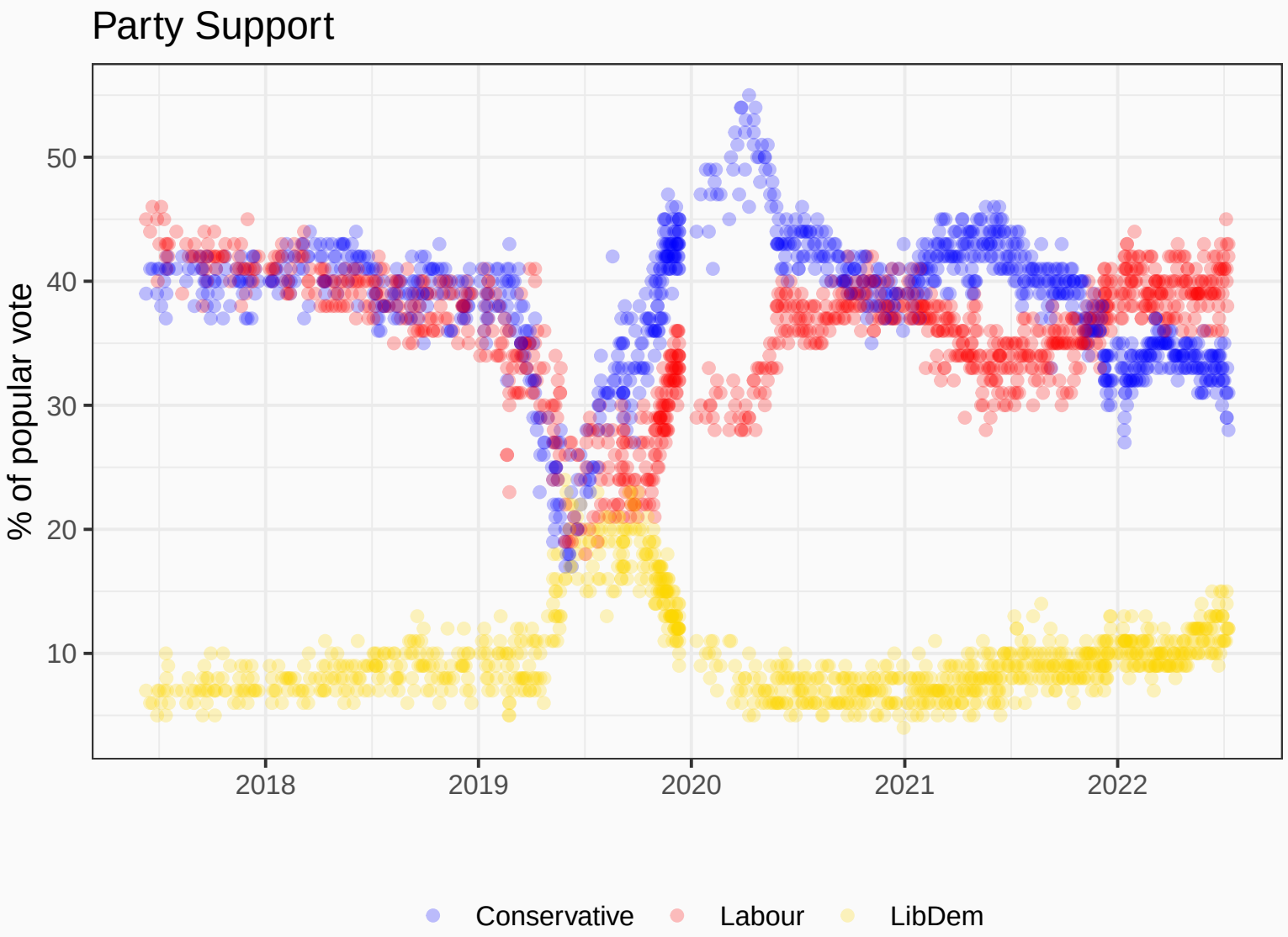
19th July 2023

Motivation

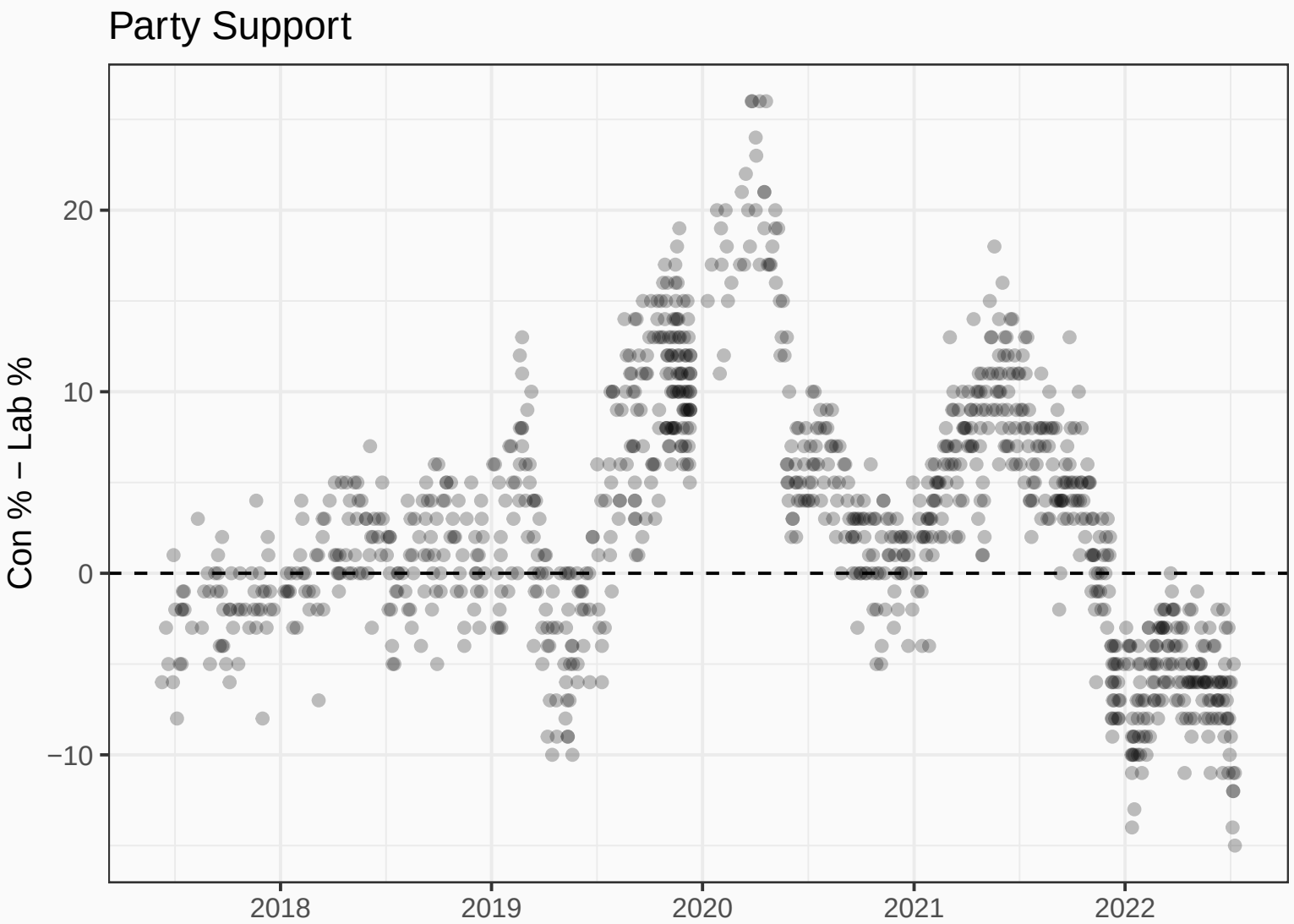
How popular are the UK political parties?

The last few years have seen dramatic changes in UK politics, most of which has been tracked closely by a variety of political polling companies. Individual polls are noisy manifestations of underlying trends in public opinion. We can think of the task of measuring public support for a given party as a prediction problem, where results in each polls are data and we want to predict the true average level of support at each point in time. What is the best model for predicting such trends?

Motivation


Party Support
% of popular vote
50
40
30
20
10
2018
2019
2020
2021
2022
Conservative
Labour
LibDem

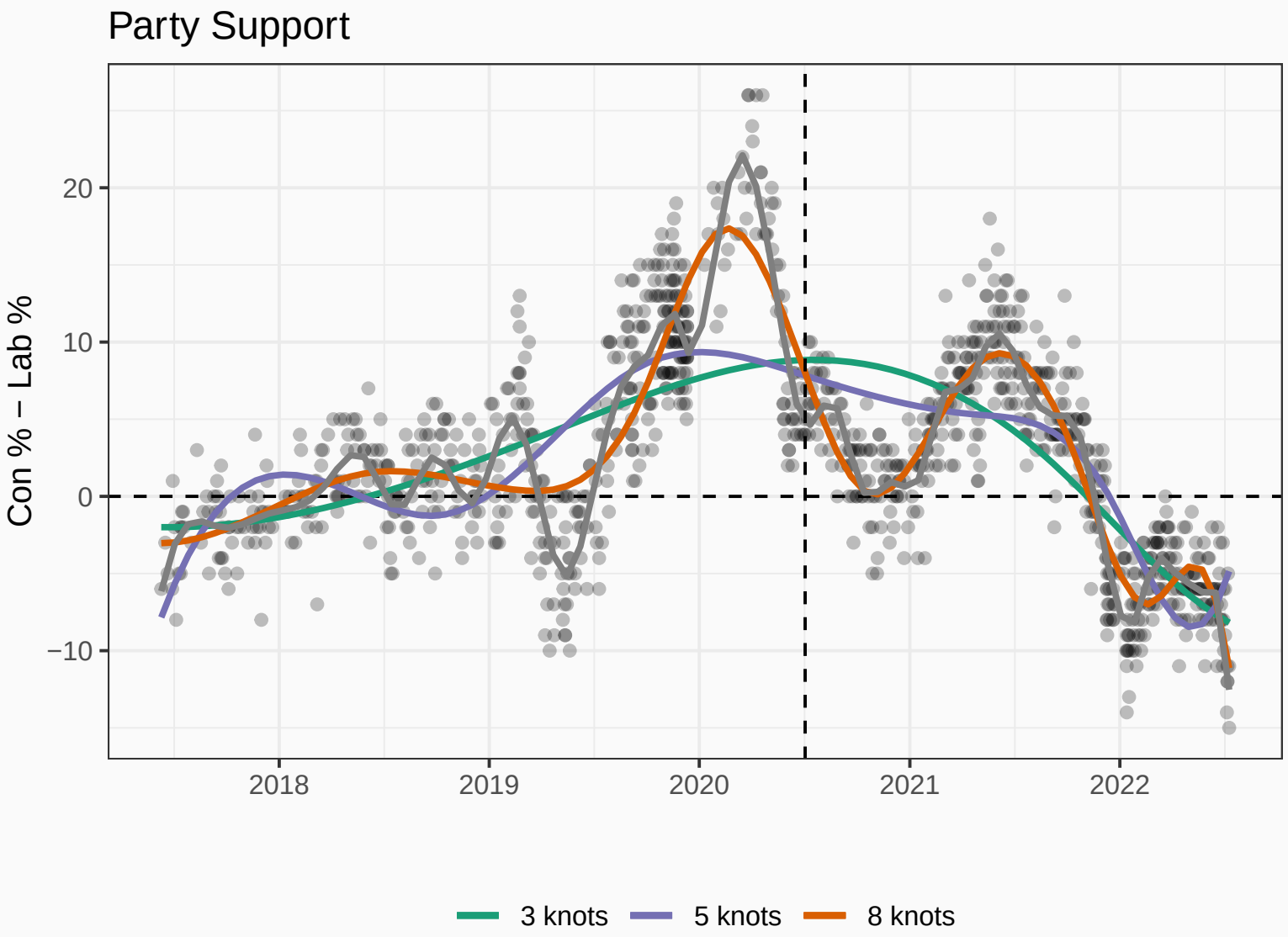
Motivation



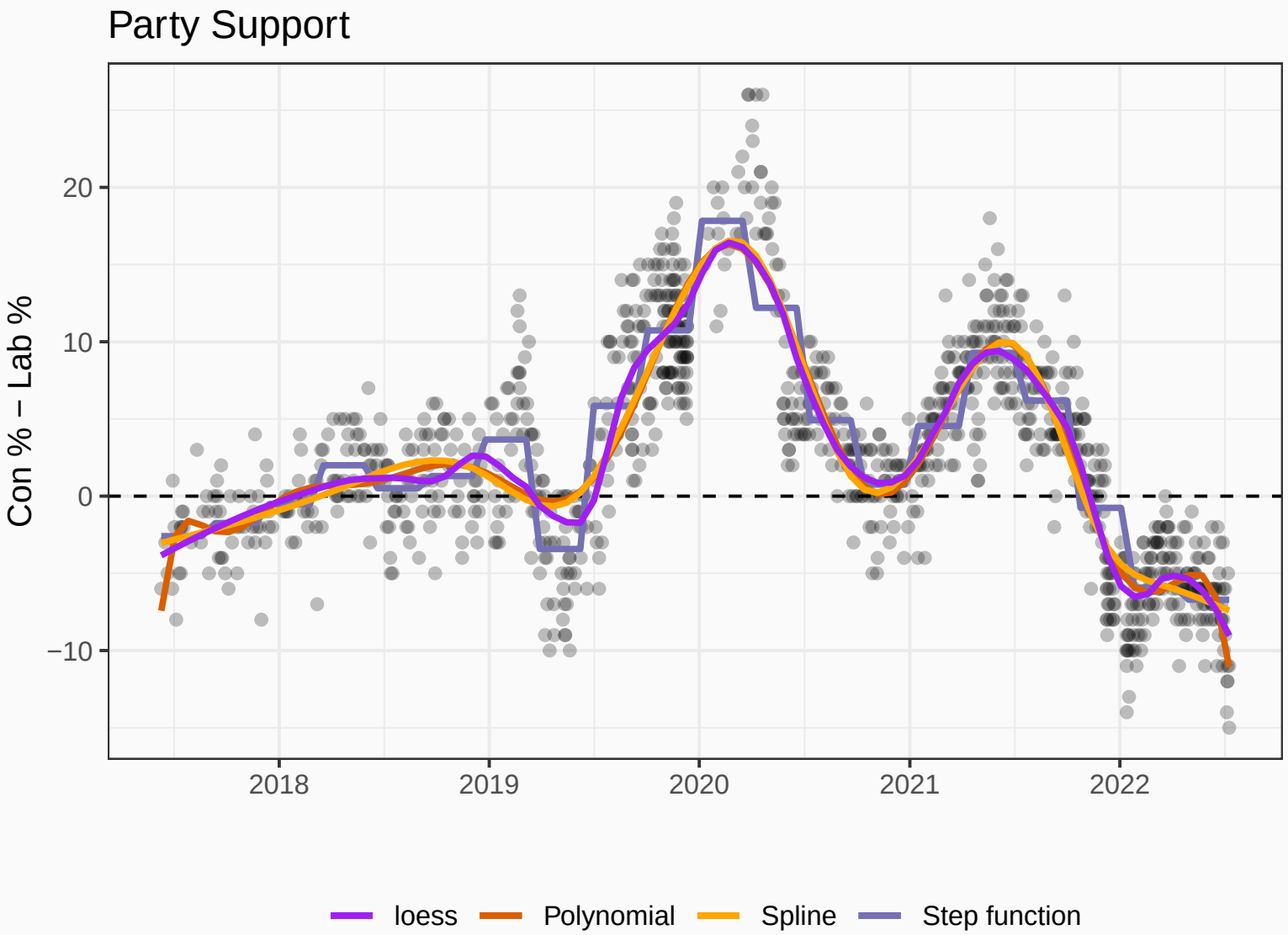
Splines

```
library(splines)
spline_mod_3 <- lm(con_lead ~ bs(date, df = 6, degree = 3), data = polls)
spline_mod_5 <- lm(con_lead ~ bs(date, df = 8, degree = 3), data = polls)
spline_mod_8 <- lm(con_lead ~ bs(date, df = 11, degree = 3), data = polls)
```

Splines


Party Support
Con % – Lab %
20
10
0
−10
2018
2019
2020
2021
2022
3 knots
5 knots
8 knots

Which Wiggle is Best?



Motivation

Almost all quantitative analyses require the analyst to make modelling decisions:

- Which variables should I use to predict my outcome?
- Should I use a linear model, or a non-linear model?
- Should I just use X in my regression? Or should I also use X^2? (or X^3? or X^4?)
- How many knots should I include in my spline? 2? 3? More?

These decisions can have important consequences for our results, but it is not always clear how to make them.

Day 7 Outline

Cross-validation

Bootstrap

Linear Model Selection and Regularization

Cross-validation

Training Error versus Test error

- The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training.
- Training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

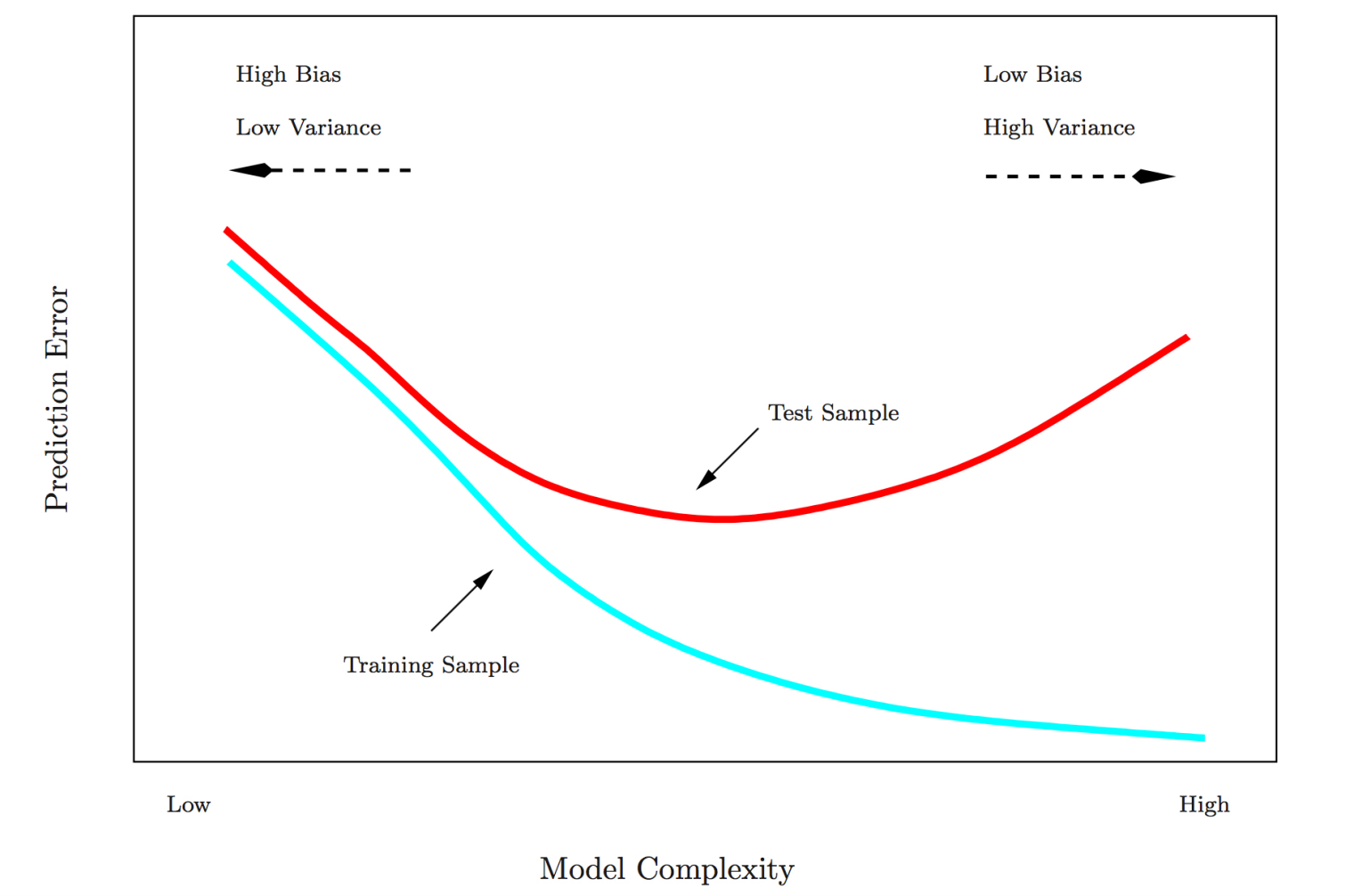
Bias-Variance Trade-Off

We can think of the test error associated with any given statistical estimator as coming from two fundamental quantities:

1. **Bias**
    - The bias of an estimator is the error that is introduced by approximating a complicated set of relationships with a simple model that doesn't characterise the full complexity
2. **Variance**
    - The variance of an estimator is the amount that the predictions produced by the estimator would change if it had been estimated on different data

Ideally we would like to minimize both variance and bias, but these goals are often at odds with each other.

Training- versus Test-Set Performance



Training- versus Test-Set Performance

- As we use more flexible models, the variance will increase and the bias will decrease
- The relative rate of change of these two quantities determines whether the test error increases or decreases
- As we start to make the model more flexible the bias will tend to decrease faster than the variance will increase
- After some point, adding more flexibility will decrease the the bias only a small amount, but the variance will increase a lot

Implication: We need tools which tell us when we have reached the optimal balance between bias and variance.

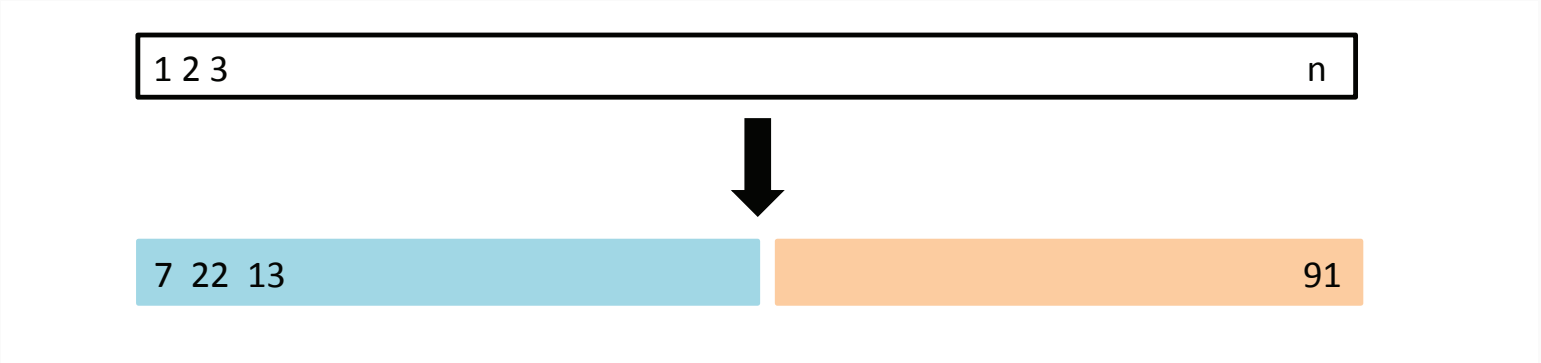
More on prediction-error estimates

- Best solution: a large designated test set. Often not available.
- Some methods make a mathematical adjustment to the training error rate in order to estimate the test error rate. These include the Cp statistic, AIC and BIC.
- We consider a class of methods that estimate the test error by holding out a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.

Validation-set approach

- We randomly divide the available set of samples into two parts: a training set and a validation or hold-out set.
- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.
- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response and misclassification rate for qualitative response models.

The Validation process



A random splitting into two halves: left part is training set, right part is validation set.

Example I

- We want to select the df parameter in our spline model for the polling data
- We randomly divide the polling observations into a training set and a validation set
- We estimate the model on the training set, once for each value of df
- We calculate the MSE for *both* the observations in the training set and the validation set

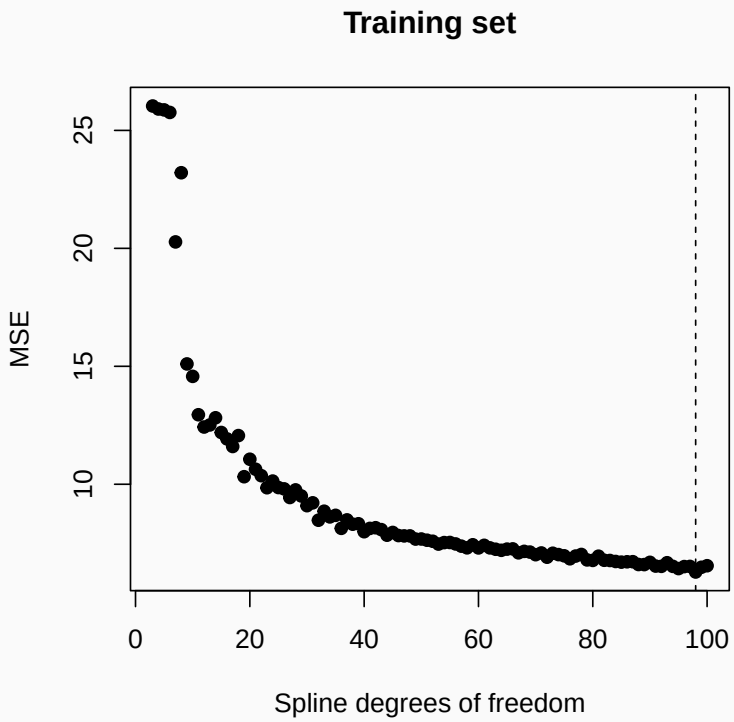
Example I

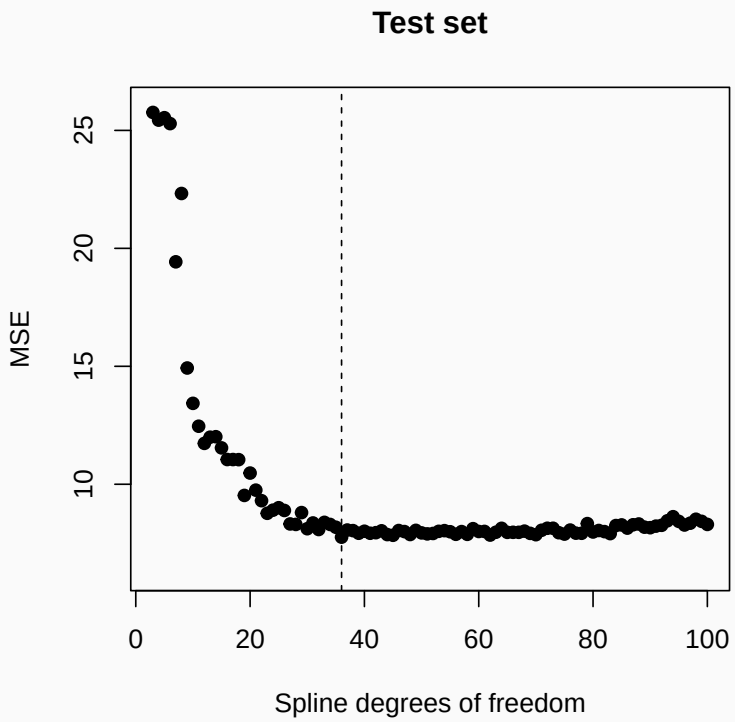
```
set.seed(221186)

train <- sample(1:nrow(polls), nrow(polls)/2)
polls_train <- polls[train,]
polls_test <- polls[-train,]

# Set of values for df parameter
dfs <- 3:100

# Data.frame to store output
out <- data.frame(df = dfs, mse_train = NA, mse_test = NA)
```

Example I

```
for(i in 1:length(dfs)){

    mod_out <- lm(con_lead ~ bs(date, df = dfs[i], degree = 3),
                  data = polls_train) # Estimate model

    # Predict for training set
    polls_train$fitted_con_lead <- predict(mod_out, newdata = polls_train)

    # Predict for test set
    polls_test$fitted_con_lead <- predict(mod_out, newdata = polls_test)

    # Training MSE
    out$mse_train[i] <- mean((polls_train$con_lead - polls_train$fitted_con_lead)^2

    # Test MSE
    out$mse_test[i] <- mean((polls_test$con_lead - polls_test$fitted_con_lead)^2)

}
```

Example I

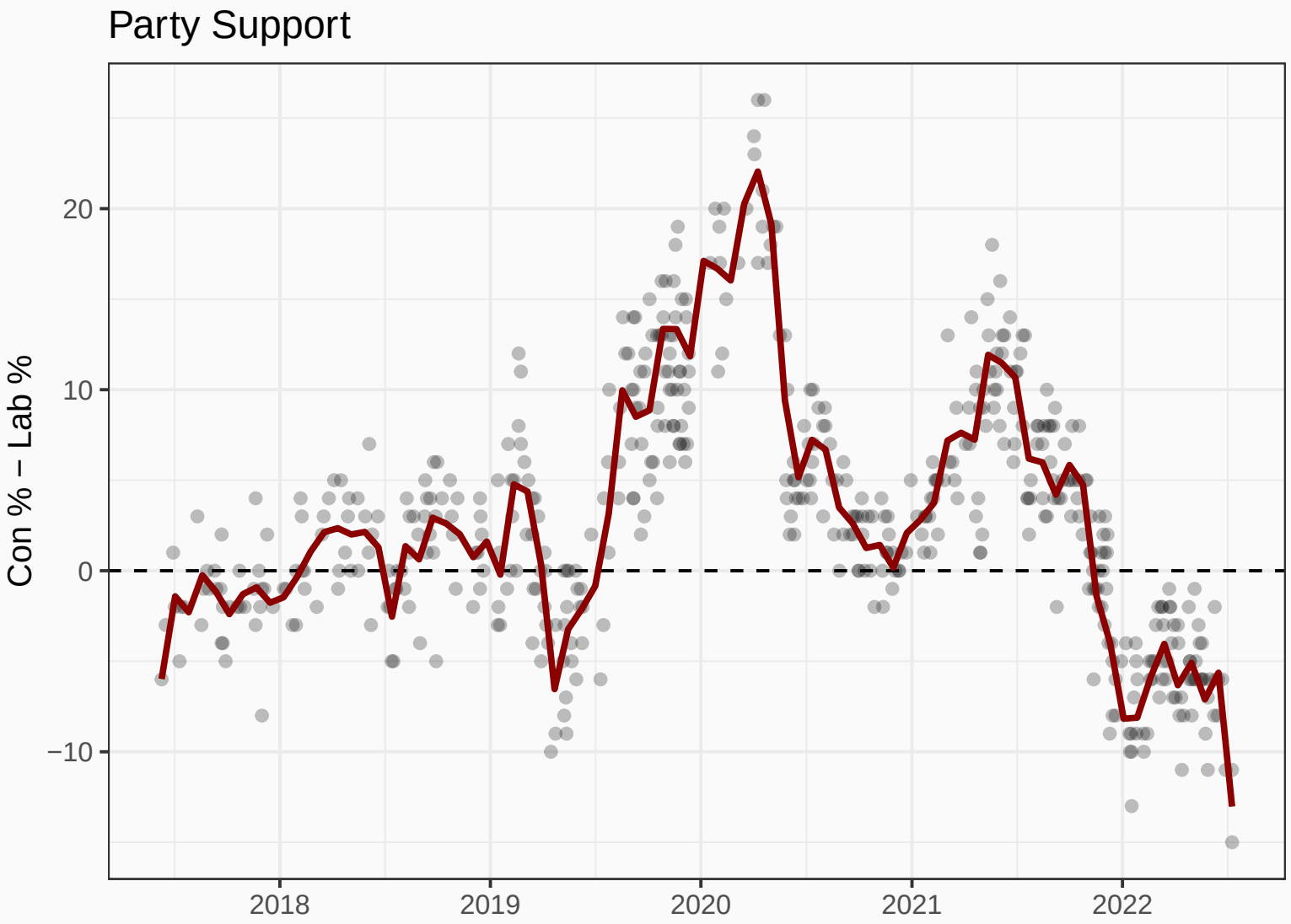

Training set
MSE
10
15
20
25
0
20
40
60
80
100
Spline degrees of freedom


Test set
MSE
10
15
20
25
0
20
40
60
80
100
Spline degrees of freedom

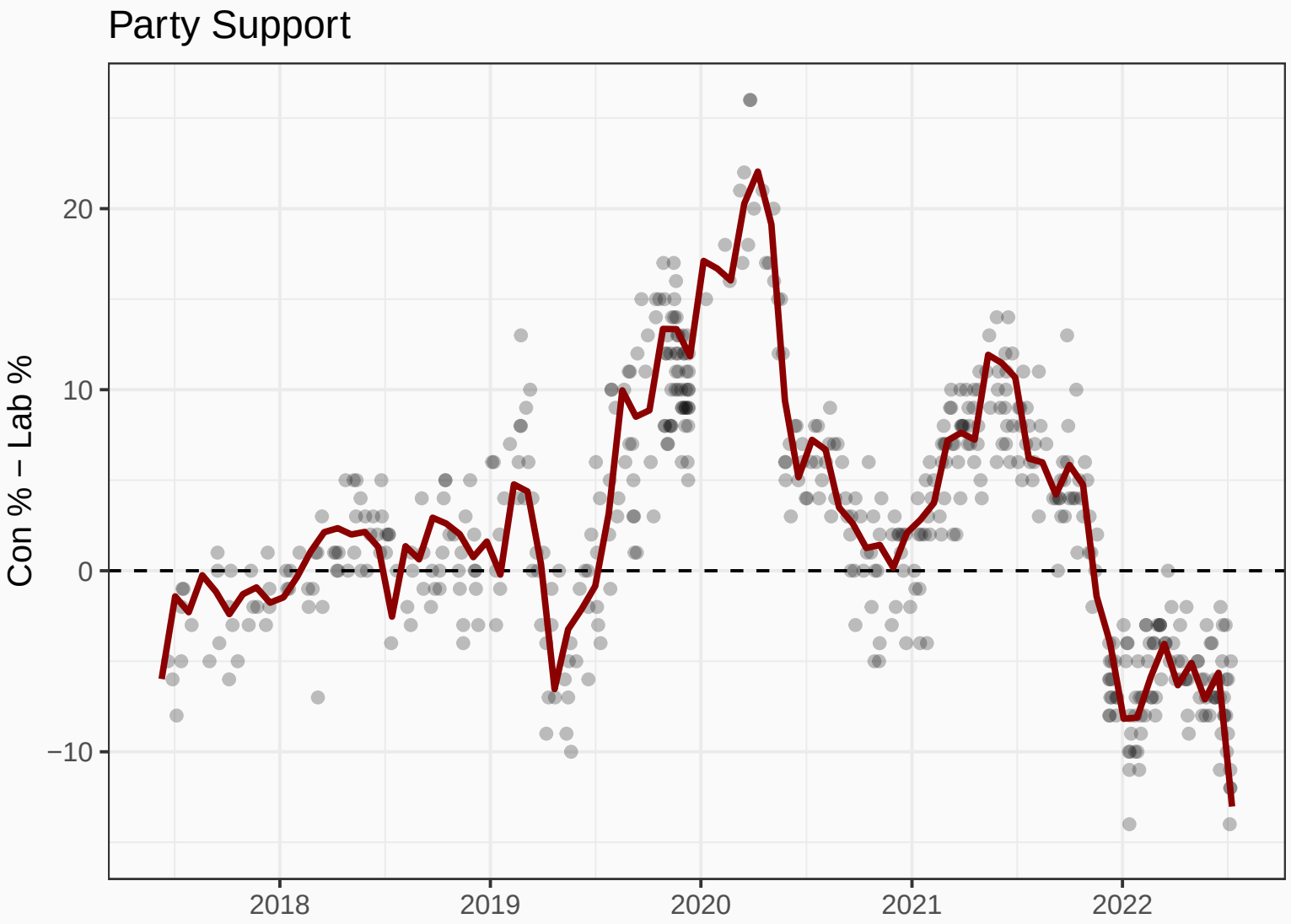
Why does this happen?

- As we increase the value for df, we are asking the model to fit an increasingly flexible (wiggly) line to our data
- We can make the line arbitrarily flexible so it fits the training set data better and better
- But, the wiggly line fit to the training data may capture idiosyncrasies that are not present in the validation data!
- After a while, increases to df make our predictions for the validation set worse

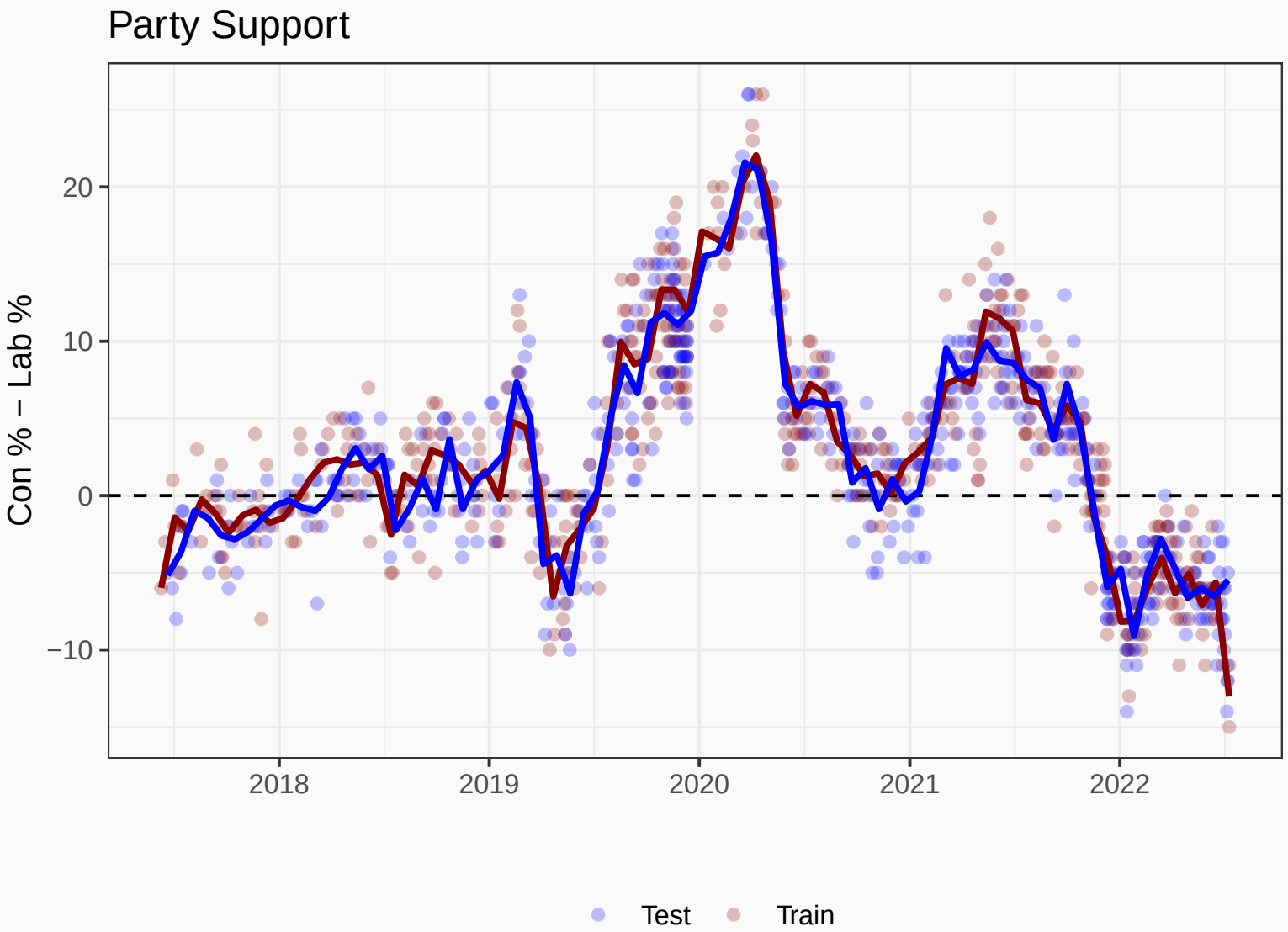
Why does this happen?



Why does this happen?



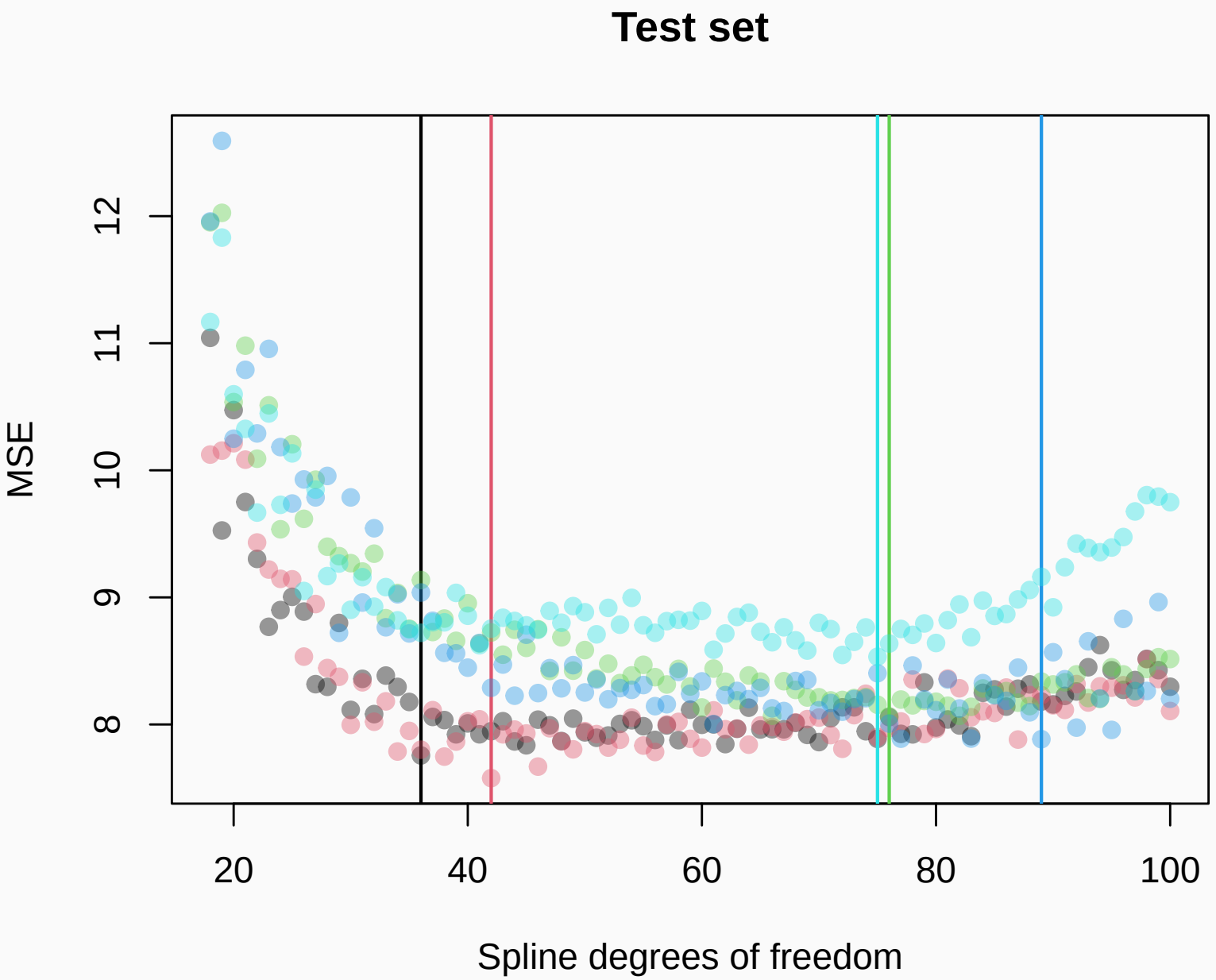
Why does this happen?



Drawbacks of validation set approach

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- 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.
- This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire dataset.

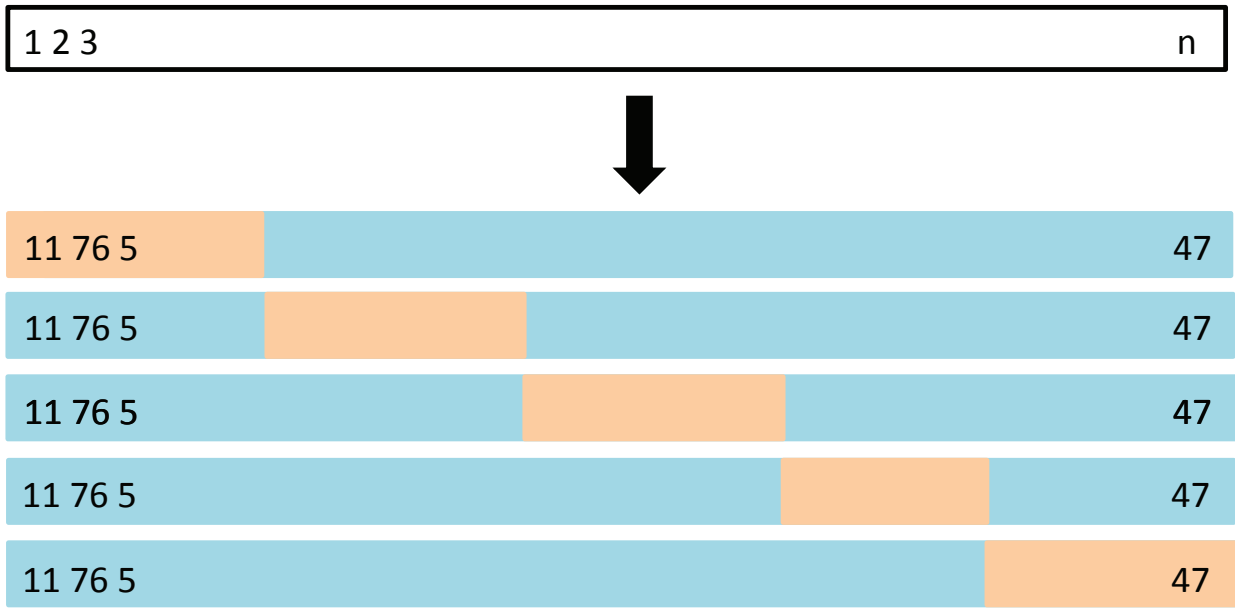
Variable Estimates of the Test Error



K-fold Cross-validation

- K-fold cross-validation generalises the idea of the training/validation approach and is a very popular way of estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Idea is to randomly divide the data into K equal-sized parts. We leave out part k, fit the model to the other $K - 1$ parts (combined), and then obtain predictions for the left-out kth part.
- This is done in turn for each part $k = 1, 2, \dots, K$, and then the results are combined.

5-fold CV



Mechanism

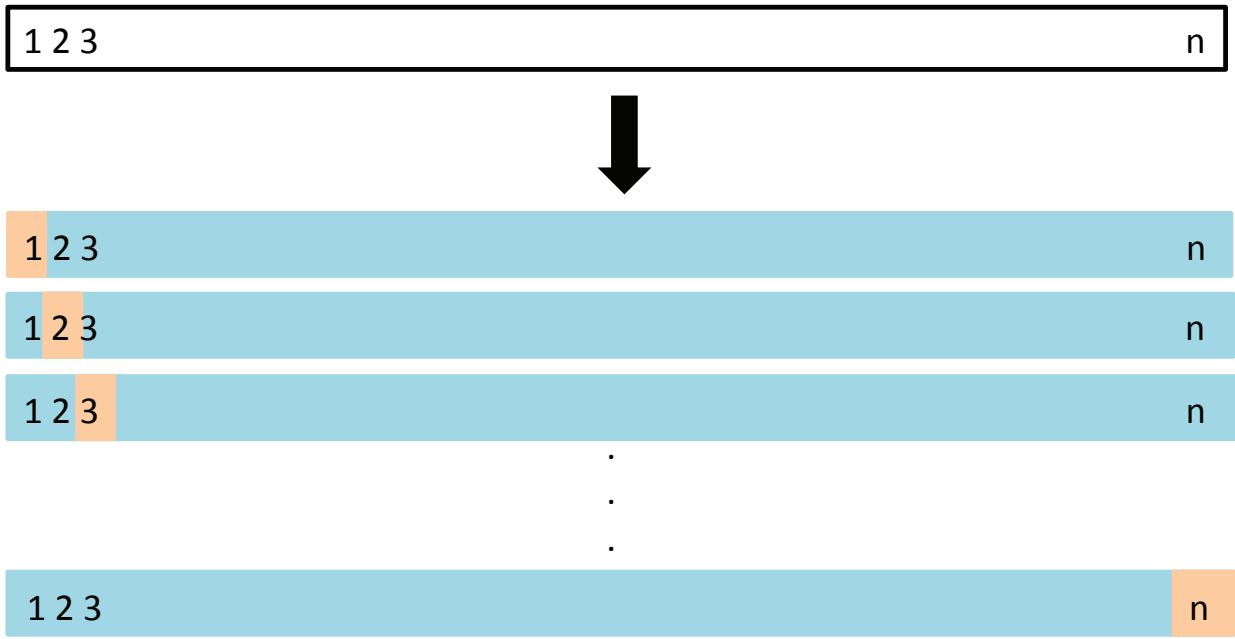
- Let the K parts be $C_1, C_2, \ldots, C_K$, where C_K denotes the indices of the observations in part k. There are n_k observations in part k: if N is a multiple of K, then $n_k = n/K$.
- Compute

$$CV_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} \text{MSE}_k$$

  where $\text{MSE}_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$, and $\hat{y}_i$ is the fit for observation i, obtained from the data with part k removed.

- Setting $K = n$ yields n-fold or leave-one out cross-validation (LOOCV).

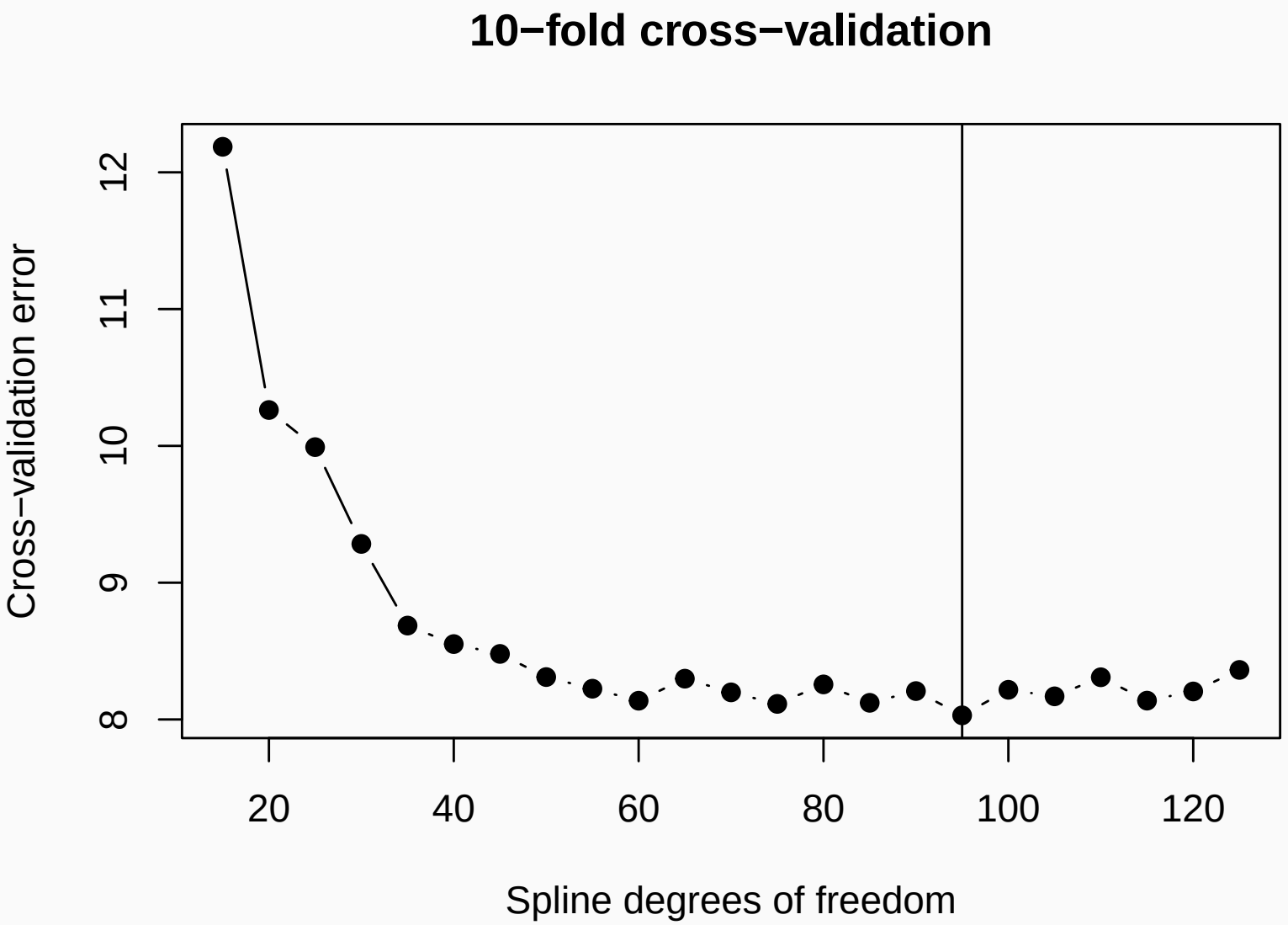
LOOCV



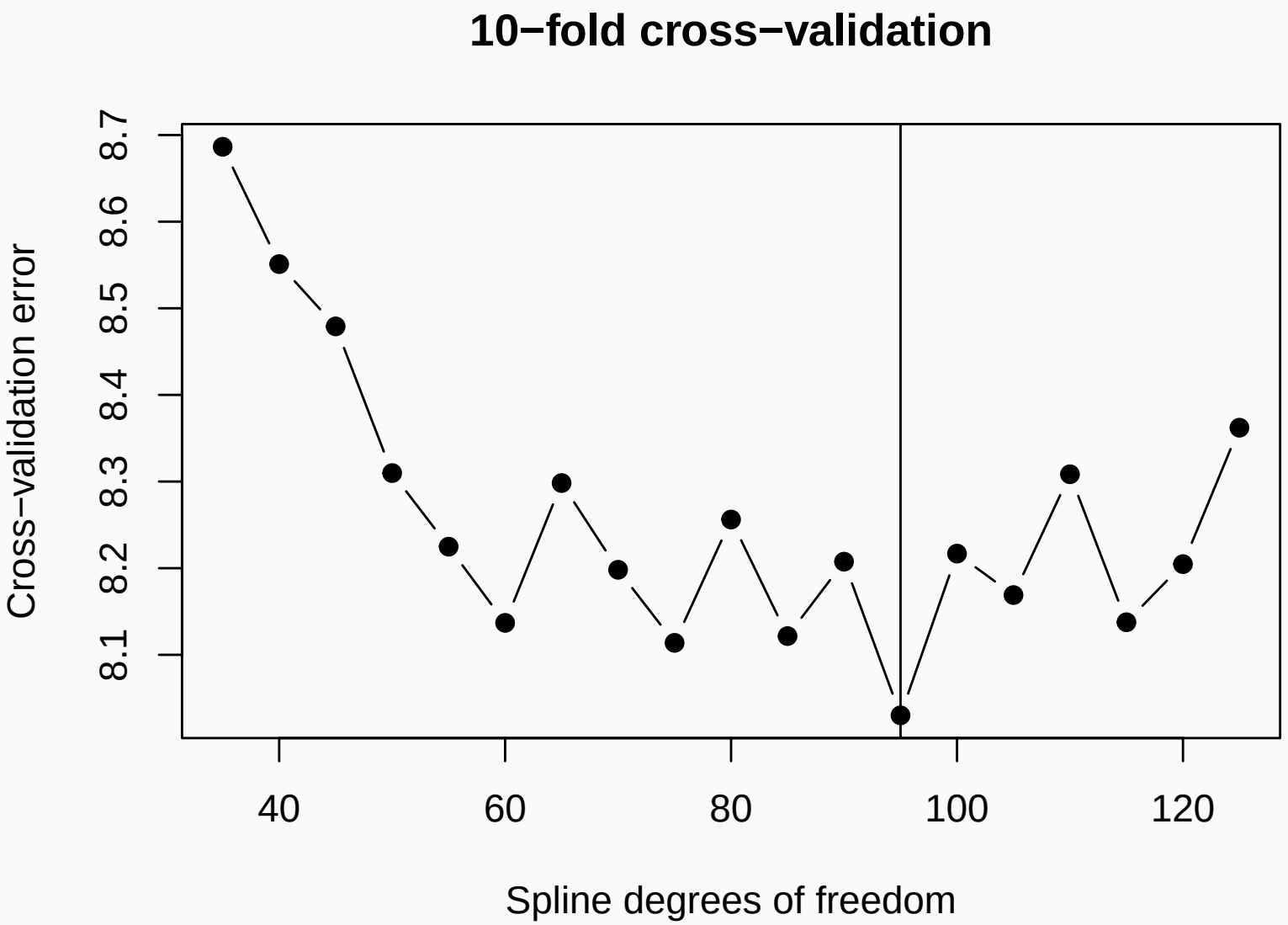
LOOCV

- With OLS, LOOCV can be attractive because there is an analytical solution which can be calculated from a single fit
- With other methods (logistic regression, GAMs, Random Forests, etc), LOOCV can still be used but is computationally expensive
- In addition, the estimates from each fold are highly correlated and hence their average can have high variance.
- A better choice is normally $K = 5$ or $K = 10$.

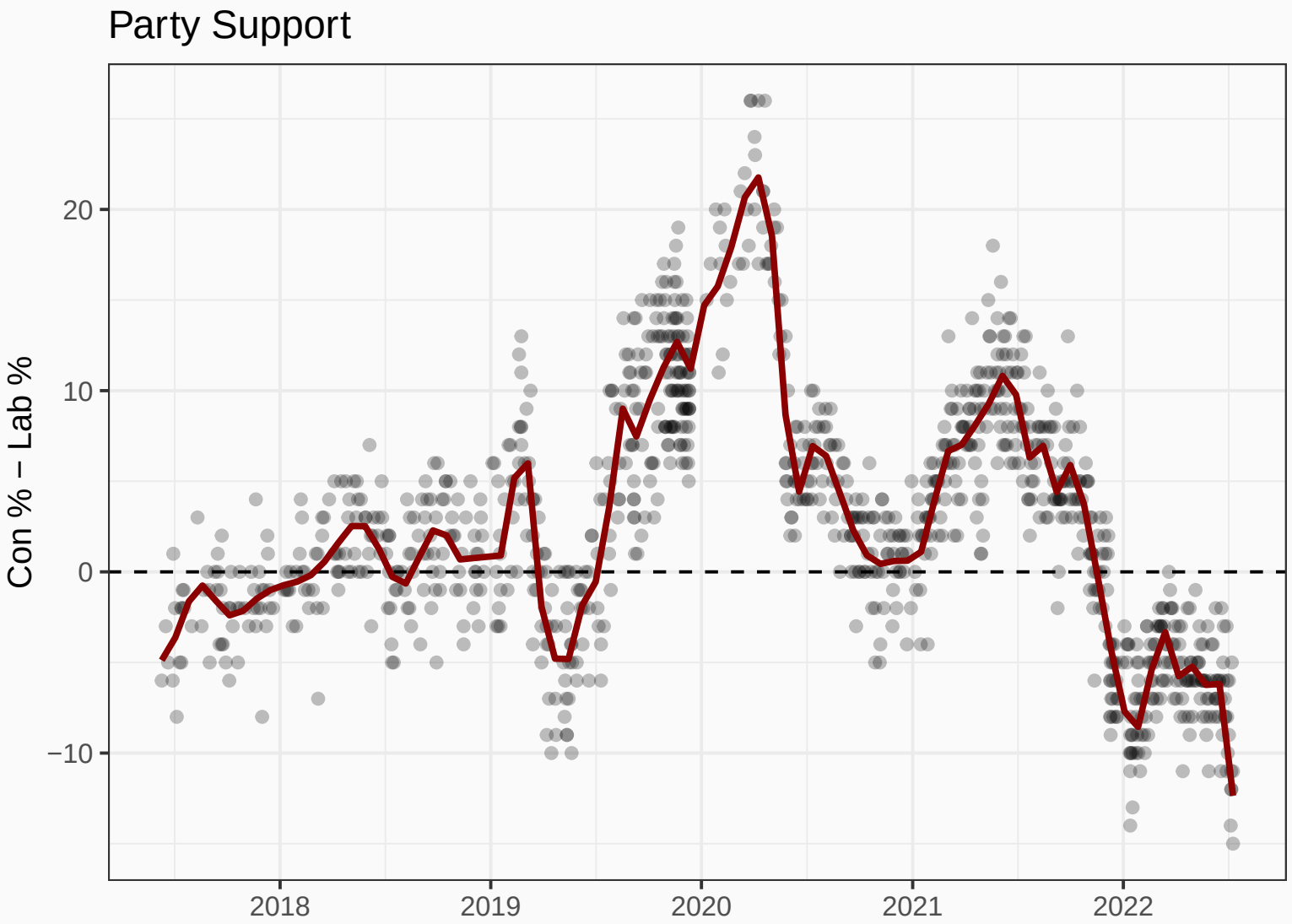
K-fold Cross-validation



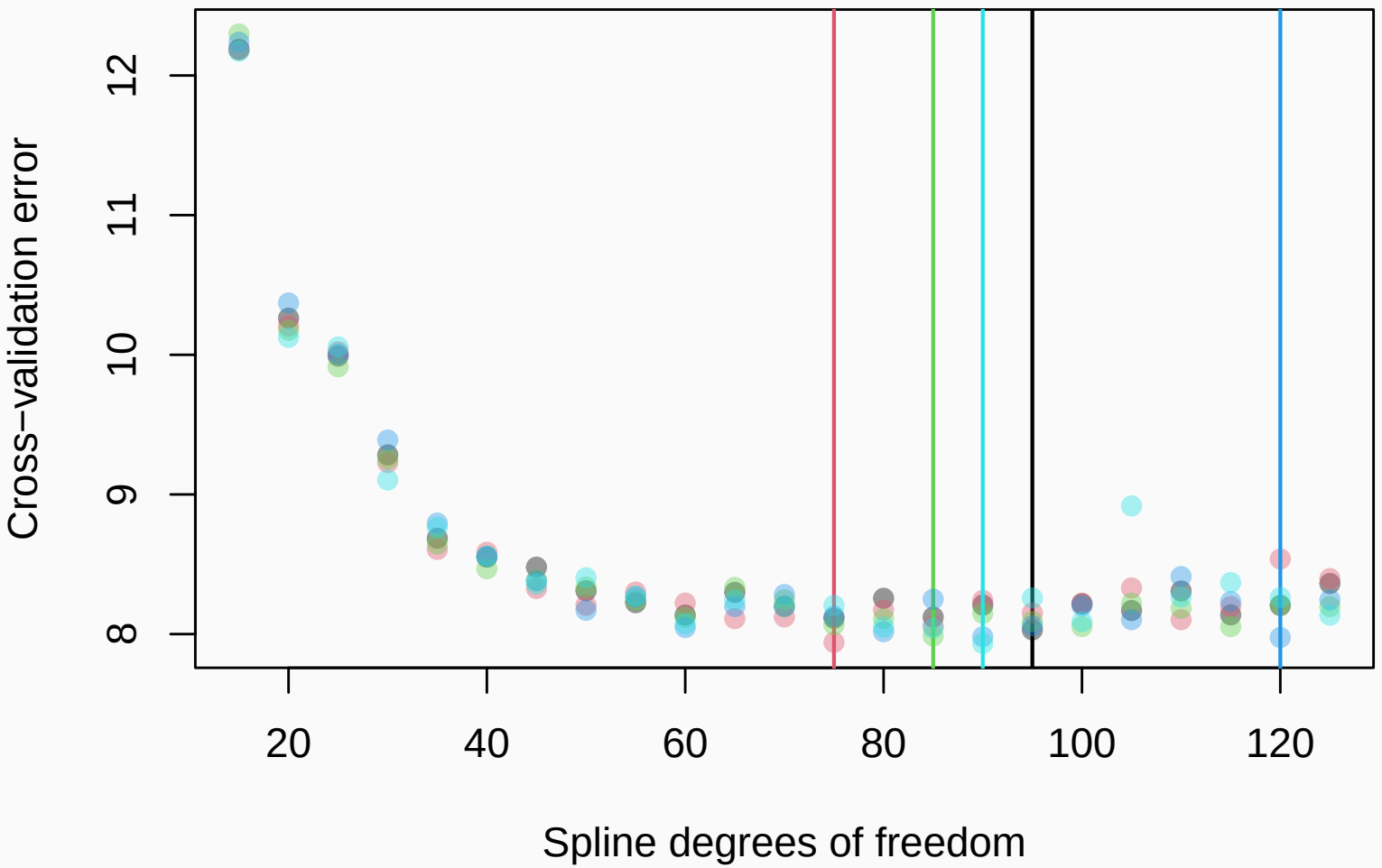
K-fold Cross-validation



K-fold Cross-validation



K-fold Cross-validation



Additional issues with CV

- Since each training set is only $(K-1)/K$ as big as the original training set, the estimates of prediction error will typically be biased upward.
- This bias is minimized when $K = n$ (LOOCV), but this estimate has high variance, as noted earlier.
- $K = 5$ or 10 provides a good balance for this bias-variance tradeoff.

CV for classification

- We divide the data into K roughly equal-sized parts $C_1, C_2, \ldots, C_K$. C_k denotes the indices of the observations in part k. There are n_k observations in part k: if n is a multiple of K, then $n_k = n/K$.
- Compute

$$CV_K = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{Err_k}$$

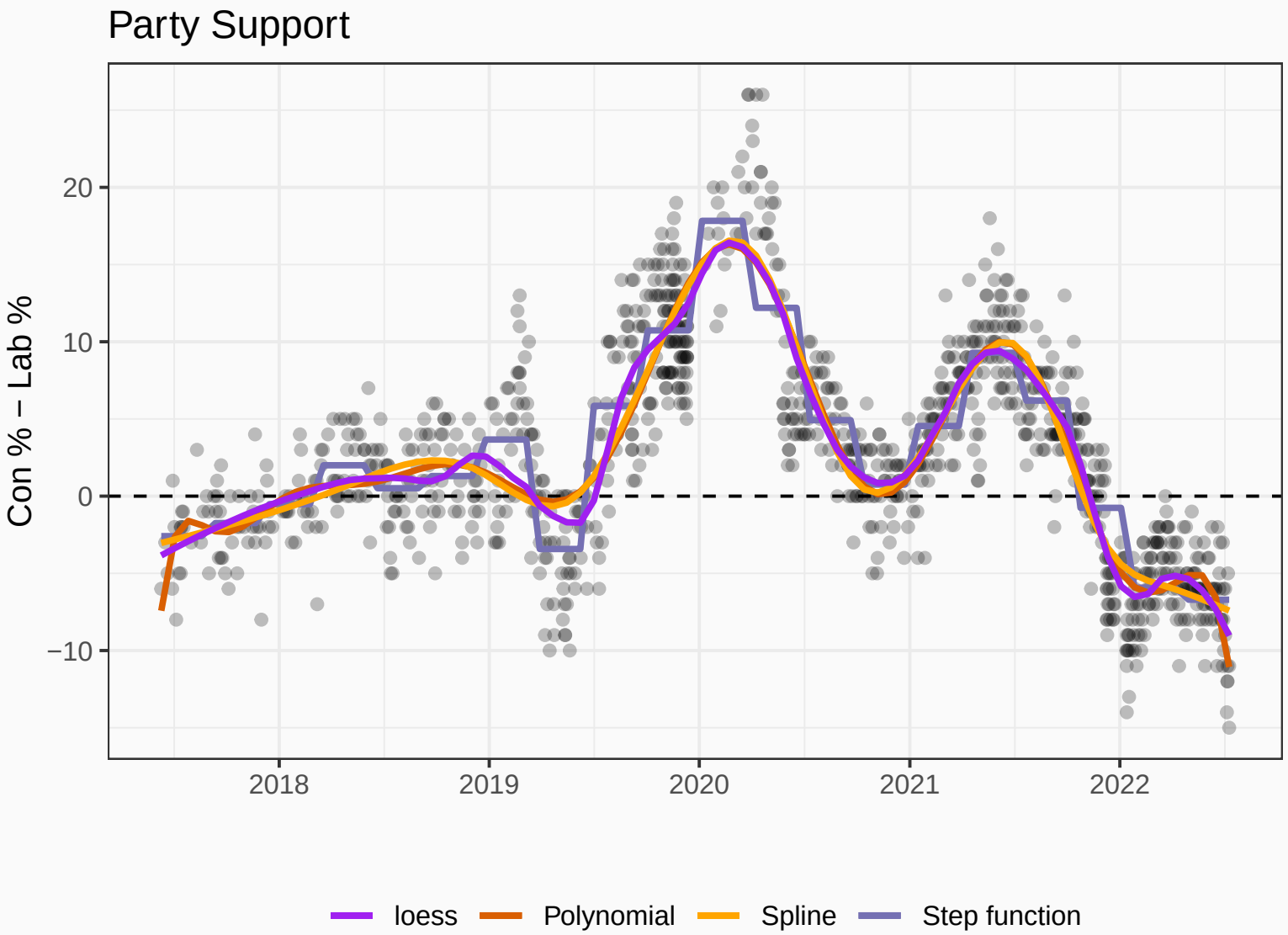
where $\mathrm{Err_k} = \sum_{i \in C_k} I(y_i \neq \hat{y}_i)/n_k$.

CV for other models and modelling decisions

Cross-validation is a very general strategy for evaluating predictive fit

- Which variables should I use to predict my outcome?
    - Which ones minimize the CV error?
- Should I use a linear model, or a non-linear model?
    - Which minimizes the CV error?
- Should I just use X in my regression? Or should I also use X^2? (or X^3? or X^4?)
    - Which minimizes the CV error?
- How many knots should I include in my spline? 2? 3? More?
    - Which minimizes the CV error?

Which Wiggle is Best?

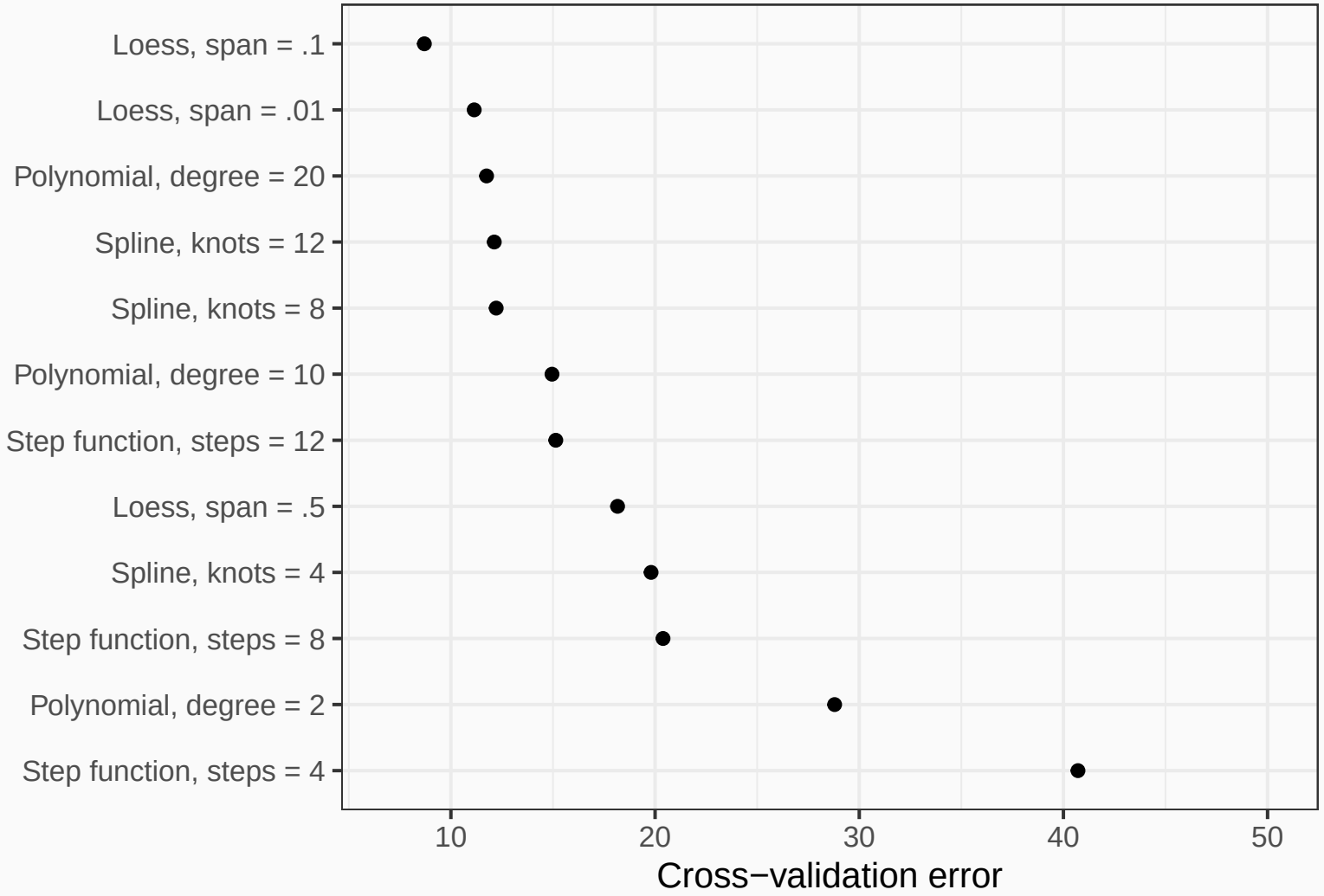


Which Wiggle is Best?

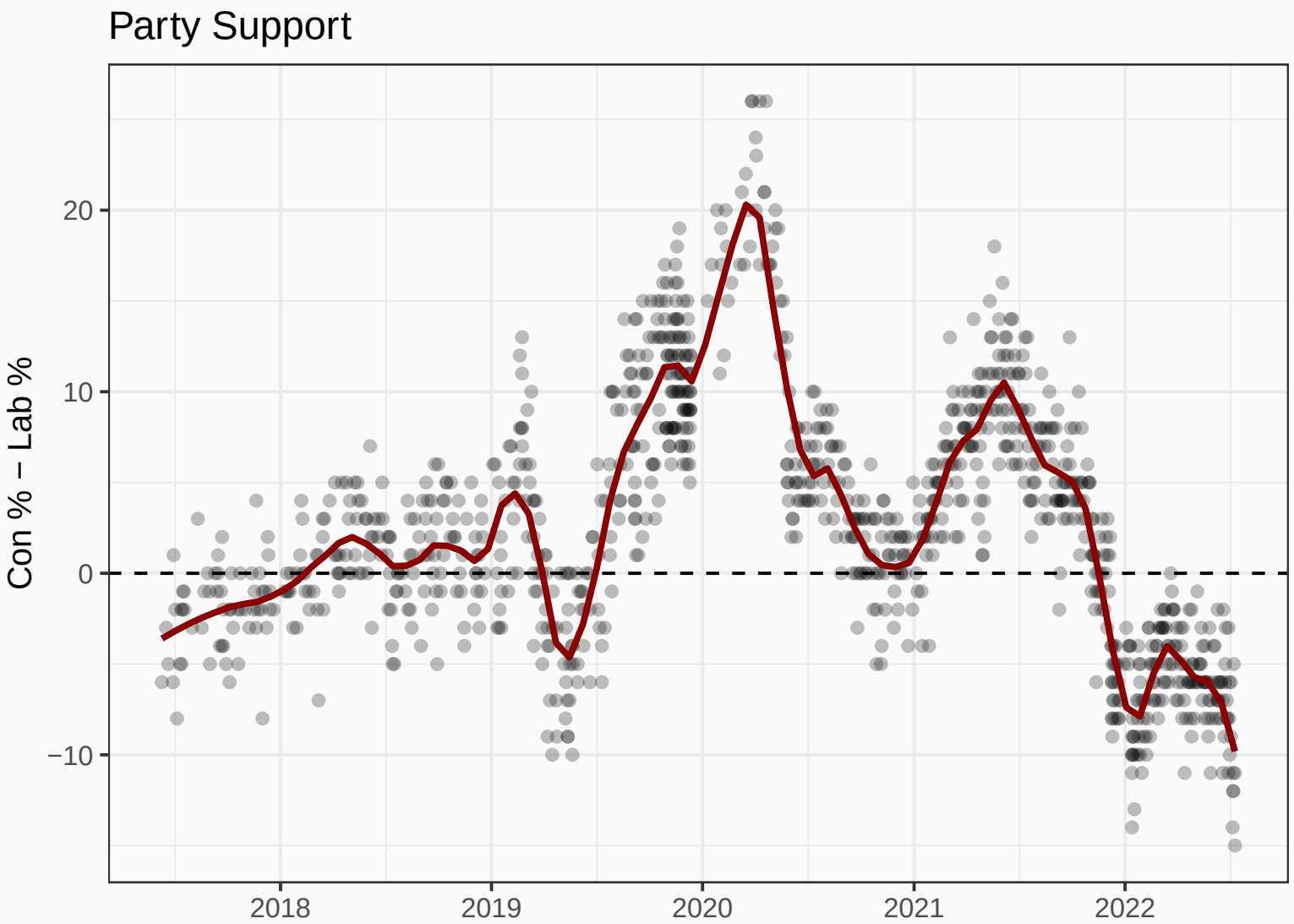
Conduct 10-fold cross-validation for the following 12 models:

1. **Step function**
    - 4 steps; 8 steps; 12 steps
2. **Polynomials**
    - Degree = 2; Degree = 10; Degree = 20
3. **Cubic Splines**
    - 4 knots; 8 knots; 12 knots
4. **Loess**
    - span = .5; span = .1; span = .01

Which Wiggle is Best?



Which Wiggle is Best?



Bootstrap

Bootstrap

- The bootstrap is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
- E.g., it can provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.

Where does the name came from?

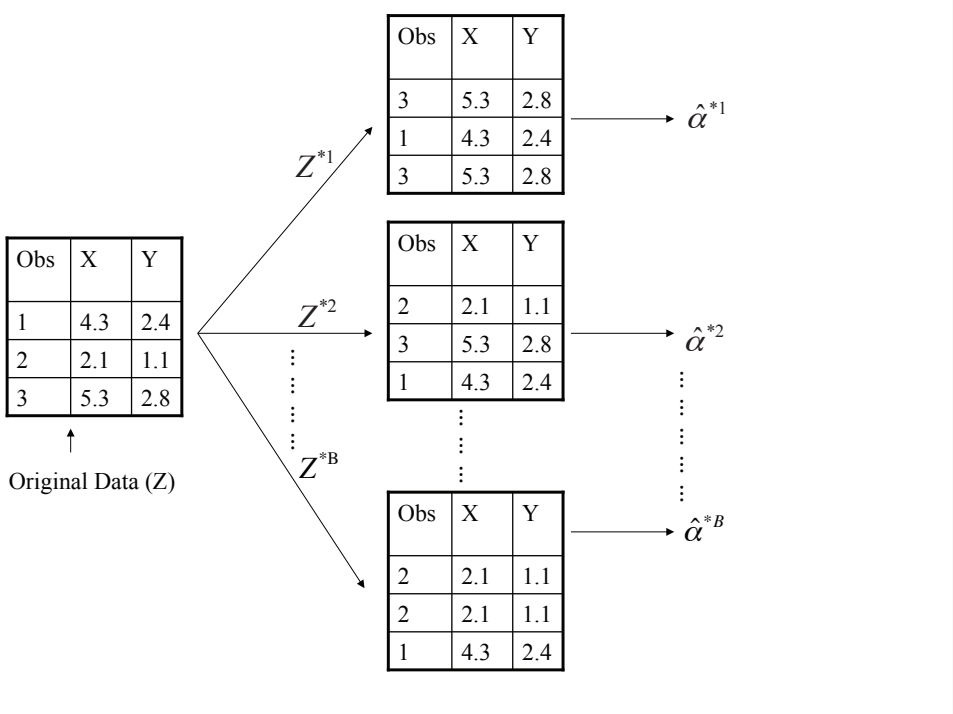
- The use of the term bootstrap derives from the phrase to pull oneself up by one's bootstraps, widely thought to be based on one of the eighteenth century "The Surprising Adventures of Baron Munchausen" by Rudolph Erich Raspe:

  The Baron had fallen to the bottom of a deep lake. Just when it looked like all was lost, he thought to pick himself up by his own bootstraps.

Intuition

- We usually care about uncertainty around our estimates from a sample. One way would be to repeatedly sample the population. But we cannot do that in real world.
- The bootstrap approach allows us to replicate this process of obtaining new data sets, so that we can estimate the variability of our estimate without generating additional samples.
- Rather than repeatedly obtaining independent data sets from the population, we instead obtain distinct data sets by repeatedly sampling observations from the original data set with replacement.
- Each of these “bootstrap data sets” is created by sampling with replacement, and is the same size as our original dataset. As a result some observations may appear more than once in a given bootstrap data set and some not at all.

Example with three observations



- Each bootstrap dataset contains n observations, sampled with replacement from the original data set.
- Each bootstrap data set is used to obtain an estimate of α.

Procedure

The bootstrap is commonly used to construct standard errors or related quantities like confidence intervals. How is this achieved?

- Denoting the first bootstrap dataset by Z^{*1}, we use Z^{*1} to produce a new bootstrap estimate for α, which we call $\hat{\alpha}^{*1}$.
- This procedure is repeated B times for some large value of B (say 1000 or 10,000), in order to produce B different bootstrap datasets, $Z^{*1}, Z^{*2}, \ldots, Z^{*B}$, and B corresponding α estimates, $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$.
- We then estimate the standard error of the parameter α by calculating the standard deviation of the bootstrap estimates:

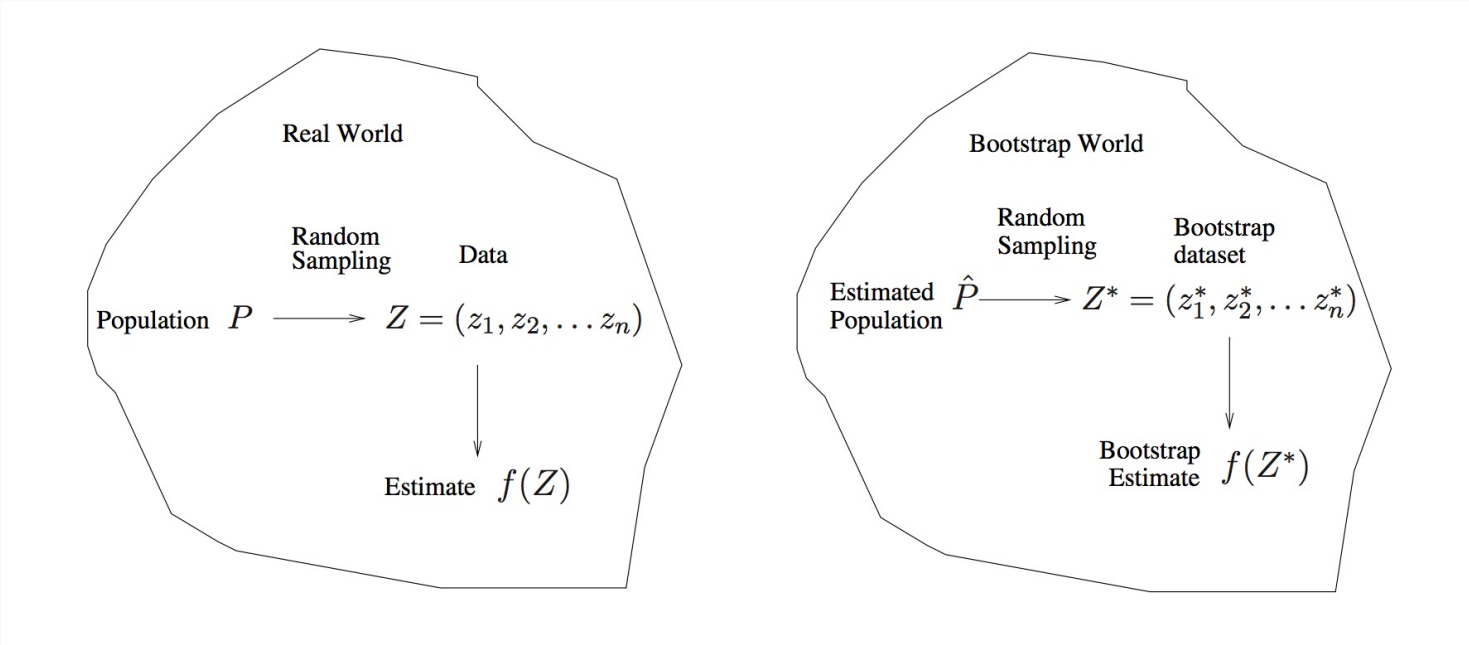
$$\mathrm{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \bar{\hat{\alpha}}^{*})^2}$$

- This serves as an estimate of the standard error of $\hat{\alpha}$ estimated from the original data set.

What is going on here?

- Recall the definition of a standard error from an intro stats class: the standard error is the *estimate* of the standard deviation of the sampling distribution
- Recall also that the sampling distribution is *unobservable*! It is the distribution of estimates of a parameter that would result if we sampled repeatedly from the population
- The idea behind the bootstrap is to treat our *data* as the entire population, and resample from it repeatedly, estimating our parameter for each sample
- We can then directly calculate the standard deviation of the bootstrapped estimates to estimate the standard error

A general picture for the bootstrap



Example - bootstrapping a regression standard error

Why do polling firms like YouGov tweak polls? Because they are scared of being wrong

Peter Kellner

The pressure to avoid mistakes can force pollsters to second-guess themselves, just as YouGov did in 2017

Question: Are YouGov polls more pro-Conservative than those of other pollsters?

Example - bootstrapping a regression standard error

```
dim(polls)
```

```
## [1] 1047   31
```

Example - bootstrapping a regression standard error

```
dim(polls)
```

```
## [1] 1047   31
```

```
table(polls$you_gov)
```

```
##
## FALSE  TRUE
##   788   259
```

Example - bootstrapping a regression standard error

```
dim(polls)
```

```
## [1] 1047   31
```

```
table(polls$you_gov)
```

```
##
## FALSE  TRUE
##   788   259
```

```
true_model <- lm(con_lead ~ you_gov, data = polls)
summary(true_model)
```

```
...
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.3756     0.2481   9.574  < 2e-16 ***
## you_govTRUE   1.4854     0.4989   2.977  0.00298 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
...
```

Example - bootstrapping a regression standard error

- The OLS estimate of the ”yougov” coefficient is 1.4853693
- The OLS standard error on the ”yougov” coefficient is 0.4989103
- Can we replicate this coefficient using a bootstrap procedure?

Example - bootstrapping a regression standard error

```
set.seed(221186)
polls_boot <- polls[sample(1:nrow(polls),nrow(polls),replace = TRUE),]
dim(polls_boot)

## [1] 1047   31
```

Example - bootstrapping a regression standard error

```
set.seed(221186)
polls_boot <- polls[sample(1:nrow(polls),nrow(polls),replace = TRUE),]
dim(polls_boot)
```

```
## [1] 1047   31
```

```
boot_model <- lm(con_lead ~ you_gov, data = polls_boot)
coef(boot_model)
```

```
## (Intercept) you_govTRUE
##    2.224874    2.185484
```

Example - bootstrapping a regression standard error

```
set.seed(221186)
polls_boot <- polls[sample(1:nrow(polls),nrow(polls),replace = TRUE),]
dim(polls_boot)
```

```
## [1] 1047   31
```

```
boot_model <- lm(con_lead ~ you_gov, data = polls_boot)
coef(boot_model)
```

```
## (Intercept) you_govTRUE
##    2.224874    2.185484
```

- For a single bootstrap estimate, the estimate of the ”yougov” coefficient is 2.1854842
- This is close to, but not the same as, the OLS estimate
- What happens if we try another bootstrap dataset?

Example - bootstrapping a regression standard error

```
polls_boot <- polls[sample(1:nrow(polls),nrow(polls),replace = TRUE),]
boot_model <- lm(con_lead ~ you_gov, data = polls_boot)
coef(boot_model)
```

```
## (Intercept) you_govTRUE
##    2.329574    1.148338
```

- We get a slightly different estimate of the ”yougov” coefficient (1.1483377)
- This is close to, but not the same as, the OLS estimate
- This is close to, but not the same as, the first bootstrap estimate
- Let’s repeat this process thousands of times

Example - bootstrapping a regression standard error

```
boot_func <- function(){
    polls_boot <- polls[sample(1:nrow(polls),nrow(polls),replace = TRUE),]
    boot_model <- lm(con_lead ~ you_gov, data = polls_boot)
    boot_est <- coef(boot_model)[2]
    return(boot_est)
}
boot_ests <- replicate(2000, boot_func())
```

Example - bootstrapping a regression standard error

```
boot_func <- function(){
    polls_boot <- polls[sample(1:nrow(polls),nrow(polls),replace = TRUE),]
    boot_model <- lm(con_lead ~ you_gov, data = polls_boot)
    boot_est <- coef(boot_model)[2]
    return(boot_est)
}
boot_ests <- replicate(2000, boot_func())
```

```
head(boot_ests)
```

```
## you_govTRUE you_govTRUE you_govTRUE you_govTRUE you_govTRUE you_govTRUE
##    1.604624    1.255498    1.551409    1.955199    1.222042    1.156817
```

- We get a slightly different estimates of the ”yougov” coefficient for each bootstrap sample
- What does the distribution of these samples look like?

Example - bootstrapping a regression standard error

```
hist(boot_ests)
```

- The estimates from the bootstrap samples are centred close to the true parameter estimate (1.4853693)
- There is variability across the bootstrap samples

Example - bootstrapping a regression standard error

How close are our estimates of the regression coefficient and the standard error from the bootstrap to the original regression model?

```
mean(boot_ests)
```

```
## [1] 1.480512
```

```
sd(boot_ests)
```

```
## [1] 0.4921922
```

Example - bootstrapping a regression standard error

How close are our estimates of the regression coefficient and the standard error from the bootstrap to the original regression model?

```
mean(boot_ests)
```

```
## [1] 1.480512
```

```
sd(boot_ests)
```

```
## [1] 0.4921922
```

```
coef(summary(true_model))
```

```
##             Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 2.375635  0.2481413 9.573717 7.185826e-21
## you_govTRUE 1.485369  0.4989103 2.977227 2.975745e-03
```

- Pretty close!

But we already knew this!

- In this case, we already had a simpler way to estimate the standard error
- The bootstrap, however, can be used in more complicated situations where analytical solutions are not available

But we already knew this!

For instance, if we have two estimation processes chained together, we may want to incorporate the estimation uncertainty from the first process into the second process

- You have a large dataset with hundreds of covariates
- You run a principle component analysis to extract a low-dimensional summary of your data
- You then include the principle components into a subsequent regression for predicting an outcome
- The standard errors for the regression do not incorporate the uncertainty in the PCA
- $\longrightarrow$ The regression standard errors will be too small!

Bootstrapping the *entire* process, including both the PCA and the regression, will result in the correct standard errors.

The bootstrap in general

- In more complex data situations, figuring out the appropriate way to generate bootstrap samples can require some thought.
- For example, if the data is a time series, we can't simply sample the observations with replacement.
- We can instead create blocks of consecutive observations, and sample those with replacements. Then we paste together sampled blocks to obtain a bootstrap dataset.

Bootstrap and prediction error

- In cross-validation, each of the K validation folds is distinct from the other $K-1$ folds used for training: there is no overlap. This is crucial for its success.
- To estimate prediction error using the bootstrap, we could think about using each bootstrap dataset as our training sample, and the original sample as our validation sample.
- But each bootstrap sample has significant overlap with the original data. About two-thirds of the original data points appear in each bootstrap sample.
- This will cause the bootstrap to seriously underestimate the true prediction error.

Removing the overlap

- We can fix this problem by only using predictions for those observations that did not (by chance) occur in the current bootstrap sample.
- This is the out-of-bag error estimate that we used yesterday when discussing bagging and Random Forests
- Generally, this method is a little more complicated to code and cross-validation tends to provide a simpler, more attractive approach for estimating test error

Linear Model Selection and Regularization

Regularization

- We have seen already that sometimes the model that fits best out-of-sample is not the one that fits best in-sample.
- We have already seen cross-validation as a way to select among a set of models to optimize out-of-sample fit.
- Regularization is a general idea, with lots of implementations in different situations
    - Regularization methods "intentionally fail" to optimize in-sample fit, in order to better optimize out-of-sample fit.
    - Built into each of these methods is a *penalty* for the kind of over-fitting that tends to lead to poor out-of-sample fit, in that class of models.

- Recall the linear model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon.$$

- Despite its simplicity, the linear model has distinct advantages in terms of its interpretability and often shows good predictive performance.
- However, the predictive performance of the simple linear model can be improved, by replacing ordinary least squares fitting with some alternative fitting procedures.

Why alternatives to least squares?

- Prediction Accuracy: especially when $p > n$, to control the variance.
- Model Interpretability: By removing irrelevant features – that is, by setting the corresponding coefficient estimates to zero – we can obtain a model that is more easily interpreted.
- Some approaches for automatically performing feature selection.

Three classes of methods

- Subset Selection. We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables.
- Shrinkage. We fit a model involving all p predictors, but the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as regularization) has the effect of reducing variance and can also perform variable selection.
- Dimension Reduction. We project the p predictors into a M-dimensional subspace, where $M < p$. This is achieved by computing M different linear combinations, or projections, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.

Running Example - Predicting Credit Usage

```r
library(ISLR)
data("Credit")
ols_fit <- lm(Balance ~ Income + Rating + Cards + Age +
                  Education + Gender + Student + Married + Ethnicity + Limit, data = Credit)
summary(ols_fit)
```

```
##
## Call:
## lm(formula = Balance ~ Income + Rating + Cards + Age + Education +
##     Gender + Student + Married + Ethnicity + Limit, data = Credit)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -161.64  -77.70  -13.49   53.98  318.20
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -479.20787   35.77394 -13.395  < 2e-16 ***
## Income               -7.80310    0.23423 -33.314  < 2e-16 ***
## Rating                1.13653    0.49089   2.315   0.0211 *
## Cards                17.72448    4.34103   4.083 5.40e-05 ***
## Age                  -0.61391    0.29399  -2.088   0.0374 *
## Education            -1.09886    1.59795  -0.688   0.4921
## GenderFemale        -10.65325    9.91400  -1.075   0.2832
## StudentYes          425.74736   16.72258  25.459  < 2e-16 ***
## MarriedYes           -8.53390   10.36287  -0.824   0.4107
## EthnicityAsian       16.80418   14.11906   1.190   0.2347
## EthnicityCaucasian   10.10703   12.20992   0.828   0.4083
## Limit                 0.19091    0.03278   5.824 1.21e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 98.79 on 388 degrees of freedom
## Multiple R-squared:  0.9551, Adjusted R-squared:  0.9538
## F-statistic: 750.3 on 11 and 388 DF,  p-value: < 2.2e-16
```

Running Example - Predicting Credit Usage

How well does this model perform?

```
library(boot)
set.seed(221186)
# Estimate the model (using glm to work with cv.glm)
glm_fit <- glm(Balance ~ ., data = Credit[,-1])
# Conduct 5-fold cross validation and extract the CV MSE
cv_mse <- cv.glm(Credit, glm_fit, K = 5)$delta[1]
cv_mse
```

```
## [1] 10085.35
```

Shrinkage Methods

Ridge regression and Lasso

- We can alternatively 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 (and, thus, the variance of the predictions we make).

Ridge regression

- Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, \dots, \beta_p$ using the values that minimize

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

- In contrast, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

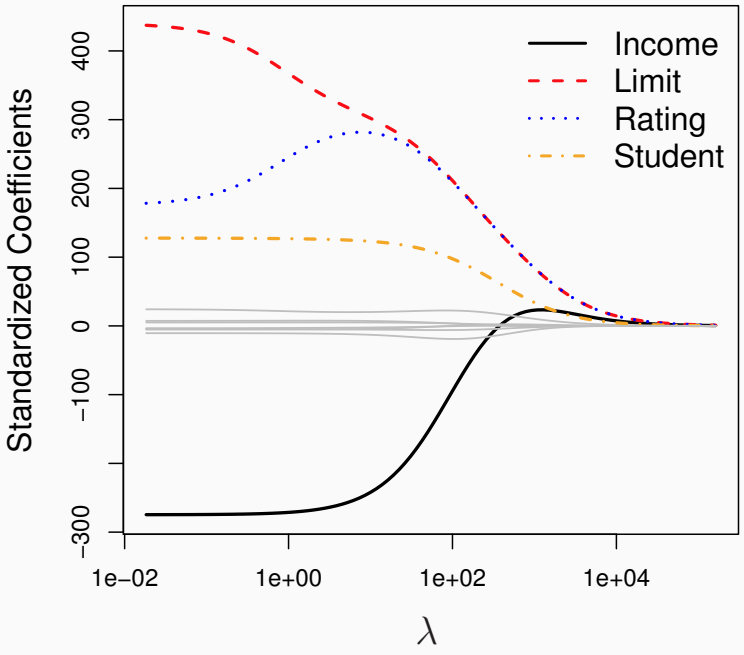
$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda \sum_{j=1}^{p} \beta_j^2,$$

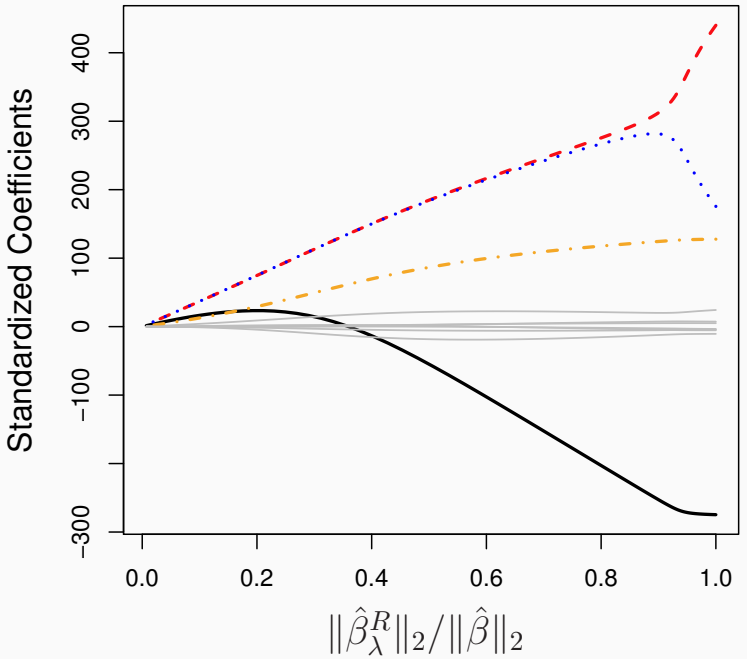
where $\lambda \geq 0$ is a tuning parameter, to be determined separately.

Ridge regression: continued

- As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small.
- However, the second term, $\lambda \sum_{j=1} \beta_j^2$, called a shrinkage penalty, is small when $\beta_1, \dots, \beta_p$ are close to zero, and so it has the effect of shrinking the estimates of β_j towards zero.
- The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates.
- Selecting a good value for λ is therefore critical - how should we pick this?
    - Cross-validation, of course!

Example: Credit data





Details of Previous Figure

- In the left-hand panel, each curve corresponds to the ridge regression coefficient estimate for one of the ten variables, plotted as a function of λ.
- The right-hand panel displays the same ridge coefficient estimates as the left-hand panel, but instead of displaying λ on the x-axis, we now display $\|\hat{\beta}^R_\lambda\|_2/\|\hat{\beta}\|_2$, where $\hat{\beta}$ denotes the vector of least squares coefficient estimates.
- The notation $\|\beta\|_2$ denotes the ℓ_2 norm (pronounced "ell 2") of a vector, and is defined as $\|\hat{\beta}\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$.

Ridge regression: scaling of predictors

- The standard least squares coefficient estimates are scale equivariant: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same.
- In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function.
- Therefore, it is best to apply ridge regression after standardizing the predictors by dividing by the standard deviation:

$$\tilde{x}_{ij} = \frac{x_{ij}}{sd(x_{ij})}$$

Why Does Ridge Regression Improve Over Least Squares?

Reminder: The Bias-Variance tradeoff

- The *variance* of an estimator refers to the amount that our estimator, and thus our predictions, would change if we estimated it using a different dataset
    - The variance will typically increase when we *overfit* a model, as we will find that the model parameters will have very different estimated values from one sample to another
- The *bias* of an estimator refers to the error that occurs from using a model that is too simplistic relative to the real data generating process in the world
    - The bias will typically increase when we *underfit* a model, as we will find that we are unable to capture important features of the data because our model is too inflexible

Why Does Ridge Regression Improve Over Least Squares?

The Bias-Variance tradeoff

- Ridge regression (and other shrinkage methods) implicitly try to strike a balance between over-fitting and under-fitting
- They trade off a little bias (making our models simpler) and in so doing they reduce the variance (because the simpler models will be less variable when applied to new data)
- The goal is to try and hit a sweet spot in which we reduce the variance by a lot, and increase the bias by only a bit

The Lasso

- Ridge regression shrinks coefficients *towards* zero, but never exactly to zero. Ridge regression will therefore include all p predictors in the final model.
- The Lasso regression that overcomes this disadvantage. The lasso coefficients, $\hat{\beta}_\lambda^L$, minimize the quantity

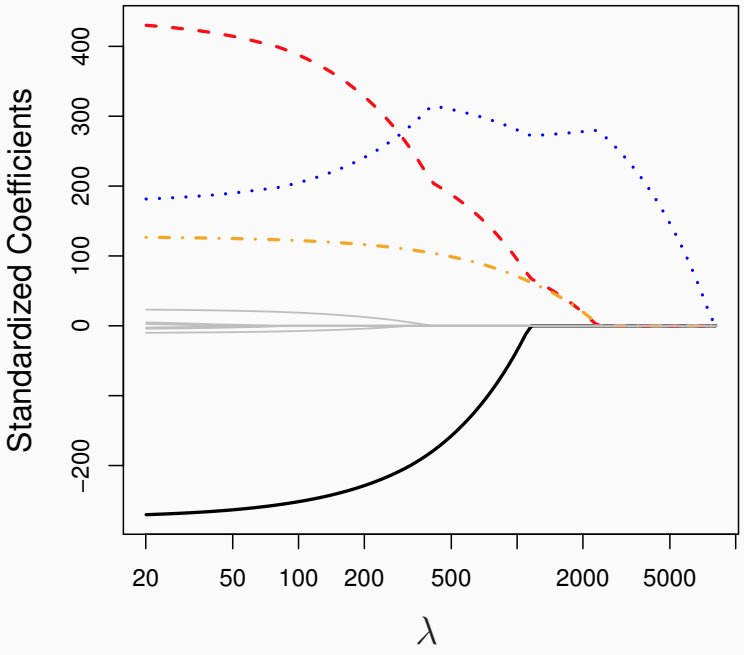
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

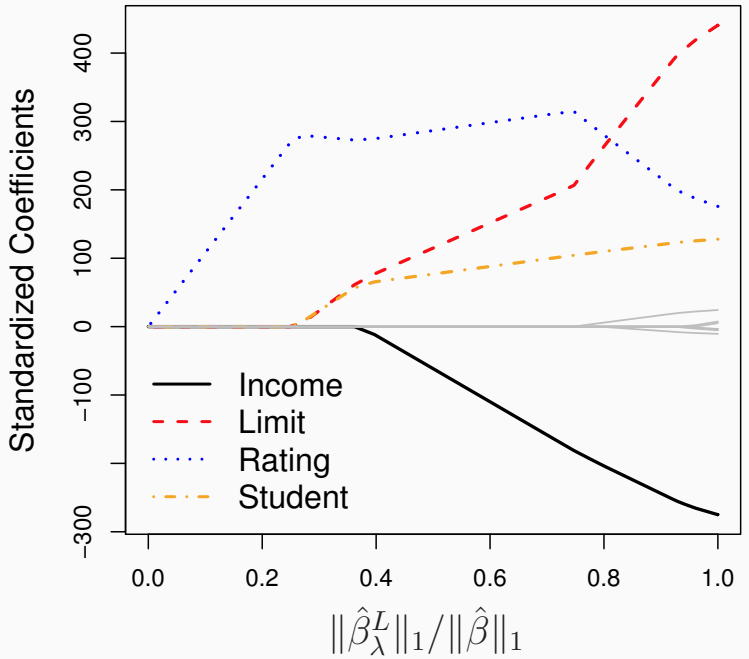
- In statistical parlance, the lasso uses an ℓ_1 (pronounced "ell 1") penalty instead of an ℓ_2 penalty. The ℓ_1 norm of a coefficient vector β is given by $\|\beta\|_1 = \sum|\beta_j|$.

The Lasso: continued

- As with ridge regression, the lasso shrinks the coefficient estimates towards zero.
- However, in the case of the lasso, the ℓ_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large.
- The lasso performs variable selection.
- We say that the lasso yields sparse models – that is, models that involve only a subset of the variables.
- As in ridge regression, selecting a good value of λ for the lasso is critical; cross-validation is again the method of choice.

Example: Credit data





Comparing the Lasso and Ridge Regression

- Neither ridge regression nor the lasso will universally dominate the other.
- In general, one might expect the lasso to perform better when the response is a function of only a relatively small number of predictors.
- However, the number of predictors that is related to the response is never known *a priori* for real data sets.
- Cross-validation can also be used in order to determine which approach is better on a particular data set.

Selecting the Tuning Parameter for Ridge Regression and Lasso

- We require a method selecting a value for the tuning parameter λ or equivalently, the value of the constraint s.
- Cross-validation provides a simple way to tackle this problem. We choose a grid of λ values, and compute the cross-validation error rate for each value of λ.
- We then select the tuning parameter value for which the cross-validation error is smallest.
- Finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter.

Example

- Let's say you are interested in predicting US presidential elections.
- In particular, you want to predict the vote share of the incumbent party in each election.
- You collect 13 variables, including variables on incumbency, economic performance, presidential approval.
- But these variables are only available starting with the 1952 election...
- ...so there are only 16 observations.
- What, if anything, can we learn from these data?

Why not linear regression?

- What happens to linear regression in this context?
- If we fit a model with all 13 variables, we get the following...
- Note: I have standardized all the variables (subtract the mean, divide by the standard deviation). This does not change the fit of the model, it just makes the coefficients more comparable across models.

Linear regression (n = 16, k = 13)}

```
Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                -2.647e-17  7.573e-02   0.000    1.000
HibbsFatalities             1.067e-01  2.873e-01   0.371    0.746
Year4DisposibleIncomeGrowth 3.797e-01  2.334e-01   1.627    0.245
Year3DisposibleIncomeGrowth -3.876e-01  4.138e-01  -0.937    0.448
Year2DisposibleIncomeGrowth -6.782e-02  3.757e-01  -0.181    0.873
Year1DisposibleIncomeGrowth 4.263e-02  1.969e-01   0.217    0.849
IncumbentParty             -3.737e-01  3.326e-01  -1.124    0.378
Q2GDP                       2.417e-01  2.027e-01   1.192    0.355
NetApproval                 5.942e-01  2.901e-01   2.048    0.177
TwoTerms                   -6.241e-01  6.357e-01  -0.982    0.430
PreviousTerms              -1.396e-01  4.027e-01  -0.347    0.762
PresidentRunning           -3.266e-01  5.063e-01  -0.645    0.585
PreviousWinnerRunning      -1.127e-02  2.284e-01  -0.049    0.965
UnemploymentChange         -2.470e-02  1.680e-01  -0.147    0.897

Residual standard error: 0.3029 on 2 degrees of freedom
Multiple R-squared: 0.9878,Adjusted R-squared: 0.9082
F-statistic: 12.42 on 13 and 2 DF,  p-value: 0.0769
```

Mixed signals

- Mixed diagnostics:
  - Nothing is significant.
  - F-test of all explanatory variables has $p = 0.08$.
  - But adjusted R^2 is still 0.91!
- So, is this model any good?
- Some of these diagnostics are breaking because their assumptions rely on having $k \ll n$.

Assessing linear regression

- Given that we get mixed signals from our standard diagnostics, how might we better assess the performance of this linear regression?
- Cross-validation!
    - It turns out that the LOOCV MSE is 1.6 for this model
- How does that compare to the MSE for LASSO and Ridge estimates?

Comparison of Cross-Validation Scores

- In this application, ridge regression produces the best model by leave-one-out-cross-validation.
- The OLS model performs even worse than the intercept-only model!
- Simulation studies suggest ridge regression works better with highly collinear explanatory variables, lasso regression when explanatory variables are not highly collinear.

Model	LOOCV MSE
Intercept Only	1.07
OLS	1.6
Lasso	0.215
Ridge	0.128

Regression Coefficient Comparison

- The ridge regression coefficients are generally similar to (but smaller than) the original linear regression coefficients we started with...
- ...yet the model performs *much* better in cross-validation.

	All Vars	Lasso	Ridge
NetApproval	0.5942	0.3699	0.3291
Year4DispIncGrowth	0.3797	0.3278	0.3345
Year3DispIncGrowth	-0.3876	0.0028	0.0509
Year2DispIncGrowth	-0.0678		0.0978
Year1DispIncGrowth	0.0426		-0.0221
PreviousTerms	-0.1396	-0.1600	-0.2006
IncumbentParty	-0.3737		-0.0832
Q2GDP	0.2417	0.0613	0.1580
TwoTerms	-0.6241	-0.1179	-0.2384
PresidentRunning	-0.3266		-0.0399
PrevWinnerRunning	-0.0113		-0.0191
UnemplChange	-0.0247		-0.0258
HibbsFatalities	0.1067		-0.0764

Summary

- Model selection methods are an essential tool for data analysis, especially for big datasets involving many predictors.
- Research into methods that give sparsity, such as the lasso is an especially hot area.
- The methods that we have discussed today have involved fitting linear regression models, via least squares or a shrunken approach, using the original predictors, $X_1, X_2, \ldots, X_p$.
- Tomorrow we will explore a class of approaches that transform the predictors and then fit a least squares model using the transformed variables.