Day 6: Nonlinear Models and Tree-Based Methods

ME314: Introduction to Data Science and Machine Learning

Jack Blumenau

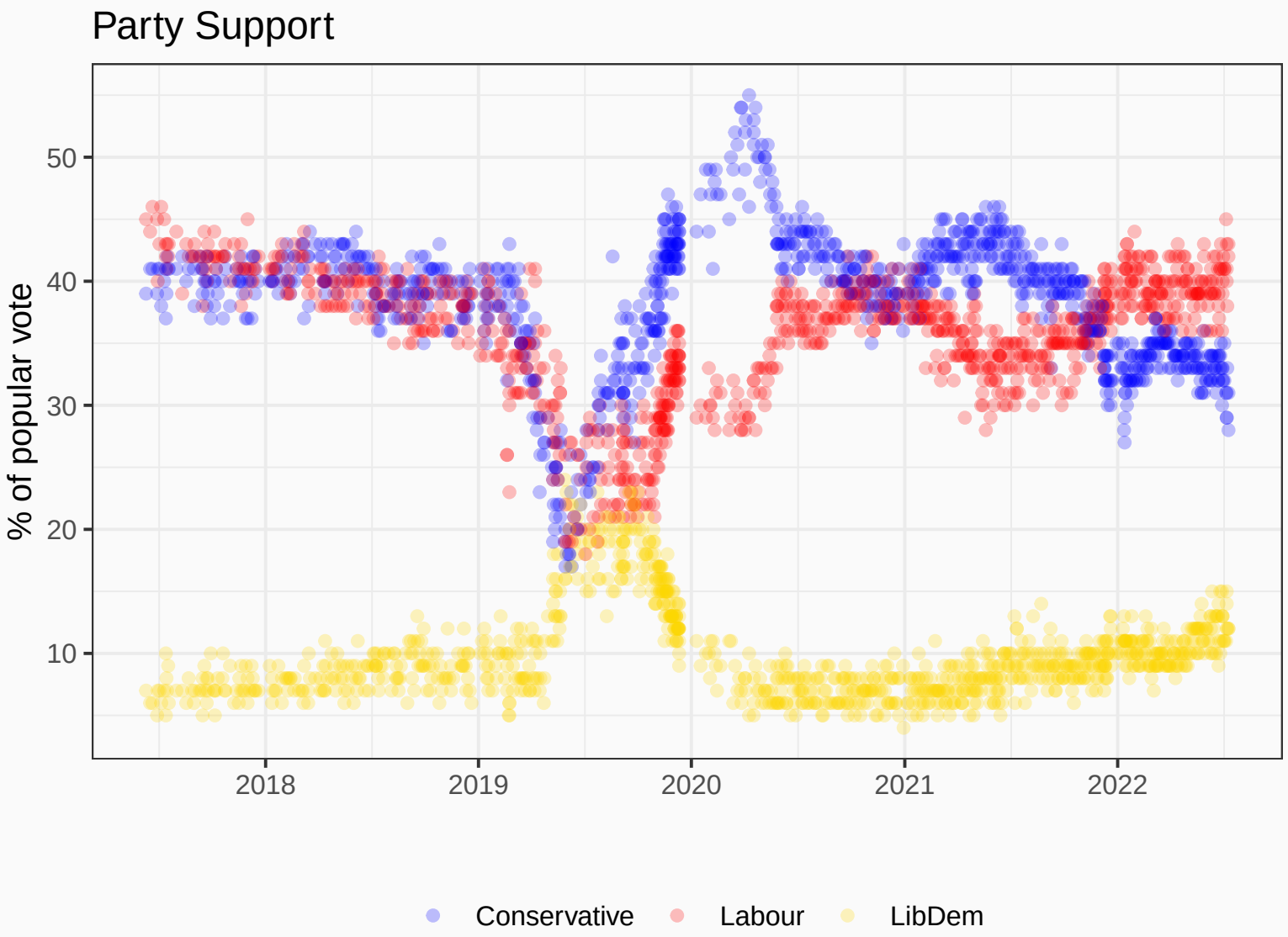
19th July 2022

Motivation

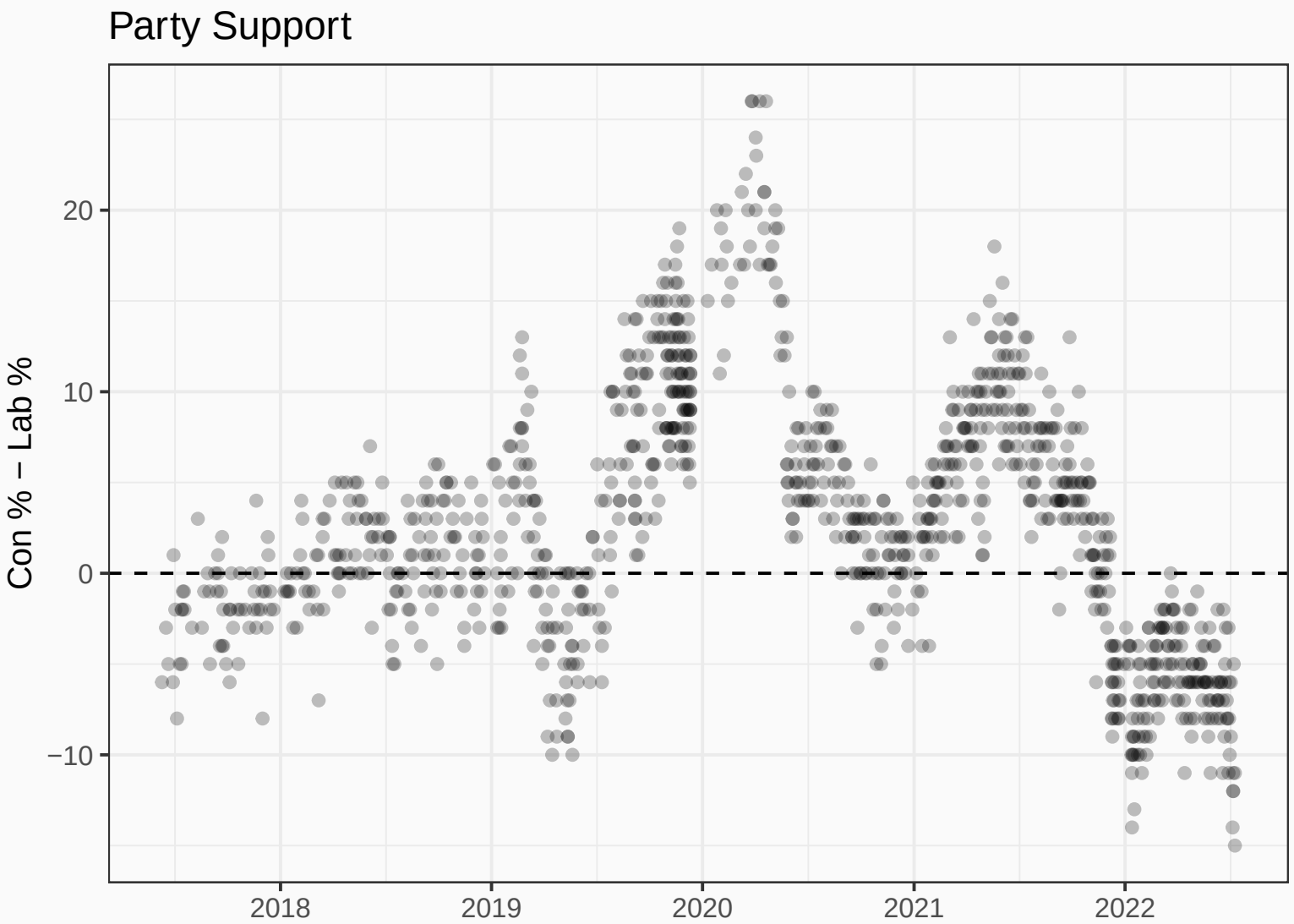
How popular are the UK political parties?

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



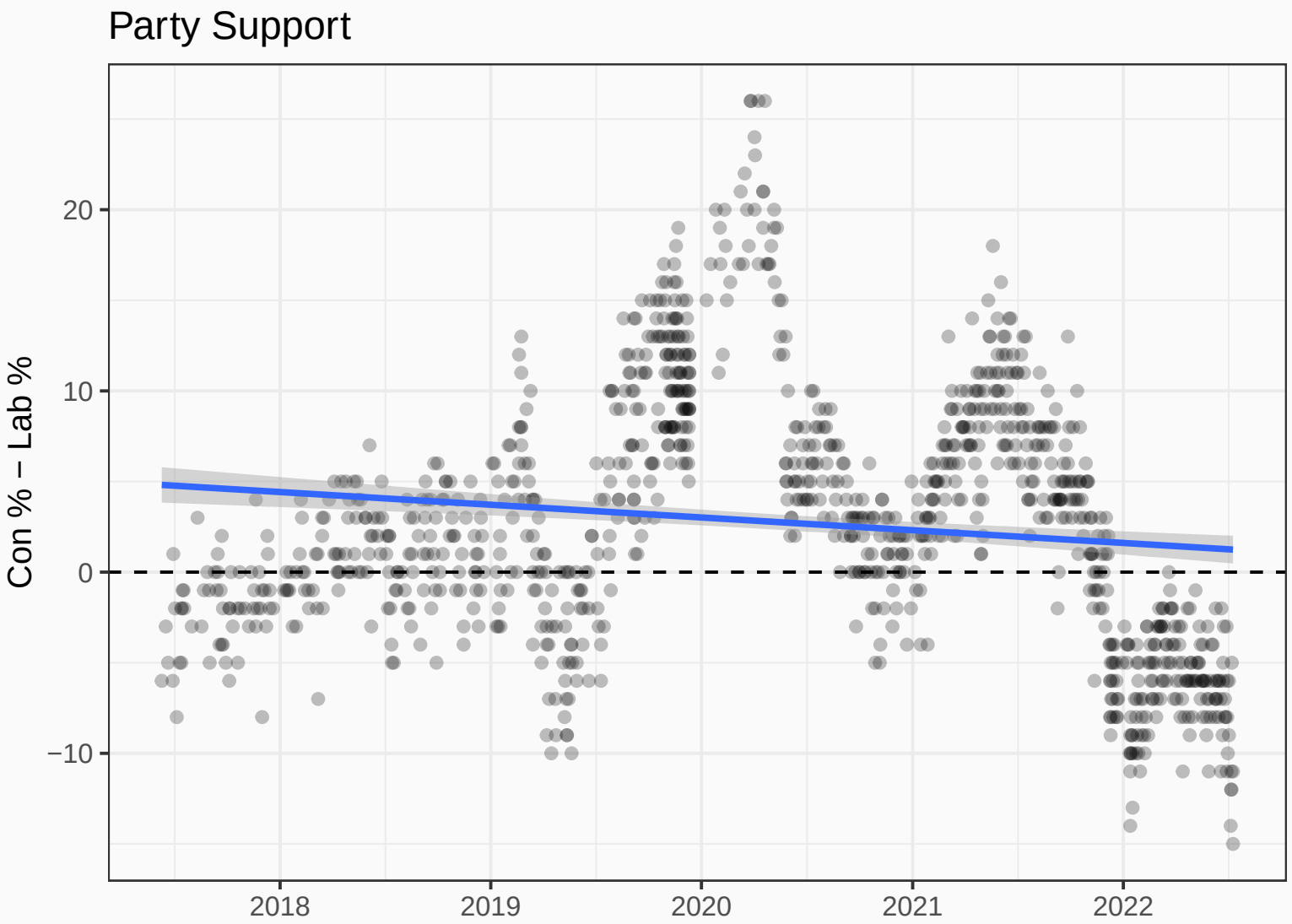
Motivation

- We are interested in describing how the Conservative lead varies over time.
- What is an appropriate model for this data?
- We could, of course, use a linear regression:

```
lin_mod <- lm(con_lead ~ date, data = polls)
summary(lin_mod)
```

```
...
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 38.1294918  7.6819754   4.964 8.08e-07 ***
## date        -0.0019228  0.0004173  -4.608 4.56e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
...
```

Motivation



This is a dumb model for this data.

Day 6 Outline

Moving Beyond Linearity

Tree-based Methods

Bagging

Random Forests

Moving Beyond Linearity

Motivation

- Traditional models – like linear and logistic regression – can be good for evaluating theories that imply specific functional forms for the relationship between outcomes and predictors
- For many social science domains, the theories we have are not sufficiently detailed to account for the complexity of the data we have to hand
- We therefore need a set of methods which allow us to *learn* complicated non-linearities and interactions from the data
- Today's lecture introduces some of those tools

Non-linear models

1. Polynomial regression
2. Step functions
3. Splines
4. Local regression
5. Generalised additive models

Basis functions

- Many of the approaches we discuss today are special cases of what are known as “basis function” approaches.
- They share the idea that, in order to capture non-linear relationships between predictors and outcome, we can transform X in some way and then just use a linear model
- Where $b(x)$ is a function applied to our predictor, we have:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + ... + \beta_K b_K(x_i)$$

- This is just a linear model on the transformed predictors!
- Advantage: inference tools – such as standard errors, confidence intervals, hypothesis tests, coefficient estimates, F-tests, etc – all apply as before

Polynomial regression

Polynomial regression models

Polynomial regression models take the following form:

$$
\begin{aligned}
\text{Linear:} \quad & y_i = \alpha + \beta_1 x_{i1} + \epsilon_i \\
\text{Quadratic:} \quad & y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i1}^2 + \epsilon_i \\
\text{Cubic:} \quad & y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i1}^2 + \beta_3 x_{i1}^3 + \epsilon_i \\
\text{General:} \quad & y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i1}^2 + ... + \beta_d x_{i1}^d + \epsilon_i
\end{aligned}
$$

Where x_{i1}^2 is just $x_{i1} \cdot x_{i1}$ and x_{i1}^3 is just $x_{i1} \cdot x_{i1} \cdot x_{i1}$, and so on (i.e. the basis function for the quadratic is $b(x_i) = x_i^2$).

The more polynomial terms we add, the more flexible we are allowing the relationship between our outcome and our predictor to be.

Polynomial regression

Why do polynomial terms allow for non-linear relationships?

- When we include a quadratic term in the model, we are essentially including an interaction term
    - i.e. the interaction between X_1 and itself (because $X_1 \cdot X_1 = X_1^2$)
- This implies that the association between X_1 and Y will depend on the specific value of X_1 where we evaluate the relationship
- $\rightarrow$ the effect of a one-unit change in X_1 will depend on the value of X_1 we are changing

Polynomial interpretation

Interpreting polynomial coefficients is somewhat difficult:

- It is no longer possible to hold constant all other variables
    - i.e. If you increase X_1 by one-unit, then you also increase X_1^2
- We can say something by looking at the signs of β_{X^2}, β_{X^3}, etc but inferring substantive meaning is often difficult
- We can interpret the significance of the squared term: the null hypothesis is that there is a linear relationship between X and Y. If $p < 0.05$, we can reject the null of linearity.

Polynomial interpretation

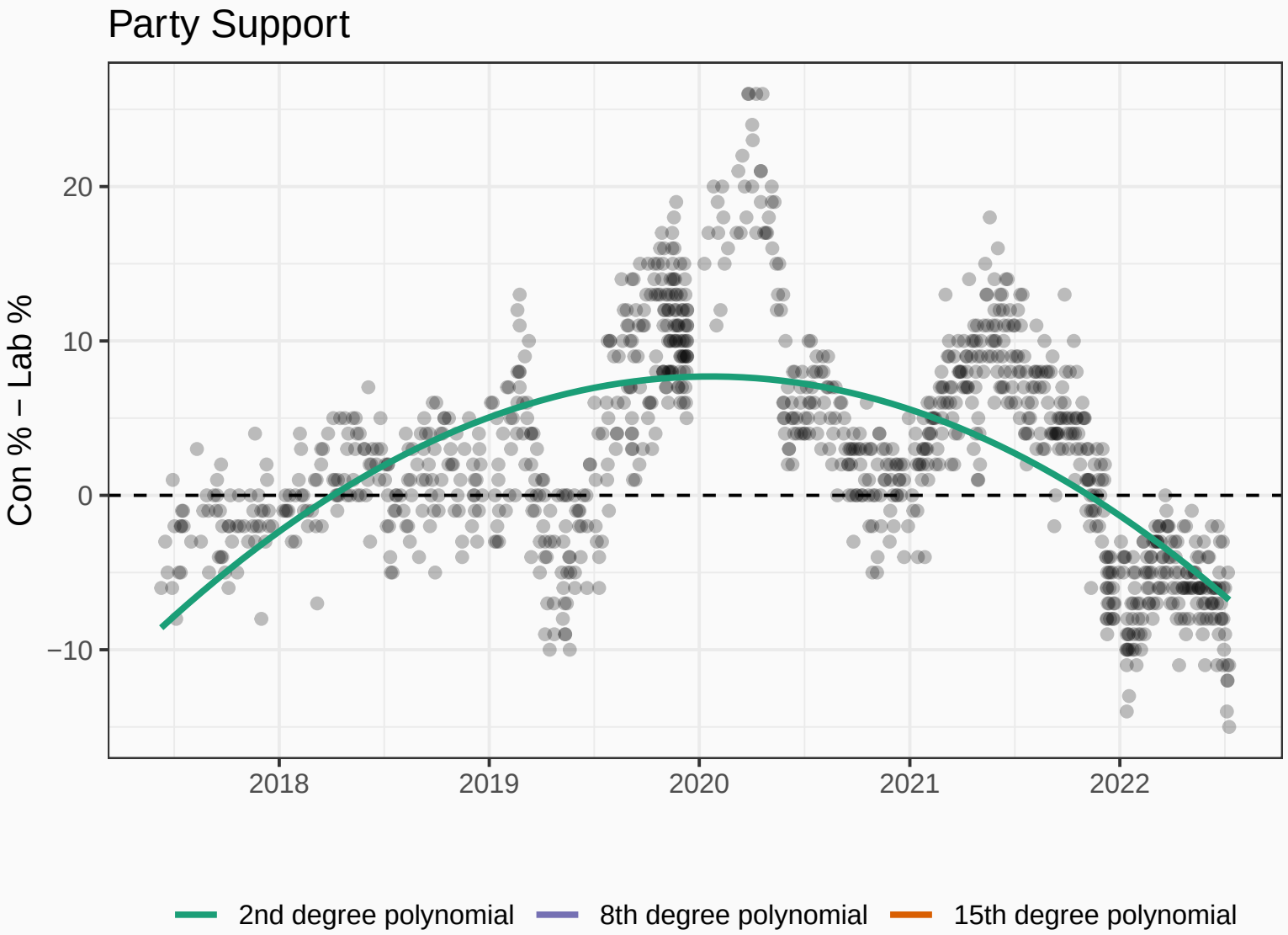
- In general it is much more straightforward to produce fitted value plots to describe the relationship between X and Y
- This message applies to all the non-linear methods today: don't show tables, plot graphs!

Polynomial regression – application

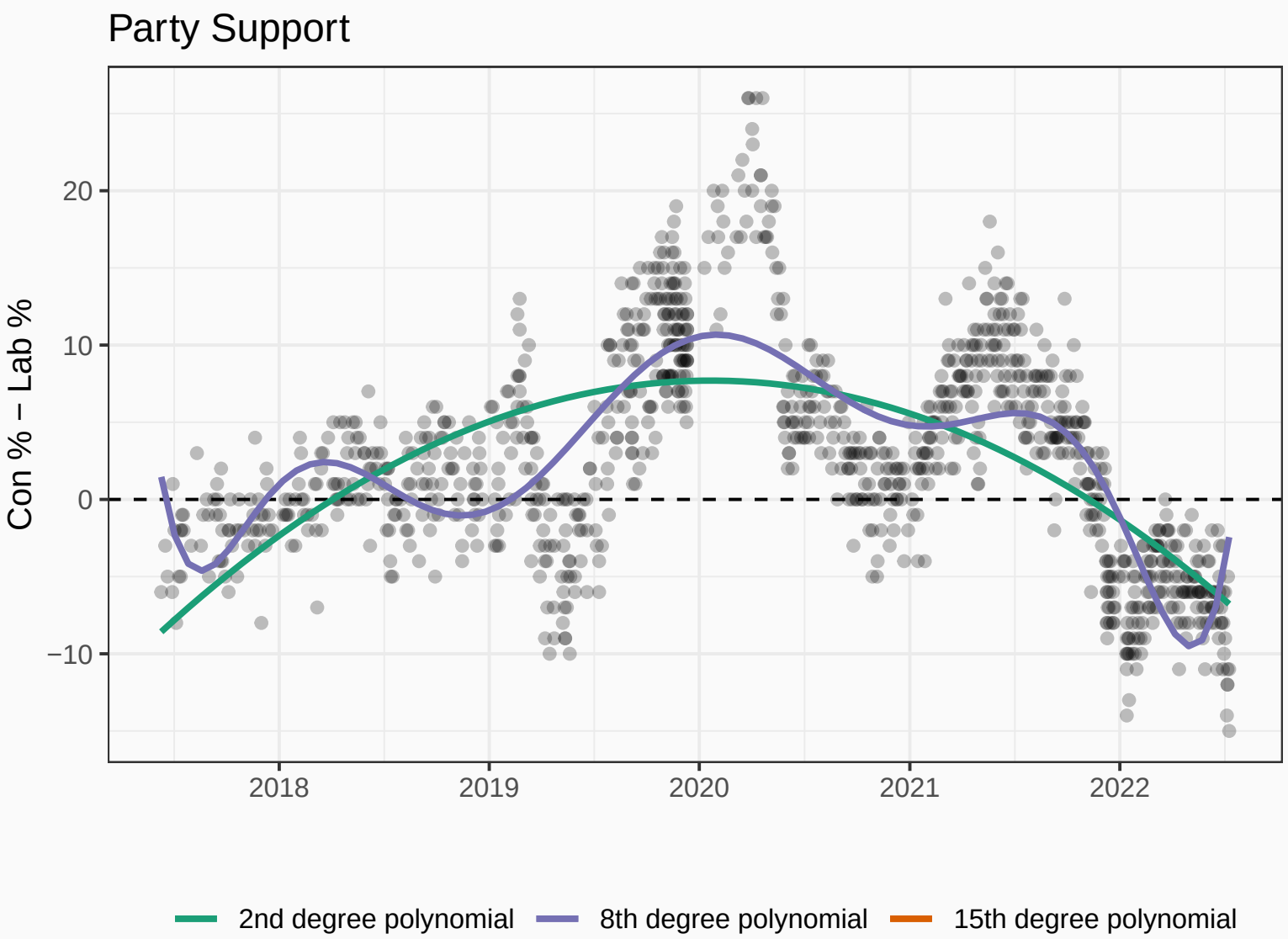
```r
poly_mod_2 <- lm(con_lead ~ poly(as.numeric(date), 2), data = polls)
poly_mod_8 <- lm(con_lead ~ poly(as.numeric(date), 8), data = polls)
poly_mod_15 <- lm(con_lead ~ poly(as.numeric(date), 15), data = polls)

summary(poly_mod_2)
```

```
...
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    2.7431     0.1658   16.55  < 2e-16 ***
## poly(as.numeric(date), 2)1   -31.9124     5.3638   -5.95 3.66e-09 ***
## poly(as.numeric(date), 2)2 -141.7014     5.3638  -26.42  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
```

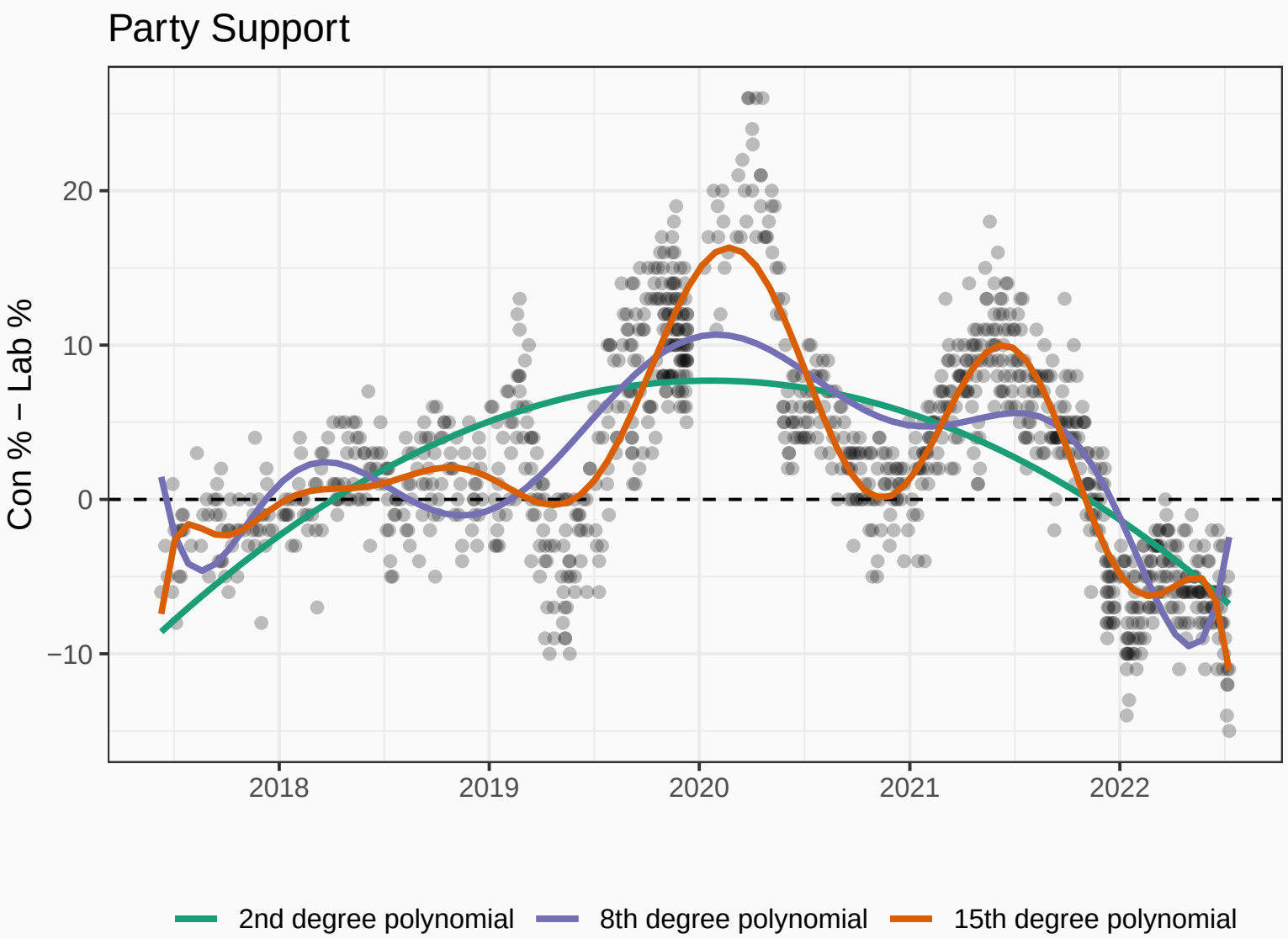
Polynomial regression – application



Polynomial regression – application



Polynomial regression – application



Polynomial regression

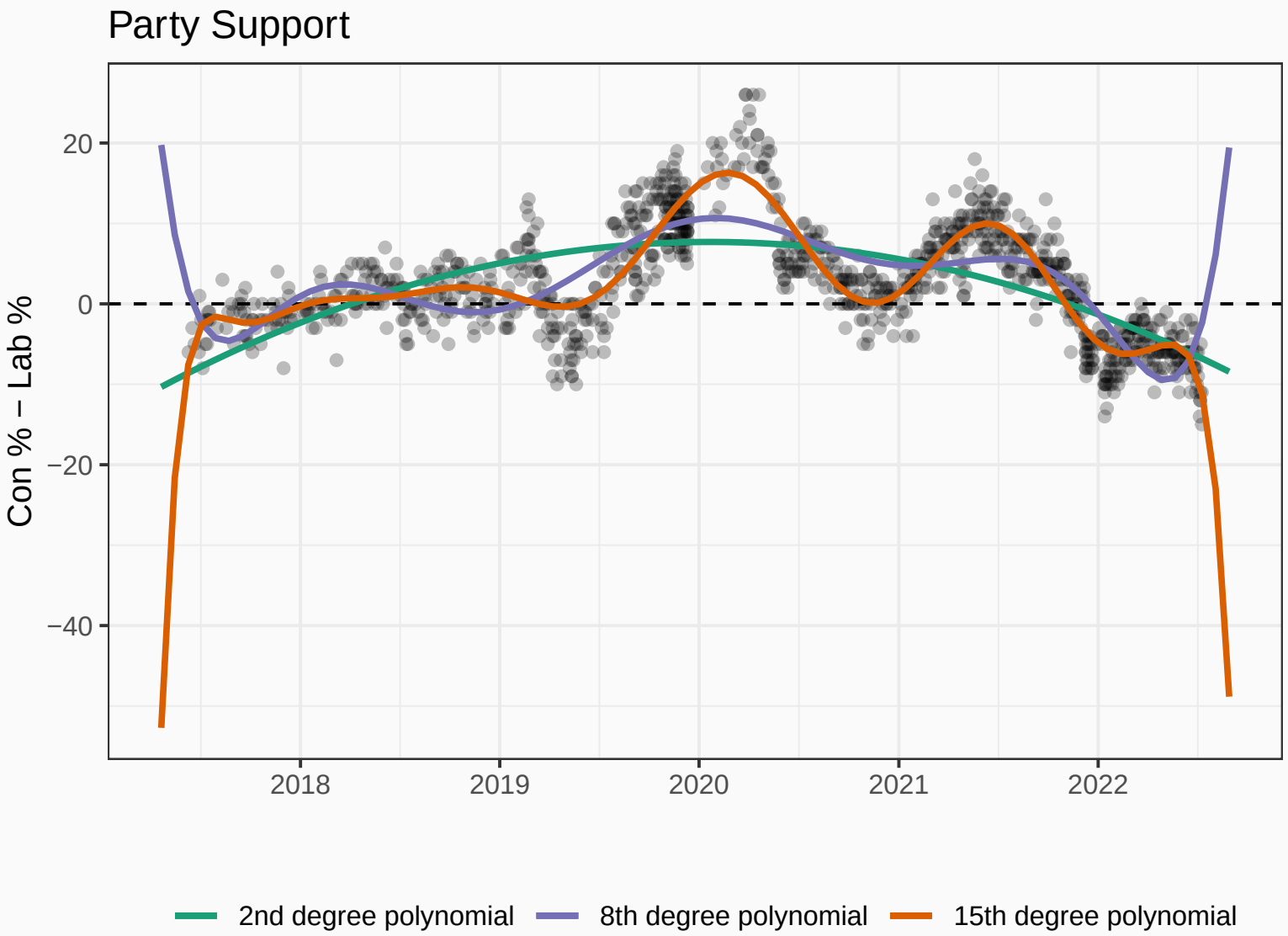
Advantages:

- Very easy to implement: e.g. `poly(x, degree = 3)`
- Can be used in any regression framework (for instance, logit)

Disadvantages:

- Can have very poor behaviour in the tails, which makes it a poor tool for extrapolation beyond the domain of X
- For example, let's extend the x-asis of our party support date by 50 days in either direction...

Polynomial regression – extrapolation



Step Functions

Step function regression models

Step function, or piecewise-constant, models take the following form:

$$y_i = \alpha + \beta_1 C_1(x_{i1}) + \beta_2 C_2(x_{i1}) + \ldots + \beta_d C_d(x_{i1}) \epsilon_i$$

Where we transform x into a set of dummy variables by defining a set of cutpoints, $c_1, c_2, \ldots, c_k$ across the range of x:

$$\begin{aligned} C_0(x) &= I(X < c_1) \\ C_1(x) &= I(c_1 \leq x < c_2) \\ C_2(x) &= I(c_2 \leq x < c_3) \\ &\ldots \\ C_K(x) &= I(c_k \leq x) \end{aligned}$$

The more cutpoints we add, the more flexible we are allowing the relationship between our outcome and our predictor to be.

Step functions – application

We can use the `cut()` function to cut a variable into equal-length intervals:

```
table(cut(polls$date, 6))
```

```
##
## 2022-07-10 2021-09-03 2020-10-29 2019-12-11 2019-02-18 2018-04-14
##         95        128        234        134        203        253
```

We can then include these in the model (as a set of dummy variables):

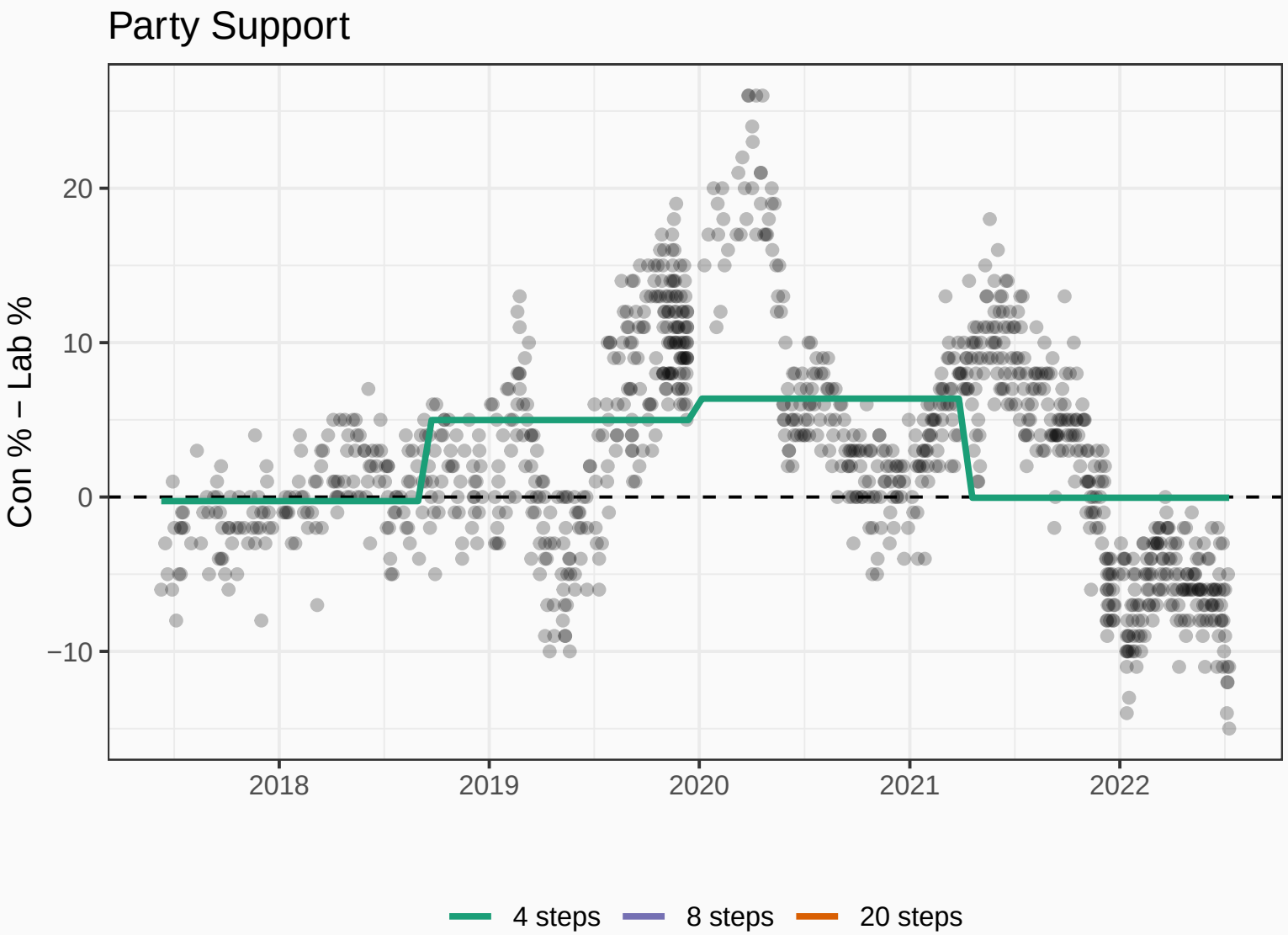
```
step_mod_4 <- lm(con_lead ~ cut(date, 4), data = polls)
step_mod_8 <- lm(con_lead ~ cut(date, 8), data = polls)
step_mod_20 <- lm(con_lead ~ cut(date, 20), data = polls)
```

Step functions – application

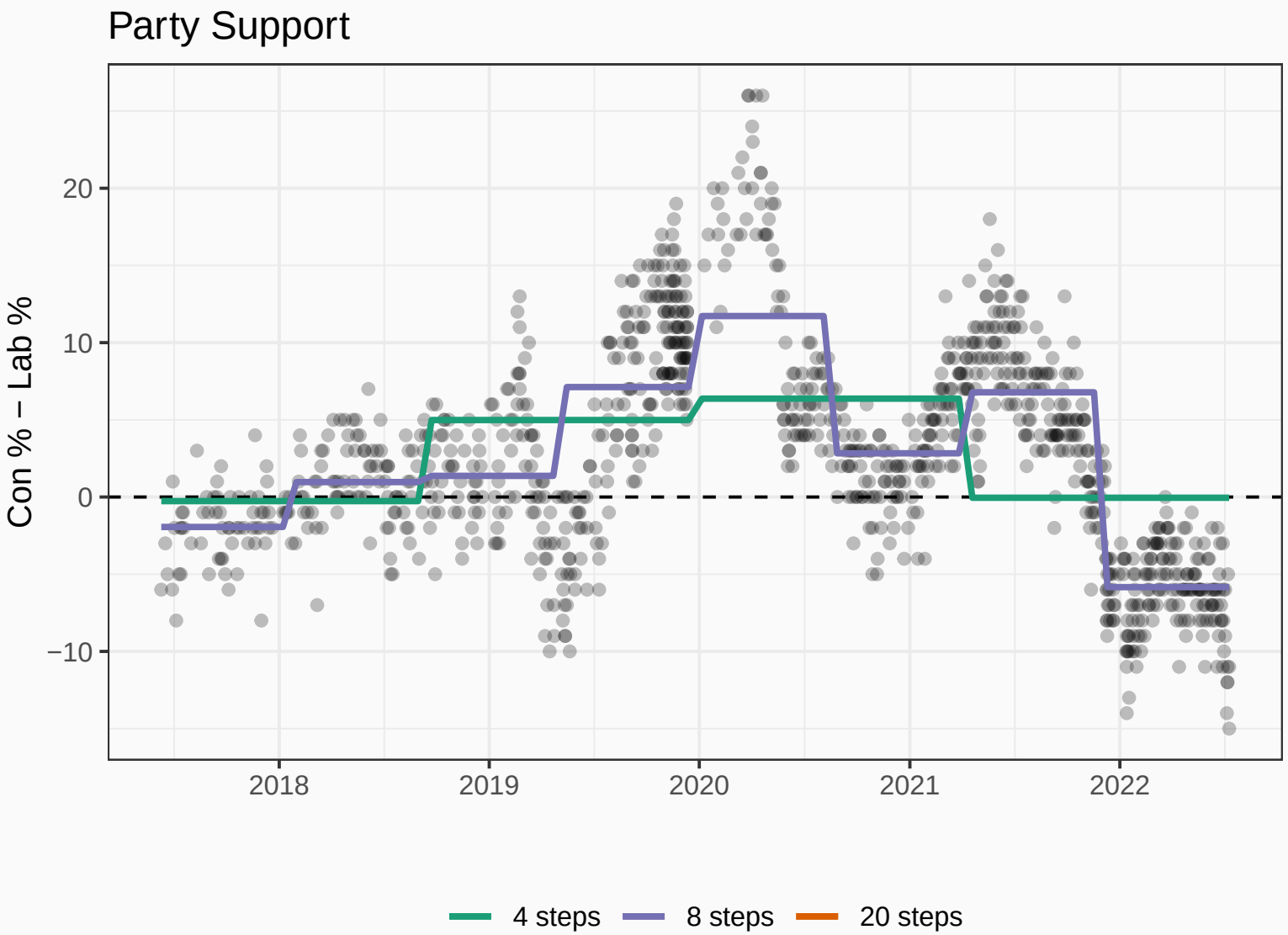
```
summary(step_mod_4)
```

```
...
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.2658     0.5072  -0.524    0.600
## cut(date, 4)2021-04-01    5.2524     0.6270   8.377   <2e-16 ***
## cut(date, 4)2019-12-11    6.6375     0.6611  10.040   <2e-16 ***
## cut(date, 4)2018-09-13    0.2191     0.6074   0.361    0.718
## ---
...
```

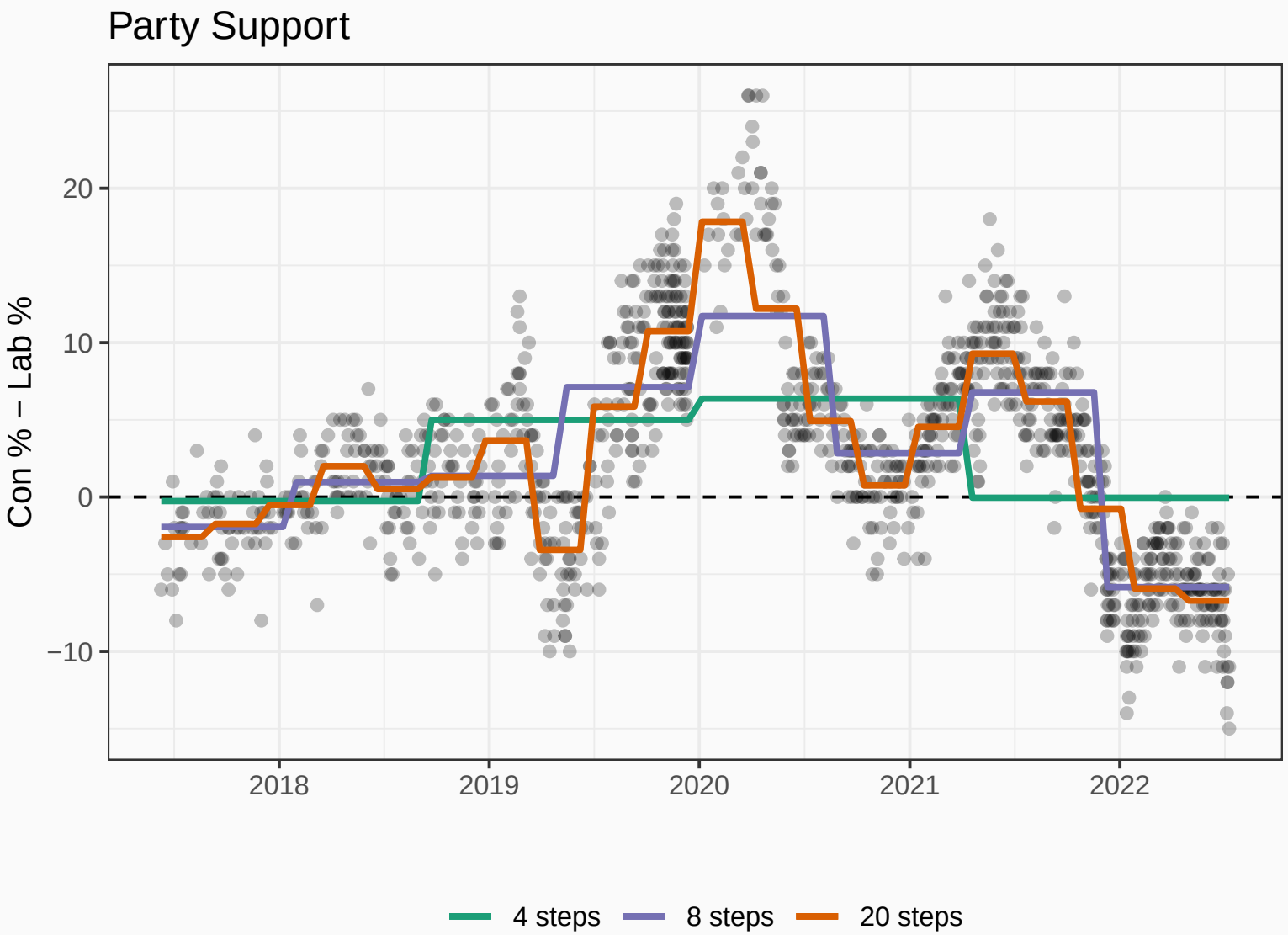
Step functions – application



Step functions – application



Step functions – application



Splines – Piecewise polynomials

- Polynomial models: specify a single polynomial across the domain of X
- Step function models: estimate a single mean parameter for different ranges of X
- Splines: combine both approaches by specifying separate polynomial models for different regions of X

Piecewise polynomial models

Regression splines generalise the step-function approach above by allowing for more flexible functional forms for different ranges of X:

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

where c is a "knot" which defines a point in X at which the coefficients change.

Splines – Piecewise polynomials

Piecewise polynomial models

Regression splines generalise the step-function approach above by allowing for more flexible functional forms for different ranges of X:

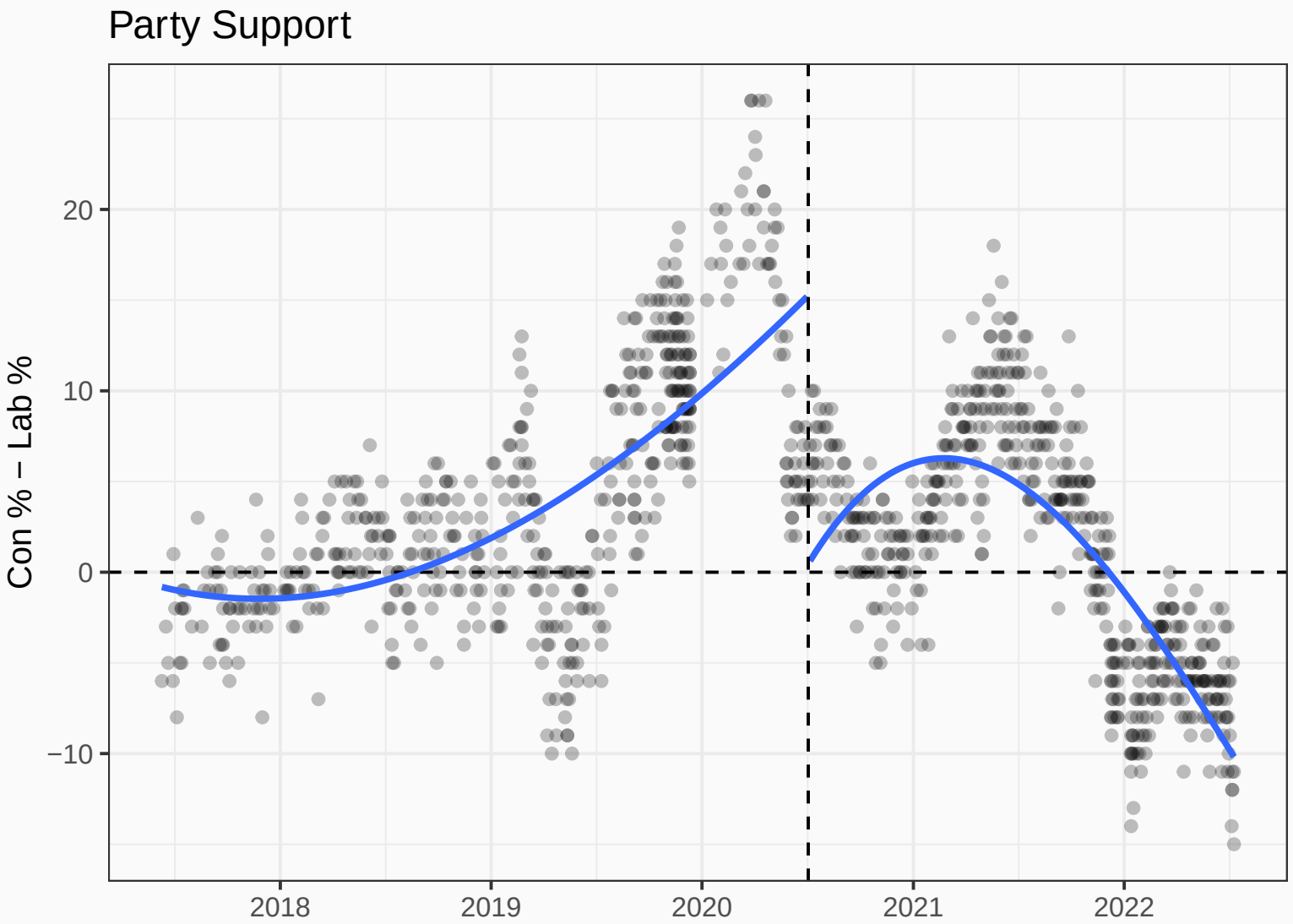
$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

where c is a "knot" which defines a point in X at which the coefficients change.

More cutpoints, and higher order polynomials, imply a more flexible relationship between our outcome and our predictor. E.g.

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c_1; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } c_1 \geq x_i < c_2 \\ \beta_{03} + \beta_{13} x_i + \beta_{23} x_i^2 + \beta_{33} x_i^3 + \epsilon_i & \text{if } x_i \geq c_2. \end{cases}$$

Splines – Piecewise polynomials



What is wrong here?

Splines – Piecewise polynomials

- Even though we would like to model separate polynomials on either side of the knot/cutpoint, we don't want a big discontinuity at that point
- Instead, we would like to constrain the estimates so that
  - There is no *disconuity* at the knots
  - The piecewise polynomials are *smooth* at the knots
- We can constrain the estimation to achieve these properties by using truncated basis functions of X

Cubic splines

We can represent this model with *truncated* power basis functions:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

Where the b_k are basis functions:

$$b_1(x_i) = x_i; b_2(x_i) = x_i^2; b_3(x_i) = x_i^3;$$
$$b_{k+3}(x_i) = (x_i - \xi_k)_+^3,\ k = 1, \ldots, K$$

where

$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3 & \text{if } x_i > \xi_k; \\ 0 & \text{otherwise.} \end{cases}$$

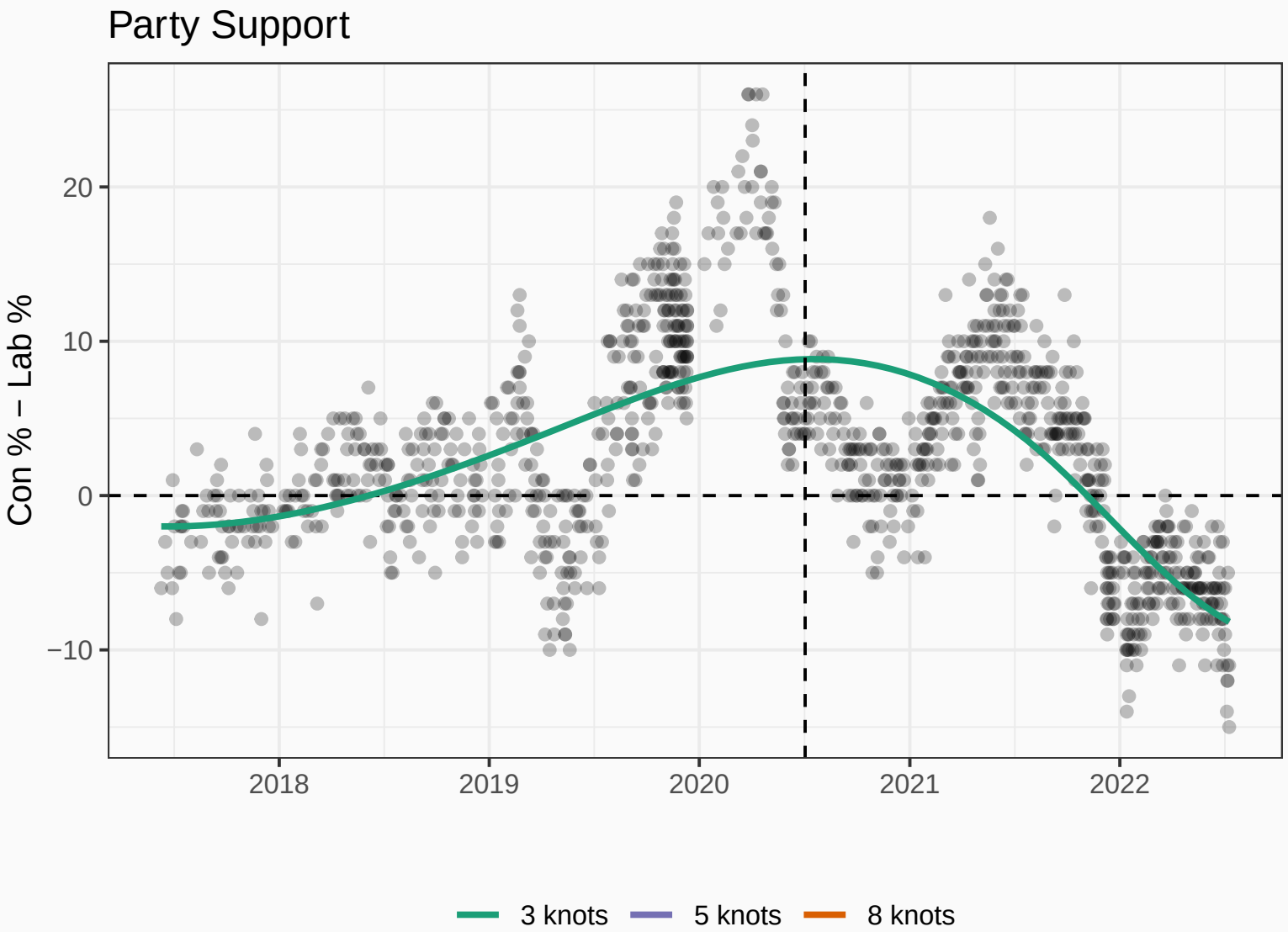
Splines – implementation

- As before, we can estimate splines by transforming x and adding the basis-spline transformation (`bs()`) to a linear regression

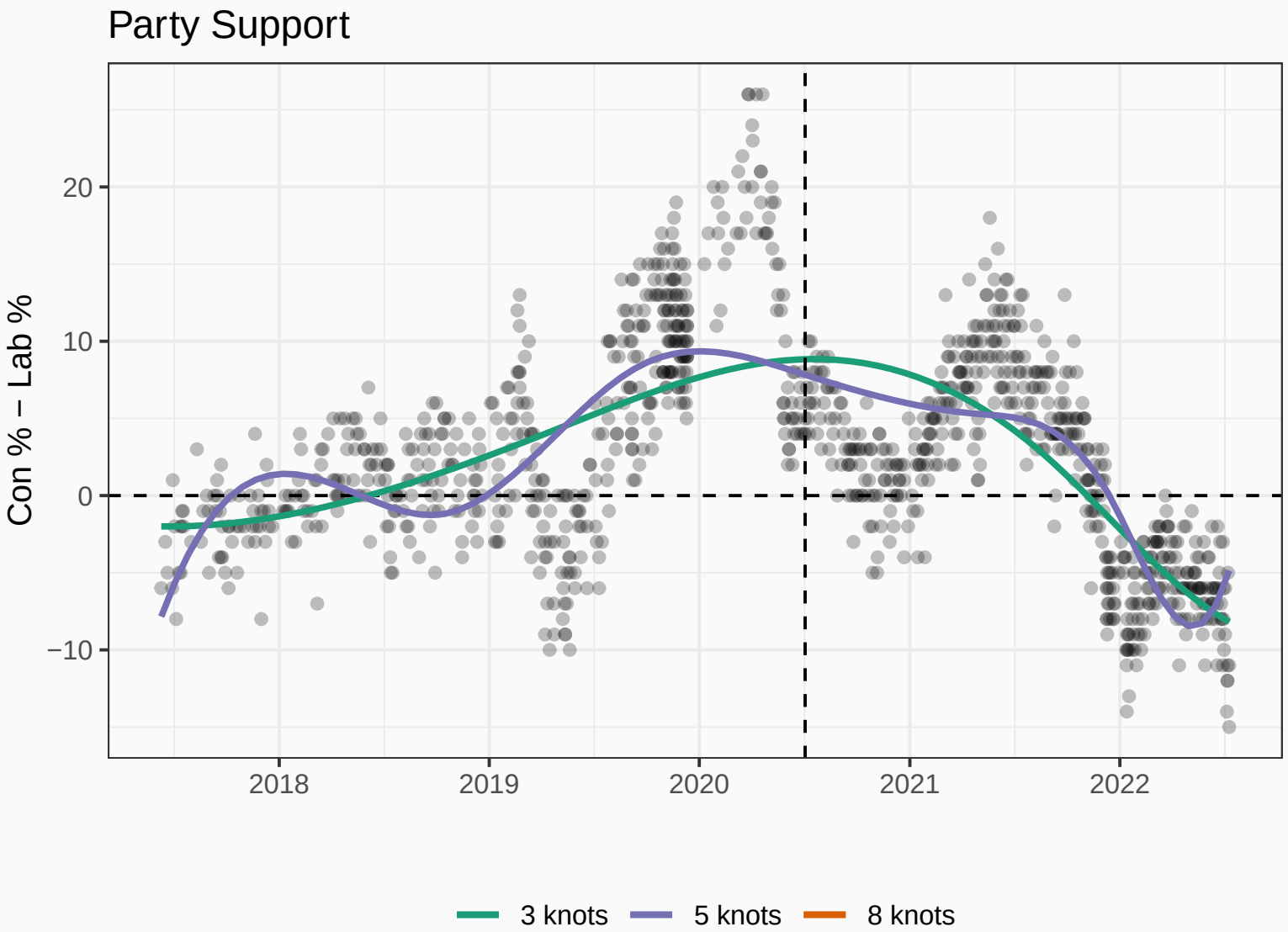
```
library(splines)
spline_mod_3 <- lm(con_lead ~ bs(date, df = 6, degree = 3), data = polls)
spline_mod_5 <- lm(con_lead ~ bs(date, df = 8, degree = 3), data = polls)
spline_mod_8 <- lm(con_lead ~ bs(date, df = 11, degree = 3), data = polls)
```

- Note that `df` is degrees of freedom, and is equal to number of knots + 3
- Note also here that `degree = 3` means we are fitting cubic splines, but we could fit more flexible relationships between knots
- We are *placing* the knots at uniform quantiles here, but we could also put more knots in regions of x which have more variability
- $\longrightarrow$ lots of choices!

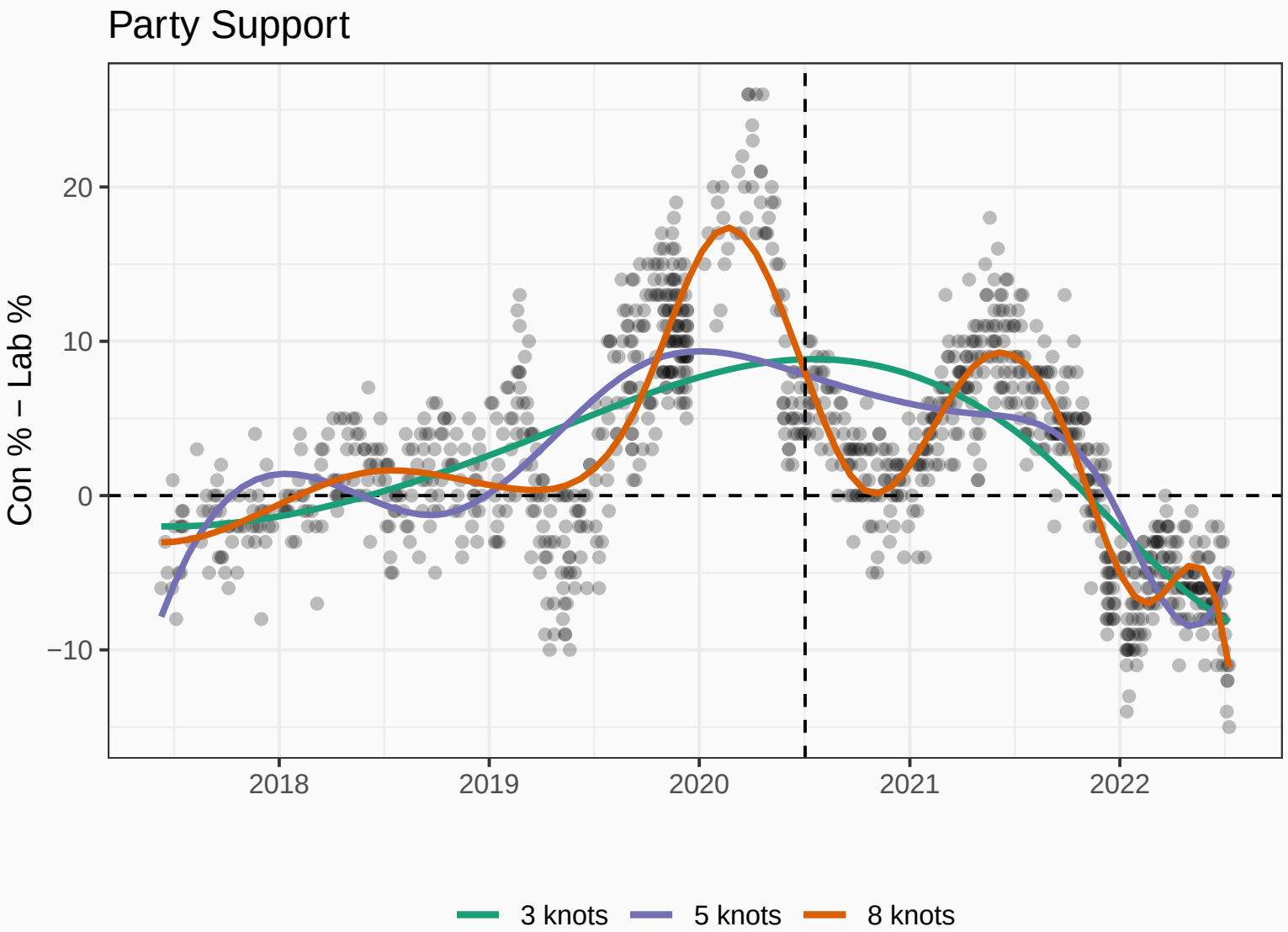
Splines – Piecewise polynomials



Splines – Piecewise polynomials



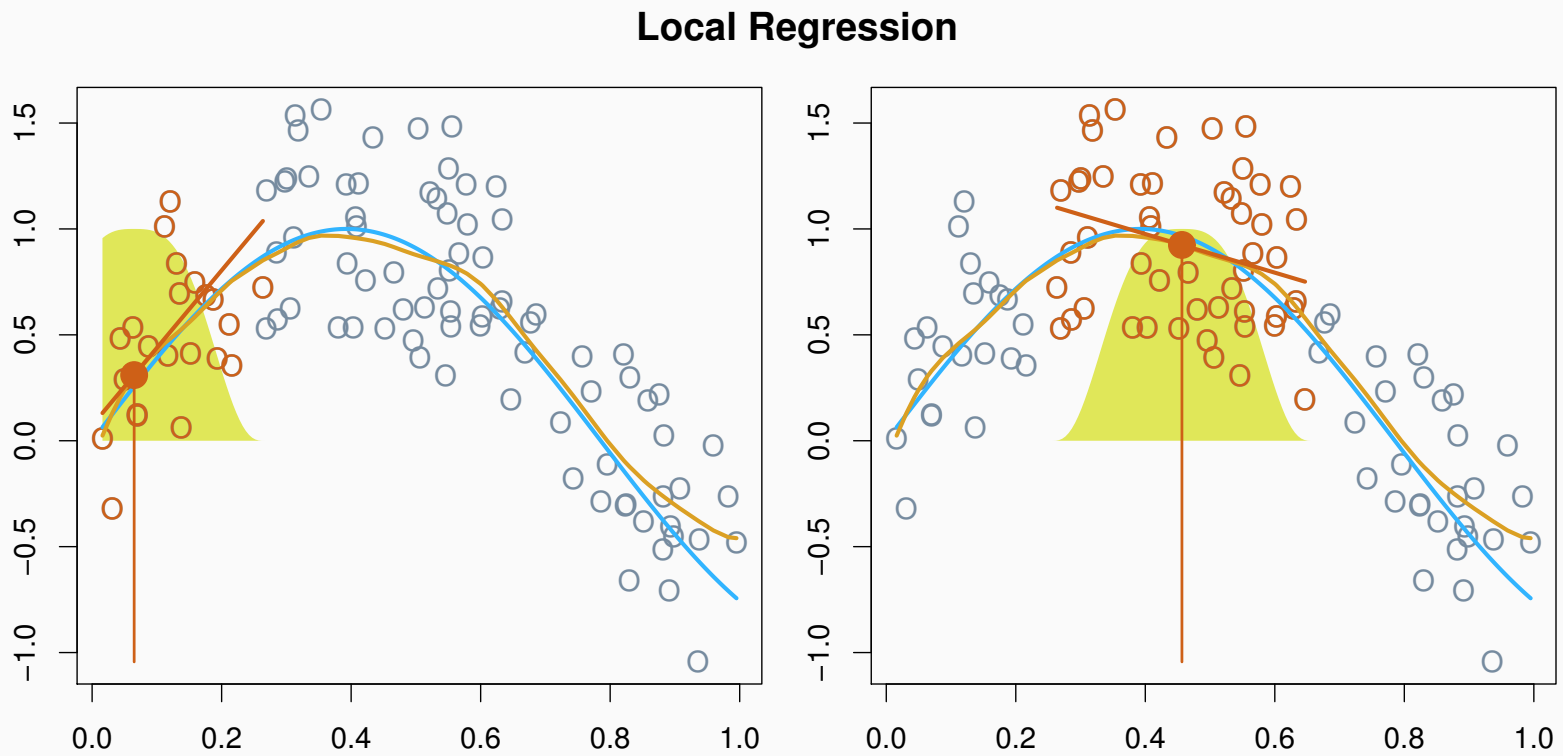
Splines – Piecewise polynomials



Local linear regression

- Local regression models do not account for non-linearities not by transforming x using a basis function
- Instead, they estimate the relationship between x and y at different points in the range of x
- For each point, x_0, we fit a weighted linear regression model using only those observations that are within some distance of x_0
- Observations that are closer to x_0 are assigned a higher weight in determining the local regression slope than observations further away
- The key parameter here is the *span*, s, which controls the proportion of points used to estimate the local regression
    - High values of $s \longrightarrow$ less flexible model
    - Low values of $s \longrightarrow$ more flexible model

Local linear regression

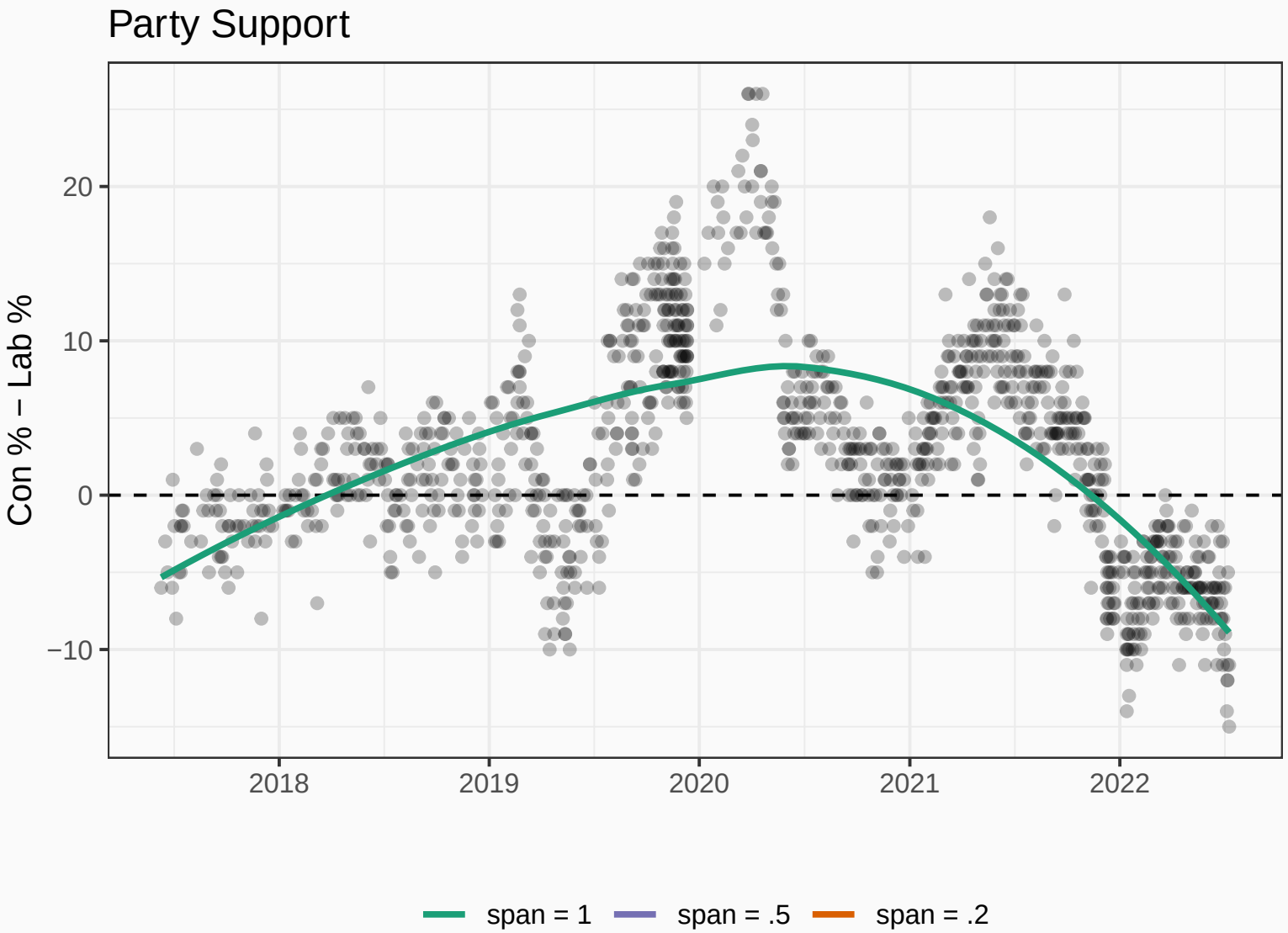


- With a sliding weight function, we fit separate linear fits over the range of X by weighted least squares.
- As s decreases, so does the size of the sliding window

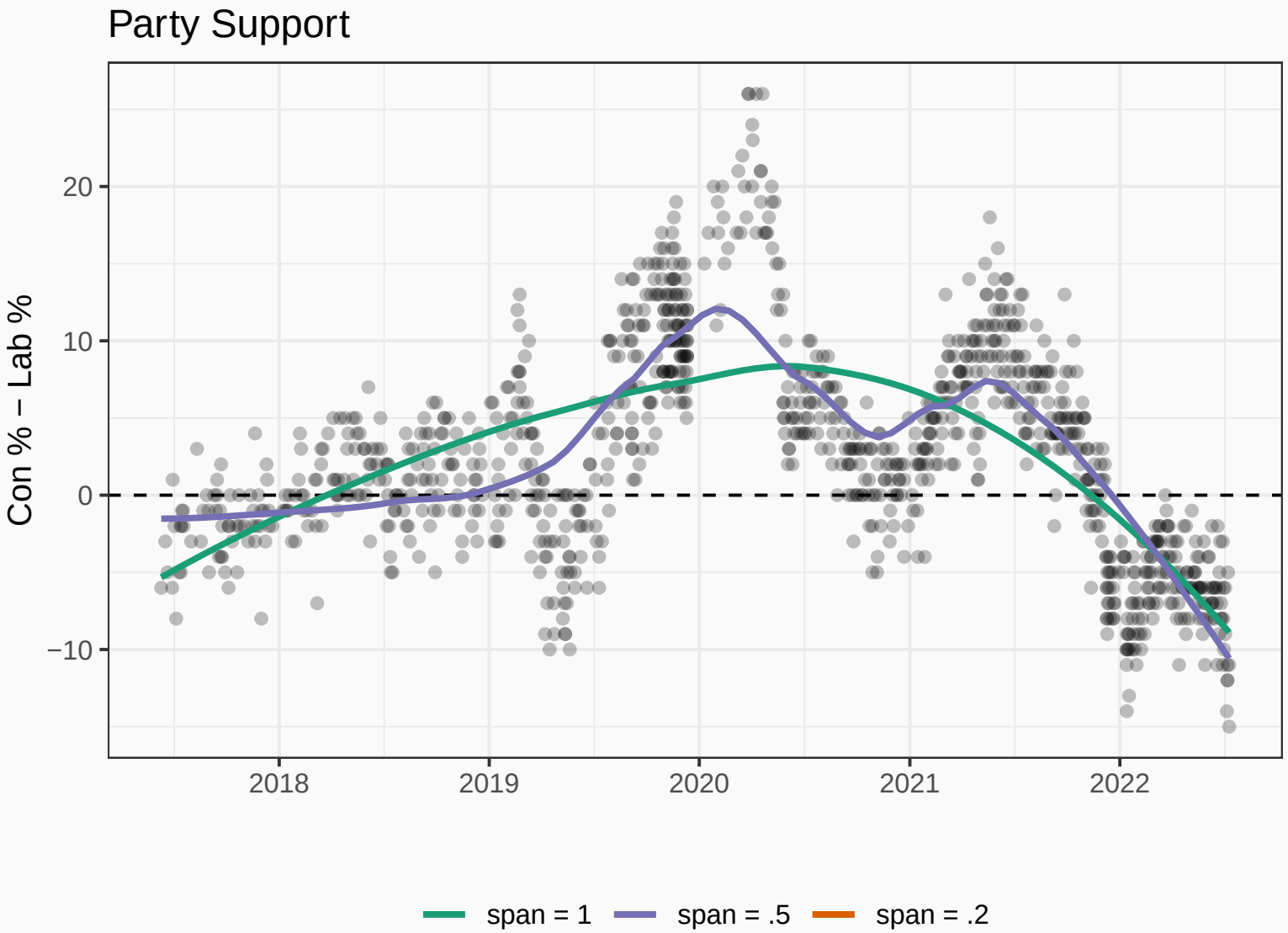
Local linear regression – application

```
loess_1 <- loess(con_lead ~ as.numeric(date), data = polls, span = 1)
loess_.5 <- loess(con_lead ~ as.numeric(date), data = polls, span = .5)
loess_.2 <- loess(con_lead ~ as.numeric(date), data = polls, span = .2)
```

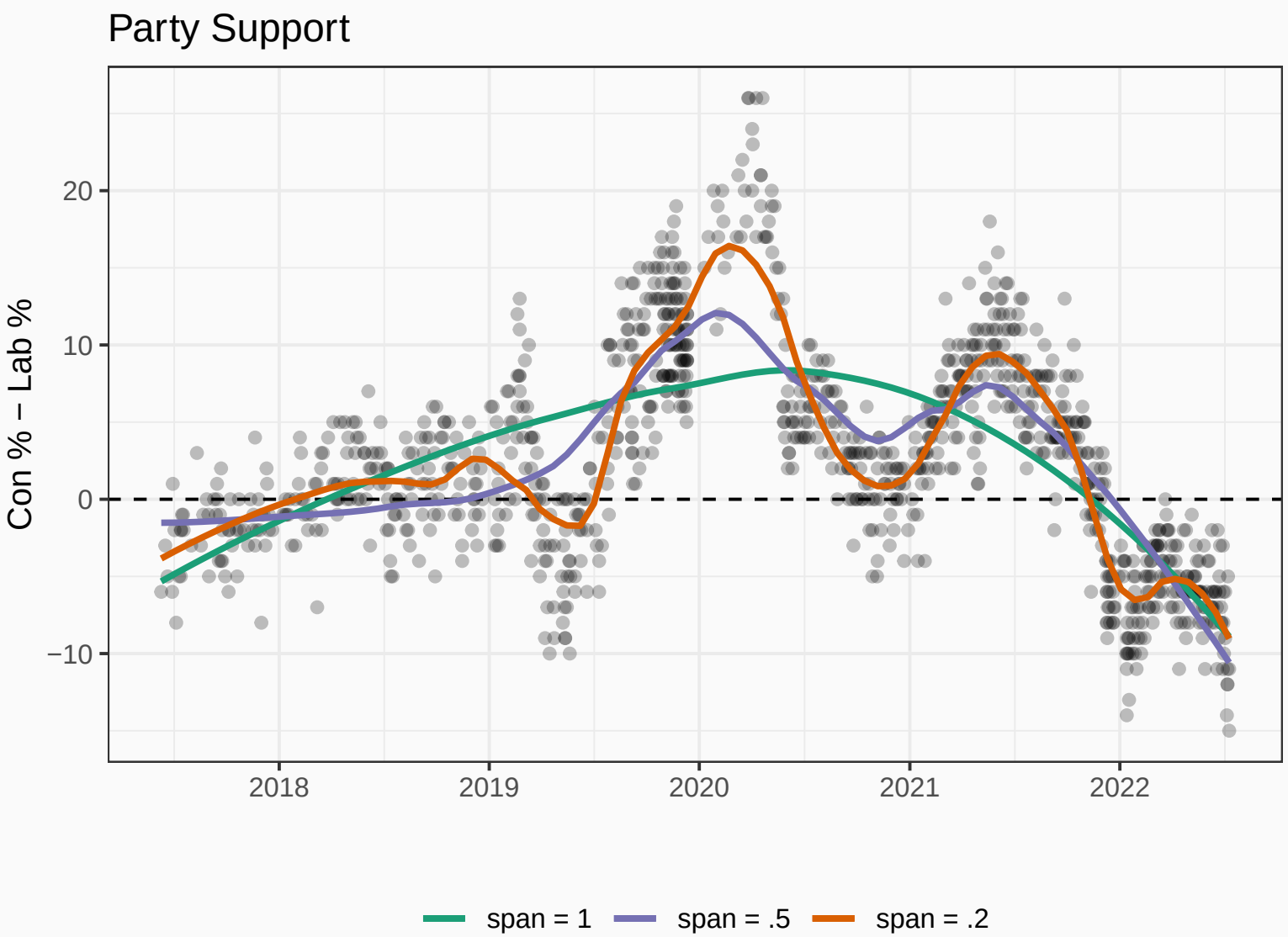
Local linear regression – application



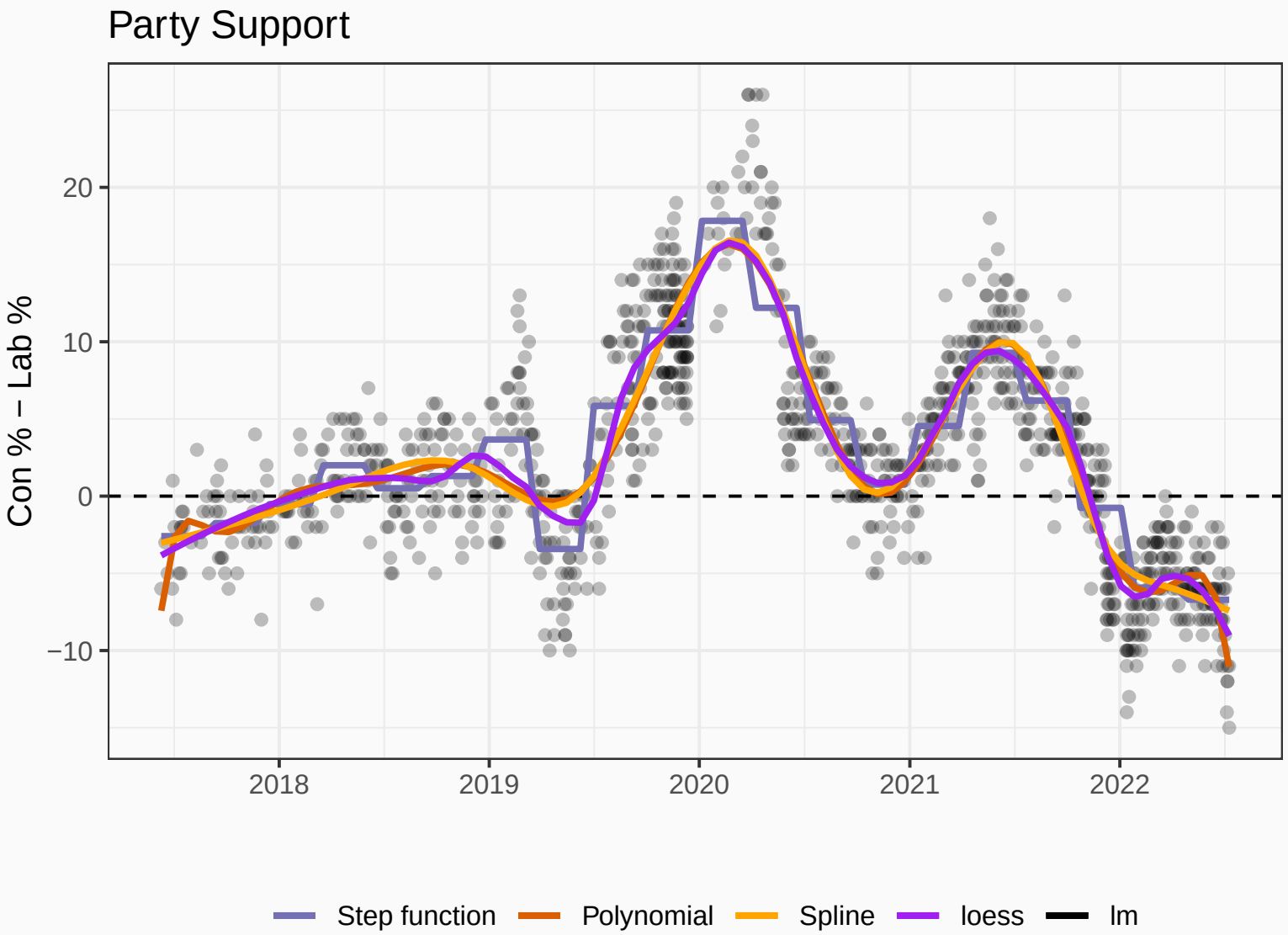
Local linear regression – application



Local linear regression – application



Which Wiggle is Best?



Choices, choices

Note that each of these methods requires the researcher to make modelling decisions which are consequential for prediction:

1. **Polynomial regression**
    - Choices: Order of polynomial (X^2, X^2, X^3, etc)
2. **Step functions**
    - Choices: Number of steps; where they are placed
3. **Splines**
    - Choices: Number of knots; where they are placed; order of polynomial
4. **Local regression**
    - Choices: Span
5. **Generalised additive models**
    - Choices: Possibly all of the above!

Are Some Wiggles Better Than Others?

We have just seen...

- ...that there are many ways to fit non-linear relationships to our data.
- ...each method gives similar, though not identical, predictions.
- ...the results each method gives depends on the modelling decisions we make (order of polynomial; degrees-of-freedom; number of steps; bandwidth; etc)

Are Some Wiggles Better Than Others?

We have just seen...

- ...that there are many ways to fit non-linear relationships to our data.
- ...each method gives similar, though not identical, predictions.
- ...the results each method gives depends on the modelling decisions we make (order of polynomial; degrees-of-freedom; number of steps; bandwidth; etc)

Question: Which wiggly line is the best wiggly line?

Are Some Wiggles Better Than Others?

We have just seen...

- ...that there are many ways to fit non-linear relationships to our data.
- ...each method gives similar, though not identical, predictions.
- ...the results each method gives depends on the modelling decisions we make (order of polynomial; degrees-of-freedom; number of steps; bandwidth; etc)

Question: Which wiggly line is the best wiggly line?

Answer:

Are Some Wiggles Better Than Others?

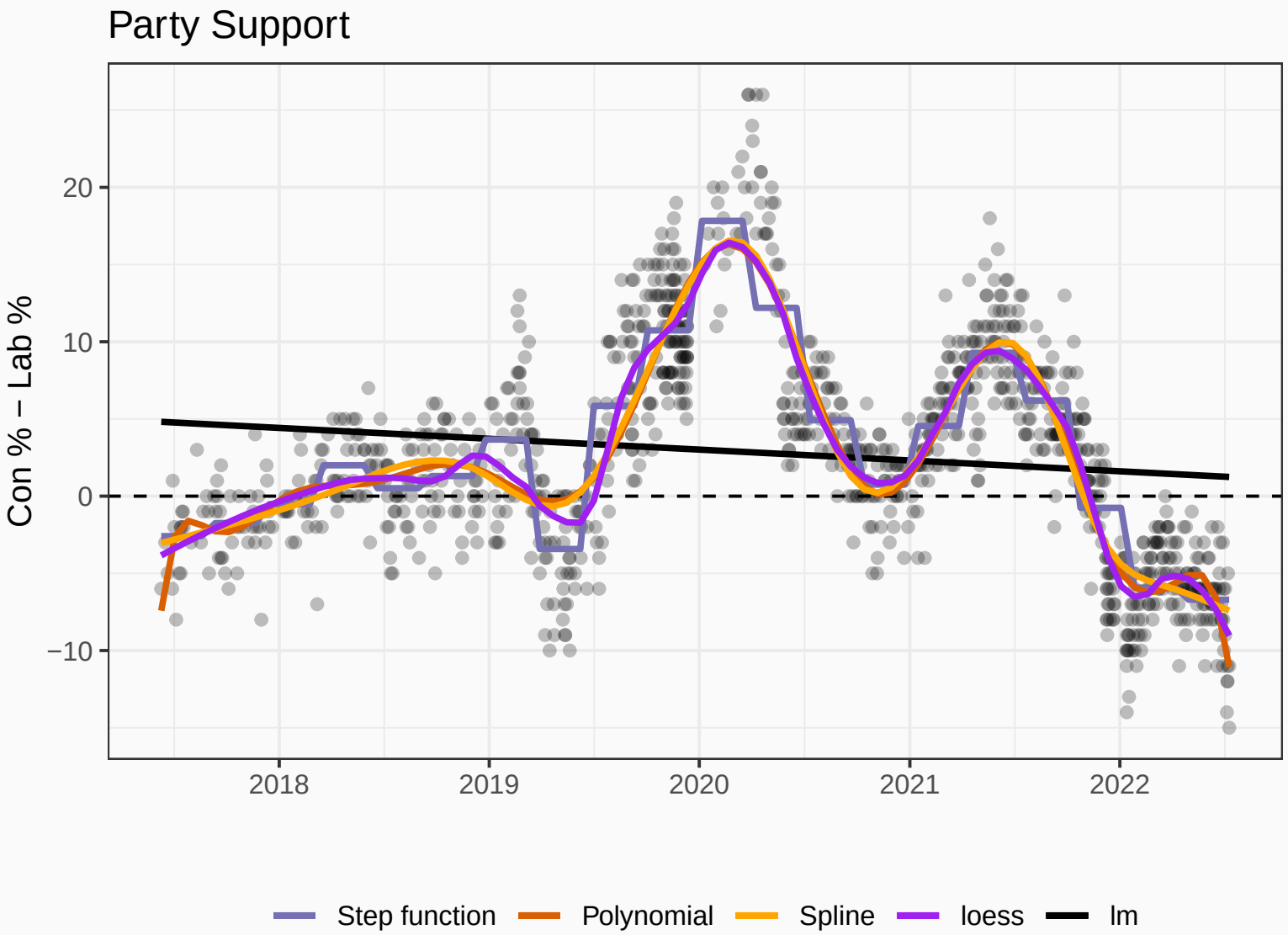
We have just seen...

- ...that there are many ways to fit non-linear relationships to our data.
- ...each method gives similar, though not identical, predictions.
- ...the results each method gives depends on the modelling decisions we make (order of polynomial; degrees-of-freedom; number of steps; bandwidth; etc)

Question: Which wiggly line is the best wiggly line?

Answer: Find out tomorrow! (Spoiler: it depends on the task, but we have tools to work it out.)

Easier Question: Which Wiggle is Worst?



Generalized Additive Models

One of the great things about linear regression (or, regression in general) is that we can include *many* predictors:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i.$$

Generalised additive models provide a framework in which we can incorporate flexible non-linearities for several variables into the additive structure of linear models:

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i$$

"Additive" because we calculate f_j for each X_j and then add them together!

Generalized Additive Models

- We can incorporate many of the methods we've just learned into a gam
    - Splines – `bs()` or `ns()`
    - Local regression – `lo()`
    - Step functions – `cut()`
    - Polynomials – `poly()`
- We can also mix terms – some linear, some non-linear – depending on our prior beliefs about functional form
- We can compare models with and without non-linear components using `anova()`
- Although they are additive, we can include interactions in the GAM just as we would in regression

GAM Example

Imagine we want to predict the wage of an individual (Y), given a list of covariates:

1. Age (continuous)
2. Year of measurement (continuous)
3. Race (categorical)
4. Education (categorical)
5. Marital status (categorical)

We might reasonably think that the outcome varies non-linearly with both age and year of measurement.

GAM Example

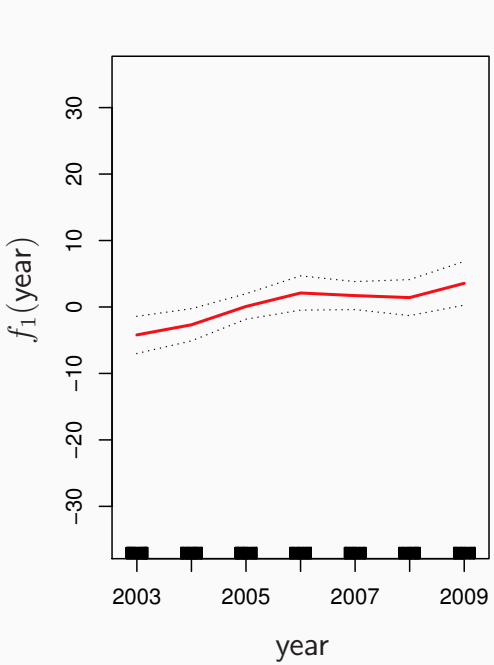
```
## 'data.frame':    3000 obs. of  11 variables:
##  $ year       : int  2006 2004 2003 2003 2005 2008 2009 2008 2006 2004 ...
##  $ age        : int  18 24 45 43 50 54 44 30 41 52 ...
##  $ maritl   : Factor w/ 5 levels "1. Never Married",..: 1 1 2 2 4 2 2 1 1 2 ...
##  $ race     : Factor w/ 4 levels "1. White","2. Black",..: 1 1 1 3 1 1 4 3 2 1 ...
##  $ education : Factor w/ 5 levels "1. < HS Grad",..: 1 4 3 4 2 4 3 3 3 2 ...
##  $ region    : Factor w/ 9 levels "1. New England",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ jobclass  : Factor w/ 2 levels "1. Industrial",..: 1 2 1 2 2 2 1 2 2 2 ...
##  $ health   : Factor w/ 2 levels "1. <=Good","2. >=Very Good": 1 2 1 2 1 2 2 1 2 2 ...
##  $ health_ins: Factor w/ 2 levels "1. Yes","2. No": 2 2 1 1 1 1 1 1 1 1 ...
##  $ logwage    : num  4.32 4.26 4.88 5.04 4.32 ...
##  $ wage       : num  75 70.5 131 154.7 75 ...
```

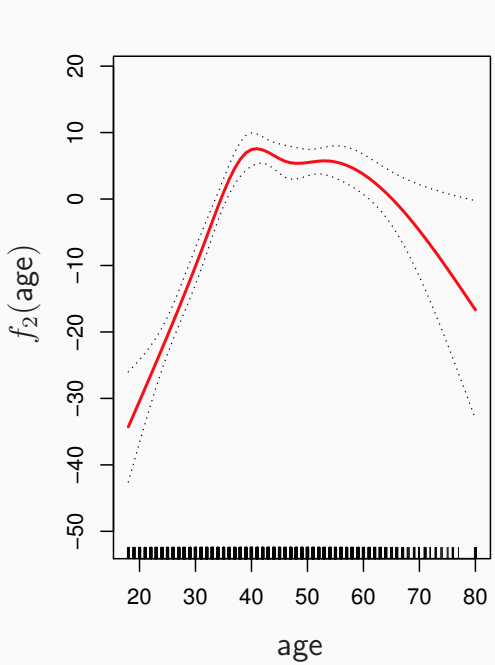
GAM Example

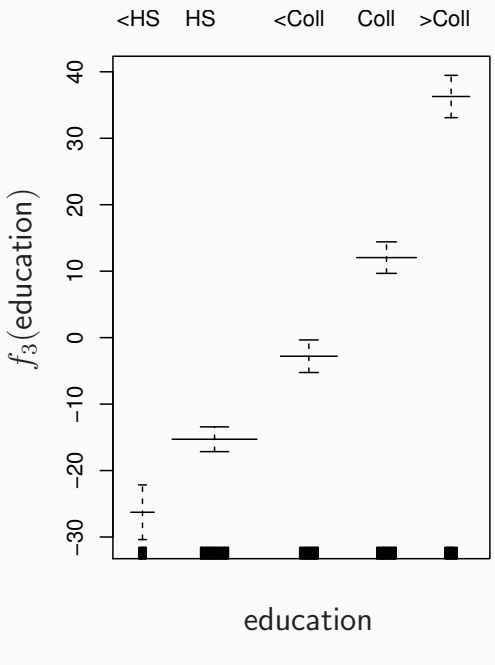
```
library(gam)
linear_mod <- gam(wage ~ year + age + race + education + maritl,
                  data = Wage)
gam_mod <- gam(wage ~ bs(year, 5) + bs(age,4) + race + education + maritl,
               data = Wage)

anova(linear_mod, gam_mod) # F test
```

```
## Analysis of Deviance Table
##
## Model 1: wage ~ year + age + race + education + maritl
## Model 2: wage ~ bs(year, 5) + bs(age, 4) + race + education + maritl
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      2986    3679441
## 2      2979    3595977  7    83464  2.2e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

GAM Example

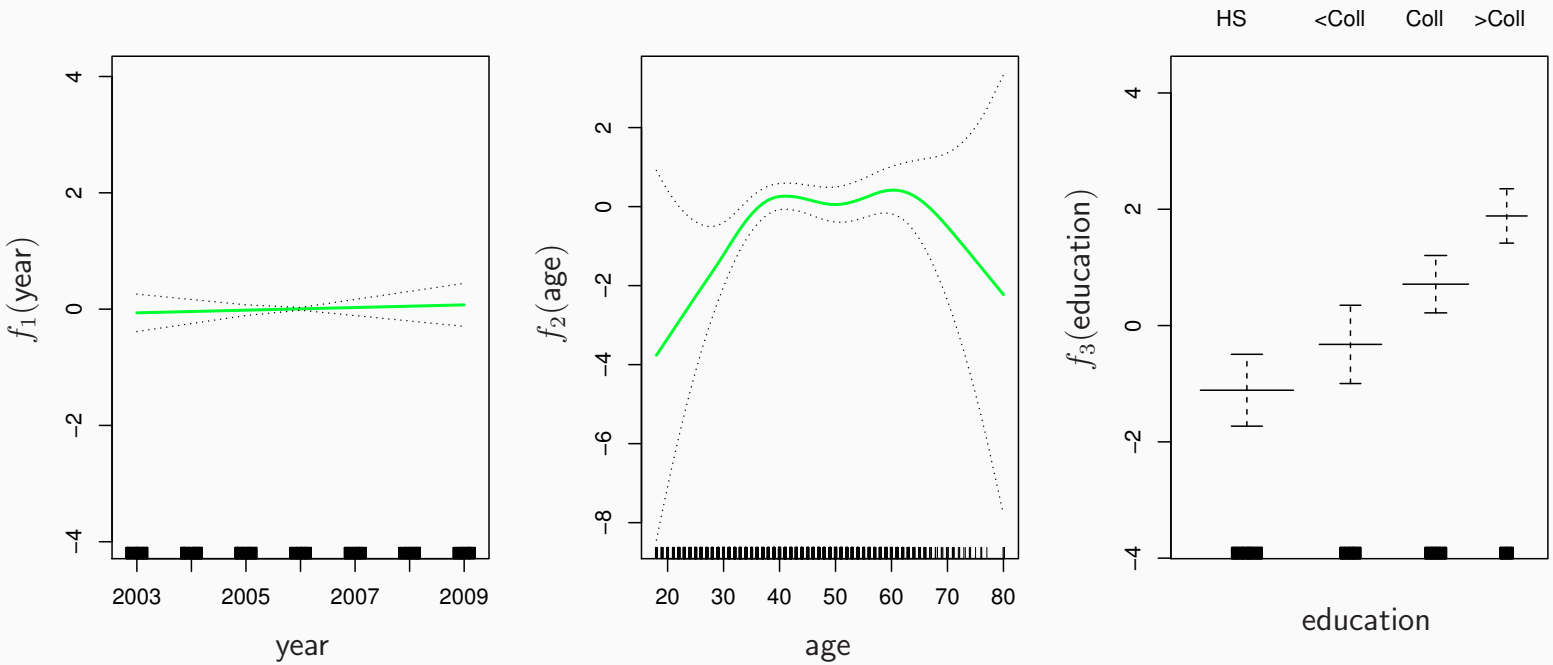

$f_1(\text{year})$
30
20
10
0
−10
−20
−30
2003
2005
2007
2009
year


$f_2(\text{age})$
20
10
0
−10
−20
−30
−40
−50
20
30
40
50
60
70
80
age


$f_3(\text{education})$
<HS
HS
<Coll
Coll
>Coll
40
30
20
10
0
−10
−20
−30
education

GAMs for classification

$$log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$



$$gam(I(wage > 250) \sim year + bs(age, df = 5) + education, family = binomial)$$

A Call to (Wiggly) Arms

> *Theory doesn't tell us to expect linearity, and our machines don't compel us to use it. Linear regression is then employed for no better reason than that users know how to type* `lm` *but not* `gam`*. You now know better, and can spread the word.*

– Cosma Shalizi

Break

Tree-based Methods

Motivation

Predicting women's wages

Can we predict how much women earn? Using data from the National Longitudinal Survey of Youth, we will try to predict women's wages from a set of individual and occupational characteristics. In addition to building our predictive model, we will focus on which variables are most important for constructing these predictions.

Motivation

```
## Rows: 1,184
## Columns: 15
## $ wage      <dbl> 6700, 1300, 2900, 1538, 1625, 2452, 1400, 1450, 1500, 1016~
## $ age       <dbl> 32, 33, 32, 30, 31, 30, 30, 29, 32, 29, 33, 32, 33, 31, 29~
## $ selfemp    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ numChildren <dbl> 1, 2, 0, 0, 1, 1, 1, 0, 0, 0, 2, 2, 0, 1, 0, 0, 1, 2, 1, 0~
## $ educ      <chr> "4.College", "2.High school", "4.College", "4.College", "2~
## $ school    <lgl> FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FAL~
## $ tenure    <dbl> 0.4807692, 1.0000000, 4.3461538, 0.3076923, 1.5384615, 4.7~
## $ fullTime   <lgl> TRUE, FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FA~
## $ unionized  <dbl> NA, 0, 1, NA, 0, 1, 1, 0, 0, NA, 0, 0, NA, 0, 1, 0, 1, 0, ~
##  $ firmSize      <chr> NA, "1. Less than 30", "3. 300+", NA, "2. 30-
299", "3. 300~
## $ marstat   <chr> "Cohabiting", "No romantic union", "No romantic union", "N~
## $ urban      <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ region     <dbl> 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ hazardous  <dbl> 1.153185, 1.119433, 1.952752, 1.638367, 1.265751, 1.119433~
## $ teamwork   <dbl> 4.166900, 4.583686, 4.141300, 4.310577, 4.307690, 4.583686~
```

Tree-based Methods

While we could use linear regression to predict this outcome, here we will focus on tree-based methods.

Approach:

1. Stratify or segment the predictor space into a number of simple regions
2. Calculate the mean outcome in each region
3. Predict $\hat{y}_i$ on the basis of the region into which i falls

Since the set of splitting rules used to segment the predictor space can be summarised in a tree, these types of approaches are known as decision-tree methods.

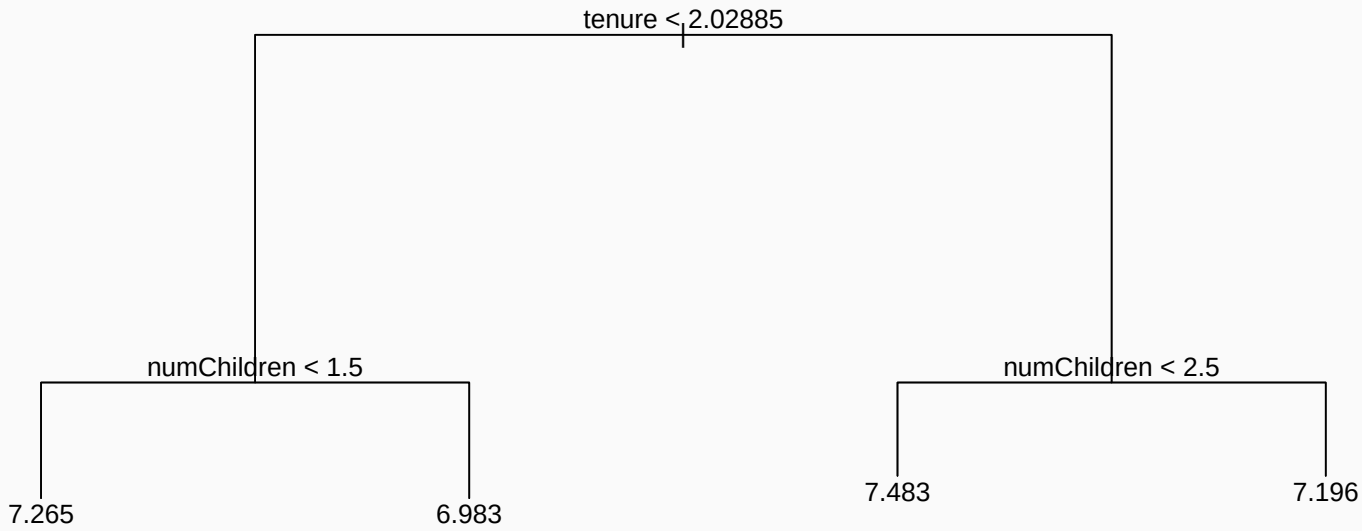
Pros and Cons

- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss bagging, and random forests. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

A Simple Tree

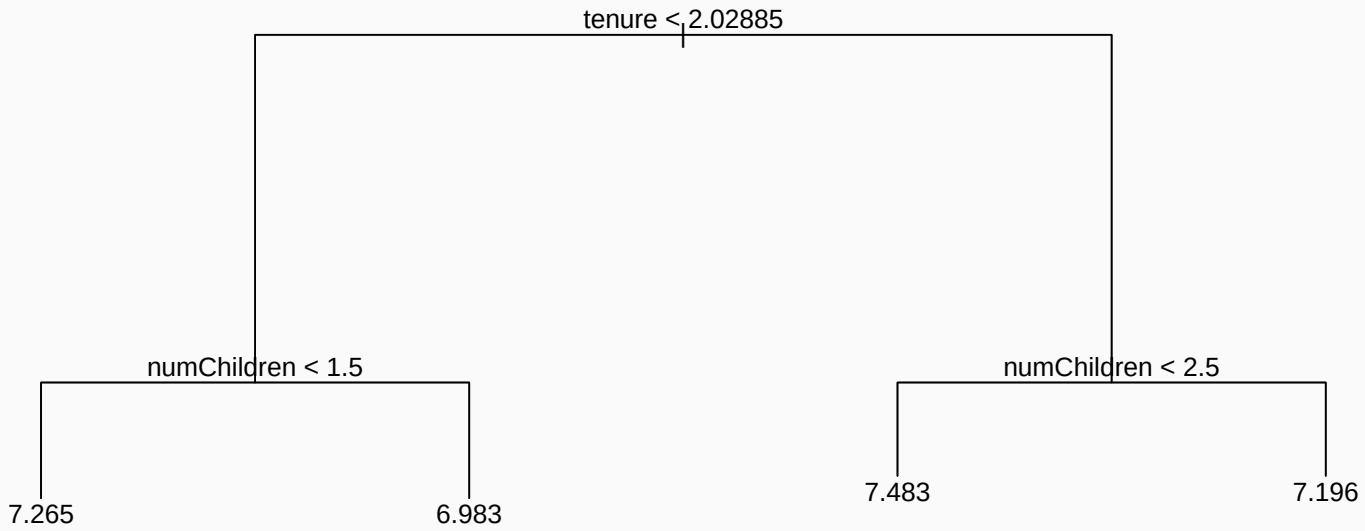
```
library(tree)

wage_tree <- tree(log(wage) ~  tenure + numChildren, data = wages)
```

A Simple Tree



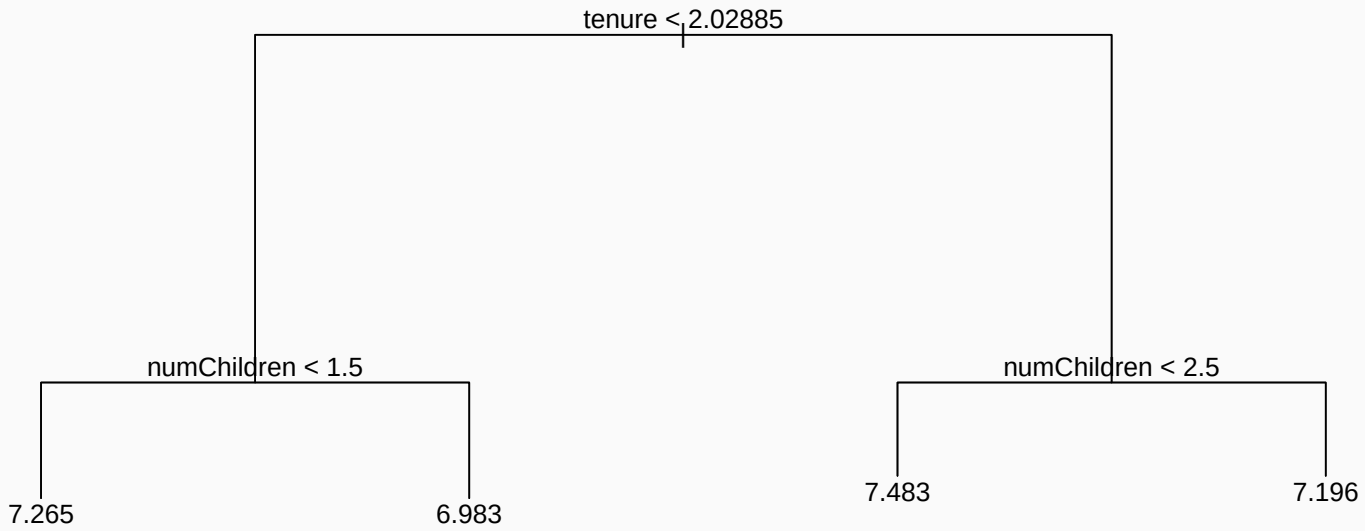
A regression tree predicting log wage as a function of tenure and number of children

A Simple Tree



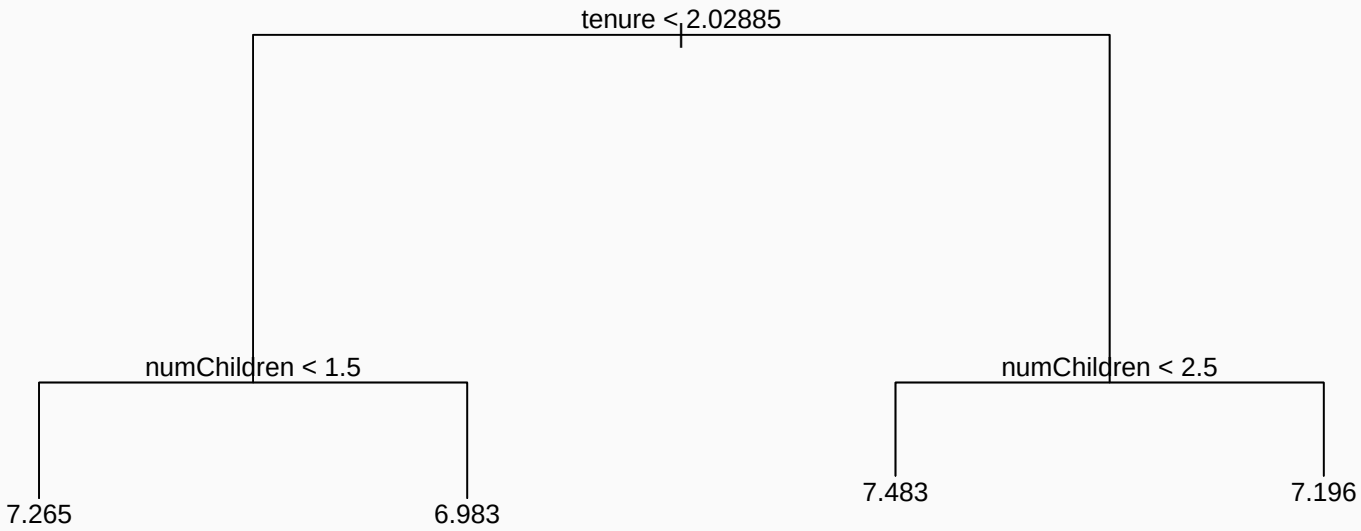
At any internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$

A Simple Tree



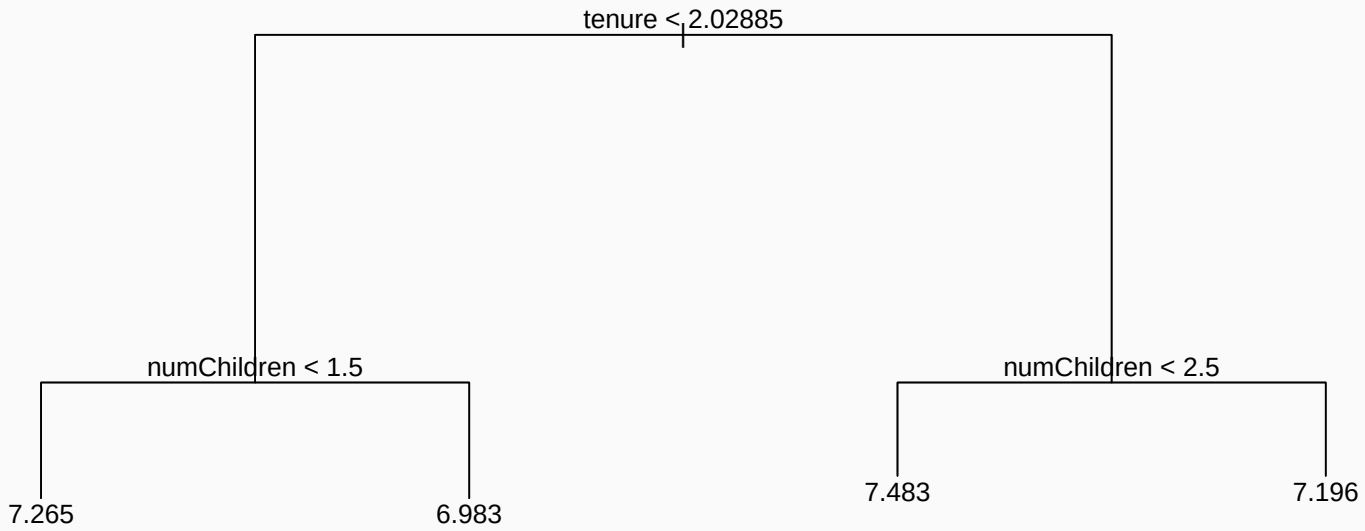
Here, the left-hand branch corresponds to $tenure < 2.02$, and the right-hand branch corresponds to $tenure \geq 2.02$

A Simple Tree



The points along the tree where the predictor space is split are referred to as internal nodes.

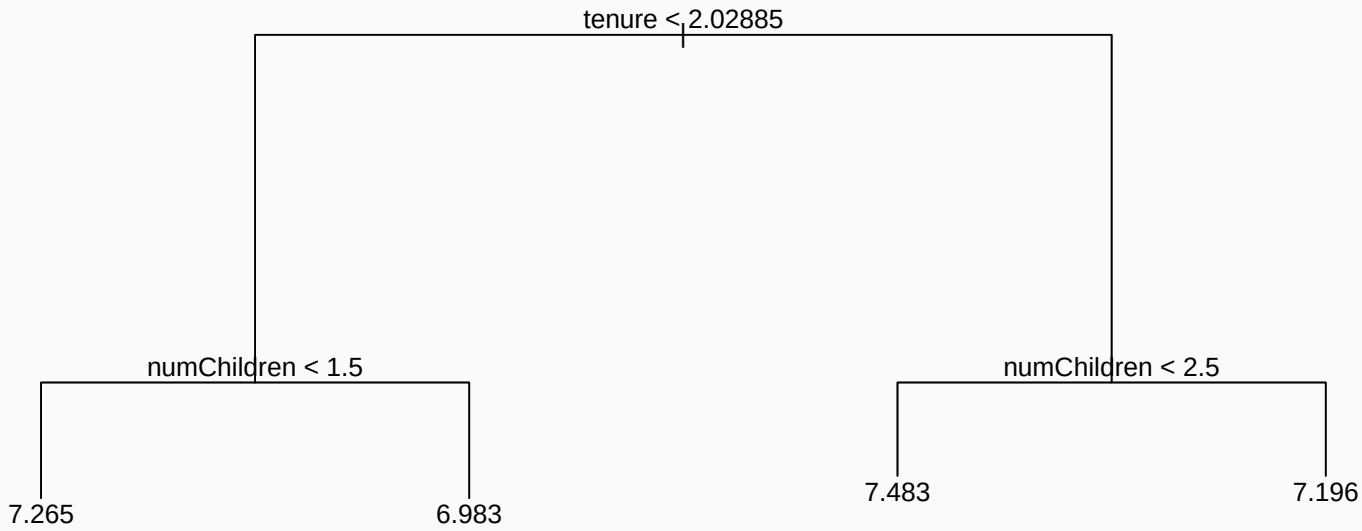
A Simple Tree



In this tree, three internal nodes are indicated:

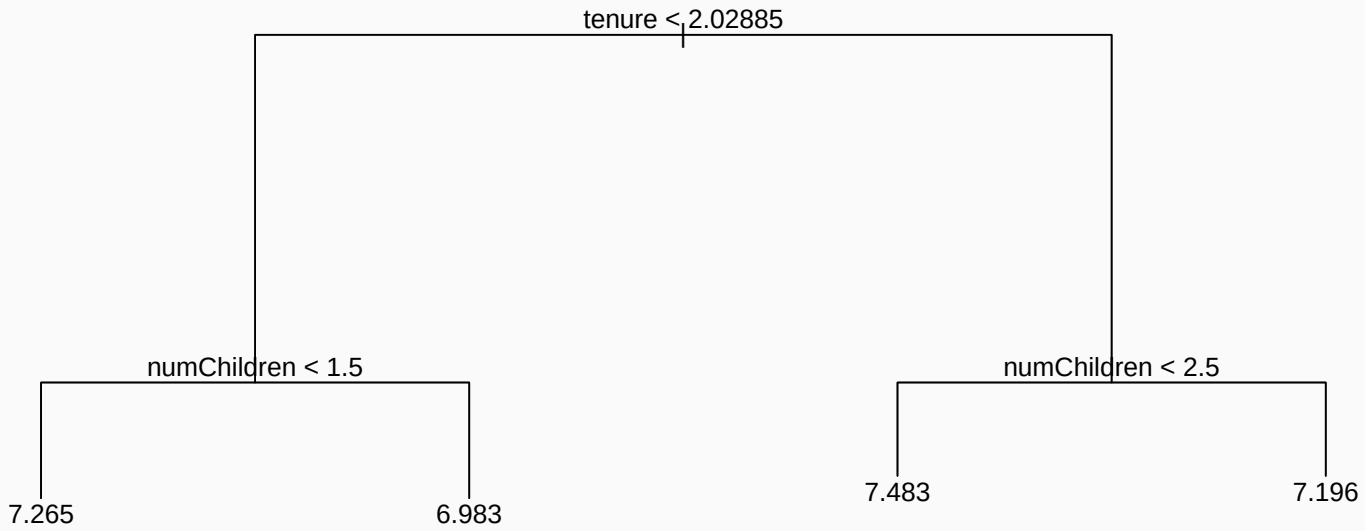
- $tenure < 2.02$
- $numChildren < 1.5$
- $numChildren < 2.5$

A Simple Tree



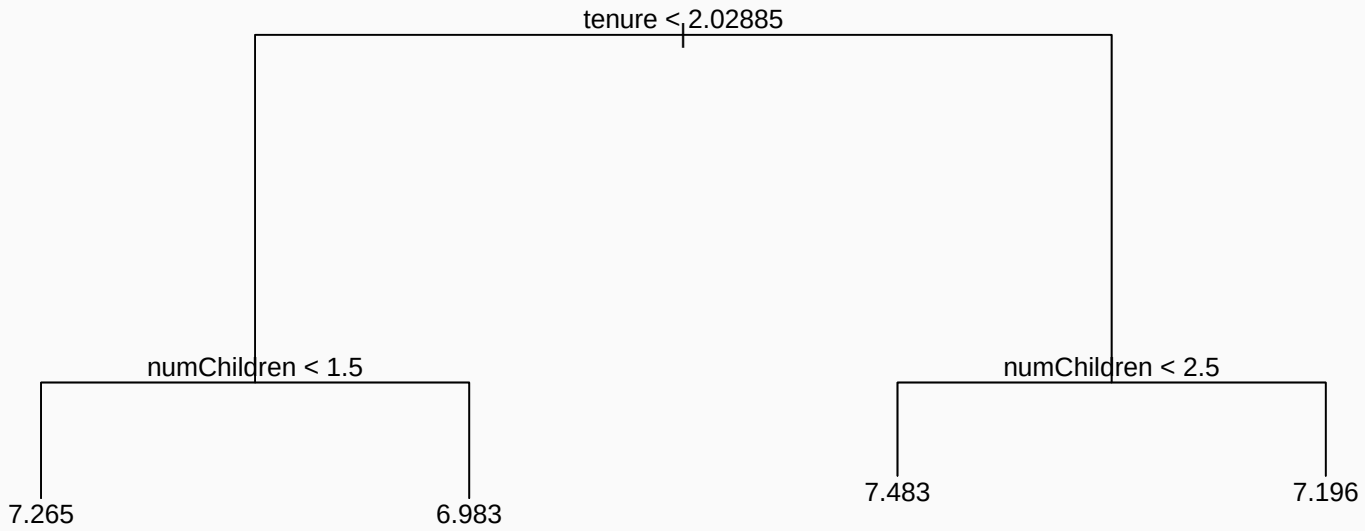
Note that the splitting rule is different for the two internal nodes involving `numChildren`! (More on this later)

A Simple Tree



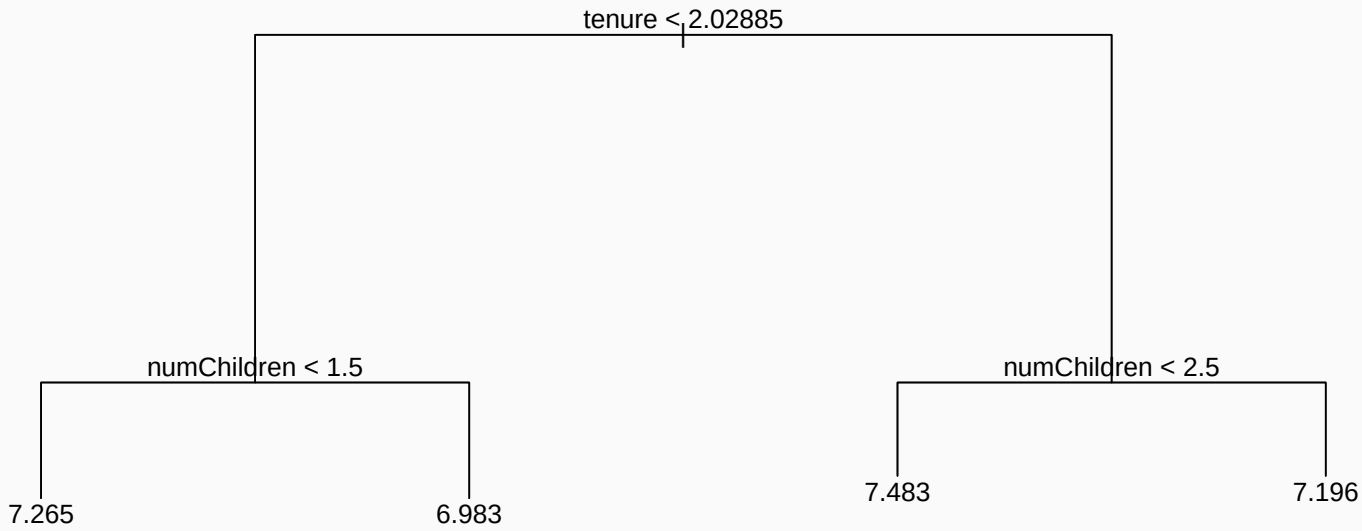
The terminal nodes, or *leaves*, indicate the mean of Y for the observations in that partition.

A Simple Tree



We use the values in the leaves as the predicted values, $\widehat{y}$, for new observations.

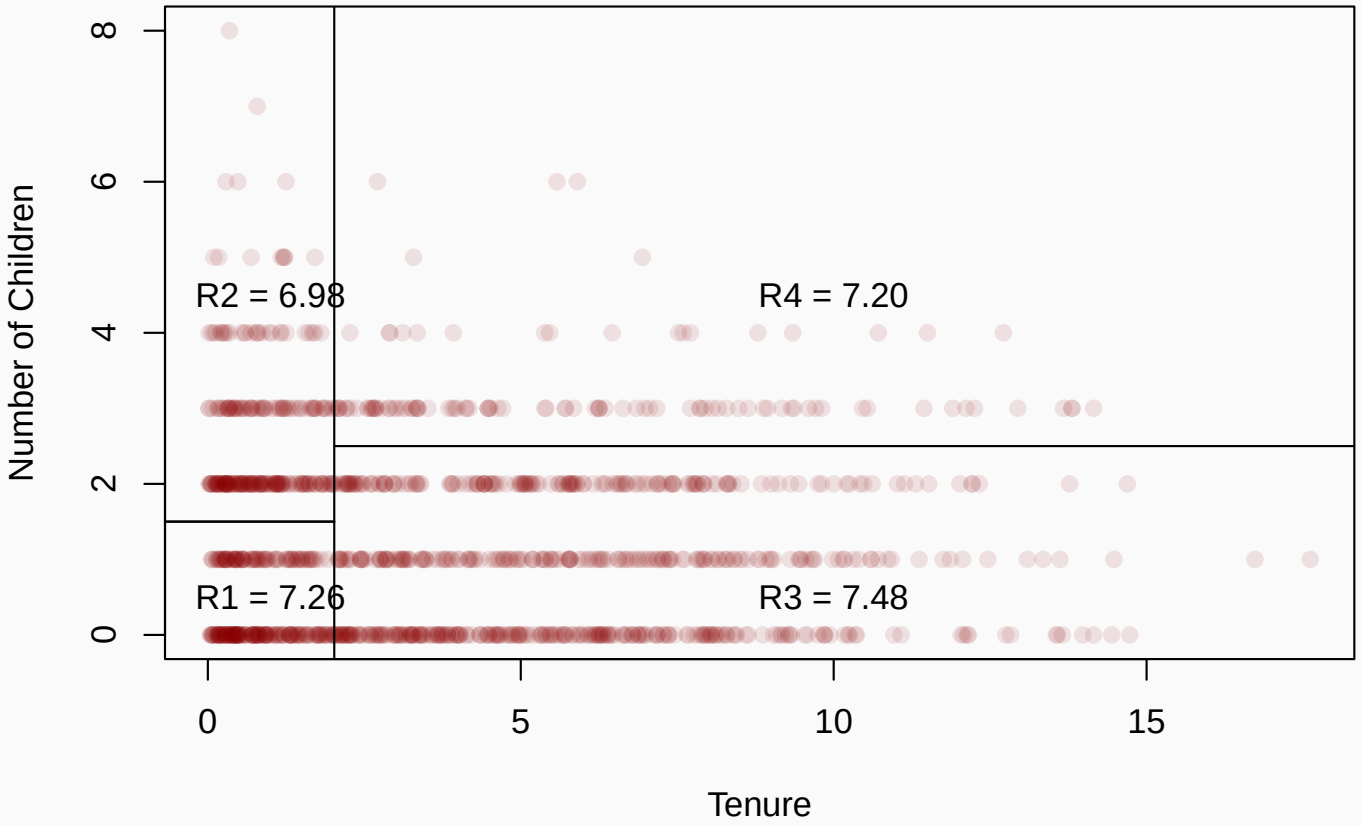
A Simple Tree



Interpretation:

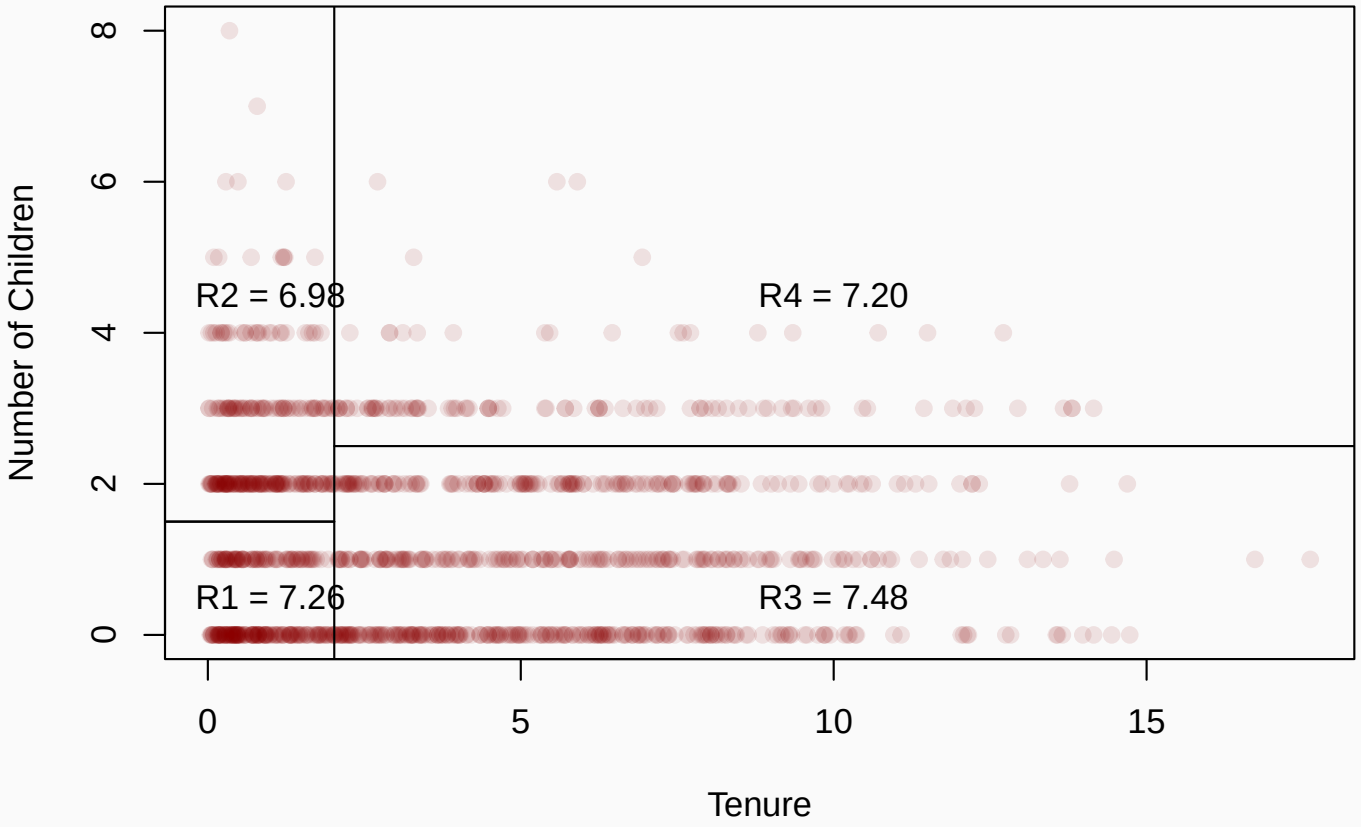
- $tenure$ is the most important factor in determining $wage$, and individuals with less experience earn lower salaries than more experienced individuals
- Conditional on $tenure$, people with fewer children earn more than people with more children

Results



The tree stratifies or segments individuals into four regions of predictor space

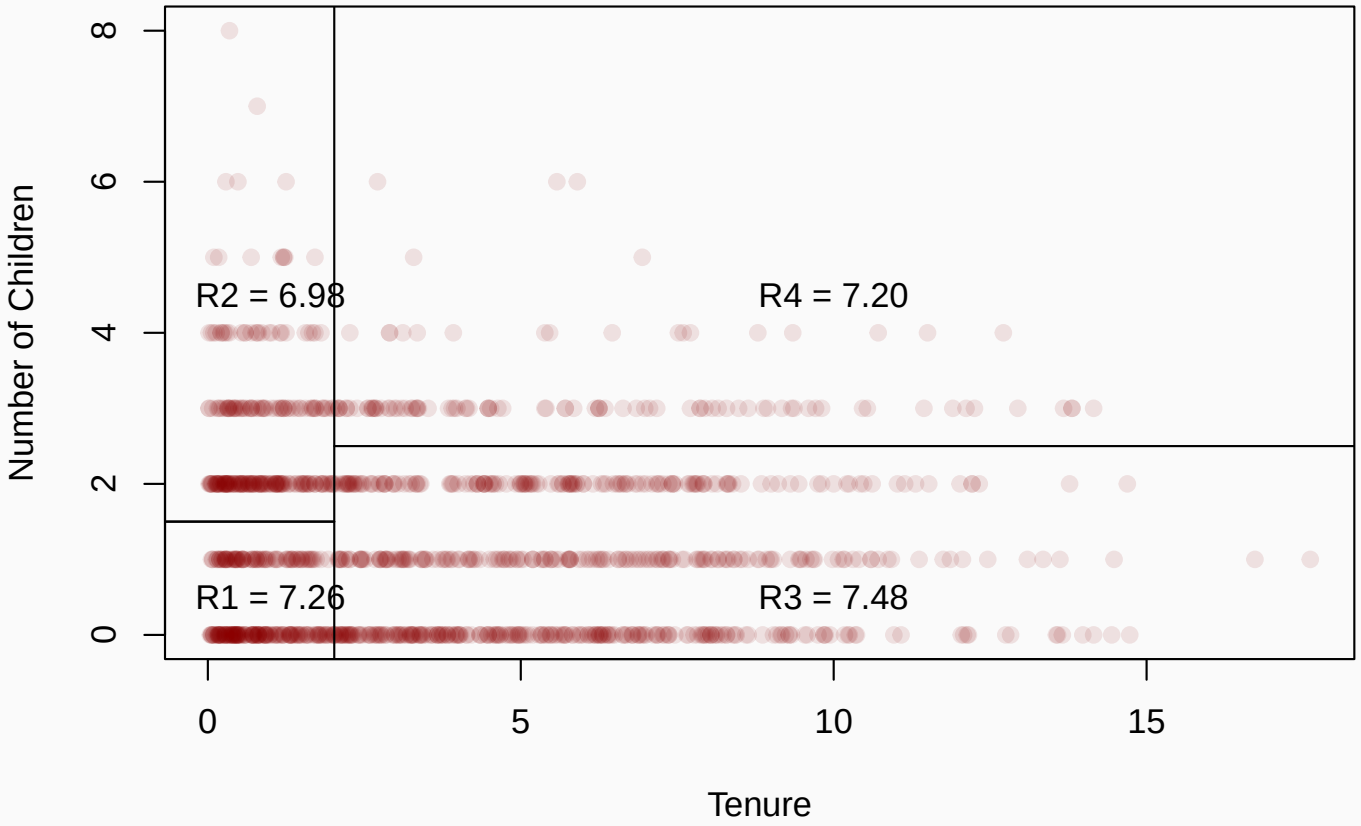
Results



$$R_1 = \{X | Tenure < 2.02, Children < 1.5\}$$

$$R_2 = \{X | Tenure < 2.02, Children > 1.5\}$$

Results



$$R_3 = \{X | Tenure > 2.02, Children < 2.5\}$$

$$R_4 = \{X | Tenure > 2.02, Children > 2.5\}$$

How to Grow a Tree

1. We divide the predictor space – that is, the set of possible values for $X_1, X_2, \ldots, X_p$ – into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

How to Grow a Tree

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the Residual Sum of Squares (RSS), given by

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

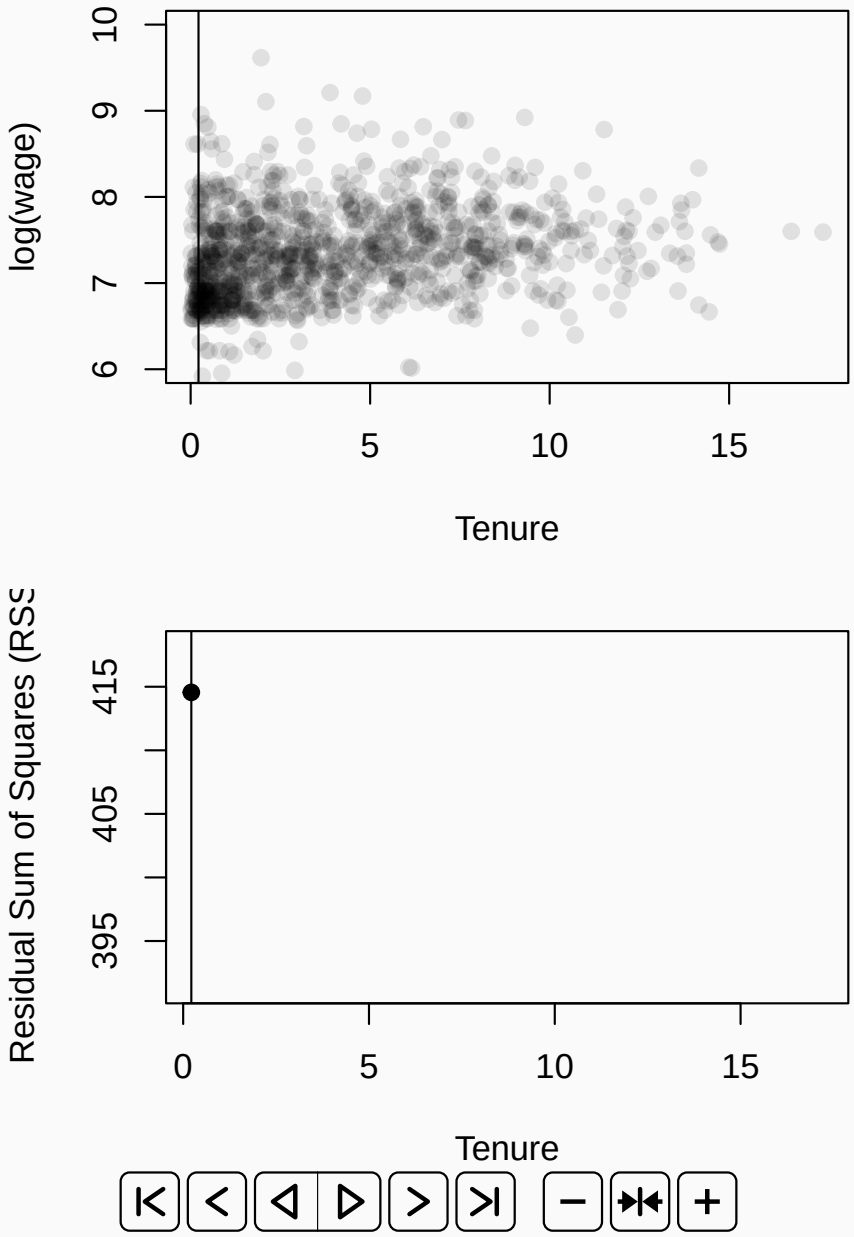
How to Grow a Tree

- Unfortunately, it is not computationally feasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

How to Grow a Tree

- We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

How to Grow a Tree – illustration



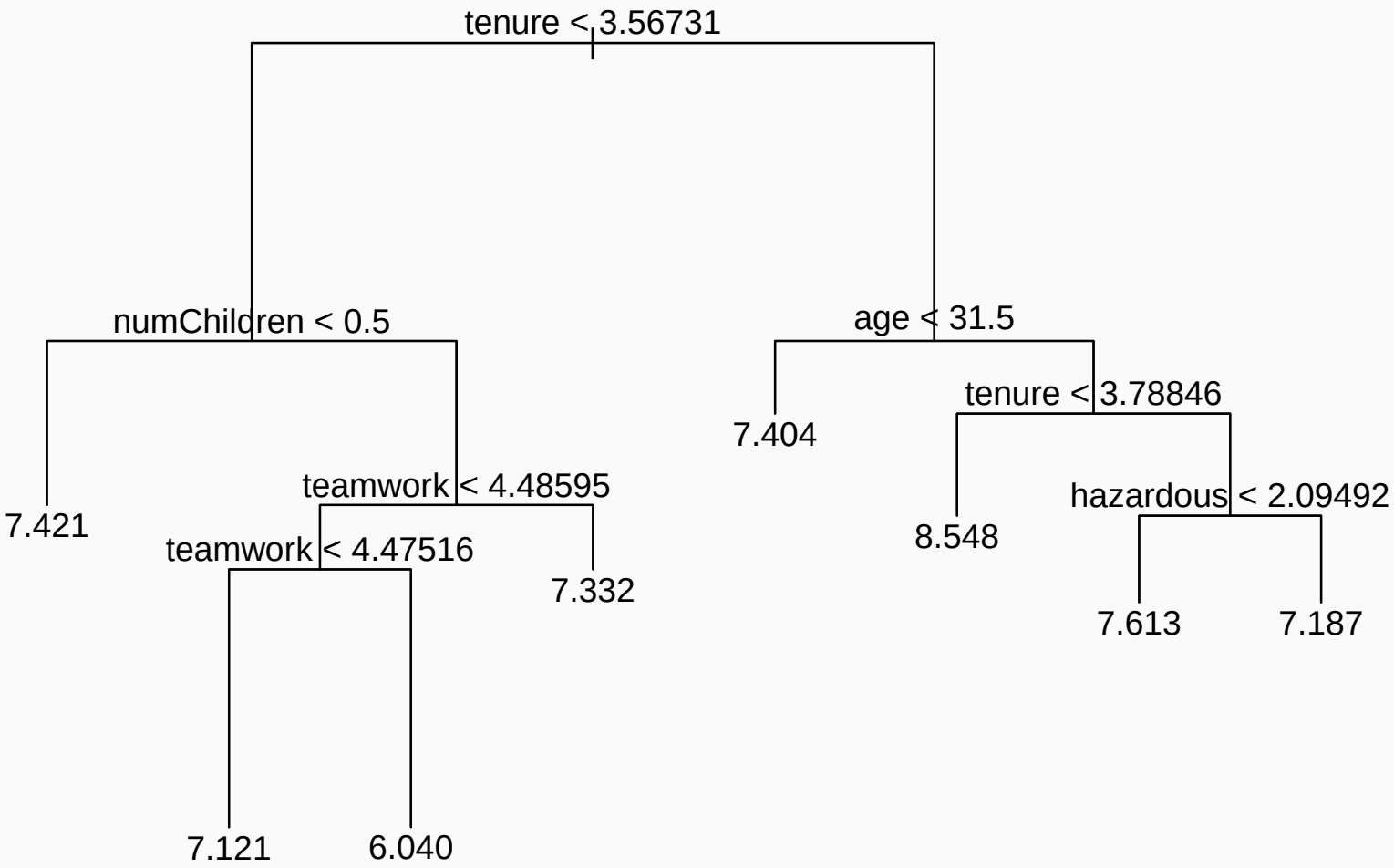
Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.
- Note that one interpretation of the tree-building process is similar to the basis functions we discussed earlier.
- We seek a transformation of *a set of* X variables, $f(x)$, which we then include in a regression for the outcome.
- The transformation we use is to create a set of dummy variables, X_i, defined by "rectangular" segmentations of the covariate space
- With these, we then estimate $y_i = \sum_{r=1}^{R} \beta_r X_{ri} + \epsilon_i$, and make predictions using this fitted model

Application

```
wage_tree <- tree(log(wage) ~ age + selfemp + numChildren +
                              educ + school + tenure + fullTime +
                              unionized + firmSize + marstat +
                              urban + region  + hazardous +
                              teamwork,
                            data = wages)
```

Application



Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split – that is, a split that leads to a large reduction in RSS later on.

Pruning a tree

- Alternative strategy: grow a very large tree T_0, and then prune it back in order to obtain a subtree.
- Cost complexity pruning – also known as weakest link pruning – is used to do this.
    - **Intuition:** Prefer (sub)trees that fit the data well, but penalise them for having too many leaves
- Rather than minimizing RSS, we instead minimize $RSS + \alpha \cdot (\#\text{terminal nodes})$

Pruning a tree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- When $\alpha = 0$, then we get the same tree as without the penalty
- As α increases, we recursively "snip" off the least important splits

Pruning a tree

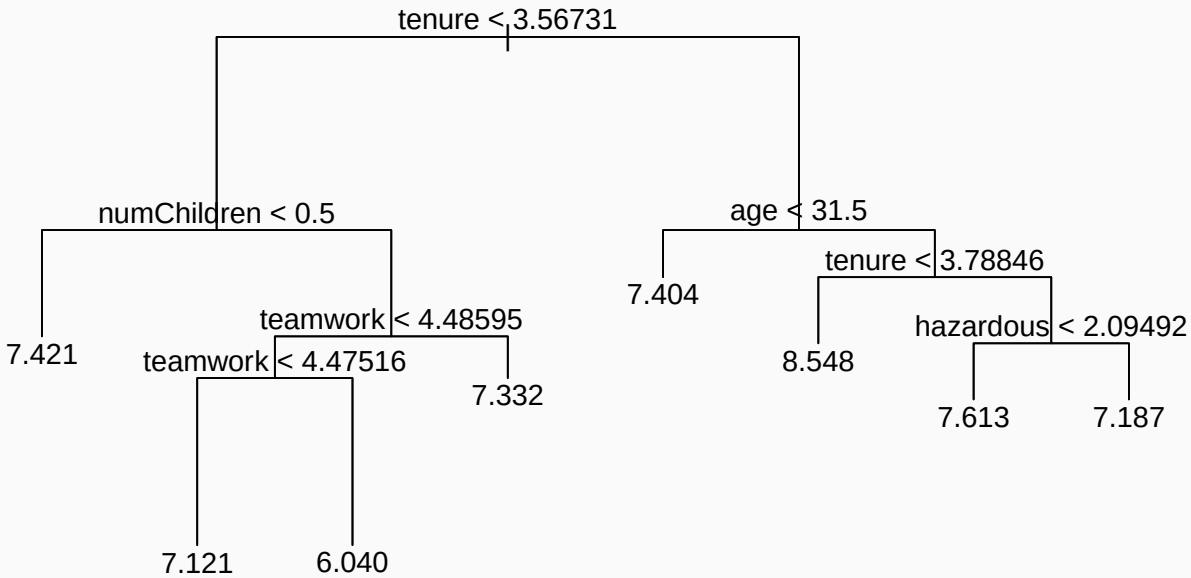
Nice tree!

Pruning a tree

Nicer tree!

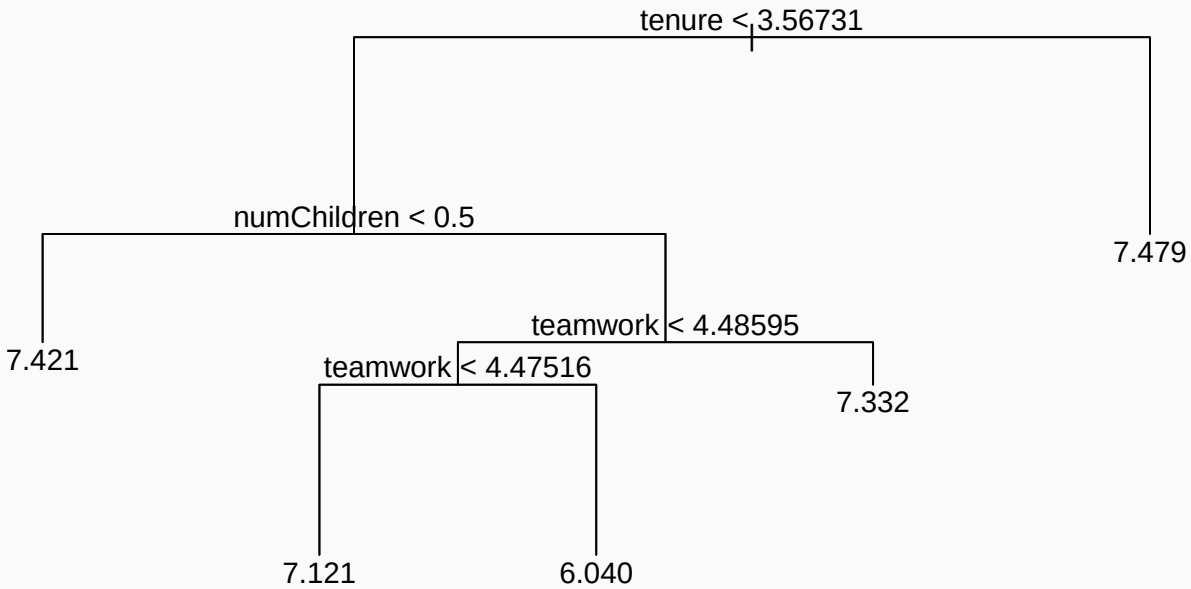
Pruning a tree – application

```
wage_tree_alpha_0 <- prune.tree(wage_tree, 0) # alpha = 0
plot(wage_tree_alpha_0)
text(wage_tree_alpha_0)
```



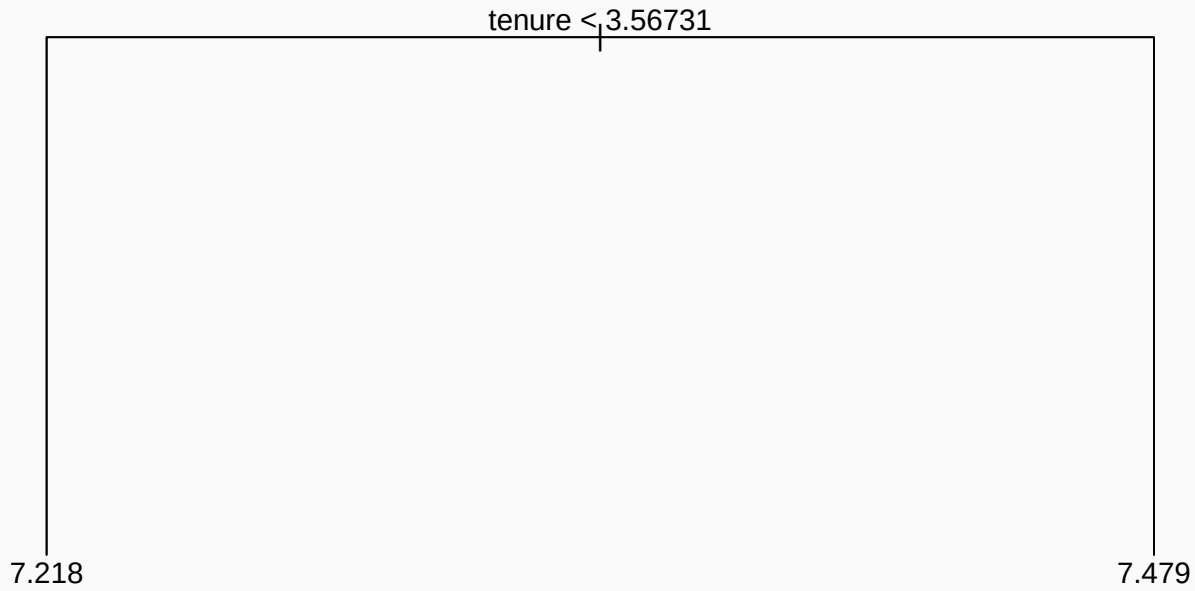
Pruning a tree – application

```
wage_tree_alpha_5 <- prune.tree(wage_tree, 5) # alpha = 5
plot(wage_tree_alpha_5)
text(wage_tree_alpha_5)
```



Pruning a tree – application

```
wage_tree_alpha_8 <- prune.tree(wage_tree, 8) # alpha = 8
plot(wage_tree_alpha_8)
text(wage_tree_alpha_8)
```



Choosing the best subtree

- We select an optimal value $\hat{\alpha}$ using cross-validation.
    - More on this tomorrow!
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: tree algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:

    3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.

    3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

Details of classification trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits.
- A natural alternative to RSS is the classification error rate. This is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \underbrace{max}_{k}(\hat{p}_{mk}).$$

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index and Deviance

- The Gini index is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.

- For this reason the Gini index is referred to as a measure of node purity – a small value indicates that a node contains predominantly observations from a single class.

- An alternative to the Gini index is cross-entropy, given by

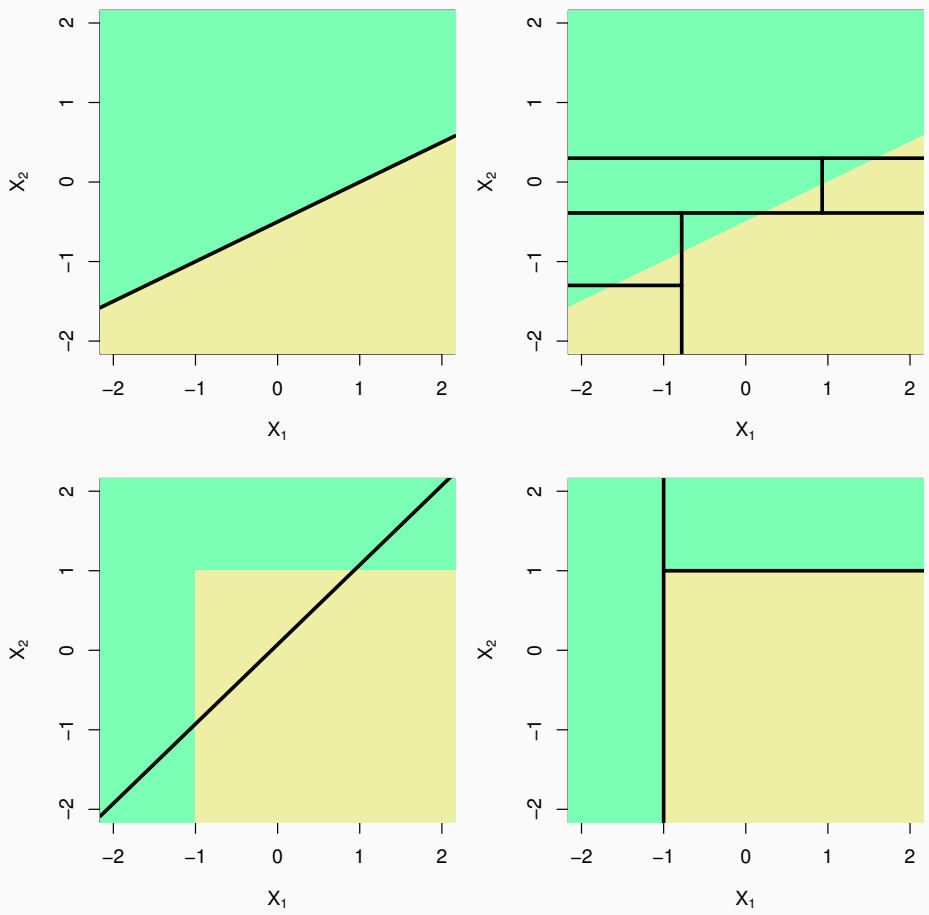
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk}.$$

- Both the Gini index and the cross-entropy are very similar numerically.

When Should I Grow A Tree?

- Linear models
    - If the relationship between predictors and outcome is *actually* linear, then linear regression will give the best predictions
- Tree-based models
    - If the relationship between the features and the response is non-linear and interactive between features, then trees are likely to be successful

Trees Versus Linear Models

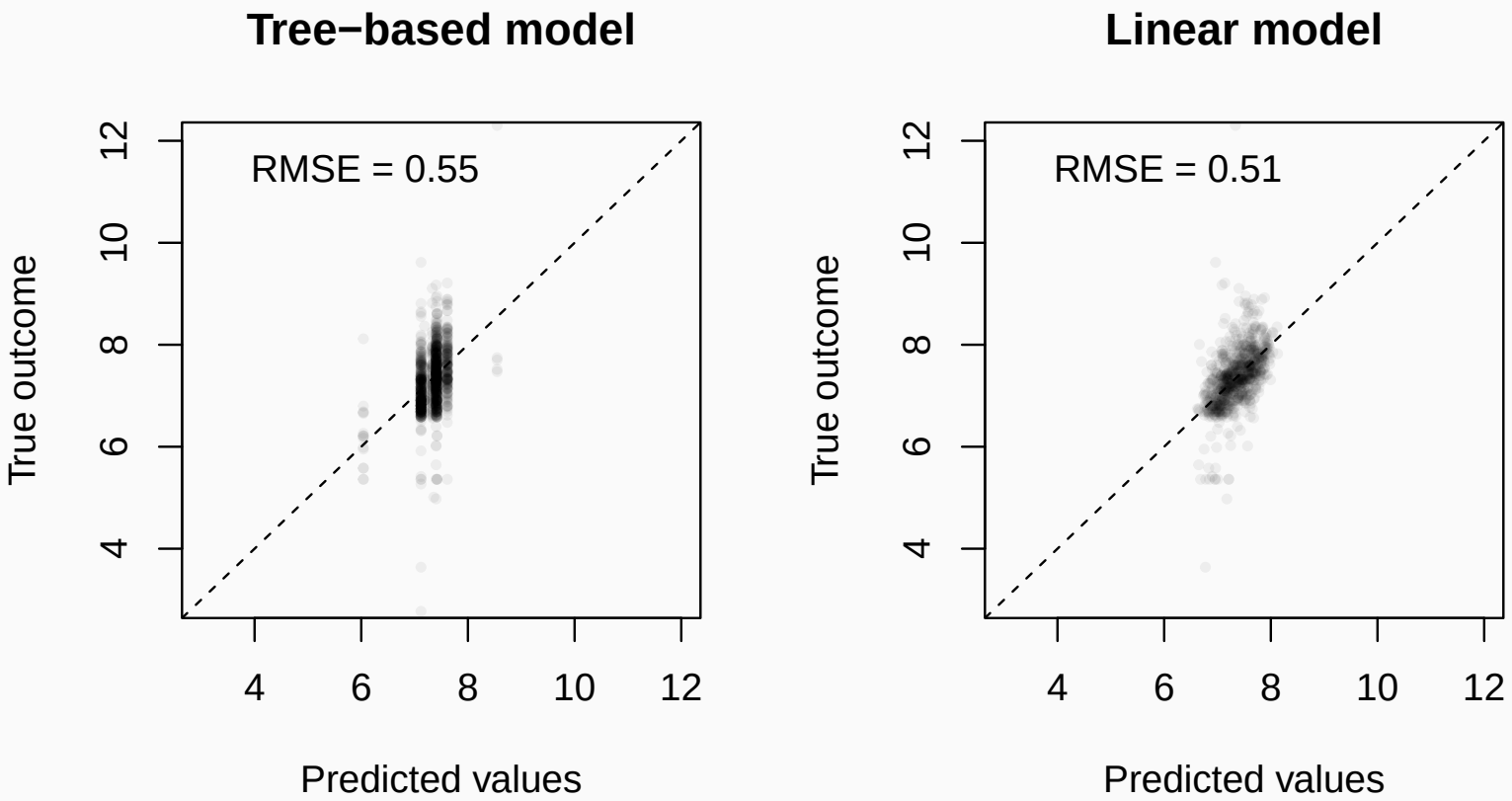


- Top Row: True linear boundary; Bottom row: true non-linear boundary.
- Left column: linear model; Right column: tree-based model

Trees Versus Linear Models

```
wage_tree <- tree(log(wage) ~ age + selfemp + numChildren + educ + school +
                      tenure + fullTime + unionized + firmSize + marstat +
                      urban + region  + hazardous + teamwork,
                  data = wages)

wage_lm <- lm(log(wage) ~ age + selfemp + numChildren + educ + school +
                  tenure + fullTime + unionized + firmSize + marstat +
                  urban + region  + hazardous + teamwork,
              data = wages)
```

Trees Versus Linear Models



Advantages of Trees

- Trees are very easy to explain to people (often even easier than linear regression).
- Some people believe that decision trees more closely mirror human decision-making than do other common regression/classification approaches.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees do not require you to specify the functional form between the predictors and the response and can automatically detect often complex interactions).

Disadvantages of Trees

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches we've seen previously.
- Trees tend to be very high variance: a small change in the data used to estimate the model can cause large changes in the estimated tree.

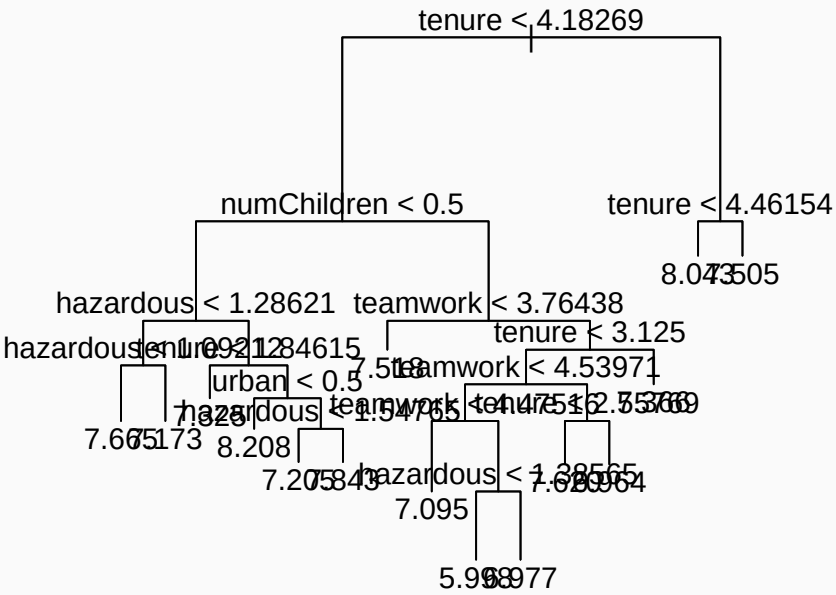
However, by aggregating many decision trees, the predictive performance of trees can be substantially improved.

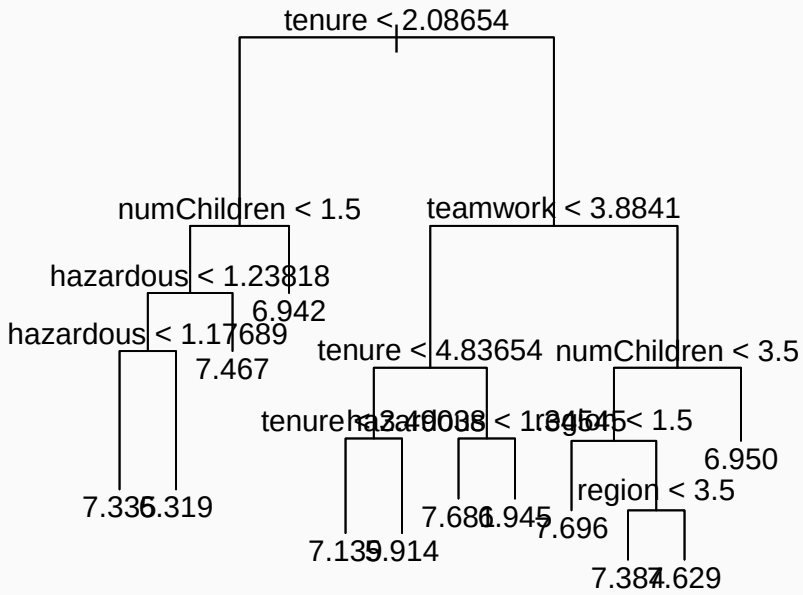
Bagging

High-Variance Trees

```
split <- sample(1:nrow(wages), nrow(wages)/2)

wage_tree <- tree(log(wage) ~ age + selfemp + numChildren + educ + school +
                      tenure + fullTime + unionized + firmSize + marstat +
                      urban + region  + hazardous + teamwork,
                  data = wages[split, ])

wage_tree2 <- tree(log(wage) ~ age + selfemp + numChildren + educ + school +
                      tenure + fullTime + unionized + firmSize + marstat +
                      urban + region  + hazardous + teamwork,
                  data = wages[-split, ])
```

High-Variance Trees





Bagging

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method.
- Recall that given a set of n independent observations $Z_1, \dots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- $\rightarrow$ averaging a set of observations reduces variance
- Practical constraint: we generally do not have access to multiple training sets.

Bagging

- Instead, we can bootstrap, by taking repeated samples (with replacement) from the (single) training data set.
    - More detail on this tomorrow!
- In this approach we generate B different bootstrapped training data sets.
- We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x.
- We then average all the predictions to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

- We do not prune these trees. Instead, we rely on the averaging process to prevent against overfitting

Bagging classification trees

- Take B samples (with replacement) from the training data
- Train a classification tree on each sample
- For each test observation, record the class predicted by each of the B trees
- Take a majority vote: the final prediction is the most commonly occurring class among the B predictions

Out-of-Bag Error Estimation

- The key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.
- On average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB.
- This will yield around $B/3$ predictions for the ith observation, which we average.
- $\longrightarrow$ our estimates of prediction error are based on out-of-sample predictions!
    - More on this tomorrow!

Random Forests

Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen from the full set of p predictors.
- The model is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$

Random Forests – WTF?

Why the hell does it help to have *fewer* predictors to predict our outcome?

- Imagine there is just one very strong predictor in the data, and a bunch of middling predictors
- In pretty much every bagged dataset, the strong predictor will be used at the top of the tree $\longrightarrow$ all the trees will look the same!
- Averaging highly correlated quantities doesn't reduce variance very much
- RF excludes the strong predictor sometimes, giving the other (also important) predictors more of a chance

Key point: Reducing the correlation in predictions across bootstrapped samples decreaes variance, which is good for *test set* error.

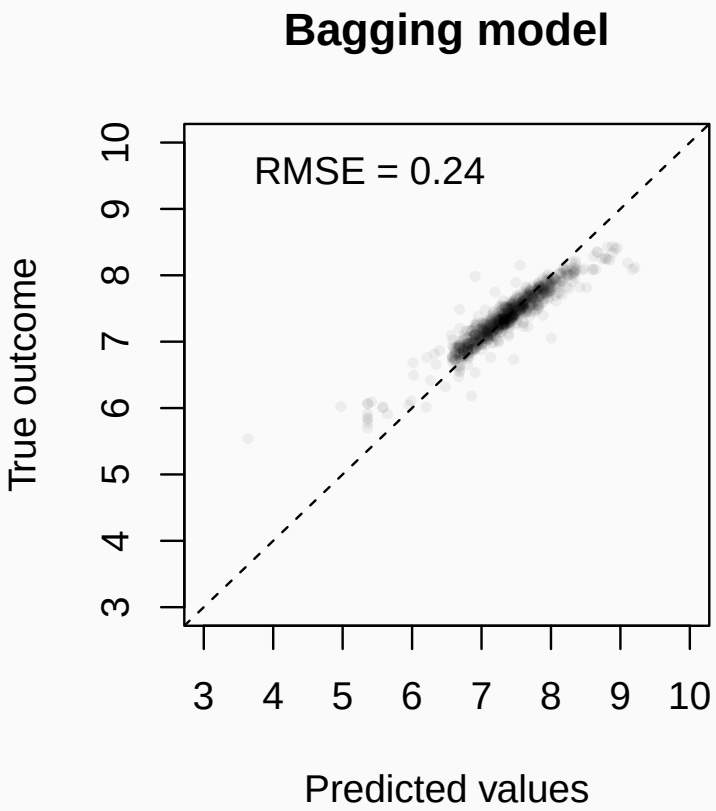
Bagging and Random Forests Application

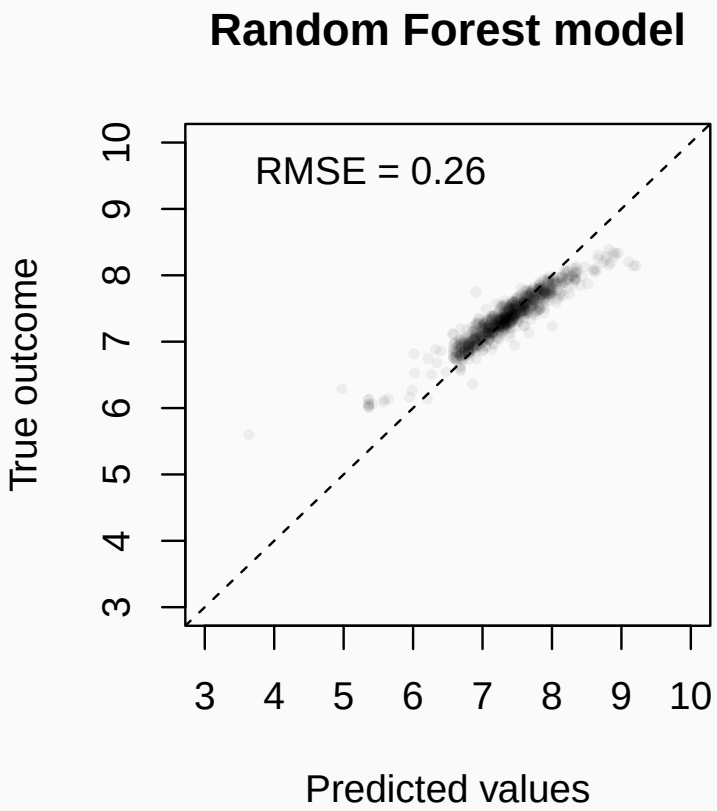
```
library(randomForest)

wage_bag <- randomForest(log(wage) ~ age + selfemp + numChildren + educ + school +
                         tenure + fullTime + unionized + firmSize + marstat +
                         urban + region + hazardous + teamwork,
                       data = wages,
                       na.action = na.omit,
                       mtry = 14,# Bagging: m = p
                       importance = TRUE)
```

Bagging and Random Forests Application

```
wage_rf <- randomForest(log(wage) ~ age + selfemp + numChildren + educ + school +
                        tenure + fullTime + unionized + firmSize + marstat +
                        urban + region + hazardous + teamwork,
                    data = wages,
                    na.action = na.omit,
                    mtry = 4, # RF: m = sqrt(p)
                    importance = TRUE)
```

Bagging and Random Forests Application



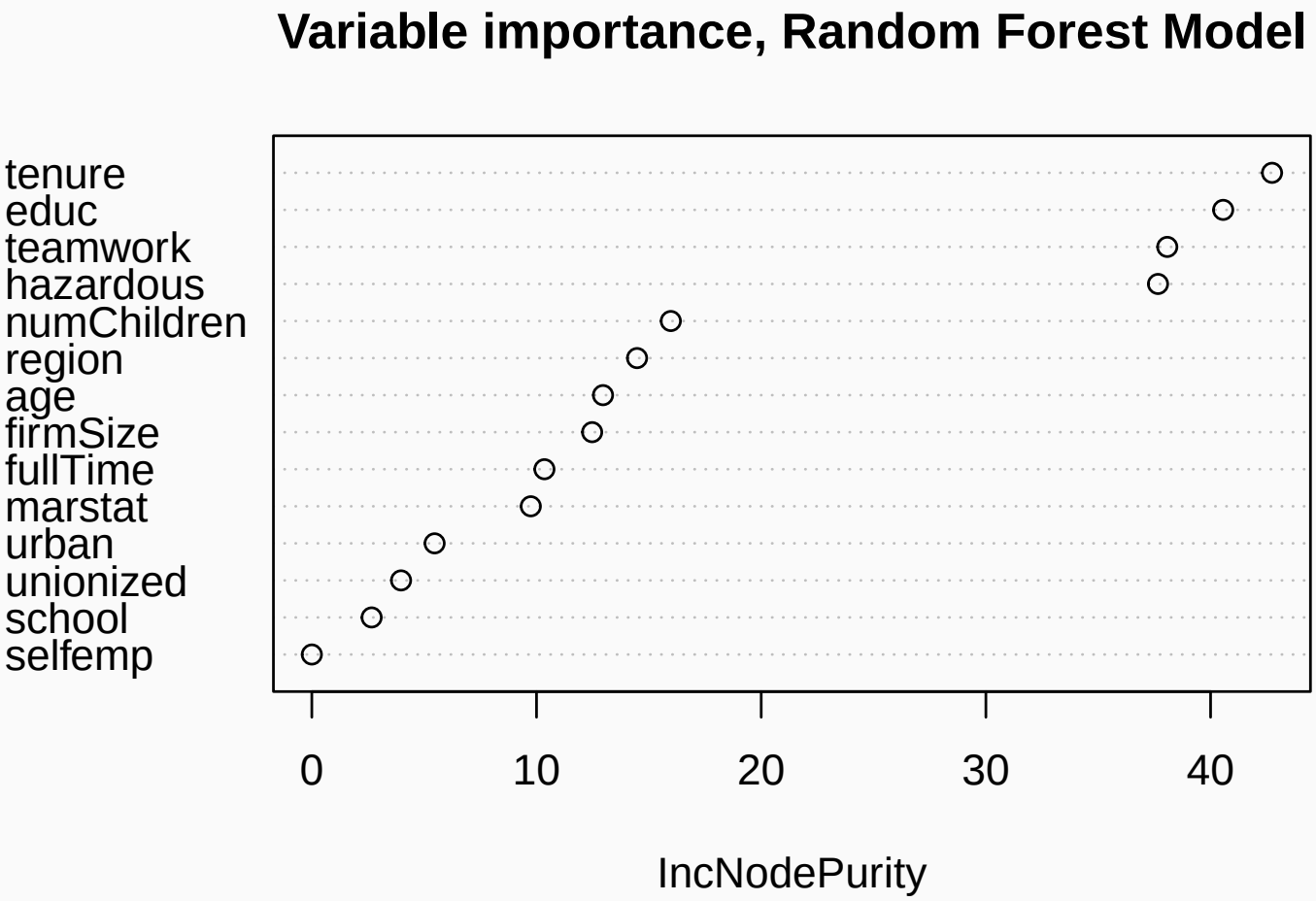


Variable importance measure

- We can't interpret either bagged of Random Forest models using a single tree
- Instead, we can calculate statistics that measure the importance of each feature the classification/prediction task
- For each variable, we measure how the following concepts change for each predictor, averaged across all B trees used in the model
    - 1. Decrease in prediction accuracy (MSE; error rate)
    - 2. Increase in node purity (RSS; Gini)
- Large values $\longrightarrow$ important predictor

Variable importance plot for the `wages` data.

```
varImpPlot(wage_rf, main = "Variable importance, Random Forest Model", type = 2)
```



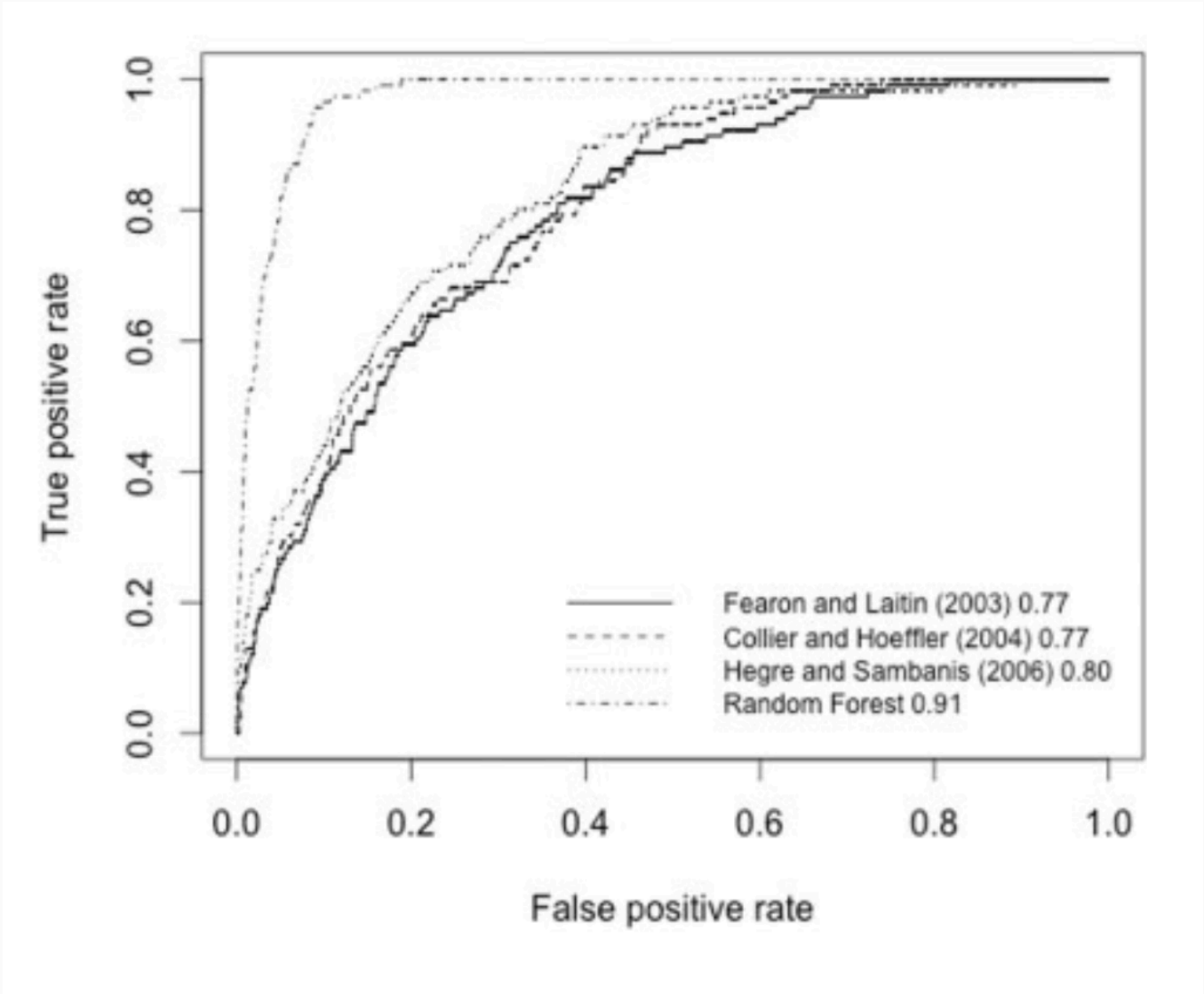
Application – Civil War Onset

Can we predict the onset of civil war?

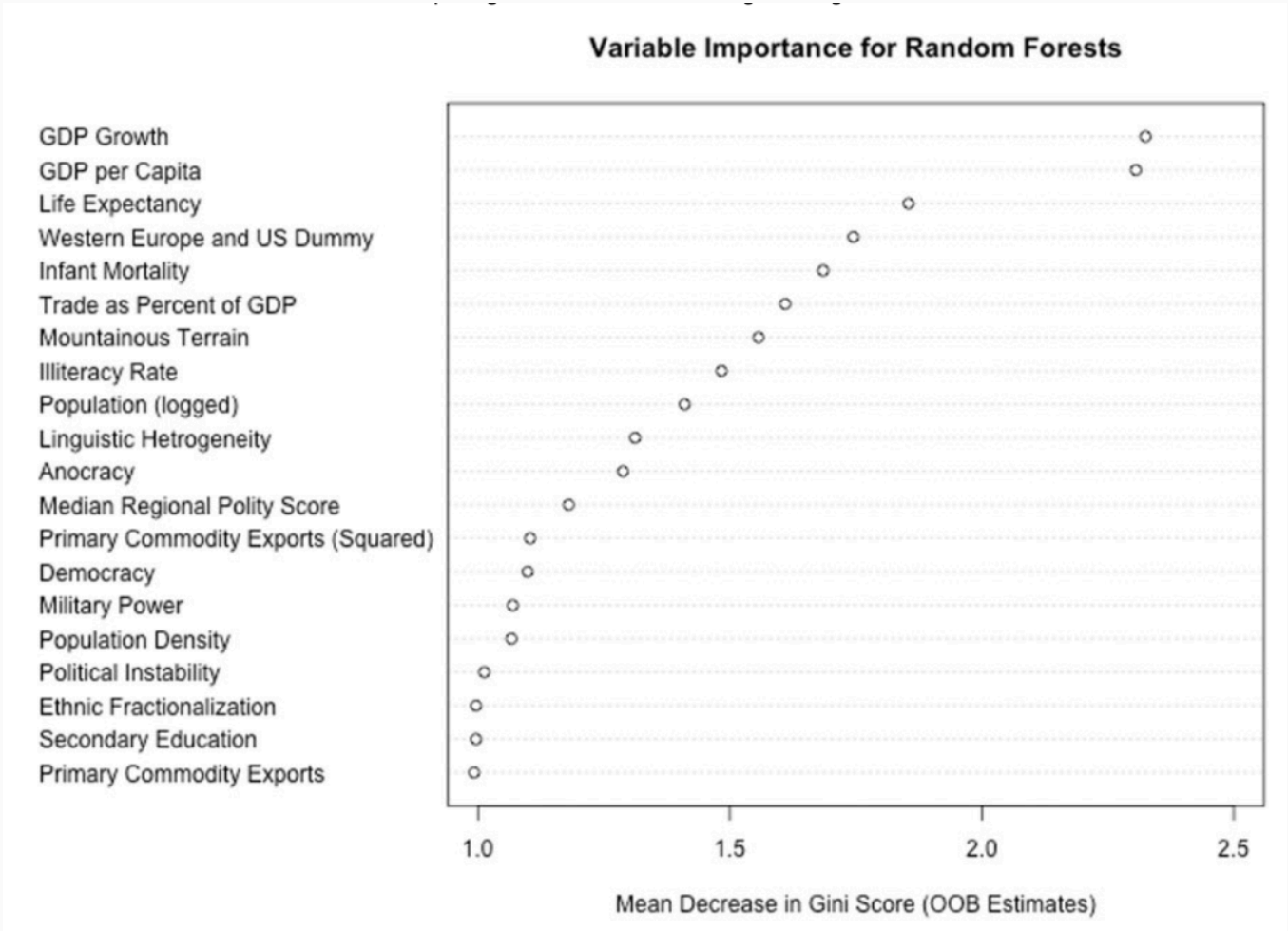
Prediction of the onset of civil war is a critical issue for policymakers who must try to anticipate conflicts and find ways to prevent or contain them. Existing approaches for predicting the start of wars are very poor, not least because they have tended to use quantitative methods that are unnecessarily restrictive. Muchlinski et al (2016) use Random Forest models to try to improve on the current state of the art.

- Y_{it} = 1 if a civil war started in country i in year t (very unbalanced, only one war year for every 100 peace years)
- $N = 7141$ country-year observations
- $X_{i,t}$ is a matrix of 88 predictor variables

Application – Civil War Onset



Application – Civil War Onset



Summary

- Decision trees are simple and interpretable models for regression and classification.
- However they are often not competitive with other methods in terms of prediction accuracy.
- Bagging and random forests are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- Random forest is among the (new) methods that are “democratising” machine learning.