Introduction

- For the remainder of the course, we will focus on nonparametric regression and classification
- The regression problem involves modeling how the expected value of a response y changes in response to changes in an explanatory variable x:

$$\mathbb{E}(y|x) = f(x)$$

- Linear regression, as its name implies, assumes a linear relationship; namely, that $f(x) = \beta_0 + \beta_1 x$

Parametric vs. nonparametric approaches

- This reduction of a complicated function to a simple form with a small number of unknown parameters is very similar to the parametric approach to estimation and inference involving the unknown distribution function
- The nonparametric approach, in contrast, is to make as few assumptions about the regression function f as possible
- Instead, we will try to use the data as much as possible to learn about the potential shape of f – allowing f to be very flexible, yet smooth

Simple local models

- One way to achieve this flexibility is by fitting a different, simple model separately at every point x_0 in much the same way that we used kernels to estimate density
- As with kernel density estimates, this is done using only those observations close to x_0 to fit the simple model
- As we will see, it is possible to extend many of the same kernel ideas we have already discussed to smoothly "blend" these local models to construct a smooth estimate $\hat{f}$ of the relationship between x and y

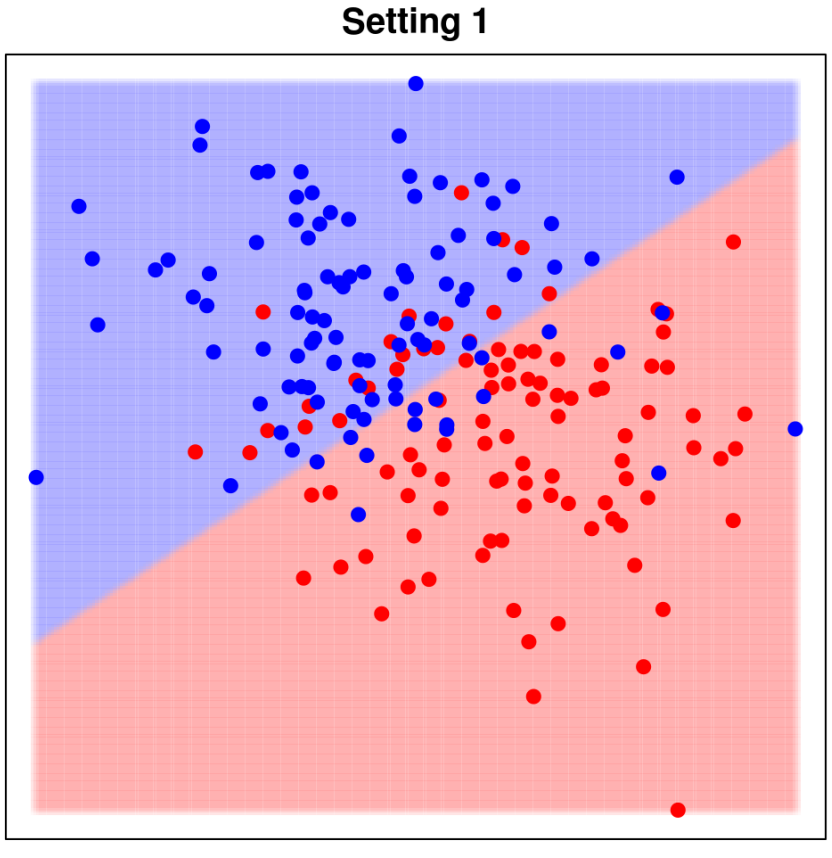
Introduction

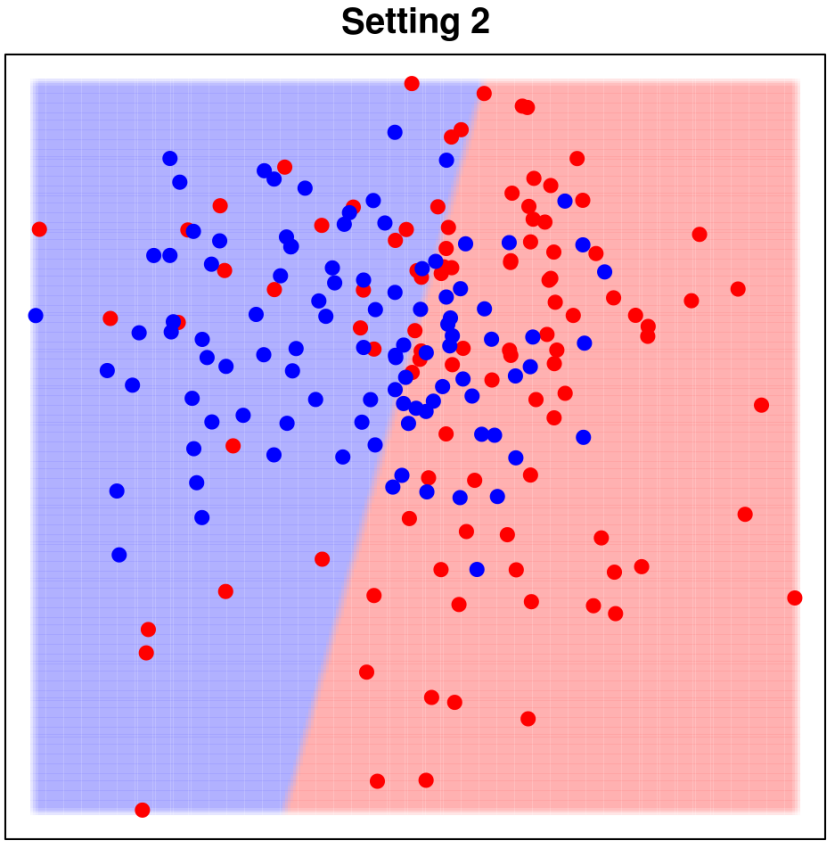
- Before we do so, however, let us get a general feel for the contrast between parametric and nonparametric classification by contrasting two simple, but very different, methods: the ordinary least squares regression model and the k-*nearest neighbor* prediction rule
- The linear model makes huge assumptions about the structure of the problem, but is quite stable
- Nearest neighbors is virtually assumption-free, but its results can be quite unstable
- Each method can be quite powerful in different settings and for different reasons

Simulation settings

- To examine which method is better in which setting, we will simulate data from a simple model in which y can take on one of two values: -1 or 1
- The corresponding x values are derived from one of two settings:
  - Setting 1: x values are drawn from a bivariate normal distribution with different means for $y = 1$ and $y = -1$
  - Setting 2: A mixture in which 10 sets of means for each class $(1, -1)$ are drawn; x values are then drawn by randomly selecting a mean from the appropriate class and then generating a random bivariate normal observation with that mean
- A fair competition between the two methods is then how well they do at predicting whether a future observation is 1 or -1 given its x values

Linear model results





Linear model remarks

- The linear model seems to classify points reasonably in setting 1
- In setting 2, on the other hand, there are some regions which seem questionable
- For example, in the lower left hand corner of the plot, does it really make sense to predict "blue" given that all of the nearby points are "red"?

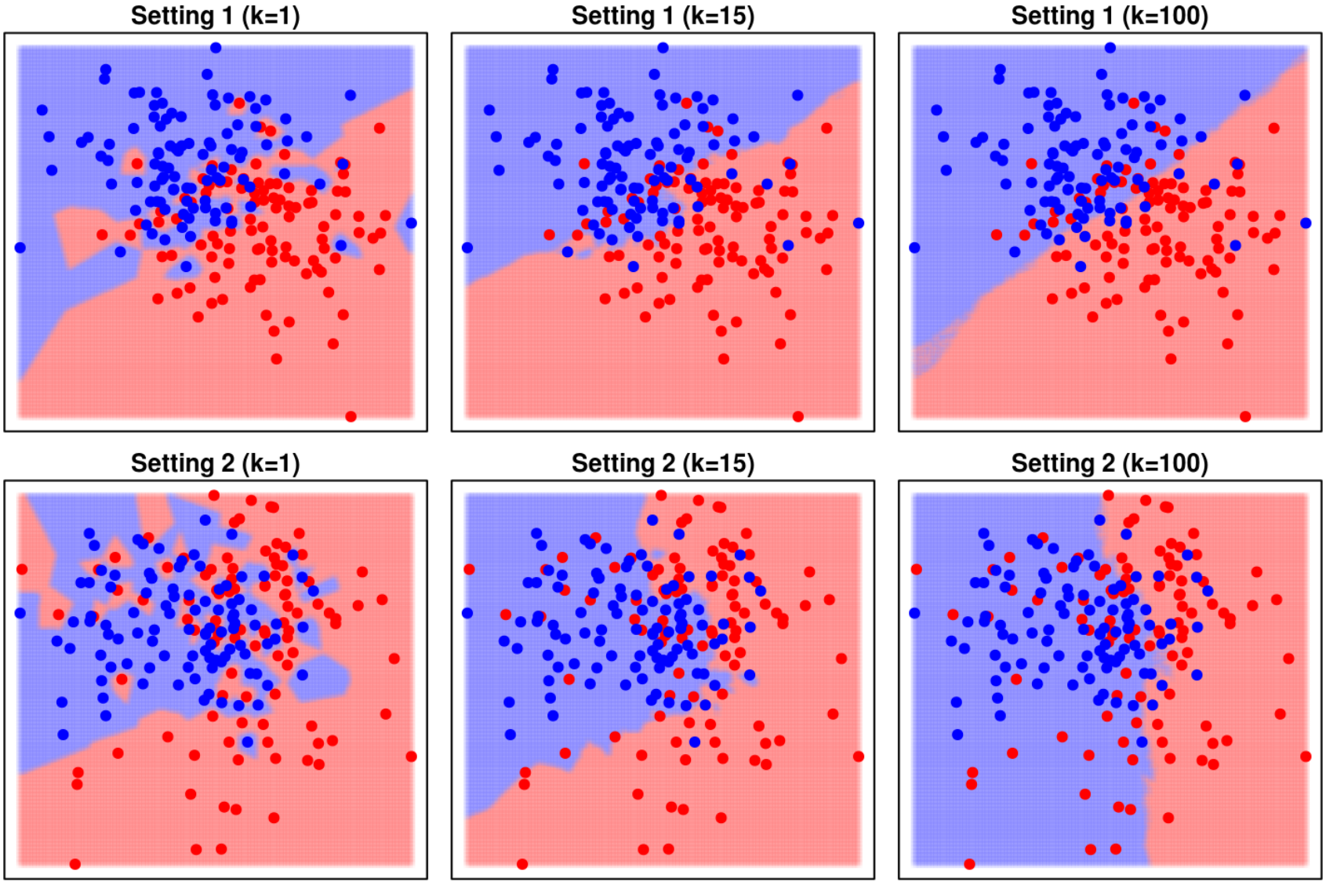
Nearest neighbors

- Consider then a completely different approach in which we don't assume a model, a distribution, a likelihood, or anything about the problem: we just look at nearby points and base our prediction on the average of those points
- This approach is called the *nearest-neighbor* method, and is defined formally as

$$\hat{y}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i,$$

where $N_k(\mathbf{x})$ is the neighborhood of $\mathbf{x}$ defined by its k closest points in the sample

Nearest neighbor results



Nearest neighbor remarks

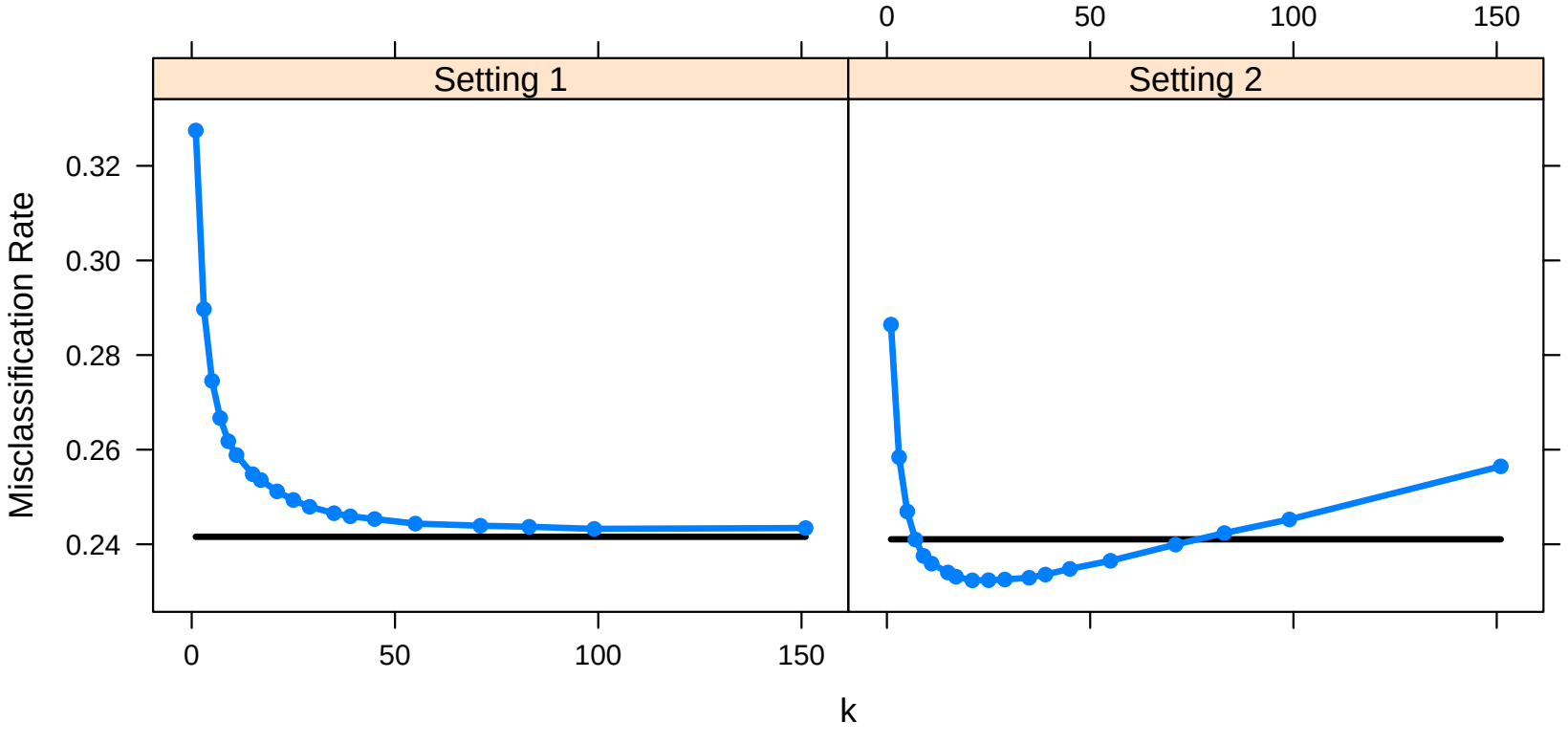
- Nearest neighbor seems not to perform terribly well in setting 1, as its classification boundaries are unnecessarily complex and unstable
- On the other hand, the method seemed perhaps better than the linear model in setting 2, where a complex and curved boundary seems to fit the data better
- Furthermore, the choice of k plays a big role in the fit, and the optimal k might not be the same in settings 1 and 2

Inference

- Of course, it is potentially misleading to judge whether a method is better simply because it fits the sample better
- What matters, of course, is how well its predictions generalize to new samples
- Thus, consider generating 100 data sets of size 200, fitting each model, and then measuring how well each method does at predicting 10,000 new, independent observations

Simulation results

Black line = least squares; blue line = nearest neighbors



Remarks

- In setting 1, linear regression was always better than nearest neighbors
- In setting 2, nearest neighbors was usually better than linear regression
- However, it wasn't *always* better than linear regression – when k was too big or too small, the nearest neighbors method performed poorly
- In setting 1, the bigger k was, the better; in setting 2, there was a "Goldilocks" value of k (about 25) that proved optimal in balancing the bias-variance tradeoff

Conclusions

Thus,

- Fitting an ordinary linear model is rarely the best we can do
- On the other hand, nearest-neighbors is rarely stable enough to be ideal, even in modest dimensions, unless our sample size is very large (recall the curse of dimensionality)

Conclusions (cont'd)

- These two methods stand on opposite sides of the methodology spectrum with regard to assumptions and structure
- The methods we will discuss for the remainder of the course involve bridging the gap between these two methods – making linear regression more flexible, adding structure and stability to nearest neighbor ideas, or combining concepts from both
- As with kernel density estimation, the main theme that emerges is the need to apply methods that bring the right mix of flexibility and stability that is appropriate for the data

Simple local models

- In the previous lecture, we examined a very simple local model: k-nearest neighbors
- Another simple local regression model is the local average:

$$\hat{f}(x_0) = \frac{\sum_i y_i I(|x_i - x_0| < h)}{\sum_i I(|x_i - x_0| < h)}$$

- However, both of these approaches have the disadvantage that they lead to discontinuous estimates — as an observation enters/leaves the neighborhood, the estimate changes abruptly

The Nadaraya-Watson kernel estimator

- As with kernel density estimators, we can eliminate this problem by introducing a continuous kernel which allows observations to enter and exit the model smoothly
- Generalizing the local average, we obtain the following estimator, known as the *Nadaraya-Watson kernel estimator*:

$$\hat{f}(x_0) = \frac{\sum_i y_i K_h(x_i, x_0)}{\sum_i K_h(x_i, x_0)},$$

  where $K_h(x_i, x_0) = K(\frac{x_i - x_0}{h})$ is the kernel, and if K is continuous, then so is $\hat{f}$
- As with kernel density estimates, we need to choose a bandwidth h, which controls the degree of smoothing

Expected loss and prediction error for regression

- Because it is customary to treat x as fixed in regression, instead of integrating over x to obtain the expected loss, we average over the observed values of x:

$$\mathbb{E}L(f, \hat{f}) = \frac{1}{n} \sum_i \mathbb{E}L(f(x_i), \hat{f}(x_i))$$

- The expected squared error loss is particularly convenient in regression, as it is directly related to the *expected prediction error*:

$$\mathrm{EPE} = \mathbb{E} \left\{ \frac{1}{n} \sum_i (Y_i - \hat{f}(x_i))^2 \right\},$$

where Y_i and $\hat{f}$ are independent variables

Bias-variance decomposition of EPE

- **Theorem:** At a given point $\mathbf{x}_0$,

$$\text{EPE} = \sigma^2 + \text{Bias}^2(\hat{f}) + \text{Var}(\hat{f}),$$

  where σ^2 is the variance of $Y|\mathbf{x}_0$
- Thus, expected prediction error consists of three parts:
  - *Irreducible error*: this is beyond our control and would remain even if we were able to estimate f perfectly
  - *Bias* (squared): the difference between $\text{E}\{\hat{f}(\mathbf{x}_0)\}$ and the true value $f(\mathbf{x}_0)$
  - *Variance*: the variance of the estimate $\hat{f}(\mathbf{x}_0)$

Relationship between expected loss and EPE

- Furthermore,

$$EPE = \mathbb{E}L(f, \hat{f}) + \sigma^2$$

- Thus, the expected prediction error and the expected loss are equal up to a constant
- This is attractive because prediction error is easy to evaluate via cross-validation

Cross-validation

- Specifically, we can estimate the expected prediction error with

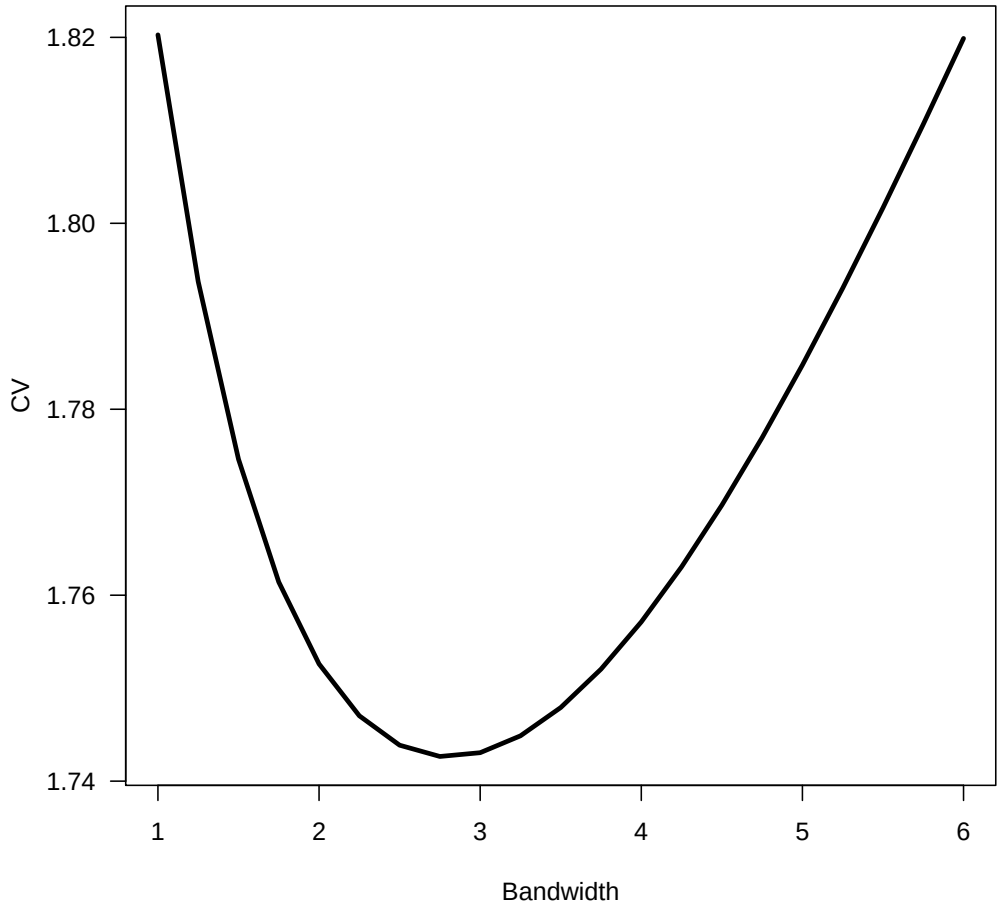
$$CV = \frac{1}{n} \sum_i \left\{ y_i - \hat{f}_{(-i)}(x_i) \right\}^2,$$

  where $\hat{f}_{(-i)}$ is the estimate of f obtained by omitting the pair $\{x_i, y_i\}$
- Furthermore, as we will see, one can obtain a closed form expression for the leave-one-out cross validation score above for any "linear smoother", without actually refitting the model n times

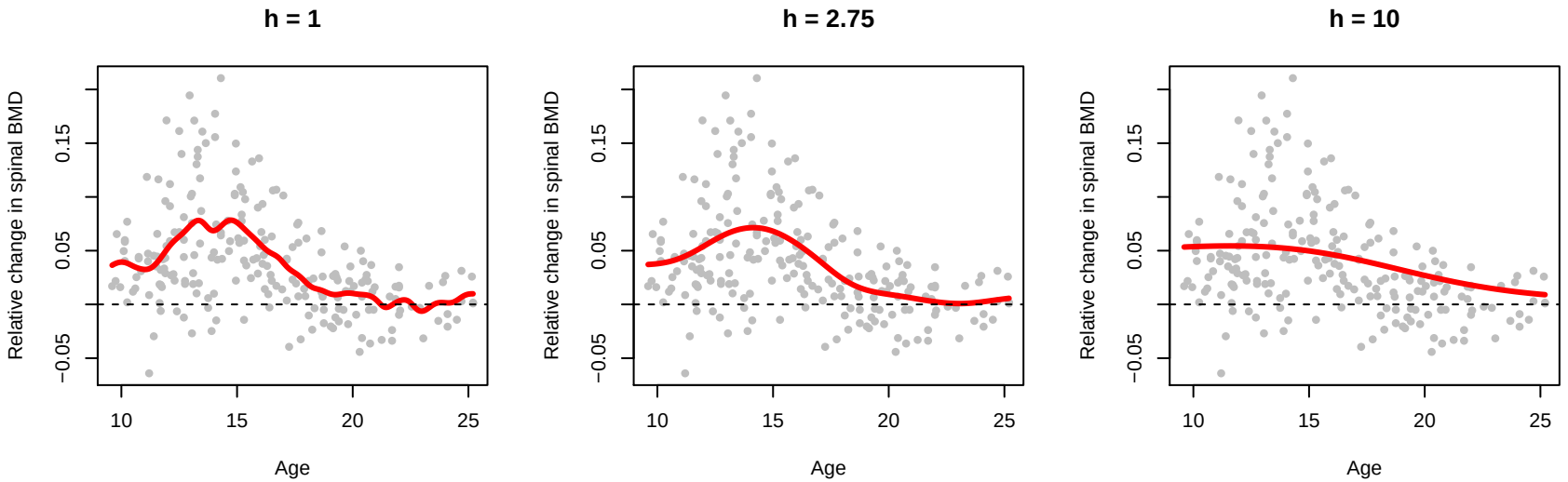
Bone mineral density data

- As an example of a real data set with an interesting change in $\mathbb{E}(y|x)$ as a function of x, we will look at a study of changes in bone mineral density in adolescents
- The outcome is the difference in spinal bone mineral density, taken on two consecutive visits, divided by the average of the two measurements
- Age is the average age over the two visits
- A person's bone mineral density generally increases until the individual is done growing, then remains relatively constant until old age

Cross-validation to choose bandwidth

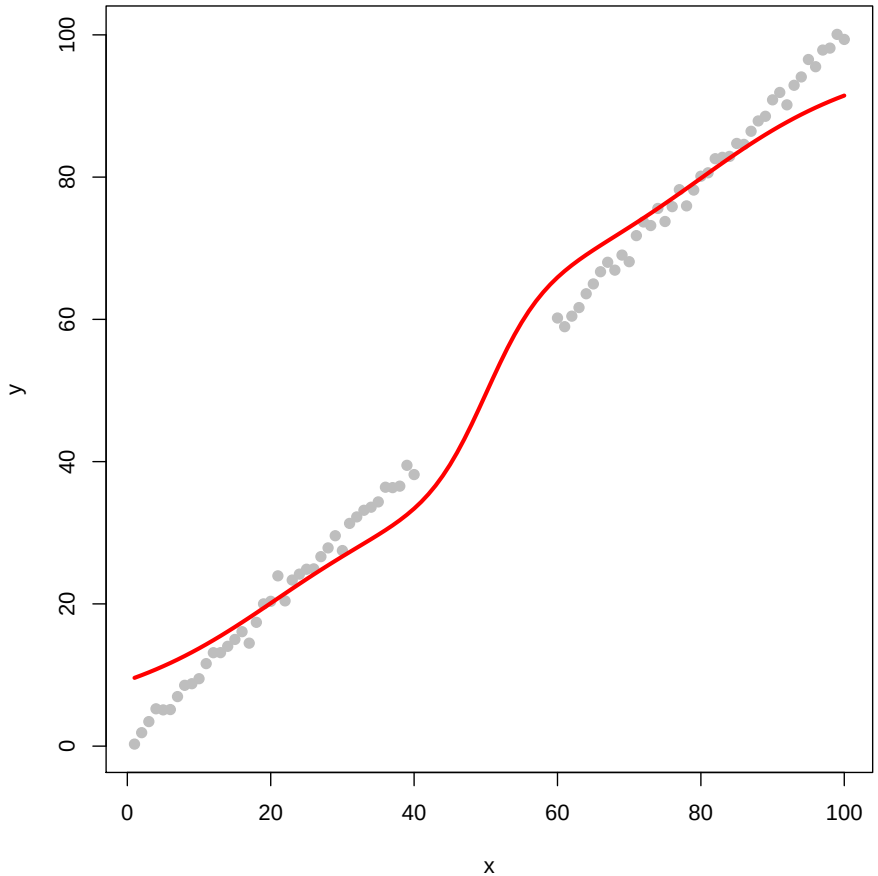


Estimates of the regression function



The problem with kernel weighted averages

Unfortunately, the Nadaraya-Watson kernel estimator suffers from bias, both at the boundaries and in the interior when the x_i's are not uniformly distributed:



Loess

- This arises due to the asymmetry effect of the kernel in these regions
- However, we can (up to first order) eliminate this problem by fitting straight lines locally, instead of constants
- In locally weighted regression, also known as *lowess* or *loess*, we solve a separate weighted least squares problem at each target point x_0:

$$(\hat{\alpha}, \hat{\beta}) = \text{arg min}_{\alpha,\beta} \sum_i K_h(x_0, x_i)(y_i - \alpha - x_i\beta)^2$$

- The estimate is then $\hat{f}(x_0) = \hat{\alpha} + x_0\hat{\beta}$

Loess is a linear smoother

- Let $\mathbf{X}$ denote the $n \times 2$ matrix with ith row $(1, x_i - x_0)$, and $\mathbf{W}$ denote the $n \times n$ diagonal matrix with ith diagonal element $w_i(x_0) = K_h(x_0, x_i)$
- Then,

$$\begin{aligned}\hat{f}(x_0) &= e_1'[\mathbf{X}'\mathbf{W}\mathbf{X}]^{-1}\mathbf{X}'\mathbf{W}\mathbf{y} \\ &= \sum_i l_i(x_0) y_i,\end{aligned}$$

  where $e_1 = (1, 0)' = (1, x_0 - x_0)'$ and it is important to keep in mind that both $\mathbf{X}$ and $\mathbf{W}$ depend implicitly on x_0
- Thus, our estimate is a linear combination of the y_i's; such estimates of f are called *linear smoothers*

Some facts about the linear weights

Homework: Show that the linear weights $\{l_i(x_0)\}$ defined on the previous slide satisfy

(a) $\sum_i l_i(x_0) = 1$ for all x_0

(b) $\sum_i l_i(x_0)(x_i - x_0) = 0$ for all x_0

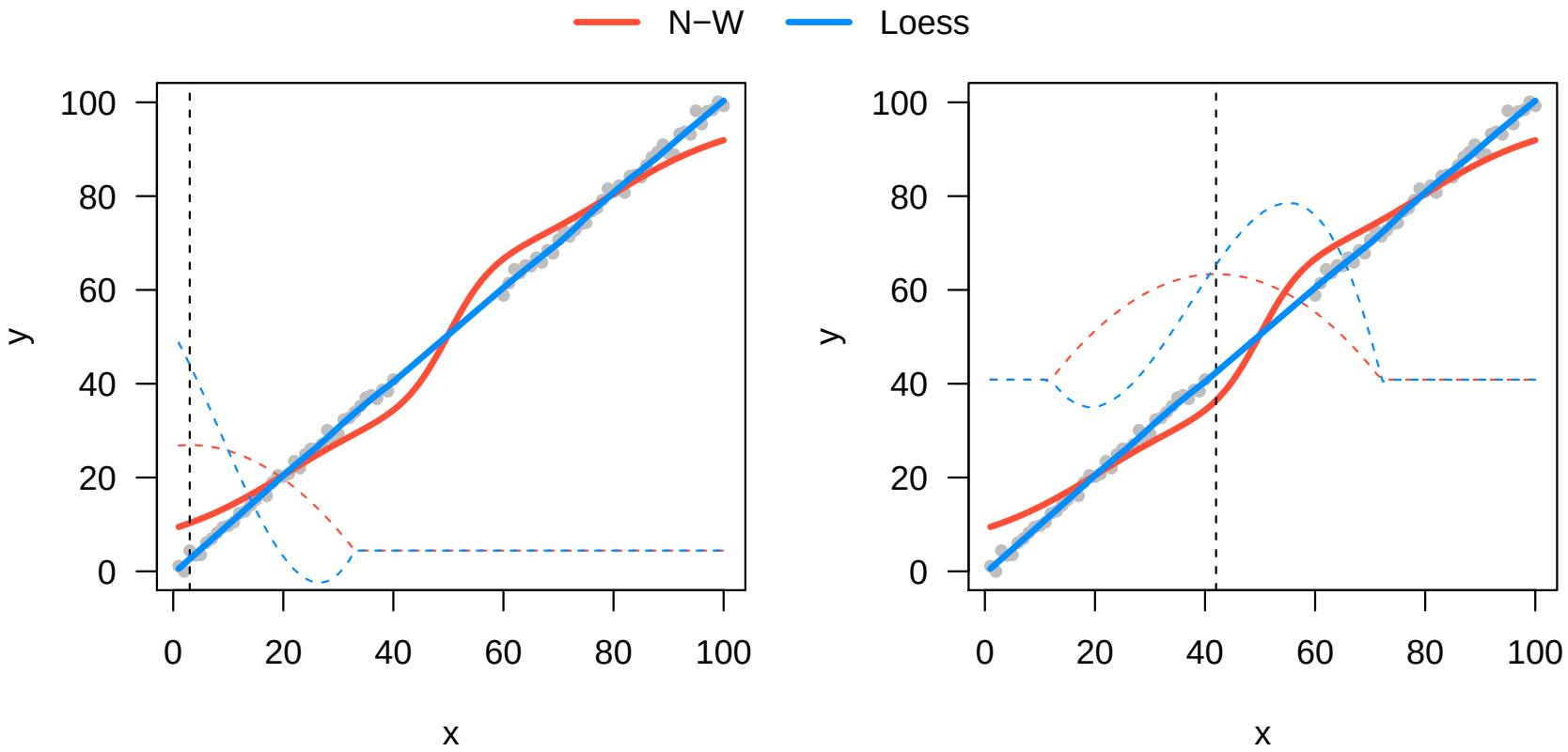
(c) If $K(x_i, x_0) = 0$, then $l_i(x_0) = 0$

(Note that property (a) ensures that the estimate preserves constant curves)

Effective kernels

- The loess approach is similar to the Nadaraya-Watson approach in that both are taking linear combinations of the responses $\{y_i\}$
- In loess, however, the weights $\{l_i(x_0)\}$ are constructed by combining both kernel weighting and least squares operations, forming what is sometimes called an *effective kernel* or *equivalent kernel*
- Before the development of loess, a fair amount of research focused on deriving adaptive modifications to kernels in order to alleviate the bias that we previously discussed
- However, local linear regression automatically modifies the kernel in such a way that this bias is largely eliminated, a phenomenon known as *automatic kernel carpentry*

Automatic kernel carpentry



Loess: Expectation and variance

- At any given target point x_0, $\hat{f}$ is a simple linear combination of random variables
- Thus,

$$\begin{aligned}
\mathbb{E}\hat{f}(x_0) &= \sum_i l_i(x_0) f(x_i) \\
\mathbb{V}\hat{f}(x_0) &= \sigma^2 \sum_i l_i(x_0)^2 \\
&= \sigma^2 \left\| l(x_0) \right\|^2,
\end{aligned}$$

where $\sigma^2 = \mathbb{V}(y)$

Bias: Loess vs. Nadaraya-Watson

- **Theorem:** Suppose that f is continuously differentiable up to second order and that $K(x, x_0) = 0$ if $|x - x_0| > h$. Then

$$\text{Loess: } \text{Bias}\{f(x_0)\} = O(h^2)$$

$$\text{N-W: } \text{Bias}\{f(x_0)\} = f'(x_0) \sum_i w_i(x_i - x_0) + O(h^2),$$

  where $w_i = K_h(x_i, x_0) / \sum_j K_h(x_j, x_0)$

- The leading term for the bias of the Nadaraya-Watson estimator is referred to as *design bias*; note that it is not present for loess estimators
- In other words, the automatic kernel carpentry that loess performs naturally eliminates design bias, and the resulting estimator is free of bias up to second order

The smoothing matrix

- Recall that loess is a linear smoother; thus,

$$\hat{\mathbf{f}} = \mathbf{L}\mathbf{y},$$

  where $\mathbf{L}$ is called the *smoothing matrix* whose elements consists of the linear weights $l_j(x_i)$
- Having our predictions take on this linear form greatly simplifies leave-one-out cross-validation

Closed form for leave-one-out cross-validation

- **Homework:** Show that

$$\frac{1}{n}\sum_i \left\{y_i - \hat{f}_{(-i)}(x_i)\right\}^2 = \frac{1}{n}\sum_i \left(\frac{y_i - \hat{f}_i}{1 - l_{ii}}\right)^2$$

- Thus, we have a closed form solution for the leave-one-out cross-validation score that can be obtained from a single fit (*i.e.*, no need to refit anything)

Generalized cross-validation

- An alternative to cross-validation is to replace the individual l_{ii}'s by their average $n^{-1}\sum_i l_{ii} = \nu/n$, where $\nu = \mathrm{tr}(\mathbf{L})$
- This approach is called *generalized cross-validation*:

$$\mathrm{GCV} = \frac{1}{n}\sum_i \left(\frac{y_i - \hat{f}_i}{1-\nu}\right)^2$$

- GCV is equal to CV if all the l_{ii}'s are equal; otherwise, they will be different, although usually quite close

Generalized cross-validation (cont'd)

- Note that, for $x \approx 0$, $1/(1-x)^2 \approx 1 + 2x$; thus,

$$GCV \approx \frac{1}{n}\sum_i (y_i - \hat{f}_i)^2 + \frac{2\hat{\sigma}^2\nu}{n},$$

  where $\hat{\sigma}^2 = n^{-1}\sum_i (y_i - \hat{f}_i)^2$
- If we multiply by n and divide by $\hat{\sigma}^2$, we have that GCV is approximately proportional to $-2\text{loglik} + 2\nu$, the AIC of the fit (treating $\hat{\sigma}^2$ as a known constant)
- Note that, in this approximation, $\nu = \text{tr}(\mathbf{L})$ acts as the *effective degrees of freedom* in the fit

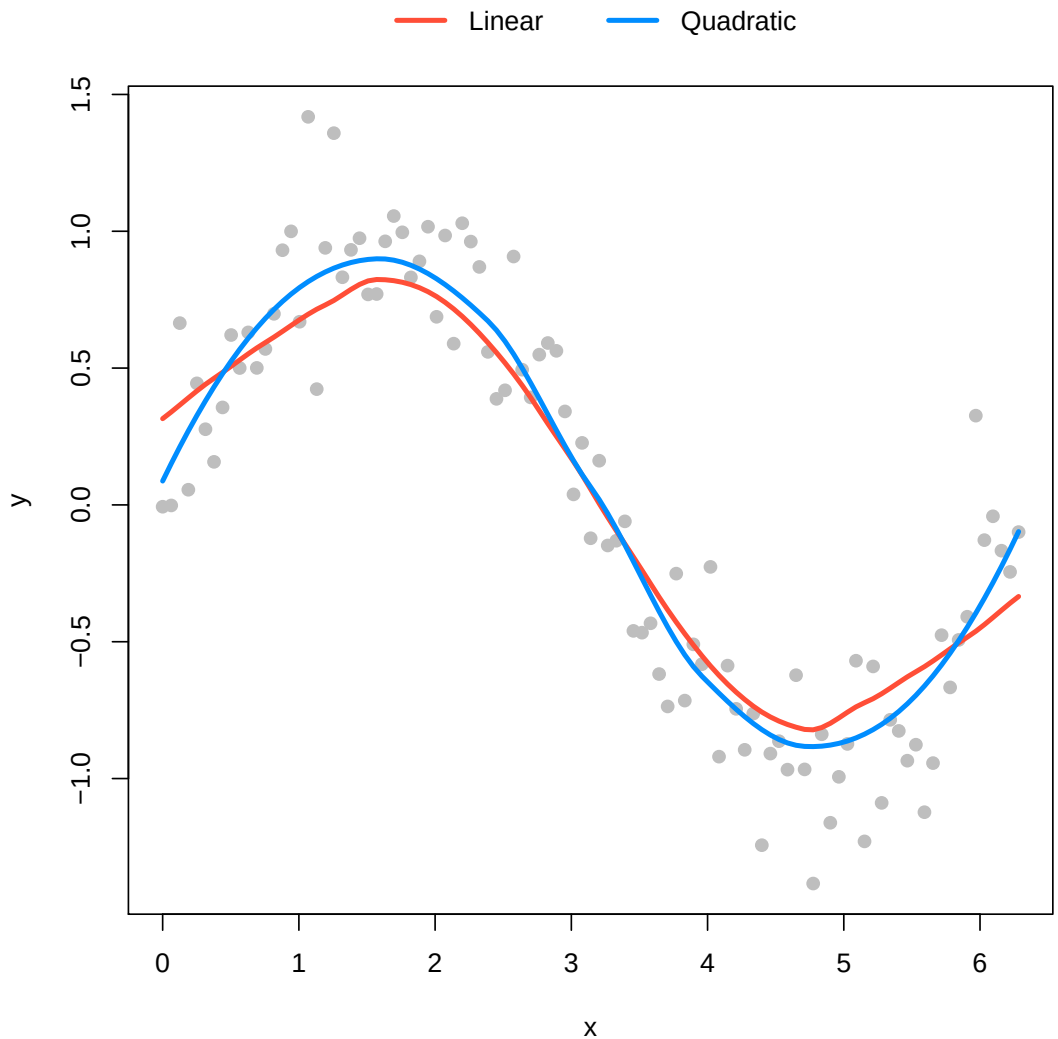
Effective degrees of freedom

- Note that the smoothing matrix is quite similar to the *projection matrix* or *hat matrix* from linear regression ($\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$), for which $\hat{\mathbf{f}} = \mathbf{H}\mathbf{y}$
- In linear regression, $\mathrm{tr}(\mathbf{H})$ is equal to the degrees of freedom; for linear smoothers, $\mathrm{tr}(\mathbf{L})$ defines the effective degrees of freedom (this analogy can be further justified in a number of ways)

Local polynomials

- Of course, one may ask: why stop at local linear regression? Why not add a quadratic term?
- It is not difficult to fit quadratic or even higher order terms: we simply let $\mathbf{X}$ have rows $\left[1, x_i - x_0, \ldots, (x_i - x_0)^d\right]$, where d is the degree of the polynomial
- The weight matrix $\mathbf{W}$, determined entirely by the kernel, remains the same, and we solve separate linear systems of equations for each target point $\mathbf{x}_0$
- By the same mechanism as our earlier theorem, it is straightforward to establish that the bias of a local polynomial fit is $O(h^{d+1})$

Bias due to local linear fitting



Local linear versus local quadratic fitting

- As the figure on the previous slide indicates, local linear models tend to be biased in regions of high curvature, a phenomenon referred to as "trimming the hills and filling in the valleys"
- Higher-order local polynomials correct for this bias, but at the expense of increased variability
- The conventional wisdom on the subject of local linear versus local quadratic fitting says that:
  - Local linear fits tend to be superior at the boundaries
  - Local quadratic fits tend to be superior in the interior
  - Local fitting to higher order polynomials is possible in principle, but rarely necessary in practice

Constant vs. adaptive h

- Our discussion of kernels has focused on keeping the half-width h constant
- An alternative approach is to use a nearest-neighbors type of kernel, in which h changes as a function of x_0 so that the number of points inside $(x_0 - h, x_0 + h)$ remains constant (as we will see, this is the default approach in `R`)
- The smoothing parameter in loess can therefore be made readily interpretable as the fraction of the sample size used in constructing the local fit at any point x_0

The `loess` function

- In `R`, local linear regression is implemented through the `loess` function, which uses a formula interface similar to that of other regression functions:

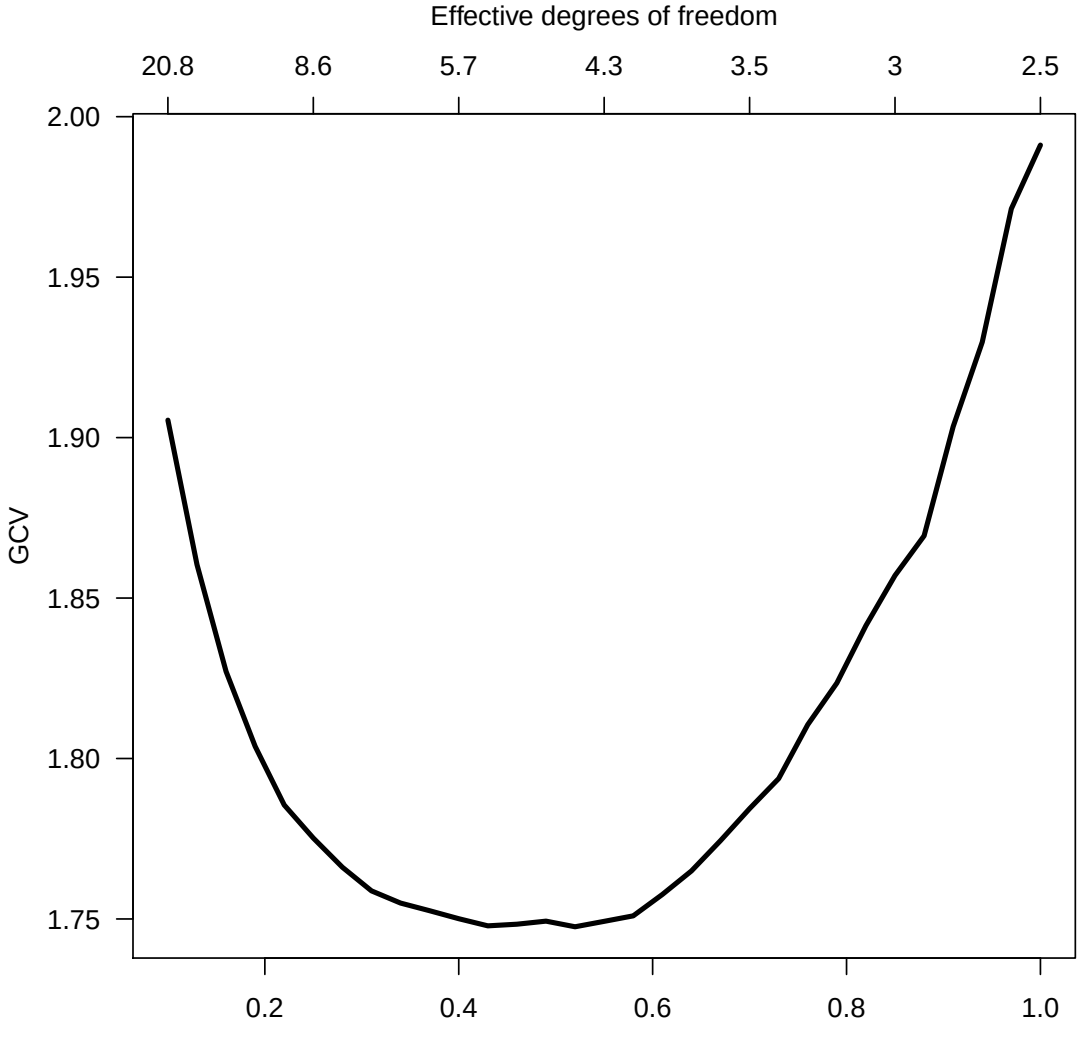
```
fit <- loess(spnbmd~age, m, span=0.3, degree=1)
```

- The two key options are
  - `span`: this is the smoothing parameter which controls the bias-variance tradeoff
  - `degree`: this lets you specify local constant regression (*i.e.*, the Nadaraya-Watson estimator, `degree=0`), local linear regression (`degree=1`), or local polynomial fits (`degree=2`, the default)

The `span` argument

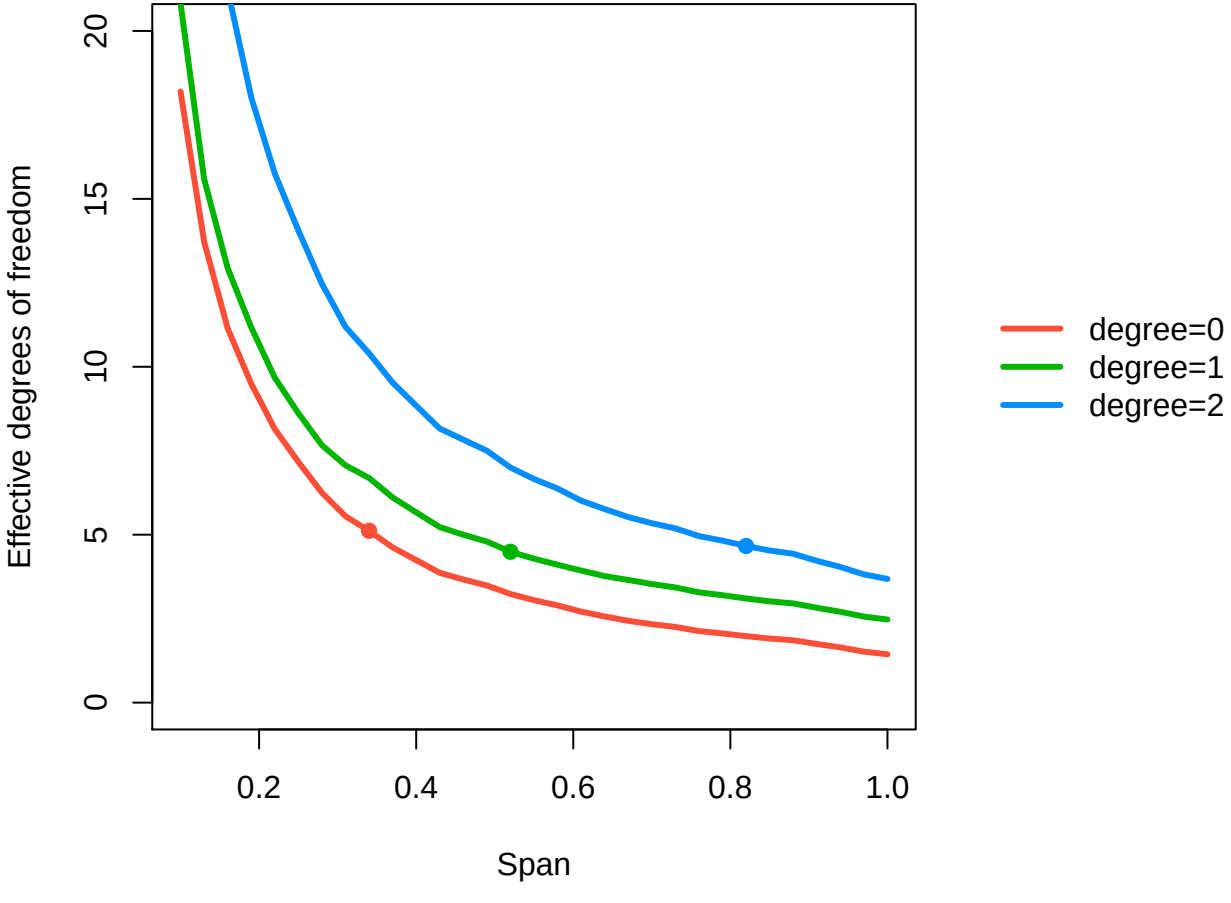
- Unlike `density`, `loess` does not allow you to choose your own kernel; only the tricube kernel is implemented, and `span` refers to the proportion of the observations $\{x_i\}$ within its compact support
- Also unlike `density`, the kernel in `loess` is adaptive
- Thus, specifying `span=0.2` means that the bandwidth of the kernel at x_0 is made just wide enough to include 20% of the x_i values
- The default value is 0.75, but this is just an ad-hoc suggestion; by no means is this always a good choice for the smoothing parameter

Selection of smoothing parameter

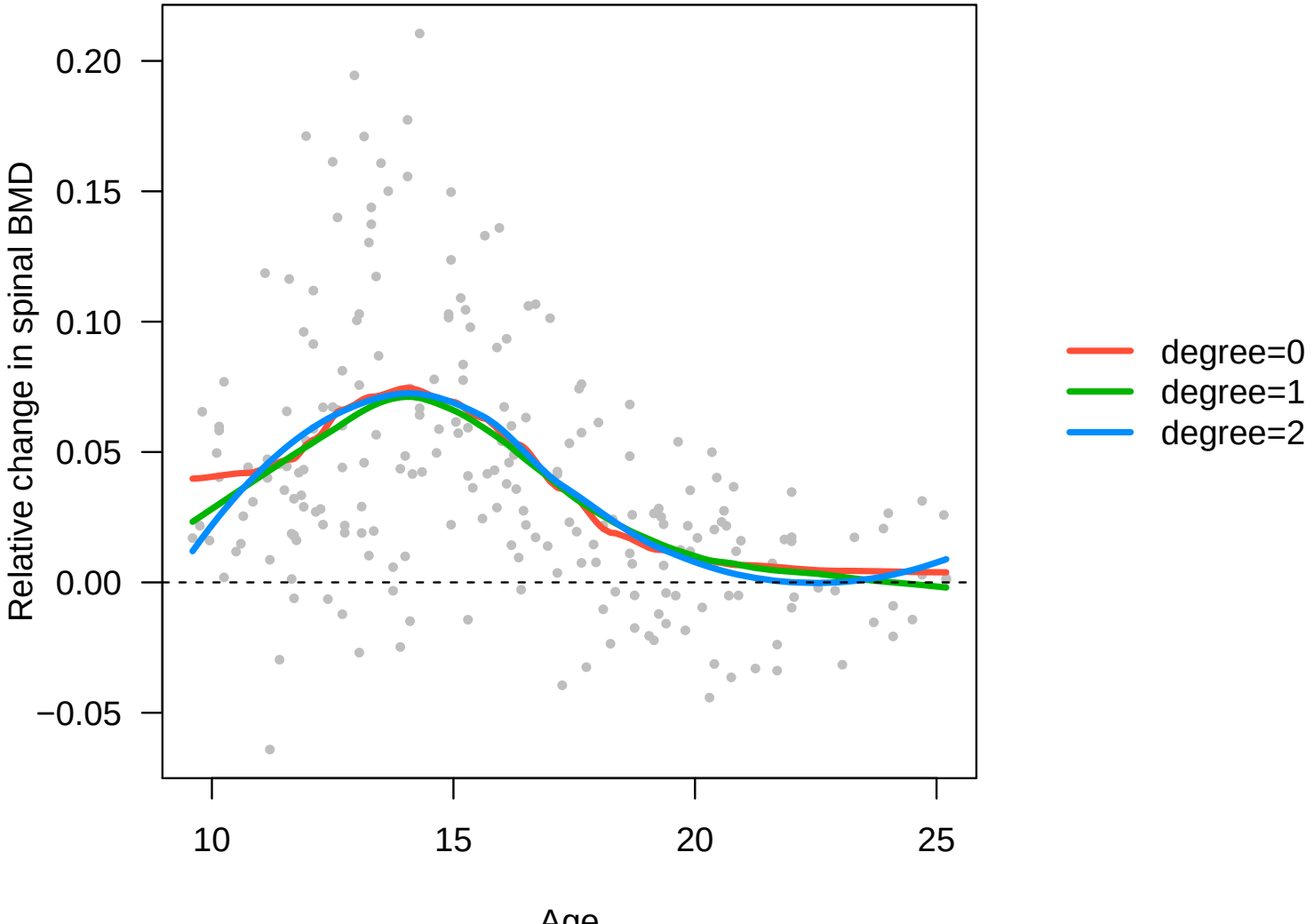


Effective degrees of freedom versus span

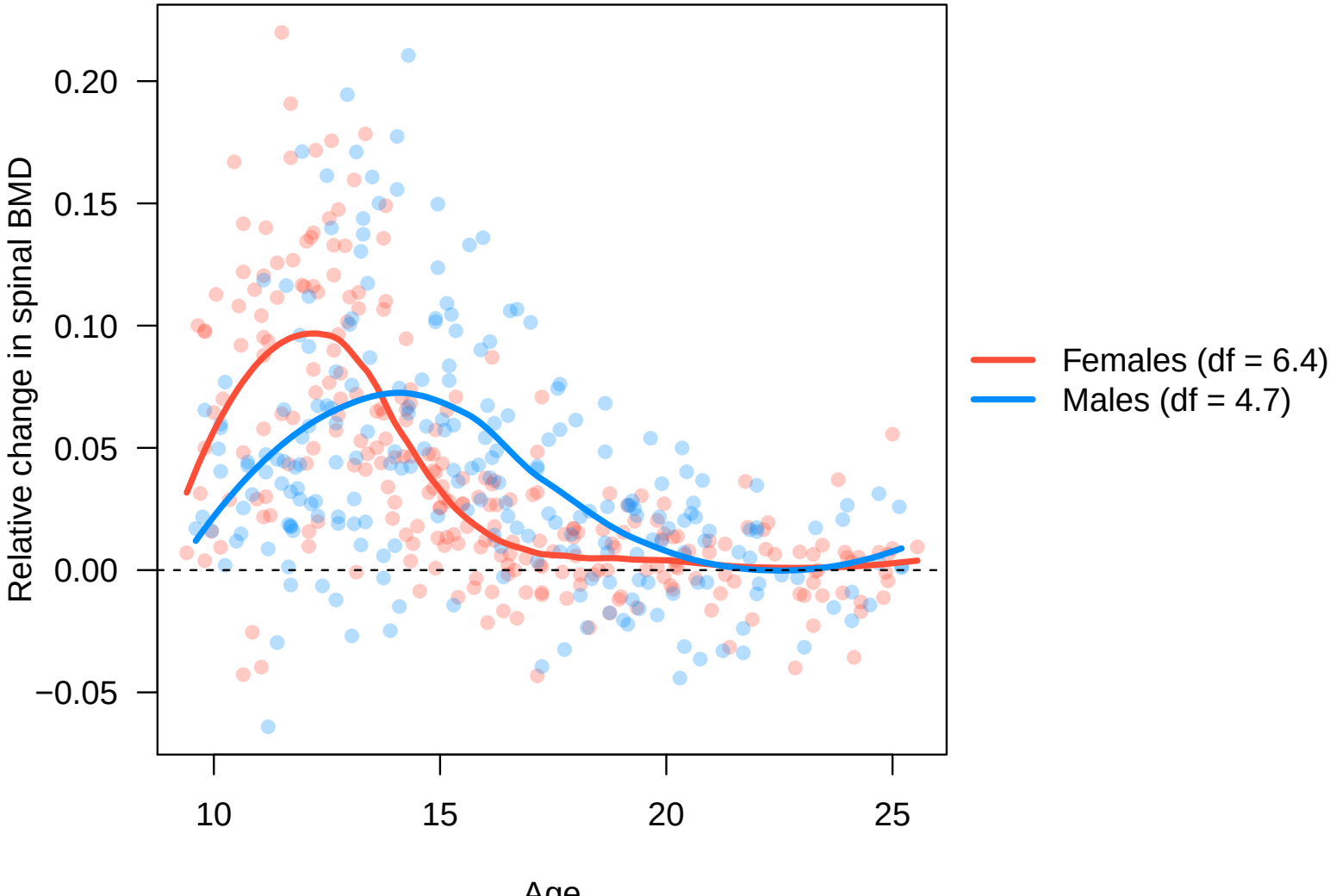
Dots indicate optimal smoothing, as chosen by GCV:



Optimal fits for the bone mineral density data



Bone mineral density data – males versus females



Other implementations: `gam` and `locfit`

- The `loess` function provides a convenient interface to fitting local polynomials, but carries out no inference (confidence bands, hypothesis tests, etc.)
- Inferential results are provided by the `gam` and `locfit` packages
- Each package has a somewhat different perspective and provides different tools, so they are both worth knowing about

Estimation of σ^2

- Note that

$$\mathbb{E}\sum_i (y_i - \hat{f}_i)^2 = \sigma^2(n - 2\nu + \tilde{\nu}) + \mathbf{b}'\mathbf{b},$$

where

$$\begin{aligned}\mathbf{b} &= \mathbb{E}(\mathbf{y}) - \mathbb{E}(\hat{\mathbf{f}}),\\ \nu &= \mathrm{tr}(\mathbf{L}),\\ \tilde{\nu} &= \mathrm{tr}(\mathbf{L}'\mathbf{L})\end{aligned}$$

- The bias term presents a problem, as it depends on the true regression function $\mathbf{f}$

Estimation of σ^2 (cont'd)

- However, if n is reasonably large compared with ν and $\tilde{\nu}$ and the bandwidth reasonably small, the effect of the bias term is negligible, and the following is a nearly unbiased estimator for σ^2:

$$\hat{\sigma}^2 = \frac{\sum_i (y_i - \hat{f}_i)^2}{n - 2\nu + \tilde{\nu}}$$

- The quantity $2\nu - \tilde{\nu}$ is known as the *equivalent number of parameters*, by analogy with linear regression

The `locfit` function

- This estimate, along with ν and $\tilde{\nu}$, are provided by the `locfit` package
- The basic syntax of model fitting with `locfit` is as follows:

  ```
  fit <- locfit(spnbmd~lp(age, nn=.7, deg=2))
  ```

  where `lp` controls the local polynomial which is fit to the data
- Just like `loess`, there is a `nn` parameter (analogous to `span`), which adaptively determines the bandwidth by setting the number of points in the neighborhood of x_0 equal to `nn`
- There is also a `deg` parameter, which controls the degree of the local polynomial (like `loess`, the default is 2)

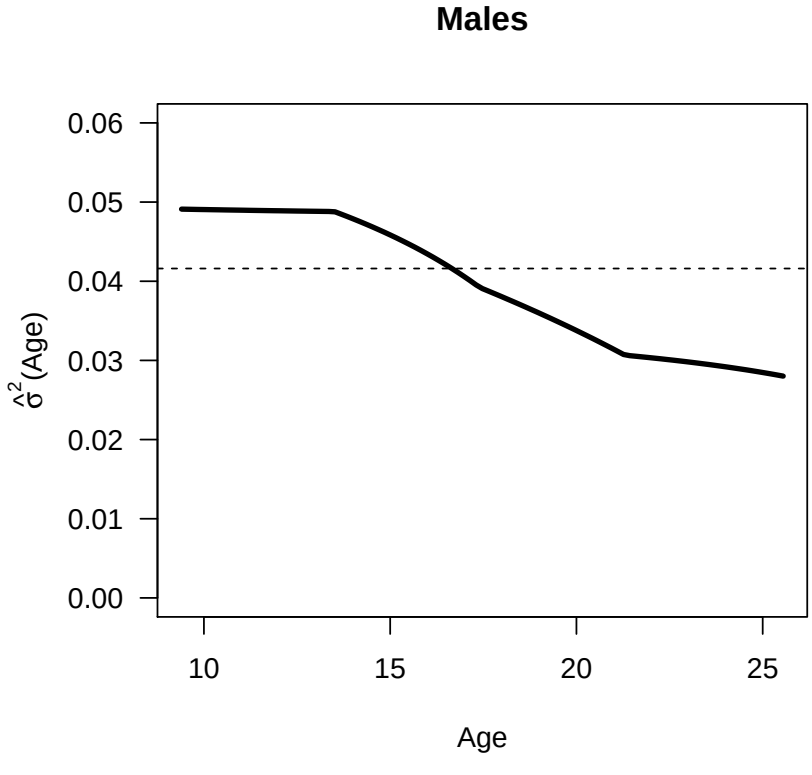
Obtaining $\hat{\sigma}^2$ from `locfit`

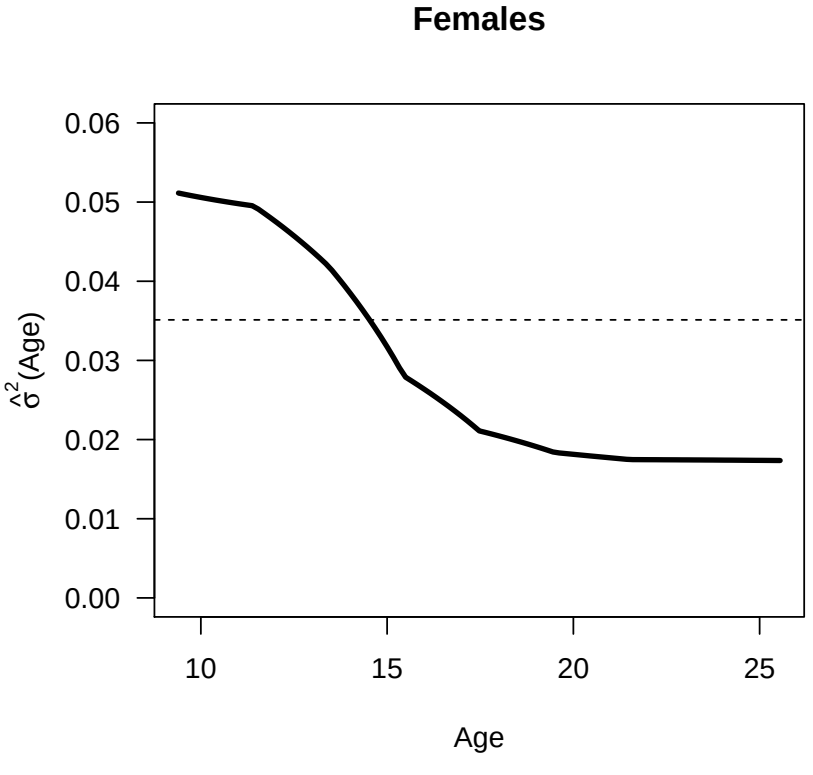
- The fitted object returned by `locfit` contains a (rather poorly documented) component called `dp` which contains information about the fit:
  - `df1`: ν
  - `df2`: $\tilde{\nu}$
  - `lk`: the log-likelihood (actually, $-\mathrm{RSS}/2$)
  - `rv`: $\hat{\sigma}^2$
- Note that `-2*lk/(n-2*df1+df2)` equals `rv`
- For the BMD data, $\hat{\sigma} = 0.42$ for the males and $\hat{\sigma} = 0.35$ for the females

Estimation of σ^2 (cont'd)

- The preceding estimate assumes homoskedasticity (constant variance)
- An alternative, given that we are fitting local linear models, is to estimate $\hat{\sigma}^2(x_0)$ from the usual linear regression estimate for the model fitted at x_0
- The information needed to calculate these estimates is available with `predict.locfit` (see the accompanying code for details)

Local variance estimates





Pointwise inference

- Recall that

$$\mathbb{E}\hat{f}(x_0) = \sum_i l_i(x_0) f(x_0)$$
$$\mathbb{V}\hat{f}(x_0) = \sigma^2 \|l(x)\|^2,$$

- One method of constructing pointwise confidence intervals, then, is via

$$\hat{f}(x_0) \pm z_{\alpha/2}\hat{\sigma} \|l(x_0)\|,$$

The bias problem

- However, as we have remarked several times, $\hat{f}$ is not an unbiased estimate of f:

$$\frac{\hat{f} - f}{\mathrm{SE}} \sim Z + \frac{\mathrm{bias}}{\mathrm{SE}},$$

  where $Z \sim N(0, 1)$
- Thus, usual normal-theory methods for confidence intervals will result in confidence intervals for $\bar{f} = \mathbb{E}(\hat{f})$, not for f itself
- Generally, however, the bias term is just ignored and we just accept the fact that the interval is technically an interval for $\bar{f}$, not f

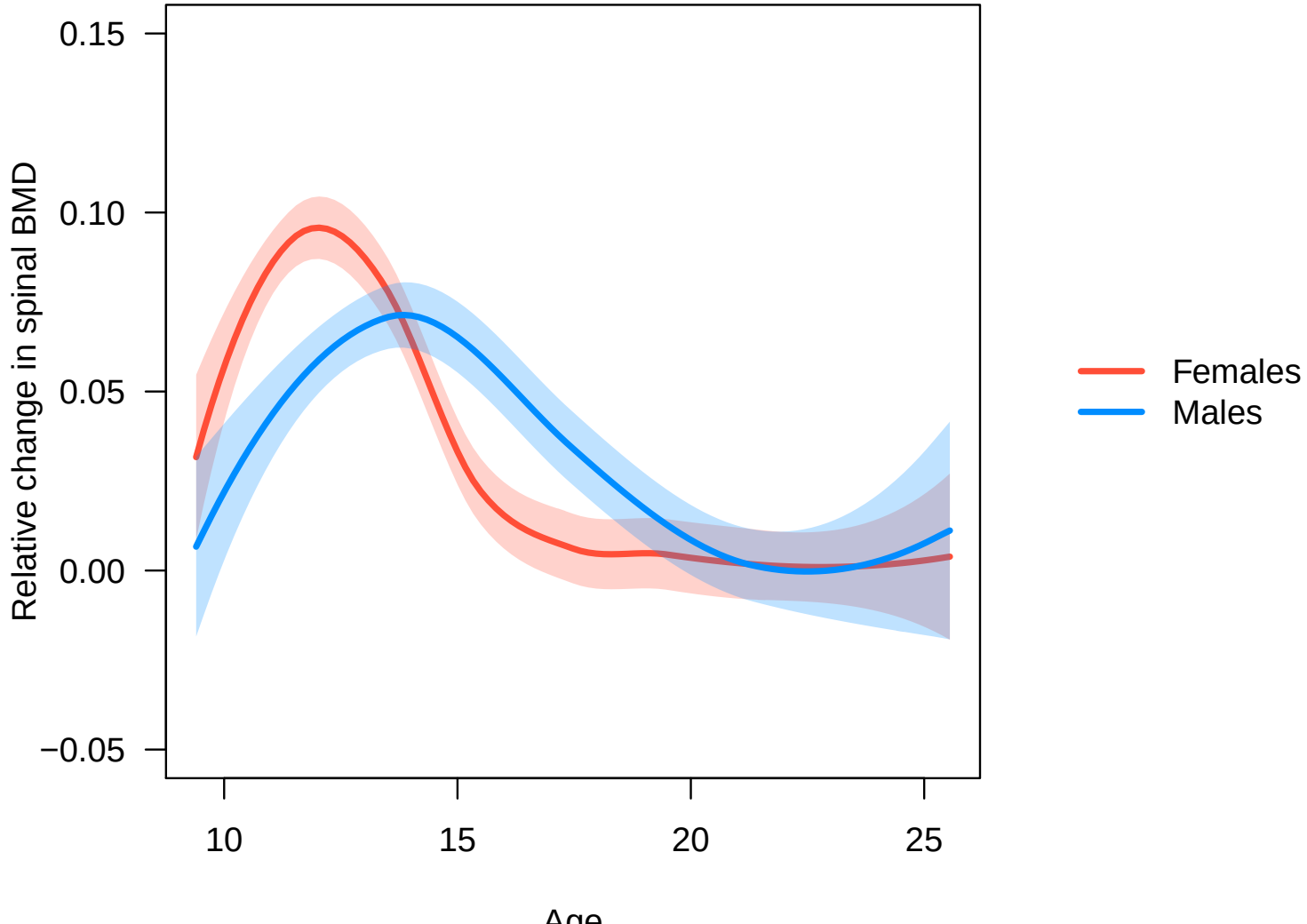
Global confidence bands

- It is also possible to obtain simultaneous confidence bands across the entire range of x
- The details are fairly complicated and rest on representing

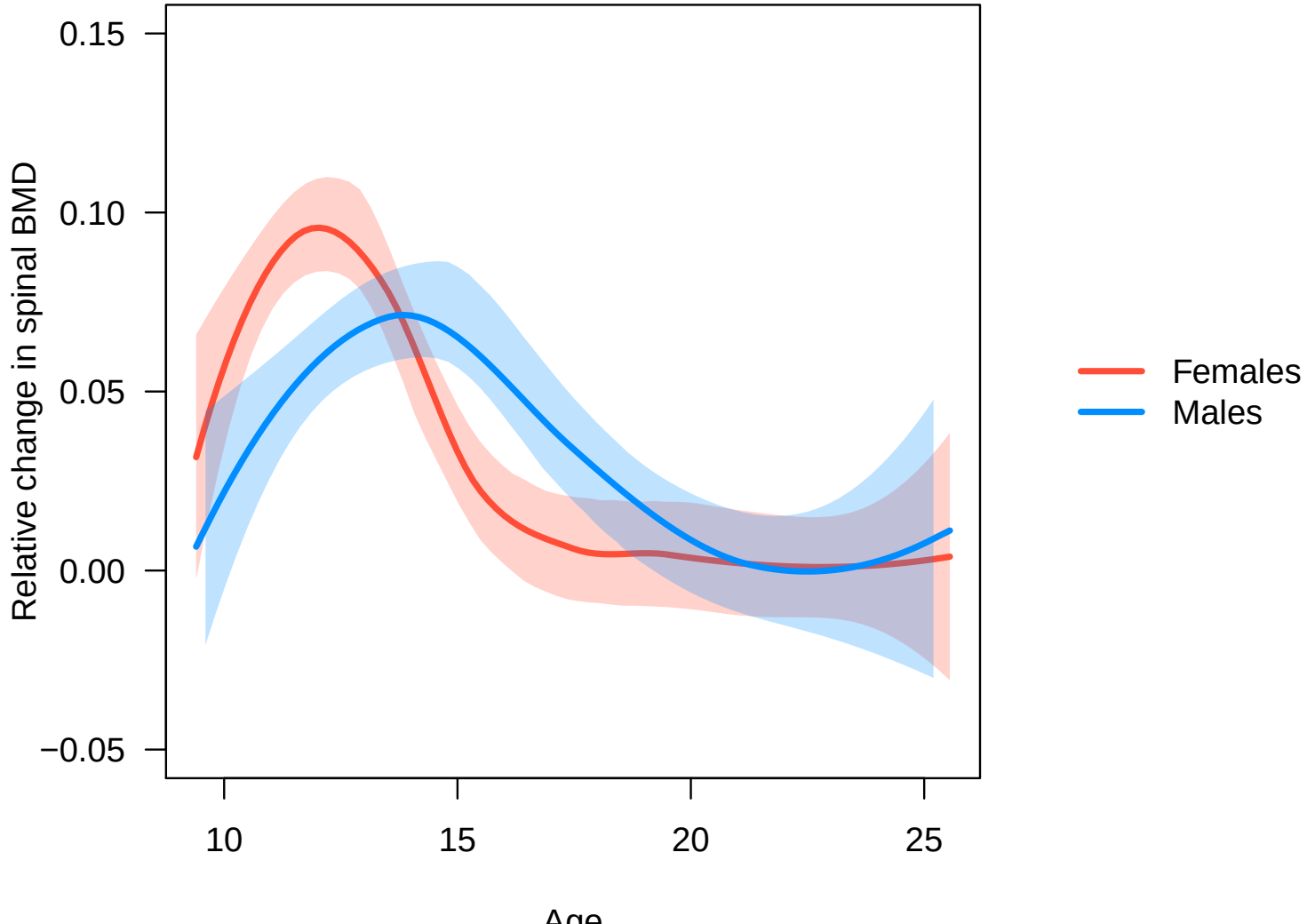
$$W(x) = \frac{\hat{f}(x) - \bar{f}(x)}{\sigma \, \|l(x)\|}$$

  as a Gaussian process
- We won't go into the details (they are in Section 5.7 of our text), but we will mention that the `scb` function in `locfit` computes simultaneous confidence bands (again, details are in the accompanying code)

Pointwise CIs for the bone mineral density data



Simultaneous CIs for the bone mineral density data



Hypothesis testing

- Lastly, we consider the issue of hypothesis testing
- Consider the following nested sequence of models:

$$\text{M0: } \mathbb{E}(Y|x) = \alpha$$
$$\text{M1: } \mathbb{E}(Y|x) = \alpha + \beta x$$
$$\text{M2: } \mathbb{E}(Y|x) = f(x)$$

- We may be interested in testing null hypotheses such as whether there is any relationship between x and y, or whether a linear parameterization of the relationship provides an adequate fit

F tests

- The idea of the F test from conventional linear models applies here as well:

$$F = \frac{(\mathrm{RSS}_0 - \mathrm{RSS}_1)/(\mathrm{df}_1 - \mathrm{df}_0)}{\mathrm{RSS}_1/(n - \mathrm{df}_1)},$$

  where $\mathrm{df} = 2\nu - \tilde{\nu}$

- Recall that there is a bias term present that we are ignoring; thus F follows an $F_{\mathrm{df}_1 - \mathrm{df}_0, n - \mathrm{df}_1}$ distribution only asymptotically

The gam function

- One can obviously calculate this quantity by hand using the `locfit` package, but it is usually more convenient to use the `gam` package
- The basic syntax of model fitting with `gam` is very similar to `loess` and `locfit`:

  ```
  fit <- gam(spnbmd~lo(age, span=.5, deg=1))
  ```

  where `lo` controls the local polynomial which is fit to the data

The anova function

One can then compare a sequence of nested models using `anova.gam`:

```
fit0 <- gam(spnbmd~1, data=m)
fit1 <- gam(spnbmd~age, data=m)
fit2 <- gam(spnbmd~lo(age), data=m)
anova(fit0, fit1, fit2, test="F")
```

	Rdf	RSS	Δdf	ΔRSS	F	$\mathbb{P}(>F)$
M0	225	0.53				
M1	224	0.45	1	0.08	47.91	< 0.0001
M2	221	0.38	3	0.07	16.25	< 0.0001

Moving beyond least squares

- Thus far, we have fit local least squares models
- More generally, we may allow the outcome Y_i to follow a distribution $f(y|\theta_i)$, *e.g.*,
  - Exponential: $f(y|\theta) = \theta^{-1}\exp(y/\theta), \quad y \geq 0$
  - Binomial: $f(1|\theta) = \theta, \quad f(0|\theta) = 1 - \theta$
- For regression problems, θ_i depends on some covariate x_i
- A parametric model would involve the specification $\theta_i = \alpha + \beta x_i$; today we will let $\theta_i = \theta(x_i)$ represent an unknown smooth function we wish to estimate

Local likelihood

- One way to achieve that flexibility is by fitting separate, local models at each target point x_0 and smoothing those models together using kernel weighting
- Specifically, at x_0, we estimate $\hat{\alpha}$ and $\hat{\beta}$ by maximizing

$$\sum_i K_h(x_0, x_i) l(\alpha + \beta x_i | y_i)$$

  where $l(\theta|y) = \log\{f(y|\theta)\}$
- In principle, any distribution and likelihood could be extended to this approach, but in practice it is usually applied to generalized linear models

Fitting local GLMs

- Letting the ith row of the design matrix be $(1, x_i - x_0)$ as in local linear regression, the local likelihood estimate $\widehat{\boldsymbol{\beta}}$ at x_0 can be found by solving

$$\mathbf{X}'\mathbf{W}\mathbf{u} = \mathbf{0},$$

  where $\mathbf{W}$ is the diagonal matrix of kernel weights and $\mathbf{u} = \frac{\partial}{\partial\theta} l(y_i, \hat{\theta}_i)$ is the score vector
- Unlike local linear regression, this equation typically does not have a closed form solution and must be solved by iterative methods

Linearization of the score

- As with regular GLMs, we may proceed by constructing a linear approximation to the score via Taylor series expansion around the current estimate, $\tilde{\boldsymbol{\theta}}$:

$$\mathbf{u} \approx \mathbf{V}(\mathbf{z} - \boldsymbol{\theta}),$$

  where $\mathbf{V}$ is a diagonal matrix with entries $-\frac{\partial^2}{\partial \theta^2} l(\hat{\theta}_i | y_i)$ (*i.e.*, the observed information) and $\mathbf{z} = \tilde{\boldsymbol{\theta}} + \mathbf{V}^{-1}(\mathbf{y} - \tilde{\mu})$ is the "pseudoresponse"
- The solution to our local maximum likelihood solution is therefore

$$\tilde{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{W}\mathbf{V}\mathbf{X})^{-1}\mathbf{X}\mathbf{W}\mathbf{V}\mathbf{z}$$

- It is important to keep in mind, however, that both $\mathbf{z}$ and $\mathbf{V}$ depend on $\tilde{\boldsymbol{\theta}}$, and thus we need to update them via $\tilde{\boldsymbol{\theta}} \leftarrow \mathbf{X}\tilde{\boldsymbol{\beta}}$ and iterate until convergence

Deviance and degrees of freedom

- The analogous concept to the residual sum of squares for generalized linear models is the *deviance*:

$$D(\mathbf{y}|\hat{\boldsymbol{\theta}}) = 2\left\{l(\boldsymbol{\theta}_{\max}|\mathbf{y}) - l(\hat{\boldsymbol{\theta}}|\mathbf{y})\right\},$$

  where $\boldsymbol{\theta}_{\max}$ is the vector of parameters that maximize $l(\boldsymbol{\theta}_{\max}|\mathbf{y})$ over all $\boldsymbol{\theta}$ (the "saturated" model)
- Continuing with the analogy to local linear regression, we may define our two effective degree of freedom terms:

$$\begin{aligned}\nu &= \mathrm{tr}(\mathbf{R}) \\ \tilde{\nu} &= 2\mathrm{tr}(\mathbf{R}) - \mathrm{tr}(\mathbf{R}'\mathbf{V}\mathbf{R}\mathbf{V}^{-1}),\end{aligned}$$

  where $\mathbf{R} = \mathbf{X}(\mathbf{X}'\mathbf{W}\mathbf{V}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{V}$

Caveats

- Unlike the residual sum of squares, the deviance is not χ^2 distributed, not even asymptotically
- Nevertheless, inference based on deviance and approximate degrees of freedom is useful in practice, aids with interpretation, and usually provides adequate empirical accuracy in terms of preserving coverage and type I error rates

Selection of h

- As always, there is the issue of how to choose the bandwidth h
- One approach is to carry out leave-one-out cross-validation with deviance replacing squared error loss:

$$\mathrm{CV} = \sum_i D\left(y_i | \hat{\theta}_{-i}(x_i)\right)$$

- However, unlike local linear regression, non-gaussian GLMs are not linear smoothers and there is no convenient way to calculate $\hat{\theta}_{-i}(x_i)$ without refitting the model
- For this reason, it is customary to use a criterion such as AIC instead:

$$\mathrm{AIC} = \sum_i D(y_i | \hat{\theta}_i) + 2\nu$$

Confidence intervals

One can obtain confidence intervals for $\theta(x_0)$ via quadratic approximations, as is often done with GLMs themselves:

$$\hat{\theta}(x_0) = \mathbf{Rz}$$

Thus,

$$\begin{aligned}\mathbb{V}\{\hat{\theta}(x_0)\} &= \mathbf{R}\mathbb{V}(\mathbf{z})\mathbf{R}' \\ &= \mathbf{R}\mathbf{V}^{-1}\mathbf{R}'\end{aligned}$$

Generalized likelihood ratio tests

- Finally, we can carry out hypothesis testing between two nested models via approximate generalized likelihood ratio tests:

$$\Lambda = 2\left\{l(\hat{\boldsymbol{\theta}}_1|\mathbf{y}) - l(\hat{\boldsymbol{\theta}}_0|\mathbf{y})\right\}$$

  or equivalently,

$$\Lambda = 2\left\{D(\mathbf{y}|\hat{\boldsymbol{\theta}}_0) - D(\mathbf{y}|\hat{\boldsymbol{\theta}}_1)\right\}$$

- Under the null hypothesis that model 0 is correct, Λ follows a distribution very similar to a χ^2 distribution with $\tilde{\nu}_1 - \tilde{\nu}_0$ degrees of freedom

Syntax

- The syntax for fitting generalized linear models in `R` is straightforward; both `locfit` and `gam` provide a `family` argument that works exactly the same as it does in `glm`
- Thus, for `locfit`:

  ```
  locfit(chd~lp(sbp), data=heart, family="binomial")
  ```

  and for `gam`:

  ```
  gam(chd~lo(sbp), data=heart, family="binomial")
  ```

Local logistic regression

- By default, both `gam` and `locfit` incorporate a *link function*; rather than model $\mathbb{E}(Y)$ directly, they model

$$g\left\{\mathbb{E}(Y|x)\right\} = \theta(x),$$

  where g is a known function
- For logistic regression, g is usually chosen to be the logit function:

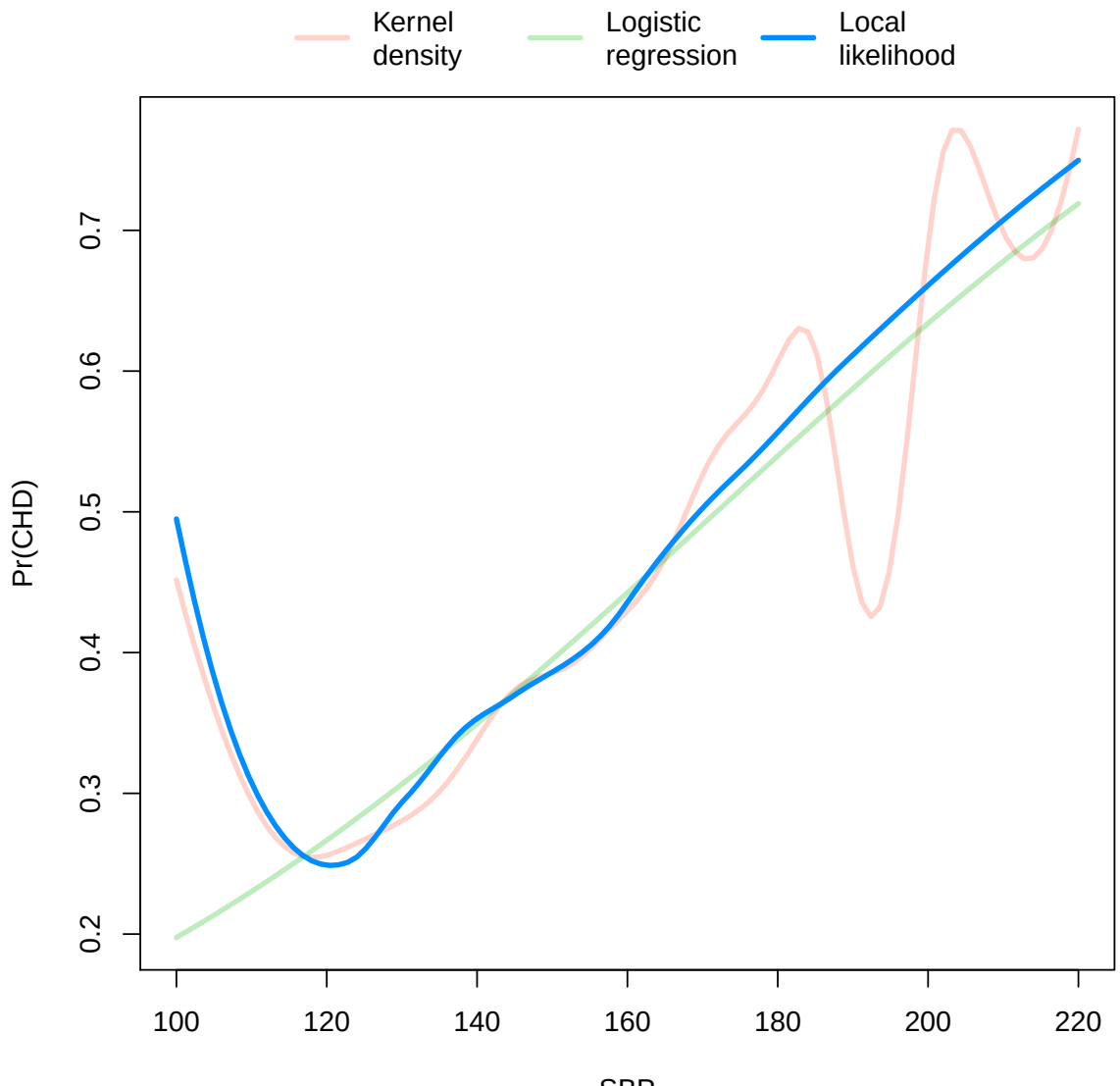
$$g(\pi) = \log\left\{\frac{\pi}{1-\pi}\right\},$$

  where $\pi = \mathbb{P}(Y = 1)$, thus implying

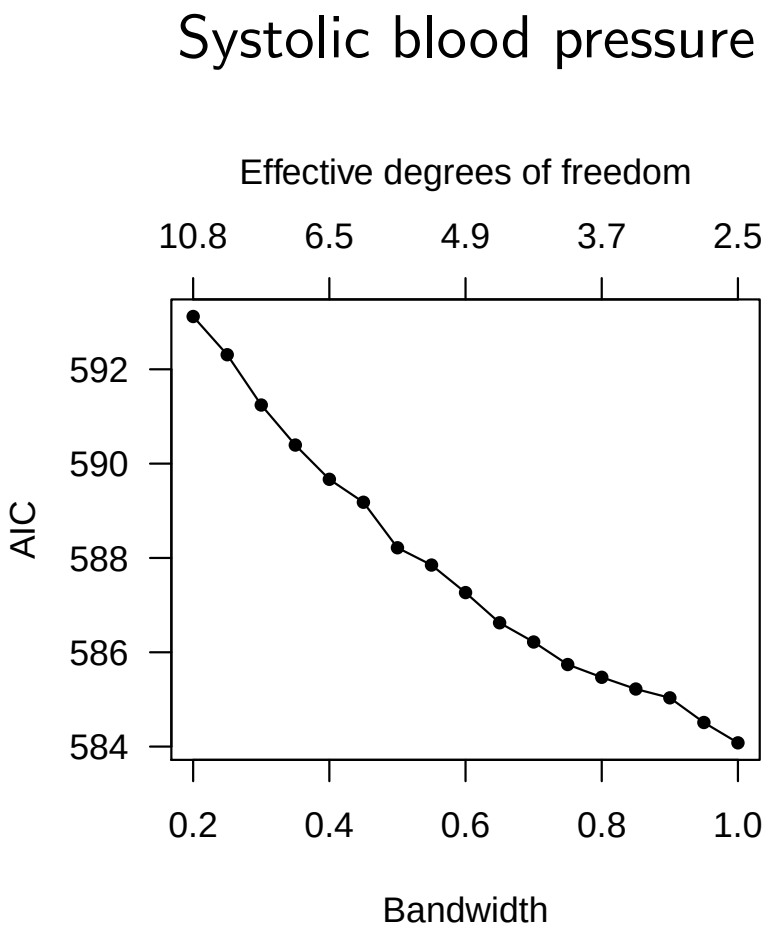
$$\pi = \frac{e^{\theta}}{1 + e^{\theta}}$$

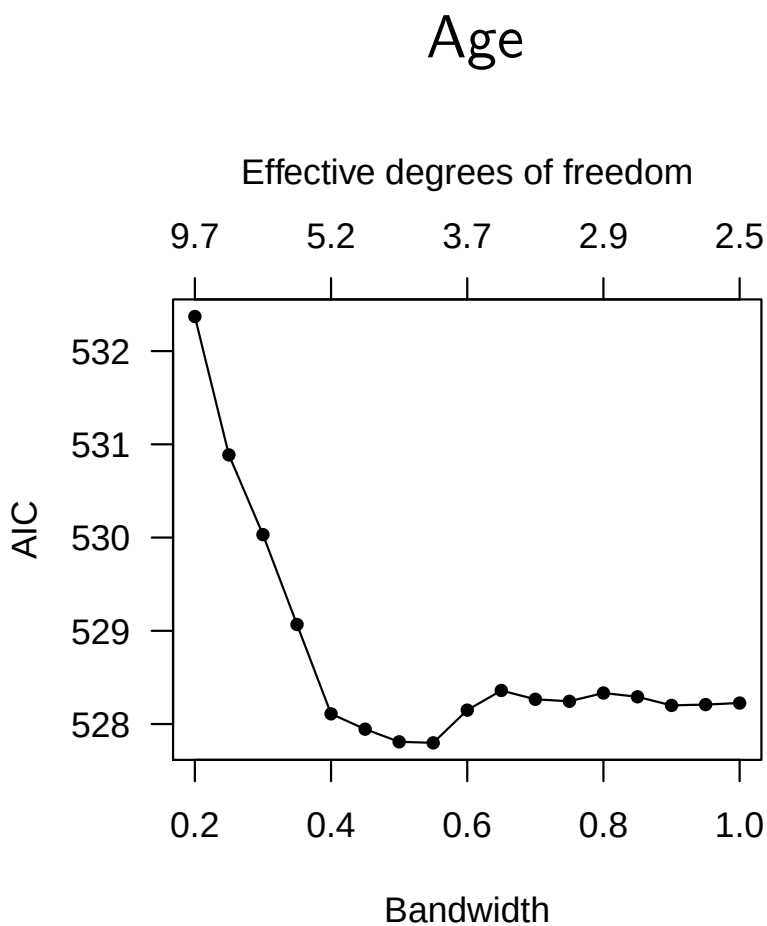
- This is the *canonical link* for a binomial likelihood; in general, canonical links have many attractive statistical properties, such as ensuring that $\mathbb{E}(Y)$ stays within the support of Y

Comparison of local likelihood with other methods

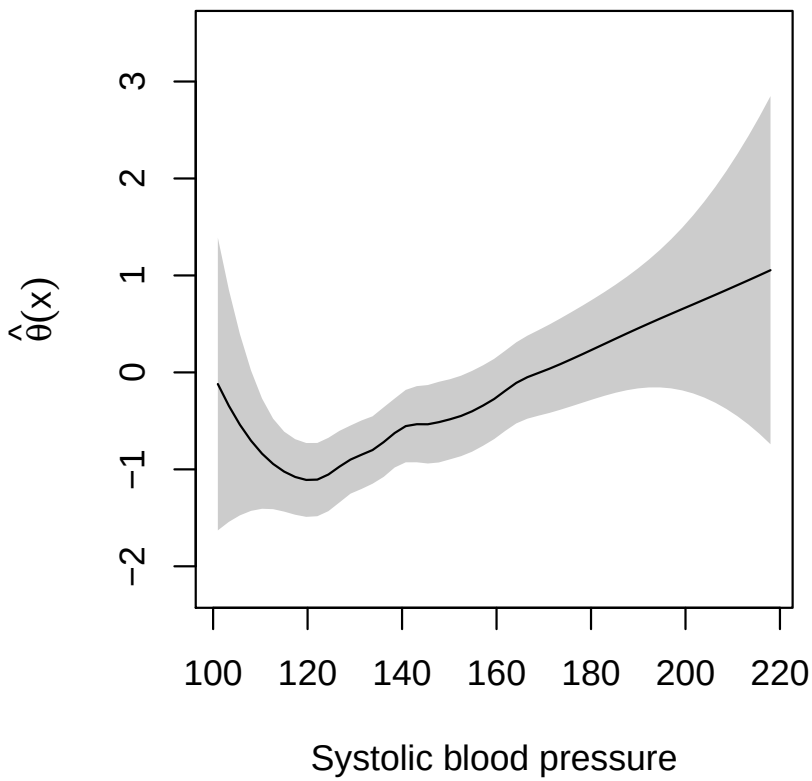


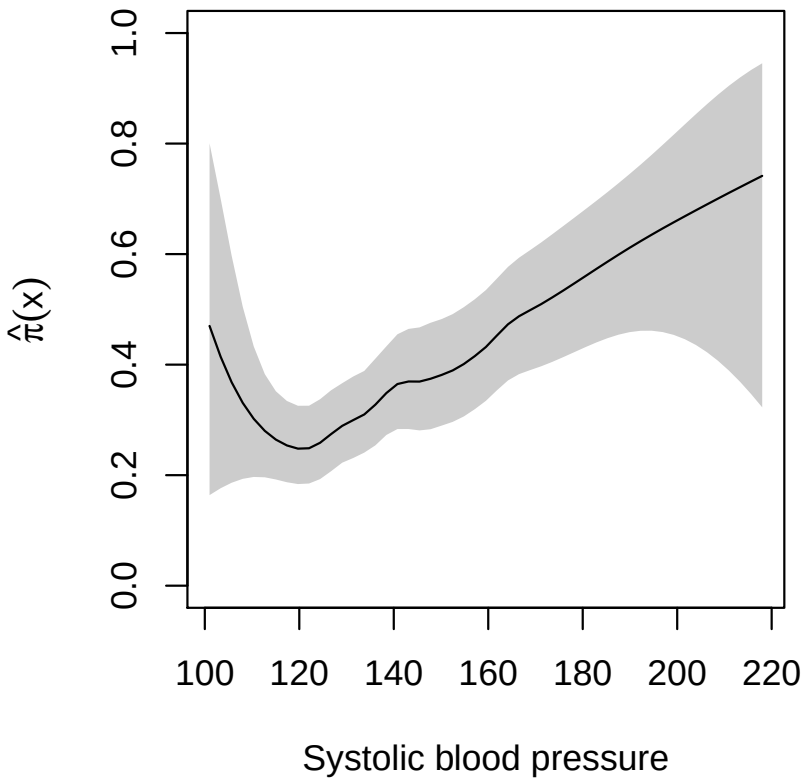
Using AIC to choose bandwidth



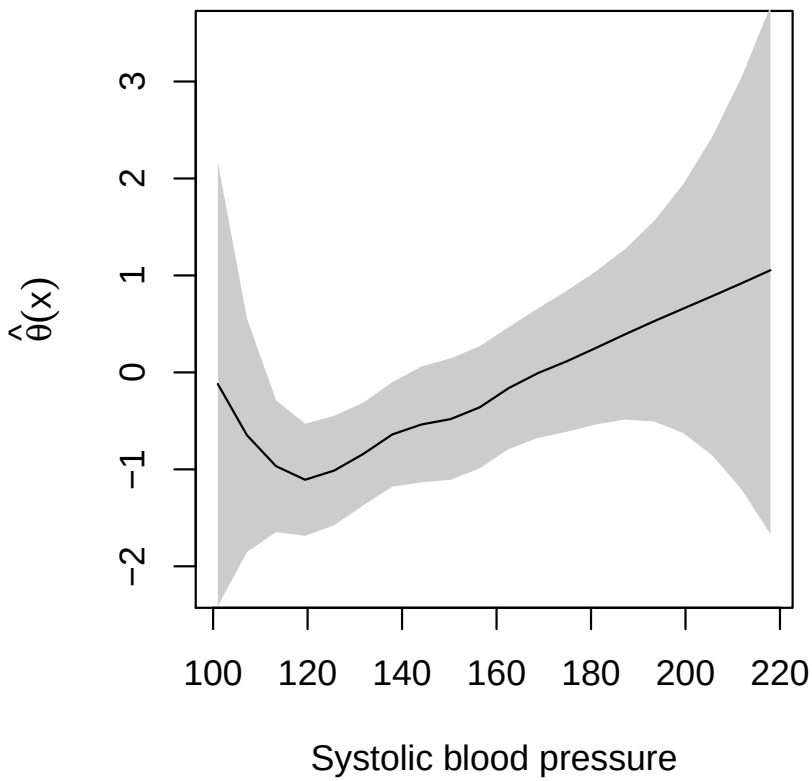


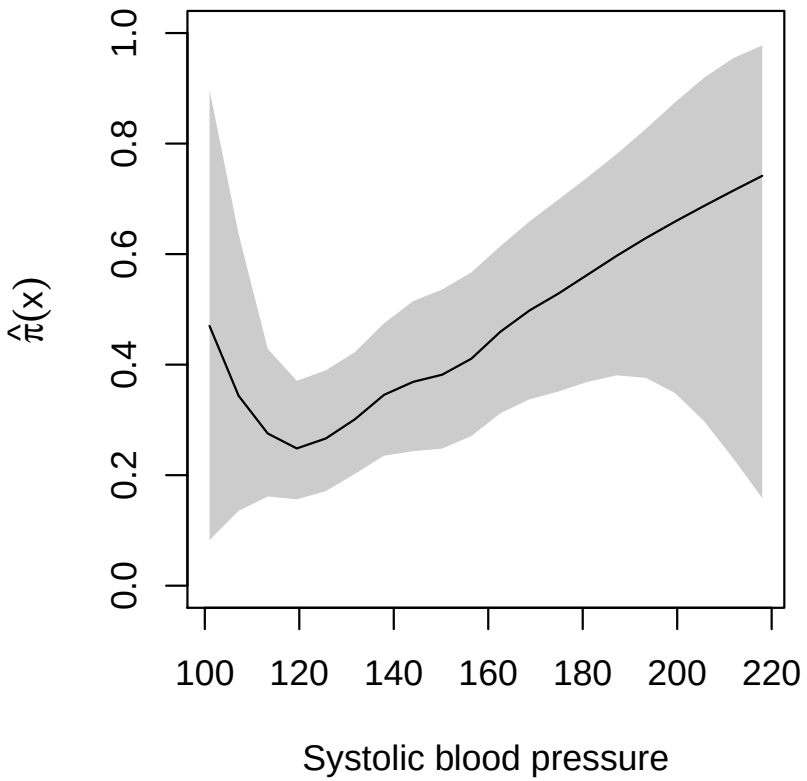
SBP: Pointwise bands



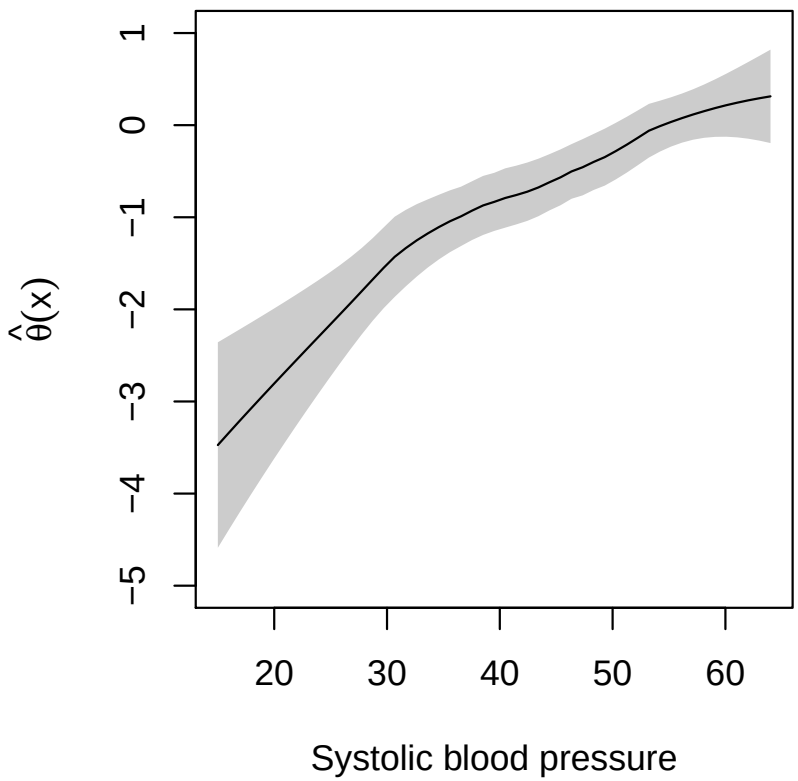


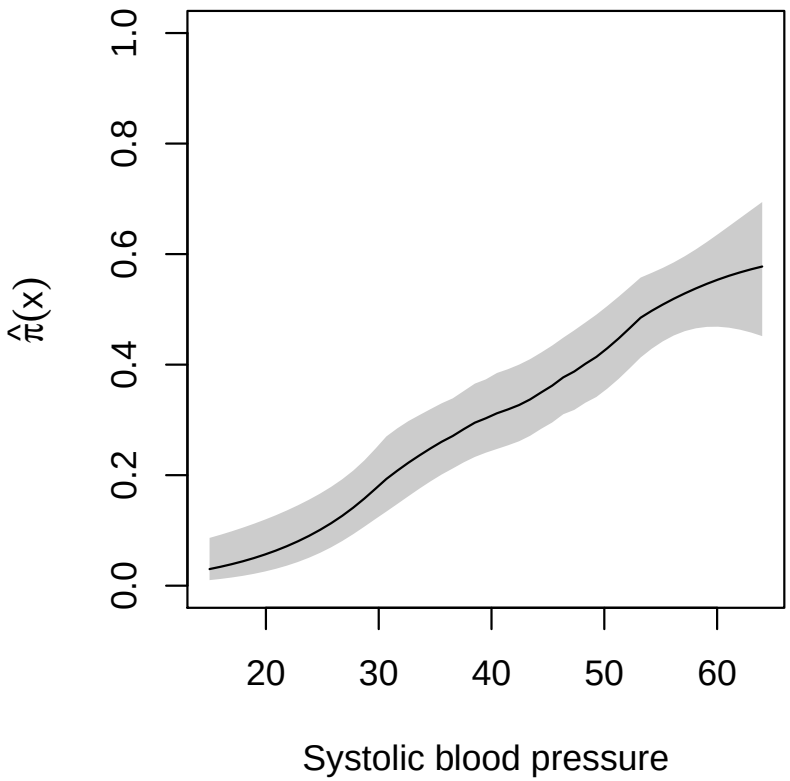
SBP: Simultaneous bands



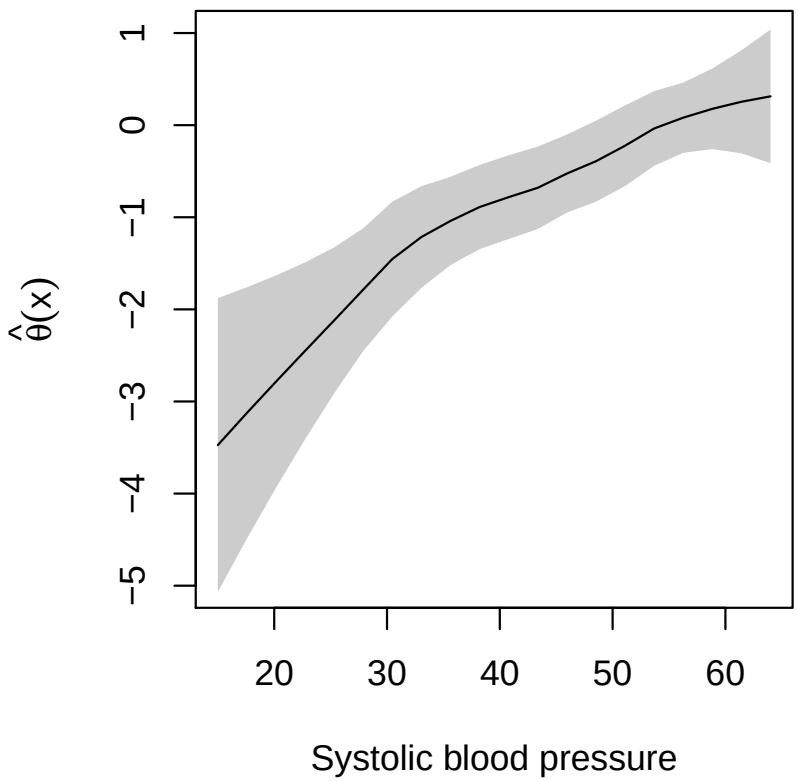


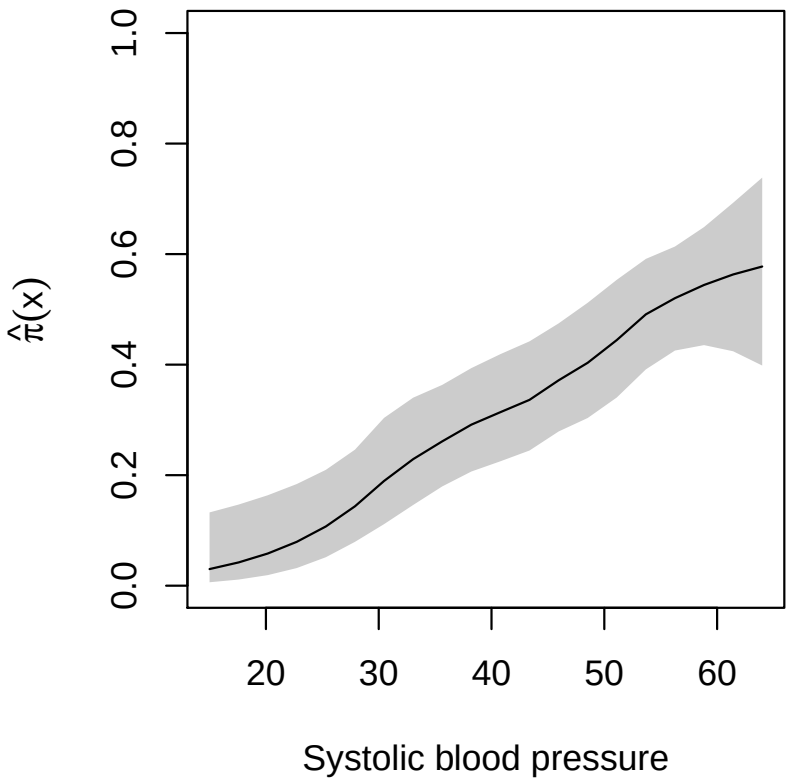
Age: Pointwise bands





Age: Simultaneous bands





ANOVA table: SBP

	Resid. df	Deviance	$\tilde{\nu}$	ΔDev	p
Null	461	596.1			
Linear	460	579.3	1	16.79	< 0.0001
Local	457.4	577.7	2.6	1.60	0.58

ANOVA table: Age

	Resid. df	Deviance	$\tilde{\nu}$	ΔDev	p
Null	461	596.1			
Linear	460	525.6	1	70.55	< 0.0001
Local	457.5	519.9	2.5	5.65	0.09

Introduction

- We are discussing ways to estimate the regression function f, where

$$\mathbb{E}(y|x) = f(x)$$

- One approach is of course to assume that f has a certain shape, such as linear or quadratic, that can be estimated parametrically
- We have also discussed locally weighted linear/polynomial models as a way of allowing f to be more flexible
- An alternative approach is to introduce *local basis functions*

Basis functions

- A common approach for extending the linear model is to augment the linear component of x with additional, known functions of x:

$$f(x) = \sum_{m=1}^{M} \beta_m h_m(x),$$

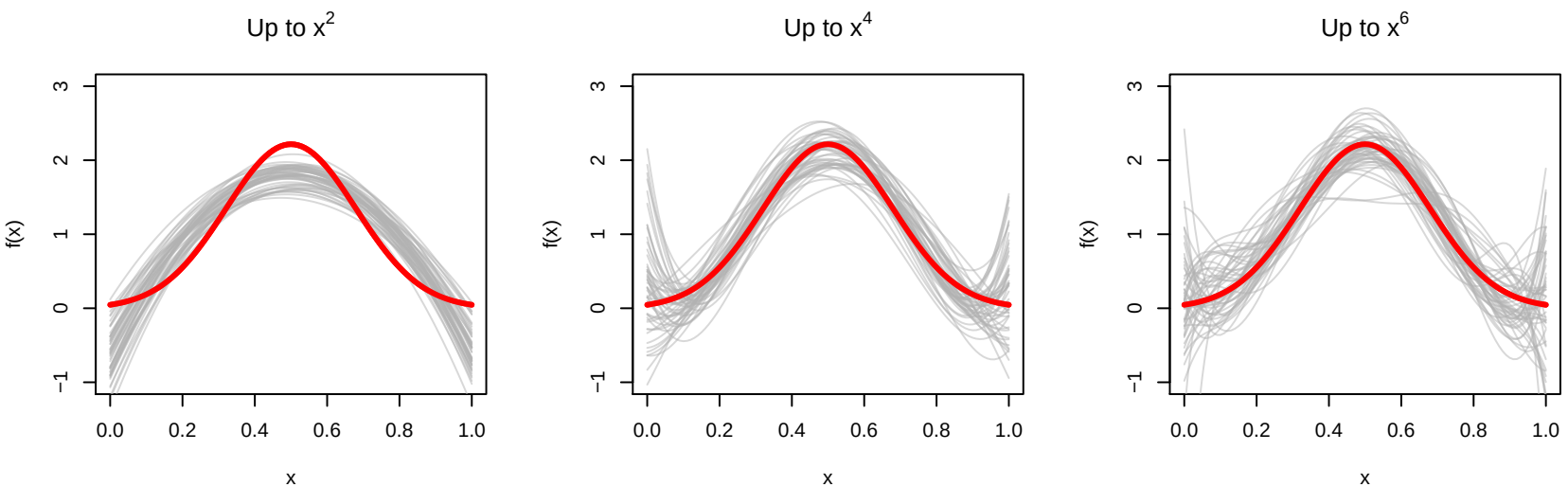
  where the $\{h_m\}$ are known functions called *basis functions*
- Because the basis functions $\{h_m\}$ are prespecified and the model is linear in these new variables, ordinary least squares approaches for model fitting and inference can be employed
- This idea is not new to you, as you have encountered transformations and the inclusion of polynomial terms in models in earlier courses

Problems with polynomial regression

- However, polynomial terms introduce undesirable side effects: each observation affects the entire curve, even for x values far from the observation
- Not only does this introduce bias, but it also results in extremely high variance near the edges of the range of x
- As Hastie *et al.* (2009) put it, "tweaking the coefficients to achieve a functional form in one region can cause the function to flap about madly in remote regions"

Problems with polynomial regression (cont'd)

To illustrate this, consider the following simulated example (gray lines are models fit to 100 observations arising from the true f, colored red):



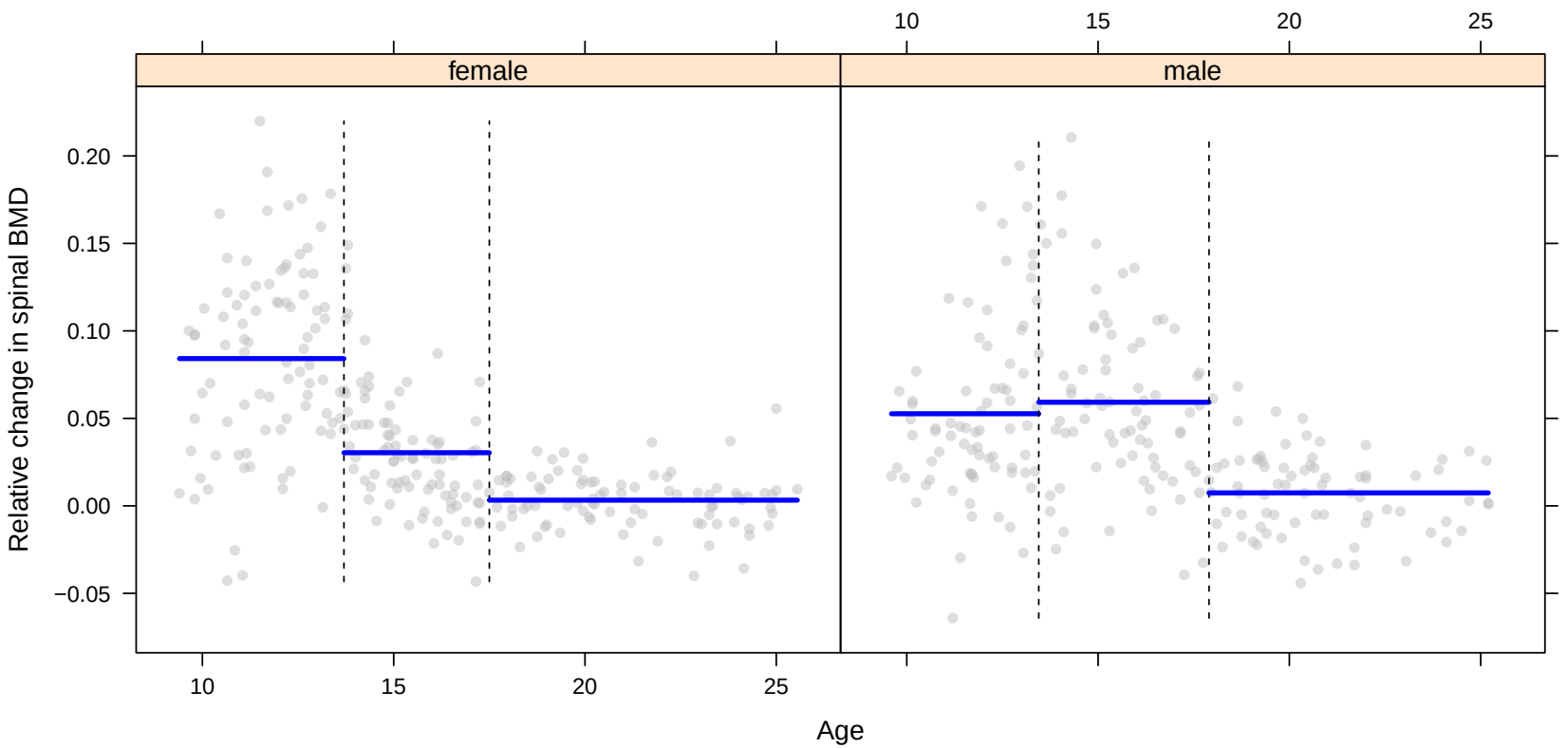
Global versus local bases

- We can do better than this
- Let us consider instead *local* basis functions, thereby ensuring that a given observation affects only the nearby fit, not the fit of the entire line
- In this lecture, we will discuss *splines*: piecewise polynomials joined together to make a single smooth curve

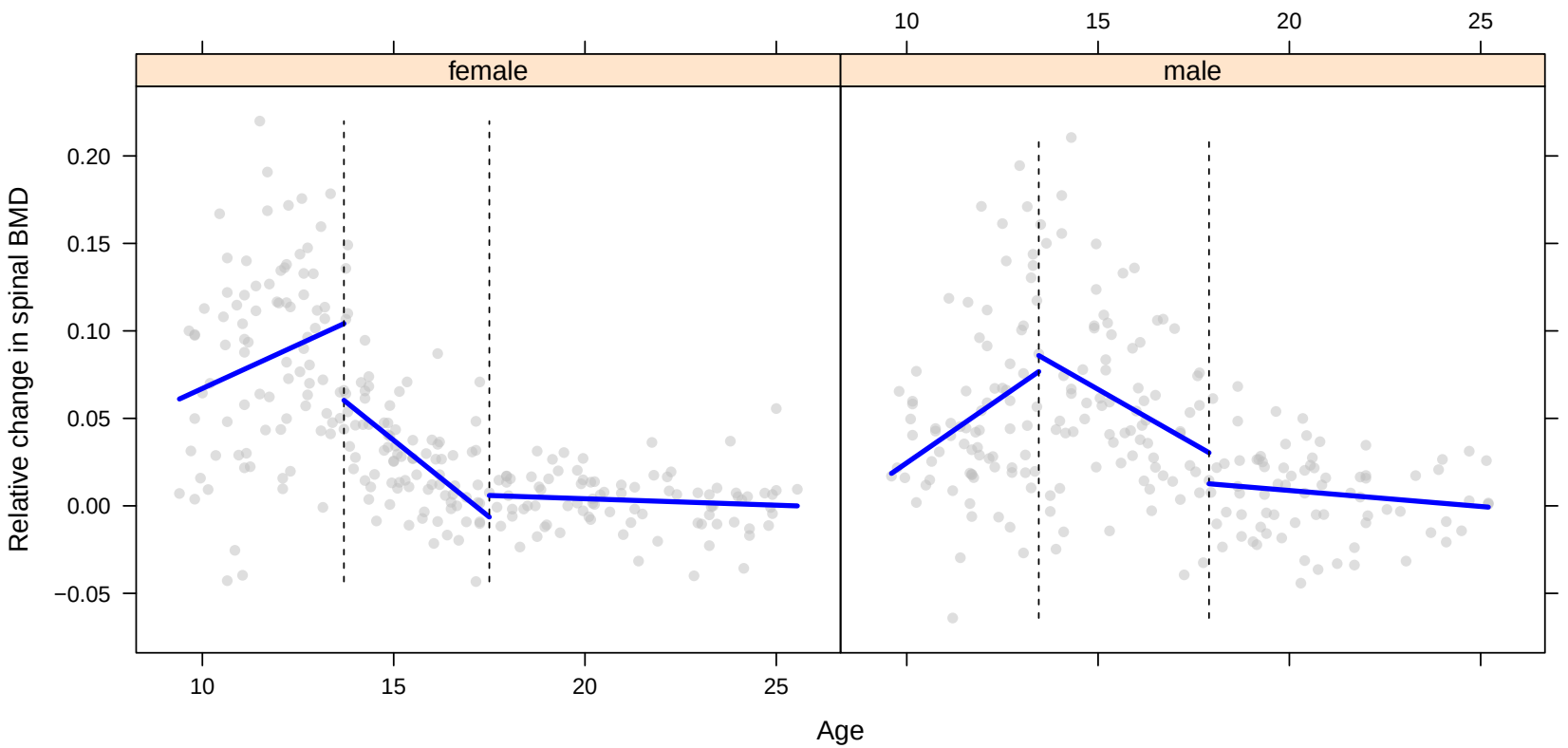
The piecewise constant model

- To understand splines, we will gradually build up a piecewise model, starting at the simplest one: the piecewise constant model
- First, we partition the range of x into $K + 1$ intervals by choosing K points $\{\xi_k\}_{k=1}^{K}$ called *knots*
- For our example involving bone mineral density, we will choose the tertiles of the observed ages

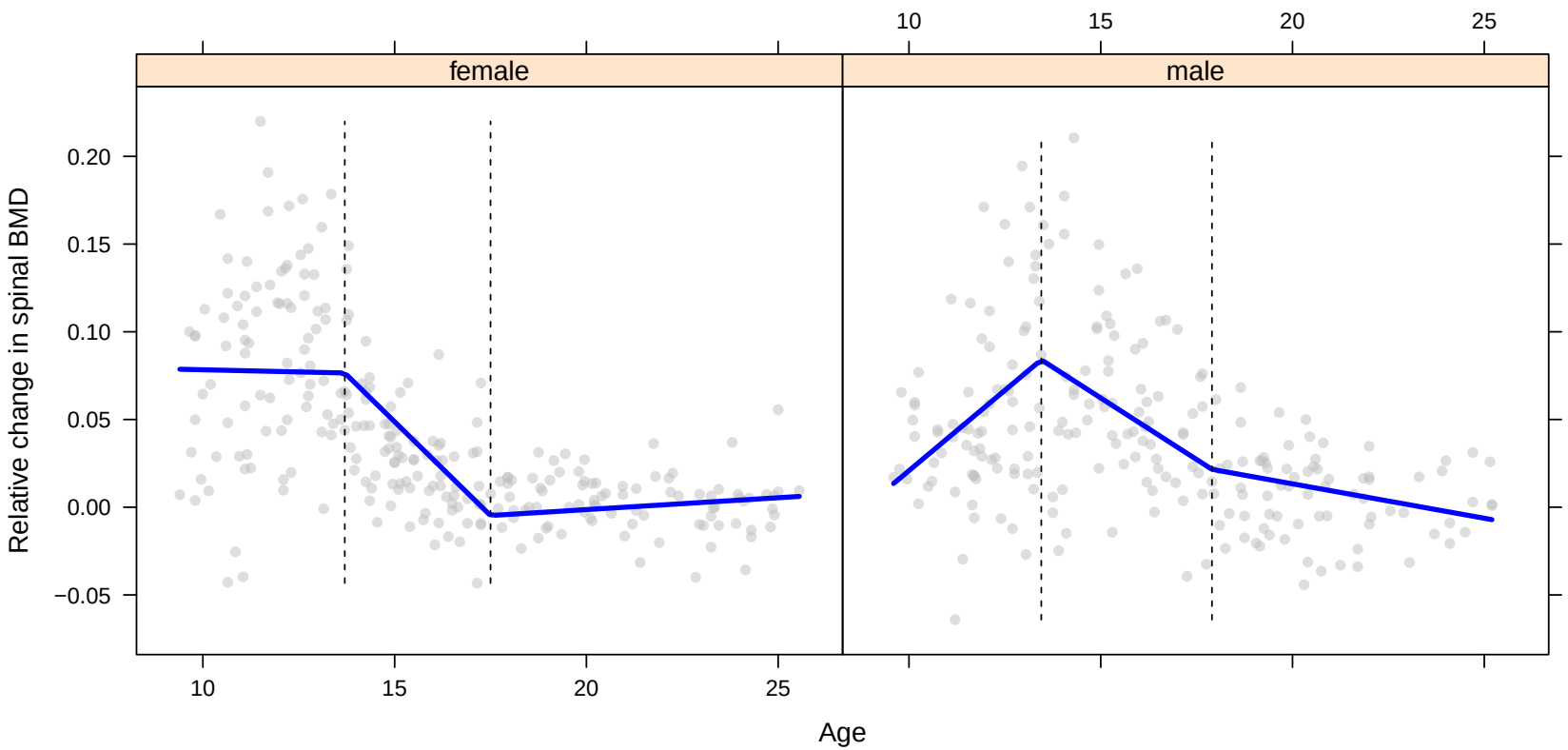
The piecewise constant model (cont'd)



The piecewise linear model



The continuous piecewise linear model



Basis functions for piecewise continuous models

These constraints can be incorporated directly into the basis functions:

$$h_1(x) = 1, \quad h_2(x) = x, \quad h_3(x) = (x - \xi_1)_+, \quad h_4(x) = (x - \xi_2)_+,$$

where $(\cdot)_+$ denotes the positive portion of its argument:

$$r_+ = \begin{cases} r & \text{if } r \geq 0 \\ 0 & \text{if } r < 0 \end{cases}$$

Basis functions for piecewise continuous models

- It can be easily checked that these basis functions lead to a composite function $f(x)$ that:
  - Is linear everywhere except the knots
  - Has a different intercept and slope in each region
  - Is everywhere continuous
- Also, note that the degrees of freedom add up: 3 regions $\times$ 2 degrees of freedom in each region - 2 constraints $=$ 4 basis functions

Splines

- The preceding is an example of a *spline*: a piecewise $m-1$ degree polynomial that is continuous up to its first $m-2$ derivatives
- By requiring continuous derivatives, we ensure that the resulting function is as smooth as possible
- We can obtain more flexible curves by increasing the degree of the spline and/or by adding knots
- However, there is a tradeoff:
  - Few knots/low degree: Resulting class of functions may be too restrictive (bias)
  - Many knots/high degree: We run the risk of overfitting (variance)

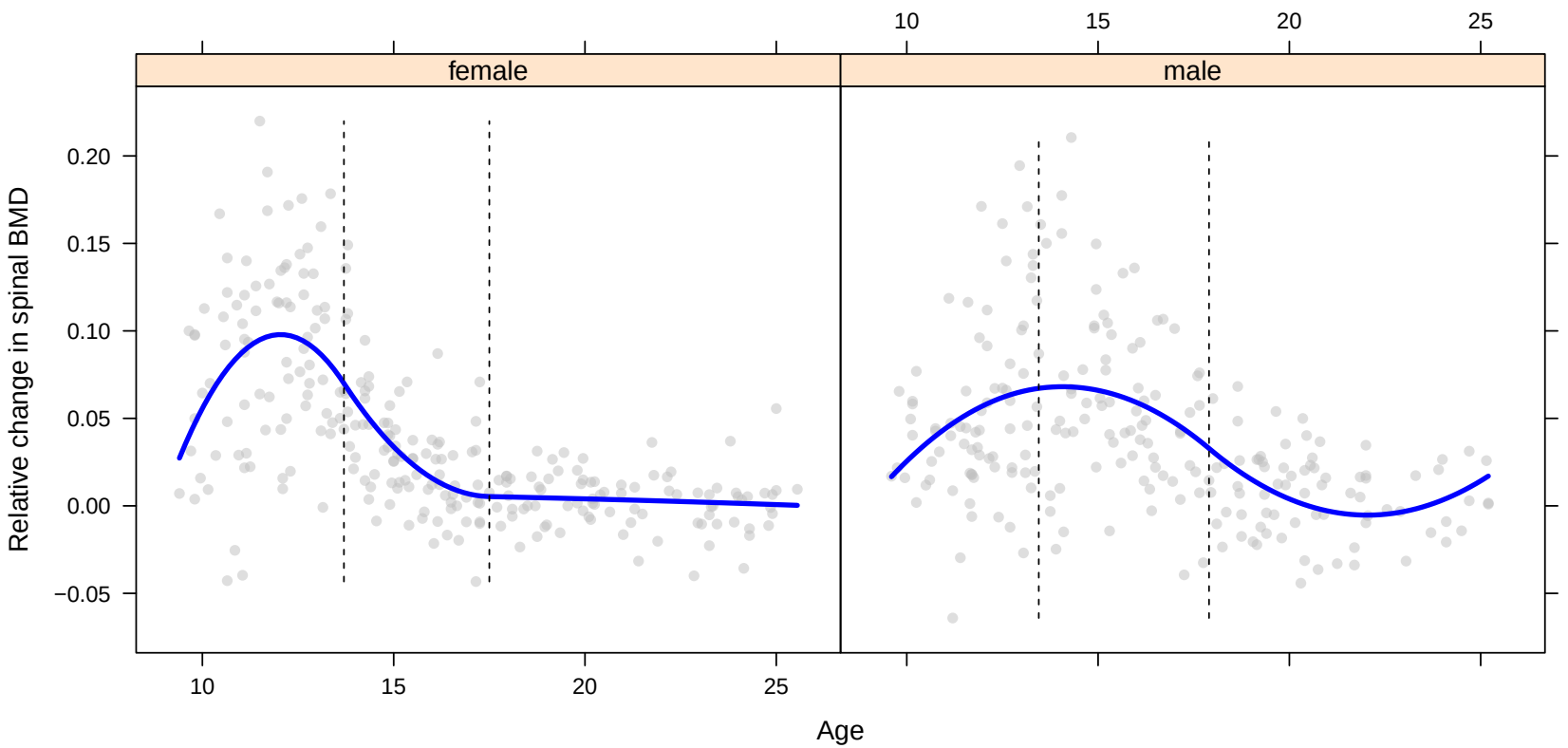
The truncated power basis

- The set of basis functions introduced earlier is an example of what is called the *truncated power basis*
- Its logic is easily extended to splines of order m:

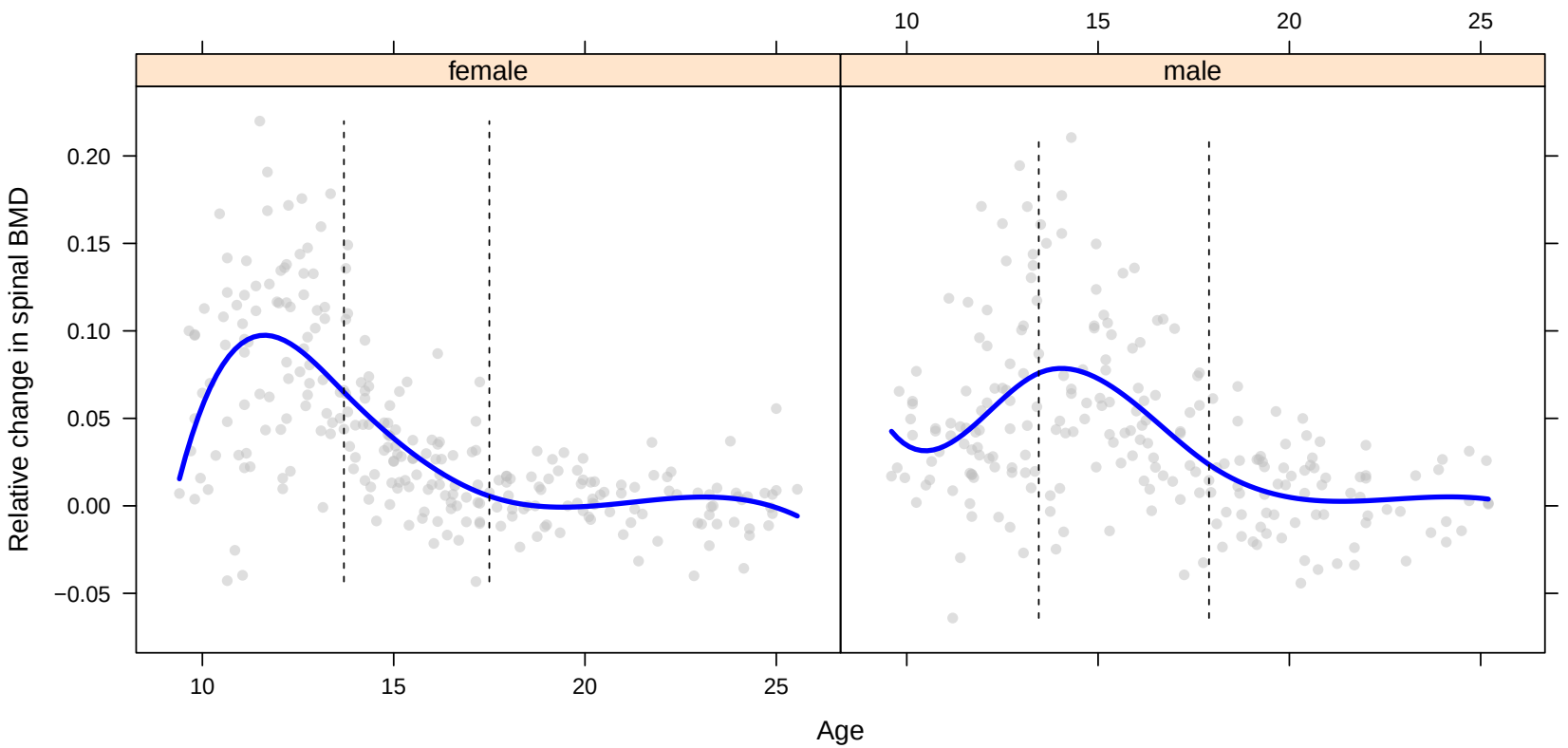
$$h_j(x) = x^{j-1} \qquad j = 1, \ldots, m$$
$$h_{m+k}(x) = (x - \xi_k)_+^{m-1} \qquad k = 1, \ldots, K$$

- Note that a spline has $m + K$ degrees of freedom
- **Homework:** Write the set of truncated spline basis functions for representing a cubic spline function with three knots.

Quadratic splines



Cubic splines



Additional notes

- These types of fixed-knot models are referred to as *regression splines*
- Recall that cubic splines contain $4 + K$ degrees of freedom: $K + 1$ regions $\times$ 4 parameters per region - K knots $\times$ 3 constraints per knot
- It is claimed that cubic splines are the lowest order spline for which the discontinuity at the knots cannot be noticed by the human eye
- There is rarely any need to go beyond cubic splines, which are by far the most common type of splines in practice

Implementing regression splines

- The truncated power basis has two principal virtues:
  - Conceptual simplicity
  - The linear model is nested inside it, leading to simple tests of the null hypothesis of linearity
- Unfortunately, it has several computational/numerical flaws – it's inefficient and can lead to overflow and nearly singular matrix problems
- The more complicated but numerically much more stable and efficient *B-spline* basis is often employed instead

B-splines in R

- Fortunately, one can use B-splines without knowing the details behind their complicated construction
- In the `splines` package (which by default is installed but not loaded), the `bs()` function will implement a B-spline basis for you

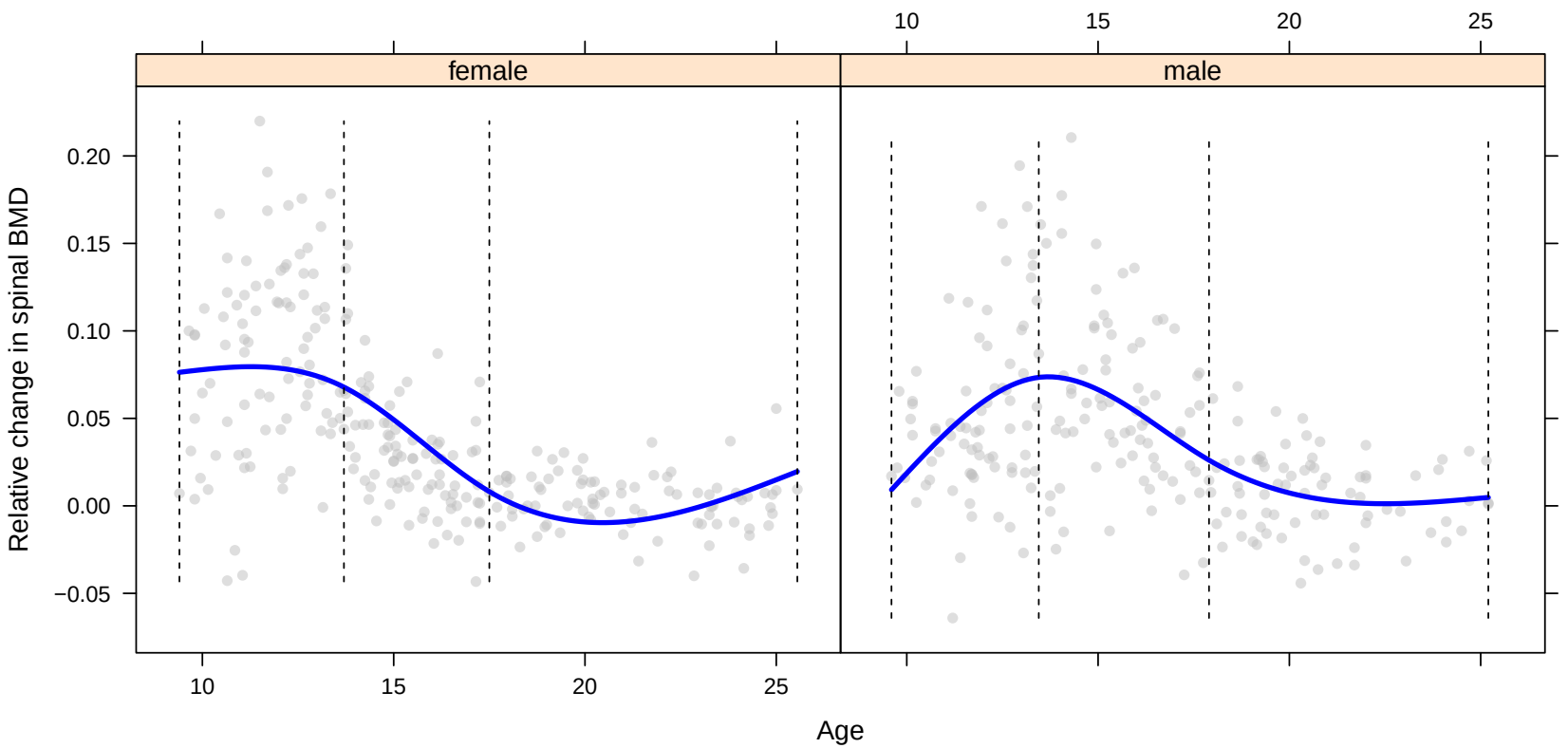
```
X <- bs(x,knots=quantile(x,p=c(1/3,2/3)))
X <- bs(x,df=5)
X <- bs(x,degree=2,df=10)
Xp <- predict(X,newdata=x)
```

- By default, `bs` uses `degree=3`, knots at evenly spaced quantiles, and does not return a column for the intercept

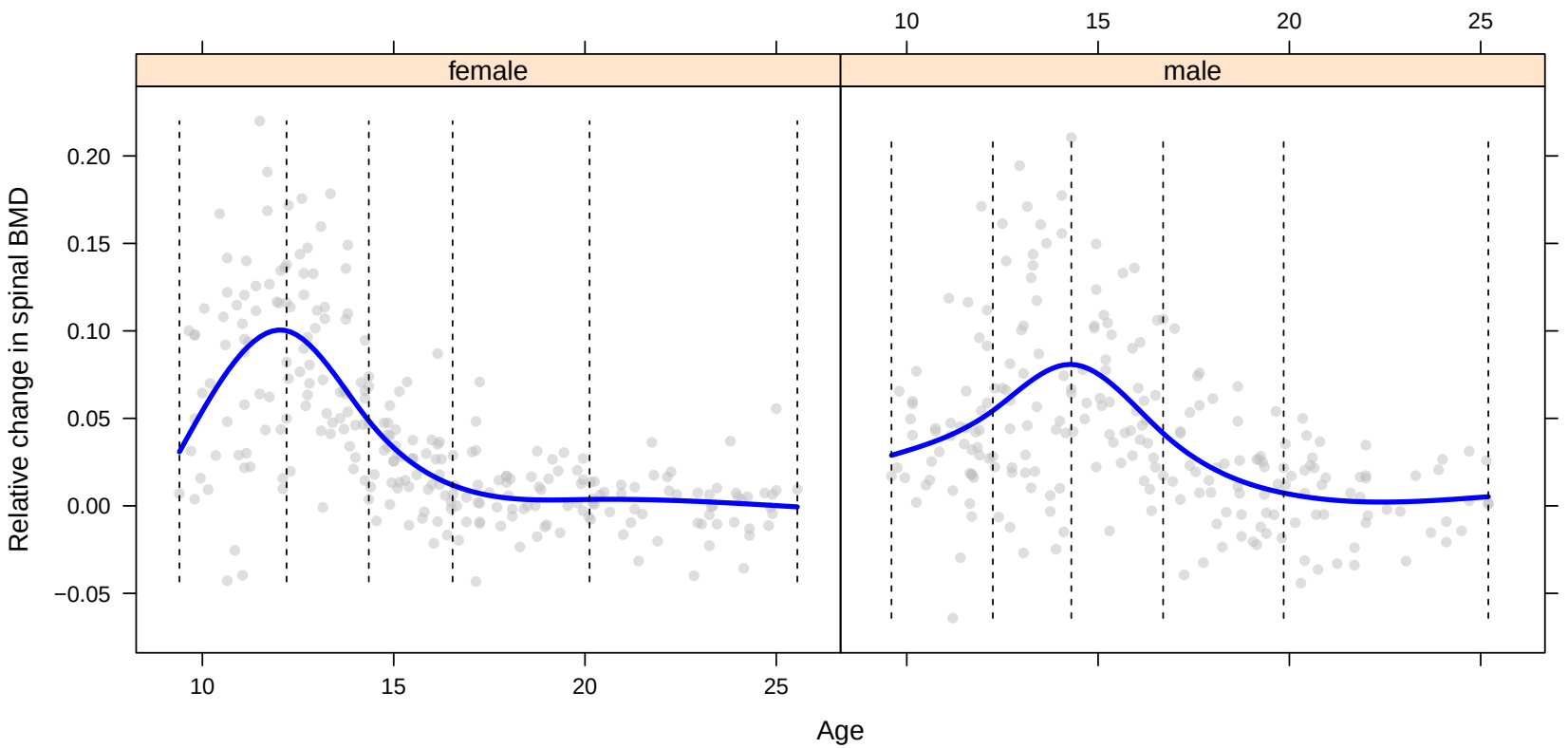
Natural cubic splines

- Polynomial fits tend to be erratic at the boundaries of the data; naturally, cubic splines share the same flaw
- *Natural cubic splines* ameliorate this problem by adding the additional (4) constraints that the function is linear beyond the boundaries of the data

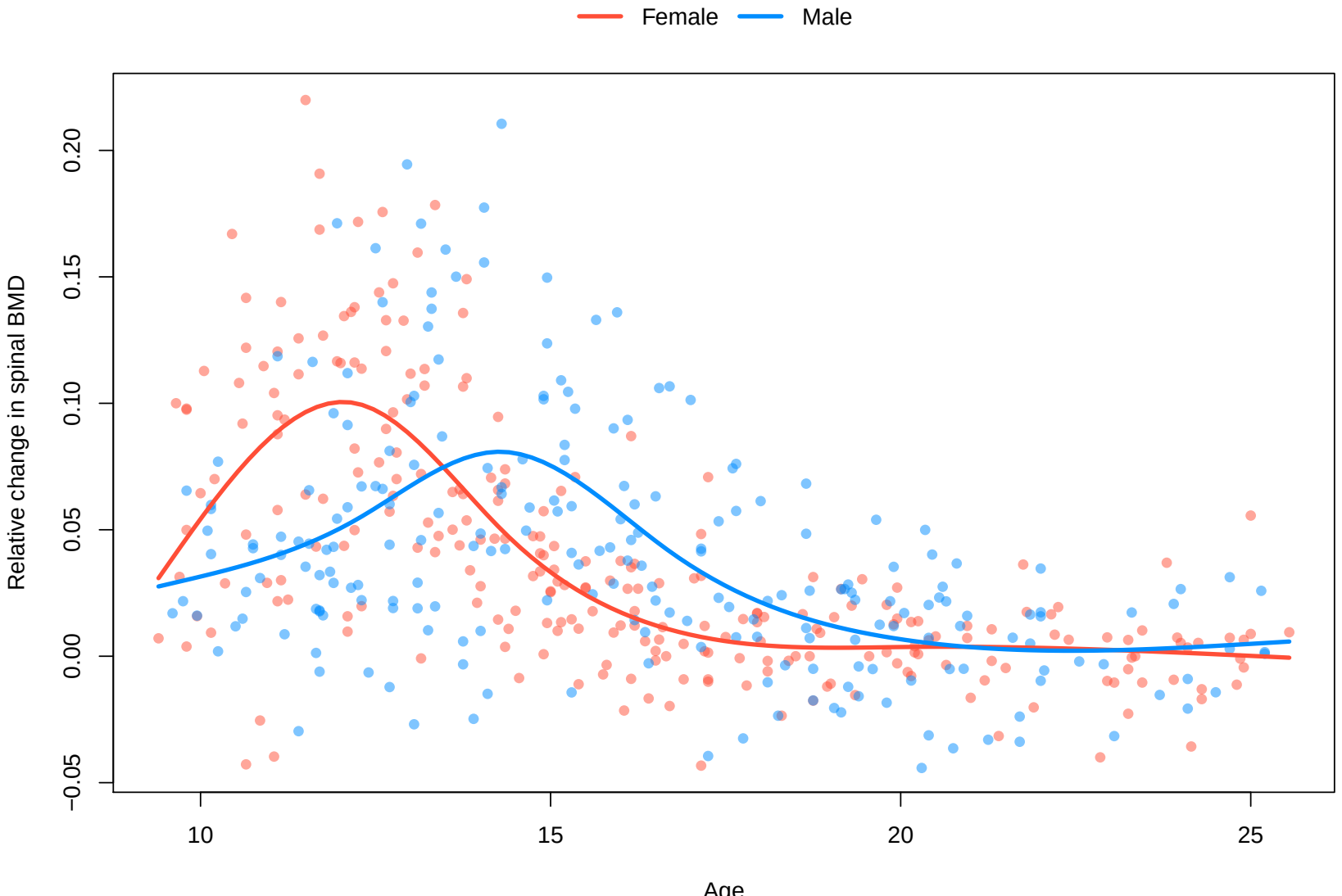
Natural cubic splines (cont'd)



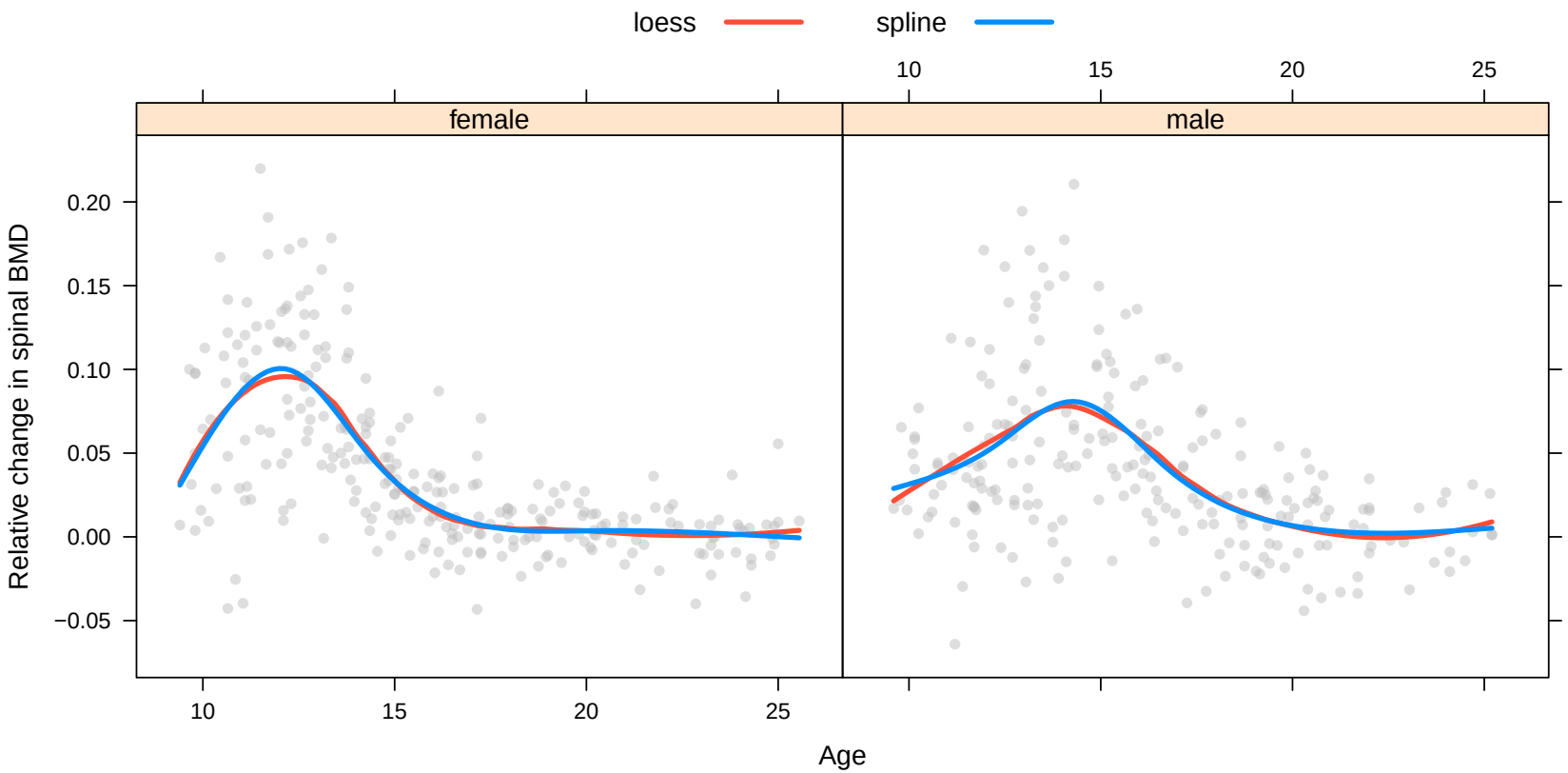
Natural cubic splines, 6 df



Natural cubic splines, 6 df (cont'd)



Splines vs. Loess (6 df each)



Natural splines in R

- R also provides a function to compute a basis for the natural cubic splines, ns, which works almost exactly like bs, except that there is no option to change the degree
- Note that a natural spline has $m + K - 4$ degrees of freedom; thus, a natural cubic spline with K knots has K degrees of freedom

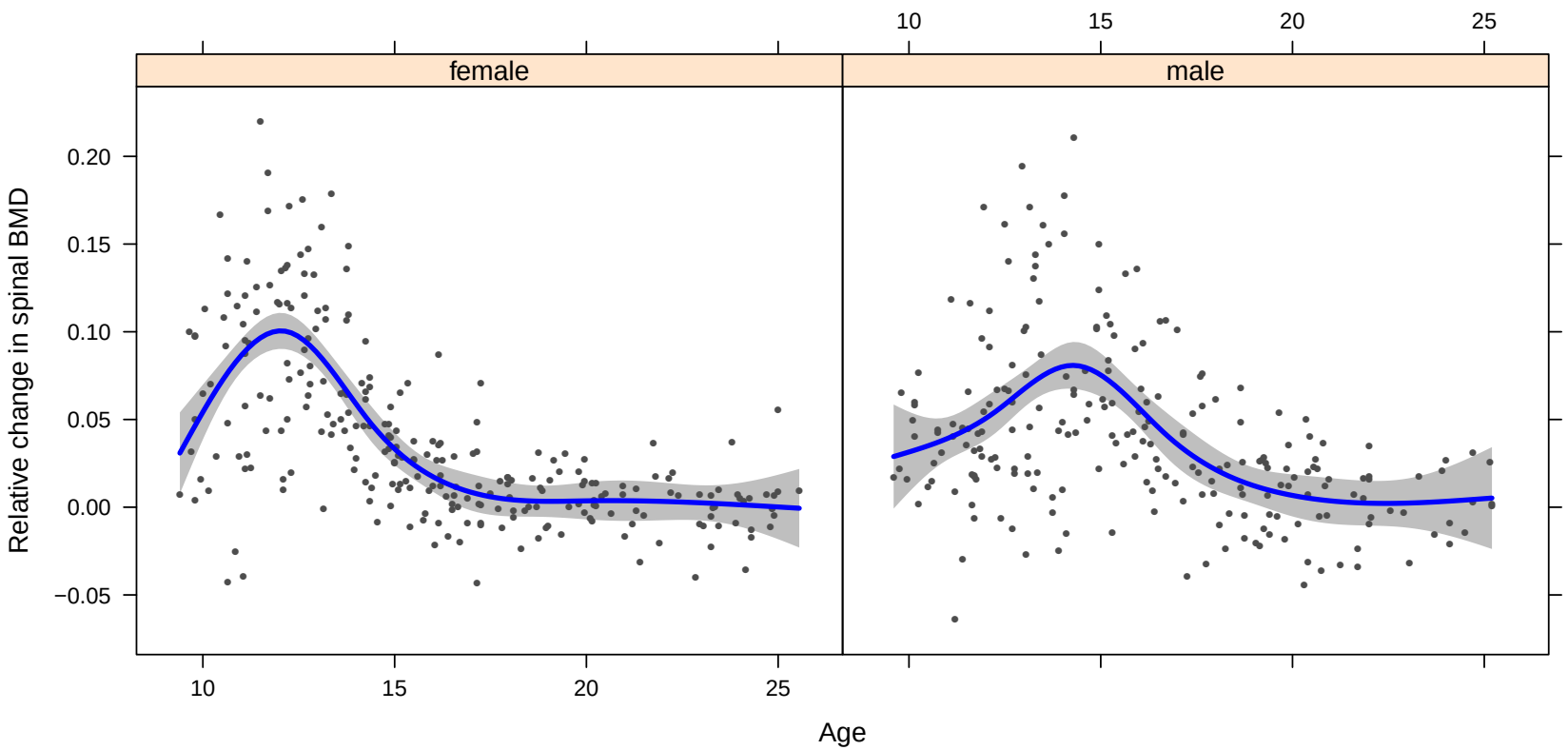
Mean and variance estimation

- Because the basis functions are fixed, all standard approaches to inference for regression are valid
- Furthermore, note that the resulting estimate is a linear smoother with $\mathbf{L} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$; thus

$$\mathbb{E}(\hat{\mathbf{f}}) = \mathbf{L}\mathbf{f} \quad \text{where } \mathbf{f} = \mathbb{E}(\mathbf{y}|\mathbf{x})$$
$$\mathbb{V}(\hat{\mathbf{f}}) = \sigma^2\mathbf{L}\mathbf{L}'$$
$$CV = \frac{1}{n}\sum_i \left(\frac{y_i - \hat{y}_i}{1 - l_{ii}}\right)^2$$

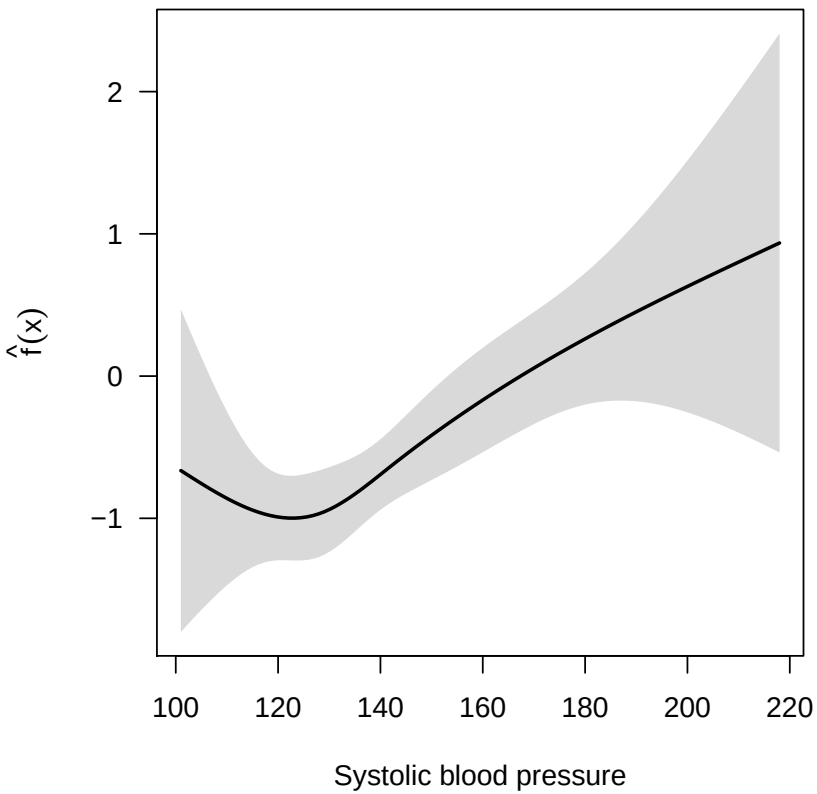
- Furthermore, extensions to logistic regression, Cox proportional hazards regression, etc., are straightforward

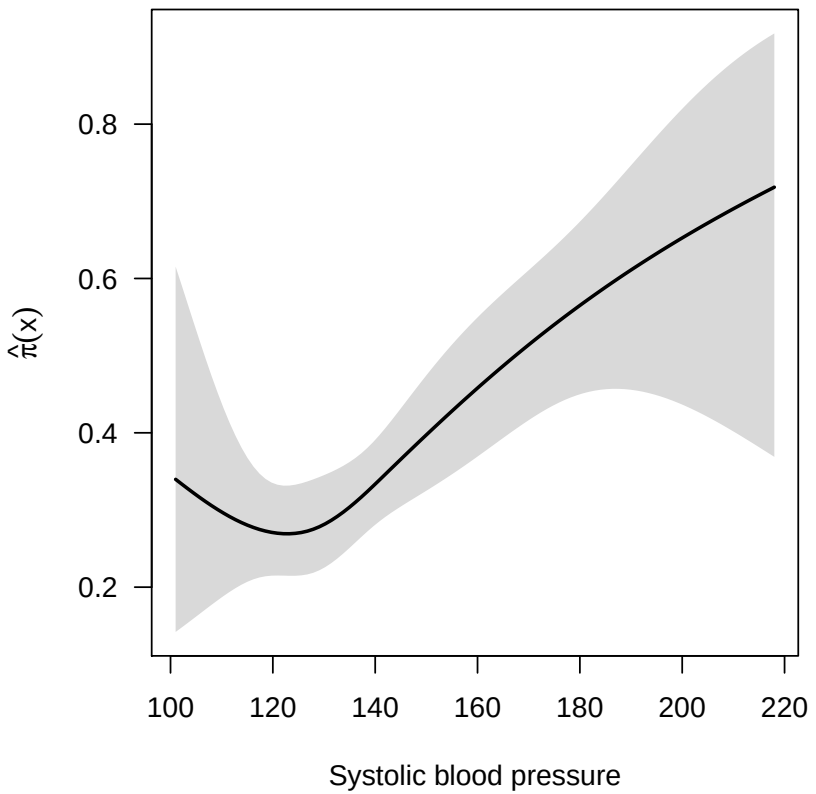
Mean and variance estimation (cont'd)



Mean and variance estimation (cont'd)

Coronary heart disease study, $K = 4$





Problems with knots

- Fixed-df splines are useful tools, but are not truly nonparametric
- Choices regarding the number of knots and where they are located are fundamentally parametric choices and have a large effect on the fit
- Furthermore, assuming that you place knots at quantiles, models will not be nested inside each other, which complicates hypothesis testing
- An alternative, more direct approach is *penalization*

Controlling smoothness with penalization

- Here, we directly solve for the function f that minimizes the following objective function, a penalized version of the least squares objective:

$$\sum_{i=1}^{n}\{y_i - f(x_i)\}^2 + \lambda \int \{f''(u)\}^2 du$$

- The first term captures the fit to the data, while the second penalizes curvature – note that for a line, $f''(u) = 0$ for all u
- Here, λ is the smoothing parameter, and it controls the tradeoff between the two terms:
  - $\lambda = 0$ imposes no restrictions and f will therefore interpolate the data
  - $\lambda = \infty$ renders curvature impossible, thereby returning us to ordinary linear regression

Controlling smoothness with penalization (cont'd)

- This avoids the knot selection problem altogether by formulating the problem in a nonparametric manner
- It may sound impossible to solve for such an f over all possible functions, but the solution turns out to be surprisingly simple: as we will show, the solution to this problem lies in the family of natural cubic splines

Terminology

First, some terminology:

- The parametric splines with fixed degrees of freedom that we have talked about so far are called *regression splines*
- A spline that passes through the points $\{x_i, y_i\}$ is called an *interpolating spline*, and is said to interpolate the points $\{x_i, y_i\}$
- A spline that describes and smooths noisy data by passing close to $\{x_i, y_i\}$ without the requirement of passing through them is called a *smoothing spline*

Natural cubic splines are the smoothest interpolators

Theorem: Out of all twice-differentiable functions passing through the points $\{x_i, y_i\}$, the one that minimizes

$$\lambda \int \{f''(u)\}^2 du$$

is a natural cubic spline with knots at every unique value of $\{x_i\}$

Natural cubic splines solve the nonparametric formulation

Theorem: Out of all twice-differentiable functions, the one that minimizes

$$\sum_{i=1}^{n}\{y_i - f(x_i)\}^2 + \lambda \int \{f''(u)\}^2 du$$

is a natural cubic spline with knots at every unique value of $\{x_i\}$

Design matrix

Let $\{N_j\}_{j=1}^n$ denote the collection of natural cubic spline basis functions and $\mathbf{N}$ denote the $n \times n$ design matrix consisting of the basis functions evaluated at the observed values:

- $\mathbf{N}_{ij} = N_j(x_i)$
- $f(x) = \sum_{j=1}^n N_j(x)\beta_j$
- $f(\mathbf{x}) = \mathbf{N}\boldsymbol{\beta}$

Solution

- **Homework:** Show that the objective function for penalized splines is

$$(\mathbf{y} - \mathbf{N}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{N}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}'\boldsymbol{\Omega}\boldsymbol{\beta},$$

  where $\boldsymbol{\Omega}_{jk} = \int N_j''(t)N_k''(t)dt$
- The solution is therefore

$$\widehat{\boldsymbol{\beta}} = (\mathbf{N}'\mathbf{N} + \lambda\boldsymbol{\Omega})^{-1}\mathbf{N}'\mathbf{y}$$

Smoothing splines are linear smoothers

- Note that the fitted values can be represented as

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{N}(\mathbf{N}'\mathbf{N} + \lambda\boldsymbol{\Omega})^{-1}\mathbf{N}'\mathbf{y} \\ &= \mathbf{L}_\lambda \mathbf{y}\end{aligned}$$

- Thus, smoothing splines are linear smoothers, and we can use all the results that we derived back when discussing local regression

Smoothing splines are linear smoothers (cont'd)

In particular:

$$CV = \frac{1}{n}\sum_i \left(\frac{y_i - \hat{y}_i}{1 - l_{ii}}\right)^2$$

$$\mathbb{E}\hat{f}(x_0) = \sum_i l_i(x_0) f(x_0)$$

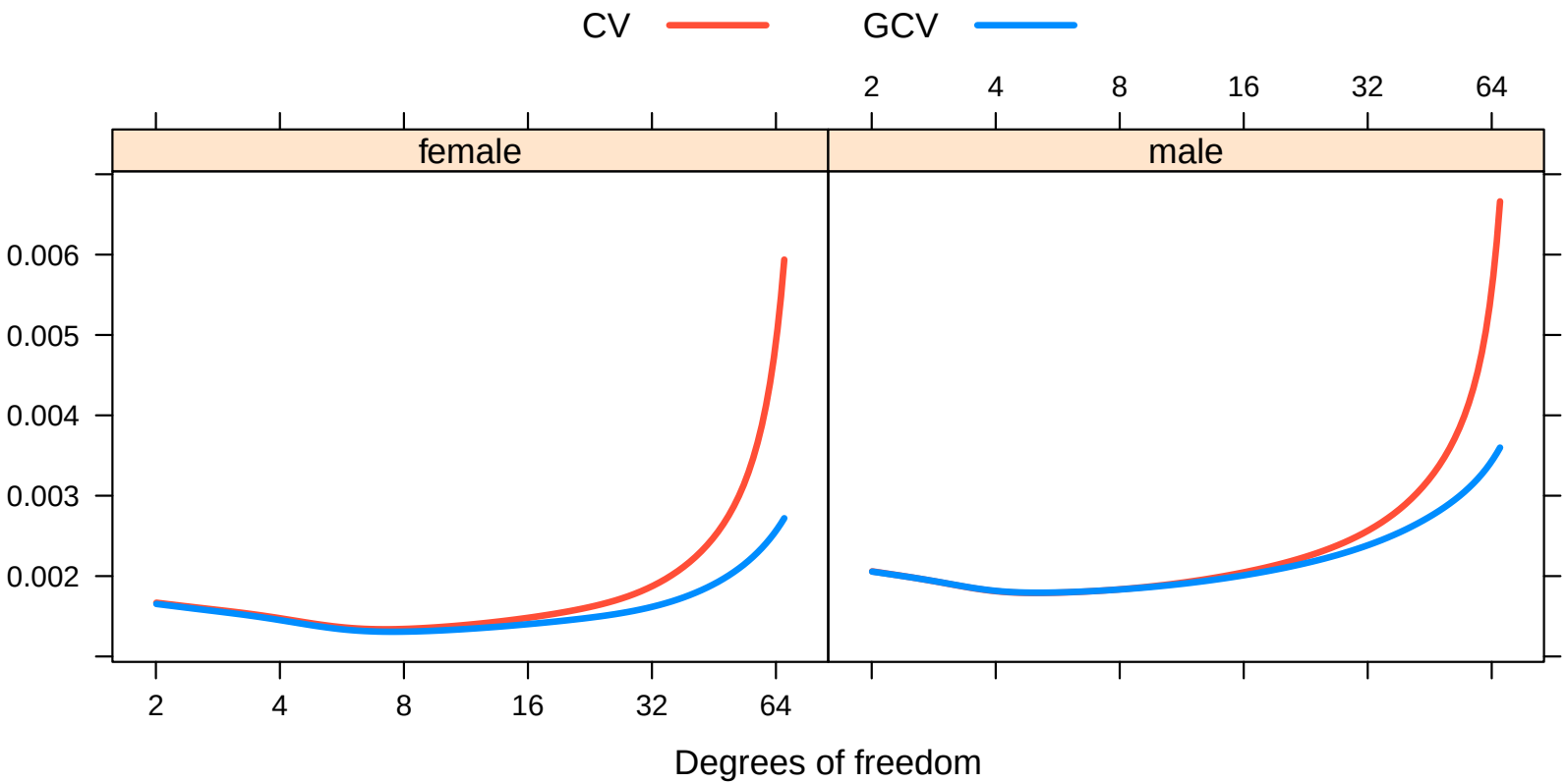
$$\mathbb{V}\hat{f}(x_0) = \sigma^2 \left\| l(x_0) \right\|^2$$

$$\hat{\sigma}^2 = \frac{\sum_i (y_i - \hat{y}_i)^2}{n - \tilde{\nu}}$$

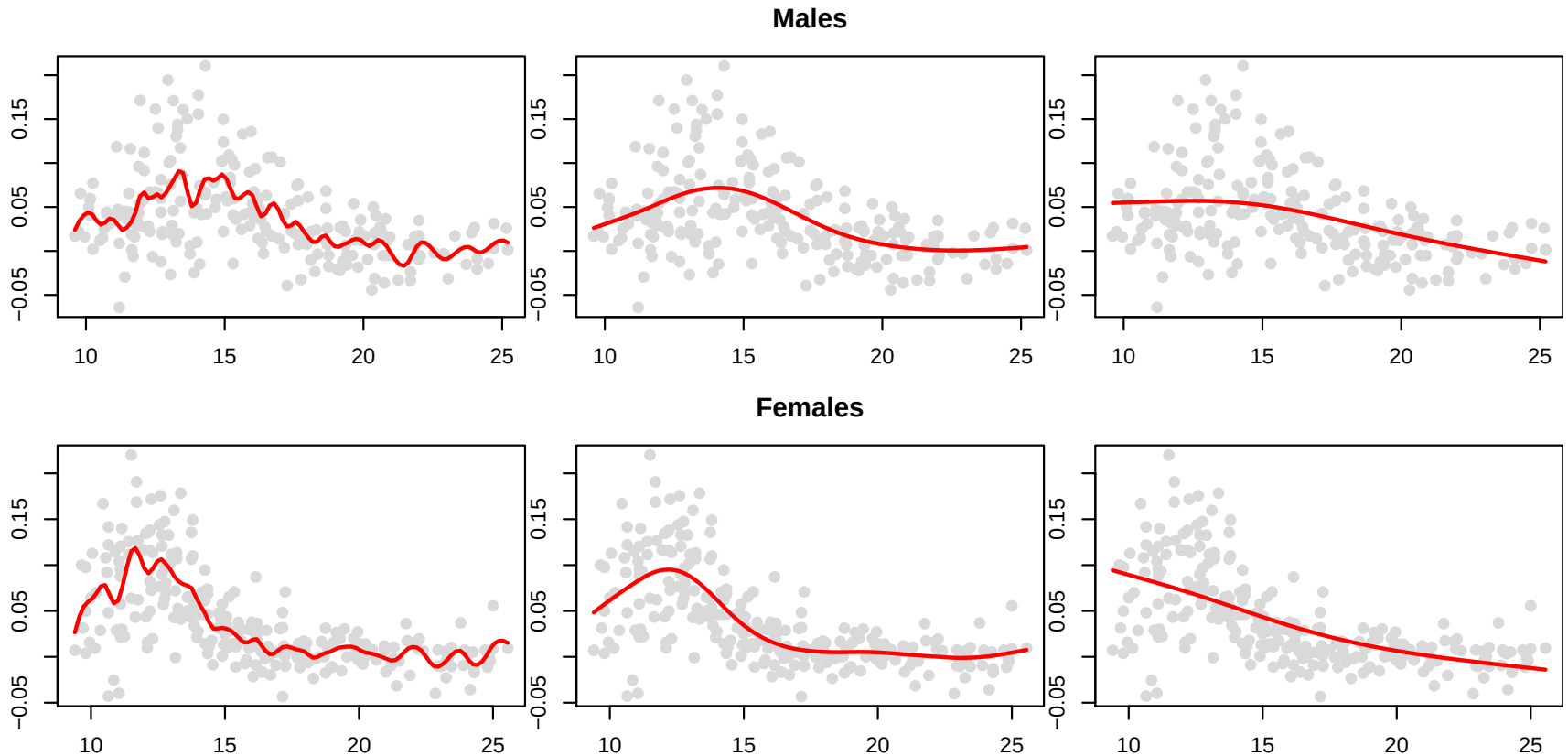
$$\nu = \mathrm{tr}(\mathbf{L})$$

$$\tilde{\nu} = 2\mathrm{tr}(\mathbf{L}) - \mathrm{tr}(\mathbf{L}'\mathbf{L})$$

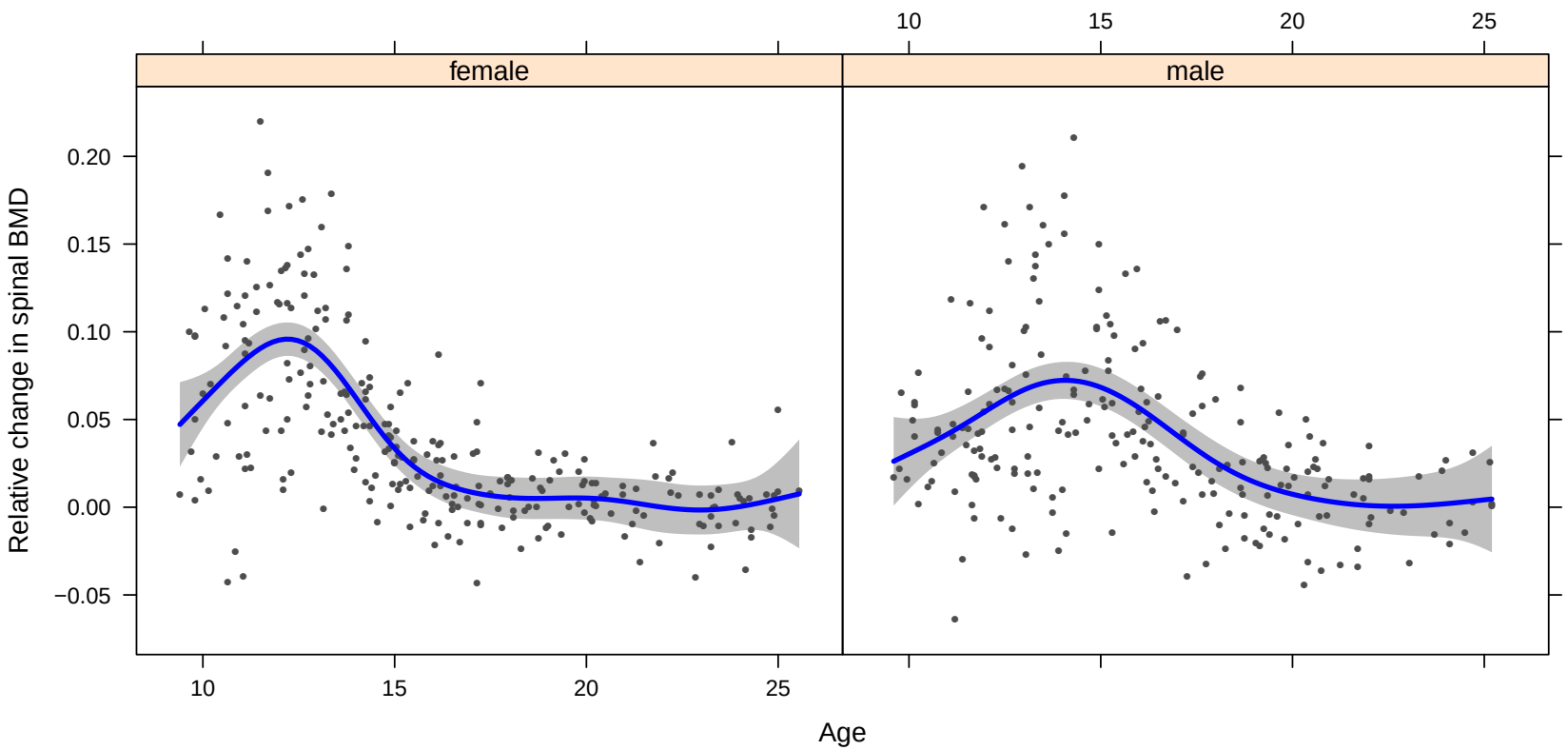
CV, GCV for BMD example



Undersmoothing and oversmoothing of BMD data



Pointwise confidence bands



R implementation

- Recall that local regression had a simple, standard function for basic one-dimensional smoothing (`loess`) and an extensive package for more comprehensive analyses (`locfit`)
- Spline-based smoothing is similar
- `smooth.spline` does not require any packages and implements simple one-dimensional smoothing:

```
fit <- smooth.spline(x,y)
plot(fit,type="l")
predict(fit,xx)
```

- By default, the function will choose λ based on GCV, but this can be changed to CV, or you can specify λ

The `mgcv` package

- If you have a binary outcome variable or multiple covariates or want confidence intervals, however, `smooth.spline` is lacking
- A very extensive package called `mgcv` provides those features, as well as much more
- The basic function is called `gam`, which stands for *generalized additive model* (we'll discuss GAMs more in a later lecture)
- Note that the main function of the `gam` package was also called `gam`; it is best to avoid having both packages loaded at the same time

The mgcv package (cont'd)

The syntax of gam is very similar to glm and locfit, with a function s() placed around any terms that you want a smooth function of:

```
fit <- gam(y~s(x))
fit <- gam(y~s(x),family="binomial")
plot(fit)
plot(fit,shade=TRUE)
predict(fit,newdata=data.frame(x=xx),se.fit=TRUE)
summary(fit)
```

Hypothesis testing

- As with local regression, we are often interested in testing whether the smaller of two nested models provides an adequate fit to the data
- As before, we may construct F tests and generalized likelihood ratio tests:

$$
\begin{aligned}
F &= \frac{(\mathrm{RSS}_0 - \mathrm{RSS}_1)/q}{\hat{\sigma}^2} \\
&\dot{\sim} F_{q,n-\nu_1} \\
\Lambda &= 2\left\{l(\widehat{\boldsymbol{\beta}}|\mathbf{y}) - l(\widehat{\boldsymbol{\beta}}_0|\mathbf{y})\right\} \\
&\dot{\sim} \chi^2_q
\end{aligned}
$$

where q is the difference in degrees of freedom between the two models

Hypothesis testing (cont'd)

- As with local regression, these tests do not strictly preserve type I error rates, but still provide a way to compute useful approximate p-values in practice
- Like the `gam` package, `mgcv` provides an `anova` method:

```
anova(fit0,fit,test="F")
anova(fit0,fit,test="Chisq")
```

- It should be noted, however, that such tests treat λ as fixed, even though in reality it is often estimated from the data using CV or GCV

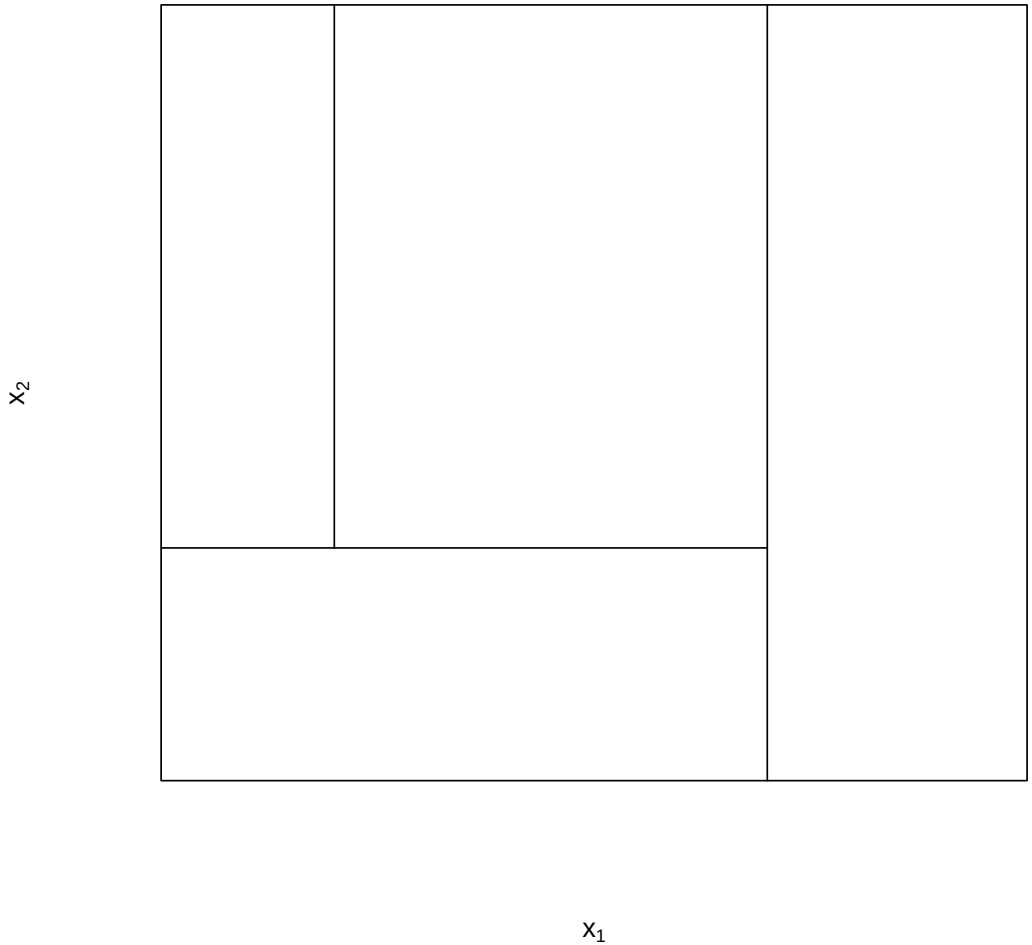
Introduction

- We've seen that local methods and splines both operate locally – either by using kernels to introduce local weights or by using piecewise basis functions
- Either way, the kernels/basis functions were prespecified – *i.e.*, the basis functions are defined and weights given to observations regardless of whether they are needed to improve the fit or not
- Another possibility is to use the data to actively seek partitions which improve the fit as much as possible
- This is the main idea behind *tree-based methods*, which recursively partition the sample space into smaller and smaller rectangles

Recursive partitioning

- To see how this works, consider a linear regression problem with a continuous response y and two predictors x_1 and x_2
- We begin by splitting the space into two regions on the basis of a rule of the form $x_j \leq s$, and modeling the response using the mean of y in the two regions
- The optimal split (in terms of reducing the residual sum of squares) is found over all variables j and all possible split points s
- The process is then repeated in a recursive fashion for each of the two sub-regions

Partitioning illustration



The regression model

- This process continues until some stopping rule is applied
- For example, letting $\{R_m\}$ denote the collection of rectangular partitions, we might continue partitioning until $|R_m| = 10$
- The end result is a piecewise constant model over the partition $\{R_m\}$ of the form

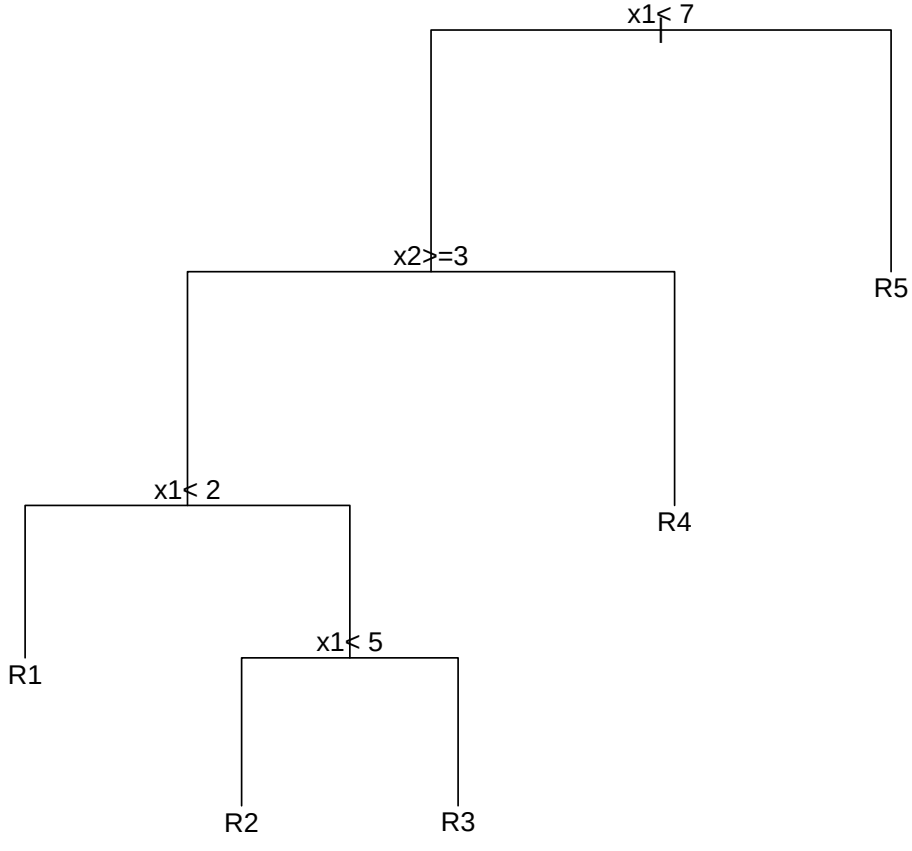
$$f(\mathbf{x}) = \sum_m c_m I(\mathbf{x} \in R_m)$$

where c_m is the constant term for the mth region (*i.e.*, the mean of y_i for those observations $\mathbf{x}_i \in R_m$)

Trees

- The same model can be neatly expressed in the form of a binary tree
- The regions $\{R_m\}$ are then referred to as the *terminal nodes* of the tree
- The non-terminal nodes are referred to as *interior nodes*
- The splits are variously referred to as "splits", "edges", or "branches"

Equivalent tree for example partition

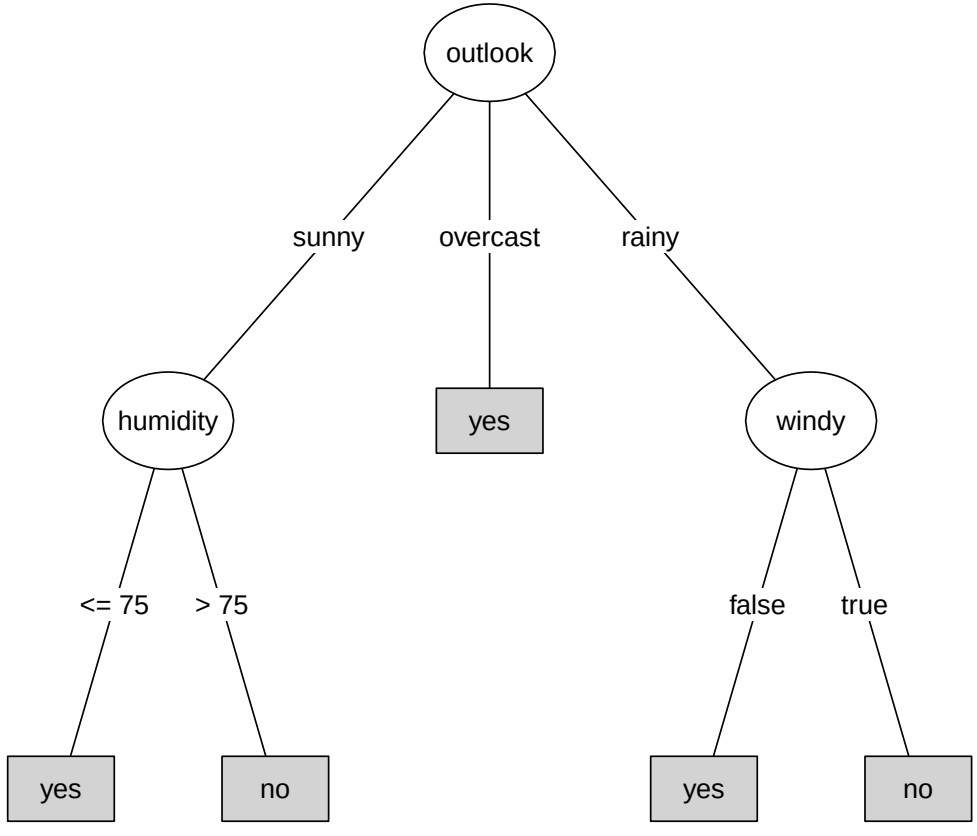


Trees and interpretability

- The ability to represent the model as a tree is the key to its interpretability and popularity
- With more than two explanatory variables, the earlier partition diagram becomes difficult to draw, but the tree representation can be extended to any dimension
- Trees are one of the most easily interpreted statistical methods: no understanding of statistics – or even mathematics – are required to follow them, and, to some extent, they mimic the way that human beings naturally think about things and make decisions

Artificial example

I know what the weather is like outside . . . should I play?



Algorithms vs. models

- Tree-based methods are not "statistical" models in the traditional sense – there is no distribution, no likelihood, no design matrix, none of the things we usually associate with modeling
- The thinking behind them is really more algorithmic, and treats the mechanism by which the data were generated as unknown and unknowable
- Admittedly, this is a bit foreign; however, in the words of Leo Breiman, one of the key pioneers of tree-based methods,

> *The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems.*

Regression trees

- We now turn to some of the details involved in fitting trees, and begin with the case where our outcome is continuous (such trees are referred to as *regression trees*)
- First, note that if we adopt the least squares criterion as our objective, then our estimate for c_m is simply the average of the y_i's in that region:

$$\hat{c}_m = \frac{\sum_i y_i I(\mathbf{x} \in R_m)}{\sum_i I(\mathbf{x} \in R_m)}$$

- Our task is then to find the optimal splitting variable j and split point s that bring about the largest drop in the residual sum of squares

Regression tree algorithm

- For a given splitting variable j, this amounts to finding the value of s that minimizes

$$\sum_{i:x_j \leq s} (y_i - \hat{c}_1)^2 + \sum_{i:x_j > s} (y_i - \hat{c}_2)^2$$

- This may seem like a burdensome task, but if x_j has been sorted already, it can be done rather quickly (Homework)
- Thus, we simply have to perform the above search for each variable j and then pick the best (j, s) pair for our split
- Having made this split, we then perform the whole process again on each of the two resulting regions, and so on

Categorical predictors

- In the preceding, we assumed that our predictors were continuous
- The exact same approach works for ordinal predictors
- For unordered categorical (*i.e.* nominal) predictors with q categories, there are $2^{q-1} - 1$ possible splits
- This actually makes things easier when q is small, but causes two problems when q is large:
  - The number of calculations grows prohibitive
  - The algorithm favors variables with a large number of possible splits, as the more choices we have, the better chance we can find one that seems to fit the data well

What size tree?

- How large should our tree be?
- A small tree might be too simple, while a large tree might overfit the data
- There are two main schools of thought on this matter:
  - The decision of whether to split or not should be based on a hypothesis test of whether the split significantly improves the fit or not
  - Tree size is a tuning parameter, and we can choose it using methods such as cross-validation

Hypothesis-testing approach

The hypothesis-testing approach is straightforward:

- Carry out an appropriate hypothesis test for each variable
- If the lowest p-value is significant after adjusting for the number of comparisons, partition the data using the optimal split for the variable with the lowest p-value
- When no significant variables can be found, stop growing the tree

Pruning

- The upside of this approach, of course, is that you get p-values for each split, and they are guaranteed to be significant
- Furthermore, it alleviates the problem alluded to earlier, whereby explanatory variables with a large number of possible splits are more likely to be selected
- One downside, however, is that a seemingly unimportant split might lead to a very important split later on
- An alternative is to "grow" a large tree, and then use a model-selection criterion to "prune" the tree back to its optimal size

Rules for growing trees

- Some common rules for when to stop growing a tree are:
  - When the number of terminal nodes exceeds some cutoff
  - When the number of observations in the terminal nodes reaches some cutoff
  - When the depth of the tree reaches a certain level
- Denote this tree, the largest tree under consideration, as T_0

Node impurity

- Now consider a subtree T that can be obtained by pruning T_0 – that is, by collapsing any number of its internal nodes
- Let $|T|$ denote the number of terminal nodes in tree T, and index those nodes with m, with node m representing region R_m
- We now define the *node impurity measure*:

$$Q_m(T) = \frac{1}{N_m} \sum_{i:x_i \in R_m} (y_i - \hat{c}_m)^2,$$

where N_m is the number of observations in node m

Cost-complexity pruning

- Finally, we define the *cost-complexity* criterion:

$$C_\alpha(T) = \sum_m N_m Q_m(T) + \alpha |T|$$

- The tuning parameter α behaves like the other regularization parameters we have seen, balancing stability (tree size) with goodness of fit
- For any given α, there is a tree T_α which minimizes the above criterion
- As the notation suggests, with $\alpha = 0$ we get T_0, the full tree
- α itself is usually chosen via cross-validation

Cotinine data

- To get a sense of how trees and their implementations in R work, we now turn to an example involving second hand smoke exposure in children
- Cotinine is a metabolite of nicotine, and is found in elevated levels in people exposed to second-hand smoke
- Measurement of cotinine requires lab tests, which cost time and money
- It is easier, of course, to simply ask parents about the extent of second hand smoke that their children are exposed to – but how accurate are their answers?

Cotinine data (cont'd)

To assess the correspondence (or lack thereof) between self-reported exposure and cotinine levels, the following variables were recorded:

- `SmokerTime`: Time spent with smokers (Daily/Intermittent/None)
- `TSHours`: Hours/day spent with a smoker
- `Nsmokers`: Number of smokers who live in the household
- `PPD`: Packs per day smoked by the household
- `PctOut`: Percentage of time spent smoking that is done outdoors
- `SHS`: Self-reported second-hand smoke exposure (None/Mild/Moderate/Heavy)

Tree packages in R

- In R, there are two primary packages one can use to fit tree-based models:
  - `rpart`, which is based on cost-complexity pruning
  - `party`, which is based on hypothesis test-based stopping
- The `party` package has considerably better tools for plotting and displaying trees, but thankfully, we can use these plotting tools to plot trees fit using `rpart` as well

Usage

- In `rpart`, the model-fitting function is `rpart`:

  ```
  fit0 <- rpart(Cotinine~., data=shs)
  ```

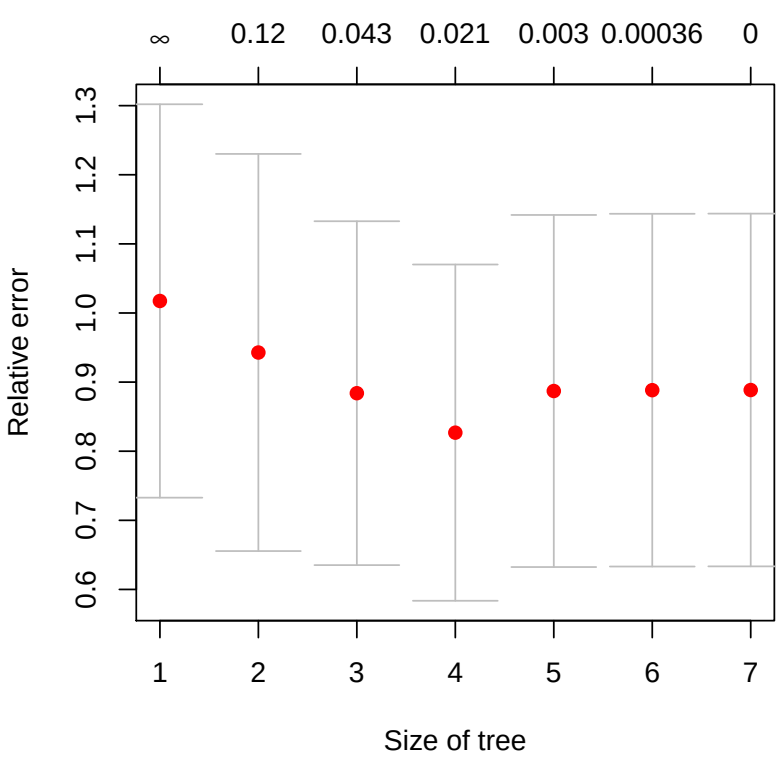
- In `party`, the model-fitting function is `ctree`:

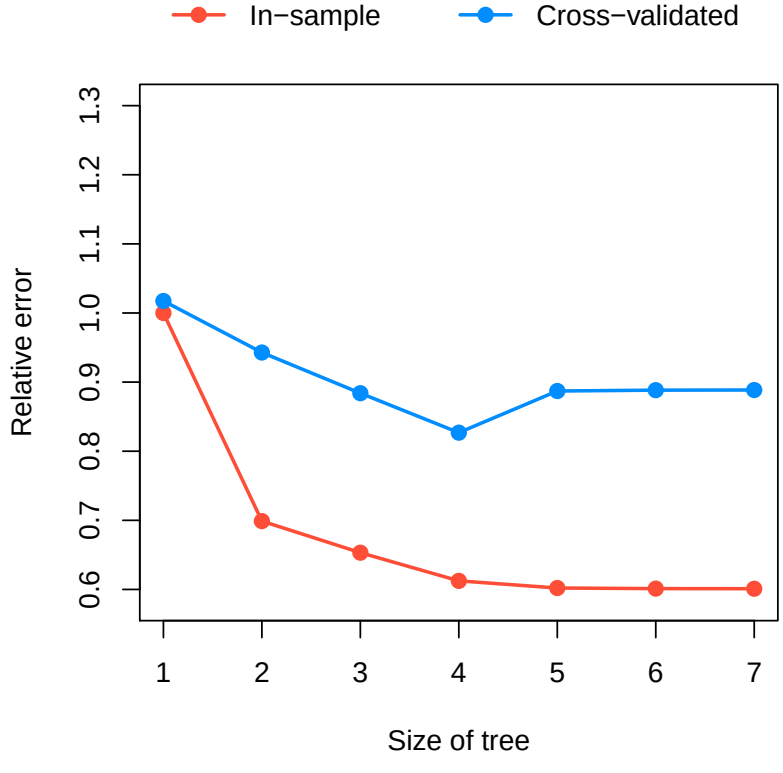
  ```
  fit <- ctree(Cotinine~., data=shs)
  ```

Cost-complexity pruning

- In `party`, the algorithm stops automatically when further splits no longer significantly improve the fit
- In `rpart`, one still has to prune the tree after it has been grown
- Thankfully, `rpart` carries out cross-validation for you and stores the results in `fit$cptable`
- α is referred to as `cp`, and the cross-validation error is `xerror`

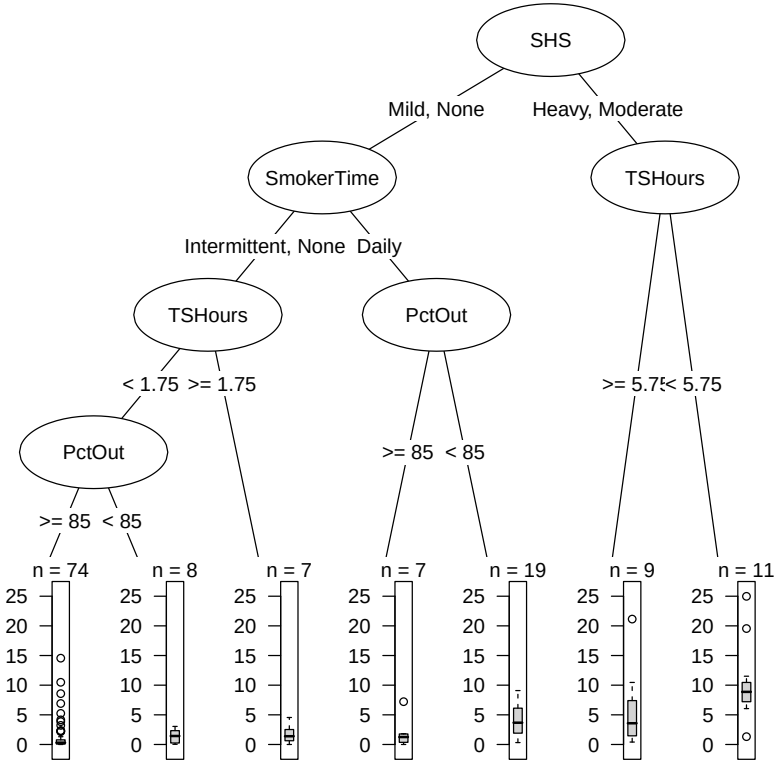
Cost-complexity pruning

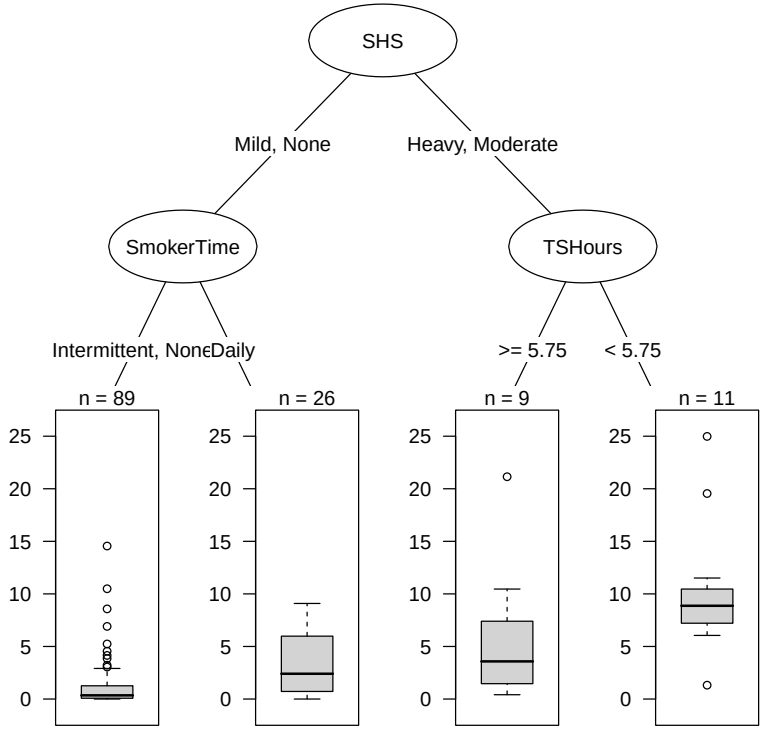





```
alpha <- fit0$cptable[which.min(fit0$cptable[,"xerror"]),"CP"]
fit <- prune(fit0,alpha)
```

Original and pruned trees





Plotting trees

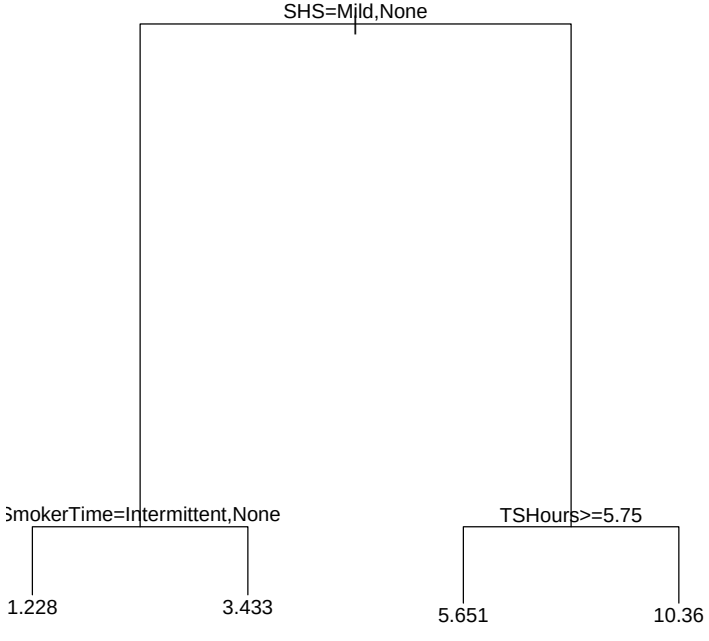
- `rpart` comes with its own plotting method:

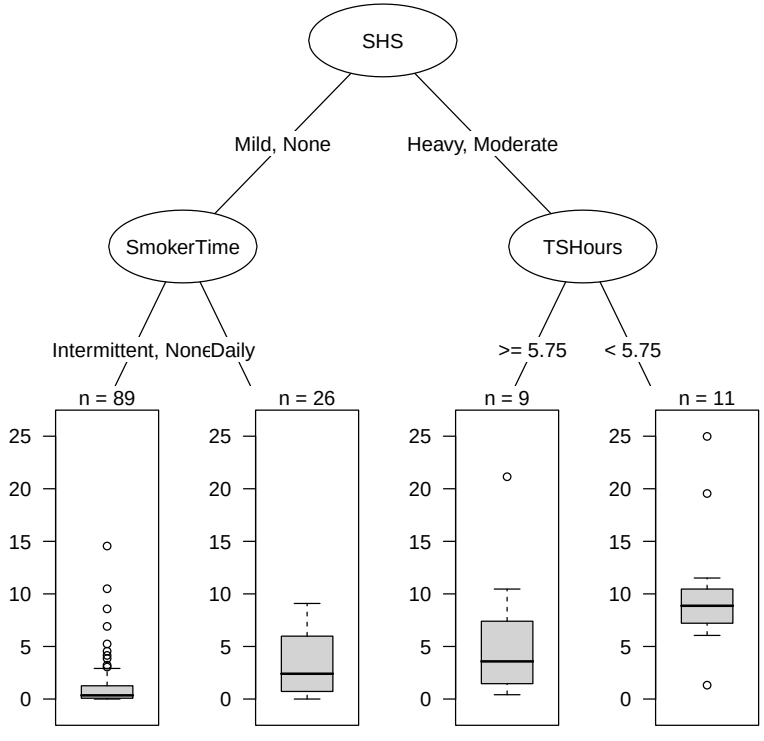
```
plot(fit)
text(fit)
```

- However, the end result is not particularly beautiful
- The plotting functions in `party` are much nicer; thankfully, you can use `party`'s tools to plot `rpart` objects using the package `partykit`:

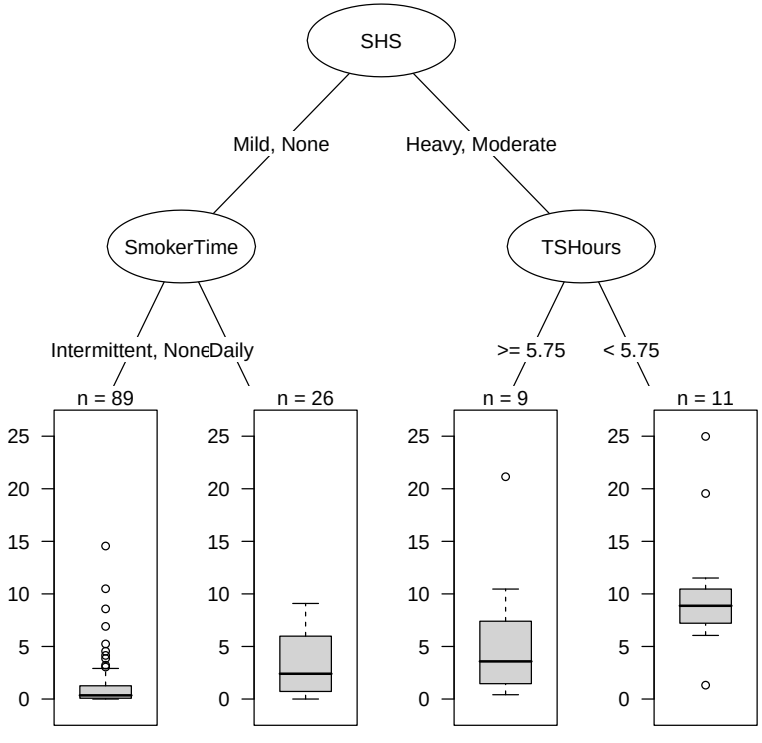
```
require(partykit)
plot(as.party(fit))
```

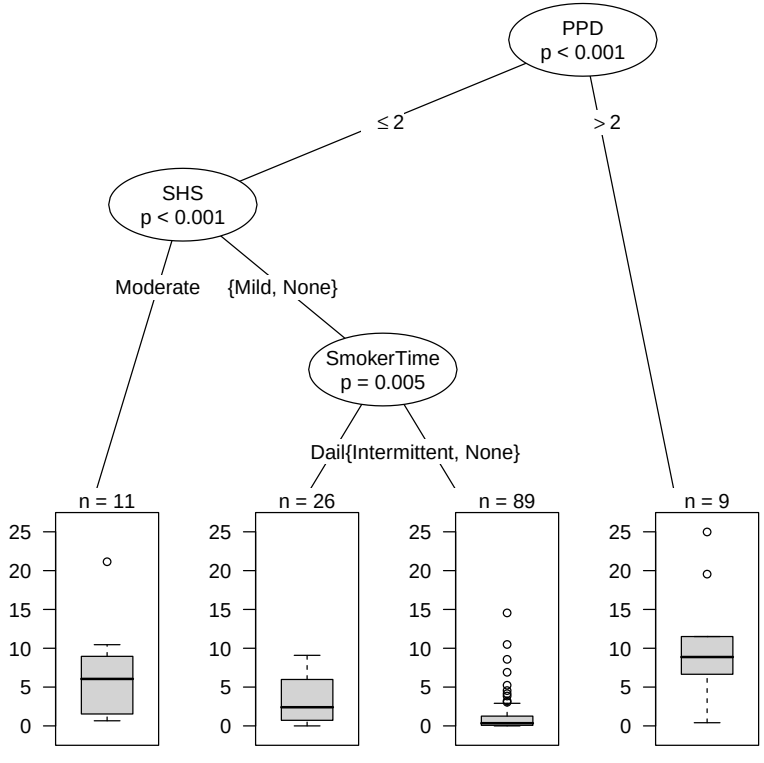
Plotting trees (cont'd)





rpart vs. ctree





Classification trees

- Tree-based methods can be extended to categorical outcomes as well; these are referred to as *classification trees*
- The main idea is the same: we recursively partition the sample space, fitting a very simple model in each partition
- In the case of classification, our model is to simply use the observed proportions, estimating $\Pr(G = k|x)$ by

$$\sum_m \hat{\pi}_{mk} I(\mathbf{x} \in R_m),$$

Node impurity

- Besides the model being fit at each node, the only other difference is the criterion used for splitting and pruning
- There are three commonly used measures of node impurity $Q_m(T)$ for classification trees:

Misclassification error: $$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq \arg\max_k \hat{\pi}_{mk})$$

Gini index: $$\sum_k \hat{\pi}_{mk}(1 - \hat{\pi}_{mk})$$

Deviance: $$-\sum_k \hat{\pi}_{mk} \log(\hat{\pi}_{mk})$$

- All three are similar, but the Gini index and deviance are differentiable, and thus easier to optimize numerically

Assumption-free

- Perhaps the primary advantage of tree-based methods is that they are virtually assumption-free
- Consequently, they are very simple to fit and interpret, since no time or effort has to go into making, checking, or explaining assumptions
- Furthermore, they are capable of discovering associations, such as higher-order interactions, that would otherwise go utterly unsuspected

Missing data

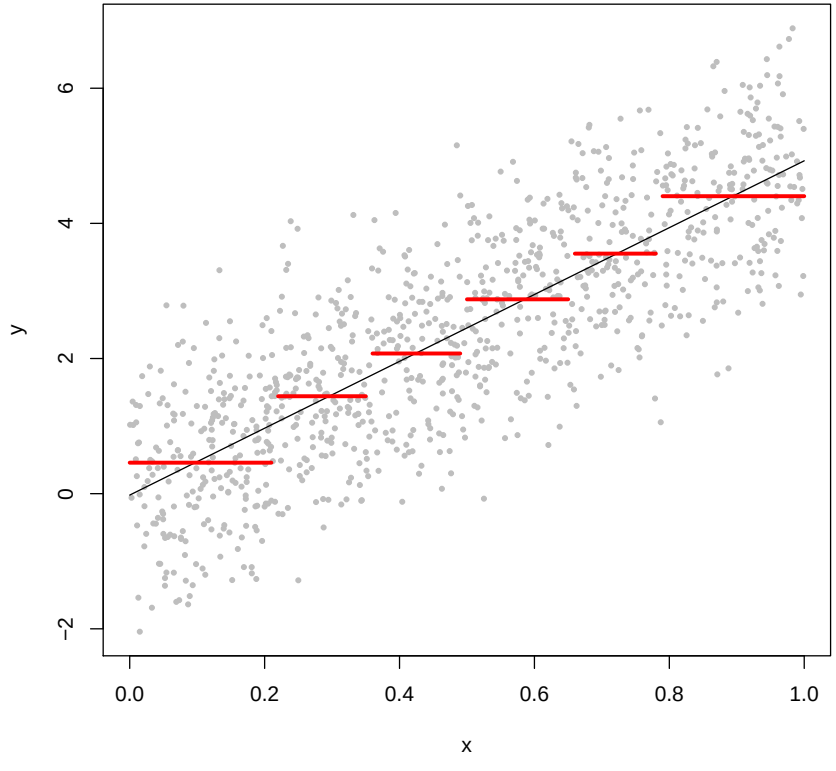
- In addition, trees have a rather elegant option for handling missing data besides the usual options of discarding or imputing observations
- For a given split, we can find *surrogate* splits, which best mimic the behavior of the original split
- Then, when sending an observation down the tree, if the splitting variable is missing, we simply use the surrogate split instead

Instability of trees

- The primary disadvantage of trees is that they are rather unstable (*i.e.*, have high variance)
- In other words, small change in the data often results in a completely different tree – something to keep in mind while interpreting trees
- One major reason for this instability is that if a split changes, all the splits under it are changes as well, thereby propagating the variability
- A related methodology, *random forests*, uses the bootstrap to grow a large number of trees and then averages across them in order to stabilize the tree-based approach, although obviously there is a cost as far as interpretation is concerned

Difficulty in capturing additive structure

In addition, trees have a difficult time capturing simple additive structures:



Concluding remarks

- In some sense, the weaknesses of tree-based methods are precisely the strengths of linear models, and vice versa
- For this reason, I personally have often found the two methods to be useful complements to each other
- For example, when embarking on an extensive analysis using a linear or generalized linear model, it doesn't hurt to check your results against a regression tree
- If you find that the regression tree chooses a very different set of important variables and achieves a much better R^2, you may want to reconsider the additive model