Week 5: Nonparametric Regression

MATH-517 Statistical Computation and Visualization

Tomas Masak

October 21st 2022

KDE

One-dimensional KDE (from last week):

$$\widehat{f}(x) = \frac{1}{nh_n} \sum_{i=1}^{n} K\left(\frac{X_i - x}{h_n}\right)$$

Multidimensional generalization (separable) when $X_1, \ldots, X_n \in \mathbb{R}^d$:

$$\widehat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{X_{i,1} - x_1}{h}\right) \cdot \ldots \cdot K\left(\frac{X_{i,d} - x_d}{h}\right)$$

Non-parametric Regression Setup

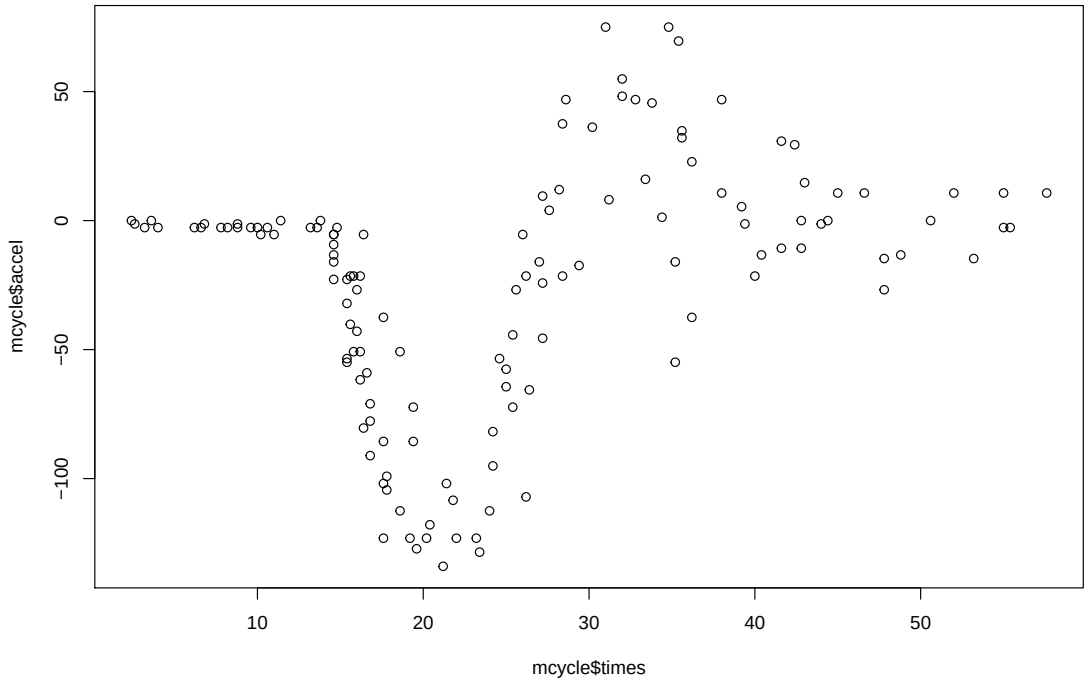
- we observe i.i.d. copies of a bivariate random vector $(X, Y)^\top$
  - a random sample $(X_1, Y_1)^\top, \ldots, (X_n, Y_n)^\top$
- we are interested in modeling the conditional expectation of the response variable Y given the single predictor variable X, i.e. the regression function

$$m(x) := \mathbb{E}\big[Y \big| X = x\big]$$

- we want to avoid parametric assumptions

Data Example

```
library(MASS)
data(mcycle)
plot(mcycle$times,mcycle$accel)
```



- head acceleration Y depending on time X in a simulated motorcycle accident use to crash test helmets

Naive Non-parametric Regression

Goal: estimate $m(x) := \mathbb{E}\left[Y \middle| X = x\right]$ from $(X_1, Y_1)^\top, \ldots, (X_n, Y_n)^\top$ i.i.d.

Since $m(x) = \int_{\mathbb{R}} y f_{Y|X}(y|x)dy = \frac{\int_{\mathbb{R}} y f_{X,Y}(x,y)dy}{f_X(x)}$ and we can now estimate densities, let's plug in those estimators

$$\widehat{f}_X(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right)$$

$$\widehat{f}_{X,Y}(x,y) = \frac{1}{nh^2}\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right) K\left(\frac{Y_i - y}{h}\right)$$

to obtain

$$\widehat{m}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right) Y_i}{\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right)}$$

Local Constant Regression

$$\widehat{m}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right) Y_i}{\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right)}$$

Since this is a weighted average, it must be a solution to weighted least squares:

$$\widehat{m}(x) = \arg\min_{\beta_0 \in \mathbb{R}} \sum_{i=1}^{\infty} K\left(\frac{X_i - x}{h}\right) (Y_i - \beta_0)^2$$

For a fixed x, this is a weighted intercept-only regression, with weights given by the kernel.

What if we went for better than intercept-only regression? We have at least one covariate X...

Local Polyomial Regression

$$\arg\min_{\beta\in\mathbb{R}^{p+1}} \sum_{i=1}^{n} [Y_i - \beta_0 - \beta_1(X_i - x) - \ldots - \beta_p(X_i - x)^p]^2 K\left(\frac{X_i - x}{h}\right)$$

Taylor expansion:

$$m(X_i) \approx m(x) + (X_i - x)m'(x) + \frac{(X_i - x)^2}{2!}m''(x) + \ldots + \frac{(X_i - x)^p}{p!}m^p(x)$$

$\widehat{\beta}_j(x)$ estimates $\frac{m^{(j)}(x)}{j!}$

$$\Rightarrow \widehat{m}(x) = \widehat{\beta}_0(x)$$

Vizualization

Manual 10 Shiny App

(rather an example than a manual)

Local Linear Regression

Choosing the order $p = 1$ leads to the local linear estimator

$$(\widehat{\beta}_0(x), \widehat{\beta}_1(x)) = \arg\min_{\mathbf{b}\in\mathbb{R}^2} \sum_{i=1}^{n} [Y_i - \beta_0 - \beta_1(X_i - x)]^2 K\left(\frac{X_i - x}{h}\right),$$

- $\widehat{m}(x) = \widehat{\beta}_0(x)$
- $\widehat{m}'(x) = \widehat{\beta}_1(x)$

Can be shown (**blackboard calculations**) that $\widehat{\beta}_0(x) = \sum_{i=1}^{n} w_{ni}(x) Y_i$, where

$$w_{ni}(x) = \frac{\frac{1}{nh} K\left(\frac{X_i - x}{h}\right) \left[S_{n,2}(x) - (X_i - x) S_{n,1}(x)\right]}{S_{n,0}(x) S_{n,2}(x) - S_{n,1}^2(x)}$$

with $S_{n,k}(x) = \frac{1}{nh} \sum_{i=1}^{n} \left(\frac{X_i - x}{h}\right)^k K\left(\frac{X_i - x}{h}\right)$

- $\sum_{i=1}^{n} w_{ni}(x) = 1$ (but not necessarily $w_{ni}(x) \geq 0$)

Bias and Variance

For local linear regression, similarly to (but more tediously than) KDE:

$$\operatorname{bias}(\widehat{m}(x)) = \frac{1}{2} m''(x) h_n^2 \int z^2 K(z) dz + o_P(h_n^2)$$

$$\operatorname{var}(\widehat{m}(x)) = \frac{\sigma^2(x) \int [K(z)]^2 dz}{f_X(x) n h_n} + o_P\left(\frac{1}{n h_n}\right)$$

where $\sigma^2(x) = \operatorname{var}(Y_1 | X_1 = x)$ is the local variance.

- for other orders, similar expressions can be obtained

Bandwidth Selection

Similarly to what we did last week with KDEs, we consider

$$MSE(\widehat{m}(x)) = \mathrm{var}(\widehat{m}(x)) + \left[\mathrm{bias}(\widehat{m}(x))\right]^2$$

and, dropping the little-o terms, we obtain

$$AMSE(\widehat{m}(x)) = \frac{\sigma^2(x)\int[K(z)]^2dz}{f_X(x)nh_n} + \frac{1}{4}\left[m''(x)\right]^2 h_n^4 \left(\int z^2 K(z)dz\right)^2.$$

Now, a local bandwidth choice can be obtained by optimizing AMSE. Taking derivatives and setting them to zero, we obtain

$$h_{opt}(x) = n^{-1/5}\left[\frac{\sigma^2(x)\int[K(z)]^2dz}{m''(x)f_X(x)\int z^2K(z)dz}\right]^{1/5}.$$

Bandwidth Selection

$$h_{opt}(x) = n^{-1/5} \left[\frac{\sigma^2(x) \int [K(z)]^2 dz}{\left[m''(x) \int z^2 K(z) dz \right]^2 f_X(x)} \right]^{1/5}.$$

This is somewhat more complicated compared to the KDE case, because we have to estimate

- the marginal density $f_X(x)$,
  - let's say that we already know how to do this, e.g. by KDE even though that requires a choice of yet another bandwidth
- the local variance function $\sigma^2(x)$, and
- the second derivative of the regression functions $m''(x)$.

Again, like in the case of KDEs, the global bandwidth choice can be obtained by integration:

- calculate $AMISE(\hat{m}) = \int AMSE(\hat{m}(x)) f_X(x) dx$, and
- set $h_{opt} = \arg\min_{h>0} AMISE(\hat{m})$.

Rule of Thumb Algorithm

starting from some initial bandwidth:

- use a higher order p (e.g. $p = 3$) to estimate $m''(x)$ with an inflated value of your current bandwidth
- assume that $\sigma^2(x) = \sigma^2$ is constant and estimate it from the residuals as $\frac{1}{n-d} \sum_{i=1}^{n} [Y_i - \widetilde{m}''(X_i)]$, where $\widetilde{m}''$ is the estimator from the previous step and d is the degrees of freedom of $\widetilde{m}''$
  - defined the same as last week
- update the current bandwidth by the $h_{opt}(x)$ formula above

and iterate the steps above couple of times

- it is not possible to prove convergence
- showcases the importance of computational considerations
  - similar situation to KDEs, local linear regression is a *linear smoother*, FFT can be used to speed up computations

Why Local Linear?

Bias and variance can be calculated similarly also for higher order local polynomial regression estimators. In general:

- bias decreases with an increasing order
- variance increases with increasing order, but only for $p = 2k + 1 \rightarrow p + 1$, i.e. when increasing an odd order to an even one

For this reason, odd orders are preferred to even ones

- $p = 1$ is easy to grasp as it corresponds to locally fitted simple regression line
- increasing p has a similar effect to decreasing the bandwidth h
  - hence $p = 1$ is usually fixed and only h is tuned

Section 1

Other Smoothers (Informatively)

`loess()` and `lowess()`

`loess()` (LOcal regrESSion) should be local linear regression with tricube kernel

`lowess()` (LOcally WEighted Scatterplot Smoothing) is an iterative algorithm:

- fit `loess()`
- repeat
  - calculate residuals of the current fit
  - calculate weights based on the residuals
  - fit weighted `loess()`
- until convergence (or just 3-times)

The goal of `lowess()` is to improve `loess()` by making it *robust* (immune) to outliers.

Smoothing Splines

$$\arg\min_{g \in C^2} \sum_{i=1}^{n} \left[Y_i - m(X_i) \right]^2 + \lambda \int \left[m''(x) \right]^2 dx$$

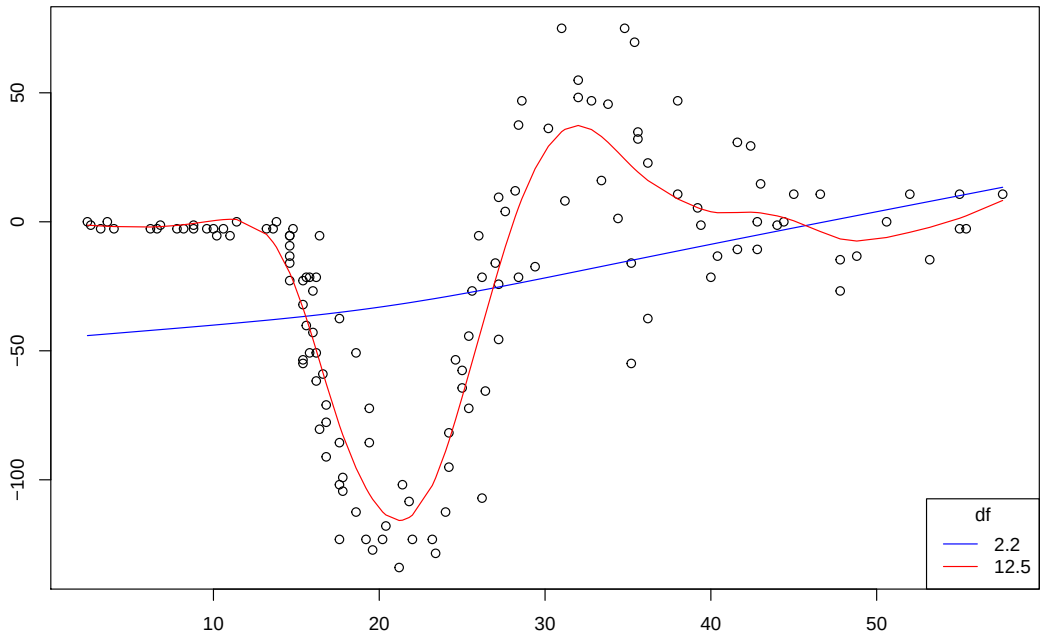
- Gaussian likelihood + smoothness penalty
- $\lambda > 0$ controls the trade-off between fit and smoothness

Unique solution: the **natural cubic spline**

- piece-wise polynomial of degree three between
- knots at X_i, $i = 1, \ldots, n$
- two continuous derivatives at the knots

Smoothing Spline: Example

```
spl1 <- smooth.spline(mcycle$times,mcycle$accel,lambda=1)
spl2 <- smooth.spline(mcycle$times,mcycle$accel,lambda=0.0001)
plot(mcycle$times,mcycle$accel,xlab="",ylab="")
lines(spl1,col="blue")
lines(spl2,col="red")
legend("bottomright",legend=round(c(spl1$df,spl2$df),1),
       title="df",lty=c(1,1),col=c("blue","red"))
```



Smoothing Splines

A natural cubic spline g with n knots can be expressed w.r.t. the natural spline basis $\{e_j\}$ (something specific, doesn't matter what it is) as

$$m(x) = \sum_{j=1}^{n} \gamma_j e_j(x)$$

Let

- $\gamma = (\gamma_j) \in \mathbb{R}^n$
- $E := (e_{ij}) := (e_j(x_i))_{i,j=1}^{n} \in \mathbb{R}^{n \times n}$ and
- $\Omega = (\omega_{ij}) \in \mathbb{R}^{n \times n}$ with $\omega_{ij} = \int e_i''(x) e_j''(x) dx$

Then

$$\sum_{i=1}^{n} \left[Y_i - m(X_i) \right]^2 + \lambda \int \left[m''(x) \right]^2 dx \quad \equiv \quad (Y - E\gamma)^\top (Y - E\gamma) + \lambda \gamma^\top \Omega \gamma$$

Smoothing Splines

$$\sum_{i=1}^{n} \left[Y_i - m(X_i)\right]^2 + \lambda \int \left[m''(x)\right]^2 dx \quad \equiv \quad (Y - E\gamma)^\top (Y - E\gamma) + \lambda \gamma^\top \Omega \gamma$$

- $\widehat{\gamma} = (E^\top E + \lambda\Omega) E^\top Y$
- $\widehat{Y} = B\widehat{\gamma} = S_\lambda Y$ with $S_\lambda = (E^\top E + \lambda\Omega) E^\top$

$\Rightarrow$ smoothing splines are linear smoothers

- degrees of freedom: $\mathrm{df} = \mathrm{tr}(S_\lambda)$
- 1-1 correspondence between df and λ
  - specifying df easier for interpretation
- bias-variance trade-off

Orthogonal Series

- take a pre-defined set of orthogonal functions $\{e_j\}_{j=1}^{\infty}$
  - customarily some basis (for some space), e.g. Fourier basis, B-splines, etc.
- truncate it to $\{e_j\}_{j=1}^{p}$
- approximate $m(x) \approx \sum_{j=1}^{p} \gamma_j e_j(x)$

Then estimate m by least-squares:

$$\arg\min_{\gamma \in \mathbb{R}^p} \sum_{i=1}^{n} \left[Y_i - \gamma_1 e_1(x) - \ldots - \gamma_p e_p(x) \right]^2$$

- just a single linear regression
- no penalty term, simplicity achieved via truncation
  - bias-variance trade-off controlled by the choice of p
  - can be related to smoothness when e_j's get more wiggly with increasing j (which is typical for most bases)
  - choices = assumptions

Assignment 3 [5 %]

Create a Shiny App or (alternatively) an R Markdown report exploring the (local, asymptotically optimal) bandwidth choice $h_{opt}(x)$ on slides 11 and 12 on simulated data example:

- locations X from beta distribution `rbeta(n,shape1,shape2)`
- response values $Y = m(X) + \epsilon$ where
  - the regression function m is given by `sin(1/(x/3+0.1))`
  - $\epsilon \sim \mathcal{N}(0, \sigma^2)$
- fix σ^2 at some visually appealing value (e.g. $\sigma^2 = 1$ should be fine)
- explore how $h_{opt}(x)$ behaves depending on the following parameters (in case of a Shiny App, these will be sliders for the user to tweak):
  - sample size `n`
  - location `x`
  - beta parameters `shape1` and `shape2`
- visualize (local, asymptotically optimal) bandwidth
  - also find a way to visually incorporate the beta density