Chapter 4: Kernel Smoothing

DJM, Revised NAK

13 February 2017

A Simplified Workflow for Doing Statistics

1. Choose a set of models.
2. Split the data in half (randomly): Set A, Set B
3. For each potential model, Use Set A to do the following:
    1. Use CV to estimate risk.
    2. Adjust/optimize tuning parameters to lower risk.
4. Choose a model by picking the **model** with the lowest estimate of the risk.
5. Evaluate your model on Set B, and describe your model. Make plots, interpret coefficients, make predictions, etc.
6. If you see things if 5 you don't like, propose a new model(s) to handle these issues and return to step 3.

"Smoothers"

Objective: We want to estimate $\mu(x) = \mathbb{E}[Y|X = x]$.

We have our data: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

The general form of a "smoother" in regression is the following:

$$\widehat{\mu}(x) = \sum_{i=1}^{n} y_i w(x, x_i, h)$$

Essentially, we take a weighted average of y_i values with x_i's close to x.

Closeness is defined by h which is often called the **bandwidth**.

Linear smoothers

- Recall S431:

  The "Hat Matrix" puts the hat on Y: $\widehat{Y} = HY$.

- If I want to get fitted values from the linear model

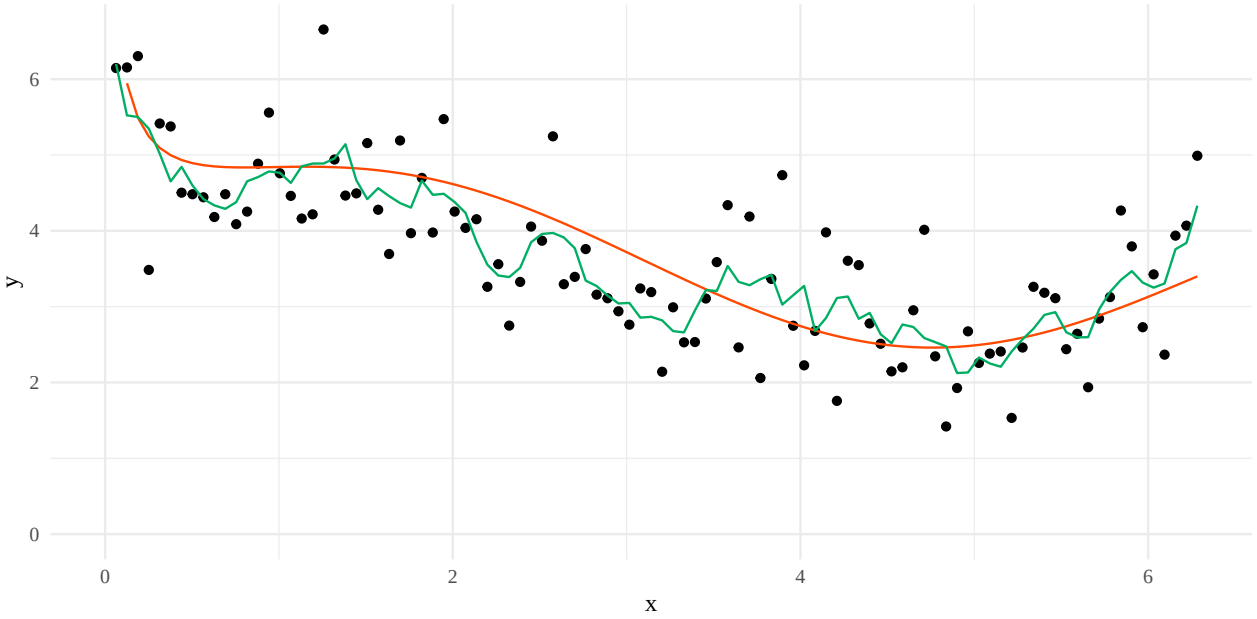
$$\underline{\widehat{Y}} = \underline{X}\underline{\widehat{\beta}} = \left[\underline{X}(\underline{X}^\top \underline{X})^{-1}\underline{X}^\top\right]\underline{Y} = \underline{HY}$$

- We generalize this to arbitrary matrices:

  A linear smoother is any predictor $\widehat{\mu}$ that gives fitted values via $\widehat{\mu}(X) = WY$.

- Today, we will learn other ways of predicting Y from X.
- If I can get the fitted values at my original datapoints X by multiplying Y by a matrix, then that is a linear smoother.

Example



At each x, find 2 points on the left, and 2 on the right. Average their y values with that of your current point.

```
W = toeplitz(c(rep(1,3),rep(0,n-3)))
W = sweep(W, 1, rowSums(W), '/')
df$Yhat = W %*% df$y
geom_line(mapping = aes(x,Yhat), color=green)
```

This is a linear smoother. What is W?

What is W?

- An example with a 10 x 10 matrix:

```
W = toeplitz(c(rep(1,3),rep(0,7)))
round(sweep(W, 1, rowSums(W), '/'), 2)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,] 0.33 0.33 0.33 0.00  0.0  0.0 0.00 0.00 0.00  0.00
##  [2,] 0.25 0.25 0.25 0.25  0.0  0.0 0.00 0.00 0.00  0.00
##  [3,] 0.20 0.20 0.20 0.20  0.2  0.0 0.00 0.00 0.00  0.00
##  [4,] 0.00 0.20 0.20 0.20  0.2  0.2 0.00 0.00 0.00  0.00
##  [5,] 0.00 0.00 0.20 0.20  0.2  0.2 0.20 0.00 0.00  0.00
##  [6,] 0.00 0.00 0.00 0.20  0.2  0.2 0.20 0.20 0.00  0.00
##  [7,] 0.00 0.00 0.00 0.00  0.2  0.2 0.20 0.20 0.20  0.00
##  [8,] 0.00 0.00 0.00 0.00  0.0  0.2 0.20 0.20 0.20  0.20
##  [9,] 0.00 0.00 0.00 0.00  0.0  0.0 0.25 0.25 0.25  0.25
## [10,] 0.00 0.00 0.00 0.00  0.0  0.0 0.00 0.33 0.33  0.33
```

- This is a simplification of the idea of "kernel smoothing".

Kernel Smoothing

- The mathematics:

  A kernel (in statistics) is typically a fancy word for a well-behaved, symmetric probability density function.

The usual assumptions on a kernel K, are the following:

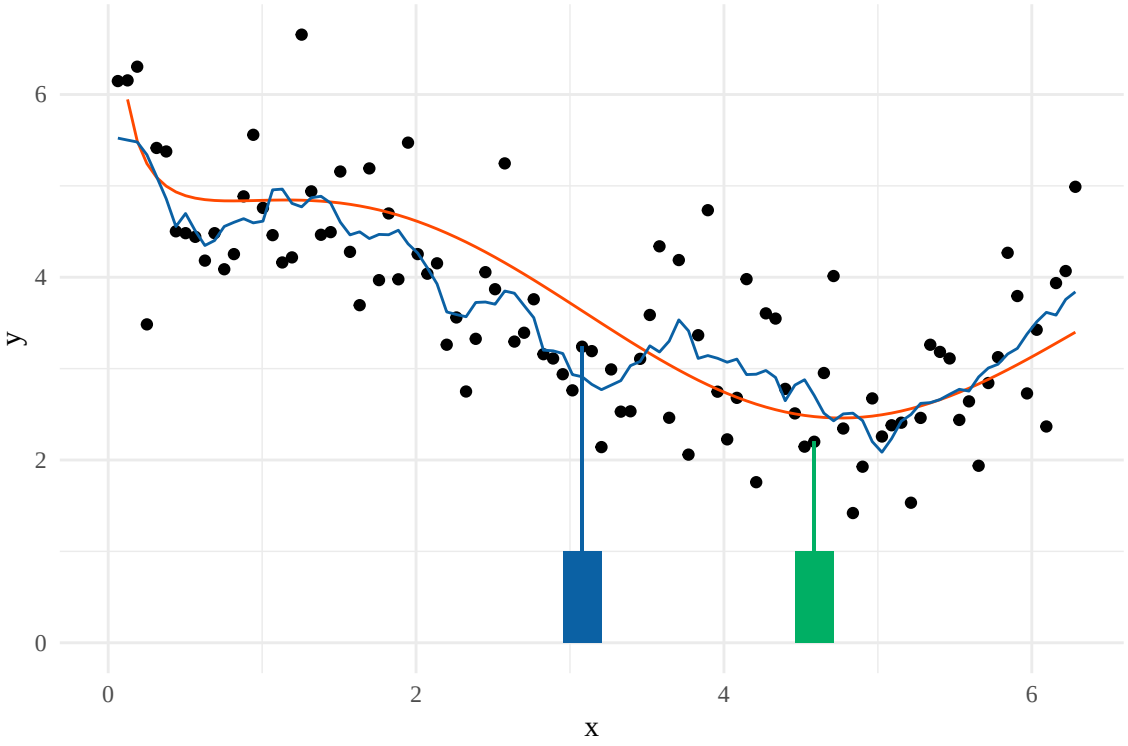
- $K(u) \geq 0$ (Non-negativity)
- $\int K(u)\, du = 1$
- $K(-u) = K(u)$ (Symmetry about 0, and usually unimodal)
- $\int u^2 K(u)\, du < \infty$ (Finite second moment/variance)

The idea: a kernel is a nice way to take weighted averages. The kernel function gives the weights.

The kernel smoother formula is

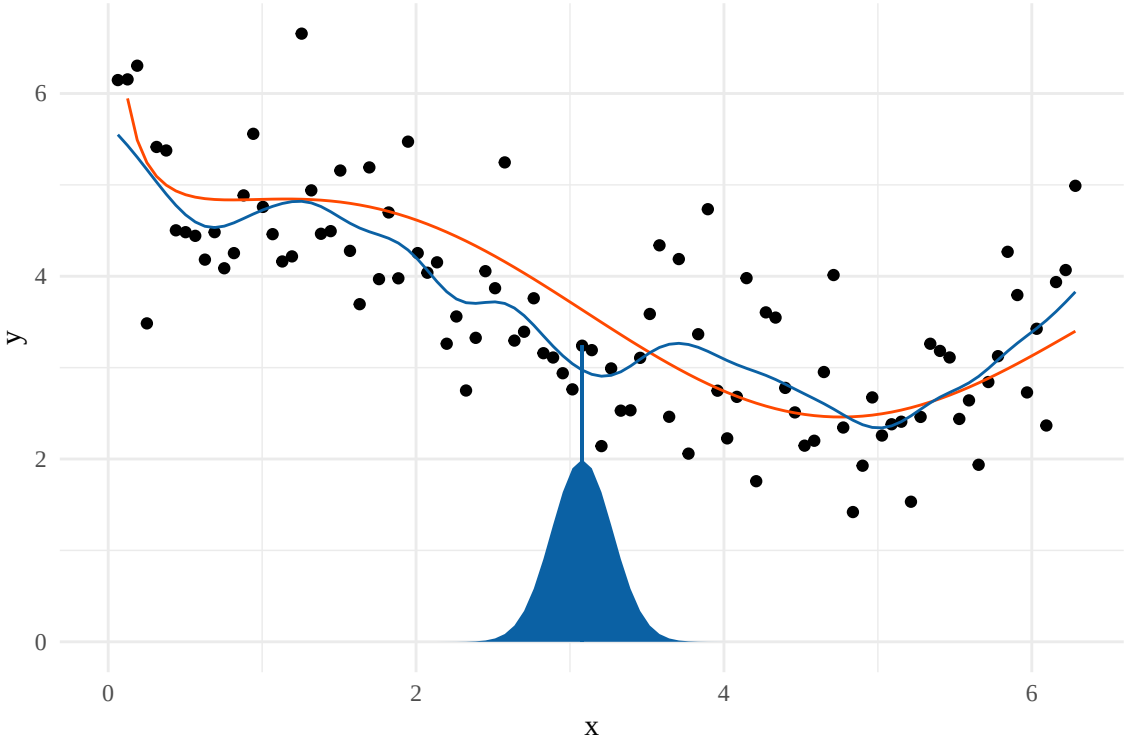
$$\widehat{\mu}(x) = \sum_{i=1}^{n} y_i w(x, x_i, h) = \sum_{i=1}^{n} \frac{y_i K\left(\frac{x-x_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)}$$

- The previous example is related to using the **boxcar** kernel.
- A box of a specified width is placed around the chosen estimation point x; all the points inside the box get the same weight, all the rest get 0.
- What is the the probability distribution associated with the boxcar kernel?



Other kernels

- Most of the time, we don't use the boxcar kernel because it gives jagged estimates.
- A much more common kernel is the Gaussian kernel:

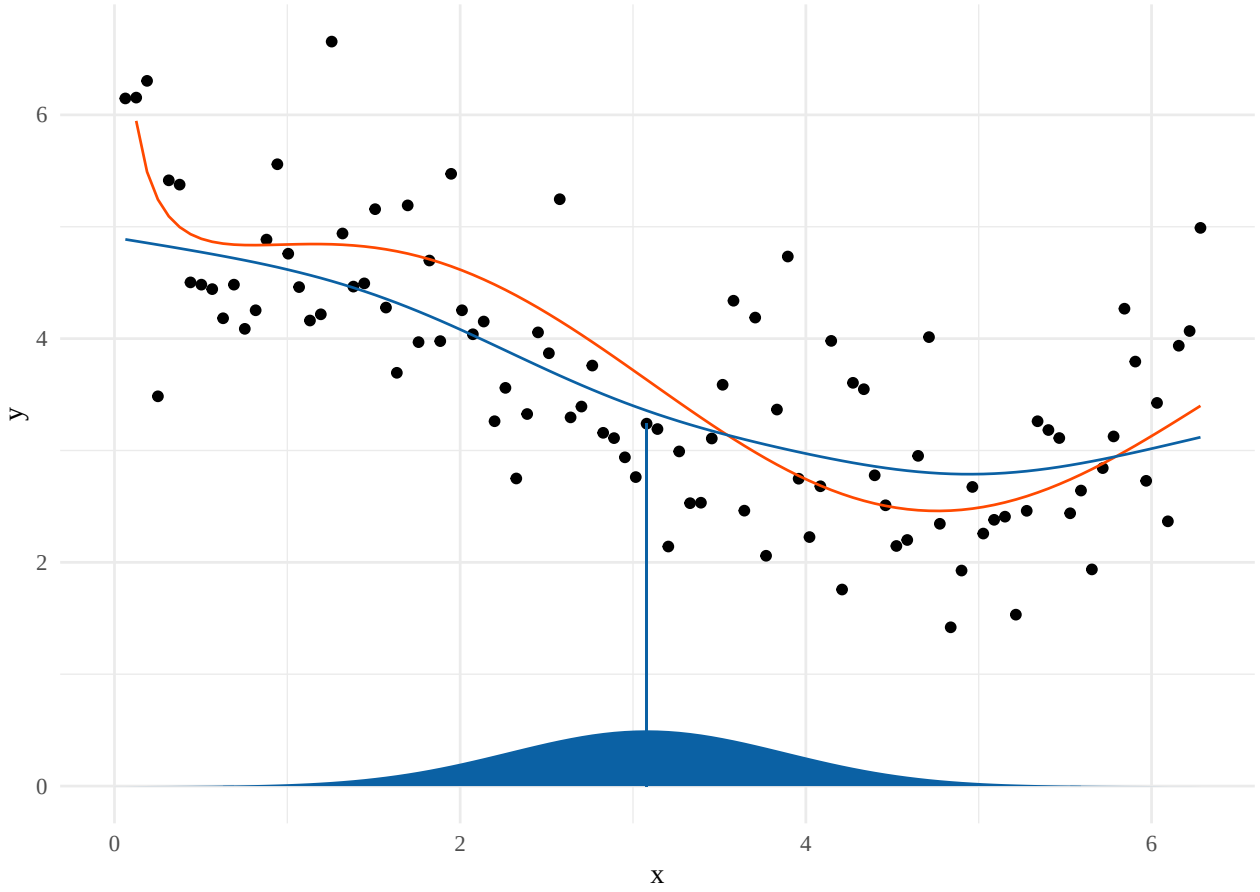


- Let's look at row 49 of the W matrix here:

$$W_{49,j} = \frac{1}{\sqrt{2\pi h^2}} \exp\left(-\frac{1}{2h^2}(x_j - x_{49})^2\right)$$

- For the plot, $h = .2$.

- What if I made $h = 0.8$?

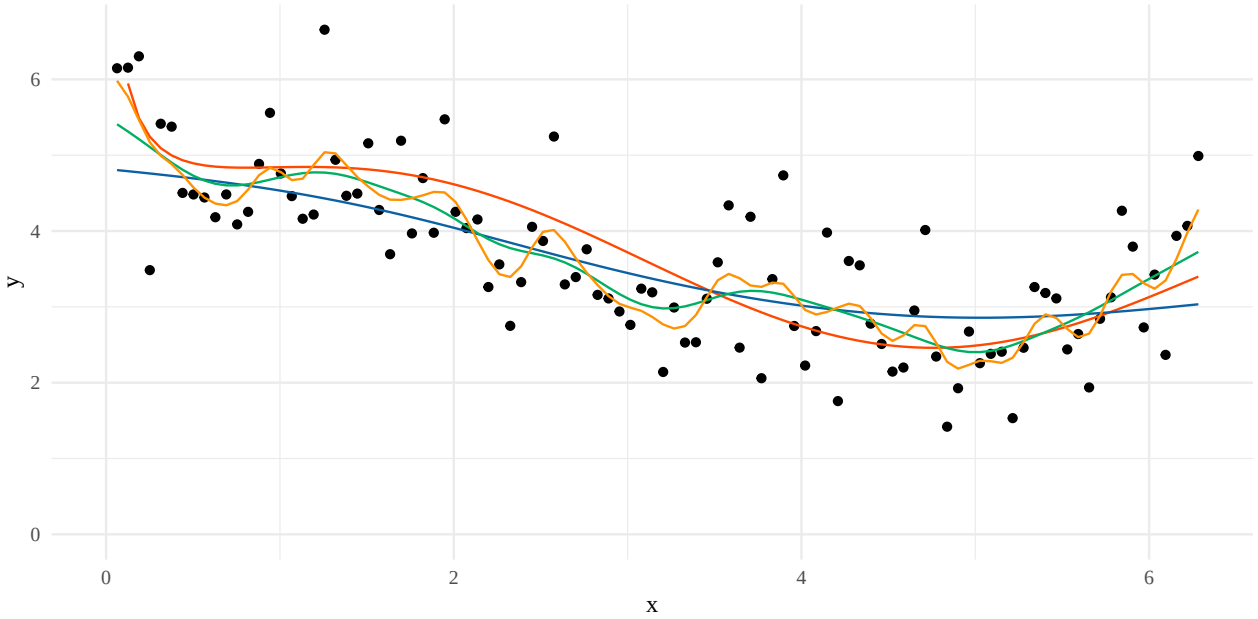


- Before, points far from x_{49} got very small weights for predicting at x_{49}, now they have more influence.
- For the Gaussian kernel, h determines something like the "range" of the smoother.

Attempting Several Bandwidths

- Using this formula for W, we can calculate different linear smoothers with different h

```
df$W1 = with(df, Wgauss(1) %*% y)
df$W.25 = with(df, Wgauss(.25) %*% y)
df$W.1 = with(df, Wgauss(.1) %*% y)
ggplot(df, aes(x, y)) + geom_point() + xlim(0,2*pi) + ylim(0,max(df$y)) +
  stat_function(fun=trueFunction, color=red) +
  geom_line(aes(x, W1), color=blue) +
  geom_line(aes(x, W.25), color=green) +
  geom_line(aes(x, W.1), color=orange)
```



- Which ones seems best?

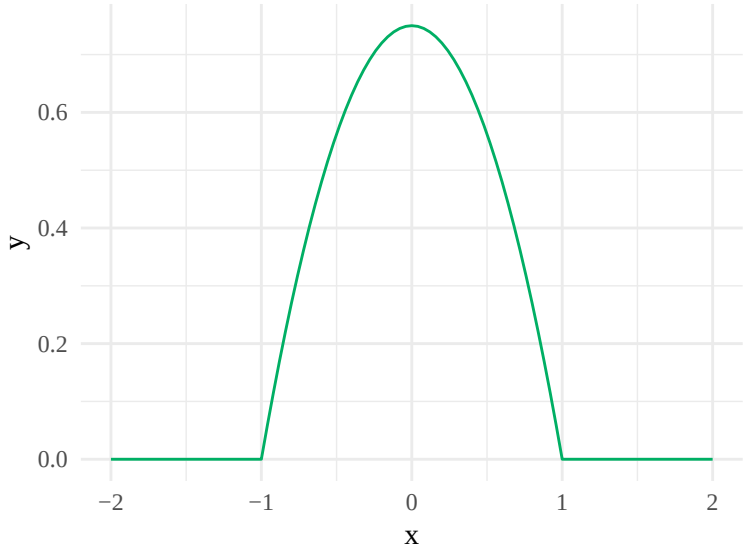
Kernel Choice vs. Bandwidth Choice

- Choosing the bandwidth h is **very** important.
- It essentially decides the effective range of the kernel being used. Its effect changes depending on the Kernel.
- There are many potential kernels we can choose from, but choice of h is usually much much more important.
- An "optimal" kernel is called the **Epanechnikov** kernel:

$$K(u) = \frac{3}{4}(1 - u^2)$$

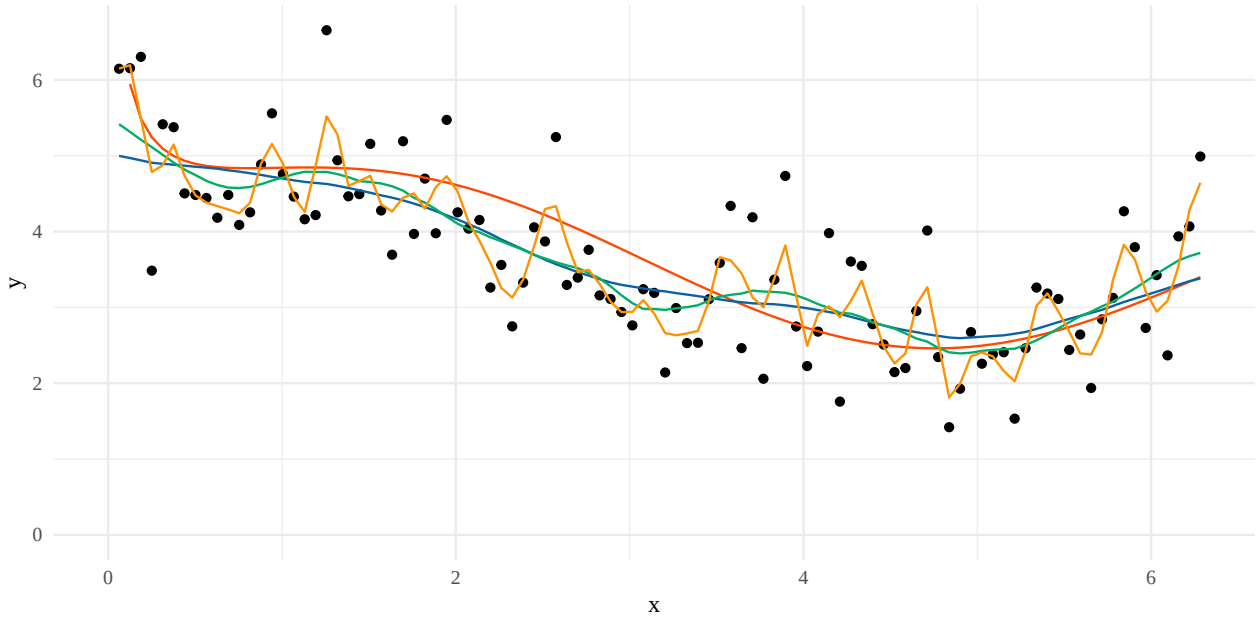
Note that this function only takes inputs from -1 to 1. It is 0 otherwise.

```
epan <- function(x) 3/4*(1-x^2)*(abs(x)<1)
ggplot(data.frame(x=c(-2,2)), aes(x)) + stat_function(fun=epan,color=green)
```



Example using Epanechnikov Kernel

```
dmat = as.matrix(dist(x))
Wepan <- function(h){
  gg = 3/(4*h)*(1-(dmat/h)^2)*((dmat/h)<1)
  sweep(gg, 1, rowSums(gg),'/')
}
df$W1 = with(df, Wepan(1) %*% y)
df$W.5 = with(df, Wepan(.5) %*% y)
df$W.1 = with(df, Wepan(.1) %*% y)
ggplot(df, aes(x, y)) + geom_point() + xlim(0,2*pi) + ylim(0,max(df$y)) +
  stat_function(fun=trueFunction, color=red) +
  geom_line(aes(x, W1), color=blue) +
  geom_line(aes(x, W.5), color=green) +
  geom_line(aes(x, W.1), color=orange)
```



Choosing the Bandwidth

- This is a topic with *tons* of publications and debates...
- One of the most straightforward approaches is Cross Validation. First, lets look at how Leave One Out Cross Validation can be applied to linear smoothers.
- The trick:

  For linear smoothers, one can show (after pages of tedious algebra that I honestly wouldn't want to go through again... ever) that for $\underline{\widehat{Y}} = \underline{WY}$,

$$\text{LOO-CV} = \frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \widehat{y}_i)^2}{(1-w_{ii})^2} = \frac{1}{n}\sum_{i=1}^{n}\frac{\widehat{e}_i^2}{(1-w_{ii})^2}.$$

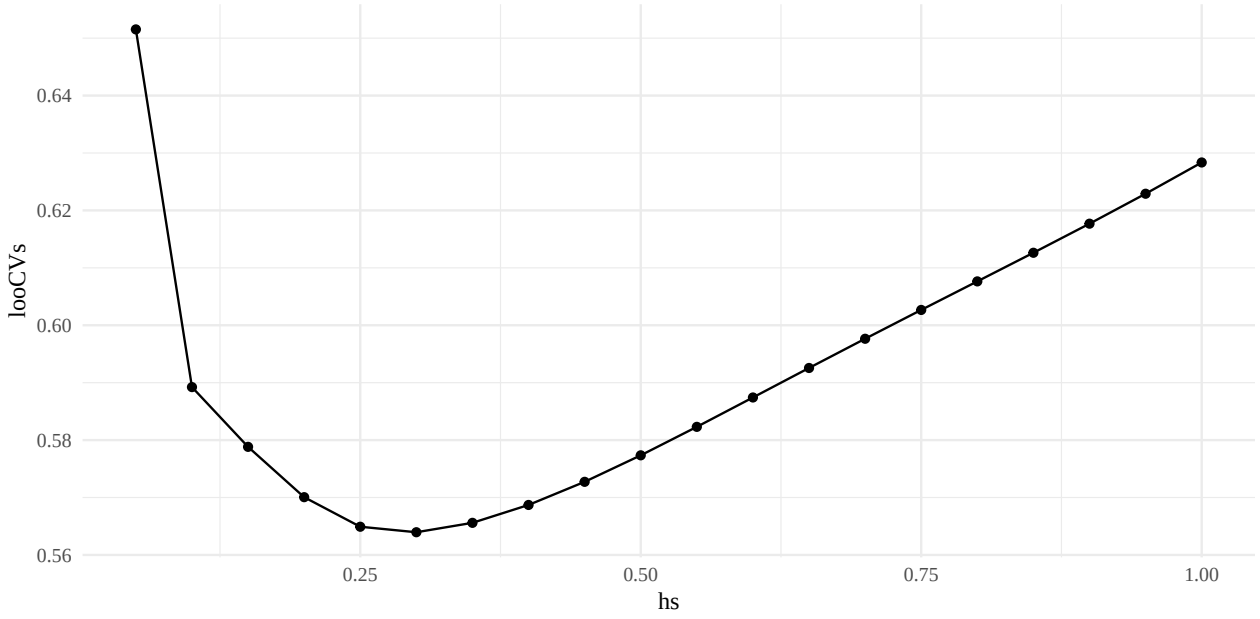
- This trick means that you only have to fit the model once rather than n times!
- You still have to calculate this for each model!

Back to the Gaussian Example

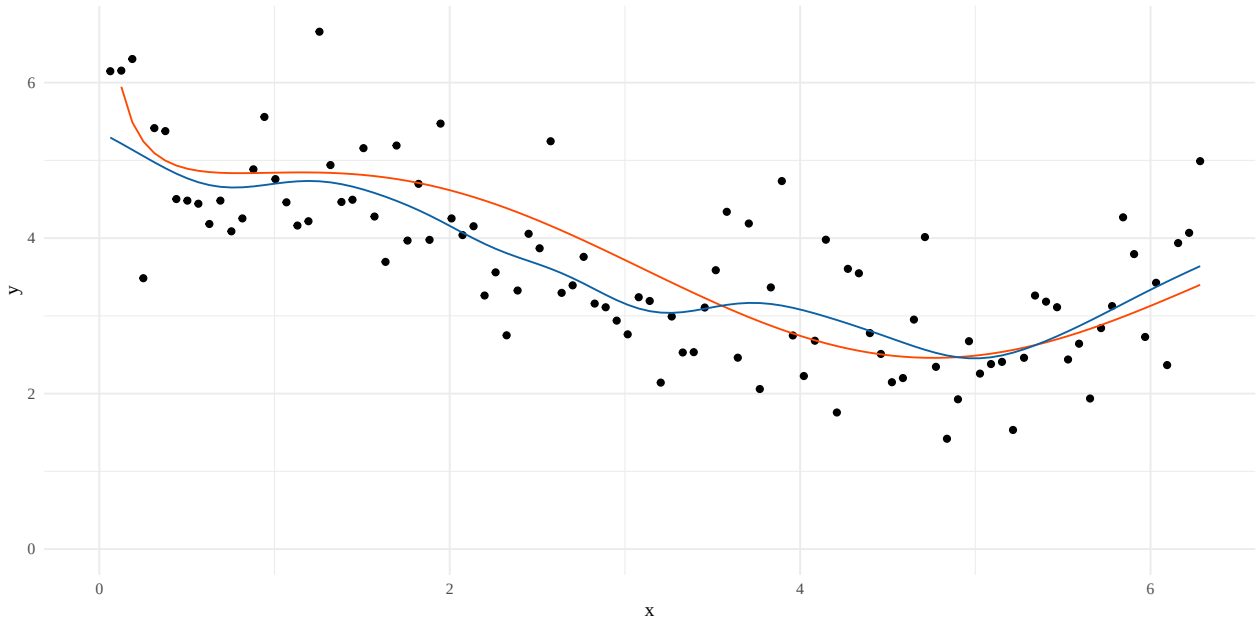
```
looCV <- function(y, W){
  n = length(y)
  resids2 = ((diag(n)-W) %*% y)^2
  denom = (1-diag(W))^2
  return(mean(resids2/denom))
}

looCV.forNiceModels <- function(mdl){
  mean(residuals(mdl)^2/(1-hatvalues(mdl))^2)
}

looCVs = double(20)
hs = seq(.05, 1, length.out=length(looCVs))
for(i in 1:length(looCVs)){
  W = Wgauss(hs[i])
  looCVs[i] = looCV(df$y, W)
}
ggplot(data.frame(hs,looCVs),aes(hs,looCVs)) + geom_point() + geom_line()
```



```
df$Wstar = with(df, Wgauss(hs[which.min(looCVs)]) %*% y)
ggplot(df, aes(x, y)) + geom_point() + xlim(0,2*pi) + ylim(0,max(df$y)) +
  stat_function(fun=trueFunction, color=red) +
  geom_line(aes(x, Wstar), color=blue)
```



Prediction MSE of Kernel Regression

Bias and Variance of Kernel Regression:

When we try to look at the MSE of a kernel estimator, $\widehat{\mu}(x)$, there are a lot off issues that prevent tractable equations.

We "bypass" these issues by using Taylor expansions of the Bias and Variance of the estimator. (A very common technique in advanced statistics.)

$$\text{Bias}(\widehat{\mu}(x)) \approx \frac{h^2}{2}\left(\mu''(x) + \frac{2\mu'(x)f'_X(x)}{f_X(x)}\right)\sigma_K^2 = O(h^2)$$

$$\text{Var}(\widehat{\mu}(x)) \approx \frac{1}{nh}\frac{\sigma^2(x)}{f_X(x)}R(K) = O(\frac{1}{nh})$$

Notes:

- f_X is the density function of the predictor variable X
- $\sigma^2(x)$ is $\text{Var}(Y|X = x)$
- σ_K^2 is the variance of the Kernel function
- $R(K) = \int K^2(u)\,du$; it's some weird tic of kernel estimation literature.
- The "big-Oh" notation means we have removed a bunch of constants that don't depend on n or h. We mainly concentrate on how quickly the O notation terms decrease. The rest is fluff.

MSE

- The highest level overview is equation (4.16):

$$MSE - \sigma^2(x) = O(h^4) + O(1/nh).$$

- Note: we have moved **irreducible noise** to the left of =.
- The first term on the right is the **squared bias** while the second term on the right is the **variance**.

Optimal Bandwidth

- The **Optimal Bandwidth** minimizes the MSE:

$$\begin{aligned} h_{opt} &= \arg\min_h C_1 h^4 + \frac{C_2}{nh} \\ \Rightarrow 0 &\stackrel{set}{=} 4C_1 h^3 - \frac{C_2}{nh^2} \\ \Rightarrow h^5 &= O\left(\frac{1}{n}\right) \\ \Rightarrow h_{opt} &= O\left(\frac{1}{n^{1/5}}\right). \end{aligned}$$

- If we plug this in, we get the **Oracle MSE**—the MSE for the optimal, though unavailable estimator.

$$\begin{aligned} MSE - \sigma^2 &= O(h_{opt}^4) + O(1/nh_{opt}) \\ &= O(1/n^{4/5}) + O(1/n^{4/5}) \\ &= O\left(\frac{1}{n^{4/5}}\right) \end{aligned}$$

By **Oracle** we mean, "if we knew the truth about $\mu(x)$ and f_X"

Kernels and interactions

- In multivariate kernel regressions, you estimate a **surface** over the input variables.
- This means we are trying to estimate $\widehat{\mu}(x_1, \ldots, x_p)$ in its entirety
- Therefore, this function **by construction** includes interactions, polynomials, etc.
- This is in contrast with **linear models** which need you to specify these things.
- This extra complexity (automatically including interactions, as well as other things) comes with tradeoffs.

Issue 1

- More complicated functions (smooth Kernel regressions vs. linear models) tend to have **lower bias** but **higher variance**.
- For $p = 1$, equations (4.19) and (4.20) show this:
- **Bias**
    1. The bias of using a linear model when it is wrong is a number $b(x, \theta_0)$ which doesn't depend on n.

2. The bias of using kernel regression is $O(1/n^{4/5})$. This goes to 0 as $n \to \infty$.

- **Variance**

1. The variance of using a linear model is $O(1/n)$
2. The variance of using kernel regression is $O(1/n^{4/5})$.

- To conclude: bias of kernels goes to zero (not for lines) but variance of lines goes to zero faster than for kernels.
- If the linear model is right, you win. But if it's wrong, you (eventually) lose.
- How do you know if you have enough data? Do model selection (CV to choose models).
- Compare of the kernel version with CV-selected tuning parameter (the CV estimate of the risk), with the CV estimate of the risk for the linear model.

Issue 2

- For $p > 1$, there is more trouble.
- First, lets look again at
$$MSE(h) - \sigma^2(x) = O(1/n^{4/5}).$$
That is for $p = 1$. It's not **that much** slower than $O(1/n)$, the variance for linear models.
- If $p > 1$ similar calculations show,
$$MSE(h) - \sigma^2(x) = O(1/n^{4/(4+p)}) \qquad MSE(\theta_0) - \sigma^2(x) = b(x, \theta_0) + O(p/n).$$
- What if p is big?

1. Then $O(1/n^{4/(4+p)})$ is still big.
2. But $O(p/n)$ is small.
3. So unless $b(x, \theta_0)$ is big, we should use the linear model.

- How do you tell? Use CV to decide.

npreg, The Kernel Regression Workhorse

We briefly went over using the `npreg` function in Lecture 03. Read section 4.6 carefully for more detailed help.

First, everything is in the `np` library, so you will want to install and load that.

```
library(np)
```

```
## Nonparametric Kernel Methods for Mixed Datatypes (version 0.60-9)
## [vignette("np_faq",package="np") provides answers to frequently asked questions]
## [vignette("np",package="np") an overview]
## [vignette("entropy_np",package="np") an overview of entropy-based methods]
```

The main function is the `npreg` function. It has syntax that is very similar to `lm`.

```
model <- npreg(y ~ x1 + x2 + x3)
```

- The `+` just means "use these variables"
- There's no reason to use `I(x1^2)` or `x1*x2`, it already does that. (Why?)
- `npreg` takes a little while to run, be sure to set `cache=TRUE` so you need only run it once.

- You can use `ordered(x2)` or `factor(x2)`. This may improve the speed a bit.
- DO NOT CROSS VALIDATE. `npreg` does it automatically. The CV risk estimate is in `out$bws$fval`.

Some more npreg discussion

- `npreg` is using CV and optimization to try to choose the bandwidth(s) for you.
- The `tol` and `ftol` arguments control how close the solution needs to be to an optimum.
- Very basic minimization (called Gradient descent):
  - Suppose I want to minimize $f(x) = (x-6)^2$ numerically.
  - If I start at a point (say $x_1 = 23$), vaguely, I want to "go" in the negative direction of the gradient.
  - The gradient (at $x_1 = 23$) is $f'(23) = 2(23-6) = 34$.
  - Gradient descent says, ok go that way by some small amount: $x_2 = x_1 - \gamma 34$, for γ. small.
  - In general, $x_{n+1} = x_n - \gamma f'(x_n)$.

```
niter = 10
gam = 0.1
x = double(niter)
x[1] = 23
grad <- function(x) 2*(x-6)
for(i in 2:niter) x[i] = x[i-1] - gam*grad(x[i-1])
x
```

```
##  [1] 23.000000 19.600000 16.880000 14.704000 12.963200 11.570560 10.456448
##  [8]  9.565158  8.852127  8.281701
```

- How do I decide if I'm done? The easiest way is to check how much I'm moving.

Fixing my gradient descent code

```
maxiter = 1000
conv = FALSE
gam = 0.1
x = 23
tol = 1e-3
grad <- function(x) 2*(x-6)
for(iter in  1:maxiter){
  x.new = x - gam * grad(x)
  conv = (x - x.new < tol)
  x = x.new
  if(conv) break
}
x
```

```
## [1] 6.003531
```

```
iter
```

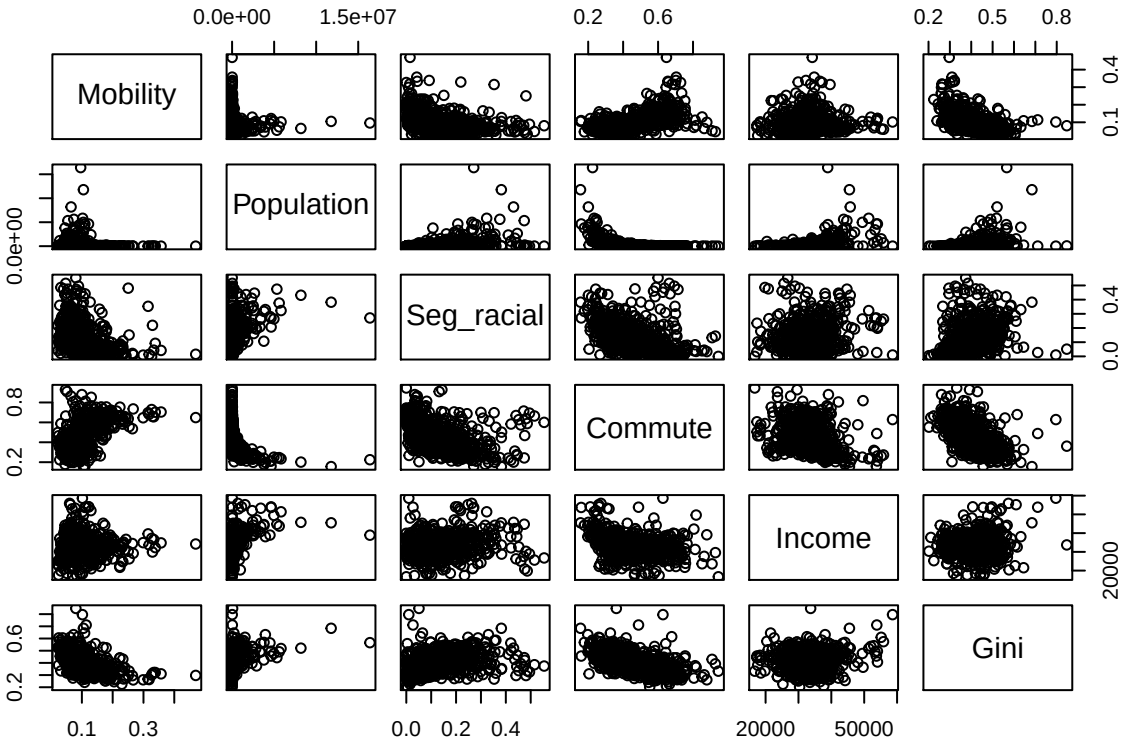
```
## [1] 38
```

- What happens if I change `tol` to `1e-7`?

npreg Example From Before

```
mob <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/15/hw/01/mobility.csv")
attach(mob)
mob <- data.frame(Mobility, Population, Seg_racial, Commute, Income, Gini)

pairs(mob)
```




```
mob.np <- npreg(Mobility ~ Population + Seg_racial + Commute + Income + Gini, data=mob)
```

```
##
Multistart 1 of 5 |
Multistart 1 of 5 |
Multistart 1 of 5 |
Multistart 1 of 5 /
Multistart 1 of 5 -
Multistart 1 of 5 \
Multistart 1 of 5 |
Multistart 1 of 5 /
Multistart 1 of 5 |
Multistart 1 of 5 |
Multistart 2 of 5 |
Multistart 2 of 5 |
Multistart 2 of 5 /
Multistart 2 of 5 -
Multistart 2 of 5 \
Multistart 2 of 5 |
Multistart 2 of 5 /
Multistart 2 of 5 |
Multistart 2 of 5 |
```

```
Multistart 3 of 5 |
Multistart 3 of 5 |
Multistart 3 of 5 /
Multistart 3 of 5 -
Multistart 3 of 5 \
Multistart 3 of 5 |
Multistart 3 of 5 /
Multistart 3 of 5 |
Multistart 3 of 5 |
Multistart 4 of 5 |
Multistart 4 of 5 |
Multistart 4 of 5 /
Multistart 4 of 5 -
Multistart 4 of 5 \
Multistart 4 of 5 |
Multistart 4 of 5 |
Multistart 5 of 5 |
Multistart 5 of 5 |
Multistart 5 of 5 /
Multistart 5 of 5 -
Multistart 5 of 5 \
Multistart 5 of 5 |
Multistart 5 of 5 /
Multistart 5 of 5 |
Multistart 5 of 5 |
```

- We can look at a summary:

```
summary(mob.np)
```

```
##
## Regression Data: 729 training points, in 5 variable(s)
##
## No. Complete Observations:  729
## No. Incomplete (NA) Observations:  12
## Observations omitted or excluded:  374 376 386 410 440 459 485 542 613 616 637 652
##               Population Seg_racial   Commute   Income       Gini
## Bandwidth(s):    1647235  0.1624958 0.0387193 2380.359 0.03178804
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.03022864
## R-squared: 0.6734398
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 5
```

- We can look at fitted values and residuals:

```
head(fitted(mob.np))
```

```
## [1] 0.06430381 0.06742757 0.07515042 0.05630083 0.06187224 0.06751265
```

```
tail(residuals(mob.np))
```

```
##           736           737           738           739           740
```

```
## -4.473357e-02 -3.446399e-02 -6.518525e-08  2.774182e-02 -7.624059e-03
##           741
##  1.798506e-02
```

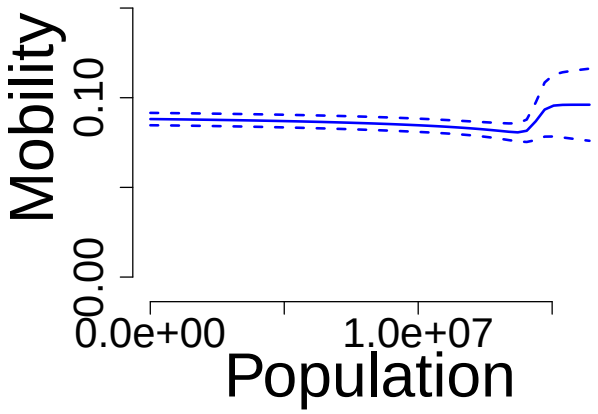
*We can make predictions:

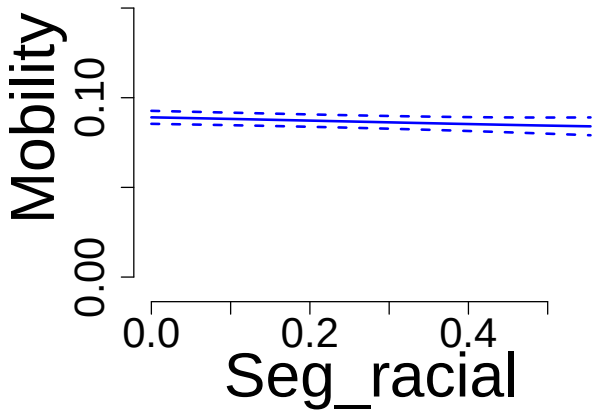
```
predict(mob.np, newdata=data.frame(Population=1.5e6, Seg_racial=0,
        Commute=0.5, Income=3e4, Gini=median(mob$Gini)))
```

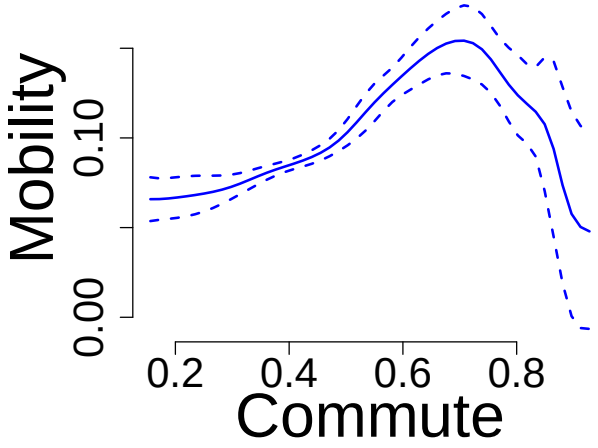
```
## [1] 0.0984894
```

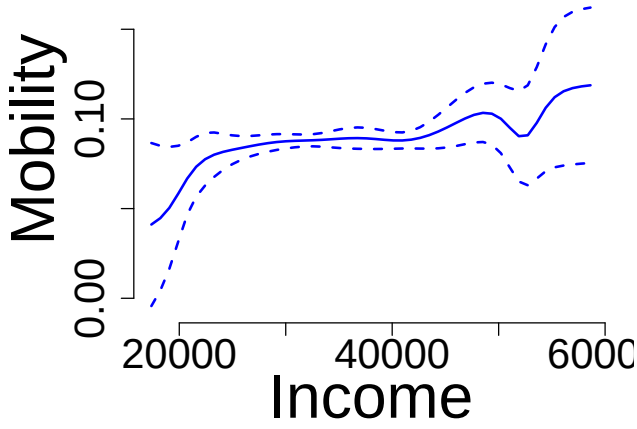
- and we can plot things

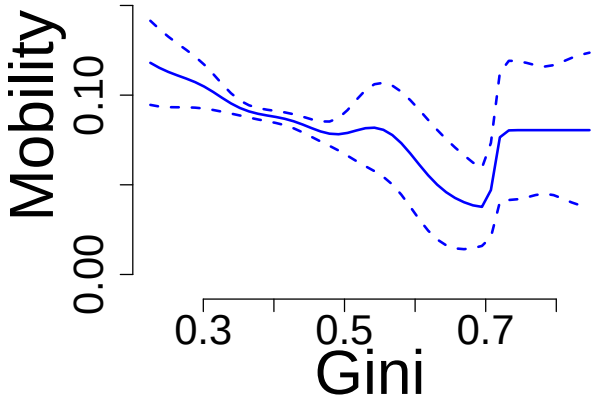
```
par(mar=c(5,5,1,1),cex.lab=3,cex.axis=2,lwd=2,col=4,bty='n')
plot(mob.np,plot.errors.method='bootstrap')
```


Mobility
0.00
0.10
0.0e+00
1.0e+07
Population


Mobility
0.00
0.10
0.0
0.2
0.4
Seg_racial


Mobility
0.00
0.10
0.2
0.4
0.6
0.8
Commute


Mobility
0.00
0.10
20000
40000
6000
Income


Mobility
0.00
0.10
0.3
0.5
0.7
Gini