SDS 383D Ex 03:
Linear Smoothing and Gaussian Processes

February 18, 2016

Jennifer Starling

Basic Concepts

Bias-Variance Decomposition

Let $\hat{f}(x)$ be a noisy estimate of some function $f(x)$, evaluated at some point x. Define the mean-squared error of the estimate as

$$MSE(\hat{f}, f) = E\{[f(x) - \hat{f}]^2\}$$

Prove that $MSE\left(f, \hat{f}\right) = B^2 + v$, where

$$B = E\{\hat{f}(x)\} - f(x) \text{ and } v = var\{f(x)\} = E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)^2\right]$$

Begin with the definition of MSE.

$$\begin{aligned}
MSE\left[f, \hat{f}\right] &= E\left[\left(f - \hat{f}\right)^2\right] \\
&= E\left[\left(\hat{f} - f\right)^2\right] \\
&= E\left[\left(\hat{f} - E\left(\hat{f}\right) + E\left(\hat{f}\right) - f\right)^2\right], \text{adding/subtracting } E\left(\hat{f}\right) \\
&= E\left[\left(\underbrace{\hat{f} - E\left(\hat{f}\right)}_{} + \underbrace{E\left(\hat{f}\right) - f}_{}\right)\left(\underbrace{\hat{f} - E\left(\hat{f}\right)}_{} + \underbrace{E\left(\hat{f}\right) - f}_{}\right)\right] \\
&= E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)^2\right] - E\left[\left(E\left(\hat{f}\right) - f\right)^2\right] + 2E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)\left(E\left(\hat{f}\right) - f\right)\right] \\
&= E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)^2\right] - \left(E\left(\hat{f}\right) - f\right)^2 + 2E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)\left(E\left(\hat{f}\right) - f\right)\right] \\
&\qquad \text{since } E\left(E\left(X\right)\right) = E\left(X\right) \\
&= Var\left(\hat{f}\right) - \left(Bias\left(\hat{f}, f\right)\right)^2 + 0
\end{aligned}$$

The last term reduces to zero as shown below.

$$\begin{aligned}
&2E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)\left(E\left(\hat{f}\right) - f\right)\right] \\
&= E\left[\hat{f}E\left(\hat{f}\right) - E\left(\hat{f}\right)E\left(\hat{f}\right) - \hat{f}f + E\left(\hat{f}\right)f\right]
\end{aligned}$$

Then $\hat{f} = E\left(\hat{f}\right)$, giving us

$$\begin{aligned}
&= E\left[E\left(\hat{f}\right)E\left(\hat{f}\right) - E\left(\hat{f}\right)E\left(\hat{f}\right) - E\left(\hat{f}\right)f + E\left(\hat{f}\right)f\right] \\
&= 0
\end{aligned}$$

Curve Fitting by Linear Smoothing

Consider a nonlinear regression problem with one predictor and one response: $y_i = f(x_i) + \epsilon_i$, where the ϵ_i are mean-zero random variables.

Part A

Suppose we want to estimate the value of the regression function y^* at some new point x^*, denoted $\hat{f}(x^*)$. Assume for the moment that $f(x)$ is linear, and that y and x have already had their means subtracted, in which case $y_i = \beta x_i + \epsilon_i$. Return to your least-squares estimator for multiple regression. Show that for the one-predictor case, your prediction $y^* = f(x^*) = \hat{\beta} x^*$ may be expressed as a linear smoother of the following form:

$$\hat{f}(x^*) = \sum_{i=1}^{n} w\left(x_i, x^*\right) y_i$$

for any x^*. Inspect the weighting function you derived. Briefly describe your understanding of how the resulting smoother behaves, compared with the smoother that arises from an alternate form of the weight function $w(x_i, w^*)$:

$$w_K(x_i, x^*) = \begin{cases} 1/K, \text{ } x_i \text{ one of the closest K sample points to } x_i \\ 0, \text{ otherwise} \end{cases}$$

This is referred to as K-nearest-neighbor smoothing.

The weighting function for smoothing is derived using the following.

$$\begin{aligned} \hat{f}(x^*) &= \hat{\beta} x^* \\ &= X^*(X'X)^{-1}X'y \\ &= x^* \cdot \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ &= \frac{\sum_{i=1}^{n} x_i x^* y_i}{\sum_{i=1}^{n} x_i^2} \end{aligned}$$

since we are working in the single-predictor, single-response, mean-zero case.

Therefore,

$$\hat{f}(x^*) = \sum_{i=1}^{n} w\left(x_i, x^*\right) y_i, \text{ with}$$
$$w\left(x_i, x^*\right) = \frac{\sum_{i=1}^{n} x_i x^*}{\sum_{i=1}^{n} x_i^2}$$

This is a linear smoother in the sense that all x^* points have their corresponding $y^* = \hat{f}(x^*)$ estimates 'smoothed' to the regression line represented by intercept 0, slope $\hat{\beta}$.

This is a different behavior than the *K-nearest-neighbor* smoothing. KNN smoothing is not fitting a line which is constructed by using all of the points. KNN smoothing is calculating each new y^* as the straight average of the closest K points y_i.

Part B

A kernel function $K(x)$ is a smooth function satisyfing

$$\int_{\mathbb{R}} K(x)dx = 1\,, \quad \int_{\mathbb{R}} xK(x)dx = 0\,, \quad \int_{\mathbb{R}} x^2K(x)dx > 0\,.$$

A very simple example is the uniform kernel,

$$K(x) = \frac{1}{2}I(x) \quad \text{where} \quad I(x) = \begin{cases} 1, & |x| \leq 1 \\ 0, & \text{otherwise}\,. \end{cases}$$

Another common example is the Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,.$$

Kernels are used as weighting functions for taking local averages. Specifically, define the weighting function

$$w(x_i, x^\star) = \frac{1}{h} K\left(\frac{x_i - x^\star}{h}\right)\,,$$

where h is the bandwidth. Using this weighting function in a linear smoother is called kernel regression. (The weighting function gives the unnormalized weights; you should normalize the weights so that they sum to 1.)
Write your own R function that will fit a kernel smoother for an arbitrary set of x-y pairs, and arbitrary choice of (positive real) bandwidth h. Set up an R script that will simulate noisy data from some nonlinear function, $y = f(x) + \epsilon$; subtract the sample means from the simulated x and y; and use your function to fit the kernel smoother for some choice of h. Plot the estimated functions for a range of bandwidths large enough to yield noticeable changes in the qualitative behavior of the prediction functions.

See **R Appendix, R Functions**. Function is called **linear_smoother**. There are also functions to specify which kernel function the linear smoother should use. These functions are called **K_gaussian** and **K_uniform**.

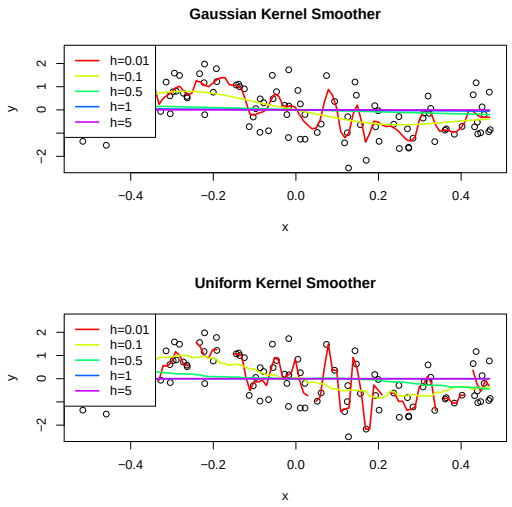

Figure 1: Kernel smoother for varying bandwidths h

Cross Validation

Left unanswered so far in our previous study of kernel regression is the question: how does one choose the bandwidth h used for the kernel? Assume for now that the goal is to predict well, not necessarily to recover the truth. (These are related but distinct goals.)

Part A

Presumably a good choice of h would be one that led to smaller predictive errors on fresh data. Write a function or script that will: (1) accept an old ("training") data set and a new ("testing") data set as inputs; (2) fit the kernel-regression estimator to the training data for specified choices of h; and (3) return the estimated functions and the realized prediction error on the testing data for each value of h. This should involve a fairly straightforward "wrapper" of the function you've already written.

See **R Appendix, R Functions**. Function is called **tune_h**.

Part B

Imagine a conceptual two-by-two table for the unknown, true state of affairs. The rows of the table are "wiggly function" and "smooth function," and the columns are "highly noisy observations" and "not so noisy observations." Simulate one data set (say, 500 points) for each of the four cells of this table, where the x's take values in the unit interval. Then split each data set into training and testing subsets. You choose the functions. Apply your method to each case, using the testing data to select a bandwidth parameter. Choose the estimate that minimizes the average squared error in prediction, which estimates the mean-squared error:

$$L_n(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n^\star}(y_i^\star - \hat{y}_i^\star)^2\,,$$

where $(y_i^\star, x_i^\star)$ are the points in the test set, and $\hat{y}_i^\star$ is your predicted value arising from the model you fit using only the training data. Does your out-of-sample predictive validation method lead to reasonable choices of h for each case?

My function found optimal bandwidths of h as below, using a 70/30 train-test split. My functions were $sin(2\pi x)$ for smooth, and $sin(2\pi x)$ for wiggly.

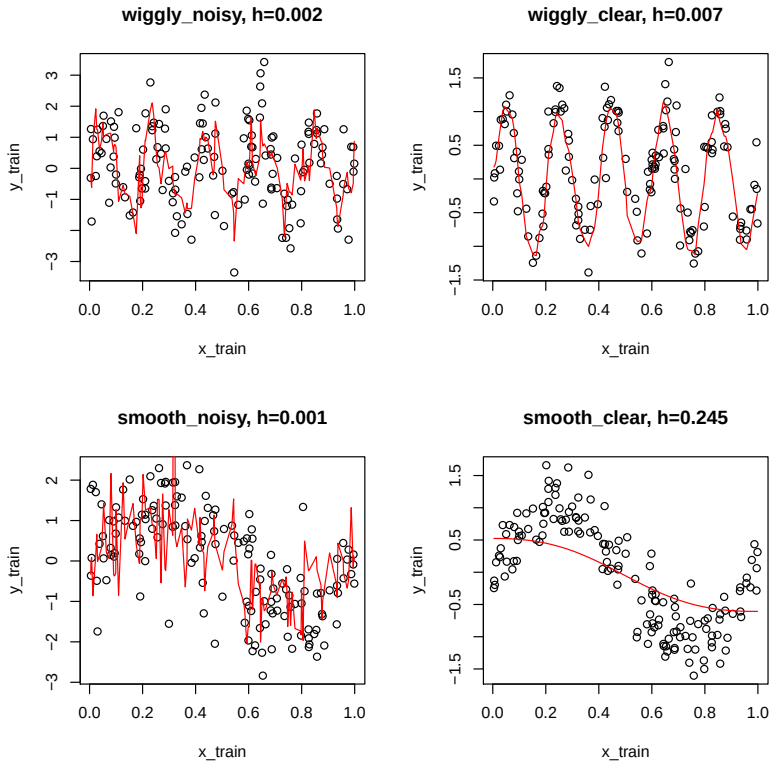

Figure 2: Bandwidth selection for various function types

These bandwidths look generally reasonable in terms of recovering the underlying functions, though the smooth/noisy function's bandwidth was smaller than I anticipated, and looks rather overfitted. My results also varied noticeably in quality as I reran the simulations for various random test/train splits, enough that I would recommend cross-validation each bandwidth selection via a few different test/train splits.

Part C

A few problems regarding splitting the data into test/train chunks as in previous problem:

1. Lose potentially valuable information about outliers or patterns in the data by only using a portion of your data to train the model.
2. As mentioned previously, optimal bandwidth selection and the resulting model fit depended

Appendix: R Code

R Functions

```
#SDS 383D – Exercise 3
#Functions
#Jennifer Starling
#Feb 15, 2017

#_____________________________________________________________________
### Kernel–Related Functions (Data simulation, linear smoother)

sim_noisy_data = function(x,f,sig2){
    #FUNCTION:  Simulates noisy data from some nonlinear function f(x).
    #INPUTS:    x = independent observations.
    #           f = function f(x) to simulate from
    #           sig2 = variance of the e ~ N(0,sig2) noise.
    #OUTPUTS:   x = generated x values.
    #           y = generated y = f(x) + e values.

    fx = f(x)
    e = rnorm(length(x),0,sqrt(sig2))
    return(y = fx+e)
}

linear_smoother = function(x,y,x_star,h=1,K){
    #FUNCTION:  Linear smoothing function for kernel regression.
    #INPUTS:    x = a scalar or vector of regression covariates.
    #           x_star = scalar or vector of new x values for prediction.
    #           h = a positive bandwidth.
    #           K = a kernel function.  Default is set to Gaussian kernel.
    #OUTPUT:    yhat = a scalar or vector of smoothed x values.

    yhat=0  #Initialize yhat.

    for (i in 1:length(x_star)){
        w = (1/h) * K((x-x_star[i])/h) #Calculates weights.
        w = w / sum(w)                  #Normalize weights.
        yhat[i] = crossprod(w,y)
    }
    return(yhat)
}

#_____________________________________________________________________
### Kernel Functions

K_uniform = function(x){
    #FUNCTION:  Uniform kernel.
    #INPUTS:    x = a scalar or vector of values.
    #OUTPUT:    k = a scalar or vector of smoothed x values.

    k = .5 * ifelse(abs(x)<=1,rep(1,length(x)),rep(0,length(x)))
    return(k)
}

K_gaussian = function(x){
    #FUNCTION: Gaussian kernel.
    #INPUTS:    x = a scalar or vector of values.
```

```
    #OUTPUT:    k = a scalar or vector of smoothed x values.

    k = (1/sqrt(2*pi)) * exp(-x^2/2)
    return(k)
}

#-------------------------------------------------------------------
### Cross-Validation Functions (tuning bandwidth)

tune_h = function(test,train,K,h){
    #FUNCTION:  Function to tune bandwidth h for
    #               specified test/train data sets and
    #               specified kernel K.
    #INPUTS:    test = a test data set. Must have two cols, x and y.
    #           train = a training data set. Must have two cols, x and y.
    #           K = the kernel function
    #           h = a scalar or vector of bandwidths to try.
    #OUTPUTS:   pred_err_test = prediction error for testing data for each h.
    #           fhat_test = predicted values for testing data's x vals.

    #Extract training and test x and y vectors.
    x = train[,1]
    y = train[,2]
    x_star = test[,1]
    y_star = test[,2]

    #Calculate predicted points for test data set, and prediction error.
    fhat_star = linear_smoother(x,y,x_star,h,K)

    #Calculate predicted points for test data set, and prediction error.
    pred_err_test = sum(fhat_star-y_star)^2 / (length(x))

    #Return function outputs:
    return(list(fhat_star=fhat_star, pred_err_test = pred_err_test))
}

tune_h_loocv = function(x,y,K,h){
    #FUNCTION:  Function to use leave on out cross-validation to tune
    #               bandwidth h for specified test/train data sets and
    #               specified kernel K.
    #INPUTS:    x = a scalar or vector; the dependent variable.
    #           y = a scalar or vector; the independent var.  Same length as x.
    #           K = the kernel function
    #           h = a scalar or vector of bandwidths to try.
    #OUTPUTS:   pred_err_test = prediction error for testing data for each h.
    #           fhat_test = predicted values for testing data's x vals.

    #Define ppm H, the 'smoothing matrix'.
    # {H_ij} = 1/h * K((xi-xj*)/h)
    #For loocv, x=x*, since calculating on single data set instead of test/train.

    Hat = matrix(0,nrow=length(x),ncol=length(x)) #Empty matrix.

    for (i in 1:length(x)){          #Loop through H rows.
        for (j in 1:length(x)){      #Loop through H cols.
            Hat[i,j] = (1/h) * K((x[j] - x[i])/h)
        }
        #Normalize weights by dividing H by rowsums(H).
        Hat[i,] = Hat[i,] / sum(Hat[i,])
```

```
    }

    #Calculate predicted values.
    yhat = Hat %*% y

    #Calculate loocv prediction error.
    loocv_err = sum(((y-yhat)/(1-diag(Hat)))^2)

    #Return function outputs:
    return(list(yhat=yhat,loocv_err=loocv_err))
}
```

Part B Script - Curve Fitting by Linear Smoothing

```
#Stats Modeling 2
#Exercise 3
#Curve Fitting by Linear Smoothing – Part B
#Jennifer Starling
#Feb 15, 2017

#-------------------------------------------------------------
###     Simulate noisy data from a non-linear function,
###     subtract the sample means from the simulated x and y,
###     and use the smoother function to fit the kernel smoother
###     for some choice of h.  Plot estimated functions for a
###     range of bandwidths large enough to yield noticeable changes
###     in the qualitative behavior of the prediction functions.
#-------------------------------------------------------------

### Environment setup.

#Housekeeping.
rm(list=ls())

#Load functions.
source("/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/R Files/
    SDS383D_Ex3_FUNCTIONS.R")

#-------------------------------------------------------------
### Data generation.

# Simulate x and y from a non-linear function x.
x = runif(100,0,1)  #Generate a sample of x values.
f = function(x) sin(2*pi*x) #Set the non-linear function.

#Generate noisy data, and extract x and y.
y = sim_noisy_data(x,f,sig2=.75)

#Subtract means from simulated x and y.
x = x - mean(x)
y = y - mean(y)

#Plot noisy data, with means subtracted.
pdf(file='/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    noisydata.pdf')
plot(x,y,main='Noisy data',xlab='x',ylab='y')
dev.off()

#Set up a vector of x_star values, and estimate function value at these points.
x_star = seq(min(x),max(x),by=.01)
fhat_star = linear_smoother(x,y,x_star,h=.01,K=K_gaussian)

plot(x,y,main='Gaussian Kernel Smoother')
lines(x_star,fhat_star,col='red')

#-------------------------------------------------------------
### Plot linear smoothing output for various bandwidths.

#Set up bandwidths to try, and corresponding colors.
H = c(.01,.1,.5,1,5)
col = rainbow(length(H))
```

```
#Open pdf file for two plots.
pdf(file='/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    Smoothers.pdf')
par(mfrow=c(2,1))

#Plot for a variety of bandwidths using Gaussian kernel.
plot(x,y,main='Gaussian Kernel Smoother')
for (i in 1:length(H)){
    fhat_star = linear_smoother(x,y,x_star,h=H[i],K=K_gaussian)
    lines(x_star,fhat_star,col=col[i],lwd=2)
}
legend('topleft',legend=paste("h=",H,sep=''),lwd=2,lty=1,col=col,bg='white')

#Plot for a variety of bandwidths using uniform kernel.
plot(x,y,main='Uniform Kernel Smoother')
for (i in 1:length(H)){
    fhat_star = linear_smoother(x,y,x_star,h=H[i],K=K_uniform)
    lines(x_star,fhat_star,col=col[i],lwd=2)
}
legend('topleft',legend=paste("h=",H,sep=''),lwd=2,lty=1,col=col,bg='white')

dev.off() #Close pdf file for two plots.
```

Part C Script - Cross Validation

```
#Stats Modeling 2
#Exercise 3
#Cross-Validation - Part B
#Jennifer Starling
#Feb 15, 2017

#------------------------------------------------------------
### CV PART B Script:  Imagine a 2x2 tablefor the unknown, true
### state of affairs.  Rows are 'wiggly' and 'smooth' functions,
### and cols are 'highly noisy obs' and 'less noisy obs'.
### Simulate one data set (n=500) for each of four cells of table.
### Split each data set into test and train sets.  Apply method to
### each case.  Apply function from part A to select bandwidth.
#------------------------------------------------------------

### Environment setup.
#Housekeeping.
rm(list=ls())

#Load latex table library.
library(xtable) #For table output to latex.
options(xtable.floating = FALSE)
options(xtable.timestamp = "")

#Load functions.
source('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/RCode/SDS383D_
    Ex3_FUNCTIONS.R')

#------------------------------------------------------------
### Data and function generation.

#Set up wiggly and less-wiggly functions.
fwiggly = function(x) sin(10*pi*x)
fsmooth = function(x) sin(2*pi*x)

#Set up x values on the unit interval.
n = 500
x = runif(n,0,1)

#Generate noisy and less-noisy data for
#each fwiggly and fsmooth function.
y_wiggly_noisy  = sim_noisy_data(x,fwiggly,sig2=.75)
y_wiggly_clear  = sim_noisy_data(x,fwiggly,sig2=.1)
y_smooth_noisy  = sim_noisy_data(x,fsmooth,sig2=.75)
y_smooth_clear  = sim_noisy_data(x,fsmooth,sig2=.1)

#Combine into single matrix for convenience.
y = matrix(c(y_wiggly_noisy,y_wiggly_clear,y_smooth_noisy,y_smooth_clear),ncol=4,byrow
    =F)
colnames(y) = c('wiggly_noisy','wiggly_clear','smooth_noisy','smooth_clear')

#Scatter plots of the four data sets.
pdf(file='/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/cv_
    function_grid.pdf')
    par(mfrow=c(2,2))
    for(j in 1:ncol(y)){
        plot(x,y[,j],main=paste(colnames(y)[j]),xlab='x',ylab='y',ylim=c(-5,5))
```

```
    }
dev.off()

#------------------------------------------------------------
### Split each data set into train and test sets. (70-30)

test_idx = sample(1:n,size=n*.3,replace=F)
test = cbind(x[test_idx],y[test_idx,])
train = cbind(x[-test_idx],y[-test_idx,])

#------------------------------------------------------------
### Optimize bandwidth h.

# For each case, select a bandwidth parameter. (Used Gaussian kernel.)
h_opt = rep(0,4)    #Vector to hold optimal h values.
names(h_opt) = colnames(test)[-1]
H = seq(.001,1,by=.001) #Candidate h values.
fx_hat = list()         #To hold estimated function values.
x_star = test[,1]

#Iterate through each setup.
for (i in 1:4){
    #Extract x and just the y column required.
    tr = train[,c(1,i+1)]
    te = test[,c(1,i+1)]

    #Temp vector to hold prediction errors.
    temp_pred_err = rep(0,length(H))

    for (j in 1:length(H)){
        h = H[j]
        results = tune_h(test=te,train=tr,K=K_gaussian,h=h)
        temp_pred_err[j] = results$pred_err_test
    }

    h_opt[i] = H[which.min(temp_pred_err)]
    fx_hat[[i]] = tune_h(te,tr,K=K_gaussian,h_opt[i])$fhat_star
}

#------------------------------------------------------------
### Output results as table and plot.

#Format optimal h results as matrix.
h_opt_mat = matrix(h_opt,nrow=2,byrow=T)
colnames(h_opt_mat) = c('noisy','clear')
rownames(h_opt_mat) = c('wiggly','smooth')
xtable(h_opt_mat,digits=3)  #Output latex table.

#Plot output with fitted data using optimal h values.
#pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    Bandwidth_selection.pdf')
par(mfrow=c(2,2))
for (i in 1:4){
    #Scatterplot of test x/y.
    plot(test[,1],test[,i+1],
        main=paste(names(h_opt)[i],", h=",h_opt[i],sep=''),
        xlab='x_train',ylab='y_train')
    #Overlay estimated fit.
    idx = sort(x_star, index.return = T)$ix
```

```
    lines(sort(x_star),fx_hat[[i]][idx],col='red') #Fitted line.
}
#dev.off()

#-----------------------------------------------------------------
### CV PART C Script:  Find optimal bandwidths h for each
### combination of wiggly/smooth and noisy/clear functions using
### LOOCV.
#-----------------------------------------------------------------

# For each case, select a bandwidth parameter. (Used Gaussian kernel.)
h_opt_loocv = rep(0,4)  #Vector to hold optimal h values.
names(h_opt_loocv) = colnames(y)
#H = seq(.001,1,by=.001)    #Candidate h values.
H = seq(.001,.01,.1)
fx_hat_loocv = list()           #To hold estimated function values.

#Iterate through each setup.
for (i in 1:4){

    #Extract y column.
    ytemp = y[,i]

    #Temp vector to hold prediction errors.
    temp_pred_err = rep(0,length(H))

    for (j in 1:length(H)){
        h = H[j]
        results = tune_h_loocv(x=x,y=ytemp,K=K_gaussian,h=h)
        temp_pred_err[j] = results$loocv_err
    }

    h_opt_loocv[i] = H[which.min(temp_pred_err)]
    fx_hat_loocv[[i]] = tune_h_loocv(x,ytemp,K=K_gaussian,h_opt_loocv[i])$yhat
}

#Sanity check
cbind(y[,4],fx_hat_loocv[[4]])

#-----------------------------------------------------------------
### Output results as table and plot.

#Format optimal h results as matrix.
h_opt_mat_loocv = matrix(h_opt_loocv,nrow=2,byrow=T)
colnames(h_opt_mat_loocv) = c('noisy','clear')
rownames(h_opt_mat_loocv) = c('wiggly','smooth')
xtable(h_opt_mat_loocv,digits=3)    #Output latex table.

#Plot output with fitted data using optimal h values.
#pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    Bandwidth_selection_loocv.pdf')
par(mfrow=c(2,2))
for (i in 1:4){
    #Scatterplot of test x/y.
    plot(x,y[,i],
        main=paste(names(h_opt_loocv)[i],", h=",h_opt_loocv[i],sep=''),
        xlab='x',ylab='y')
    #Overlay estimated fit.
    idx = sort(x, index.return = T)$ix
```

```
    lines(sort(x),fx_hat_loocv[[i]][idx],col='red') #Fitted line.
}
#dev.off()

par(mfrow=c(1,2))
plot(x,y[,1])
plot(x,fx_hat[[1]])
```