SDS 383D: Exercises 3 – Linear smoothing and Gaussian processes

February 17, 2017

Professor Scott

Spencer Woody

Problem 1

Basic Concepts

(A)

Problem 2

Curve fitting by linear smoothing

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

(A) For now, consider a linear regression on a response y_i with one predictor x_i, and both y_i and x_i have had their means subtracted, so the $y_i = \beta x_i + \epsilon_i$. Define $S_x := \sum_{i=1}^n x_i^2$. The least squares estimate for the coefficient, from Exercises 1, is

$$\begin{aligned}\hat{\beta} &= (X^T X)^{-1} X^T y \\ &= (x^T x)^{-1} x^T y \\ &= \frac{\sum_{i=1}^n x_i \cdot y_i}{\sum_{i=1}^n x_i^2} \\ &= \frac{\sum_{i=1}^n x_i \cdot y_i}{S_x} \\ &= \sum_{i=1}^n \frac{x_i}{S_x} \cdot y_i.\end{aligned}$$

So now our prediction $y^\star | x^\star$ is,

$$\begin{aligned}\hat{y}^\star &= \hat{f}(x^\star) \\ &= \hat{\beta} x^\star \\ &= \left(\sum_{i=1}^n \frac{x_i}{S_x} \cdot y_i \right) \cdot x^\star \\ &= \sum_{i=1}^n \left(\frac{x_i}{S_x} \cdot x^\star \right) \cdot y_i,\end{aligned}$$

which we recognize as being in the form of the general *linear smoother*

$$\hat{f}(x^\star) = \sum_{i=1}^n w(x_i, x^\star) \cdot y_i$$

for some weight function $w(x_i, x^\star)$. In particular, the weight function for linear regression gives weight to each y_i proportional to the value of x_i. Contrast this with the k-nearest neighbors smoothing weight function,

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

which gives *equal* weight to y_is but *only* to the k-nearest neighbors of $x^\star$.

(B) Now we have the very general weight function

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

where $K(\bullet)$ is some kernel

Problem 3

Cross validation

(A)

Problem 4

Local polynomial regression

(A)

Problem 5

Gaussian processes

(A)

Problem 6

In nonparametric regression and spacial smoothing

(A)

R code for `myfuns03.R`

```
#############################################################
######### Created by Spencer Woody on 11 Feb 2017 #########
#############################################################

# ==========================================================================
# Linear smoothing ==========================================================
# ==========================================================================

lin.smooth <- function(x.new, x, y, kern.fun, h) {
    # -------------------------------------------------------------------------
    # Linear smoother for some kernel function
    # -------------------------------------------------------------------------
    # INPUTS:
    # x.new — a new point for which to estimate f(x.new)
    # x — a vector of covariates from previous observations
    # y — a vector of responses from previous observations
    # kern.fun — some kernel function (e.g. Gaussian)
    #             *** takes 2 arguments: distance (dist) and bandwidth (h)
    # h is the bandwidth for the kernel function
    # -------------------------------------------------------------------------
    # OUTPUT:
    # weights — a vector of weights for a new observation
    # -------------------------------------------------------------------------

    weights <- kern.fun(dist = x - x.new, h = h) / h
    weights <- weights / sum(weights)

    fit <- crossprod(weights, y)

    return(fit)
}

kern.unif <- function(dist, h) {
    # -------------------------------------------------------------------------
    # Uniform kernel function
    # -------------------------------------------------------------------------
    # INPUTS:
    # dist
    # h is
    # Sigma is the covariance matrix
    # -------------------------------------------------------------------------
    # OUTPUT:
    # kern — the value of the uniform kernel function
    # -------------------------------------------------------------------------

    kern <- ( (dist / h) <= 1) / 2

    return(kern)
}

kern.norm <- function(dist, h) {
    # -------------------------------------------------------------------------
```

```
    # Gaussian (normal) kernel function
    # ---------------------------------------------------------------------------
    # INPUTS:
    # dist the distance
    # h is
    # Sigma is the covariance matrix
    # ---------------------------------------------------------------------------
    # OUTPUT:
    # kern is the value of the Gaussian kernel function
    # ---------------------------------------------------------------------------

    kern <- 1 / sqrt(2 * pi) * exp(-dist^2 / 2)

    return(kern)
}

sprintf("Be sure to ")

make.noise <- function(x, f, res.fun) {
    # ---------------------------------------------------------------------------
    # Simulate noisy data from some non-linear function
    # ---------------------------------------------------------------------------
    # INPUTS:
    # x - the number of points from noisy distribution
    # f - a function for the expected value, E(y) = f(x)
    # res.fun - a mean-zero function for the distribution of residuals
    #           (e.g. rnorm(), etc.)
    # ---------------------------------------------------------------------------
    # OUTPUT:
    # noise - the simulated data
    # ---------------------------------------------------------------------------

    noise <- f(x) + res.fun(n = length(x))

    return(noise)
}

# ==========================================================================
# Gaussian process =========================================================
# ==========================================================================

my.mvn <- function(n, mu, Sigma) {
    #  Simulate n draws from MVN(mu, Sigma)
    #
    #  Note: this function assumes that X already has an intercept term
    # (or doesn't, if we want to force OLS through the origin)
    #
    # INPUTS:
    # n is the number of draws
    # mu is the mean vector
    # Sigma is the covariance matrix
    #
    # OUTPUT:
```

```
    # x is matrix of n draws from MVN(mu, Sigma) [with n rows, p columns]
    #

    # dimension of MVN
    p <- length(mu)

    # Check if inputs are valid (dimensions match, Sigma is square and p.s.d.)
    cond<- (ncol(Sigma) != p) |
           (nrow(Sigma) != p) |
           (max(eigen(Sigma)$values) <= 0)

    if (cond) {
        return("Try again...")
    }

    # Generate n*p univariate standard normal variables
    z      <- matrix(rnorm(n*p), nrow = p)

    # Create a matrix containing copies of mu
    mumat <- matrix(rep(mu, n), nrow = p)

    # Decompose Sigma into Sigma = L %*% Lt
    Lt <- chol(Sigma)

    # Generate sample with affine transformation of z
    x <- crossprod(Lt, z) + mumat

    return(t(x))
}

ell2 <- function(x) {
    # Compute the ell2 norm of x, a vector in Euclidean space

    return(sqrt(sum(x^2)))
}

C.SE <- function(x.i, x.j, params = NA) {
    # ---------------------------------------------------------------------------
    #  Compute the (i, j) element of a squared exp. covariance matrix
    #
    # ---------------------------------------------------------------------------
    # INPUTS:
    # x.i and x.j are two vectors in same space (need not be [0, 1])
    # params should be a vector of three hyperparameters
    #      1) b
    #      2) tau1.sq
    #      3) tau2.sq
    #
    # ---------------------------------------------------------------------------
    # OUTPUT:
    # c.se is the value of the Matern-5/2 covariance matrix for x.i and x.j
    # ---------------------------------------------------------------------------
```

```
    if (prod(is.na(params))) {
        return("Must have three valid parameters.")
    }

    if (length(params) != 3) {
        return("Must have three valid parameters.")
    }

    b       <- params[1]
    tau1.sq <- params[2]
    tau2.sq <- params[3]


    b       <- params[1]
    tau1.sq <- params[2]
    tau2.sq <- params[3]

    # Euclidean distance between x.i and x.j
    d <- ell2(x.i - x.j)

    c.se <- tau1.sq * exp(-0.5 * (d / b)^2) + tau2.sq * (x.i == x.j)

    return(c.se)
}

C.M52 <- function(x.i, x.j, params = NA) {
    # ------------------------------------------------------------------------
    #  Compute the (i, j) element of a Matern-5/2 covariance matrix
    #
    # ------------------------------------------------------------------------
    # INPUTS:
    # x.i and x.j are two vectors in same space (need not be [0, 1])
    # params should be a vector of three hyperparameters
    #      1) b
    #      2) tau1.sq
    #      3) tau2.sq
    #
    # ------------------------------------------------------------------------
    # OUTPUT:
    # c.m52 is the value of the Matern-5/2 covariance matrix for x.i and x.j
    # ------------------------------------------------------------------------

    if (prod(is.na(params))) {
        return("Must have three valid parameters.")
    }

    if (length(params) != 3) {
        return("Must have three valid parameters.")
    }

    b       <- params[1]
    tau1.sq <- params[2]
    tau2.sq <- params[3]
```

```
    # Euclidean distance between x.i and x.j
    d <- ell2(x.i - x.j)

    c.m52 <- tau1.sq * ( 1 + (5^0.5 * d / b) + (5 / 3 * (d / b)^2) ) *
             exp(-5^0.5 * d / b)  + tau2.sq * (x.i == x.j)

    return(c.m52)
}

make.covmat <- function(x, cov.fun, params = NA) {
    # -------------------------------------------------------------------------
    #  Compute the covariance matrix for a GP, given some cov. function
    #
    # -------------------------------------------------------------------------
    # INPUTS:
    # x is a vector of N values in [0, 1]
    # params should be a vector of three hyperparameters
    #      1) b
    #      2) tau1.sq
    #      3) tau2.sq
    #
    # -------------------------------------------------------------------------
    # OUTPUT:
    # covmat is the covariance matrix of GP
    # -------------------------------------------------------------------------

    if (prod(is.na(params))) {
        return("Must have three valid parameters.")
    }

    if (length(params) != 3) {
        return("Must have three valid parameters.")
    }

    N <- length(x)

    covmat <- matrix(nrow = N, ncol = N)

    for (j in 1:N) {
        for (i in j:N) {
            covmat[i, j] <- cov.fun(x[i], x[j], params = params)
            covmat[j, i] <- covmat[i, j]
        }
    }

    return(covmat)
}
```

R code for `exercises03.R`

```r
############################################################
######### Created by Spencer Woody on 11 Feb 2017 #########
############################################################

# ==========================================================================
# Gaussian process ==========================================================
# ==========================================================================

library(ggplot2)

source("myfuns03.R")

x.seq <- seq(0, 1, length.out = 100)

b <- 1
tau1.sq <- 1e-6
tau2.sq <- 1e-5

myparams <- c(b, tau1.sq, tau2.sq)

xCM52 <- make.covmat(x.seq, C.M52, params = myparams)
xSE <- make.covmat(x.seq, C.SE, params = myparams)
```