ISyE 7406: Data Mining & Statistical Learning
HW#4

Local Smoothing. The goal of this homework is to help you better understand the statistical properties and computational challenges of local smoothing such as loess, Nadaraya-Watson (NW) kernel smoothing, and spline smoothing.

For this purpose, we will compute empirical bias, empirical variances, and empirical mean square error (MSE) based on $m = 1000$ Monte Carlo runs, where in each run we simulate a data set of $n = 101$ observations from the additive noise model

$$Y_i = f(x_i) + \epsilon_i \tag{1}$$

with the famous Mexican hat function

$$f(x) = (1 - x^2)\exp(-0.5x^2), \qquad -2\pi \le x \le 2\pi, \tag{2}$$

and $\epsilon_1, \cdots, \epsilon_n$ are independent and identically distributed (iid) $N(0, 0.2^2)$. This function is known to pose a variety of estimation challenges, and below we explore the difficulties inherent in this function.

(1) Let us first consider the (deterministic fixed) design with equi-distant points in $[-2\pi, 2\pi]$.

(a) For each of $m = 1000$ Monte Carlo runs, simulate or generate a data set of the form $(x_i, Y_i))$ with $x_i = 2\pi(-1 + 2\frac{i-1}{n-1})$ and Y_i is from the model in (1). Denote such data set as $\mathcal{D}_j$ at the j-th Monte Carlo run for $j = 1, \cdots, m = 1000$.

(b) For each data set $\mathcal{D}_j$ or each Monte Carlo run, compute the three different kinds of local smoothing estimates at every point in $\mathcal{D}_j$: loess (with span = 0.75), Nadaraya-Watson (NW) kernel smoothing with Gaussian Kernel and bandwidth = 0.2, and spline smoothing with the default tuning parameter.

(c) At each point x_i, for each local smoothing method, based on $m = 1000$ Monte Carlo runs, compute the empirical bias, empirical variance, and empirical mean square error (MSE), which are defined as

$$\begin{aligned}
\widehat{Bias\{f(x_i)\}} &= \bar{\mathbf{f}}_m(x_i) - f(x_i), \quad \text{with } \bar{\mathbf{f}}_m(x_i) = \frac{1}{m}\sum_{j=1}^{m} \hat{f}^{(j)}(x_i) \\
\widehat{Var\{f(x_i)\}} &= \frac{1}{m}\sum_{j=1}^{m} \left(\hat{f}^{(j)}(x_i) - \bar{\mathbf{f}}_m(x_i)\right)^2, \\
\widehat{MSE\{f(x_i)\}} &= \frac{1}{m}\sum_{j=1}^{m} \left(\hat{f}^{(j)}(x_i) - f(x_i)\right)^2,
\end{aligned}$$

Here we use the true function value $f(x_i)$ in (2) in the definition of empirical Bias and empirical MSE, which better helps us understanding the performance of different local smoothing methods when estimating the true function f. Moreover, we purposely use the coefficient $\frac{1}{m}$ (instead of the standard coefficient $\frac{1}{m-1}$) in the definition of empirical variance, so that the well-known relation $\text{MSE} = \text{Bias}^2 + \text{Var}$ is applicable to the empirical version.

(d) Plot these quantities against x_i for all three kinds of local smoothing estimators: loess, NW kernel, and spline smoothing.

(e) Provide a through analysis of what the plots suggest, e.g., which method is better/worse on bias, variance, and MSE? Do you think whether it is fair comparison between these three methods? Why or why not?

(2) Repeat part (1) with another (deterministic) design that has non-equidistant points in the interval $[-2\pi, 2\pi]$. The following R code is used to generate the design points x_i's in my laptop, denoted by x_2 below (you can keep these x_i's fixed in the $m = 1000$ Monte Carlo runs):

```
set.seed(79)
x2 <- round(2*pi*sort(c(0.5, -1 + rbeta(50,2,2), rbeta(50,2,2))), 8)
```

For those students who use Python or other softwares, below are the values of $x2$, which is also available in the dataset "HW04part2.x.csv":

```
  [1] -6.15270162 -5.96542722 -5.34857373 -5.25045325 -4.93408232
  [6] -4.82399219 -4.79756399 -4.72091648 -4.56066454 -4.52796823
 [11] -4.37012804 -4.35883971 -4.35650125 -4.14444058 -4.08772246
 [16] -3.99254444 -3.85521732 -3.79979095 -3.68856521 -3.66394780
 [21] -3.61158538 -3.49345211 -3.43019447 -3.19560754 -3.18518010
 [26] -3.12475790 -2.97880845 -2.89425664 -2.88764749 -2.70117864
 [31] -2.65330733 -2.63339792 -2.43171710 -2.23984616 -2.20285070
 [36] -1.95028199 -1.89814133 -1.70630816 -1.66101539 -1.41528948
 [41] -1.35654375 -1.32557550 -1.31997953 -1.28553107 -1.13468911
 [46] -1.06343202 -0.96219478 -0.79705203 -0.61365764 -0.61076290
 [51]  0.06614811  0.16881684  0.78634836  0.92627384  0.98082320
 [56]  1.23222276  1.26040912  1.35389378  1.38890407  1.46203893
 [61]  1.64117907  1.64643080  1.72974830  1.77209389  1.83964734
 [66]  1.91725385  2.03169983  2.07038412  2.28388853  2.28508649
 [71]  2.44458357  2.54947996  2.68015883  2.72649568  2.73596466
 [76]  2.90513928  2.99685314  3.10296534  3.14159265  3.43473453
 [81]  3.46738955  3.80582747  3.89114580  3.96880166  3.97037737
 [86]  4.36755363  4.41302445  4.46414213  4.49439898  4.49756067
 [91]  4.59615407  5.01854928  5.04809437  5.07324581  5.12980909
 [96]  5.13371469  5.14677578  5.14849898  5.55918177  5.62584723
[101]  5.95526169
```

Here for simplicity and reasonable comparison, when estimating and predicting by the local smoothing methods, please use $span = 0.3365$ for loess, $bandwidth = 0.2$ for NW kernel smoothing, and $spar = 0.7163$ for spline splines.

Discuss the statistical challenges and the computational challenges when using these three local smoothing methods to estimate the Mexican hat function under this non-equidistant design, including the suitable choices of tuning parameters.

Appendix: below are some sample R codes that may be useful to this homework, and please note that you might need to modify/revise these R codes!

```
## Part #1  deterministic equidistant design
## Generate n=101 equidistant points in [-2\pi, 2\pi]
m <- 1000
n <- 101
x <- 2*pi*seq(-1, 1, length=n)

## Initialize the matrix of fitted values for three methods
fvlp <- fvnw <- fvss <- matrix(0, nrow= n, ncol= m)

##Generate data, fit the data and store the fitted values
for (j in 1:m){
  ## simulate y-values
  ## Note that you need to replace $f(x)$ below by the mathematical definition in eq. (2)
  y <- f(x) + rnorm(length(x), sd=0.2);

  ## Get the estimates and store them
  fvlp[,j] <- predict(loess(y ~ x, span = 0.75), newdata = x);
  fvnw[,j] <- ksmooth(x, y, kernel="normal", bandwidth= 0.2, x.points=x)$y;
  fvss[,j] <- predict(smooth.spline(y ~ x), x=x)$y
}

## Below is the sample R code to plot the mean of three estimators in a single plot
meanlp = apply(fvlp,1,mean);
meannw = apply(fvnw,1,mean);
meanss = apply(fvss,1,mean);

dmin = min( meanlp,  meannw,  meanss);
dmax = max( meanlp,  meannw,  meanss);
matplot(x, meanlp, "l", ylim=c(dmin, dmax), ylab="Response")
matlines(x, meannw, col="red")
matlines(x, meanss, col="blue")
## You might add the raw observations to compare with the fitted curves
# points(x,y)
## Can you adapt the above codes to plot the empirical bias/variance/MSE?


## Part #2  non-equidistant design
## assume you save the file "HW04part2.x.csv" in the local folder "C:/temp",
x2 <- read.table(file= "C:/temp/HW04part2.x.csv", header=TRUE);

## within each loop, you can consider the three local smoothing methods:
##   please remember that you need to first simulate Y values within each loop
  y <- (1-x2^2) * exp(-0.5 * x2^2) + rnorm(length(x2), sd=0.2);
  predict(loess(y ~ x2, span = 0.3365), newdata = x2);
  ksmooth(x2, y, kernel="normal", bandwidth= 0.2, x.points=x2)$y;
  predict(smooth.spline(y ~ x2, spar= 0.7163), x=x2)$y
```