Stat 432 Homework 7

Giang Le (gianghl2)

Assigned: Oct 4, 2021; Due: 11:59 PM CT, Oct 12, 2021

Contents

Question 1: Writing your own KNN and Kernel Regression 1

Question 1: Writing your own KNN and Kernel Regression

For this question, you are not allowed to use existing functions that directly calculate KNN or kernel regression, but you can still use any R function to calculate the components you need. For an example of such implementation of NW kernel regression, read the lecture note this week. The end result of this question is to write two functions: `myknn(x0, x, y, k)` and `mynw(x0, x, y, h)` that calculates the prediction of these two models at a target point `x0`, given the data vectors `x` and `y`, and the corresponding tuning parameters `k` and `h`. For `mynw()`, you should use the Gaussian kernel function. An additional requirement is that you **cannot use for-loop** in your code.

a) [25 points] Write your own `myknn()` and `mynw()`
b) [15 points] Test your code using the artificial data in the lecture note. Try to predict a target point at $x_0 = 2.5$. Compare your results with existing R functions `kknn()` and `locpoly()`, using `k = 10` and `h = 0.5`, respectively, make sure that your implementation is correct. For the `locpoly()` function, you may not be able to directly obtain the prediction at $x_0 = 2.5$. Hence, demonstrate your result in a figure to show that they are correct.

Below I implemented myknn() and mynw() functions.

```r
myknn <- function(x0, x, y, k) {
  dist = rep(0, length(x))
  # Calculate the euclidean distances between x and x0
  dist = sqrt((x - x0)^2)
  # Find the indices of the k nearest neighbors
  kindices = order(dist)[1:k]
  # calculate the estimator by summing y values corresponding to the indices
  # and take the average.
  fhat = sum(y[kindices])/k
  return(fhat)
}

# Use the Gaussian kernel function here.
mynw <- function(x0, x, y, h) {
  # calculate the kernel weights
  w = dnorm( (x0 - x)/h )/h
  # calculate the NW estimator
  fhat = sum(w*y)/sum(w)
  return(fhat)
}
```

First I generated some data and then I used the functions I implemented to predict fhat.

```
# generate some data
set.seed(662095561)
x <- runif(40, 0, 2*pi)
y <- 2*sin(x) + rnorm(length(x))
x_0 = 2.5

# Use my functions to predict
myknn(x_0, x, y, k=10)
```

```
## [1] 0.7578885
```

```
mynw(x_0, x, y, h=0.5)
```

```
## [1] 0.9782341
```

Use the built in R functions to predict. The knn.fit object returns the exact same value as the value returned by the function I implemented, as shown below.

```
install.packages("kknn", repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpoPzgFB/downloaded_packages
```

```
library("kknn")
knn.fit = kknn(y ~ x, train = data.frame("x" = x, "y" = y),
                 test = data.frame("x" = x_0), k = 10, kernel = "rectangular")

knn.fit$fitted.values
```

```
## [1] 0.7578885
```

```
knn.fit$fitted.values == myknn(x_0, x, y, k=10)
```

```
## [1] TRUE
```

Use the built-in R function to predict and plot the true y (blue), the estimated line by locpoly (orange), and the predicted value produced by mynw. We can see that the estimated line by locpoly passes through the predicted value produced by mynw, suggesting that the implementation agrees with the implementation of locpoly.

```
install.packages("KernSmooth", repos = "http://cran.us.r-project.org")
```

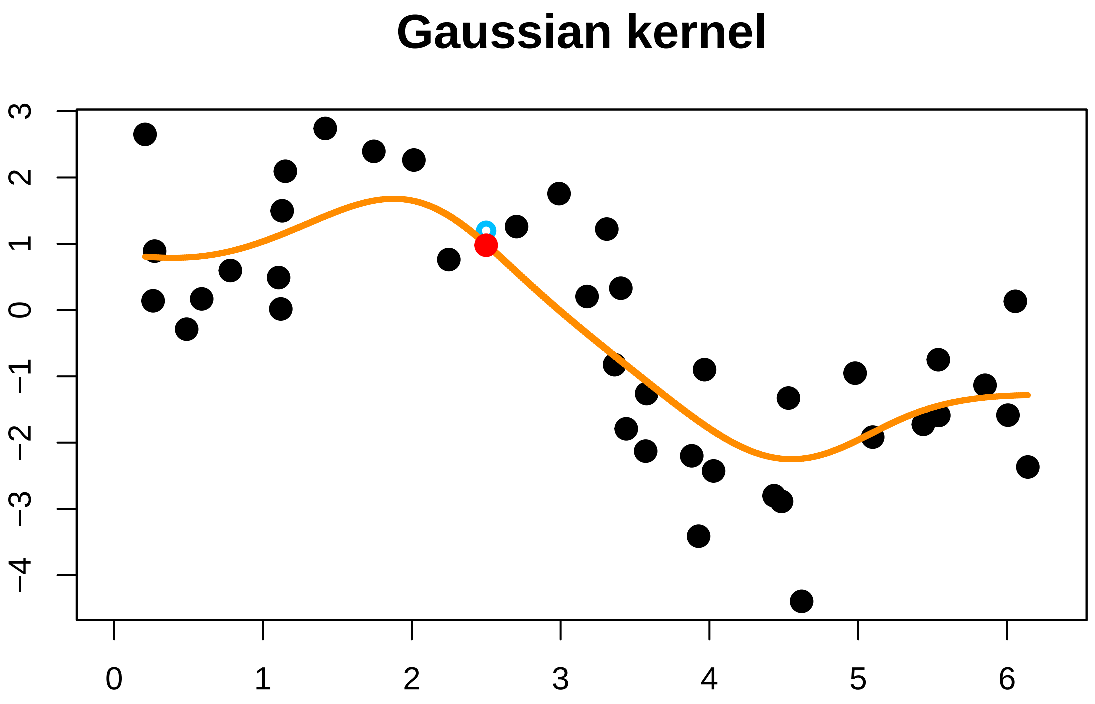
```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpoPzgFB/downloaded_packages
```

```
library(KernSmooth)
```

```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

```
NW.fit = locpoly(x, y, degree = 0, bandwidth = 0.5, kernel = "normal")

plot(x, y, xlim = c(0, 2*pi), cex = 1.5, xlab = "", ylab = "",
         cex.lab = 1.5, pch = 19)
title(main=paste("Gaussian kernel"), cex.main = 1.5)

# the true function
```

```r
points(x_0, 2*sin(x_0), col = "deepskyblue", lwd = 3)

# Kernel estimated function
lines(NW.fit$x, NW.fit$y, type = "s", col = "darkorange", lwd = 3)

# point estimated using mynw
points(x_0, mynw(x_0, x, y, h=0.5), col = "red", pch = 19, cex = 1.5)
box()
```



## Question 2: The Bias-variance Trade-off

We are going to perform a slightly more complicated simulation analysis of the bias-variance trade-off using prediction errors. Hence, you will need to utilize the functions you wrote in the previous question. We can then vary the tuning parameter k or h to see how they changes. Following the idea of simulation studies in previous HW assignments, setup a simulation study. Complete this question by performing the following steps. Note that you would have a triple-loop to complete this question, the first loop for k and h, the second loop for repeating the simulation `nsim` times, and the third loop for going through all testing points.

- Generate data using the same model in the previous question
- Generate 100 random testing points uniformly within the range of $[0, 2\pi]$ and also generate their outcomes.
- For each testing point, calculate both the kNN and kernel predictions using your own functions, with a given k and h
- Summarize your model fitting results by calculating the mean squared prediction error
- (the second loop) Run this simulation `nsim` $= 200$ times to obtain the averaged prediction errors.
- (the first loop) Vary your k from 1 to 20, and vary your h in `seq(0.1, 1, length.out = 20)`. Since both have length 20, you could write them in the same loop.

After obtaining the simulation results, provide a figure to demonstrate the bias-variance trade-off. What are the optimal k and h values based on your simulation? What kind of values of k or h would have large bias and small variance? And what kind of values would give small bias and large variance?

```r
# generate some data
set.seed(662095561)
```

```r
x <- runif(40, 0, 2*pi)
y <- 2*sin(x) + rnorm(length(x))
# Generate 100 random testing points uniformly within the range of $[0, 2\pi]$
testx = seq(from=0, to=2*pi, length.out=100)

# Generate outcomes
testy = 2*sin(testx)

# Predictions using my own functions, using k = 10 and h = 0.5
knn_pred = rep(0, length(testx))
nw_pred = rep(0, length(testx))
for (i in 1:length(testx)) {
  knn_pred[i] <- myknn(testx[i], x, y, k=10)
  nw_pred[i] <- mynw(testx[i], x, y, h=0.5)
}

knn_pred
```

```
##   [1]  0.82555376  0.82555376  0.82555376  0.82555376  0.82555376  0.82555376
##   [7]  0.82555376  0.82555376  0.82555376  0.82555376  0.82555376  0.82555376
##  [13]  0.82555376  0.83452031  0.83452031  0.83452031  1.06008662  1.06008662
##  [19]  1.06008662  1.19750705  1.19750705  1.19750705  1.30256765  1.30256765
##  [25]  1.30256765  1.30256765  1.41164409  1.41164409  1.41164409  1.41164409
##  [31]  1.52764858  1.52764858  1.52764858  1.52764858  1.49908464  1.61960819
##  [37]  1.21106750  1.21106750  1.21106750  0.75788846  0.75788846  0.75788846
##  [43]  0.30560972  0.30560972  0.30560972 -0.04672018 -0.04672018 -0.04672018
##  [49] -0.04672018 -0.34287483 -0.34287483 -0.34287483 -0.34287483 -0.80993377
##  [55] -0.80993377 -1.07550960 -1.07550960 -1.33872472 -1.33872472 -1.33872472
##  [61] -1.33872472 -1.33872472 -1.94738537 -2.11309546 -2.37333471 -2.37333471
##  [67] -2.37333471 -2.37333471 -2.25570873 -2.32138627 -2.32138627 -2.32138627
##  [73] -2.32138627 -2.32138627 -2.27386734 -2.07666953 -2.07666953 -2.07666953
##  [79] -1.94731365 -1.94731365 -1.94731365 -1.94731365 -1.94731365 -1.82573504
##  [85] -1.52380693 -1.62783929 -1.62783929 -1.62783929 -1.62783929 -1.62783929
##  [91] -1.62783929 -1.62783929 -1.62783929 -1.62783929 -1.62783929 -1.62783929
##  [97] -1.62783929 -1.62783929 -1.62783929 -1.62783929
```

```r
nw_pred
```

```
##   [1]  0.854491589  0.836936869  0.821336122  0.808173830  0.797972110
##   [6]  0.791275421  0.788629394  0.790554455  0.797515890  0.809893123
##  [11]  0.827951671  0.851821454  0.881484390  0.916772727  0.957377458
##  [16]  1.002864083  1.052691373  1.106228187  1.162763915  1.221509646
##  [21]  1.281589357  1.342022951  1.401705373  1.459387984  1.513669476
##  [26]  1.563003453  1.605727954  1.640118743  1.664463415  1.677148775
##  [31]  1.676751063  1.662118968  1.632443180  1.587311739  1.526754190
##  [36]  1.451276425  1.361880931  1.260057638  1.147725763  1.027113853
##  [41]  0.900584213  0.770430506  0.638690616  0.507012721  0.376593840
##  [46]  0.248187561  0.122161807 -0.001417644 -0.122698542 -0.241961690
##  [51] -0.359554463 -0.475838917 -0.591152259 -0.705776400 -0.819913484
##  [56] -0.933664935 -1.047012490 -1.159800655 -1.271720948 -1.382299191
##  [61] -1.490887834 -1.596665770 -1.698648093 -1.795707680 -1.886609267
##  [66] -1.970054979 -2.044738501 -2.109403760 -2.162903625 -2.204255002
##  [71] -2.232688475 -2.247692752 -2.249055472 -2.236901744 -2.211729532
##  [76] -2.174437181 -2.126334128 -2.069123185 -2.004843748 -1.935771140
##  [81] -1.864277386 -1.792670102 -1.723034368 -1.657103864 -1.596180998
##  [86] -1.541113783 -1.492324455 -1.449875466 -1.413554773 -1.382963803
##  [91] -1.357596114 -1.336900278 -1.320325248 -1.307349635 -1.297497985
##  [96] -1.290347551 -1.285528778 -1.282722079 -1.281652768 -1.282085384
```

Here I calculate the mean squared prediction errors when compared with testy. The nw predictions have a lower mean squared prediction errors compared to the knn method.

```
mean((knn_pred - testy)^2)
```

```
## [1] 0.3046665
```

```
mean((nw_pred - testy)^2)
```

```
## [1] 0.1735574
```

I write additional loops for nsim=200 and varying values of k and h below. I save the errors in a matrix.

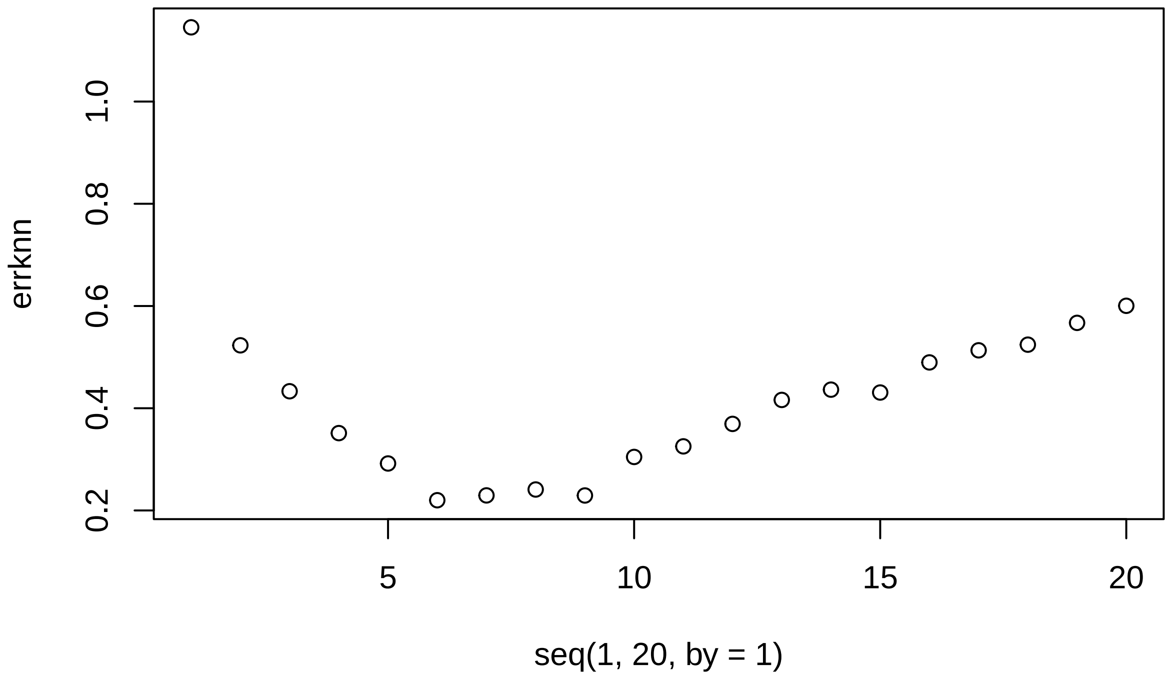
```
h = seq(0.1, 1, length.out=20)
k = seq(1, 20, by=1)
nsim=200

# Create a table for x (200 row simulations, 100 testing points each row)
#tablex = data.frame(matrix(ncol = 100, nrow = nsim))
#tabley = data.frame(matrix(ncol = 100, nrow = nsim))

tablex = array(0,dim=c(200,100,20))
tabley = array(0,dim=c(200,100,20))

# Create a table for predicted values
tableknn_pred = array(0,dim=c(200,100,20))
tablenw_pred = array(0,dim=c(200,100,20))


for (p in 1:length(k)) {    # this will take care of both k and h looping.
  for (j in 1:nsim) {   # 200 simulations
    for (m in 1:100) {
      # Generate 100 random testing points uniformly within the range of $[0, 2\pi]$
      tablex[,m,] = seq(from=0, to=2*pi, length.out=100)
      tabley[,m,] = 2*sin(tablex[,m,])

      # Make predictions
      tableknn_pred[j,m,p] <- myknn(tablex[j,m,p], x, y, k=k[p])
      tablenw_pred[j,m,p] <- mynw(tablex[j,m,p], x, y, h=h[p])

    }
  }
}
 # Calculate the mean squared errors
knnmeanerr = mean((tableknn_pred[j,m,p] - tabley[j,m,p])^2)
nwmeanerr = mean((tablenw_pred[j,m,p] - tabley[j,m,p])^2)
knnmeanerr
```

```
## [1] 3.498648
```

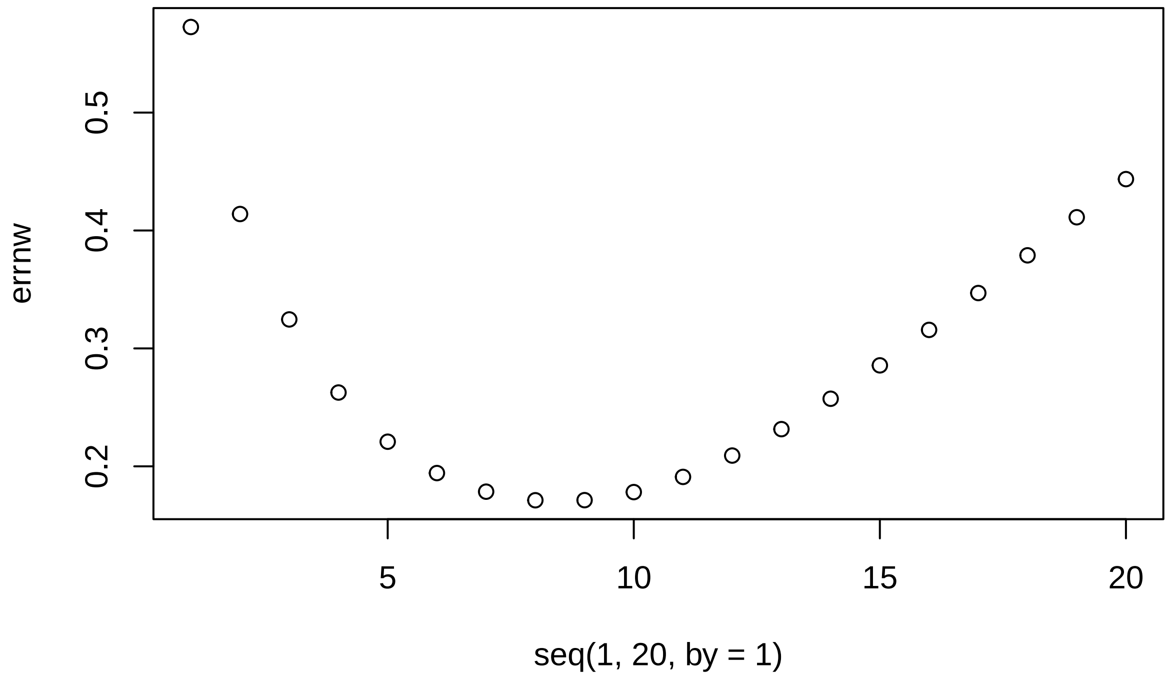
```
nwmeanerr
```

```
## [1] 2.228983
```

```r
# now I plot the mean squared error based on values of k and h
errknn = rep(0, 20)
errnw = rep(0, 20)
for (p in 1:20) {
  errknn[p] = mean((tableknn_pred[,,p] - tabley[,,p])^2)
  errnw[p] = mean((tablenw_pred[,,p] - tabley[,,p])^2)
}
plot(seq(1, 20, by=1), errknn)
```



```r
plot(seq(1, 20, by=1), errnw)
```



```r
h[6]
```

```
## [1] 0.3368421
```

```r
h[8]
```

```
## [1] 0.4315789
```

According to the plot, the mean squared prediction error decreases initially and then increases again. The optimal k and h are around index 6 for KNN and index 8 for NR. This corresponds to k = 6 or 8 and the values of h extracted above.

These plots show the relationship between the bias variance trade off. As k increases, we have a stable model with smaller variance but also higher bias. With smaller k, we have a less stable model with higher variance but also smaller bias. The prediction error is the sum of the irreducible error, bias squared, and variance. The decreasing prediction error is due to smaller variance initially as k increases. Similarly, the bandwidth h also reduces variance as it increases. However, eventually the prediction error increases again, probably due to the bias outgrowing gain in variance.

What kind of values of k or h would have large bias and small variance? high k and h And what kind of values would give small bias and large variance? small k and h