CS 598: Homework 3

Fall 2019, by Apoorva (apoorva6)

Contents

Question 1 A Simulation Study 1
Question 2 Multi-dimensional Kernel and Bandwidth Selection 3

Question 1 A Simulation Study

We will perform a simulation study to compare the performance of several different spline methods. Consider the following settings:

- Training data $n = 30$: Generate x from $[-1, 1]$ uniformly, and then generate $y = \sin(\pi x) + \epsilon$, where ϵ's are iid standard normal

```
set.seed(1)
n <- 30
x <- seq(from = -1, to = 1, length = n)
eps <- rnorm(n, mean = 0, sd = 1)
y <- sin(pi*x) + eps
```

- Consider several different spline methods:
  - Write your own code (you cannot use `bs()` or similar functions) to implement a continuous piecewise linear spline fitting. Choose knots at $(-0.5, 0, 0.5)$

```
myknots = c(-0.5, 0, 0.5)

basis_func <- function(x)
{
basis <- cbind("int" = 1, "x_1" = x,
                    "x_2" =  ifelse(x > myknots[1], x - myknots[1],0),
                    "x_3" =  ifelse(x > myknots[2], x - myknots[2],0),
                    "x_4" =  ifelse(x > myknots[3], x - myknots[3],0))
}

x_train_basis <- basis_func(x)
linear_fit <- lm(y ~ ., data = data.frame(x_train_basis))
```

- Use existing functions to implement a quadratic spline 2 knots. Choose your own knots.

```
library(splines)
quad_fit <- lm(y ~ bs(x,degree=2 , knots =c(0.5,0.5)))
```

- Use existing functions to implement a natural cubic spline with 3 knots. Choose your own knots.

```
nat_fit <- lm(y~ ns(x,knots =c(-0.5,0,0.5)))
```

- Use existing functions to implement a smoothing spline. Use the built-in ordinary leave-one-out cross-validation to select the best tuning parameter.

```
smooth_fit <- smooth.spline(x,y,cv=TRUE)
```

- After fitting these models, evaluate their performances by comparing the fitted functions with the true function value on an equispaced grid of 1000 points on $[-1, 1]$. Use the squared distance as the metric.

```r
x_test <- seq(from = -1, to = 1, length = 1000)
y_true <-  sin(pi*x_test)

######### Predict y values using continuous piecewise linear spline #############
x_test_basis <- basis_func(x_test)
pred_linear_fit <- predict(linear_fit,data.frame(x_test_basis))
err <- sum((y_true-pred_linear_fit)^2)
err
```

```
## [1] 68.50255
```

```r
######### Predict y values using quadratic spline #############
pred_quad <- predict(quad_fit,newdata= list(x= x_test),se=T)
err<- sum((y_true-pred_quad$fit)^2)
err
```

```
## [1] 128.6484
```

```r
######### Predict y values using Natural cubic spline #############
pred_nat <- predict(nat_fit,newdata= list(x= x_test),se=T)
err<- sum((y_true-pred_nat$fit)^2)
err
```

```
## [1] 57.29681
```

```r
######### Predict y values using smoothing spline #############
pred_smooth <- predict(smooth_fit, x_test)$y
err<- sum((y_true-pred_smooth)^2)
err
```

```
## [1] 63.19859
```

- Repeat the entire process 200 times. Record and report the mean, median, and standard deviation of the errors for each method. Also, provide an informative boxplot that displays the error distribution for all models side-by-side.

```r
error_cont_linear <-  c()
error_quad <-  c()
error_natural_cs <-  c()
error_smoothing <-  c()
for(i in 1:200)
{
  # x and y for Training
  n <- 30
  x <- seq(from = -1, to = 1, length = n)
  eps <- rnorm(n, mean = 0, sd = 1)
  y <- sin(pi*x) + eps

  # Continuous piecewise linear spline
  x_train_basis <- basis_func(x)
  linear_fit <- lm(y ~ ., data = data.frame(x_train_basis))
  pred_linear_fit <- predict(linear_fit,data.frame(x_test_basis))
  error_cont_linear[i] <- sum((y_true-pred_linear_fit)^2)

  # Quad spline
  quad_fit <- lm(y ~ bs(x,degree=2 , knots =c(0.5,0.5)))
  pred_quad <- predict(quad_fit,newdata= list(x= x_test),se=T)
  error_quad[i] <- sum((y_true-pred_quad$fit)^2)
```

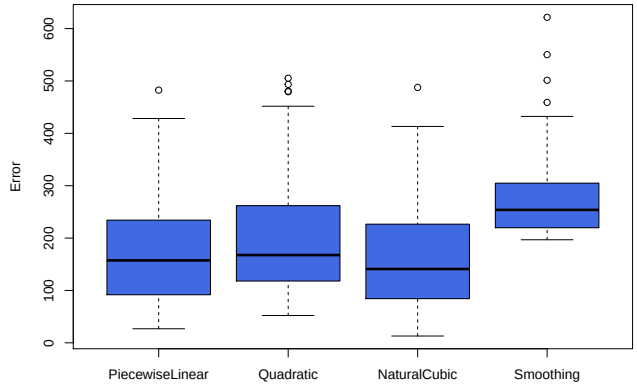
```
  # Natural cubic spline
  nat_fit <- lm(y~ ns(x,knots =c(-0.5,0,0.5)))
  pred_nat <- predict(nat_fit,newdata= list(x= x_test),se=T)
  error_natural_cs[i] <- sum((y_true-pred_nat$fit)^2)

  # Smoothing spline
  smooth_fit <- smooth.spline(x,y,df=2)
  pred_smooth <- predict(smooth_fit, x_test)$y
  error_smoothing[i] <- sum((y_true-pred_smooth)^2)
}

report <- data.frame(mean = mean(error_cont_linear),median = median(error_cont_linear), sd = sqrt(var(e
report_mod2 <- c(mean = mean(error_quad),median = median(error_quad), sd = sqrt(var(error_quad)))
report_mod3 <- c(mean = mean(error_natural_cs),median = median(error_natural_cs), sd = sqrt(var(error_na
report_mod4 <- c(mean = mean(error_smoothing),median = median(error_smoothing), sd = sqrt(var(error_smoo
report <- rbind(report,report_mod2,report_mod3,report_mod4)
rownames(report) <- c("Continuous.Linear.piecewise","Quadratic.spline","Natural.cubic","Smoothing.spline
print(report)
```

```
##                                mean   median       sd
## Continuous.Linear.piecewise 172.3292 157.3947 99.12947
## Quadratic.spline            194.3465 167.5876 99.29416
## Natural.cubic               162.9203 140.9405 99.13505
## Smoothing.spline            273.2919 253.8536 70.30389
```

```
alis <- list(error_cont_linear,error_quad,error_natural_cs,error_smoothing)
boxplot(alis, names =c("PiecewiseLinear","Quadratic","NaturalCubic","Smoothing"),ylab = "Error",col="ro
```



- Comment on your findings. Which method would you prefer?

From the above report we see that the Natural cubic spline model performs the best, as it has the lowest error innterms of mean, median and variance. The smoothing spline seems to have the higest error among the four models.

Question 2 Multi-dimensional Kernel and Bandwidth Selection

Let's consider a regression problem with multiple dimensions. For this problem, we will use the Combined Cycle Power Plant (CCPP) Data Set available at the UCI machine learning repository. The goal is to predict the net hourly electrical energy output (EP) of the power plant. Four variables are available: Ambient Temperature (AT), Ambient Pressure (AP), Relative Humidity (RH), and Exhaust Vacuum (EV). For more details, please go to the dataset webpage. We will use a kernel method to model the outcome. A multivariate

Gaussian kernel function defines the distance between two points:

$$K_{\boldsymbol{\lambda}}(x_i, x_j) = e^{-\frac{1}{2}\sum_{j=1}^{p}((x_i k - x_j k)/\lambda_k)^2}$$

The most crucial element in kernel regression is the bandwidth λ_j. A popular choice is the Silverman formula. The bandwidth for the jth variable is given by

$$\lambda_k = \left(\frac{4}{p+2}\right)^{\frac{1}{p+4}} n^{-\frac{1}{p+4}} \widehat{\sigma}_k,$$

where $\widehat{\sigma}_j$ is the estimated standard deviation for variable j, p is the number of variables, and n is the sample size. Based on this kernel function, use the Nadaraya-Watson kernel estimator to fit and predict the data. You should consider the following:

- Randomly select 2/3 of the data as training data, and rest as testing. Make sure you set a random seed. You do not need to repeat this process — just fix it and complete the rest of the questions

```
# Read the data
CCPP <- read.csv("CCPP.csv")

# Splitting data into training and test
set.seed(1010)
smp <- floor(2/3 * nrow(CCPP))
index <- sample(seq_len(nrow(CCPP)), size = smp)
train_df <- CCPP[index, ]
test_df <-  CCPP[-index, ]
train <- as.matrix(CCPP[index, ])
test <-  as.matrix(CCPP[-index, ])
```

- Fit the model on the training samples using the kernel estimator and predict on the testing sample. Calculate the prediction error and compare this to a linear model
- The bandwidth selection may not be optimal in practice. Experiment a few choices and see if you can achieve a better result.
- During all calculations, make sure that you write your code efficiently to improve computational performance

```
p<- ncol(train)-1
n<- nrow(train)
lambda <- function(x) (x-2)

mat <- matrix(,nrow=nrow(test),ncol=nrow(train),byrow=T)
newmat <- matrix(0,nrow=nrow(test),ncol= nrow(train) , byrow=T)
for(k in 1:4)
{
  for (i in 1:nrow(test) )
  {

    lam <- lambda(p)
    mat[i,] <-  (exp(-0.5*(((test[i,k]-train[,k])/lam)^2))) /sqrt(2*pi)
  }
  newmat <- newmat +  mat
}
test_predicted<- rowSums(newmat %*% train[,5]) /(rowSums(newmat))

mse <- mean((test[,5]-test_predicted)^2)
```

```
# Linear model
linearmodel <- lm(EP ~ .,data=train_df)
pred <- predict(linearmodel,test_df[,1:4])
mse <- mean((pred-test_df[,5])^2)
```

- The MSE for the Kernel regression is 17.2636416
- MSE for Linear regression is 20.8167345
- Thus Kernel regression performs better on the test dataset than Linear regression.

```
system.time({lambda <- function(x) (x-2)

mat <- matrix(,nrow=nrow(test),ncol=nrow(train),byrow=T)
newmat <- matrix(0,nrow=nrow(test),ncol= nrow(train) , byrow=T)
for(k in 1:4)
{
  for (i in 1:nrow(test) )
  {

    lam <- lambda(p)
    mat[i,] <-  (exp(-0.5*(((test[i,k]-train[,k])/lam)^2))) /sqrt(2*pi)
  }
  newmat <- newmat +  mat
}
test_predicted<- rowSums(newmat %*% train[,5]) /(rowSums(newmat))})
```

```
##    user  system elapsed
##   3.132   0.832   3.999
```