Kernels

Rishabh Vaish

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:

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

The most crucial element in kernel regression is the bandwidth λ_k. A popular choice is the Silverman formula. The bandwidth for the kth 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 $\widehat{\sigma}_k$ is the estimated standard deviation for variable k, 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 xlxs
library(readxl)
dataset <- as.matrix(read_excel("CCPP/Folds5x2_pp.xlsx"))

#generate data
set.seed(1)
sample <-
  sample.int(n = nrow(dataset),
             size = floor(.66 * nrow(dataset)),
             replace = F)
#split test and train
train <- dataset[sample,]
xtrain <- dataset[sample, 1:4]
ytrain <- dataset[sample, 5]
test  <- dataset[-sample,]
xtest <- dataset[-sample, 1:4]
ytest <- dataset[-sample, 5]
```

- 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

```r
#noting time for computayional performance
start.time <- Sys.time()

#kernel function
kernel <- function(x_test, x_train, lambda) {
  s1 <- sweep(x_train, 2, x_test)
  s2 <- sweep(s1, 2, lambda, FUN = '/')
  s3 <- rowSums(s2 ^ 2)
  return(exp(-s3 / 2))
}

p <- ncol(xtrain)
n <- nrow(xtrain)

#silverman formula
lambda <- rep(0, p)
for (i in 1:p) {
  lambda[i] <-
    ((4 / (p + 2)) ^ (1 / (p + 4))) * (n ^ (-1 / (p + 4))) * (sd(as.matrix(xtrain[, i])))
}

y_hat <- rep(1, nrow(test))

# prediction using kernel for alll test cases
for (i in 1:nrow(test)) {
  kweight <- kernel(xtest[i,], xtrain, lambda)
  y_hat[i] <- sum(kweight * ytrain) / sum(kweight)
}

#linear model
linear <- lm(PE ~ ., data = data.frame(train))
y_hat_linear <- predict(linear, newdata = data.frame(test))

# Mean Square Error
error_kernel <- mean((ytest - y_hat) ^ 2)
error_linear <- mean((ytest - y_hat_linear) ^ 2)

#results
error_kernel
```

```
## [1] 17.77328
```

```r
error_linear
```

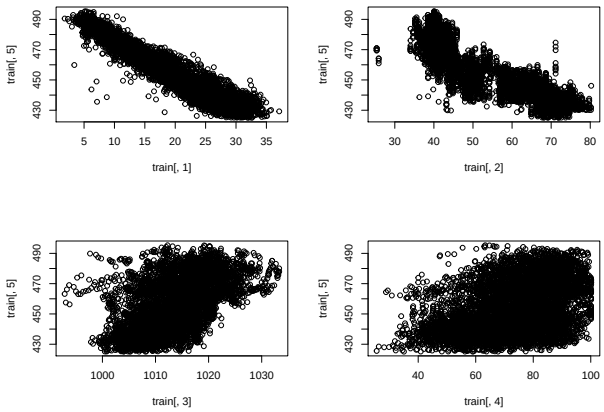
```
## [1] 20.9405
```

```r
end.time <- Sys.time()
```

- The bandwidth selection may not be optimal in practice. Experiment a few choices and see if you can achieve a better result.

For selecting a better lambda value we should visualise the spread of y with all predictors.

```
par( mfrow = c( 2, 2 ) )
plot(train[,1], train[,5])
plot(train[,2], train[,5])
plot(train[,3], train[,5])
plot(train[,4], train[,5])
```



The column 4 values are pretty well spread irrespective of Y. Hence we can choose a higher bandwidth for this. Similarly for column 3, we can keep reduce the bandwidth but not by much. For column 1 and 2 the Y values change significantly with change in X. Hence we should reduce the bandwidth for these columns.

```
#fixing all the lambda except for column 1. Reducing its bandwidth by a factor of 10
lambda_new <- lambda
lambda_new[1] <-  lambda_new[1]/10

y_hat_new <- rep(0, nrow(test))
for (i in 1:nrow(test)) {
  kweight <- kernel(xtest[i, ], xtrain, lambda_new)
  y_hat_new[i] <- sum(kweight * ytrain) / sum(kweight)
}
error_kernel_new <- mean((ytest - y_hat_new) ^ 2)
error_kernel_new
```

```
## [1] 15.31421
```

```
#similarly now reducing bandwidth of second column by a factor of 10
lambda_new[2] <-  lambda_new[2]/10

y_hat_new <- rep(1, nrow(test))
for (i in 1:nrow(test)) {
  kweight <- kernel(xtest[i, ], xtrain, lambda_new)
  y_hat_new[i] <- sum(kweight * ytrain) / sum(kweight)
}
error_kernel_new <- mean((ytest - y_hat_new) ^ 2)
error_kernel_new
```

```
## [1] 13.59389
```

```r
#similarly now reducing bandwidth of third column by a factor of 5
lambda_new[3] <-  lambda_new[3]/5

y_hat_new <- rep(1, nrow(test))
for (i in 1:nrow(test)) {
  kweight <- kernel(xtest[i, ], xtrain, lambda_new)
  y_hat_new[i] <- sum(kweight * ytrain) / sum(kweight)
}
error_kernel_new <- mean((ytest - y_hat_new) ^ 2)
error_kernel_new
```

```
## [1] 13.47304
```

```r
lambda_new
```

```
## [1] 0.2360708 0.4027452 0.3798649 4.6533968
```

- During all calculations, make sure that you write your code efficiently to improve computational performance

```r
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 3.507602 secs
```