Homework 5

DS Student

January 15, 2020

Randomly split the mcycle data into training (75%) and validation (25%) subsets.

```
library('MASS')
data <- mcycle
dim(data)[1]
```

```
## [1] 133
```

```
train_data_ind <- sample(dim(data)[1], dim(data)[1]*0.75)
train_data <- data[train_data_ind,]
test_data <- data[-train_data_ind,]
train_y <- train_data$accel
train_x <- matrix(train_data$times, length(train_data$times), 1)
test_y <- test_data$accel
test_x <- matrix(test_data$times, length(test_data$times), 1)
```

Using the mcycle data, consider predicting the mean acceleration as a function of time. Use the Nadaraya-Watson method with the k-NN kernel function to create a series of prediction models by varying the tuning parameter over a sequence of values. (hint: the script already implements this)

```
Nadaraya_Watson <- function(y,x,x0,kern,...){
  a <- t(apply(x0, 1, function(x0_){
    k <- kern(x,x0_,...)
    k/sum(k)
    }))

    y_hat <- drop(a %*% y)
    attr(y_hat,"k") <- a
    return(y_hat)
}

kernel_k_nearest_neighbors <- function(x, x0, t=1) {
  ## compute distance betwen each x and x0
  z <- t(t(x) - x0)
  d <- sqrt(rowSums(z*z))

  ## initialize kernel weights to zero
  w <- rep(0, length(d))

  ## set weight to 1 for k nearest neighbors
  w[order(d)[1:t]] <- 1
```

```
  return(w)
}
k1 <- seq(1,20,1)
for (i in k1){
  y_hat <- Nadaraya_Watson(y = train_y,x = train_x,x0 = test_x, kern = kernel_k_nearest_neighbors, t = i
}
```

With the squared-error loss function, compute and plot the training error, AIC, BIC, and validation error (using the validation data) as functions of the tuning parameter.

```
effective_df <- function(y, x, kern, ...) {
  y_hat <- Nadaraya_Watson(y, x, x,
    kern=kern, ...)
  sum(diag(attr(y_hat, 'k')))
}


loss_squared_error <- function(y, yhat)
  (y - yhat)^2

error <- function(y, yhat, loss=loss_squared_error)
  mean(loss(y, yhat))

aic <- function(y, yhat, d)
  error(y, yhat) + 2/length(y)*d

bic <- function(y, yhat, d)
  error(y, yhat) + log(length(y))/length(y)*d

AIC <- rep(0,1)
BIC <- rep(0,1)
training_error <- rep(0,1)
testing_error <- rep(0,1)
s = 1

for (i in (1:length(k1))){
  edf <- effective_df(train_y, train_x, kernel_k_nearest_neighbors, t=k1[i])
  y_hat_train <- Nadaraya_Watson(y = train_y,x = train_x,x0 = train_x, kern = kernel_k_nearest_neighbors
  y_hat_test <- Nadaraya_Watson(y = train_y,x = train_x,x0 = test_x, kern = kernel_k_nearest_neighbors,
  AIC[i] <- aic(train_y, y_hat_train,edf)
  BIC[i] <- bic(train_y, y_hat_train,edf)
  training_error[i] <- error(train_y, y_hat_train)
  testing_error[i] <- error(test_y, y_hat_test)
}
print(training_error)
```

```
##  [1] 320.1546 372.5687 457.5817 463.8927 454.4295 517.9466 500.9482 521.6720
##  [9] 537.4512 555.6848 569.4499 566.2044 582.1698 586.7487 596.6058 617.0251
## [17] 631.5184 653.4601 682.1540 698.1436
```

```
print(testing_error)
```

```
##  [1] 805.6288 477.4496 429.3591 448.2053 467.5124 492.8298 491.7712 477.8368
##  [9] 546.3712 589.8410 585.5567 596.5917 636.0091 673.9999 693.5575 755.5977
## [17] 759.3287 776.1342 822.4382 865.0851
```

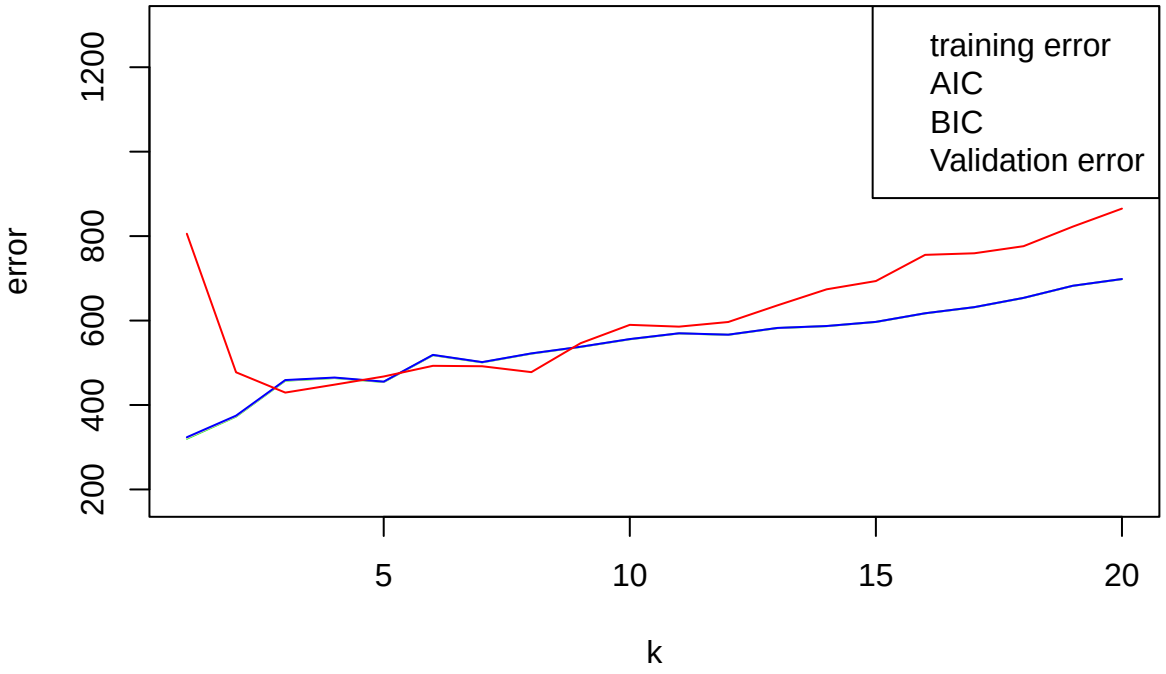
```
print(AIC)
```

```
##  [1] 321.6900 373.5081 458.2282 464.3876 454.8295 518.2799 501.2339 521.9220
##  [9] 537.6734 555.8848 569.6317 566.3711 582.3237 586.8916 596.7392 617.1501
## [17] 631.6361 653.5712 682.2593 698.2436
```

```
print(BIC)
```

```
##  [1] 323.6822 374.7270 459.0670 465.0299 455.3485 518.7124 501.6046 522.2464
##  [9] 537.9617 556.1443 569.8676 566.5874 582.5233 587.0770 596.9122 617.3123
## [17] 631.7887 653.7154 682.3959 698.3733
```

```
plot(k1,training_error,type = "l", col = "green",ylim = c(180,1300),xlab = "k",ylab = "error")
lines(k1,AIC,col = "pink",type = "l")
lines(k1,BIC,col = "blue",type = "l")
lines(k1,testing_error,col = "red",type = "l")
legend("topright", c("training error", "AIC", "BIC", "Validation error"), col = c("green","pink","blue"
```



For each value of the tuning parameter, Perform 5-fold cross-validation using the combined training and validation data. This results in 5 estimates of test error per tuning parameter value.

```
f <- 5
k1 <- seq(1,20,1)
folds <- sample(rep(1:5,length(data)))
err <- rep(0,1)
for (i in (1:length(k1))){
  e <- rep(0,5)
  for (j in (1:5)){
    train_y <- train_data$accel[folds!= j]
```

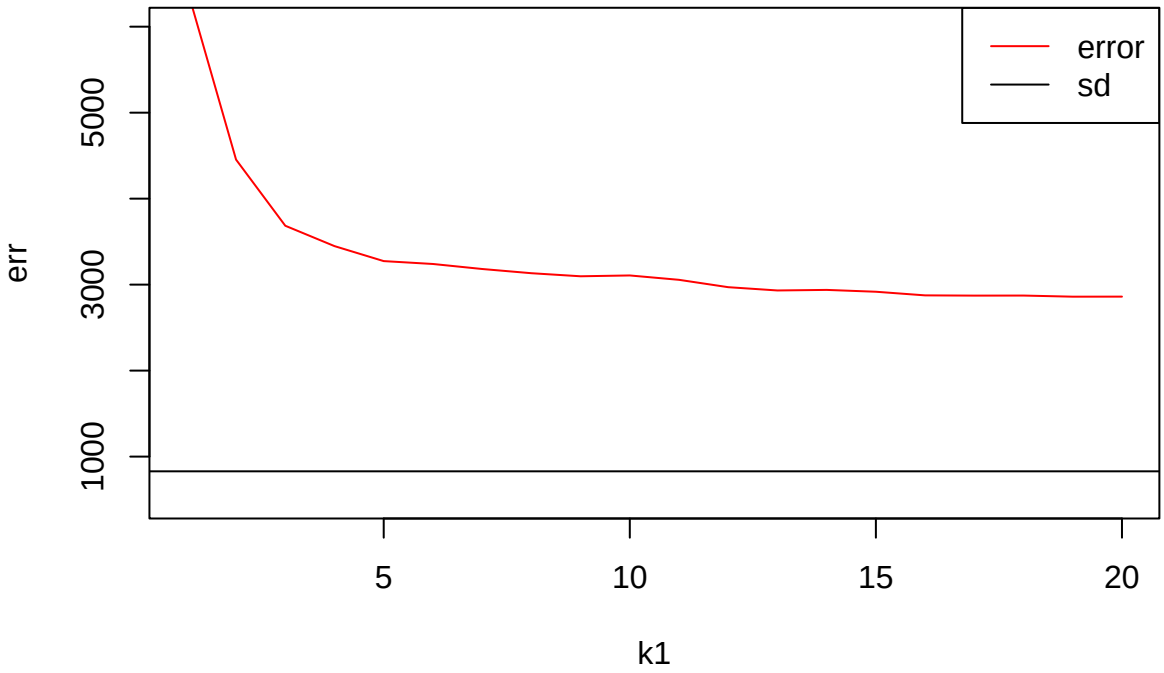
```
    train_x <- matrix(data$times[folds!= j], length(train_data$times[folds!= j]), 1)
    test_y <- test_data$accel
    test_x <- matrix(test_data$times[folds== j], length(test_data$times[folds== j]), 1)
    # train model
    y_hat <- Nadaraya_Watson(y = train_y,x = train_x,x0 = test_x, kern = kernel_k_nearest_neighbors, t =
    # error of every validation

    e[j] <- error(test_y, y_hat)
  }
  print(e)
  err[i] <- mean(e)
}
```

```
## [1] 6745.673 2822.838 8006.009 7492.444 7176.011
## [1] 3574.041 2819.122 6264.540 4282.447 5329.627
## [1] 2949.285 3157.726 4779.390 3948.273 3589.609
## [1] 3032.822 3232.783 4323.946 3345.568 3304.321
## [1] 3109.740 3322.604 3749.966 3158.972 3026.015
## [1] 3027.840 3224.041 3688.455 2986.209 3274.349
## [1] 2998.463 3077.788 3658.339 3032.905 3141.144
## [1] 2874.457 3131.028 3715.379 3058.293 2884.891
## [1] 2969.626 3019.244 3538.303 3038.185 2920.006
## [1] 2938.256 3052.619 3549.980 3076.946 2915.235
## [1] 2971.246 2966.557 3357.519 3104.923 2878.523
## [1] 2901.640 2881.613 3182.644 2989.697 2895.195
## [1] 2944.280 2894.273 3003.405 2933.907 2884.274
## [1] 2948.171 2972.288 3009.193 2885.173 2878.962
## [1] 2931.702 2910.696 3053.058 2813.649 2876.318
## [1] 2816.147 2884.160 3048.740 2789.855 2836.798
## [1] 2793.361 2849.494 2997.985 2830.572 2889.772
## [1] 2828.832 2803.281 2987.282 2859.722 2885.488
## [1] 2762.623 2781.441 2957.015 2877.291 2920.157
## [1] 2722.827 2773.355 2935.994 2917.280 2953.272
```

Plot the CV-estimated test error (average of the five estimates from each fold) as a function of the tuning parameter. Add vertical line segments to the figure (using the segments function in R) that represent one "standard error" of the CV-estimated test error (standard deviation of the five estimates from each fold).

```
sd <- sd(err)
plot(k1,err,type = "l",col = "red", ylim = c(500,6000))
abline(h = sd)
legend("topright",c("error", "sd"), col = c("red", "black"),lty = 1 )
```



Interpret the resulting figures and select a suitable value for the tuning parameter.

From the plot we can see that when k = 10, the test and train plot is the smallest. When k >10 and k increases, test and train error also increase. Waht's more, AIC, BIC are similar with the train error. Therefore, suitable k value is about 10.