K-fold Cross Validation

Plot

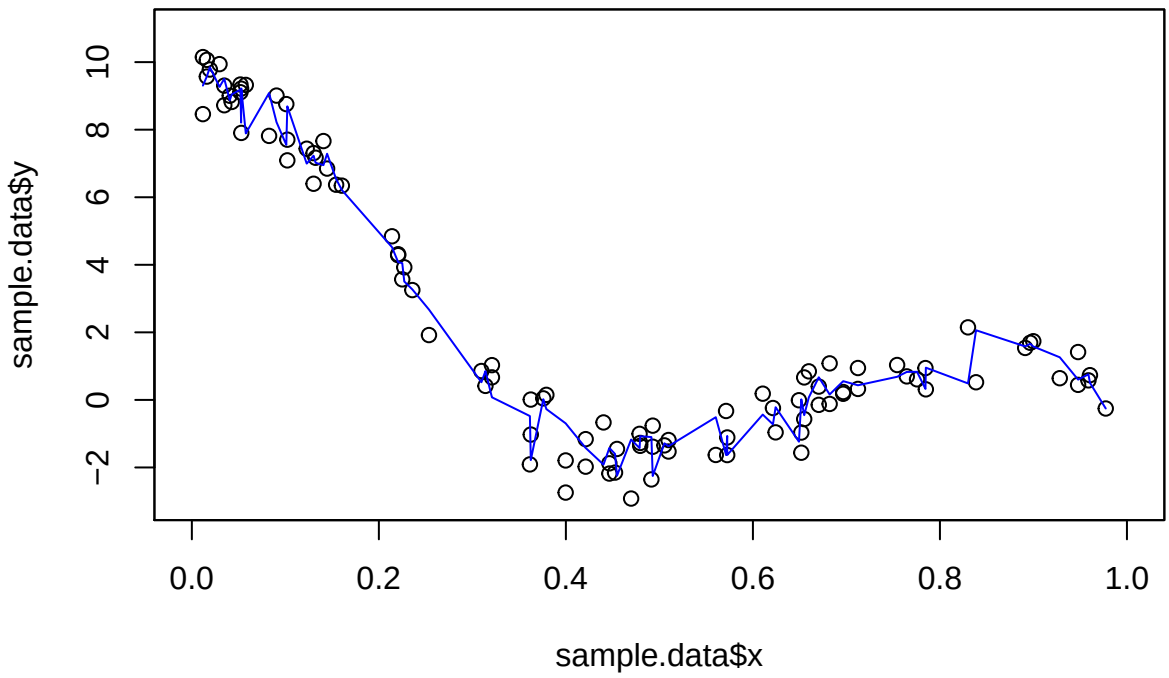
```
options(warn=-1)

rm(list=ls())
n=100
set.seed(341)
N <- 125
# Get some x's
x <- runif(N, 0, 1)
x = rep(x, each=2)
mu <- function(x) { sin( 3*pi*x )/x }
# generate some ys
y <- mu(x) + rnorm(N, 0, .6)
# Here's the data
fake.data = data.frame(x=x, y=y)
fake.sample = sample(nrow(fake.data), n)
sample.data = fake.data[fake.sample,]

getmuFun <- function(pop, xvarname, yvarname){
  ## First remove NAs
  pop <- na.omit(pop[, c(xvarname, yvarname)])
  x <- pop[, xvarname]
  y <- pop[, yvarname]
  xks <- unique(x)
  muVals <- sapply(xks,
                   FUN = function(xk) {
                     mean(y[x==xk])
                   })
  ## Put the values in the order of xks
  ord <- order(xks)
  xks <- xks[ord]
  xkRange <-xks[c(1,length(xks))]
  minxk <- min(xkRange)
  maxxk <- max(xkRange)
  ## mu values
  muVals <- muVals[ord]
  muRange <- muVals[c(1, length(muVals))]
  muFun <- function(xVals){
    ## vector of predictions
    ## same size as xVals and NA in same locations
    predictions <- xVals
    ## Take care of NAs
    xValsLocs <- !is.na(xVals)
    ## Just predict non-NA xVals
    predictions[xValsLocs] <- sapply(xVals[xValsLocs],
                                     FUN = function(xVal) {
                                       if (xVal < minxk) {
                                         result <- muRange[1]
```

```
                                        } else
                                          if(xVal > maxxk) {
                                            result <- muRange[2]
                                          } else
                                          {
                                            xlower <- max(c(minxk, xks[xks < xVal]))
                                            xhigher <- min(c(maxxk, xks[xks > xVal]))
                                            mulower <- muVals[xks == xlower]
                                            muhigher <- muVals[xks == xhigher]
                                            interpolateFn <- approxfun(x=c(xlower, xhigher),
                                                                       y=c(mulower, muhigher))
                                            result <- interpolateFn(xVal)
                                          }
                                        result
                                      }
    )
    ## Now return the predictions (including NAs)
    predictions
  }
  muFun
}


muhat <- getmuFun(sample.data, "x", "y")
plot(sample.data$x,sample.data$y, xlim = c(0,1), ylim = c(-3,11))
par(new = TRUE)
plot(x = sort(sample.data$x), y = muhat(sort(sample.data$x)), xlim = c(0,1), ylim = c(-3,11), yaxt = "n
     xaxt = "n", col = "blue", xlab ="", ylab = "", type = "l")
```




```
par(new = FALSE)
```

Create k-fold function

```
popSize <- function(pop) {nrow(as.data.frame(pop))}


### This function will return a data frame containing
### only two variates, an x and a y
getXYSample <- function(xvarname, yvarname, samp, pop) {
  sampData <- pop[samp, c(xvarname, yvarname)]
  names(sampData) <- c("x", "y")
  sampData
}


kfoldsampfn<- function(k, pop, xvarname,yvarname) {
  #list = list(Ssamples, Tsamples)
  #split pop into k groups randomly
  N <- popSize(pop)
  grpsize <- N%/% k
  N <- grpsize * k  #make sure divisible
  ind = sample(rep(1:k,each = grpsize))
  indlst <- split(1:N,ind)
```

```
  Ssamples <- list()
  Tsamples <- list()

  for (i in 1:k){
   Sind <- unlist(indlst[paste(combn(k,k-1)[,i])])
   Ssamples[[i]] <- getXYSample(xvarname, yvarname, Sind, pop)
   Tind <- unlist(indlst[paste(k-i+1)])
   Tsamples[[i]] <- getXYSample(xvarname, yvarname, Tind, pop)
   }

  list(Ssamples, Tsamples)
}
```

Estimate APSE using k=5 fold cv when complexity = 2

```
getmuhat <- function(sampleXY, complexity = 1) {
  formula <- paste0("y ~ ",
                    if (complexity==0) {
                      "1"
                    } else
                      paste0("poly(x, ", complexity, ", raw = TRUE)")
  )

  fit <- lm(as.formula(formula), data = sampleXY)

  ## From this we construct the predictor function
  muhat <- function(x){
    if ("x" %in% names(x)) {
      ## x is a dataframe containing the variate named
      ## by xvarname
      newdata <- x
    } else
      ## x is a vector of values that needs to be a data.frame
    {newdata <- data.frame(x = x) }
    ## The prediction
    predict(fit, newdata = newdata)
  }
  ## muhat is the function that we need to calculate values
  ## at any x, so we return this function from getmuhat
  muhat
}


ave_y_mu_sq <- function(sample, predfun, na.rm = TRUE){
  mean((sample$y - predfun(sample$x))^2, na.rm = na.rm)
}

apse <- function(Ssamples, Tsamples, complexity){
  ## average over the samples S
  ##
  N_S <- length(Ssamples)
```

```
  mean(sapply(1:N_S,
              FUN=function(j){
                S_j <- Ssamples[[j]]
                ## get the muhat function based on
                ## the sample S_j
                muhat <- getmuhat(S_j, complexity = complexity)
                ## average over (x_i,y_i) in a
                ## single sample T_j the squares
                ## (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_y_mu_sq(T_j,muhat)
              }
  )
  )
}

k <- 5
kfoldsamps <- kfoldsampfn(k, sample.data, "x", "y")
apse(kfoldsamps[[1]],kfoldsamps[[2]], complexity=2)
```

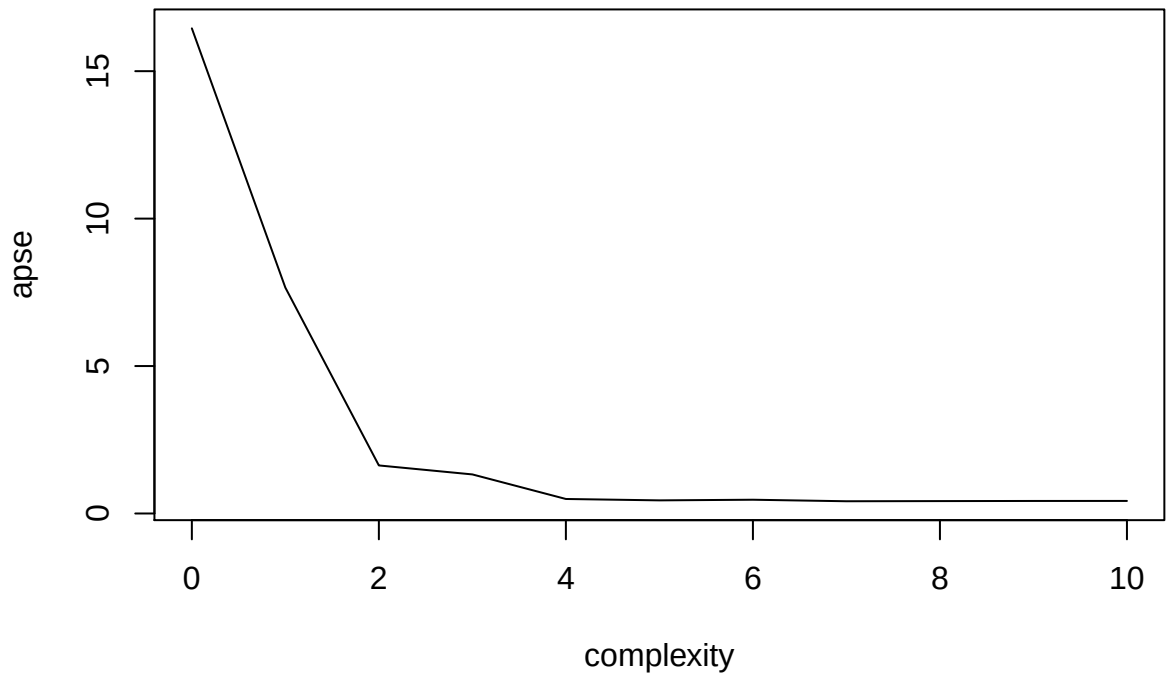
```
## [1] 1.629065
```

Perform k=5-fold cv to estimate the complexity parameter from 0-10. Plot APSE vs Complexity.

```
apse_vals <- lapply(0:10, function(i){apse(kfoldsamps[[1]],kfoldsamps[[2]], complexity=i)})
apse_vals <- unlist(apse_vals)
match(min(apse_vals), apse_vals)-1
```

```
## [1] 7
```

```
plot(x = 0:10, y = apse_vals, xlab  = "complexity", ylab = "apse", type = "l")
```



apse is lowest at complexity = 7.

Perform B = 100 bootstrap by resampling the errors to quantify the uncertainly in the complexity parameter. Construct a histogram of the complexity parameters.

```
muhat7 <- getmuhat(sample.data, complexity=7)
yhat <- muhat7(sample.data$x)
res <- sample.data$y - yhat
B <- 100
minlst <- rep(0,B)

for (i in 1:B){
  ystar <- yhat + res[sample(n,n,replace = TRUE)]
  boot.data <- sample.data
  boot.data$y <- ystar
  #bootmuhat <- getmuFun(boot.data, "x", "y")
  boot.kfold <- kfoldsampfn(k=5, boot.data, "x", "y")
  apse_vals <- lapply(0:10, function(c){apse(boot.kfold[[1]],boot.kfold[[2]], complexity=c)})
  apse_vals <- unlist(apse_vals)
  minlst[i] <- match(min(apse_vals), apse_vals)-1
}
#occurence <- sapply(0:10, function(i){sum(minlst == i)})
```

```r
hist(minlst)
```