Homework9

Jun Rao

10/12/2020

Homework 9: Dynamic programming for optimal changepoint detection

library(data.table)  
library(ggplot2)  
library(microbenchmark)

The goal of this homework is to explore dynamic programming algorithms for optimal changepoint detection, and compare to the binary segmentation algorithm. programmatically download the following files, which contain raw/noisy data signals and labels for several different sequenceID numbers. For each file, what is the min/max number of data/labels per sequenceID?

# use data.table::fread to read the compressed CSV data file into R as a data table.  
dt\_signals<- fread("data-for-LOPART-signals.csv.gz")  
df\_signals<-as.matrix(dt\_signals)  
head(sort(table(dt\_signals[["sequenceID"]])))

##   
## 145.19 173.19 66.21 162.19 371.14 332.18   
## 39 39 40 42 51 53

tail(sort(table(dt\_signals[["sequenceID"]])))

##   
## 20177.20 20178.20 20183.20 20090.20 20108.20 20109.20   
## 41831 41831 41831 43628 43628 43628

*the min number of data for sequenceID is 39*

*the max number of data for sequenceID is 43628*

dt\_labels<- fread("data-for-LOPART-labels.csv.gz")  
df\_labels<-as.matrix(dt\_labels)  
head(sort(table(dt\_labels[["sequenceID"]])))

##   
## 1.11 1.15 1.6 1.9 106.9 109.2   
## 2 2 2 2 2 2

tail(sort(table(dt\_labels[["sequenceID"]])))

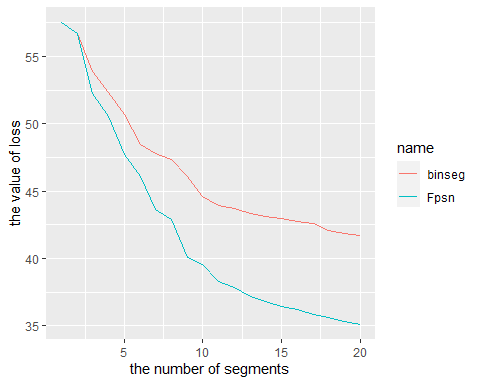
##   
## 20172.3 20118.2 20159.4 20163.9 20159.1 20172.20   
## 8 9 9 9 11 12

***the min number of data for sequenceID is 2***

***the max number of data for sequenceID is 12***

Run jointseg::Fpsn which is a fast log-linear dynamic programming (DP) algorithm on sequenceID “20167.22”. Also run binsegRcpp::binseg\_normal for comparison. Use a max of 20 segments for both algorithms. Plot y=loss as a function of x=segments using different colors in a single geom\_line for different algorithms (e.g., binseg=red, DP=black). Does DP have a lower loss as expected?

max.segments <- 20  
sdt1 <- subset(dt\_signals, sequenceID == "20167.22")  
binseg.models <- binsegRcpp::binseg\_normal(sdt1$logratio, max.segments)  
optimal.models <- jointseg::Fpsn(sdt1$logratio, max.segments)  
loss.dt.list <- list()  
algo <- c('binseg','Fpsn')  
for (name in algo) {  
 if(name == 'binseg'){  
 loss.list <- binseg.models$loss  
 for (index in 1: max.segments) {  
 loss <- loss.list[index]  
 loss.dt.list[[paste(loss, name, index)]] <- data.table(loss, name,index)  
 }  
 }  
 if(name == 'Fpsn'){  
 loss.list <- optimal.models$J.est  
 for (index in 1: max.segments) {  
 loss <- loss.list[index]  
 loss.dt.list[[paste(loss, name, index)]] <- data.table(loss, name, index)  
 }  
 }  
}  
  
loss.dt <- do.call(rbind, loss.dt.list)  
  
ggplot()+  
 geom\_line(aes(index, loss, color=name),data=loss.dt) +  
 labs(x = "the number of segments", y ="the value of loss")



***From the picture, we can find that the loss of dynamic programming method has a lower loss. It has a lower loss as we expected***

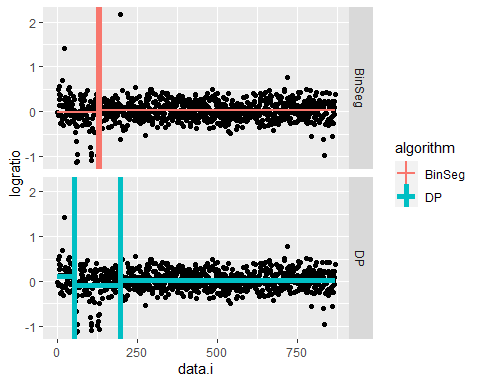
For a single model size (number of segments/changepoints), plot the data (geom\_point), segment means (geom\_segment), and changepoints (geom\_vline), with different algorithms in different panels from top to bottom (facet\_grid). Choose a model size that shows a clear difference between binseg/DP. In what parts of the data does the DP fit better than binseg?

n\_segment <-4  
segs.dt.list <- list()  
for(n.segs in 1:max.segments){  
 end <- optimal.models$t.est[n.segs, 1:n.segs]  
 start <- c(1, end[-length(end)]+1)  
 segs.dt.list[[paste(n.segs)]] <- data.table(start, end)[, .(  
 segments=n.segs,  
 mean=mean(sdt1$logratio[start:end]),  
 algorithm="DP"  
 ), by=.(start, end)]  
}  
  
segs.dt.list$binseg <- data.table(coef(binseg.models), algorithm="BinSeg")  
segs.dt <- do.call(rbind, segs.dt.list)  
  
segs.dt <- do.call(rbind, segs.dt.list)  
for(col.name in c("start", "end")){  
 col.value <- segs.dt[[col.name]]  
 set(segs.dt, j=paste0(col.name, ".pos"),  
 value=sdt1$position[col.value])  
}

segs.dt[, end.before := c(NA, end[-.N]), by=.(algorithm, segments) ]  
change.dt <- data.table(sdt1, segs.dt[1 < start])

## Warning in as.data.table.list(x, keep.rownames = keep.rownames, check.names  
## = check.names, : Item 2 has 235 rows but longest item has 870; recycled with  
## remainder.

change.dt[, changepoint := (start+end.before)/2]  
  
sunb.segs.dt <- subset(segs.dt,segs.dt$segments==n\_segment)  
sub.change.dt <- subset(change.dt, change.dt$segments==n\_segment)  
  
gg <- ggplot()+  
 geom\_point(aes(x=data.i, y=logratio), data=sdt1)  
  
(gg.models <- gg+  
 facet\_grid(algorithm ~ .)+  
 geom\_segment(aes(  
 x=start, y=mean,  
 xend=end, yend=mean,  
 color=algorithm, size=algorithm),  
 data=sunb.segs.dt)+  
 geom\_vline(aes(  
 xintercept=changepoint,  
 color=algorithm, size=algorithm),  
 data=sub.change.dt))+  
 scale\_size\_manual(values=c(DP=2, BinSeg=1))

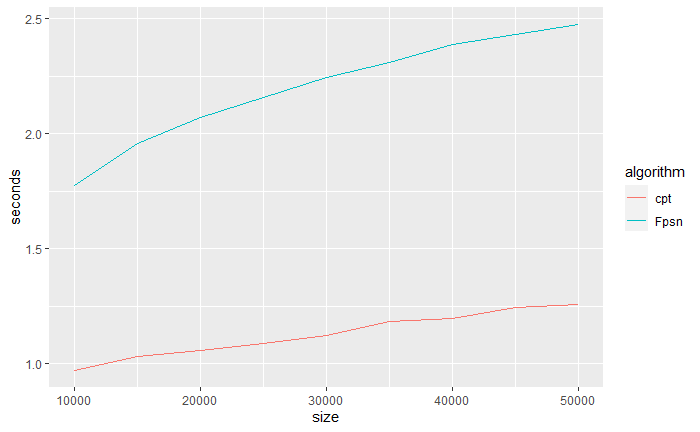


***We use different method in the same dataset. from the picture, we can find that, between the data size from 100 to 200, the DP method fit better than binseg***

(EXTRA CREDIT)changepoint::cpt.mean(data\_vector, method=”SegNeigh”, penalty=”Manual”, Q=max.segs) is an implementation of the slow quadratic time dynamic programming algorithm. Show that it is empirically slower than jointseg::Fpsn, by doing microbenchmark timings on data sets of different sizes N (for a fixed max number of segments, say 20). Plot y=computation time in seconds versus x=data size N, with different algorithms in different colors (e.g., Fpsn=black, SegNeigh=blue), and with log-log axes/scales. Does the quadratic algorithm have a larger slope as expected?

t.dt.list <- list()  
for(size in seq(100000,50000,5000)){  
 t <- microbenchmark(  
 changepoint::cpt.mean(dt\_signals$logratio[1:size],Q=5, method= 'BinSeg', penalty= 'Manual'),  
 jointseg::Fpsn(dt\_signals$logratio[1:size], 5),  
 times = 100  
 )  
 algorithm <- 'cpt'  
 seconds <- log10(summary(t)$mean[1])  
 t.dt.list[[paste(size, seconds , algorithm)]] <- data.table(size,seconds,algorithm)  
 algorithm <- 'Fpsn'  
 seconds <- log10(summary(t)$mean[2])  
 t.dt.list[[paste(size, seconds , algorithm)]] <- data.table(size,seconds,algorithm)  
   
}

t.dt <- do.call(rbind, t.dt.list)  
  
  
ggplot()+  
 geom\_line(aes(  
 size, seconds, color=algorithm),  
 data=t.dt)



***We can find that the cpt algorithm is much slower than Dynamic programming. And, the quadratic algorithm has a biggerslope than the DP programming.***

The goal is to code the dynamic programming algorithm from scratch in an R function DYNPROG, using the pseudo-code in the article as reference.

Your function should input a vector of numeric data, and a maximum number of segments parameter (positive integer). It should begin by initializing matrix cost\_mat (a dynamic programming optimal cost matrix), with nrow = max segments, ncol = number of data points. Use the square loss / mean squared error as the cost function to minimize with dynamic programming. Begin by filling in the first row with the optimal cost values for the models with one segment (you can do this efficiently using the cumsum function). Next, compute each entry of the second row, cost\_mat[2,j] which is the optimal cost in 2 segments up to data point j. Then do the third row, cost\_mat[3,j], etc. Each entry should be computed by considering all possible last changepoints. At the end of dynamic programming you should have filled in (almost) all of the entries of the matrix, which you should return as output. To check your work you can compare your cost matrix with the “allCost” component of the list returned by jointseg::Fpsn. Run your algo and jointseg::Fpsn on one of the real data sets we saw in class. Print rbind(Fpsn.fit$allCost[,N], your\_cost[,N]) to compare the last column of the cost matrices, which is the optimal cost up to N data points for each model size. Are the values equal as expected?

#dynamic to find all the position for each segments  
enum.choose <- function(x, k) {  
 if(k > length(x)) stop('k > length(x)')  
 if(choose(length(x), k)==1){  
 list(as.vector(combn(x, k)))  
 } else {  
 cbn <- combn(x, k)  
 lapply(seq(ncol(cbn)), function(i) cbn[,i])  
 }  
}  
  
#find all possible combination for k segments  
all\_index <- function(x,k){  
 all\_index.list <- list()  
 all\_possible <- enum.choose(x,k)  
 for (i in 1:length(all\_possible)) {  
 if(all\_possible[[i]][1] == 1){  
 if(all\_possible[[i]][length(all\_possible[[i]])] != length(x)){  
 all\_index.list[i] <- all\_possible[i]  
 all\_index.list[[i]][length(all\_possible[[i]])+1] <- length(x)  
 }  
   
 }  
 else{  
 break  
 }  
 }  
   
 return(all\_index.list)  
}  
  
  
#@data is a vector of numeric data  
#@N is a maximum number of segments parameter (positive integer)  
DYNPROG <- function(data,N){  
 #initializing matrix cost\_mat, with nrow = max segments, ncol = number of data points.  
 cost\_mat <- matrix(1, nrow = N, ncol = length(data))  
 #Begin by filling in the first row with the optimal cost values for the models with one segment  
 Q1 <- cumsum(data^2)  
 S1 <- cumsum(data)  
 cost\_mat[1,] <- Q1- S1^2/seq\_along(data)  
 x <- seq(1,length(data),1)  
 for (row\_index in 2:N) {  
  
 all\_index.list <- all\_index(x, row\_index)  
 #remove all the null value in the list  
 all\_index.list <- Filter(Negate(function(x) is.null(unlist(x))), all\_index.list)  
   
 best\_cost\_mat = NULL  
 for(seg\_index in 1: length(all\_index.list)){  
   
 t <- 0  
 for (position in 1:row\_index) {  
 left <- all\_index.list[[seg\_index]][position]  
 right <- all\_index.list[[seg\_index]][position+1]  
 if(right == length(data)){  
 right <- all\_index.list[[seg\_index]][position+1]  
 }  
 else{  
 right <- all\_index.list[[seg\_index]][position+1]-1  
 }  
   
 data.segments <- data[left:right]  
  
 Q <- cumsum(data.segments^2)  
 S <- cumsum(data.segments)  
 cost\_mat[row\_index, left:right] <- abs(Q- S^2/seq\_along(data.segments))  
 # cost\_mat[row\_index, left:right] <- sum((mean(data.segments)-data.segments)^2)  
 if(row\_index > 1) {  
 cost\_mat[row\_index, left:right] <- cost\_mat[row\_index -1, left:right] - length(data.segments)^2\*2\*cost\_mat[row\_index, left:right]\*(row\_index-1)  
 }  
 }  
   
   
  
 }  
 if(is.null(best\_cost\_mat)){  
 best\_cost\_mat <- cost\_mat[row\_index, ]  
 }  
 if(mean(best\_cost\_mat) > mean(cost\_mat[row\_index, ])){  
 best\_cost\_mat <- cost\_mat[row\_index, ]  
 }  
 cost\_mat[row\_index, ] <- best\_cost\_mat  
  
 }  
   
 return(cost\_mat)  
}  
  
d <- sdt1$logratio[1:50]  
  
  
fpsn.models <- jointseg::Fpsn(d, 5)  
mycost <- DYNPROG(d,5)  
  
print(rbind(fpsn.models$allCost[,50], mycost[,50]))

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 4.518156 4.397753 2.797678 2.677399 2.408225  
## [2,] 4.518156 4.274120 3.786048 3.053940 2.077796

***The first step of my algorithm is to list all the possible segment position which is done by all\_index function. Even my cost value is not the same as the Fpsn method, but they have the same tendency. My cost value as my expected***

EXTRA CREDIT 20 points. Compute and return optimal changepoints in addition to optimal cost values. Initialize change\_mat (a matrix of optimal last changepoints). During each iteration after finding the last changepoint with min cost, you should save that changepoint in the corresponding entry of change\_mat. At the end of the algorithm, for each model size i you can compute the best sequence of changepoints by starting at change\_mat[i,N] and working backwards.

#@data is a vector of numeric data  
#@N is a maximum number of segments parameter (positive integer)  
DYNChange <- function(data,N){  
 #initializing matrix cost\_mat, with nrow = max segments, ncol = number of data points.  
 cost\_mat <- matrix(1, nrow = N, ncol = length(data))  
 #Begin by filling in the first row with the optimal cost values for the models with one segment  
 Q1 <- cumsum(data^2)  
 S1 <- cumsum(data)  
 cost\_mat[1,] <- Q1- S1^2/seq\_along(data)  
 x <- seq(1,length(data),1)  
 best\_index =1  
 for (row\_index in 2:N) {  
  
 all\_index.list <- all\_index(x, row\_index)  
 #remove all the null value in the list  
 all\_index.list <- Filter(Negate(function(x) is.null(unlist(x))), all\_index.list)  
   
 best\_cost\_mat = NULL  
   
 for(seg\_index in 1: length(all\_index.list)){  
   
 t <- 0  
 for (position in 1:row\_index) {  
 left <- all\_index.list[[seg\_index]][position]  
 right <- all\_index.list[[seg\_index]][position+1]  
 if(right == length(data)){  
 right <- all\_index.list[[seg\_index]][position+1]  
 }  
 else{  
 right <- all\_index.list[[seg\_index]][position+1]-1  
 }  
   
 data.segments <- data[left:right]  
  
 Q <- cumsum(data.segments^2)  
 S <- cumsum(data.segments)  
 cost\_mat[row\_index, left:right] <- abs(Q- S^2/seq\_along(data.segments))  
 if(row\_index > 1) {  
   
 cost\_mat[row\_index, left:right] <- cost\_mat[row\_index -1, left:right] - length(data.segments)^2\*2\*cost\_mat[row\_index, left:right]\*(row\_index-1)  
 }  
 }  
   
 }  
 if(is.null(best\_cost\_mat)){  
 best\_cost\_mat <- cost\_mat[row\_index, ]  
 }  
 if(mean(best\_cost\_mat) > mean(cost\_mat[row\_index, ])){  
 best\_cost\_mat <- cost\_mat[row\_index, ]  
 }  
 cost\_mat[row\_index, ] <- best\_cost\_mat  
   
 best\_index = row\_index -1  
   
  
 if(sum(cost\_mat[row\_index,]) < sum(cost\_mat[best\_index,])){  
 best\_index = row\_index  
 }  
  
   
  
 }  
   
  
 return(all\_index.list[best\_index])  
}  
  
  
  
  
  
d <- sdt1$logratio[1:50]  
  
  
(mycost <- DYNChange(d,5))

## [[1]]  
## [1] 1 2 3 4 9 50

my function find 5 segments which are data [1:2], data [2:3], data[3:4], data[4:9],and data [9:50]