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

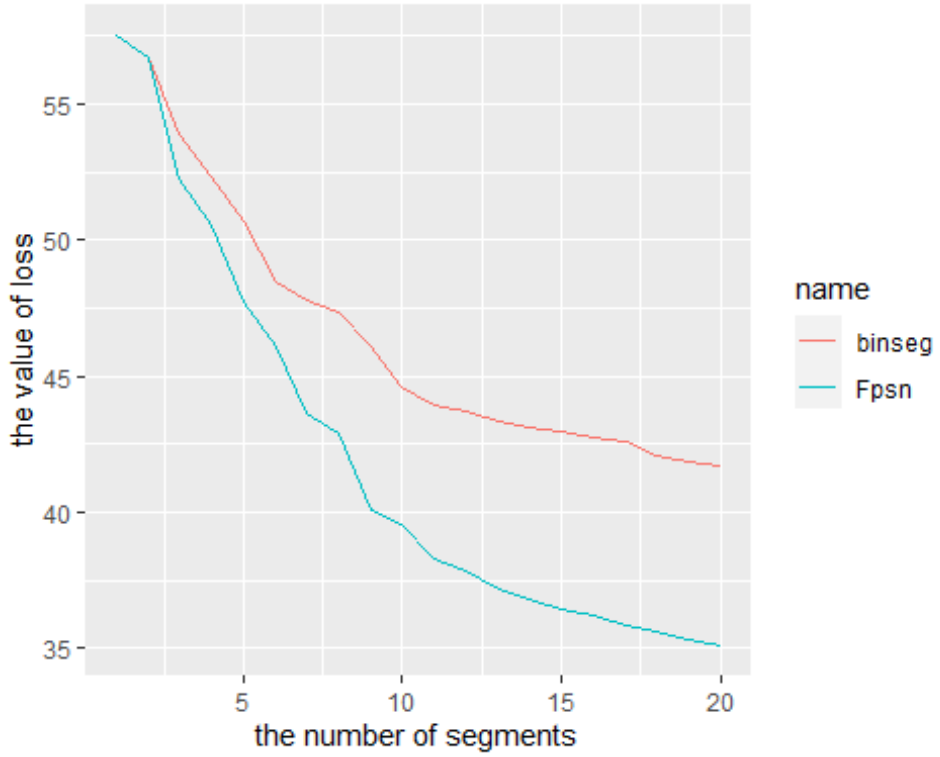
```
dt_labels<- fread("data-for-LOPART-labels.csv.gz")
df_labels<-as.matrix(dt_labels)
head(sort(table(dt_labels[["sequenceID"]])))

##
##  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$loss
      for (index in 1: max.segments) {
        loss <- loss.list[index]
        loss.dt.list[[paste(loss, name, index)]] <- data.table(loss, name,ind
ex)
      }
   }
  if(name == 'Fpsn'){
    loss.list <- optimal.models$J.est
    for (index in 1: max.segments) {
      loss <- loss.list[index]
      loss.dt.list[[paste(loss, name, index)]] <- data.table(loss, name, inde
x)
    }
  }
}

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")
```

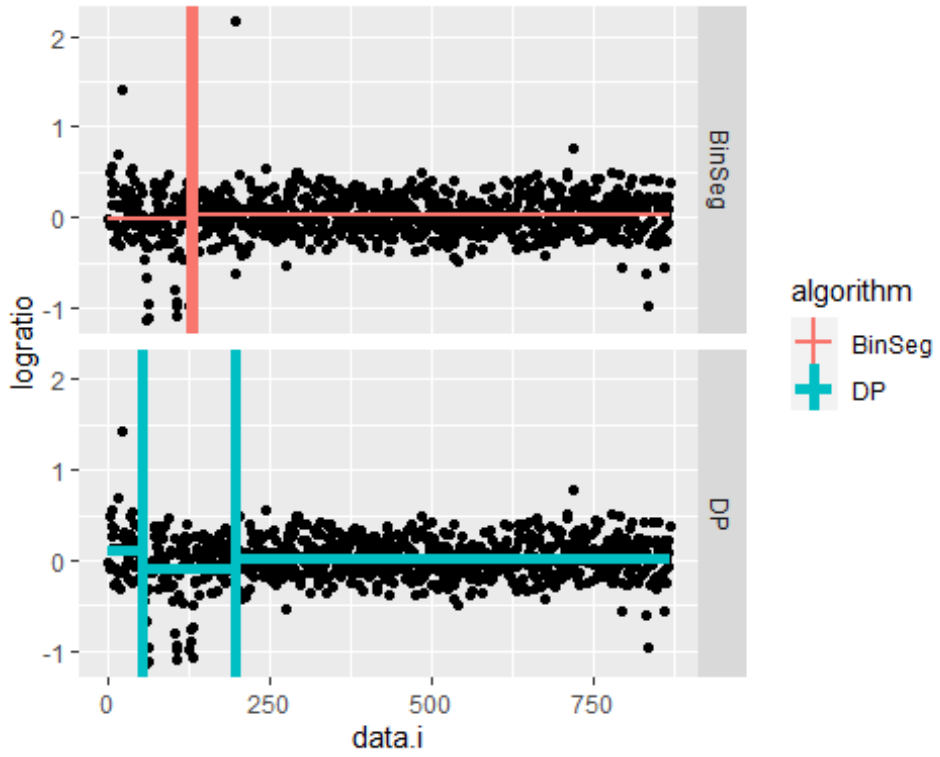


From the picture, we can find that the loss of dynamic programming method has a lower loss. It has a lower loss as we expected

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.name
s
## = check.names, : Item 2 has 235 rows but longest item has 870; recycled wi
th
## 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))
```

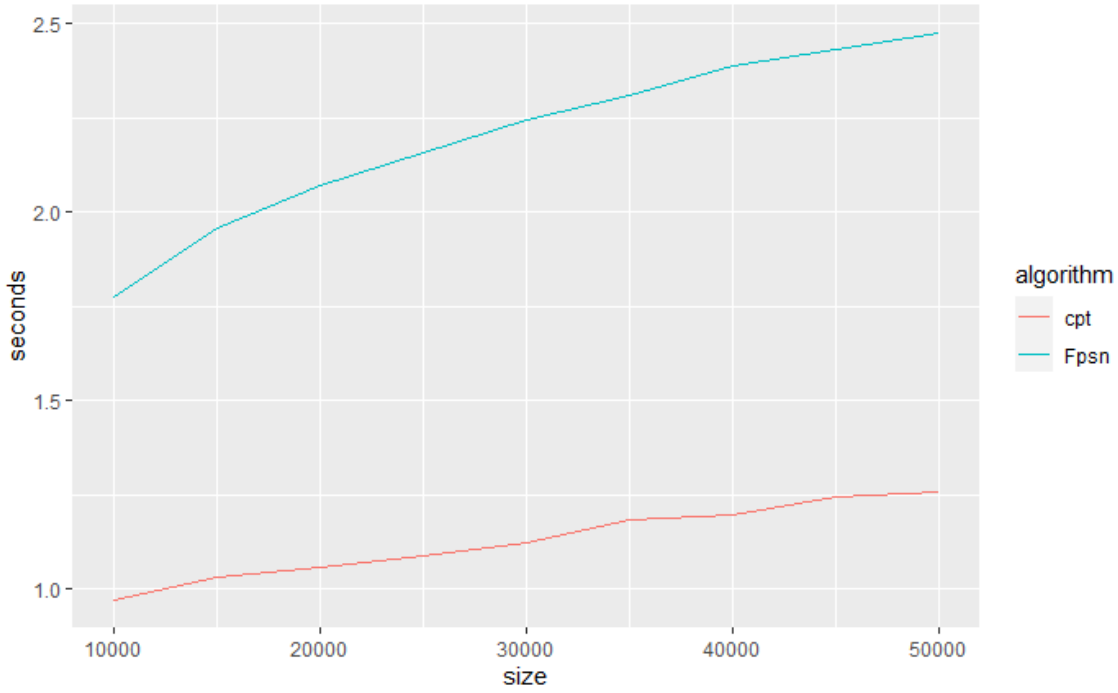


We use different method in the same dataset. from the picture, we can find that, between the data size from 100 to 200, the DP method fit better than binseg

(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$logratio[1:size],Q=5, meth
od= 'BinSeg', penalty= 'Manual'),
                    jointseg::Fpsn(dt_signals$logratio[1:size], 5),
                    times = 100
             )
     algorithm <- 'cpt'
     seconds <- log10(summary(t)$mean[1])
     t.dt.list[[paste(size, seconds , algorithm)]] <- data.table(size,second
s,algorithm)
     algorithm <- 'Fpsn'
     seconds <- log10(summary(t)$mean[2])
     t.dt.list[[paste(size, seconds , algorithm)]] <- data.table(size,second
```

```
s,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 d
ata points.
  cost_mat <- matrix(1, nrow = N, ncol = length(data))
  #Begin by filling in the first row with the optimal cost values for the mod
els 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_inde
x.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.seg
ments))
             # cost_mat[row_index, left:right] <- sum((mean(data.segments)-da
ta.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 d
ata points.
  cost_mat <- matrix(1, nrow = N, ncol = length(data))
  #Begin by filling in the first row with the optimal cost values for the mod
els 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_inde
x.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.seg
ments))
           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])
}
```

```r
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]