Homework8

Jun Rao

10/7/2020

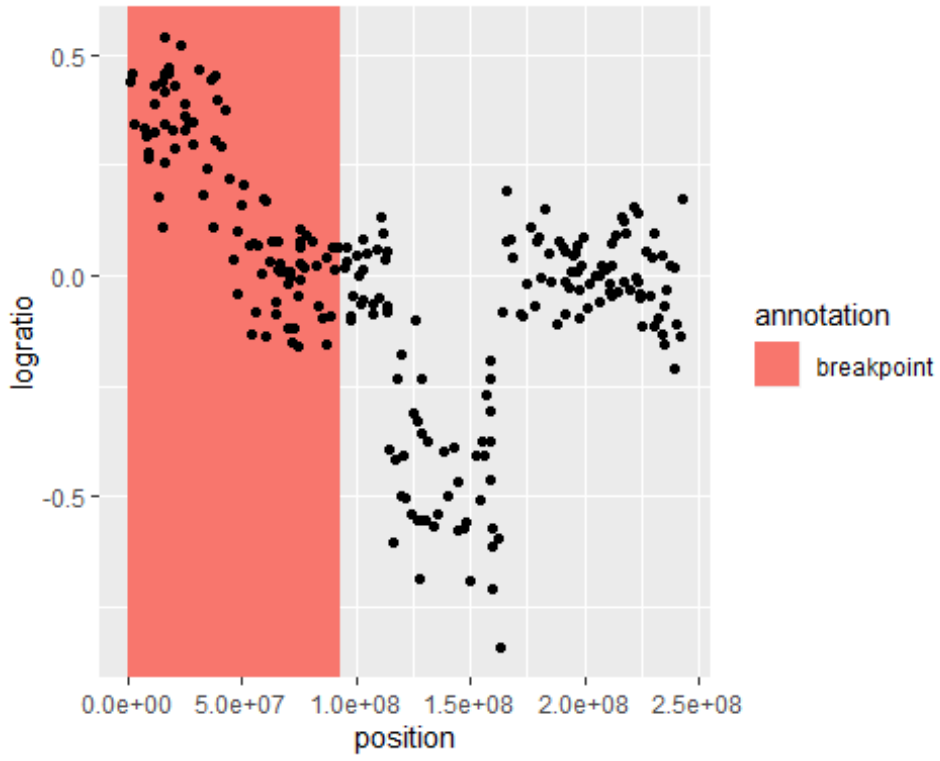
Homework 8: Binary segmentation

The goal of this homework is to compute and plot binary segmentation models for sequence data, and compare the speed of two R packages.

```
library(data.table)
library(ggplot2)
library(binsegRcpp)
library(changepoint)
```

1.install R package neuroblastoma and load the data set via data(neuroblastoma, package="neuroblastoma"). Compute just the subset of data on chromosome=2, profile.id=4. Plot the logratio as a function of position. How many data points are there on this sequence?

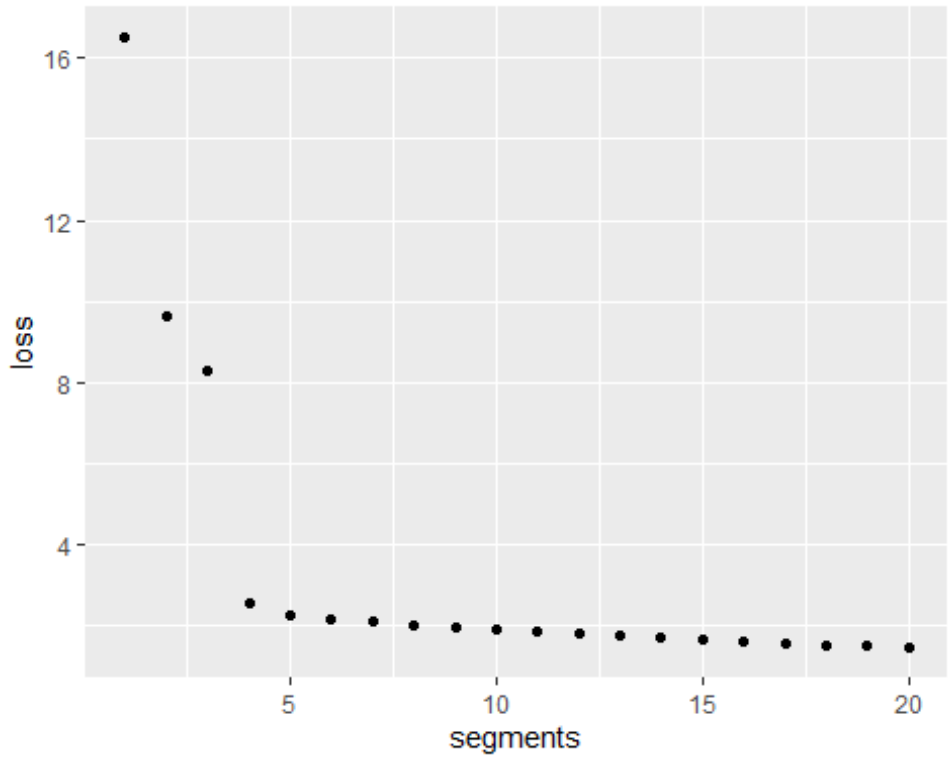
```
data(neuroblastoma, package="neuroblastoma")
data.list <- list()
one.seq.list <- list()
select.dt <- data.table(profile.id="4", chromosome="2")



for(data.name in names(neuroblastoma)){
  all.data.dt <- data.table(neuroblastoma[[data.name]])
  data.list[[data.name]] <- all.data.dt
  one.seq.list[[data.name]] <- all.data.dt[select.dt, on=names(select.dt)]
}

ggplot()+
  geom_rect(aes(
    xmin=min, xmax=max, ymin=-Inf, ymax=Inf, fill=annotation),
    data=one.seq.list$annotations)+
  geom_point(aes(x=position, y=logratio), data=one.seq.list$profiles)
```



We can find there are two different datasets

2.Use binsegRcpp::binseg_normal(logratio_vector, Kmax) to compute binary segmentation models with a Kmax of 20 segments for this data set. Use the plot method to show a plot of loss values versus model size. Does the loss always decrease as expected? After what model size is there a "kink" or inflection point past which it decreases more slowly? (that is the "good" model size which you should select)

```
pro.dt <- data.table(neuroblastoma[["profiles"]])
one.sequence <- pro.dt[profile.id=="4" & chromosome=="2"]
models <- binseg_normal(one.sequence$logratio, 20)
ggplot()+
  geom_point(aes(
    segments, loss),
    data=models)
```

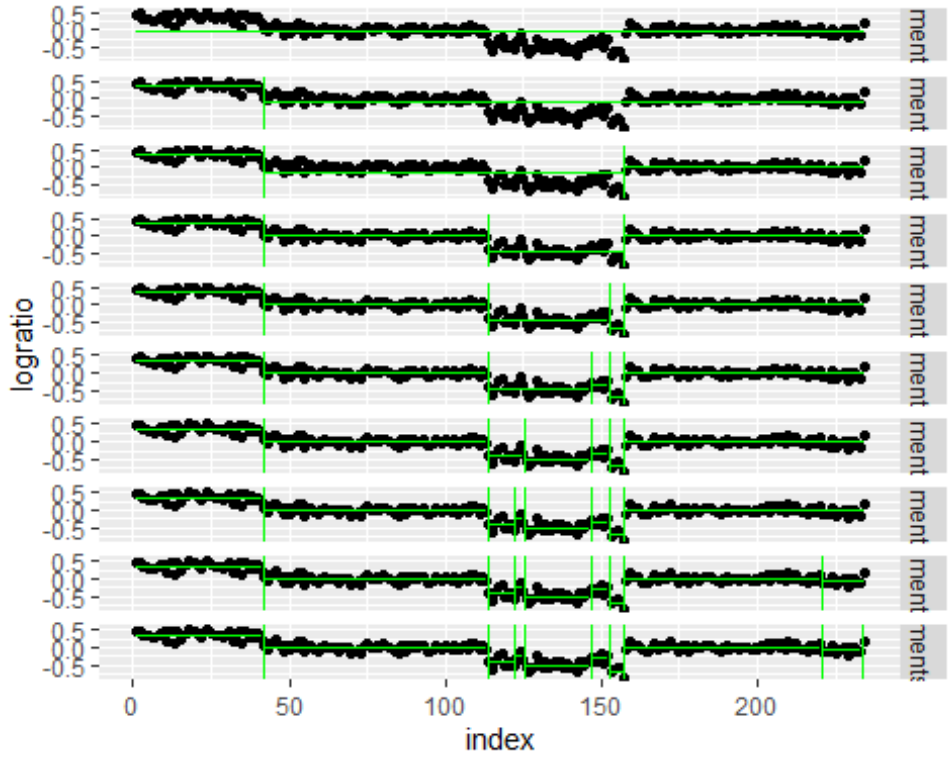


We can find that the loss not always decrees as we expected.

After segments equal 4, it decreases more slowly.

3.Make a ggplot of your selected model on top of the data set. For simplicity instead of x=position (as in problem 1), use x=seq_along(logratio) (which is just 1 to N, same scale as the numbers returned by binseg_normal). Use geom_segment to show segment means and geom_vline to show changepoints. Does the model have any obvious false positives (changepoints predicted where the data have no significant changes) or false negatives (no change predicted where the data has a significant change)?

```
segs.dt <- coef(models)
sec.seq <- one.sequence[, index := seq_along(logratio)]
ggplot()+
  geom_point(aes(
    x=index, y=logratio),
    data=sec.seq)+
  geom_segment(aes(
    x=start, y=mean,
    xend=end, yend=mean),
    color="green",
    data=segs.dt)+
  facet_grid(segments ~ ., labeller=label_both)+
  geom_vline(aes(
    xintercept=start-0.5),
    color="green",
    data=segs.dt[start>1])
```



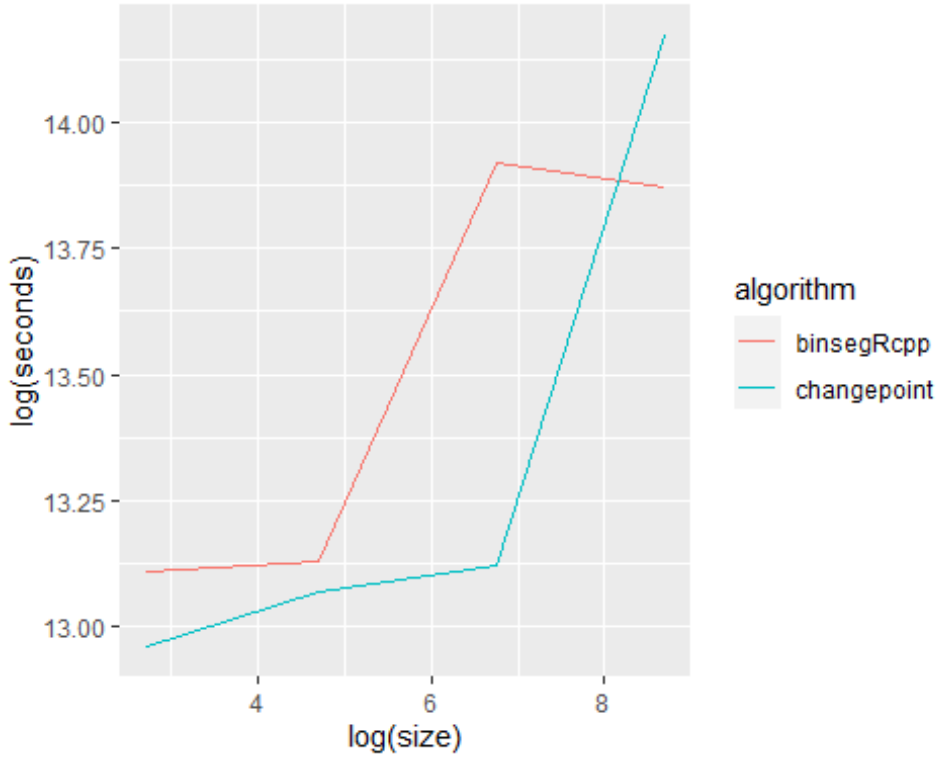
The data at here has a significant change, which has a false negative

Use changepoint::cpt.mean(logratio_vector,Q=100, method="BinSeg", penalty="Manual")to compute the same binary segmentation model, up to 100 segments, on a wide range of data sequences (each data sequence is a unique combination of profile.id/chromosome values). Use microbenchmark to compute timings of binsegRcpp with 100 segments as well. Make a ggplot with log-log axes that compares the computation times of the two algorithms. (x=data size, y=seconds, color=package) Are there any significant speed differences?

```
count.dt <- data.list$profiles[, .(count=.N), by=.(profile.id, chromosome)]
desired <- data.table(
  count=count.dt[, 10^seq(log10(min(count)), log10(max(count)), l=5)])

category.dt <- count.dt[
  desired, .(profile.id, chromosome, actual=x.count, desired=i.count),
  roll="nearest", on=.(count)]

my.all.data <- data.table(neuroblastoma[[1]])

seq.list <- list()
result.dt.list <- list()

x <- category.dt$actual


for (N in 1:nrow(category.dt)) {
```

```
    #print(as.character(category.dt[N]$chromosome))
    my.select.dt <- data.table(profile.id=as.character(category.dt[N]$profile
.id), chromosome=as.character(category.dt[N]$chromosome))

    seq.list[[N]] <- my.all.data[my.select.dt,on=names(my.select.dt)]

}




for(N in 2:nrow(category.dt)){
    algorithm <- 'binsegRcpp'
    d <- seq.list[[N]]$logratio
    size <- x[[N]]
    if(size < 100){
      t <-   microbenchmark(binseg_normal(d, size))
    }
    else{
      t <-   microbenchmark(binseg_normal(d, 100))
    }
    seconds <- median(t$time)

    result.dt.list[[paste(size, seconds , algorithm)]] <- data.table(size,sec
onds, algorithm)
}


for(N in 2:nrow(category.dt)){
    algorithm<- 'changepoint'
    d <- seq.list[[N]]$logratio
    size <- x[[N]]
    if(size < 100){
      t <-  microbenchmark(cpt.mean(d,100, method= 'BinSeg', penalty= 'Manual
'))
    }
    else{
      t <-  microbenchmark(cpt.mean(d,100, method= 'BinSeg', penalty= 'Manual
'))
    }

    seconds <- median(t$time)

    result.dt.list[[paste(size,seconds, algorithm)]] <- data.table(size,secon
ds,algorithm )
}


result.dt <- do.call(rbind, result.dt.list)
```

```
ggplot()+
  geom_line(aes(
    log(size), log(seconds), color=algorithm),
    data=result.dt)
```



Yes. It has a significant speed.

Based on the pseudocode in the article, code binary segmentation for the square loss. Your R function should be named BINSEG and input (1) a vector of numeric data, and (2) the maximum number of segments. During each iteration you should consider each possible split, compute the amount that the loss would decrease for each split, then select the split which decreases the loss the most. It should output a vector of square loss values, from 1 to the maximum number of segments (for simplicity, do not output the changepoints). Run your algorithm on the same data set as in problems 1-3. Make another plot of loss versus model size, using color for different implementations (e.g. binsegRcpp=red, yours=black). Does your code compute the same loss values as binsegRcpp?

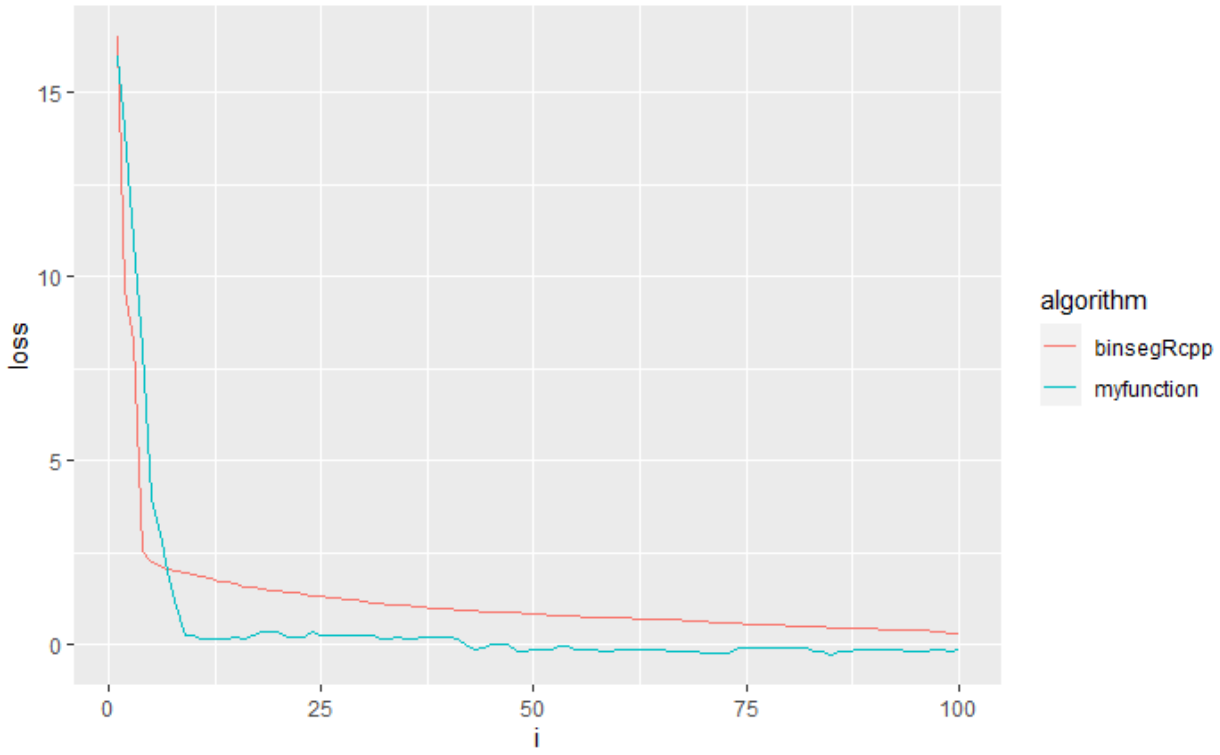
```
#' this function creates and returns all the CUSUM test scores in a given int
erval
#'
#' Accepts: data, lower open interval bound, upper open interval bound
#'
#' Returns: a vector containing all CUSUM values for all points in the interv
al
```

```r
bs_stat <- function(data, lower, upper){

  test.stat <- NULL
  cx <- cumsum(data)
  for (i in (lower+1):(upper-1)){

    #for each k in the interval we create a test score,
    #|bs_stat|

    coeff <- (i - lower)/(upper - lower)
    max_difference <- (cx[upper]-cx[lower])
    const <- coeff*max_difference

    top_fraction <- const - (cx[i]-cx[lower])
    bottom_fraction <- sqrt((i - lower)*(1-(coeff)))
    test.stat <- c(test.stat, abs((top_fraction)/(bottom_fraction)))

  }
  return(test.stat)
}




#' bs.model converts the estimated change-point locations into the estimated
signal
#'
#' Accepts: observations, intervals to partition observations
#'
#' Returns: a list of est.signal and est.means


bs.model <- function(data, interval){


  est.signal <- NULL
  means <- NULL

  for(i in 1:(length(interval)-1)){
    a <- interval[i]
    b <- interval[i+1]

    est.signal <- c(est.signal, rep(mean(data[a:b]), (b-a)))

    means <- c(means, mean(data[a:b]))
```

```
    }
  est.signal <- c(est.signal[1],est.signal)
  return(list(est.signal=est.signal, est.means=means))
}


#' bs.helper will perform one iteration of the binary segmentation algorithm
#'
#' Accepts: data (observations), former.segments (intervals to be tested), bo
undary (critical value used in testing)
#'
#' Returns: finsal.segments (intervals to test in next iteration)



bs.helper <- function(data, former.segments = NULL,segments){

  changepoints <- NULL
  est.changepoints <- NULL
  finsal.segments <- NULL


  if(is.null(former.segments)){
    current.segment <- c(1, segments)
    finsal.segments <- c(1,segments)
  } else {
      current.segment <- sort(former.segments)
      finsal.segments <- former.segments
  }



  #we need to order the estimated change-points so that the largest bs_stat i
s first
  for (i in 1:(length(current.segment)-1)){ # n changepoints gives n-1 segmen
ts
    #we take the maximum as a estimated change point
    # the zero removes the error message, but has no affect on the output of
the function
    if(max(c(bs_stat(data, current.segment[i],current.segment[i+1]), 0), na.r
m = TRUE)>= 0.05){
      ch <- which.max(bs_stat(data, current.segment[i],current.segment[i+1]))
+current.segment[i]
      #we test if the changepoinit is already accounted for
     if(!( ch %in% former.segments)){

       est.changepoints <- c(est.changepoints, ch)
       changepoints <- c(changepoints, max(bs_stat(data, current.segment[i],c
urrent.segment[i+1]), na.rm = TRUE))
```

```
    }
   }
  }

  if(!(is.null(est.changepoints))){

    bound <- cbind(est.changepoints, changepoints)

    est.changepoints <- bound[order(-bound[,2]), 1]

  }
  finsal.segments <- c(finsal.segments, est.changepoints)
  return(unname(finsal.segments))
}

#
# the function will apply binary segmentation to the supplied data.
# the function will use stored values for data with length less than 5000.
#
# inputs   {datae data, significant value for the CUSUM}
#         @ data, boundary
#
# returns  {estimated signal, estimated means, estimated changepoints}
#         @ $est.signal, $est.means, $est.changepoints
#
#

BINSEG <- function(data,segments){


  held_segments <- 0
  former.segments <- NULL
  est.signal <- NULL


  former.segments <- bs.helper(data,NULL,segments)

  while(!setequal(held_segments, former.segments)){
    # binary segmentation runs until the operation doesn't produce a new chan
ge-point
    held_segments <- former.segments
    former.segments <- bs.helper(data, held_segments,segments)

  }

  #print("in order of occurence the estimated changepoints are ")
  ordered_finsal.segments <- sort(former.segments)

  print(ordered_finsal.segments)
```

```
  output <- bs.model(data, ordered_finsal.segments)
  interval <- list(est.changepoints = ordered_finsal.segments)
  output <- c(output, interval)

  return(output)
}



result<-BINSEG(one.sequence$logratio,100)

loss.list <- list()



for (i in 1:length(result$est.mean)) {
  algorithm <- 'myfunction'
  loss <-  result$est.mean[i] - mean(result$est.mean)
   loss.list[[paste(i,loss, algorithm)]] <- data.table(i,loss,algorithm )
}



temp <- binseg_normal(one.sequence$logratio, 100)
for (i in 1:length(temp$loss)) {
  algorithm <- 'binsegRcpp'
  loss <- temp$loss[i]
   loss.list[[paste(i,loss, algorithm)]] <- data.table(i,loss,algorithm )
}

loss.dt <- do.call(rbind, loss.list)

ggplot()+
  geom_line(aes(
    i, loss, color=algorithm),
    data=loss.dt)
```



It looks like my function doesn't have the same loss as the package function.