User Guide for 'offlineChange' R package

Jiahuan Ye, Jie Ding

July 26, 2019

Contents

This vignette serves as a quick guide on how to use the 'offlineChange' R package to detect multiple change points from offline data, based on the multi-window approach proposed by Ding et al.

Detect change points of dependent data without prior knowledge on the ranges of changes or the number of change

Suppose there is a sequence of one-dimentional dependent data, and the true number of change points is unknown, then the following function can be used to return the ranges that are most likely to contain the true change points. A list of window sizes (window_list) is specified by the user. Each window size should be less than the number of observations. The purpose of these windows, as explained in the reference paper, are used to divide the original data into segments so that the data can be transformed. The largest number of changes (point_max) should be specified by the user as well.

```
library(offlineChange)
# Data
N <- 1000
N1 <- floor(0.1*N)
N2 <- floor(0.3*N)
a1 <- c(0.8, -0.3); c1 <- 0
a2 <- c(-0.5, 0.1); c2 <- 0
a3 <- c(0.5, -0.5); c3 <- 0
y <- rep(0,N)
L<-2
y[1:L] <- rnorm(L)
for (n in (L+1):N){
  if (n <= N1) {
    y[n] <- y[(n-1):(n-L)] %*% a1 + c1 + rnorm(1)
  } else if (n <= (N1+N2)) {
    y[n] <- y[(n-1):(n-L)] %*% a2 + c2 + rnorm(1)
  }
  else {
    y[n] <- y[(n-1):(n-L)] %*% a3 + c3 + rnorm(1)
  }
}
result <- MultiWindow(y,
                      window_list = c(100,50,20,10,5),
                      point_max   = 5)
```

The returned results contain the number of peak ranges and location of peak ranges:

```
result$n_peak_range
result$peak_range
```

The users can also modify some other default arguments:

```
result <- MultiWindow(y,
                      window_list = c(100, 50, 20, 10, 5),
                      point_max   = 5,
                      prior_range = NULL,
                      get_mle     = GetMle,
                      penalty     = c("bic","aic"),
                      seg_min     = 1,
                      num_init    = NULL,
                      tolerance   = 1,
                      cpp         = TRUE,
                      ret_score   = FALSE
                      )
```

The argument *cpp* indicates whether to increase computation speed using rcpp. The acceleration only works without prior range. Based on the peak ranges returned, users can use the following function to get exact change points:

```
result <- MultiWindow(y,
                      window_list = c(100,50,20,10,5),
                      point_max   = 5)

RangeToPoint(y,
             n_peak_range = result$n_peak_range,
             peak_range   = result$peak_range)
```

Based on the peak ranges returned, users can also plot them on original data. Blue line and red line represent the start and end of a peak range.

```
result <- MultiWindow(y,
                      window_list = c(100,50,20,10,5),
                      point_max   = 5)

ChangePointsPlot(y,result,main="plot of change points")
```

Or, users can plot the score that represents how likely the range contains change points. The higher score a range has, the more likely the range contains change point. The argument *ret_score* must be TRUE inorder to get score plot.

```
result <- MultiWindow(y,
                      window_list = c(100,50,20,10,5),
                      point_max   = 5,
                      ret_score=TRUE)

ScorePlot(result, main="score plot")
```

Define your own transformation function with argument get_mle

The default transformation method from dependent data to independent data is OLS estimation of AR coefficients. Users can define their own transformation functions.

```r
install.packages('MHadaptive')
library('MHadaptive')
gen_EFdata <- function(N){
  q <- 4
  Ns <- c(floor(0.2*N), floor(0.3*N), N-floor(0.2*N)-floor(0.3*N))
  thetas <- c(0.2,0.6,1)
  X <- c()
  for (k in 1:3){
    loglik <- function( x ){
      res <- - thetas[k] * abs(x)^q
    }
    mh <- Metro_Hastings(loglik,
                         rgamma(n = 1, shape = 3, rate = 1),
                         prop_sigma = NULL,
                         par_names = NULL,
                         iterations = 11000,
                         burn_in = 1000,
                         adapt_par = c(100, 20, 0.5, 0.75),
                         quiet = TRUE)
    mh0 <-  mcmc_thin(mh, thin = 20)
    X <- c(X, mh0$trace[1:Ns[k]])
  }
  return(X)
}
GetHle=function(x,window_size) {
  q <- 4
  N <- length(x)
  n_window <- ceiling(N/window_size)
  x_transformed <- rep(0,n_window)
  for (n in 1:n_window) {
    #get estimated coefficients
    xx <- x[(1+(n-1)*window_size):min(n*window_size,N)]
    x_transformed[n] <- q * (q-1) * sum(abs(xx)^(q-2)) / sum(q^2 * abs(xx)^(2*q-2))
  }
  return(x_transformed)
}

N <- 1000
X <- gen_EFdata(N)
result <- MultiWindow(X,
                      window_list=c(100,80,50,30, 20),
                      get_mle=GetHle,
                      point_max=3,
                      seg_min=1,
                      tolerance=1)
```

Define your own penalty term with argument penalty

Besides transformation function, users can also define peanlty terms they like. Be careful that, the penalty function is actually the sum of loss and penalty.

```r
toy_penatly=function(num_each, wgss, D, K, N) {
  penalty <- sum(num_each * log(wgss/num_each)) + 2 * D * K * log(N)
```

```
  return(penalty)
}
result <- MultiWindow(y,
                      window_list=c(100,80,50,30, 20),
                      penalty="toy_penalty",
                      point_max=3,
                      seg_min=1,
                      tolerance=1)
```

Detect change points of dependent data with prior change ranges

Suppose there is a sequence of one-dimentional dependent data, and the user already know prior ranges that change points should be within. Then the following function can be used to return the (narrower) ranges that are most likely to contain the true change points, with much less computational cost compared with the previous method.

```
result <- MultiWindow(y,
                      window_list = c(100,50,20,10,5),
                      prior_range = list(c(30,200),c(220,400)))
```

And the users can also change some default arguments. In particular, 'get_mle' is the data transformation method that can be provided by users. Typically, it is chosen to be the function that input data and output an estimate of unknown parameters assuming that the data is generated from a parametric family.

```
result <- MultiWindow(y,
                      window_list = c(100, 50, 20, 10, 5),
                      prior_range = list(c(30,200), c(220,400)),
                      get_mle     = GetMle,
                      tolerance   = 1)
```

Based on the peak ranges returned, users can use the following function to get exact change points:

```
result <- MultiWindow(y,
                      window_list = c(100,50,20,10,5),
                      prior_range = list(c(30,200), c(220,400)))

RangeToPoint(y,
             n_peak_range = result$n_peak_range,
             peak_range   = result$peak_range)
```

Detect change points of independent data without prior ranges and true number of change points

Suppose there is a sequence of independent data with dimension L+1, and the true number of change points is unknown, then the following function can be used to return the change points. Like before, the largest number of changes (point_max) should be specified by the user.

```
# Data
a <- matrix(rnorm(40,mean=-1,sd=1), nrow=20, ncol=2)
b <- matrix(rnorm(120,mean=0,sd=1), nrow=60, ncol=2)
c <- matrix(rnorm(40,mean=1,sd=1), nrow=20, ncol=2)
x <- rbind(a,b,c)
result <- ChangePoints(x, point_max = 5)
```

And the users can also modify some default arguments:

```
result <- ChangePoints(x,
                       point_max = 5,
                       penalty   = c("bic","aic"),
                       seg_min   = 1,
                       num_init  = NULL,
                       cpp       = TRUE)
```

Detect change points of independent data without prior ranges but with true number of change points

Suppose there is a sequence of independent multivariate data, and the true number of change points is already known, then the following function can be used to return the change points. Here, K is the number of change points instead of the number of segments.

```
result <- OrderKmeans(x, K = 2)
```

And the users can also change the suggested setting of arguments:

```
result <- OrderKmeans(x, K = 2, num_init=NULL)
```

Detect change points of independent data with prior change ranges

Suppose there is a sequence of independent multivariate data, and the user already know prior ranges that change points should be within, then the following function can be used to return change points, and the prior knowledge of ranges of change points can make the cumputation more efficient and the return change points more accurate.

```
l1 <- c(15,25)
l2 <- c(75,100)
prior_range_x <- list(l1, l2)
result <- PriorRangeOrderKmeans(x, prior_range_x = list(l1,l2))
```

And the users can also modify some default arguments:

```
result <- PriorRangeOrderKmeans(x, prior_range_x, num_init=NULL)
```