Continuous HMM for profiling copy number alterations

GENOME 541: Cancer Genomics - Probabilistic Methods for Profiling Copy Number Alterations from Sequencing Data (WGS) University of Washington Summer 2020

Lavinia Carabet

8/28/2020

Implementation of components of a copy number alteration (CNA) caller using a single-sample continuous Hidden Markov Model (HMM), a generative model in a Bayesian framework

HMMs - probabilistic graphical model used to predict a sequence of hidden (copy number) states from a set of observed variables (log2ratio normalized copy number data obtained from sequencing read coverage)

The input data `log2ratios_chr1.txt` contains a set of 411 genomic wwindows/bins in chr1 with normalized log2 ratios from a single-sample (patient)

Learn the model parameters and infer the copy number states (genotypes) using Expectation-Maximization (EM) algorithm

Annotate the copy number status for each genomic bin

Predict the copy number alteration segments (for a chromosome) using Viterbi algorithm

0. Setup the libraries and input data

0.1 Install libraries

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("HMMcopy")
```

0.2 Load libraries

```
library(HMMcopy)
library(tidyverse)
```

0.3 Load the input data

Load the input data from `log2ratios_chr1.txt`. This file contains T bins (1Mb) where each bin t has a read count (`readCount`) and a corrected $\log_2$ ratio (`log2Ratio`). The column `log2Ratio` is the corrected copy number data, $x_{1:T}$ to use, calculated using sequencing read coverage and following a Gaussian (normal) distribution (continuous measurement).

```
LogRatios <- read.delim("log2ratios_chr1.txt", as.is = TRUE)
x <- as.matrix(LogRatios$log2Ratio) # input data
copyNumberStates <- c(1,2,3,4,5)
K <- length(copyNumberStates) # number of different copy number states
```

Note: The 5 different copy number states (genotypes) correspond to: HOMD - homozygous deletion (0 copies); HEMD - hemizygous deletion (1 copy); NEUT - neutral (2 copies); GAIN (3 copies); and AMPL - amplification (4 or 5 copies)

0.4. Initialize model parameters and hyperparameters.

Initialize parameters and hyperparameters for copy number states, $\{1, 2, 3, 4, 5\}$

```
##### initial values for model parameters #####
pi.init <- c(0.1, 0.6, 0.1, 0.1, 0.1) # initial setting for pi (initial state distribution)
mu.init <- log2(c(copyNumberStates) / 2) # inital setting for Gaussian mean
var.init <- rep(var(x), times = K) # initial setting for Gaussian variance

A.init <- matrix(0, K, K) # initial setting for matrix of transition probabilities
for (i in 1:K) {
  A.init[i, ] <- (1 - 0.99999) / (K - 1) # transition to different state
  A.init[i, i] <- 0.99999  # self-transition probability
}

##### hyper-parameters for prior model #####
deltaPi.hyper <- c(2, 6, 2, 2, 2) # hyperparameter for Dirichlet prior on pi parameter

mMu.hyper <- mu.init # hyperparameter for Gaussian prior on mu parameter
sMu.hyper <- var.init # hyperparameter for Gaussian variance on mu parameter

# hyperparameters for Inverse Gamma prior on variance parameter
betaVar.hyper <- c(1,1,1,1,1)
alphaVar.hyper <- betaVar.hyper/ (apply(x, 2, var, na.rm = TRUE) / sqrt(K))

# hyperparameters for Dirichlet prior on the transition matrix
dirCounts <- 100000
deltaA.hyper <- A.init * dirCounts
```

1. Compute the Gaussian Emission Probabilities

1.1. Define a function to compute the likelihood probabilities. The emission model is a continuous HMM modeled using a k-component mixture of Gaussians

The function, `compute.gauss.lik`, computes the emission probabilities: a sequence of observation likelihoods, each expressing the probability of an observation x_t being generated from a particular state k.

`compute.gauss.lik` computes the Gaussian probability for each genomic bin t and (conditional for) each copy number state $k \in \{1, 2, 3, 4, 5\}$.

$$p(x_t|Z_t = k, \mu_{1:K}, \sigma^2_{1:K}) = \mathcal{N}(x_t|\mu_k, \sigma^2)$$

```
compute.gauss.lik <- function(x, mu.prm, var.prm){
  L <-  matrix(NA, nrow = length(x), ncol = K)
  for (i in 1:K) {
    L[,i] <- dnorm(x, mean = mu.prm[i], sd = sqrt(var.prm[i]))
  }
  return(t(L))
}
```

1.2. Compute the Gaussian likelihood Compute the Gaussian probabilities for $x_{1:T}$ of the first EM iteration using the initial Gaussian parameters set, `mu.init` and `var.init`.

Print the first 3 columns (i.e. for data points $x_{1,...,3}$ and all states k) of probabiliities for the observed likelihood.

```
obs.lik <- compute.gauss.lik(x, mu.init, var.init)
dim(obs.lik)
```

```
## [1]   5 411
```

```
obs.lik[, 1:3]
```

```
##            [,1]       [,2]        [,3]
## [1,] 0.05614798 0.12623994 0.181602980
## [2,] 0.81848272 0.82875528 0.788168624
## [3,] 0.50497414 0.32064768 0.239377684
## [4,] 0.14316797 0.06528551 0.041046514
## [5,] 0.03186925 0.01124119 0.006186008
```

2. Implement functions for EM algorithm

2.1. Compute the responsibilities in the E-Step (inference using the Forwards-Backwards algorithm (Baum-Welch))

Use compiled C code from HMMcopy to compute the $forwards\ backwards$ probabilities (responsibilities).

The `forward_backward` function computes the probability $\gamma(Z_t)$ of each bin t having every possible genotype $\forall k \in \{1, 2, 3, 4, 5\}$.

The probabilities are computed for each genomic window t and each genotype k and returns a matrix with dimensions $K\ddot{O}T$.

The function takes as arguments:

- `obs.lik`, the likelihood computed from `compute.gauss.lik`
- `piZ`, the probability of the genotypes $\pi_{1:K}$
- `A`, the matrix of transition probabilities A

Each element of the matrix of transition probabilities A, a_{ij}, represents the probability of moving from state i to j. The transition matrix shows the hidden state to hidden state transition probabilities.

fwdback <- $.Call$("forward_backward", piZ, A, obs.lik, PACKAGE = "HMMcopy")

The call to the compiled C code returns a named list of elements, including:

- `rho`, responsibilities $\gamma(Z_t)$
- `xi`, 2 slice marginals $\xi(Z_{t-1}, Z_t)$
- `loglik`, log likelihood ℓ

Again, responsibilities are the posterior of latent (hidden) states $\gamma(Z_{1:t})$ with state $Z_t = k$) being responsible for generating observation x_t. Calculated using the product of `forward` probabilities, which are the joint probabilities of observing all $past$ data up to time t when given Z_t, and `backward` probabilities, which are the conditional probabilities of all $future$ data from time t+1 to T when given Z_t

2 slice marginals are the expected number of transitions between state k at time t-1 and state j at time t.

The log likelihood ℓ is computed in the forwards recursion

Compute the responsibilities using the initial settings of the parameters `pi.init`, `A.init` and Gaussian probabilities, `obs.lik`. This is the basically the E-Step in the first iteration of EM.

Print out the log likelihood and the first 3 columns of the responsibility matrix ($\gamma(Z_{1:3})$).

```
fwdback <- .Call("forward_backward", pi.init, A.init, obs.lik, PACKAGE = "HMMcopy")

fwdback$loglik
```

```
## [1] -168.3382
```

```
fwdback$rho[, 1:3]  # gamma
```

```
##              [,1]         [,2]         [,3]
## [1,] 3.409871e-08 5.516394e-09 1.163045e-09
## [2,] 9.999995e-01 9.999998e-01 9.999999e-01
## [3,] 4.060840e-07 1.490170e-07 4.955618e-08
## [4,] 7.895511e-08 6.072052e-09 3.304384e-10
## [5,] 1.644575e-08 2.219168e-10 1.811563e-12
```

2.2. Updating the initial state distribution parameter $\pi_{1:K}$ in the M-Step

2.2.1. Write a function to update the initial state distribution parameter Write a function, `update.pi`, that computes the *maximum a posteriori* (MAP) estimate update of π_k given the responsibilities $\gamma(Z_t)$ from the E-Step and the hyperparameter δ_k^π, for Dirichlet prior on π parameter

$$\hat{\pi}_k = \frac{\gamma(Z_1 = k) + \delta^\pi(k) - 1}{\sum_{j=1}^{K} \{\gamma(Z_1 = j) + \delta^\pi(j) - 1\}}$$

The function returns a vector $\hat{\pi}_{1:k}$ with K elements, one value for each copy number state $k \in \{1, 2, 3, 4, 5\}$

```
update.pi <- function(gamma, deltaPi){
  pi.hat <- rep(NA, times = K)
  for (i in 1:K) {
    pi.hat[i] <- sum(gamma[i,1], deltaPi[i], -1)
  }
  pi.hat <- pi.hat/sum(pi.hat)
  return(pi.hat)
}
```

2.2.2. Compute $\hat{\pi}$ for the first iteration of EM Print out the values of $\hat{\pi}_{1:K}$.

```
pi.hat <- update.pi(fwdback$rho, deltaPi.hyper)
pi.hat
```

```
## [1] 0.1000000 0.5999999 0.1000000 0.1000000 0.1000000
```

2.3. Updating the Gaussian mean parameter $\mu_{1:K}$ in the M-Step

2.3.1. Write a function to update Gaussian mean parameter Write a function, `update.mu`, that computes the MAP estimate update of μ_k given the responsibilities $\gamma(Z_t)$ from the E-Step and the hyperparameters m_k and s_k for the Gaussian prior and variance on μ parameter

$$\hat{\mu}_k = \frac{s_k \sum_{t=1}^{T} \gamma(Z_t = k) x_t + m\sigma_k^2}{s_k \sum_{t=1}^{T} \gamma(Z_t = k) + \sigma_k^2}$$

The function returns a vector $\hat{\mu}_{1:k}$ with K elements, one value for each copy number state $k \in \{1, 2, 3, 4, 5\}$

```
update.mu <- function(x, gamma, sMu, mMu, var){
  mu.hat <- rep(NA, times = K)
  ex.mean <- rep(0, times = K)

  for (i in 1:K) {
    num <- 0
    denom <- 0

    for (t in seq_len(ncol(gamma))){
      num <- num + gamma[i,t] * x[t]
      denom <- denom + gamma[i,t]
    }
    mu.hat[i] <- (sMu[i] * num + mMu[i] * var[i])/(sMu[i] * denom + var[i])
    ex.mean[i] <- num/denom
```

```
  }
  update.mu.out <- list("mu.hat" = mu.hat, "ex.mean" = ex.mean)
  #return(mu.hat)
  return(update.mu.out)
}
```

2.3.2. Compute $\hat{\mu}$ for the first iteration of EM Compute and print out the values of $\hat{\mu}_{1:K}$ for the first iteration of EM

```
update.mu.out <- update.mu(x,fwdback$rho,sMu.hyper,mMu.hyper,var.init)
mu.hat <- update.mu.out$mu.hat
mu.hat
```

```
## [1] -0.74128990 -0.06420233  0.41098632  0.99999410  1.32192731
```

```
ex.mean <-  update.mu.out$ex.mean
ex.mean
```

```
## [1] -0.73828997 -0.06465811  0.41004027  0.37665916  0.32165334
```

2.4. Updating the Gaussian variance parameter $\sigma^2_{1:K}$ in the M-Step

2.4.1. Write a function to update Gaussian variance parameter Write a function, `update.var`, that computes the MAP estimate update of σ^2_k given the responsibilities $\gamma(Z_t)$ from the E-Step and the hyperparameters α_k and β_k for the Inverse Gamma prior on variance σ^2 parameter

$$\hat{\sigma}^2_k = \frac{\sum_{t=1}^{T} \gamma(Z_t = k)\,(x_t - \bar{x}_t)^2 + 2\beta_k}{\sum_{t=1}^{T} \gamma(Z_t = k) + 2\,(\alpha_k + 1)}$$

where

$$\bar{x} = \frac{\sum_{t=1}^{T} \gamma(Z_t = k) x_t}{\sum_{t=1}^{T} \gamma(Z_t = k)}$$

The function returns a vector $\hat{\sigma}^2_{1:k}$ with K elements, one value for each copy number state $k \in \{1, 2, 3, 4, 5\}$

```
update.var <- function(x, gamma, ex.mean, alphaVar, betaVar){
  var.hat <- rep(NA, times = K)

  for (i in 1:K) {
    num <- 0
    denom <- 0

    for (t in seq_len(ncol(gamma))) {
      num <- num + gamma[i,t] * (x[t] - ex.mean[i])^2
      denom <- denom + gamma[i,t]
    }
    var.hat[i] <- (num + 2*betaVar[i])/(denom + 2*(alphaVar[i] + 1))
  }
  return(var.hat)
}
```

2.4.2. Compute $\hat{\sigma}^2$ for the first iteration of EM Compute and print out the values of $\hat{\sigma}^2_{1:K}$.

```
var.hat <- update.var(x, fwdback$rho, ex.mean, alphaVar.hyper, betaVar.hyper)
var.hat
```

```
## [1] 0.06542300 0.02591266 0.04337017 0.09182823 0.09182825
```

2.5. Updating the transition probabilities A in the M-Step

2.5.1. Write a function to update the transition probabilities Write a function, `update.A`, that computes the update of $\boldsymbol{A}$ given the 2-slice marginal probabilities $\xi(Z_{t-1}, Z_t)$ from the E-Step and the hyperparameter δ^A_k for Dirichlet prior on the transition matrix

$$\hat{A}_{jk} = \frac{\sum_{t=2}^{T} \xi(Z_{t-1} = j, Z_t = k) + \delta^A(k)}{\sum_{k\prime=1}^{K} \left\{ \sum_{t=2}^{T} \xi(Z_{t-1} = j, Z_t = k\prime) + \delta^A(k) \right\}}$$

The function returns a matrix $\hat{A}$ with $K\ddot{O}K$ elements, each being the probability of transition from copy number state i at bin t (rows) to state j at bin t+1 (columns) for each copy number state $k \in \{1, 2, 3, 4, 5\}$

```
update.A <- function(xi, deltaA){
  A.hat <-  matrix(NA, nrow = K, ncol = K)
  denom <- 0
  #print(xi)    #fwdback$xi, 2 slice marginals \xi(Zt-1, Zt) - a matrix with dimensions K x K x (T-1)

  for (i in 1:K) {
    num <- 0

    for (t in 2:dim(xi)[3]) {
      num <- num + xi[i,,t]
    }
    #print(num)
    num <- num  + deltaA[i,]

    A.hat[i,] <- num
    #print(num)
    #print(A.hat[i,])

    denom <- denom + num
  }

  #print(denom)
  A.hat <- A.hat/denom

  return(A.hat)
}
```

2.5.2. Compute $\hat{\boldsymbol{A}}$ for the first iteration of EM Compute and print out the values of the K X K transition matrix $\hat{\boldsymbol{A}}$.

```
A.hat <- update.A(fwdback$xi, deltaA.hyper)
A.hat
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 9.999700e-01 3.790956e-06 1.119612e-05 2.497860e-06 2.497848e-06
## [2,] 1.248277e-05 9.999887e-01 3.788958e-06 2.496538e-06 2.496534e-06
## [3,] 1.247694e-05 2.495520e-06 9.999800e-01 2.495432e-06 2.495414e-06
## [4,] 2.500022e-06 2.500000e-06 2.500018e-06 9.999900e-01 2.500000e-06
## [5,] 2.500003e-06 2.500000e-06 2.500003e-06 2.500000e-06 9.999900e-01
```

2.6. Compute the log posterior

2.6.1. Define the function, `compute.log.posterior`, to compute the log posterior distribution. The log posterior distribution is used to monitor the convergence of EM at each iteration. This value is the sum of the log likelihood and the log of the priors in the model. The log likelihood is obtained from the Forwards-Backwards algorithm through the sum of the scaling factors for the forward recursion probabilities, $\sum_{t=1}^{T} log \sum_{k=1}^{K} \alpha(Z_t = k)$.

$$\log \mathbb{P} = \ell + \log Dirichlet(\hat{\boldsymbol{\pi}}|\boldsymbol{\delta}) + \sum_{k=1}^{K} \left\{ \log \mathcal{N}(\hat{\mu}_k|m_k, s_k) + \log InvGamma(\hat{\sigma}_k^2|\alpha_k, \beta_k) + \log Dir(A_{k,1:K}^{(0)}|\hat{A}_{k,1:K}) \right\}$$

Note: the $Dir(A_{k,1:K}^{(0)}|\hat{A}_{k,1:K})$ term corresponds to the k^{th} row and $A^{(0)}$ is the initial settings of the transition matrix, `A.init`.

```
#install.packages("LaplacesDemon")
```

```
library(LaplacesDemon)
```

```
compute.log.posterior <- function(loglik, pi.hat, deltaPi, mu.hat, mMu, sMu, var.hat,
                                  alphaVar, betaVar, A.init, A.hat){

  log.posterior <- 0

  #print(loglik)

  #Dirichet priors
  prior.dirichletpi <- 0
  prior.dirichletpi <- ddirichlet(pi.hat, deltaPi, log=TRUE)
  #print(prior.dirichletpi)

  prior.normal <- 0
  prior.invgamma <- 0
  prior.dirichletA <- 0

  for (i in 1:K) {
    prior.normal <- prior.normal + log(dnorm(mu.hat[i], mean = mMu[i], sd = sqrt(sMu[i])))
    prior.invgamma <- prior.invgamma + dinvgamma(var.hat[i],
                                                 shape = alphaVar[i], scale = betaVar[i],
                                                 log = TRUE)
    prior.dirichletA <- prior.dirichletA + ddirichlet(A.init[i,], A.hat[i,], log=TRUE)

  }
  #print(prior.normal)
  #print(prior.invgamma)
  #print(prior.dirichletA)
```

```
  log.posterior <- sum(loglik, prior.dirichletpi, prior.normal, prior.invgamma, prior.dirichletA)

  return(log.posterior)
}
```

The function returns a single number to monitor the EM algorithm for convergence.

2.6.2. Compute the log posterior for the input data Compute the log posterior for the first iteration of EM using:

- the log likelihood computed by the forwards-backwards algorithm, `loglik`,
- the updated parameters , $\hat{\pi}_{1:K}$, $\hat{\mu}_{1:K}$ and $\hat{\sigma}^2_{1:K}$, and $\hat{\boldsymbol{A}}$.
- the hyperparameters, `deltaPi.hyper`, `mMu.hyper`, `sMu.hyper`, `alphaVar.hyper`, `betaVar.hype`, `A.init` for the priors.

Print out the log posterior for the first iteration of EM.

```
compute.log.posterior(fwdback$loglik, pi.hat, deltaPi.hyper, mu.hat, mMu.hyper, sMu.hyper,
                      var.hat, alphaVar.hyper, betaVar.hyper, A.init, A.hat)
```

```
## [1] -163.7345
```

3. Learn the HMM Parameters and Predict the Copy Number Segments.

3.1. Implement and run the EM algorithm to infer the copy number states and learn the parameters.

Implement the full EM algorithm for inferring the responsibilities and estimating the HMM parameters in a Bayesian framework.

Run the EM algorithm until convergence, which is when the log posterior changes by less than $\epsilon = 10\hat{}$-2

i. Print out converged parameters:

- the Gaussian mean and variance parameters $\hat{\mu}_{1:K}$ and $\hat{\sigma}^2_{1:K}$
- the initial state distribution $\hat{\pi}_{1:K}$
- the transition matrix $\hat{A}$

ii. Print out the log posterior for all iterations.

iii. Save the Gaussian likelihood probabilities obs.lik computed in the final iteration.

```
converged = FALSE
epsilon = 10^-2

#Initialize
pi <- pi.init
mu <- mu.init
var <- var.init
A <- A.init
logP <-  -Inf
curr.iter <- 1
prev.iter <- 0

#Compute the observed likelihood using initial parameters
obs.lik <- compute.gauss.lik(x, mu, var)

while (converged == FALSE) {

  curr.iter <-  curr.iter + 1

  #E-step: Compute responsibilities
  fwdback <- .Call("forward_backward", pi, A, obs.lik, PACKAGE = "HMMcopy")

  loglik <- fwdback$loglik
  gamma <- fwdback$rho

  #M-step: Update parameters
  pi.hat <- update.pi(gamma, deltaPi.hyper)

  update.mu.out <- update.mu(x, fwdback$rho, sMu.hyper, mMu.hyper,var)
  mu.hat <- update.mu.out$mu.hat
  ex.mean <-  update.mu.out$ex.mean

  var.hat <- update.var(x, fwdback$rho, ex.mean, alphaVar.hyper, betaVar.hyper)
```

```
  A.hat <- update.A(fwdback$xi, deltaA.hyper)

  #M-step: Assign updated parameters
  pi <- pi.hat
  mu <- mu.hat
  var <- var.hat
  A <- A.hat

  #M-step: Recompute the observed likelihood using update parameters
  obs.lik <- compute.gauss.lik(x, mu, var)

  #M-step:Compute log Posterior
  logP[curr.iter] <- compute.log.posterior(fwdback$loglik, pi, deltaPi.hyper,
                                           mu, mMu.hyper, sMu.hyper, var,
                                           alphaVar.hyper, betaVar.hyper, A.init, A)
  #print(logP[curr.iter])

  prev.iter <- curr.iter - 1

  if ( (logP[curr.iter] - logP[prev.iter])  < epsilon ) {
    converged = TRUE
  }
  logP[prev.iter] <-  logP[curr.iter]
}

print(pi)
```

```
## [1] 0.1000000 0.5999999 0.1000001 0.1000000 0.1000000
```

```
print(mu)
```

```
## [1] -0.7848813 -0.0669717  0.4299975  0.9999286  1.3219278
```

```
print(var)
```

```
## [1] 0.04385666 0.02261851 0.03260682 0.09182814 0.09182825
```

```
print(A)
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 9.999589e-01 1.270577e-05 1.343537e-05 2.498079e-06 2.497965e-06
## [2,] 2.362719e-05 9.999698e-01 1.153631e-05 2.496278e-06 2.496276e-06
## [3,] 1.247785e-05 1.247748e-05 9.999701e-01 2.495560e-06 2.495551e-06
## [4,] 2.500000e-06 2.500124e-06 2.500004e-06 9.999900e-01 2.500000e-06
## [5,] 2.500000e-06 2.500002e-06 2.500000e-06 2.500000e-06 9.999900e-01
```

```
#print(obs.lik)
print(logP[1:prev.iter])
```

```
##  [1] -163.73447   99.84244  107.69116  108.82651  110.64693  115.49949
##  [7]  118.79534  120.75366  120.89278  120.92519  120.93297
```

3.2. Determine the copy number segments using the Viterbi algorithm

Compute the optimal sequence of copy number states using the Viterbi algorithm, also known as the max-sum algorithm, so computation is performed in log space.

states <- .*Call*("viterbi", log(pi.hat), log(A.hat), log(obs.lik), PACKAGE = "HMMcopy")$path

The function takes as arguments:

- `log(pi.hat)`, the converged initial state distribution in log space, $\log\hat{\pi}_{1:K}$
- `log(A.hat)`, the converged matrix of transition probabilities, $\log A$
- `log(obs.lik)`, the Gaussian likelihood probabilities in log space from the final iteration of EM, $\log\mathcal{N}(x_t|\mu_k, \sigma^2)$

The call to the compiled C code returns a named list of elements, including the sequence of copy number states (`path`).

i. Run Viterbi algorithm to obtain the sequence of copy number states using the converged parameters and Gaussian likelihood (from the final EM iteration) as input.

ii. Print out a table of the copy number segments by combining the output from the Viterbi algorithm with the original input file. A segment is defined as the consecutive set of bins with the same copy number state. The `start`, `end` and `medianLog2Ratio` for a segment are determined from the input file.

The final table will have the following columns:

- `chr`, chromosome of the segment
- `start`, start genomic coordinate of the segment
- `end`, end genomic coordinate of the segment
- `medianLog2Ratio`, median log2Ratio of the segment
- `CopyNumber`, copy number state of the segment

```
states <-  .Call("viterbi", log(pi), log(A), log(obs.lik), PACKAGE = "HMMcopy")$path
states
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 1 3 3 3 3 3 2 2 2 2 2 2
## [223] 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [260] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [297] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [334] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [371] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1
## [408] 1 1 1 1
```

```
# create table of segments
x.states <- data.frame(LogRatios) %>%
  select(-readCount) %>%
  mutate(CopyNumber = states)
#x.states

CopyNumber.init <- 0

#x.states.segs
x.states.segs <-  x.states %>%
  mutate(seg.diff = c(CopyNumber.init, diff(x.states$CopyNumber, differences = 1))) #%>%

x.states.segments <- x.states.segs %>%
  mutate(segment = cumsum(1:nrow(x.states.segs)%in%
                            which(x.states.segs$seg.diff != 0,
                                  arr.ind = TRUE, useNames = TRUE))) %>%
  group_by(segment) %>%
  summarize(
    chr = unique(chr),
    start = min(start),
    end = max(end),
    medianLog2Ratio = median(log2Ratio, na.rm = TRUE),
    CopyNumber = unique(CopyNumber)
  ) #%>%
  #select(-segment)

x.states.segments
```

```
## # A tibble: 8 x 6
##   segment   chr     start       end medianLog2Ratio CopyNumber
##     <int> <int>     <int>     <int>           <dbl>      <int>
## 1       0     1   1500001  76000000         -0.0701          2
## 2       1     1  76000001 108500000         -0.811           1
## 3       2     1 109000001 112000000         -0.102           2
## 4       3     1 112000001 114500000         -0.849           1
## 5       4     1 114500001 117000000          0.462           3
## 6       5     1 117000001 147000000         -0.0198          2
## 7       6     1 150000001 241000000          0.398           3
## 8       7     1 241000001 248500000         -0.745           1
```

```
write_delim(x.states.segments,"segments_LAC.txt")
```