به نام خدا

تحلیل داده‌های حجیم زیستی، تمرین Resequencing
مهدی کافی ۹۹۲۱۰۷۵۳

بخش صفر)
در ابتدا در این بخش باید ۳ ژنوم را آماده کنیم که یکی ژنوم Ecoli، یکی بخشی از کروموزوم ۲۰ انسان و به طول Ecoli و دیگری یک رشته تصادفی و به طول Ecoli است که به صورت زیر آن‌ها را آماده کرده‌ایم.

```r
ecoli <- Ecoli$NC_000913
ECOLI <- DNAStringSet(ecoli)
names(ECOLI) <- "Ecoli"
writeXStringSet(ECOLI, filepath = "data/ecoli.fasta")
print(ECOLI)
len <- length(ecoli)
ran_genome <- DNAString(paste0(sample(c("A", "C", "G", "T"), len, replace = TRUE), collapse = "
RAN_GENOME <- DNAStringSet(ran_genome)
names(RAN_GENOME) <- "RandomGenome"
writeXStringSet(RAN_GENOME, filepath = "data/random_genome.fasta")
print(RAN_GENOME)
human <- Views(subject = Hsapiens$chr20, start = 20000000, width = len)[[1]]
HUMAN <- DNAStringSet(human)
names(HUMAN) <- "Human"
writeXStringSet(HUMAN, filepath = "data/human.fasta")
print(HUMAN)
```

```
27:1  (Top Level)                                                    R Script
Console  Terminal  Jobs
~/Work/code/MBDA/HW3/
[1] 4639675 AGCTTTTCATTCTGACTGCAACGGGCAATAT...CAAATAAAAAACGCCTTAGTAAGTATTTTTC Ecoli
> print(RAN_GENOME)
DNAStringSet object of length 1:
      width seq                                                     names
[1] 4639675 GATAAGCGTATCATTGCCAAGCCGATGCCAC...GATACCGGTTACAAACGTGGCGACTTGCCAT RandomGenome
> print(HUMAN)
DNAStringSet object of length 1:
      width seq                                                     names
[1] 4639675 TTCAGTTTGGGAGGTGCAGCATGGGAGGTGA...GCTGCCCGGTCTCCAACTCCTCTCACCCCCC Human
>
```

سپس تابعی می‌نویسیم که با داشتن ژنوم مرجع، تعداد خوانش، طول خوانش و نرخ خطا از ژنوم مرجع خوانش تولید کند. که با استفاده از GRanges و geSeq، خوانش‌ها را تولید می‌کنیم سپس با استفاده از دستور replaceLetterAt و با ایجاد مکان‌های substitution بر اساس نرخ خطا، در خوانش‌ها خطاها را قرار می‌دهیم.

```r
generate_reads <- function(ref, read_length, read_num, error_rate, read_name)
{
  starts <- sample(1:(width(ref) - read_length + 1), size = read_num, replace = TRUE)
  gr <- GRanges(seqnames = names(ref), ranges = IRanges(start = starts, width = read_length), s
  reads <- getSeq(ref, gr)
  print(length(reads))
  # reads <- Views(ref, start = starts, width = read_length)
  indices <- rep(FALSE, length(reads) * width(reads)[1] )
  #error_rate <- 1/1000
  indices[sample(1:(length(reads) * width(reads)[1]), size = ceiling(length(reads) * width(read
  at <- matrix(data = indices, nrow = length(reads), ncol = width(reads)[1])
  letter_subject <- DNAString(paste0(sample(c("A", "C", "G", "T"), size = ceiling(length(reads)
  starts <- sample(1:(ceiling(length(reads)*width(reads)[1]*error_rate) - max(rowSums(at))), si
  letter <- as(Views(letter_subject, start = starts, width = rowSums(at)), "DNAStringSet")
  subs_reads <- replaceLetterAt(reads, at, letter)
  names(subs_reads) <- paste0("Read", 1:read_num)
  writeXStringSet(x = subs_reads, filepath = paste0(c("data/", read_name, ".fastq"), collapse =
  return(subs_reads)
}
```

و در نهایت با استفاده از پکیج velvet می‌خواهیم که از روی خوانش‌ها، ژنوم اصلی را بسازیم. که کافیست دو دستور velveth و velvetg را اجرا کنیم که اولی از روی خوانش‌ها اطلاعات و فایل‌هایی را تولید می‌کند و دیگری از روی فایل‌های قبلی، contigها را می‌سازد و آمارهایی را نیز تولید می‌کند.

```r
# 3 - Write a program that assembles genomes from a given set of reads ( you may only call avai
system("velveth ecoli_out/ 21 -fastq -short data/ecoli_reads.fastq")
system("velvetg ecoli_out/ -cov_cutoff auto")
system("velveth human_out/ 21 -fastq -short data/human_reads.fastq")
system("velvetg human_out/ -cov_cutoff auto")
system("velveth random_out/ 21 -fastq -short data/random_reads.fastq")
system("velvetg random_out/ -cov_cutoff auto")
```

خروجی این دستورات به صورت زیر است.

```
[0.538197] Removed 0 null nodes
[0.538198] Concatenation over!
[0.538198] Removing reference contigs with coverage < 0.773810...
[0.538299] Concatenation...
[0.538355] Renumbering nodes
[0.538357] Initial node count 1757
[0.538362] Removed 0 null nodes
[0.538362] Concatenation over!
[0.538547] Writing contigs into random_out//contigs.fa...
[0.563710] Writing into stats file random_out//stats.txt...
[0.568356] Writing into graph file random_out//LastGraph...
[0.608324] Estimated Coverage cutoff = 0.773810
Final graph has 1757 nodes and n50 of 153, max 491, total 269970, using 0/10000 reads
>
```

همانطور که مشاهده می‌شود در خروجی معیار n50 نیز مشخص می‌شود که در بخش بعدی به آن نیاز خواهیم‌داشت.

بخش یک)

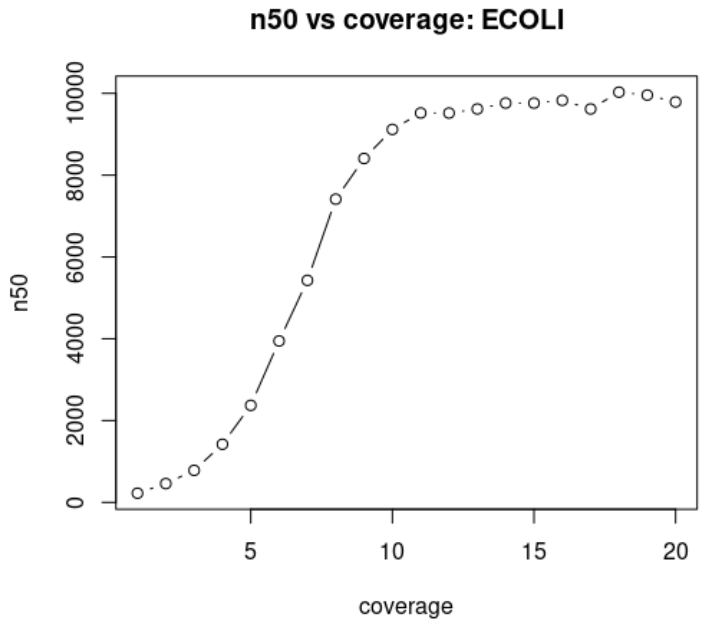
در این بخش، در ابتدا تابعی تعریف کردیم که با گرفتن خروجی دستور velvetg، مقدار n50 را به ما بدهد.

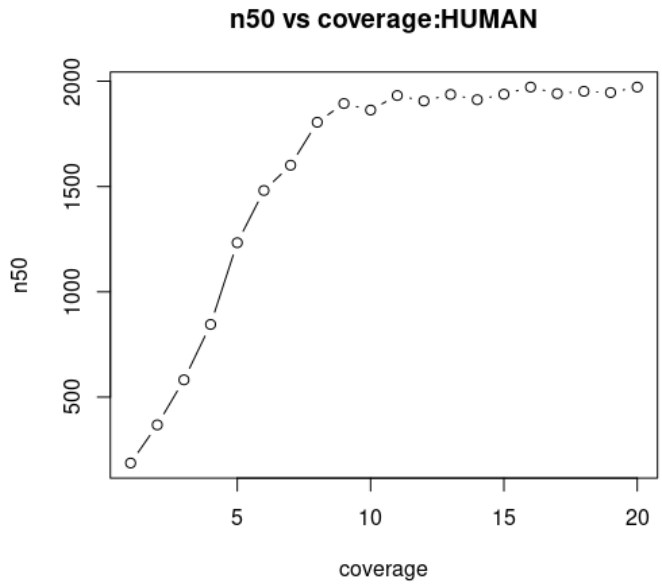
```r
n50 <- function(velvetg_output){
  str <- strsplit(strsplit(output[length(output)], ",")[[1]][1], " ")[[1]]
  n50 <- str[length(str)]
  return(strtoi(n50))
}
```

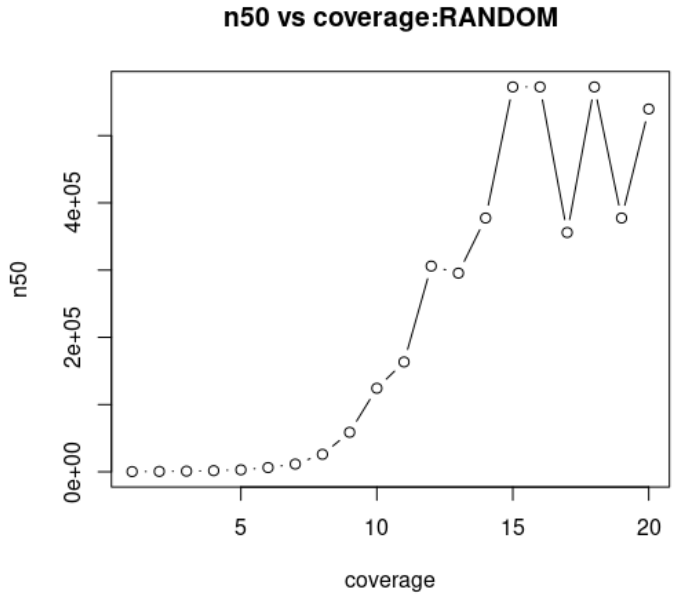
سپس برای هر ژنوم، مقدار کاورج را از ۱ تا ۲۰ اضافه کردیم و از روی آن تعداد خوانش مورد نیاز را محاسبه و بر اساس آن خوانش تولید کردیم و مقدار n50 را به دست‌آوردیم و سپس آن‌را رسم کردیم. به طور مثال دستورات برای ژنوم Ecoli به صورت زیر است.

```r
n50.vec <- c()
for (coverage in 1:20){
  read_num <- floor(coverage*len/read_length)
  reads <- generate_reads(ECOLI, read_length, read_num, error_rate, "ecoli_reads")
  system("velveth ecoli_out/ 21 -fastq -short data/ecoli_reads.fastq")
  output <- system("velvetg ecoli_out/ -cov_cutoff auto", intern = TRUE)
  n50.value <- n50(output[length(output)])
  n50.vec <- c(n50.vec, n50.value)
}
plot(1:20, n50.vec, xlab= "coverage", ylab="n50", type="b")
```

نمودارهای رسم شده به صورت زیر هستند.



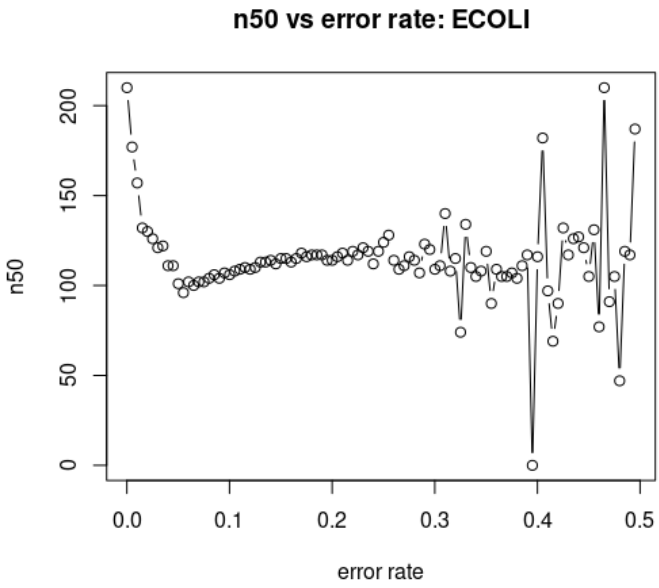


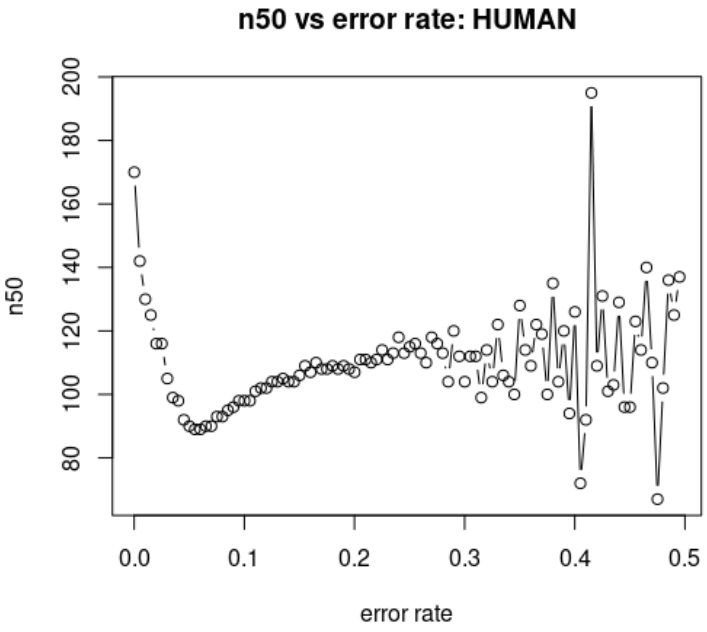


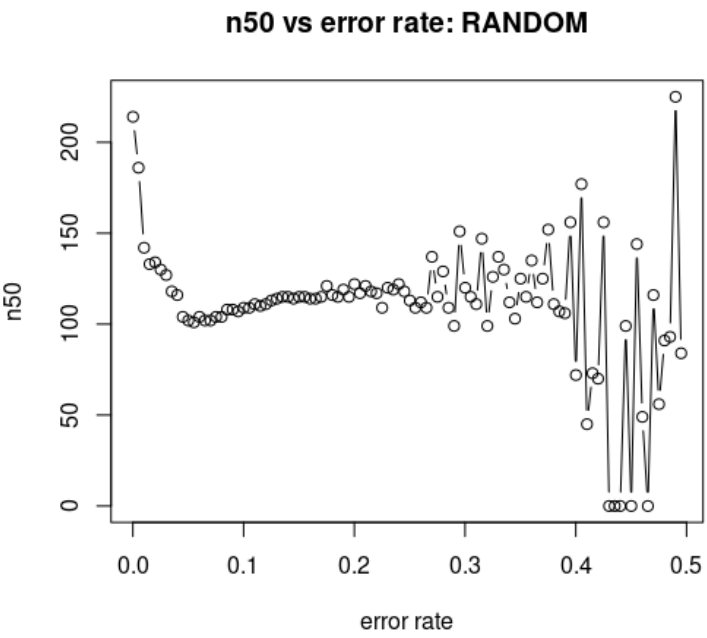
سپس می‌خواهیم که معیار n50 نسبت به مقادیر مختلف نرخ خطا در تولید خوانش تولید کنیم. در این بخش نیز مانند قبل عملی می‌کنیم و تنها تفاوت این است که مقادیر نرخ خطا را متغیر قرار می‌دهیم. در اینجا نیز کد برای Ecoli به صورت زیر است.

```r
# GOAL2: Plot N50 vs error rate.
read_length <- 150
read_num <- 25000
n50.vec.ecoli.er <- c()
for (error_rate in seq(0.0001, 0.5, 0.005)){
  reads <- generate_reads(ECOLI, read_length, read_num, error_rate, "ecoli_reads")
  system("velveth ecoli_out/ 21 -fastq -short data/ecoli_reads.fastq")
  output <- system("velvetg ecoli_out/ -cov_cutoff auto", intern = TRUE)
  n50.value <- n50(output[length(output)])
  n50.vec.ecoli.er <- c(n50.vec.ecoli.er, n50.value)
}
plot(seq(0.0001, 0.5, 0.005), n50.vec.ecoli.er, xlab= "error rate", ylab="n50", type="b")
```

و نمودارها نیز به صورت زیر هستند.




n50 vs error rate: HUMAN
n50
80
100
120
140
160
180
200
0.0
0.1
0.2
0.3
0.4
0.5
error rate


n50 vs error rate: RANDOM
n50
0
50
100
150
200
0.0
0.1
0.2
0.3
0.4
0.5
error rate

در بخش مقایسه با میزان خطا، تقریبا هر سه ژنوم رفتار یکسانی دارند که می‌تواند حاصل از تعداد کم خوانش‌ها باشد و مقدار بیشتر باعث زمان اجرای زیادی می‌شود. اما در بخش مقایسه با کاورج رفتارهای بسیار متفاوتی دارند که بسیار جالب است. دیده می‌شود که باکتری به n50 حدود ۱۰۰۰۰ همگرا می‌شود ولی ژنوم انسان تنها تا مقدار ۲۰۰۰ بالا می‌رود که می‌تواند به دلیل این باشد که ژنوم انسان قسمت‌های تکراری زیادی دارد. و مقدار n50 در ژنوم تصادفی تا ۴۰۰۰۰۰ نیز می‌رسد. و نشان می‌دهد که ژنوم انسان و یا باکتری اصلا ژنوم‌های تصادفی‌ای نیستند.

بخش دو)

در این بخش در ابتدا با استفاده از کتابخانه RSVSim، جهش‌های indel را ایجاد می‌کنیم و سپس با GRanges، جایگزینی را ایجاد می‌کنیم و از سه ژنوم، سه ژنوم جدید میسازیم.

```r
mutation <- function(genome, mutation.rate){
  #mutation.rate <- 1/1000
  mutation.size <- floor(mutation.rate*width(genome))
  no.mutation <- floor(mutation.size/10)
  sim <- simulateSV(genome = genome, dels = no.mutation/2, sizeDels = 10)
  random.genome.mutation <- paste0(sample(c("A", "C", "G", "T"), size = width(genome), replac
  sim <- DNAStringSet(c(as.character(sim[[1]]), random.genome.mutation))
  names(sim) <- c("sim", "random")
  sim <- simulateSV(genome = sim, ins = no.mutation/2, sizeIns = 10)
  sim <- sim[[1]]
  no.snp <- mutation.size
  at <- sample(1:length(sim), size = no.snp)
  snp <- sample(c("A", "C", "G", "T"), size = no.snp, replace = TRUE)
  sim <- replaceLetterAt(sim, at, snp)
  return(sim)
}
```

```r
mutation.rate <- 1/1000
mutated.ecoli <- mutation(ECOLI, mutation.rate)
mutated.random <- mutation(RAN_GENOME, mutation.rate)
mutated.human <- mutation(HUMAN, mutation.rate)
M.ECOLI <- DNAStringSet(mutated.ecoli)
M.HUMAN <- DNAStringSet(mutated.human)
M.RANDOM <- DNAStringSet(mutated.random)
writeXStringSet(M.ECOLI, "mutated/m_ecoli.fasta")
writeXStringSet(M.HUMAN, "mutated/m_human.fasta")
writeXStringSet(M.RANDOM, "mutated/m_random.fasta")
```