Sequence Statistic for Bio-Informatic

Truc Huynh

2/17/2021

Objectives

- Explore features of the Bioconductor Packages
- Query/read/Analyze sequence data.

Description

Write R code to do each of the following tasks:

Question a

Search the ACNUC "genbank" for the sequences:

i. from "Mycobacterium tuberculosis" with accession number JX303316.

```
choosebank("genbank")
Q1 <- query("Q1", "AC= JX303316")
```

ii. from the "Escherichia coli" with accession number AE005174.

```
Q2 <- query("Q2", "AC= AE005174")
```

Question b

Compute the fraction of each base in each sequence.

```
# Compute Sequence Q1 (Mycobacterium tuberculosis)

t1 = count(getSequence(Q1$req[[1]]), wordsize = 1)
(t1["a"])/sum(t1)*100

##        a
## 19.19706

(t1["g"])/sum(t1)*100

##        g
## 33.64433

(t1["c"])/sum(t1)*100

##        c
## 30.64744

(t1["t"])/sum(t1)*100
```

```
##        t
## 16.51117

# Compute Sequence Q2 (Escherichia coli)

t2 = count(getSequence(Q2$req[[1]]), wordsize = 1)

(t2["a"])/sum(t2)*100

##        a
## 24.81008

(t2["g"])/sum(t2)*100

##        g
## 25.20419

(t2["c"])/sum(t2)*100

##        c
## 25.23938

(t2["t"])/sum(t2)*100

##        t
## 24.74635
```

Question c

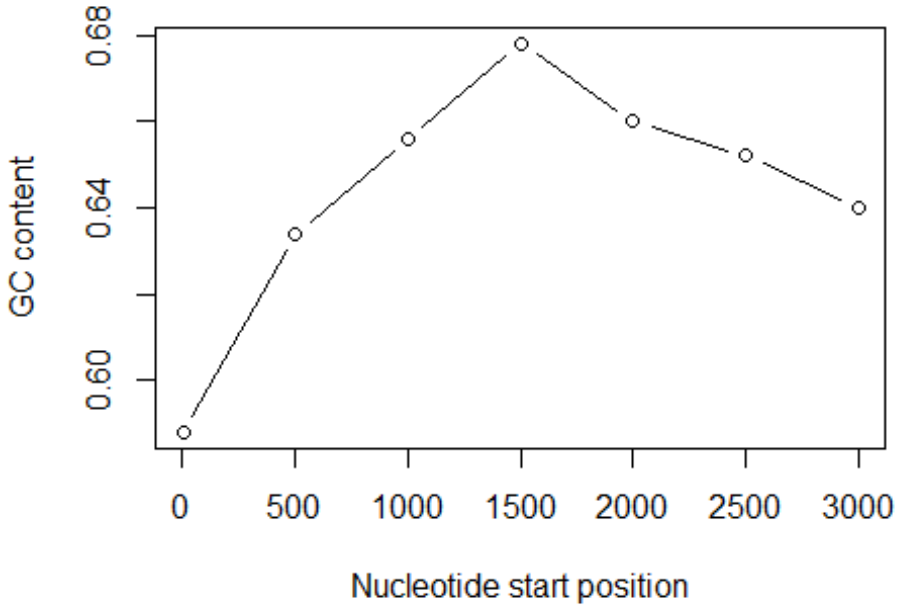
For the first sequence, calculate the GC content for each 500-nucleotide chunks of the sequence. Create a sliding window scattered plot of GC content using red lines.

```
slidingwindowplot <- function(windowsize, inputseq)
{
  starts <- seq(1, length(inputseq)-windowsize, by = windowsize)
  n <- length(starts) # Find the length of the vector "starts"
  chunkGCs <- numeric(n) # Make a vector of the same length as vector
"starts", but just containing zeroes
  for (i in 1:n) {
    chunk <- inputseq[starts[i]:(starts[i]+windowsize-1)]
    chunkGC <- GC(chunk)
    print(chunkGC)
    chunkGCs[i] <- chunkGC
  }
  plot(starts,chunkGCs,type="b",xlab="Nucleotide start position",ylab="GC
content")
}

slidingwindowplot(500, getSequence(Q1$req[[1]]))

## [1] 0.588
## [1] 0.634
```

```
## [1] 0.656
## [1] 0.678
## [1] 0.66
## [1] 0.652
## [1] 0.64
```



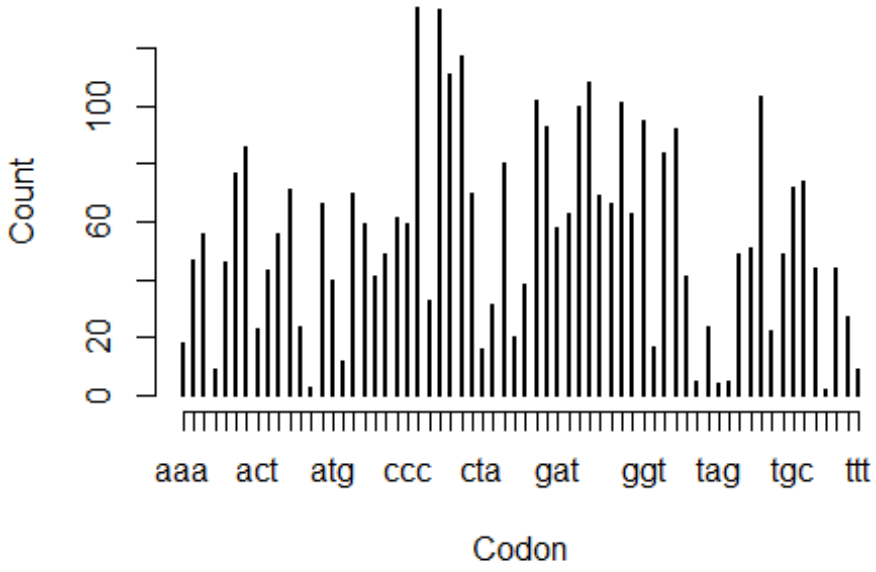
Question d

Write a function that finds and plots the count of each codon in a given sequence. Test your function on both sequences.

```
findPotentialCondonAndPlot <- function(sequence)
{
  # Define the codon vector
  CondonTable = count(sequence, wordsize = 3)
  plot(CondonTable,xlab="Codon",ylab="Count")
}
```

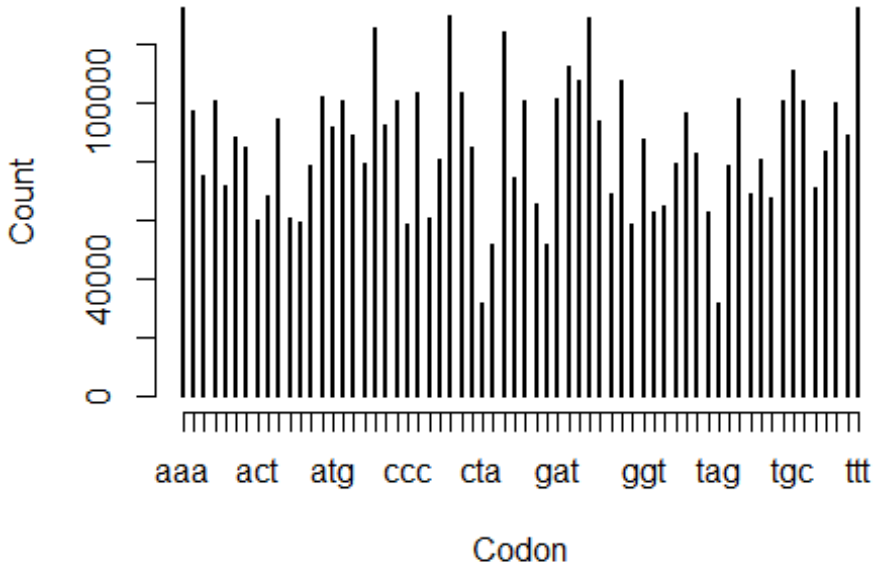
Plot of codon's frequency in Sequence 1

```
findPotentialCondonAndPlot(getSequence(Q1$req[[1]]))
```



Plot of codon's frequency in Sequence 2

```
findPotentialCondonAndPlot(getSequence(Q2$req[[1]]))
```



Question e

For the second sequence, what are the top three most frequent 3-bp words?

```
# Get the total count of word size = 3
t2.3 = count(getSequence(Q2$req[[1]]), wordsize = 3)

# Copy of the total count table
t2.3.1 <- t2.3

# Store the index of the most frequent
maxV = {}

#Search for the index of the most frequent
for (i in 1:3)
{
  maxV <- c(maxV,which.max(t2.3.1))
  t2.3.1 <- t2.3.1[-c(which.max(t2.3.1))]
}

# Store the top 3 most frequent
topThreeQ2 = {}

# Store the 3 most frequent
for (i in maxV)
{
  topThreeQ2 <- c(topThreeQ2,t2.3[i])
}

# Display the 3 most frequent
topThreeQ2
```

```
##    ttt    aaa    cga
## 132782 132616  80824
```

```
# remove all unused data
rm(i, maxV, t2.3, t2.3.1)
```

Question f

Write a function to find and return all under-represented DNA words with a specific length in a given sequence. Test your function using the second sequence for 2- and 4- bp long words.

Function 2 bp long:

```
###################################################
# Function to calculate the 2Bp nucleotides long
###################################################
underRepresent2Bp <- function(sequence)
{
```

```
  # This hold the count of 2 character words
  word_frq = count(sequence, 2) / sum(count(sequence, 2))

  # This hold the count of 1 character words
  Base_frq = count(sequence, 1) / sum(count(sequence, 1))

  # Create a data frame of 2 character words for access sub-string the base
  temp <- data.frame(word_frq)

  # container hold theρ(Rho) is used to measure
  # how over- or under-represented a particular DNA word is.
  underRepresent.2.Bp = {
  }

  for (val in 1:length(word_frq))
  {
    Names <- toString(temp[val, "Var1"]) # Get the nucleotides long names

    # Calculate theρ(Rho) based on names
    percetage <-
      word_frq[Names] / (Base_frq[substr(Names, 1, 1)] *
Base_frq[substr(Names, 2, 2)])
    if (percetage < 1)
    {
      underRepresent.2.Bp <- c(underRepresent.2.Bp, percetage)
    }
  }

  View(underRepresent.2.Bp)

  return (underRepresent.2.Bp)
}

##################################################
# Function to calculate the 3Bp nucleotides long
##################################################


underRepresent3Bp <- function(sequence)
{
  # This hold the count of 2 character words
  word_frq = count(sequence, 3) / sum(count(sequence, 3))

  # This hold the count of 1 character words
  Base_frq = count(sequence, 1) / sum(count(sequence, 1))

  # Create a data frame of 2 character words for access sub-string the base
  temp <- data.frame(word_frq)
```

```
  # container hold theρ(Rho) is used to measure
  # how over- or under-represented a particular DNA word is.
  underRepresent.3.Bp = {
  }

  for (val in 1:length(word_frq))
  {
    Names <- toString(temp[val, "Var1"]) # Get the nucleotides long

    # Calculate theρ(Rho) based on names
    percentage <-
      word_frq[Names] / (Base_frq[substr(Names, 1, 1)] *
Base_frq[substr(Names, 2, 2)] * Base_frq[substr(Names, 3, 3)])
    if (percentage < 1)
    {
      underRepresent.3.Bp <- c(underRepresent.3.Bp, percentage)
    }
  }

  View(underRepresent.3.Bp)

  return (underRepresent.3.Bp)
}


#####################################################
# Function to calculate the 4Bp nucleotides long
#####################################################

underRepresent4Bp <- function(sequence)
{
  # This hold the count of 4 character words
  word_frq = count(sequence, 4) / sum(count(sequence, 4))

  # This hold the count of 1 character words
  Base_frq = count(sequence, 1) / sum(count(sequence, 1))

  # Create a data frame of 4 character words for access sub-string the base
  temp <- data.frame(word_frq)

  # container hold theρ(Rho) is used to measure
  # how over- or under-represented a particular DNA word is.
  underRepresent.4.Bp = {
  }

  for (val in 1:length(word_frq))
  {
    Names <-
      toString(temp[val, "Var1"]) # Get the four nucleotides long names
```

```
    # Calculate theρ(Rho) based on names by substring each nucleotides
    percentage <-
      word_frq[Names] / (Base_frq[substr(Names, 1, 1)] *
Base_frq[substr(Names, 2, 2)]
                         * Base_frq[substr(Names, 3, 3)] *
Base_frq[substr(Names, 4, 4)])
    if (percentage < 1) {
      underRepresent.4.Bp <- c(underRepresent.4.Bp, percentage)
    }
  }

  View(underRepresent.4.Bp)

  return (underRepresent.4.Bp)
}


underRepresent <- function(sequence, SpecLength) {
  if (SpecLength == 4){
    underRepresent4Bp(sequence)
  } else if (SpecLength == 2){
    underRepresent2Bp(sequence)
  } else if (SpecLength == 3){
    underRepresent3Bp(sequence)
  } else{
    print("Specific Length is undefined")
  }
}

# Test the under represent two nucleotides long names
underRepresent(getSequence(Q2$req[[1]]),2)

##        ac        ag        cc        ct        ga        gg        gt
ta
## 0.8843864 0.8223558 0.9214547 0.8198983 0.9264458 0.9220757 0.8827863
0.7573850
##        tc
## 0.9260602

# test the under present four nucletides long names
underRepresent(getSequence(Q2$req[[1]]),4)

##       aact       aaga       aagg       aagt       acaa       acac
acag
## 0.89894427 0.95936672 0.75710614 0.72388134 0.94151633 0.62384561
0.97092717
##       acat       accc       acct       acga       acgg       acgt
acta
## 0.81767658 0.65910217 0.73723613 0.78064101 0.99864101 0.77847279
```

```
0.37624631
##       actc       actt       agac       agag       agat       agcc
agct
## 0.54816318 0.72567950 0.59267032 0.61413430 0.91913942 0.96114986
0.73691896
##       agga       aggc       aggg       aggt       agta       agtc
agtg
## 0.64891499 0.89329298 0.51163531 0.74557274 0.61364128 0.53156408
0.75319770
##       agtt       atac       atag       atct       atgt       caag
caca
## 0.90389084 0.77659465 0.51799600 0.92324817 0.81969564 0.52223341
0.74611750
##       cacg       cact       cata       catg       ccaa       ccac
ccca
## 0.86400608 0.76338551 0.77550982 0.84746609 0.71258985 0.99858592
0.71178852
##       cccc       cccg       ccct       ccga       ccta       cctc
cctt
## 0.50972466 0.89352081 0.52598715 0.82440201 0.22333604 0.46683011
0.75563639
##       cgac       cgag       cgga       cggg       cgta       cgtc
cgtg
## 0.89314011 0.52988557 0.95538039 0.89848545 0.76468342 0.97562820
0.87987168
##       ctaa       ctac       ctag       ctat       ctca       ctcc
ctcg
## 0.42803501 0.47402693 0.05460035 0.52484796 0.69458113 0.55228422
0.53803999
##       ctct       ctgt       ctta       cttg       gaac       gaca
gacc
## 0.60656368 0.96141135 0.56239452 0.51873575 0.99273900 0.74475106
0.74141554
##       gacg       gact       gaga       gagc       gagg       gagt
gatc
## 0.98646969 0.53899270 0.65012072 0.70794243 0.46489061 0.55113615
0.99227772
##       gcac       gccc       gcct       gcta       gctc       ggac
ggag
## 0.96192373 0.83566212 0.89863495 0.56290357 0.72265037 0.47823722
0.55774092
##       ggcc       ggct       ggga       gggc       gggg       gggt
ggta
## 0.66211418 0.96542746 0.65031733 0.81874672 0.51136390 0.66398296
0.94321740
##       ggtc       gtac       gtag       gtat       gtcc       gtcg
gtct
## 0.72740690 0.67117568 0.48702135 0.78650057 0.47195449 0.88782398
0.58539310
##       gtgc       gtgt       gtta       gttc       taac       taag
```

```
taca
## 0.95153379 0.62923456 0.98325594 0.98980590 0.98766426 0.56272493
0.62420004
##       tacc       tacg       tact       taga       tagc       tagg
tagt
## 0.93776102 0.76417270 0.61237778 0.32051799 0.56243928 0.21554473
0.37362210
##       tata       tatg       tcca       tccc       tccg       tcct
tcga
## 0.57332849 0.77688773 0.93377371 0.63760161 0.95623156 0.64454073
0.80001578
##       tcgg       tcgt       tcta       tctc       tctt       tgag
tgga
## 0.82781611 0.78839082 0.31460613 0.65290777 0.95938148 0.67411194
0.93982336
##       tggg       tgta       tgtc       tgtg       ttag       ttgg
ttgt
## 0.72151167 0.63046954 0.74770749 0.74325099 0.41718767 0.71402437
0.94837716

# Test the under represent three nucleotides long names
underRepresent(getSequence(Q2$req[[1]]),3)

##       aag       aca       acg       act       aga       agg       agt
ata
## 0.8765372 0.8381437 0.9784849 0.7029421 0.7969380 0.6999231 0.6998589
0.9344742
##       cac       ccc       cct       cga       cgt       cta       ctc
ctt
## 0.9118983 0.6606565 0.7016791 0.9282394 0.9770439 0.3693502 0.5974247
0.8759007
##       gac       gag       gga       ggg       gta       gtc       gtg
taa
## 0.7540854 0.5934494 0.7985189 0.6611745 0.7390055 0.7466893 0.9141839
0.9901523
##       tac       tag       tat       tcc       tcg       tct       tgt
tta
## 0.7359750 0.3682025 0.9419749 0.7929980 0.9312977 0.7917690 0.8391797
0.9939430

closebank()
```

Notes:

- Handwritten answers are not allowed!
- Use Rmarkdown (https://rmarkdown.rstudio.com/) and provide a neatly formatted "pdf" file showing both code and output. • Include your name as a comment at the beginning of the script file.