UBMI-IFC, UNAM
Coyoacan, CDMX

Sequece Characterization Test

Using Genomic-Benchmarks Data

GENOME FUNCTIONS ANNEX

Author: Fuentes-Mendez
David Gregorio

Table of contents

1 Introduction

1.1 Libraries used

```
# library(stringr)
# library(stringi)

library(knitr)
setwd("/home/davidfm/Projects/UBMI-IFC/EnhaProm/")
source("scripts/genome-functions.R")
source("scripts/custom-functions.R")
```

1.2 Example sequences:

```
l_seq <- "gtatgggaatcagccgggtctcactatgtgcaaa"
s_seq <- "gtatgggaat"

long_sequence <- toupper(l_seq)
short_sequence <- toupper(s_seq)

testseq1 <-
  "TGTCCGCTCCAGTCTCTCTTCCTCATCTTATAAAGCCACGAGTCCCAA"
testseq2 <-
  "CTCCAATCAGGACATGAATTCGGGGATTAAATTGCCAACACATGGCTT"
testseq3 <-
  "GTGCAGTGGCGCTATCTCGGCTCACTGCAAGCTGTTCACGCCATTCTC"

palidromeseq <- paste0("CAAGCTTGTGCAGTGTTGCTGT",
                       "TCTATCTCGGCTCACTGCAAGC",
                       "TGTTCACGCCATGTTCTGTTGG")
```

2 Basic Utilities

2.1 Counts per Base

Gets counts of each nucleotide in the sequence.

```
bases_count(long_sequence)
```

```
 A  T  C  G
 9  8  7 10
```

```
bases_count(short_sequence)
```

```
A T C G
3 3 0 4
```

2.2 Percentage per Base

Gets percentages of each nucleotide in the sequence.

```
bases_percentage(long_sequence)
```

```
        A         T         C         G
0.2647059 0.2352941 0.2058824 0.2941176
```

```
bases_percentage(short_sequence)
```

```
  A   T   C   G
0.3 0.3 0.0 0.4
```

2.3 GC Percentage

Sums percentages of cytosine (C) and guanine (G).

```
gc_percentage(long_sequence)
```

```
[1] 0.5
```

```
gc_percentage(short_sequence)
```

```
[1] 0.4
```

2.4 Base Highlight

Converts all nucleotides to lower case, except for the ones to highlight.

```
highlight_base(long_sequence, "a")
```

```
[1] "gtAtgggAAtcAgccgggtctcActAtgtgcAAA"
```

```
highlight_base(short_sequence, "at")
```

```
[1] "gtATgggaAT"
```

2.5 Reverse Complementary

Gets the reverse complementary sequence.

```
long_sequence
```

```
[1] "GTATGGGAATCAGCCGGGTCTCACTATGTGCAAA"
```

```
rev_complement(long_sequence)
```

```
[1] "TTTGCACATAGTGAGACCCGGCTGATTCCCATAC"
```

```
short_sequence
```

```
[1] "GTATGGGAAT"
```

```
rev_complement(short_sequence)
```

```
[1] "ATTCCCATAC"
```

3 Kmer Functions

3.1 Kmer Combinations

Gets all combinations of kmers of a given size (k).

```
all_k2 <- combi_kmers()
vectwrap(all_k2, width=55, padd=0)
```

```
[1] AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
```

```
all_k3 <- combi_kmers(k = 3)
vectwrap(all_k3, width=55, padd=0)
```

```
 [1] AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA
[14] ATC ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC
[27] CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG
[40] GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT
[53] TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
```

```
vectwrap(all_k3, width=55, padd=0, indexes=FALSE)
```

```
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA
ATC ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC
```

```
CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG
GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT
TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
```

3.2 Counts per Kmer

Counts occurrences of kmers inside sequences, if ‘percentage=TRUE’ provides percentages instead.

```
count_kmers(short_sequence)
```

```
AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
 1  0  0  2  0  0  0  0  1  0  2  1  1  0  1  0
```

```
count_kmers(short_sequence, all_k3)
```

```
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT C
  0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG GCT
  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT TCA TCC TCG TCT T
  1   0   1   0   1   0   0   0   0   0   0   1   0   0   0   0   0   0   1   0
TTA TTC TTG TTT
  0   0   0   0
```

```
vectwrap(count_kmers(short_sequence, all_k3), padd=0, indexes=FALSE, round_n=2)
```

```
0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
1.00 0.00 1.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00
0.00 0.00 0.00 0.00
```

```
outwrap1(count_kmers(short_sequence, all_k3), width = 55)
```

```
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA
  0   0   0   1   0   0   0   0   0   0   0   0   0
ATC ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC
  0   1   0   0   0   0   0   0   0   0   0   0   0
CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG
  0   0   0   0   0   0   1   0   0   0   0   0   0
GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT
  0   1   0   1   0   1   0   0   0   0   0   0   1
TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
  0   0   0   0   0   0   1   0   0   0   0   0
```

```
outwrap(count_kmers(short_sequence, all_k3))
```

```
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT
ATA ATC ATG ATT CAA CAC CAG CAT
0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT
GAA GAC GAG GAT GCA GCC GCG GCT
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT
TCA TCC TCG TCT TGA TGC TGG TGT
1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0

TTA TTC TTG TTT
0 0 0 0
```

```
outwrap(example_output(), width = 40)
```

```
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC
0 0 0 1 0 0 0 0 0 0

AGG AGT ATA ATC ATG ATT
0 0 0 0 1 0
```

```
count_kmers(short_sequence, percentage = TRUE)
```

```
       AA        AC        AG        AT        CA        CC        CG        CT
0.1111111 0.0000000 0.0000000 0.2222222 0.0000000 0.0000000 0.00
       GA        GC        GG        GT        TA        TC        TG        TT
0.1111111 0.0000000 0.2222222 0.1111111 0.1111111 0.0000000 0.11
```

```
another_short_sequence <- "GCGCGCGCATTCGC"
count_kmers(another_short_sequence, c("CG"))
```

```
CG
 4
```

```
count_kmers(another_short_sequence, c("CGC"))
```

```
CGC
  4
```

3.3 Kmer Windows

Splits sequences into kmers of a specified size (k). By default kmers are separated from each other by 1 nucleotide, however this separation or “stride” (s), can be also specified.

```
short_sequence
```

```
[1] "GTATGGGAAT"
```

```
kmer_windows(short_sequence)
```

```
[1] "GT" "TA" "AT" "TG" "GG" "GG" "GA" "AA" "AT"
```

```
kmer_windows(short_sequence, k = 3)
```

```
[1] "GTA" "TAT" "ATG" "TGG" "GGG" "GGA" "GAA" "AAT"
```

```
testseq1
```

```
[1] "TGTCCGCTCCAGTCTCTCTTCCTCATCTTATAAAGCCACGAGTCCCAA"
```

```
kmer_windows(testseq1, k=8)
```

```
 [1] "TGTCCGCT" "GTCCGCTC" "TCCGCTCC" "CCGCTCCA" "CGCTCCAG" "GCTC
 [7] "CTCCAGTC" "TCCAGTCT" "CCAGTCTC" "CAGTCTCT" "AGTCTCTC" "GTCT
[13] "TCTCTCTT" "CTCTCTTC" "TCTCTTCC" "CTCTTCCT" "TCTTCCTC" "CTT
[19] "TTCCTCAT" "TCCTCATC" "CCTCATCT" "CTCATCTT" "TCATCTTA" "CAT
[25] "ATCTTATA" "TCTTATAA" "CTTATAAA" "TTATAAAG" "TATAAAGC" "ATA
[31] "TAAAGCCA" "AAAGCCAC" "AAGCCACG" "AGCCACGA" "GCCACGAG" "CCA
[37] "CACGAGTC" "ACGAGTCC" "CGAGTCCC" "GAGTCCCA" "AGTCCCAA"
```

```
kmer_windows(testseq1, k=8, s=3)
```

```
 [1] "TGTCCGCT" "CCGCTCCA" "CTCCAGTC" "CAGTCTCT" "TCTCTCTT" "CTC
 [7] "TTCCTCAT" "CTCATCTT" "ATCTTATA" "TTATAAAG" "TAAAGCCA" "AGCC
[13]  "CACGAGTC"  "GAGTCCCA"
```

```
k3_tseq1 <- kmer_windows(testseq1, k = 3)
```

4 Melting Temperature (Tm) Calculation

Gets Melting Temperature of a sequence, inpendent of its length. I'd call it a fancier GC% metric.

```
tm_calc(short_sequence)
```

```
[1] 28
```

```
tm_calc(long_sequence)
```

```
[1] 65.62353
```

4.1 Tm Calculation (Sequence Length less than 14 bp)

Gets Melting Temperature of "short" sequences

```
tm_len_lt14(short_sequence)
```

```
[1] 28
```

4.2 Tm Calculation (Sequence Length more than 13 bp)

Gets Melting Temperature of "long" sequences

```
tm_len_mt13(long_sequence)
```

```
[1] 65.62353
```

5 Shannon Entropy Calculation

Gets Shannon Entropy of a sequence. Basically how entropic is a sequence given how many characters it has and what are their proportions relative to the total number of characters.

```
# Note: Add longer explanation of Shannon Entropy
shannon_entropy(testseq1)
```

```
[1] 1.895573
```

```
shannon_entropy(testseq2)
```

```
[1] 1.982964
```

6 Kmer Barcode

While thinking of a way of representing the positions of each kmer inside the whole sequence I first tried to use their positions as a binary code to then get the sum of all.

However 2^x grows a lot with just a few values. Considering I wanted to characterize more than 100 positions this was unfeasible with 2^x.

Somehow I thought about using a product of "prime numbers" given their property of having no other factors except for themselves and 1 and the fact that they don't double each other with each position. And altough the results were manageable, I still thought they were a little too big for my convinience, so I thought of getting the 'log()' of the final product.

That's when I realized that my first approximation was not that bad after all since I could just change 2 to 1.1 or 1.01 or 1.001 (and so on...) to module the growth rate I desired for them sequence positions.

After all I implemented all 3 of implementations (named: "primes", "logprimes" and "expsum"(left as default)).

```
which(k3_tseq1 == "TCT")
```

```
[1] 13 15 17 26
```

```
1.001^(which(k3_tseq1 == "TCT") - 1)
```

```
[1] 1.012066 1.014091 1.016121 1.025302
```

```
# All indexes are rested 1 so that we can also
# use 1.001^0=1 as a first possible position
kmer_barcode(kmer = "TCT", windows = k3_tseq1)
```

```
[1] 0.07164804
```

7 Functions per Windows

At first I called it a fancier 'lapply()', however later on, it became to make a more efficient way of feeding sequence data to my own functions (i.e. kmer_barcode())

```
k8_tseq1 <- kmer_windows(testseq1, k = 8)
func_per_windows(windows = k8_tseq1,func = tm_calc)
```

```
TGTCCGCT GTCCGCTC TCCGCTCC CCGCTCCA CGCTCCAG GCTCCAGT CTCCAGTC TC
    26       28       28       28       28       26       26       24
CCAGTCTC CAGTCTCT AGTCTCTC GTCTCTCT TCTCTCTT CTCTCTTC TCTCTTCC CT
    26       24       24       24       22       24       24       24
TCTTCCTC CTTCCTCA TTCCTCAT TCCTCATC CCTCATCT CTCATCTT TCATCTTA CA
    24       24       22       24       24       22       20       20
ATCTTATA TCTTATAA CTTATAAA TTATAAAG TATAAAGC ATAAAGCC TAAAGCCA AA
    18       18       18       18       20       22       22       24
AAGCCACG AGCCACGA GCCACGAG CCACGAGT CACGAGTC ACGAGTCC CGAGTCCC GA
    26       26       28       26       26       26       28       26
AGTCCCAA
      24
```

```
func_per_windows(windows = k8_tseq1,func = shannon_entropy)
```

```
TGTCCGCT GTCCGCTC TCCGCTCC CCGCTCCA CGCTCCAG GCTCCAGT CTCCAGTC TC
1.561278 1.500000 1.298795 1.548795 1.750000 1.905639 1.750000 1.
CCAGTCTC CAGTCTCT AGTCTCTC GTCTCTCT TCTCTCTT CTCTCTTC TCTCTTCC CT
1.750000 1.811278 1.811278 1.405639 0.954434 1.000000 1.000000 1.
TCTTCCTC CTTCCTCA TTCCTCAT TCCTCATC CCTCATCT CTCATCTT TCATCTTA CA
1.000000 1.405639 1.405639 1.405639 1.405639 1.405639 1.500000 1.
ATCTTATA TCTTATAA CTTATAAA TTATAAAG TATAAAGC ATAAAGCC TAAAGCCA AA
1.405639 1.405639 1.405639 1.405639 1.750000 1.750000 1.750000 1.
AAGCCACG AGCCACGA GCCACGAG CCACGAGT CACGAGTC ACGAGTCC CGAGTCCC GA
1.561278 1.561278 1.561278 1.905639 1.905639 1.905639 1.750000 1.
AGTCCCAA
1.811278
```

```
func_per_windows(kmers = all_k3, windows = k3_tseq1, func = km
```

```
      AAA         AAC         AAG         AAT         ACA         ACC
0.032500996 0.000000000 0.033533497 0.000000000 0.000000000 0.00
      ACG         ACT         AGA         AGC         AGG         AGT
0.038711510 0.000000000 0.000000000 0.034567031 0.000000000 0.05
      ATA         ATC         ATG         ATT         CAA         CAC
0.030439088 0.025302313 0.000000000 0.000000000 0.047050345 0.03
```

```
      CAG       CAT       CCA       CCC       CCG       CCT 
0.010045120 0.024278035 0.091677624 0.044959381 0.004006004 0.021211336 
      CGA       CGC       CGG       CGT       CTA       CTC 
0.039750222 0.005010010 0.000000000 0.000000000 0.000000000 0.059465509 
      CTG       CTT       GAA       GAC       GAG       GAT 
0.000000000 0.045507762 0.000000000 0.000000000 0.040789972 0.000000000 
      GCA       GCC       GCG       GCT       GGA       GGC 
0.000000000 0.035601598 0.000000000 0.006015020 0.000000000 0.000000000 
      GGG       GGT       GTA       GTC       GTG       GTT 
0.000000000 0.000000000 0.000000000 0.056939813 0.000000000 0.000000000 
      TAA       TAC       TAG       TAT       TCA       TCC 
0.031469527 0.000000000 0.000000000 0.029409678 0.023254780 0.075137667 
      TCG       TCT       TGA       TGC       TGG       TGT 
0.000000000 0.071648040 0.000000000 0.000000000 0.000000000 0.001000000 
        TTA         TTC         TTG         TTT 
0.028381297 0.019171973 0.000000000 0.000000000 
```

```r
start_time <- Sys.time()
func_per_windows(windows = k8_tseq1, func = shannon_entropy)
```

```
TGTCCGCT GTCCGCTC TCCGCTCC CCGCTCCA CGCTCCAG GCTCCAGT CTCCAGTC TCCAGTCT 
1.561278 1.500000 1.298795 1.548795 1.750000 1.905639 1.750000 1.811278 
CCAGTCTC CAGTCTCT AGTCTCTC GTCTCTCT TCTCTCTT CTCTCTTC TCTCTTCC CTCTTCCT 
1.750000 1.811278 1.811278 1.405639 0.954434 1.000000 1.000000 1.000000 
TCTTCCTC CTTCCTCA TTCCTCAT TCCTCATC CCTCATCT CTCATCTT TCATCTTA CATCTTAT 
1.000000 1.405639 1.405639 1.405639 1.405639 1.405639 1.500000 1.500000 
ATCTTATA TCTTATAA CTTATAAA TTATAAAG TATAAAGC ATAAAGCC TAAAGCCA AAAGCCAC 
1.405639 1.405639 1.405639 1.405639 1.750000 1.750000 1.750000 1.405639 
AAGCCACG AGCCACGA GCCACGAG CCACGAGT CACGAGTC ACGAGTCC CGAGTCCC GAGTCCCA 
1.561278 1.561278 1.561278 1.905639 1.905639 1.905639 1.750000 1.905639 
AGTCCCAA 
1.811278 
```

```r
round(Sys.time() - start_time, 2)
```

```
Time difference of 0 secs
```

```r
bioseq <- Biostrings::DNAString(testseq1)
rev_bioseq <- Biostrings::DNAString(stri_reverse(testseq1))
revc_bioseq <- Biostrings::DNAString(rev_complement(testseq1))

rev_align <- Biostrings::pairwiseAlignment(bioseq, rev_bioseq, type = "global")
revc_align <- Biostrings::pairwiseAlignment(bioseq, revc_bioseq, type = "global")

rev_align
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: TGTCC---GCTCCAGTCTCTCTTCCTCATCTTATAAAGCCACGAGTCCCAA
subject: AACCCTGAGCACCGAAATATTCTACTCCTTCTCTCTGACCTCG---CCTGT
score: -143.966 
```

```r
Biostrings::pid(rev_align)
```

```
[1] 41.17647
```

```r
revc_align
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: TGTCCGCTCCAGTCTCTCTTCCTCATCTTATAAAGCCACGAGTCCCAA
subject: TTGGGACTCGTGGCTTTATAAGATGAGGAAGAGAGACTGGAGCGGACA
score: -141.3069 
```

```r
Biostrings::pid(revc_align)
```

```
[1] 37.5
```