UBMI-IFC, UNAM
Coyoacan, CDMX

Sequece Characterization Test

Using Genomic-Benchmarks Data

THEORY & CODE ANNEX

Author: Fuentes-Mendez
David Gregorio

Table of contents

Appendix: Custom Functions

The following functions were made in order to provide a better presentation to the data displayed in this notebook:

Note:

Each time the programming language changes, the code-cell will display a header indicating the corresponding language.

R Code

```r
wrap_output <- function(func_out, width = 50) {
  output <- capture.output(func_out)
  wrapped_output <- lapply(output, strwrap,
                           width = width)
  cat(wrapped_output, sep = "\n")
}

outputwrap <- function(func_out, width = 50) {
  form_out <- format(round(func_out, digits = 2),
                     nsmall = 2)
  wrapped <- strwrap(gsub(',', '', toString(form_out)),
                     width = width)
  wrapped[1] <- paste("\n\n[1]", wrapped[1])
  len_w <- length(wrapped)
  wrapped[2:len_w] <- paste("\n    ", wrapped[2:len_w])
  cat(wrapped)
}
```

```r
addpadd <- function(func_out) {
  captout <- capture.output(func_out)
  captout[1] <- paste("\n", captout[1], sep="")
  captout[2:length(captout)] <- paste("\n",
            captout[2:length(captout)], sep="")
  captout <- append(captout, "\n\t")
  cat(captout)
}
```

```r
hspace <- function()
  knitr::asis_output("\\textcolor{white}
    {\\tiny\\texttt{hi}}\\normalsize")
```

```r
uniq_values <- function(vect, round_digits = 4){
  return(round(as.numeric(names(table(vect))),
               digits = round_digits))}

uniq_not <- function(vect, round_digits=4){
  return(round(unique(sort(vect)),
               digits=round_digits))}
# ^For some reason returned duplicated values when
#  processing floating values from "gc-percentage"
```

1 Data Recollection

1.1 Data Sources and Sequence Lengths

1.1.1 Promoter Elements: Core, Proximal and Distal

1.1.2 Enhancer Elements:

1.2 Isolation & Delimitation Difficulties

1.3 Promoter Bashing & Enhancer Trapping

2 Determining Sequence Characterization

2.1 Key Concepts

Biological sequences often contain dependencies, such as motifs, repeat regions, or structural constraints, that influence nucleotide placement.

2.2 Whole-Sequence Characterization

2.3 Kmer Characterization

```r
all_k3 <- combi_kmers(k = 3)
all_k4 <- combi_kmers(k = 4)
all_k5 <- combi_kmers(k = 5)
all_k6 <- combi_kmers(k = 6)

length(all_k3);length(all_k4)
length(all_k5);length(all_k6)
```

```
[1] 64        [1] 256       [1] 1024      [1] 4096
```

2.4 Kmer Distribution Characterization

2.5 Characterization Dependence on Sequence Length

3 Understanding: GC Percentage & Melting Temperature

3.1 GC Percentage: Definition & Formula

GC Percentage (also called GC content) refers to the proportion of guanine (G) and cytosine (C) bases in a DNA sequence. Since G and C bases form three hydrogen bonds (as opposed to the two bonds between adenine (A) and thymine (T) in DNA, sequences with a higher GC content are typically more stable and harder to denature. It can be calculated using ths simple formula:

$$GC\% = \frac{(\textbf{C}\text{ count} + \textbf{G}\text{ count})}{\text{Sequence Length}} \quad (1)$$

3.2 Melting Temperature: Definition & Formula

Melting Temperature (Tm) refers to the temperature at DNA denatures into two separate strands. It is critical, for example, in experiments like PCR (*Polymerase Chain Reaction*) because the annealing temperature is usually set a few degrees below the Tm to allow proper primer binding. It can be estimated using the following formulas:

- **For short sequences** (3 to 13~20 nucleotides)**:**

$$Tm = 2(\textbf{A}\text{ count} + \textbf{T}\text{ count}) + 4(\textbf{C}\text{ count} + \textbf{G}\text{ count}) \tag{2}$$

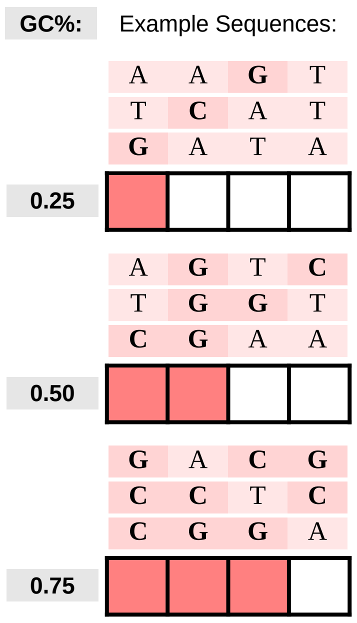
- **For long sequences** (>13~20 nucleotides)**:**

$$Tm = 64.9 + 41 * \frac{(\textbf{A}\text{ count} + \textbf{T}\text{ count}) - 16.4}{\text{Sequence Length}} \tag{3}$$

3.3 Issues in Kmer Characterization

One major issue with GC measurement is its inability to capture the nucleotide arrangement within a sequence. Two sequences can have identical GC content but different nucleotide order, leading to different structural and functional properties (**Figure 1**). For example **'GCGCTT'** will evidently behave different than **'GGAACC'**, even though they have the **same GC%** and the **same TM** (**Figure 1**).

Figure 1: GC% Redundancy



GC ratio alone will fail to capture a complete picture of a sequence's biological function since it can provide no insight into sequence motifs or complexity, meaning that it fails to differentiate between truly complex sequences and those that are repetitive or structurally constrained. However one thing it can supply is information regarding sequence length importance. We will notice two oppsting issues for each function: while sequence length does not influence *GC Percentage* computation, for *Melting Temperature* it does. This should lead to considerably different experimental approaches dependent on the kind of data available for our analysis. These approaches were dicussed in Section 2.5.

Consequently and referent to sequence length, we observe that for sequences (or kmers) of **size *N*** we will **always obtain at most *N*+1 different values** (considering the empty case of sequences that lack G/C nucleotides), using either function. A practical representation of this and the redundancy of values (considering the amount of kmers), can be observed with the following code cells:

```
#List of GC% values per kmer-set
all_k3_gc <- func_per_windows(windows = all_k3,
                              func = gc_percentage)
all_k4_gc <- func_per_windows(windows = all_k4,
                              func = gc_percentage)
all_k5_gc <- func_per_windows(windows = all_k5,
                              func = gc_percentage)
#List of TM values per kmer-set
all_k3_tm <- func_per_windows(windows = all_k3,
                              func = tm_calc)
all_k4_tm <- func_per_windows(windows = all_k4,
                              func = tm_calc)
all_k5_tm <- func_per_windows(windows = all_k5,
                              func = tm_calc)
all_k6_tm <- func_per_windows(windows = all_k6,
                              func = tm_calc)
```

```
uk3_gc <- uniq_values(all_k3_gc, round_digits = 2)
uk4_gc <- uniq_values(all_k4_gc, round_digits = 2)
uk5_gc <- uniq_values(all_k5_gc, round_digits = 2)

# kmer-set    | number of     | values #
# size        | unique values | list   #
length(all_k3); length(uk3_gc); uk3_gc
length(all_k4); length(uk4_gc); uk4_gc
length(all_k5); length(uk5_gc); uk5_gc
```

```
[1] 64      [1] 4    [1] 0.00 0.33 0.67 1.00
[1] 256     [1] 5    [1] 0.00 0.25 0.50 0.75 1.00
[1] 1024    [1] 6    [1] 0.0 0.2 0.4 0.6 0.8 1.0
```

```
uk3_tm <- unique(all_k3_tm)
uk4_tm <- unique(all_k4_tm)
uk5_tm <- unique(all_k5_tm)
uk6_tm <- unique(all_k6_tm)

length(uk3_tm); uk3_tm
length(uk4_tm); uk4_tm
length(uk5_tm); uk5_tm
length(uk6_tm); uk6_tm
```

```
[1] 4     [1]  6  8 10 12
[1] 5     [1]  8 10 12 14 16
[1] 6     [1] 10 12 14 16 18 20
[1] 7     [1] 12 14 16 18 20 22 24
```

In summary, while GC measurement is a useful and simple measure for basic sequence characterization, its inability to capture nucleotide arrangement and sequence complexity limits its utility.

4 Understanding: Shannon Entropy Coefficient

4.1 Shannon Entropy Coefficient: Definition & Formula

The *Shannon Entropy Coefficient* is a *measure of uncertainty* in a probability distribution. If we have a discrete (countable) random variable X with n possible outcomes $\{x_1, x_2, ..., x_n\}$, its' Shannon Coefficient would be calculated by means of multiplying each outcome's probability by its logaritm and then sum the products; for a system with *n* possible outcomes and probabilities $p_1, p_2, ..., p_n$ this formula would be depicted as:

$$H(X) = -\sum_{i=1}^{n} p_i \log_b(p_i) \tag{4}$$

Interpretation:

- **Minimum Entropy:** When there's only one possible outcome (i.e. a group consisting of a single element like {A,A,A,A}), the entropy value is *zero*.
- **Maximum Entropy:** When all outcomes are equally likely, the entropy is at its maximum, this indicates the highest level of uncertainty. Its value is dependent on the logaritm base and the number of possible outcomes:
  - In the case of base 2, for a system with *n* equally probable outcomes, the maximum entropy is $\log_2(n)$. For example:
    - **2 Outcomes:** $\log_2(2) = 1$ i.e. {G,C} or {A,A,T,T}
    - **4 Outcomes:** $\log_2(4) = 2$ i.e. {A,T,G,C} or {A,A,T,T,G,G,C,C}

Since we do use $\log_2(x)$ for our *Shannon Entropy* calculation, and only the 4 canonical nucleotides are considered for each sequences characterization, we might as well say that the Shannon Entropy Coefficient values in our tables are distributed in a spectrum ranging from **zero** to **two** (at most).

4.2 Issues in Kmer Characterization

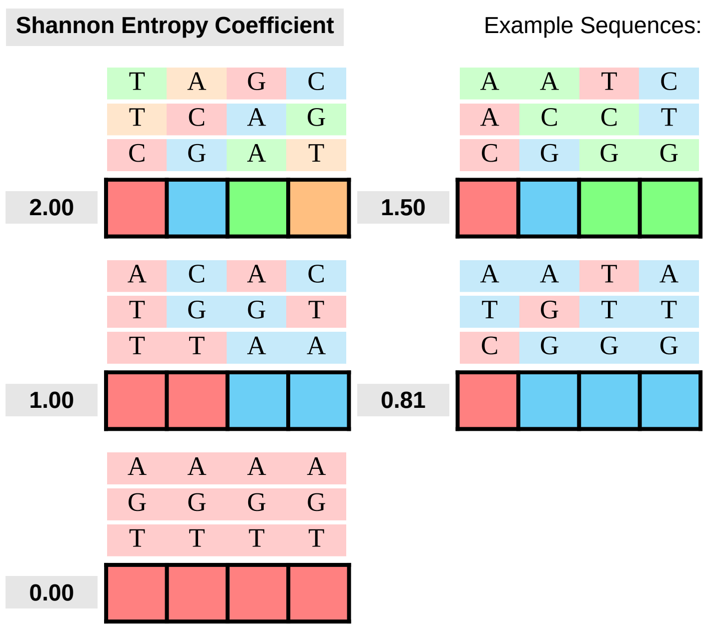
While this metric provides insights into the complexity and diversity of nucleotide compositions, through the distribution of nucleotide frequencies, it suffers from one of the same defects of GC measurement: it pays no attention to nucleotide positions.

This becomes fairly important when we consider the influence that 'order' has in a biological context. For instance, even though '**ATG**' and '**TAG**' are composed of the same nucleotides, they are a **start** and **stop** codon, respectively. Other examples are conserved motifs or regions where the order of nucleotides determines secondary structure like hairpins, loops, or binding sites. Furtherfore, given that in regulatory sequences specific motifs might restrict certain nucleotides to particular positions (thus, reducing their entropy), this function might be rather suited as a partial approach to our kmer-characterization objective.

However, I believe the biggest issue here is the lack of distinction between sequences with equal diversity but different elements. Take as an example this two sequences: '**AAAT**' and '**CGCC**', both with identical entropy values, but clearly different biological properties (**Figure 2**).

In order to know the number of possible values per size of kmer-set, we'll have to use a little bit of *Number Theory*. In particular, *partitions*.

Figure 2: Entropic Redundancy



A partition of a positive integer *n* is a multiset of positive integers that sum to *n*. The total number of partitions of *n* is denoted by p_n. Thus, given that
5 = 4 + 1 = 3 + 2 = 3 + 1 + 1 = 2 + 2 + 1 = 2 + 1 + 1 + 1 = 1 + 1 + 1 + 1 + 1
is a complete enumeration of the partitions of 5, $p(5) = 7$.

There is no simple formula for p_n, however its not hard to find a generating function for them, one of the examples is in the R library *'partitions'* (which will be used in the rest of the chapter). There, the function P() works as its homonym and parts() computes the spread-out matrix of partitions. Theres also the homolog functions for restricted partitions R() and restrictedparts(), which compute the number of partitions conditioned upon an *m* maximum number of parts.

```r
library(partitions)

P(5);                 R(4,5, include.zero=TRUE);
addpadd(parts(5));    restrictedparts(5,4)
```

```
[1] 7                           [1] 6
[1,] 5 4 3 3 2 2 1
[2,] 0 1 2 1 2 1 1              [1,] 5 4 3 3 2 2
[3,] 0 0 0 1 1 1 1              [2,] 0 1 2 1 2 1
[4,] 0 0 0 0 0 1 1              [3,] 0 0 0 1 1 1
[5,] 0 0 0 0 0 0 1              [4,] 0 0 0 0 0 1
```

Additionally we can code a function like the following:

```r
rawpart <- function(n,k){
  if (missing(k)) k <- n
  if (k == 0) return(0)
  if (n == 0) return(1)
  if (n < 0) return(0)
  return(rawpart(n, k-1) + rawpart(n-k, k))
}
```

```
rawpart(3); rawpart(4); rawpart(5); rawpart(6);
rawpart(7); rawpart(8); rawpart(9); rawpart(10);
hspace();
P(3);       P(4);       P(5);       P(6);
P(7);       P(8);       P(9);       P(10);
```

```
[1] 3          [1] 5          [1] 7          [1] 11
[1] 15         [1] 22         [1] 30         [1] 42

[1] 3          [1] 5          [1] 7          [1] 11
[1] 15         [1] 22         [1] 30         [1] 42
```

There exist also multiple mathematical formulas for computation. However the oldest one is Euler's geometric series where the coefficient of x^n is equal to p_n, and it can be shown as:

$$\frac{(1+x+x^2+x^3+\dots)(1+x^2+x^4+x^6+\dots)}{(1+x^3+x^6+x^9+\dots)\dots(1+x^k+x^{2k}+x^{3k}+\dots)} = \prod_{k=1}^{\infty}\sum_{i=0}^{\infty} x^{ik} \tag{5}$$

```
rparts <- restrictedparts
Rz <- function(n_opts, n_slots) {
  return(R(n_opts, n_slots, include.zero = TRUE))
}
rP <- function(n_opts, n_slots) {
  return(ncol(rparts(n_slots, n_opts)))
}
```

```
all_k3_sh <- func_per_windows(windows = all_k3,
                              func = shannon_entropy)
all_k4_sh <- func_per_windows(windows = all_k4,
                              func = shannon_entropy)
all_k5_sh <- func_per_windows(windows = all_k5,
                              func = shannon_entropy)
all_k6_sh <- func_per_windows(windows = all_k6,
                              func = shannon_entropy)

uk3_sh <- uniq_values(all_k3_sh, round_digits = 2)
uk4_sh <- uniq_values(all_k4_sh, round_digits = 2)
uk5_sh <- uniq_values(all_k5_sh, round_digits = 2)
uk6_sh <- uniq_values(all_k6_sh, round_digits = 2)

length(uk3_sh); rP(4,3); uk3_sh
length(uk4_sh); rP(4,4); uk4_sh
length(uk5_sh); rP(4,5); uk5_sh
length(uk6_sh); rP(4,6); outputwrap(uk6_sh, width = 30)
```

```
[1] 3    [1] 3    [1] 0.00 0.92 1.58
[1] 5    [1] 5    [1] 0.00 0.81 1.00 1.50 2.00
[1] 6    [1] 6    [1] 0.00 0.72 0.97 1.37 1.52 1.92
                  [1] 0.00 0.65 0.92 1.00 1.25 1.46
[1] 9    [1] 9        1.58 1.79 1.92
```

```
rparts(4,4);      rparts(5,4);      rparts(6,4);
```

```
addpadd(rparts(4,4));    addpadd(rparts(5,4));
addpadd(rparts(6,4));    addpadd(rparts(7,4));
```

```
[1,] 4 3 2 2 1 [1,] 5 4 3 3 2 2 [1,] 6 5 4 3 4 3 2 3 2
[2,] 0 1 2 1 1 [2,] 0 1 2 1 2 1 [2,] 0 1 2 3 1 2 2 1 2
[3,] 0 0 0 1 1 [3,] 0 0 0 1 1 1 [3,] 0 0 0 0 1 1 2 1 1
[4,] 0 0 0 0 1 [4,] 0 0 0 0 0 1 [4,] 0 0 0 0 0 0 0 1 1
```

```
[1,] 4 3 2 2 1                [1,] 5 4 3 3 2 2
[2,] 0 1 2 1 1                [2,] 0 1 2 1 2 1
[3,] 0 0 0 1 1                [3,] 0 0 0 1 1 1
[4,] 0 0 0 0 1                [4,] 0 0 0 0 0 1

[1,] 6 5 4 3 4 3 2 3 2        [1,] 7 6 5 4 5 4 3 3 4 3 2
[2,] 0 1 2 3 1 2 2 1 2        [2,] 0 1 2 3 1 2 3 2 1 2 2
[3,] 0 0 0 0 1 1 2 1 1        [3,] 0 0 0 0 1 1 1 2 1 1 2
[4,] 0 0 0 0 0 0 0 1 1        [4,] 0 0 0 0 0 0 0 0 1 1 1
```

In summary, while Shannon entropy is a valuable tool for measuring sequence diversity and randomness, it has significant limitations in biological sequence characterization. Its assumption of positional independence and inability to reflect nucleotide order make it less suitable as a standalone metric for understanding the complexity and functionality of nucleotide sequences. To gain deeper insights, Shannon entropy should be used in combination with other metrics that account for sequence structure, motifs, and functional relevance.

Blablalbalalblab