ANÁLISE DOS DADOS COM R

Genômica com R

James R. Hunter, PhD

Retrovirologia, EPM, UNIFESP

GENÔMICA BÁSICA COM R

FERRAMENTA `bioseq`

- Pacote fora de sistema Bioconductor
- Ideia central: criar classes de vetores para armanezar sequências de DNA, RNA, e aminoácidos
- Facilita utilização das funções de base R e tidyverse para manipular esses vetores
  - Tb, conduzir operações básicas úteis
  - Vetores são baseados em tipo caractere (*strings*)
- Produto de François Keck
  - Departamento de Ecologia Aquática, Eidgenössische Anstalt für Wasserversorgung, Abwasserreinigung und Gewässerschutz, Zurich, Suíça

DIFERENÇA ENTRE BIOCONDUCTOR E `bioseq`

- Pacotes de Bioconductor (alguns 2.800) utilizam o sistema S4 de R para gravar os dados
- Pacotes de base R, tidyverse, e `bioseq` utilizam o sistema original S3
- S3 e S4 são sistemas de OOP (*object oriented programming*)
  - Também, existem R6 e S7

OOP – O QUE É?

- Um paradigma de programação
  - Programação funcional é um outro paradigma
    - Que vemos em `purrr:map()`
- OOP - objetos fazem parte de classes e têm dentro:
  - campos (variáveis) e
  - metodos (funções que podem ser aplicadas aos campos
- Linguagems como Java foram criadas como linguagens de OOP
- R é mais uma linguagem funcional
- Mas, têm pacotes escritos com a lógica de OOP
  - Como pacotes de Bioconductor (sistema S4)

DNA & RNA

- Funções bioseq::dna() e bioseq::rna() marcam sequências como esse tipo de molecula
  - Sequência é um vetor das letras dentro do alfabeto IUPAC
- Alfabeto IUPAC
  - Os quatro nucleotídeos específicos: **A, C, G, T** (ou **U** para RNA)
  - As letras dos nucleotídeos ambíguous

COMPLETO ALFABETO IUPAC - DNA

```
1 Biostrings::IUPAC_CODE_MAP
```

```
    A      C      G      T      M      R      W      S      Y      K      V
  "A"    "C"    "G"    "T"   "AC"   "AG"   "AT"   "CG"   "CT"   "GT"  "ACG"
    H      D      B      N
"ACT"  "AGT"  "CGT" "ACGT"
```

IUPAC DNA Codes - UCSC		
Symbol	Bases	Origin of designation
G	G	Guanine
A	A	Adenine
T	T	Thymine
C	C	Cytosine
R	G or A	puRine
Y	T or C	pYrimidine
M	A or C	aMino
K	G or T	Keto
S	G or C	Strong interaction (3 H bonds)
W	A or T	Weak interaction (2 H bonds)
H	A or C or T	not-G, H follows G in the alphabet
B	G or T or C	not-A, B follows A
V	G or C or A	not-T (not-U), V follows U
D	G or A or T	not-C, D follows C
N	G or A or T or C	aNy

EXEMPLO SIMPLES - *GAG* GENE DO VÍRUS HIV

```
1 gag <- bioseq::read_fasta(here("gag_sequence.fasta"), type = "DNA")
```

- Função `read_fasta()` precisa como argumentos:
  - Um arquivo fasta
  - Especificação do tipo do arquivo: DNA, RNA, ou Aminoácido

RETORNA UM VETOR DO TIPO QUE PEDIMOS

```
1 gag
```

```
DNA vector of 1 sequences
 Grp6_2R_GAG  GCCTGTTAGAAACAGCAGAGGGCTGTAGACAGATAATAGAACAGCTACAACCAGCCCTTA...
+ 1231 bases
```

```
1 class(gag)
```

```
[1] "bioseq_dna" "vctrs_vctr"
```

```
1 typeof(gag)
```

```
[1] "character"
```

ANÁLISES BÁSICAS DE *GAG*

- Há vários atributos da sequência de *gag* que podemos estudar
  - Tamanho da sequência - quantos bases tem
  - Quantos de cada tipo de base a sequência tem (número e %)
  - Conteúdo G-C da sequência
  - Algumas desses precisam de um vetor das bases
  - Outras só o *string* de *gag*
- Utilizando funções de `bioseq` e de `stringr`
  - Lembrando que o `typeof()` de uma sequência é `character`

CONTAGEM DE CADA BASE – 1

- Usar a função `table()` de **base R**
- Precisa converter a sequência em um vetor das letras das bases
  - Agora, a sequência é um único elemento do objeto `gag`

```
1 length(gag)
```

```
[1] 1
```

- Criar uma função para fazer isso
- Primeiro, como vamos separar as bases em elementos do vetor?
  - Função `stringr::str_split_1()`
  - Argumentos: (1) a sequência (em aspas) e (2) caractere que vamos usar para o divisão
    - Neste caso, “” (2 aspas juntas)

```
1 str_split_1("ACTGGCC", "")
```

```
[1] "A" "C" "T" "G" "G" "C" "C"
```

CONTAGEM DE CADA BASE – 2

- Colocar esta função dentro de uma função que facilita a chamada
- Nome: `seq_table()` : retorna uma tabela da sequência
- Argumento: `s` : a sequência na formato de um string das bases

```
seq_table <- function(s) {
  table(str_split_1(s, ""))
}

seq_table(gag)
```

```
  A   C   G   T 
482 251 314 244 
```

OUTROS CÁLCULOS DA SEQUÊNCIA

- Os outros vêm de `bioseq`
- Proporção das bases na sequência (`seq_stat_prop()`)
- Proporção da sequência que é G-C (`seq_stat_gc()`)
- Nomes das bases por extenso (`seq_spellout()`)
  - para sequências curtas - pode ser muito longo

COMBINAR CÁLCULOS EM UMA FUNÇÃO

- Para facilitar uso desses funções, pode combinar em uma função
- Esta função vai tratar sequências sem caracteres ambigúous
- Vai só executar seq_spellout() se tem menos de 20 bases

```
1 stats_dna <- function(s){
2   print(seq_table(s))
3   print(seq_stat_prop(s))
4   if(nchar(s) <= 20) print(seq_spellout(s))
5 }
```

APLICADO A gag

```
1 stats_dna(gag)
```

```
  A   C   G   T 
482 251 314 244 
$` Grp6_2R_GAG`
        A         C         G         T         W         S         M 
K 
0.3733540 0.1944229 0.2432223 0.1890008 0.0000000 0.0000000 0.0000000 
0.0000000 
        R         Y         B         D         H         V         N 
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 
```

APLICADO A "GCCTGTTAGAAACAG"

- Primeiros 15 bases de gag

```
1 stats_dna(dna("GCCTGTTAGAAACAG"))
```

```
A C G T
5 3 4 3
[[1]]
        A         C         G         T         W         S         M
K
0.3333333 0.2000000 0.2666667 0.2000000 0.0000000 0.0000000 0.0000000
0.0000000
        R         Y         B         D         H         V         N
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000

[1] "guanine - cytosine - cytosine - thymine - guanine - thymine - thymine -
adenine - guanine - adenine - adenine - adenine - cytosine - adenine -
guanine"
```

`seq_stat_prop()` PRODUZ UMA LISTA

```
1 class(seq_stat_prop(gag))
```

```
[1] "list"
```

```
1 typeof(seq_stat_prop(gag))
```

```
[1] "list"
```

- Primeira vez trabalhando com este tipo de objeto

LISTAS – RAPIDINHO

- Método de acesso um pouco diferente dos tibbles (não só “$”)
- Uso dos colchetes por cada nível das listas
- Lista Simples

```
1 l <- list(name = "Jim", age = 78, rating = "good guy")
2 l
```

```
$name
[1] "Jim"

$age
[1] 78

$rating
[1] "good guy"
```

- Agora, lista não tão simples
- Equivalente a um “diccionário” em Python

```
1 ll <- list(name = list("Jim", "Juan"), age = list(78, 26), rating = list("g
```

- **Live Coding** sobre como referir aos elementos desta lista

MELHORANDO O RESULTADO DE `seq_stat_prop()`

- A função sempre mostra todos as 15 bases possíveis. Muito extenso
- Suponha que queremos incluir somente as bases que tenham um valor > 0
- Criar função

PRIMEIRO PASSO - EXECUTAR

`seq_stat_prop()`

- Atribuir resultado a uma variável (`char_list`)
- Sabemos que o resultado será uma lista

```
1 char_list <- seq_stat_prop(gag)
2 class(char_list)
```

```
[1] "list"
```

SEGUNDO PASSO – CONVERTER A LISTA PARA DATA FRAME

- Lembre que tibbles e data frames são listas ao fundo
- Letras dos nomes das bases são nomes de fileira

```
1 char_df <- as.data.frame(char_list[[1]])
2 head(char_df)
```

```
  char_list[[1]]
A      0.3733540
C      0.1944229
G      0.2432223
T      0.1890008
W      0.0000000
S      0.0000000
```

PASSO 3 – LIMPAR OS NOMES DAS VARIÁVEIS

```r
char_df <- rownames_to_column(char_df, var = "base") # nomes para uma colun
names(char_df) <- c("base", "prop")
head(char_df)
```

```
  base      prop
1    A 0.3733540
2    C 0.1944229
3    G 0.2432223
4    T 0.1890008
5    W 0.0000000
6    S 0.0000000
```

PASSO 4 – FILTRANDO VALORES POSITIVOS

```r
filter(char_df, prop > 0)
```

```
  base      prop
1    A 0.3733540
2    C 0.1944229
3    G 0.2432223
4    T 0.1890008
```

COMBINAR PASSOS EM UMA FUNÇÃO

- `return()` os valores positivos
- Argumento: `seq`

```
clean_seq_stat_prop <- function(seq) {
  char_list <- seq_stat_prop(seq)
  char_df <- as.data.frame(char_list[[1]])
  char_df <- rownames_to_column(char_df, var = "base")
  names(char_df) <- c("base", "prop")
  return(filter(char_df, prop > 0))
}
```

APLICAR FUNÇÃO A `gag` E OUTRA

```
1 clean_seq_stat_prop(gag)
```

```
  base      prop
1    A 0.3733540
2    C 0.1944229
3    G 0.2432223
4    T 0.1890008
```

```
1 clean_seq_stat_prop(dna("GCCTGYTAR")) # test
```

```
  base      prop
1    A 0.1111111
2    C 0.2222222
3    G 0.2222222
4    T 0.2222222
5    R 0.1111111
6    Y 0.1111111
```

TRATANDO BASES AMBIGÚOUS

- Só A, C, G, T são as bases verdadeiros
- Outras letras são combinações possíveis de bases que o sequenciador determinou.

```
1 Biostrings::IUPAC_CODE_MAP[5:15]
```

```
   M      R      W      S      Y      K      V      H      D      B      N
"AC"   "AG"   "AT"   "CG"   "CT"   "GT"  "ACG"  "ACT"  "AGT"  "CGT" "ACGT"
```

FUNÇÃO DE `bioseq` QUE PODE AJUDAR COM A ATRIBUIÇÃO

- `bioseq::seq_disambiguate_IUPAC()` mostra todas as alternativas para substituir as bases ambigúos
- Aplicar ela para sequências teste

```
1  seq_a <- dna("GCCTGYTAR")
2
3  seq_table(seq_a)
```

```
A C G R T Y
1 2 2 1 2 1
```

```
1  clean_seq_stat_prop(seq_a)
```

```
  base      prop
1    A 0.1111111
2    C 0.2222222
3    G 0.2222222
4    T 0.2222222
5    R 0.1111111
6    Y 0.1111111
```

APLICAR

seq_disambiguate_IUPAC()

```
1 seq_a
```

```
DNA vector of 1 sequences
>   GCCTGYTAR
```

```
1 seq_disambiguate_IUPAC(seq_a)
```

```
[[1]]
DNA vector of 4 sequences
>   GCCTGCTAA
>   GCCTGTTAA
>   GCCTGCTAG
>   GCCTGTTAG
```

AGORA, VOCÊ DEVE TOMAR A DECISÃO DE

TRANSCRIÇÃO E TRADUÇÃO

- Como tornar nucleotídeos em aminoácidos

TRANSCRIÇÃO

- DNA -> RNA
- Substituição de **U** (uracil) no lugar de **T** (thymine)
- Argumento: sequência de **DNA**
- Pode também fazer transcrição reversa com `seq_rev_transcribe()`
  - Argumento: Sequência de **RNA**

```
1 gag_rna <- seq_transcribe(gag)
2 seq_table(gag_rna)
```

```
  A   C   G   U
482 251 314 244
```

TRADUÇÃO

- RNA -> AA
- Argumento principal: sequência de **RNA**
- Mais 2 argumentos importantes
  - `code` – `int` que indica o código genético (fonte de código)
    - Padrão (1) é o código para toda célula for do mitocondrial
  - `codon_frame` – int que especifica a posição do nucleotídeo onde começar tradução
    - Mesma coisa como "*reading frame*" – valores de 1 até 6
    - Três para frente, três para trás
    - Padrão é `codon_frame = 1`

TRADUÇÃO DE gag

- Podemos comparar esta tradução aqui com a tradução feito com uma outra ferramenta do internet
- Vamos usar *reading frame* 1

```
gag_test_aa <- seq_translate(gag, codon_frame = 1)
gag_test_aa
```

```
AA vector of 1 sequences
 Grp6_2R_GAG  AC*KQQRAVDR**NSYNQPLKQDQKKLNPYIIQ*QPLYCVHQKIEVRDTKEALDKIEEEQ... + 370 amino acids
```

```
old_gag_aa <- read_fasta(here("gag protein RF1.fasta"), type = "AA")
old_gag_aa
```

```
AA vector of 1 sequences
BC_2R_GAG_1  ACYKQQRAVDRYYNSYNQPLKQDQKKLNPYIIQYQPLYCVHQKIEVRDTKEALDKIEEEQ... + 371 amino acids
```

```
seq_table(gag_test_aa)
```

```
 *  A  C  D  E  F  G  H  I  K  L  M  N  P  Q  R  S  T  V  W  Y
 4 34 10 14 30 10 27  9 23 33 24 14 24 29 35 25 22 26 22  7  8
```

```
seq_table(old_gag_aa)
```

```
 A  C  D  E  F  G  H  I  K  L  M  N  P  Q  R  S  T  V  W  X  Y
34 10 14 30 10 27  9 23 33 24 14 24 29 35 25 22 26 22  7  1 12
```

OUTRAS OPERAÇÕES ÚTEIS

- Inverso da sequência de DNA, RNA, ou AA
  - `seq_reverse()`
- Complemento da sequência de DNA ou RNA

```
1 seq <- dna("ACTTTGGCTAAG")
2 seq
```

```
DNA vector of 1 sequences
>   ACTTTGGCTAAG
```

```
1 seq_reverse(seq)
```

```
DNA vector of 1 sequences
>   GAATCGGTTTCA
```

```
1 seq_complement(seq)
```

```
DNA vector of 1 sequences
>   TGAAACCGATTC
```

AGORA BIOCONDUCTOR