Sequence Analysis in R

BCB 504: Applied Bioinformatics

Matt Settles

University of Idaho
Bioinformatics and Computational Biology Program

March 6, 2012

Outline

What to use and when

Both R and Python have significant functionality to perform sequence analysis

Python Biopython project - almost all biological sequence based

R Bioconductor project (specifically Biostrings, BSgenome, GenomicFeatures, IRanges, ShortRead, Rsamtools, rSFFreader, and others)

Both Python and R are usefull when you want to either preprocess data (ie cleanup, QA, etc.), or post processing of data (ie summarization, visualization and statistical analysis). When your dealing with one record, or read, at a time Python can be much more efficient and quicker. When you need analyze all data at once, or perform statistical assessment/analysis, R can be much more efficient.

IBEST tools: modules, rSFFreader

need to first install git: http://git-scm.com/
Then clone the package to your computer, from the command line
git clone git://github.com/msettles/rSFFreader.git

Install into R, from the command line
R CMD build rSFFreaderR CMD INSTALL
rSFFreadeR_0.99.0.tar.gz

The Genomics Resources Core module - grc
IBEST computational help

```
source("http://bioconductor.org/biocLite.R");
biocLite(c("Biostrings","BSgenome", "GenomicFeatures",
"hgu95av2probe", "BSgenome.Celegans.UCSC.ce2",
"BSgenome.Scerevisiae.UCSC.sacCer2",
"BSgenome.Hsapiens.UCSC.hg19",
"SNPlocs.Hsapiens.dbSNP.20101109",
"TxDb.Hsapiens.UCSC.hg19.knownGene",
"TxDb.Scerevisiae.UCSC.sacCer2.ensGene"))
```

Biostrings

- Data structures for representing large biological sequences (DNA/RNA/amino acids)
- Utilities for basic computations on sequences
- Tools for sequence matching and pairwise alignments

BSgenome and genome packages

- Full genomes stored in Biostring containers
- 16 organisms available via Bioconductor
- Facilities for supporting your own via BSgenomeForge

Basic Sequence Classes

Single sequence BString, DNAString, RNAString, AAString

Set of sequences BStringSet, DNAStringSet, RNAStringSet, AAStringSet,

Views on sequences Views

Masked sequences MaskedBString, MaskedDNAString, MaskedRNAString, MaskedAAString

Constructing Sequences

```
> library(Biostrings)
> dna <- DNAString("acctttGtGG-NNYaA")
> dna
  16-letter "DNAString" instance
seq: ACCTTTGTGG-NNYAA

> RNAString(dna)
  16-letter "RNAString" instance
seq: ACCUUUGUGG-NNYAA

> DNA_ALPHABET
 [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B"
[15] "N" "-" "+"
```

Constructing Sets of Sequences

```
> library(Biostrings)
> dnaset <- DNAStringSet(c("acctttGtGG-NNYaA","GATTACA"))
> dnaset

  A DNAStringSet instance of length 2
    width seq
[1]    16 ACCTTTGTGG-NNYAA
[2]     7 GATTACA

> names(dnaset) <- c("DNA1","DNA2")
> dnaset

  A DNAStringSet instance of length 2
    width seq                              names
[1]    16 ACCTTTGTGG-NNYAA                 DNA1
[2]     7 GATTACA                          DNA2
```

Basic Transformations

```
> dna <- DNAString("TCAACGTTGAATAGCGTACCG")
> reverseComplement(dna)

  21-letter "DNAString" instance
seq: CGGTACGCTATTCAACGTTGA

> translate(dna)

  7-letter "AAString" instance
seq: STLNSVP

> translate(reverseComplement(dna))

  7-letter "AAString" instance
seq: RYAIQR*
```

Alphabets

```
> alphabetFrequency(dna)

A C G T M R W S Y K V H D B N - +
6 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0

> dinucleotideFrequency(dna)

AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
 2  2  1  1  1  1  3  0  1  1  0  2  2  1  1  1

> trinucleotideFrequency(dna)[1:30]

AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG
  0   1   0   1   0   1   1   0   0   1   0   0   1   0   0
ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC
  0   1   0   0   0   0   0   1   0   0   0   0   2   0   0
```

additionally uniqueLetters(dna) and oligonucleotideFrequency(dna)

BSgenome packages

```
> library(BSgenome.Celegans.UCSC.ce2)
> #Celegans
> chrI <- Celegans$chrI
> chrI

  15080483-letter "DNAString" instance
seq: GCCTAAGCCTAAGCCTAAGCCTAAGC...GCTTAGGCTTAGGTTTAGGCTTAGGC
```

Input/Output of Sequences

```
> seqs <- read.DNAStringSet(
+   system.file("extdata","fastaEx.fa",package="Biostrings"))
> seqs

  A DNAStringSet instance of length 2
    width seq                                names
[1]    72 AGTACGTAGTCGC...GACTAAACGATGC sequence1
[2]    70 AAACGATCGATCG...TACTGCATGCGGG sequence2

> write.XStringSet(seqs,file="outseqs.fasta")
```

additionally read.BStringSet,read.AAStringSet,read.RNAStringSet

Subsequences

```
> subseq(seqs,start=c(5,9),end=c(10,35))

  A DNAStringSet instance of length 2
    width seq                          names
[1]     6 CGTAGT                       sequence1
[2]    27 GATCGTACTCGACTGATGTAGTATATA  sequence2
```

ShortRead Package

```
> library(ShortRead)
> fq <- readFastq(
+   system.file("unitTests/cases","sanger.fastq",package="ShortRead"))
> fq

class: ShortReadQ
length: 1 reads; width: 27 cycles

> quality(fq)
> sread(fq)
> id(fq)
```

- SFF files are the raw data format for 'pyrosequencing' like data (ie Roche 454 and Ion Torrent)
- Roche software provides for 2 tools: sffinfo sfffile
- To use these tools you must load the module roche454
- SFF files contain run information, read information, sequence, qualities and flowgrams

rSFFreader Package

```
> library(rSFFreader)
> sff <- readsff(
+   system.file("extdata","SmallTest.sff",package="rSFFreader"))

Total number of reads to be read: 100
reading header for sff file:/Library/Frameworks/R.framework/Versions/2.
reading file:/Library/Frameworks/R.framework/Versions/2.14/Resources/li

> sff

class: SffReadsQ
file: /Library/Frameworks/R.framework/Versions/2.14/Resources/library/r
length: 100 reads; width: 51..428 basepair; clipping mode: Full

> clipMode(sff) <- "Raw"
> sff

class: SffReadsQ
file: /Library/Frameworks/R.framework/Versions/2.14/Resources/library/r
length: 100 reads; width: 80..458 basepair; clipping mode: Raw
```

A common problem: find all the occurences (aka matches or hits) of a given pattern (typically short) in a (typically long) reference sequence (aka the subject)

- **matchPattern**: **1** pattern, **1** subject (reference sequence)
- **vmatchPattern**: **1** pattern, **N** subject (reference sequence)
- **matchPDict**: **N** pattern, **1** subject (reference sequence)
- **vmatchPDict**: **N** pattern, **N** subject (reference sequence)

pDict functions have 2 major limitations

- Dictionary must be preprocessed first
- must be constant width

matchPattern

```
> library(Biostrings)
> library(BSgenome.Celegans.UCSC.ce2)
> matchPattern("CAACTCCGATCG", Celegans$chrII)

  Views on a 15279308-letter DNAString subject
subject: CCTAAGCCTAAGCCTAAGCCTAAG...CTTAGGCTTAGGCTTAGGCTTAGT
views:
       start      end width
[1] 13490043 13490054    12 [CAACTCCGATCG]
```

matchPattern

```
> matchPattern("CAACTCCGATCG", Celegans$chrII, max.mismatch=1)

  Views on a 15279308-letter DNAString subject
subject: CCTAAGCCTAAGCCTAAGCCTAAG...CTTAGGCTTAGGCTTAGGCTTAGT
views:
        start      end width
 [1]   448786   448797    12 [CAAATCCGATCG]
 [2]  1258669  1258680    12 [CAACTCCGATGG]
 [3]  3340998  3341009    12 [CAGCTCCGATCG]
 [4]  3441302  3441313    12 [CACCTCCGATCG]
 [5]  4059036  4059047    12 [CAACTCCGATCT]
 [6]  4588202  4588213    12 [CAACTTCGATCG]
 [7]  7209941  7209952    12 [CAACTCCGATCC]
 [8]  9946308  9946319    12 [CAACTCCGATCC]
 [9] 11068482 11068493    12 [CAACTCCGATTG]
[10] 11304102 11304113    12 [CAACTCCGATCT]
[11] 13490043 13490054    12 [CAACTCCGATCG]
[12] 13760610 13760621    12 [CAACTCCGATTG]
[13] 15213851 15213862    12 [CAACTCCGATCT]
```

matchPattern

```
> matchPattern("CAACTCCGATCG", Celegans$chrII, max.mismatch=1, with.ind

  Views on a 15279308-letter DNAString subject
subject: CCTAAGCCTAAGCCTAAGCCTAAG...CTTAGGCTTAGGCTTAGGCTTAGT
views:
        start      end width
 [1]   448786   448797    12 [CAAATCCGATCG]
 [2]   861918   861928    11 [CAACTCCGATG]
 [3]  1258669  1258679    11 [CAACTCCGATG]
 [4]  1947047  1947057    11 [CAACTCCATCG]
 [5]  2022293  2022303    11 [CAACTCCATCG]
 [6]  2517033  2517043    11 [CAACTCCATCG]
 [7]  2658084  2658094    11 [CACTCCGATCG]
 [8]  2839525  2839535    11 [CAACTCCGATG]
 [9]  3340998  3341009    12 [CAGCTCCGATCG]
 ...      ...       ...   ... ...
[30] 10984646 10984658    13 [CAACTCCGATTCG]
[31] 11068482 11068493    12 [CAACTCCGATTG]
[32] 11304102 11304112    11 [CAACTCCGATC]
```

matchPDict

load the dictionary

```
> library(hgu95av2probe)
> dict0 <- DNAStringSet(hgu95av2probe)
> dict0

  A DNAStringSet instance of length 201800
          width seq
      [1]    25 TGGCTCCTGCTGAGGTCCCCTTTCC
      [2]    25 GGCTGTGAATTCCTGTACATATTTC
      [3]    25 GCTTCAATTCCATTATGTTTTAATG
      [4]    25 GCCGTTTGACAGAGCATGCTCTGCG
      [5]    25 TGACAGAGCATGCTCTGCGTTGTTG
      [6]    25 CTCTGCGTTGTTGGTTTCACCAGCT
      [7]    25 GGTTTCACCAGCTTCTGCCCTCACA
      [8]    25 TTCTGCCCTCACATGCACAGGGATT
      [9]    25 CCTCACATGCACAGGGATTTAACAA
      ...   ... ...
[201792]    25 GAGTGCCAATTCGATGATGAGTCAG
[201793]    25 ACACTGACACTTGTGCTCCTTGTCA
```

matchPDict

preprocess the dictionary

```
> pdict <- PDict(dict0)
> pdict

TB_PDict object of length 201800 and width 25 (preprocessing algo="ACtr
```

load the subject

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> chr1 <- unmasked(Hsapiens$chr1)
> chr1

  249250621-letter "DNAString" instance
seq: NNNNNNNNNNNNNNNNNNNNNNNNNN...NNNNNNNNNNNNNNNNNNNNNNNNNN
```

call matchPDict

```
> m <- matchPDict(pdict, chr1)
> m

MIndex object of length 201800
```

matchPDict

query the object

```
> m[[700]] # results for the 700th pattern

IRanges of length 3
        start       end width
[1]  59015037  59015061    25
[2] 110955918 110955942    25
[3] 197066271 197066295    25

> startIndex(m)[[700]]

[1]  59015037 110955918 197066271

> endIndex(m)[[700]]

[1]  59015061 110955942 197066295
```

More string matching functions
countPattern, vCountPattern, countPDict, vcountPDict: counts the number of matches

- **trimLRPatterns**: trim left and/or right patterns from sequences
- **matchLRPatterns**: the matches are specified by a left pattern, a right pattern and a maximum distance between them
- **matchProbePair**: finds amplicons given by a pair of primers (simulate PCR)
- **matchPWM**: finds motifs described by a Position Weight Matrix (PWM)
- **findPalindromes/findComplimentedPalindromes**
- **pairwiseAlignment**: solves (NeedlemanWunsch) global alignment, (Smith Waterman) local alignement, and (endsfree) overlap alignment problems

not all of these support with.indels **yet**, see manuals for what is supported at this time

using trimLRPattern

```
> subject <- DNAStringSet(c("TGCTTGACGCAAAGA", "TTCTGCTTGGATCGG"))
> subject

  A DNAStringSet instance of length 2
    width seq
[1]    15 TGCTTGACGCAAAGA
[2]    15 TTCTGCTTGGATCGG

> trimLRPatterns(Lpattern="TTCTGCTT", Rpattern="ATCGGAAG", subject)

  A DNAStringSet instance of length 2
    width seq
[1]     9 GACGCAAAG
[2]     2 GG
```

using pairwiseAlignment

```
> pairwiseAlignment("TTGCACCC", "TTGGATTGACCCA")

Global PairwiseAlignedFixedSubject (1 of 1)
pattern: [1] TTGCA-----CCC
subject: [1] TTGGATTGACCCA
score: -29.90804

> pairwiseAlignment("TTGCACCC", "TTGGATTGACCCA", type="global-local")

Global-Local PairwiseAlignedFixedSubject (1 of 1)
pattern: [1] TTGCACCC
subject: [6] TTG-ACCC
score: -0.1277071

> pairwiseAlignment("TTC", "ATTATTA", type="global-local")

Global-Local PairwiseAlignedFixedSubject (1 of 1)
pattern: [1] TTC
subject: [5] TTA
score: -2.596666
```