LGBIO2010 – Bioinformatics

Assignment 1

Sequence statistics

1 Introduction

In all assignments for this course, you will have to analyze real biological data with some computing methods. To do so, we suggest using `R` and we will occasionally give you some hints about how to solve things using this language. We recommend `R` (`http://www.r-project.org/`) for many reasons: it is freely available and runs on all common operating systems; it comes with good tutorials to start using it; it includes convenient graphical functions to look at the data and to plot results; it also includes loads of statistical and data processing packages maintained by an active community; it is the programming language behind the Bioconductor project (`http://www.bioconductor.org/`) gathering dedicated tools for the analysis and comprehension of high-throughput genomic data.

What **will matter** is to go beyond a "click on a WEB server" attitude. If the many existing web services are indeed particularly interesting to **search** and **get access** to some annotated biological data, this approach is rapidly limited when you want to **manipulate** large collections of data and to **compute** on them.

For instance, whenever you will have to deal with, say, a DNA fragment of one million base pairs (still a fairly limited volume of real data), trying to cut and paste with your mouse such a fragment would be both inconvenient and highly prone to errors. A better way to go is to download such a fragment, for example in a flat file (typically in FASTA format), and then load the content of this file in an appropriate data structure within your computing environment (for instance, a variable storing a string or a list of consecutive characters). A possible alternative is to access directly some public database from within your computing environment. For instance, some existing R functions allow you to directly query GenBank or RefSeq with an accession number and to store the result of this query.

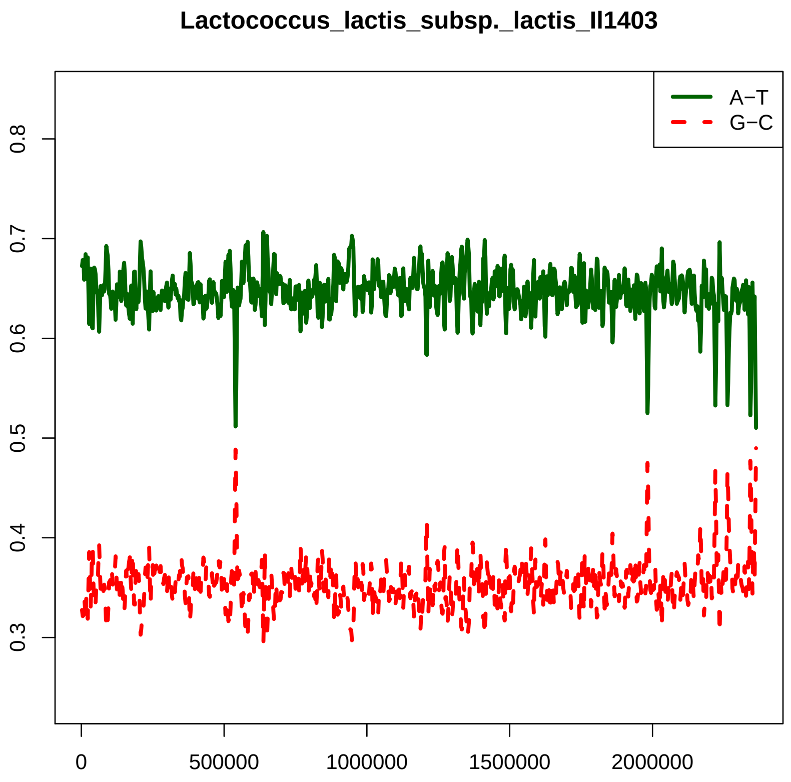
Sometimes, you might also have to write your own functions (or scripts) and/or to adapt some existing software to perform the required analysis.

Concretely, for each assignment, you will have to submit on Moodle a report (in PDF format) summarizing your answers to all questions in a frame in the assignment description + the R code you have been writing to solve the problems at hand. Please mention the software environment (typically existing R packages) you are re-using. It is OK to re-use **public** softwares but you **must** quote **precisely** your references.

2 Change Point Analysis

Your task is to perform the GC-content analysis of a specific genome. It will give you a first opportunity to get access to some real data, to compute some simple statistics from it and to better understand the algorithm for analyzing the GC-content of a DNA sequence.

1. Download the full genome of the *Lactococcus lactis subsp. lactis Il1403* (Accession number `NC_002662`) and report its ***length*** in number of base pairs (bp).
2. Determine the ***size*** (in number of bp) of the ***sliding window*** chosen to produce the plot of its GC content below.



3. Is this plot exhibiting an anomaly in terms of GC content and if so where? Report a plot on a specific fragment of the sequence if needed.
4. What is the expected influence of ***reducing*** or ***enlarging*** this window size? Describe this influence and, possibly, include some additional plots to support your analysis.

Hints

- The **`ape`** R package has convenient **`read.GenBank()`** and **`read.dna()`** functions to load actual sequence in a R session, from an accession number or a FASTA file respectively. Please check also *A very brief introduction to R*, available from Moodle to get more information in this regard.
- The **`count`** function from the **`seqinr`** R package is convenient to count k-mers in a sequence.

3 Unusual dimers

1. Download the complete genome of the *Lactococcus lactis subsp. lactis Il1403* (Accession number `NC_002662`) and report its ***length*** in number of base pairs (bp).
2. Fill in the table below with the ***observed frequencies*** of all dimers in this sequence. The row (respectively column) label denotes the first (respectively second) nucleotide in the dimer. For instance, the relative frequency of the `AG` dimer is 0.0555. You have to fill the rest of the table.

	*A	*C	*G	*T
A*			0.0555	
C*				
G*				
T*				

Table 1: Observed dimer frequencies in the *Lactococcus lactis subsp. lactis Il1403* genome.

3. What is the expected frequency of dimers in this table if all of them would be equally likely? Which are the dimers departing most from this expected value? Are they under-represented or over-represented in the sequence?
4. Fill in the table below of ***odd ratios*** (under a multinomial background model) of all dimers in this sequence.

	*A	*C	*G	*T
A*				
C*				
G*				
T*				

Table 2: Odd ratios of dimers in the *Lactococcus lactis subsp. lactis Il1403* genome.

5. Which are the most unusual dimers in this sequence? Are they unexpectedly frequent or rare? Argue whether your results depart from your initial analysis in question 3 and **why?**

4 ORF finding

1. Download the bacterial DNA of the *Bacillus cereus B4264* (Accession number `NC_011725` from GenBank or look at `BacillusCereus.fasta` from Moodle) and report its ***length*** in number of base pairs (bp).

2. ***How many ORFs*** do you find in this DNA? Compute how many ORFs are at least k nucleotides long, where k is a prescribed minimum length value. Report a table of results for $k = 10, 50, 100, 300, 500$.

   Important Note: funny as it is for computational scientists, the *so-called* ***universal code*** to translate a nucleotide sequence into amino acids (and, for the sake of our ORF finding task, to detect the locations of START and STOP codons) is **not exactly universal** (the beauty of life obviously lies in its *diversity*!). In this case[a] , specific START codons appear in such a bacterial genome, including `TTG, CTG, ATA, ATT, ATC, ATG, GTG` while the STOP codons are the usual ones (`TGA, TAA, TAG`). Fortunately, brave as you are to make sense out of this natural complexity, you will easily reconcile computation and biology **if** you pass the actual lists of start and stop codons as *parameters* to your ORF finding algorithm (on top of the sequence itself!).

3. Compute the distribution of ORF length in a NULL model. Consider in particular a NULL model estimated from a random permutation of the DNA of *Bacillus cereus B4264*. What is the maximal length of an ORF you find according to this random model? How many ORFs in the actual DNA are strictly longer than this length?

4. If you would consider a p-value of 1% as your significance threshold, ***how many candidate genes*** do you finally claim to have found in this DNA sequence based on your analysis in question 3.

5. What is the exact position and length of the longest candidate gene that you find according to this analysis? Specify also on which reading frame and whether you find it on the available sequence or its reverse complement. Is this gene related to some known function (quote your source)?

[a]See `http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c#SG11` for more details.

Hints

- This is the longest part of this assignment, **do not wait the last minute** to start working on it!
- Off the shelf software solutions are not always compatible with the peculiarities of ORF finding in specific genomes and, in particular, easily adaptable to specific lists of START and STOP codons.
- Some easily identifiable solutions (*e.g.* by "googling" a few minutes) also appeared not to be regularly maintained. This can cause problems whenever trying to access the data or running code relying on external libraries that have changed.
- Some available softwares also implement an ORF finding algorithm in a very inefficient way. The sequence to analyze here is relatively short. Yet very inefficient code can take an unreasonable time to execute (it should be significantly less than 1 second CPU time, if well implemented). You are welcome to evaluate your solution on even shorter sequences first (or increasing fractions of the sequence of interest...)
- For all the above reasons, we suggest you to implement your own ORF finding algorithm If you follow this suggestion, here are a few things to keep in mind.
  - The **`matchPattern`** function from the **`Biostrings`** Bioconductor R package may be useful to search for occurrences of a specific *pattern* in a (long) sequence.
    Installing Bioconductor R packages slightly differs from installing standard R packages. Please check `https://bioconductor.org/install/`.
  - There are 6 possible reading frames: 3 on the original sequence, 3 on its reverse complement.
  - From the computational viewpoint, finding ORFs on a sequence or its reverse complement, is the same problem! You shouldn't duplicate the code whenever unnecessary. Obviously, it does not mean that the ORFs need to lie at the same positions on both strands!
  - An ORF is defined between a START and a STOP codon on the *same* reading frame.
  - After a given START, you may find one or several codon(s) coding for `Methionine` before finding the next STOP. Those additional `Met` codons do not define true starts of ORFs: an ORF is always starting at the first START found when reading the sequence or the first START after a previous STOP. In other words, an ORF is a longest DNA stretch between a START and a STOP, on the same reading frame (!), without being interrupted by another STOP.
  - An efficient algorithm typically requires to identify first all possible start and stop positions (for the various start/stop codons of the code) and to *sort* them by increasing positions along the sequence. The rest of the algorithm needs to make good use of this initial sorting.

Results

Upload on Moodle a **zip archive file**, for this assignment, containing

- a PDF report including your answers to all questions in a frame in the present document. Your report should mention any software resource you have been reusing (typically publicly available R packages).
- the R code you have been writing for this assignment.

Submitting the information requested above for grading your work implies that each member of your group is accepting the *anti-plagarism* policy summarized below.

I hereby certify that the results and code that I will submit for this project is coming from my own work and that of my teammate (if any in my group). The submitted works will not be (even partial) copy/paste from information found on the internet or from the work of other groups.

I also certify that I will not distribute any answer or code related to these projects, in person or on any repository (github, bitbucket, Facebook groups, ...) accessible to anybody beyond my teammate, even after the deadlines.

Any violation of the above statements will be considered as cheating and will be reported as such to the President of the Jury.