Bioinformatics and Computational Biology with Biopython

Michiel J.L. de Hoon[1] mdehoon@ims.u-tokyo.ac.jp
Brad Chapman[2] chapmanb@uga.edu
Iddo Friedberg[3] idoerg@burnham.org

[1] Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokane-dai, Minato-ku, Tokyo 108-8639, Japan
[2] Plant Genome Mapping Laboratory, University of Georgia, Athens, GA 30602, USA
[3] The Burnham Institute, 10901 North Torrey Pines Road, La Jolla, CA 92037, USA

Keywords: Python, scripting language, open source

1 Introduction

In recent years, high-level scripting languages such as Python, Perl, and Ruby have gained widespread use in bioinformatics. Python [3] is particularly useful for bioinformatics as well as computational biology because of its numerical capabilities through the Numerical Python project [1], in addition to the features typically found in scripting languages. Because of its clear syntax, Python is remarkably easy to learn, making it suitable for occasional as well as experienced programmers. The open-source Biopython project [2] is an international collaboration that develops libraries for Python to facilitate common tasks in bioinformatics.

2 Summary of current features of Biopython

Biopython contains parsers for a large number of file formats such as BLAST, FASTA, Swiss-Prot, PubMed, KEGG, GenBank, AlignACE, Prosite, LocusLink, and PDB. Sequences are described by a standard object-oriented representation, creating an integrated framework for manipulating and analyzing such sequences. Biopython enables users to automatically interact with tools such as BLAST, Clustalw, EMBOSS, and E-Cell, and provides numerical routines for clustering gene expression data.

3 Example scripts

3.1 Running BLAST

After starting Python with the command `python`, we first import the relevant Biopython packages:

```
from Bio.Blast import NCBIWWW
from Bio import Fasta
```

and use Biopython to parse the FASTA file containing the sequence:

```
file_for_blast = open('mysequence.fasta', 'r')
fasta_iterator = Fasta.Iterator(file_for_blast)
fasta_record = fasta_iterator.next()
```

Now we run BLAST, retrieve the results from NCBI, and parse the BLAST output:

```
blast_results = NCBIWWW.blast('blastn', 'nr', fasta_record).read()
blast_record = NCBIWWW.BlastParser().parse_str(blast_results)
```

We report all alignments in the BLAST output with an expected value less than some threshold:

```
E_VALUE_THRESH = 0.1
for alignment in blast_record.alignments:
    for hsp in alignment.hsps:
        if hsp.expect < E_VALUE_THRESH:
            print '****Alignment****'
            print 'sequence:', alignment.title
            print 'length:', alignment.length
            print 'expected value:', hsp.expect
            print hsp.query[0:75] + '...'
            print hsp.match[0:75] + '...'
            print hsp.sbjct[0:75] + '...'
```

This script will print a total of four alignments, starting with

```
****Alignment****
sequence: >gb|BT005584.1| Arabidopsis thaliana clone U50435 putative cold acclim
ation protein
homolog (At4g37220) mRNA, complete cds
length: 640
e value: 0.034
gctatttacttgttgatattggatcgaacaaactggagaaccaa...
||||||||  |||||||||||||||| || ||||||| ||||||...
gctatttatctgttgatattggatcgtaccaactggaaaaccaa...
****Alignment****
sequence: >dbj|AP006108.1| Lotus corniculatus var. japonicus genomic DNA ...
```

3.2 Clustering gene expression data

Here, we read the gene expression data from a file, use k-means clustering to divide them into $k = 5$ clusters, and save the results in an output file that can be read by Java TreeView for visualization.

```
from Bio.Cluster import *
my_expression_data = readdatafile("my_expression_data.txt")
data = my_expression_data[0]
gene_names = my_expression_data[2]
exp_names = my_expression_data[6]
clusters, centroids, error, nfound = kcluster(data, nclusters=5)
writeclusterfiles("my_clusters", data, gene_names, exp_names, geneclusters=clusters)
```

References

[1] Ascher, D., Dubois, P. F., Hinsen, K., Hugunin, J. and Oliphant, T. *Numerical Python*, Lawrence Livermore National Laboratory, 2001. http://www.pfdubois.com/numpy/.

[2] http://www.biopython.org

[3] http://www.python.org