Genome 559
Intro to Statistical and Computational Genomics
2009

Lecture 18b:
Biopython

Larry Ruzzo

(Thanks again to Mary Kuhner for many slides)

1 Minute Responses

Biopython is neat, makes me feel silly that we made programs earlier in the class to do the same type of stuff, but it's all about the learning I suppose...

Yes, I deliberately had you do similar things, just for demystification

Today's class would have been great if Biopython actually worked. I'm not sure about the uses of it because the impromptu display of your code. It seems like it will be a very useful tool in the future though.

Biopython sounds really cool... looking forward to playing with it next class.

Excited to get my hands dirty w/ biopython. I am concerned about installing it on my primary computer b/c I'm not sure I have admin privileges - I'll look into this before Thursday.

Biopython seems intuitive. Plugging the pedigree example into the LOD formula helped me understand it better. Without doing examples on a formula makes it very difficult for me to understand what is going on. So, after we learn one new formula, doing one quick example helps.

It was interesting to look at the mechanics of the Bio.Seq class.

Biopython looks great, can't wait to try it. Genetics review was fine. I'll have to do some LOD calculation examples.

I know it wasn't planned, but I'm not sure I gained a lot from going through Biopython.

I looked at the homework and had a panic attack.

Sigh...

Biopython

Biopython is tool kit, not a program–a set of Python modules useful in bioinformatics

Features include:

- Sequence class (can transcribe, translate, invert, etc)
- Parsing files in different database formats
- Interfaces to progs/DBs like Blast, Entrez, PubMed
- Code for handling alignments of sequences
- Clustering algorithms

Useful tutorials at http://biopython.org

Making Biopython available on your computer

Runs on Windows, MaxOSX, and Linux

http://biopython.org/DIST/docs/install/installation.html

But may require "Admin" privileges

1.49 is latest; works with Python 2.5 -2.6 (& 3?)

Sequence class

```
>>> from Bio.Seq import Seq # sequence class
>>> myseq = Seq("AGTACACTGGT")
>>> myseq.alphabet
Alphabet()
>>> print myseq.tostring()
AGTACACTGGT
```

More functionality than a plain string

```
>>> myseq
Seq('AGTACACTGGT', Alphabet())
>>> myseq.complement()
Seq('TCATGTGACCA', Alphabet())
>>> myseq.reverse_complement()
Seq('ACCAGTGTACT', Alphabet())
```

A sequence in a specified alphabet

```
>>> from Bio.Seq inport Seq
>>> from Bio.Alphabet import IUPAC
>>> myseq=Seq('AGTACACTGGT',IUPAC.unambiguous_dna)
>>> myseq
Seq('AGTACACTGGT', IUPACUnambiguousDNA())
```

Transcribe/Translate

```
>>> from Bio import Transcribe
>>> mydna = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC",
... IUPAC.unambiguous_dna)
>>> myrna = mydna.transcribe()
>>> print myrna
Seq('GAUCGAUGGGCCUAUAUAGGAUCGAAAAUCGC',IUPACUnambiguousRNA())
>>> myprot = myrna.translate()
Seq('DRWAYIGSKI', ExtendedIUPACProtein())
>>> s2 = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG",
... IUPAC.unambiguous_rna)
>>>s2.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Parsing a database format

FASTA database file named "ls_orchid.fasta":

```
>gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTG
AATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTTGGG
....
from Bio import SeqIO
handle = open("ls_orchid.fasta")
for seqrecord in SeqIO.parse(handle, "fasta") :
  print seqrecord.id
  print seqrecord.seq
  print len(seqrecord.seq)
handle.close()

gi|2765658|emb|Z78533.1|CIZ78533
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTG ...',
   SingleLetterAlphabet())
740
...
```

Exercise 1

Modify the example above to print the GC% of each sequence, too.

Solution 1

```
from Bio import SeqIO
handle = open("ls_orchid.fasta")
for seqrec in SeqIO.parse(handle, "fasta"):
   print seqrec.id
   s = seqrec.seq
   print s
   print len(s),
   na = s.count('A')
   nc = s.count('C')
   ng = s.count('G')
   nt = s.count('T')
   print "GC%=",(ng+nc)*100.0/(na+nc+ng+nt)
handle.close()
```

Q1: there's also a Biopython func to calc gc%; can you find it?

Q2: Why did I *not* use (G+C)/len(s) ?

GenBank Format, too

```
from Bio import SeqIO
handle = open("ls_orchid.gbk")
for seq_record in SeqIO.parse(handle, "genbank") :
    print seq_record.id
    print repr(seq_record.seq)
    print len(seq_record)
handle.close()
```

This should give:

```
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGT
GG...CGC', IUPACAmbiguousDNA())
740
...
```

Exercise 2

Change above example to save the records in a list called `seqrecs`

Solution 2

```
from Bio import SeqIO
handle = open("ls_orchid.gbk")
seqrecs = []
for seq_record in SeqIO.parse(handle, "genbank") :
    seqrecs.append(seq_record)
    print seq_record.id
    print repr(seq_record.seq)
    print len(seq_record)
handle.close()
```

This should give:

```
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGT
GG...CGC', IUPACAmbiguousDNA())
740
...
```

And...

Feature Tables

```
>>>seqrecs[0]
SeqRecord(seq=Seq('CGTA...CGC', IUPACAmbiguousDNA()),
id='Z78533.1', name='Z78533', description='C.irapeanum
5.8S rRNA gene and ITS1 and ITS2 DNA.', dbxrefs=[])
>>> print seqrecs[0]
ID: Z78533.1
Name: Z78533
Description: C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA.
Number of features: 5
/sequence_version=1
/source=Cypripedium irapeanum
/taxonomy=['Eukaryota', ..., 'Cypripedium']
/keywords=['5.8S ribosomal RNA', ... 'ITS2']
/references=[<Bio.SeqFeature.Reference instance...]
/accessions=['Z78533']
/data_file_division=PLN
/date=30-NOV-2006
/organism=Cypripedium irapeanum
/gi=2765658
Seq('CGTAACAAGGTTTCCGTAGGTGA...CGC', IUPACAmbiguousDNA())
```

Extracting Features

(Lists of objects with dicts of lists of lists of dicts of ...Oh my!)

```
>>> seqrecs[0].annotations
{'sequence_version': 1, 'source': 'Cypripedium
irapeanum', 'taxonomy': ['Eukaryota', ... ... ... ...}

# it's a dictionary!  What keys does it have?

>>> seqrecs[0].annotations.keys()
['sequence_version', 'source', 'taxonomy', 'keywords',
'references', 'accessions', 'data_file_division', 'date',
'organism', 'gi']
# grab one dict entry
>>> seqrecs[0].annotations['keywords']
['5.8S ribosomal RNA', '5.8S rRNA gene', 'internal
transcribed spacer', 'ITS1', 'ITS2']

#It's a list!   We can index into it...

>>> seqrecs[0].annotations['keywords'][1]
'5.8S rRNA gene'
```

Searching GenBank

```
# This example & next require internet access

from Bio import GenBank
gilist = GenBank.search_for("Opuntia AND rpl16")
# (that's RPL-sixteen, not RP-one-one-six)

# gilist will be a list of all of the GenBank
# identifiers that match our query:
print gilist
['6273291', '6273290', '6273289', '6273287',
'6273286', '6273285', '6273284']
```

Searching GenBank

```
ncbidict = GenBank.NCBIDictionary("nucleotide", "genbank")
gbrecord = ncbidict[gilist[0]]
print gbrecord

LOCUS       AF191665 902 bp DNA PLN 07-NOV-1999
DEFINITION Opuntia marenae rpl16 gene; chloroplast gene for
           chloroplast product, partial intron sequence.
ACCESSION  AF191665
VERSION    AF191665.1 GI:6273291
...
```

Exercise 3: What kind of a thing is "gbrecord"? Is there other stuff hidden with it like annotations or feature tables? How do I access it?

Solution 3

```
>>> type(gbrecord)
<type 'str'>

# Aha, it's just a plain string.

>>> gbrecord
'LOCUS       AY851612                 892 bp    DNA
linear   PLN 10-APR-2007\nDEFINITION  Opuntia subulata
rpl16 gene, intron; chloroplast.\nACCESSION
AY851612\nVERSION     AY851612.1  GI:
57240072\nKEYWORDS    .\nSOURCE      chloroplast
Austrocylindropuntia subulata\n  ... ... ... ... ... '
```

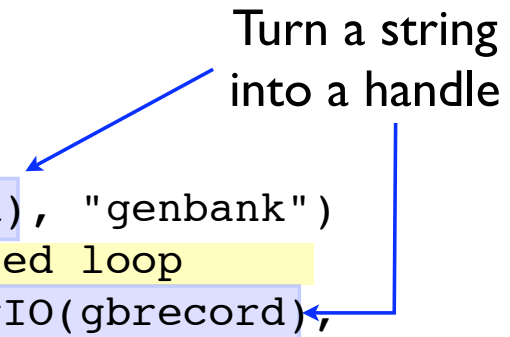
Can we get Biopython to parse it?

To parse a string

```
>>>SeqIO.parse(gbrecord, "genbank")
Traceback (most recent call last): blah blah blah...

# Oops, a string isn't a handle...

>>> import cStringIO
>>> SeqIO.parse(cStringIO.StringIO(gbrecord), "genbank")
<generator object at 0x5254b8>  ## Oops, need loop
>>> for rec in SeqIO.parse(cStringIO.StringIO(gbrecord),
... "genbank"):
...     print rec
...
ID: AY851612.1
Name: AY851612
Description: Opuntia subulata rpl16 gene, intron;
chloroplast.
Number of features: 3
/sequence_version=1
/source=chloroplast Austrocylindropuntia subulata  ...
```



How would I use Biopython?

Biopython is not a program itself; it's a collection of tools for Python bioinformatics programing

When doing bioinformatics, keep Biopython in mind

Browse the documentation;become familiar with its capabilities

Use help(), type(), dir() & other built-in features to explore

You might prefer it to writing your own code for:

- Defining and handling sequences and alignments
- Parsing database formats
- Interfacing with databases

You don't have to use it all! Pick out one or two elements to learn first

Code re-use

If someone has written solid code that does what you need, use it

Don't "re-invent the wheel" unless you're doing it as a learning project

Python excels as a "glue language" which can stick together other peoples' programs, functions, classes, etc.