Introduction to Bioinformatics

Gene Prediction and Related Python Constructs

Janyl Jumadinova
March 31, 2016
Gene Annotation

To computationally identify the genes in a genome, we must know:

- What do genes look like?
To computationally identify the genes in a genome, we must know:

- What do genes look like?
- How to find candidate genes?
To computationally identify the genes in a genome, we must know:

- What do genes look like?
- How to find candidate genes?
- How to decide which candidates are real?
What do Genes Look Like?

(A) prokaryotes

DNA

-35  -10  +1

Shine-Dalgarno

coding sequence

(ORF)

ATG

gene A

TAG

gene B

gene C

operon

(B) eukaryotes

DNA

transcription factor sites

TATA box

Inr

+1

core promoter

ATG

exon 1

intron 1

exon 2

intron 2

exon 3

coding sequence

TAG
How to Find Candidate Genes?
How to Find Candidate Genes?

Utilize Computational Approaches
Algorithms

- Alignment-based algorithms

- Sequence-based algorithms
  - Pattern-Matching Algorithm (e.g., find ORFs in a bacterial genome)

- Content-based algorithms

- Probabilistic algorithms
Algorithms

- Alignment-based algorithms
- **Sequence-based algorithms**
  - Pattern-Matching Algorithm (e.g., find ORFs in a bacterial genome)
 Algorithms

- Alignment-based algorithms
- **Sequence-based algorithms**
  - Pattern-Matching Algorithm (e.g., find ORFs in a bacterial genome)
- Content-based algorithms
- Probabilistic algorithms
How To Decide Which Are Real?

- Must distinguish real signals (unlikely patterns) from background noise (random matches)
How To Decide Which Are Real?

- Must distinguish real signals (unlikely patterns) from background noise (random matches)
- Which ORFs actually encode proteins?
- Which ORFs are false positives?
How To Decide Which Are Real?

- Hypothesis testing
  - Calculate probability of an event based on model (e.g., multinomial, Markov)
  - Make inferences (accept / reject H0)
- So we ask ...
  - What is the probability of particular ORF arising by chance?
  - If probability (p-value) is very low, then ORF may be real
How To Decide Which Are Real?

How to compute p-value of an ORF?

- Computationally (What is $P$(ORF of $k$ codons) by chance?)
- Randomization tests
How To Decide Which Are Real?

Create random sequence (with same statistical properties)

- Random permutation: Shuffle order of individual nucleotides / triplets
- Bootstrap: Sample with replacement
Gene Discovery

- Finding a gene is not the same as finding an ORF
- Identifying regions containing promoters can also help in gene recognition
Finding motifs in promoter regions

- Write a program that computes likely motifs (patterns) of length 6 (sixmers) in the promoter region (3,000 bases) of ten up-regulated genes in tomato fruits
Finding motifs in promoter regions: Steps

1. read in and concatenate ten sequences (use SeqIO)
2. compute frequencies $f$ of all sixmers (ex. ACTTTG 12)
3. compute $f_{rand}$ of each sixmer averaged over 1,000 shuffles of the sequence (ex. ACTTTG 3)
4. compute ratio $f/f_{rand}$ for each sixmer (ex. ACTTTG 4) print all sixmers having the above ratio $\geq 4$ (include their $f$, $f_{rand}$, and ratio)

FYI: I got 142 sixmers, but this number can change due to the randomization step
Read a sequence from fasta file with SeqIO package

```python
# see more at Biopython online tutorial
from Bio import SeqIO
# 'rU': open for reading using universal readline mode
handle = open("example.fasta", "rU")
for record in SeqIO.parse(handle, "fasta") :
    print record.id
    print record.seq
    print len(record.seq)
handle.close()
```
Shuffle numbers

```python
import numpy as np

shuffledNo = np.random.permutation(10)
```
**defaultdict**

- `defaultdict` type automates and simplifies initializations.
defaultdict

- **defaultdict** type automates and simplifies initializations
- A **defaultdict** works exactly like a normal dict, but it is initialized with a function ("default factory") that takes no arguments and provides the default value for a nonexistent key
defaultdict

- defaultdict type automates and simplifies initializations
- A defaultdict works exactly like a normal dict, but it is initialized with a function ("default factory") that takes no arguments and provides the default value for a nonexistent key
- A defaultdict will never raise a KeyError. Any key that does not exist gets the value returned by the default factory
**defaultdict**

- The `defaultdict` type automates and simplifies initializations.
- A `defaultdict` works exactly like a normal dict, but it is initialized with a function ("default factory") that takes no arguments and provides the default value for a nonexistent key.
- A `defaultdict` will never raise a `KeyError`. Any key that does not exist gets the value returned by the default factory.

```python
from collections import defaultdict

week = defaultdict(int)
```
Sorting a dictionary by value

`sorted(dict)`: sort the dictionary objects

`sorted` can also have a key parameter which returns a key to use for sorting purposes
**Sorting a dictionary by value**

- `sorted(dict)`: sort the dictionary objects
- `sorted` can also have a key parameter which returns a key to use for sorting purposes
- `dict.items()`: Return a copy of the dictionary's list of (key, value) pairs
- `dict.iteritems()`: Return an iterator over the dictionary's (key, value) pairs
Python supports the creation of anonymous functions (i.e. functions that are not bound to a name), using a construct called **lambda**.

- `lambda arg1, arg2, ...argN : expression using arguments`
Python supports the creation of anonymous functions (i.e. functions that are not bound to a name), using a construct called **lambda**.

- **lambda arg1, arg2, ...argN : expression using arguments**

```python
mydict = {'carl':40,'alan':2,'bob':1,'danny':3}

for key, value in sorted(mydict.iteritems(), key=lambda (k,v): (v,k)):
    print "%s: %s" % (key, value)
```
Setting the default factory to set makes the `defaultdict` useful for building a dictionary of sets.
Building a Dictionary of Sets

- Setting the default factory to set makes the `defaultdict` useful for building a dictionary of sets

```python
from collections import defaultdict
s = [('red', 1), ('blue', 2), ('red', 3), ('blue', 4), ('red', 1), ('blue', 4)]
d = defaultdict(set)
for k, v in s:
    d[k].add(v)
```