Quiz Section Week 7
May 9, 2017

HMM calculations by hand
A few quick Python notes
Recursion
Random numbers
Remember Needleman-Wunsch: determine the best “hidden” evolutionary relationship between two sequences

- Alignment score for a position is a function of a previous alignment score and a “transition” score
- Find the path through the matrix that has the best score
Viterbi: determine the likeliest hidden state sequence for an observed sequence

Observed sequence

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-rich</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-rich</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Likelihood for an “alignment” of hidden state to observed sequence is a function of likelihood of previous alignment and transition & emission probability
- Find the path through this matrix that has the highest probability
Dynamic programming to find the best path for Needleman-Wunsch and Viterbi

DP in equation form

- Align sequence $x$ and $y$.
- $F$ is the DP matrix; $s$ is the substitution matrix; $d$ is the linear gap penalty.

$$F(0,0) = 0$$

$$F(i,j) = \max \begin{cases} 
F(i-1,j-1) + s(x_i, y_j) \\
F(i-1,j) + d \\
F(i,j-1) + d 
\end{cases}$$

- “Align” observed sequence to state sequence

$$F(i,j) = \max \begin{cases} 
F(1,j-1)a(\pi_1, \pi_i)e(x_j, \pi_i) \\
F(2,j-1)a(\pi_2, \pi_i)e(x_j, \pi_i) \\
e tc. 
\end{cases}$$
Dynamic programming to find the best path for Needleman-Wunsch and Viterbi.

DP in equation form:

- Align sequence $x$ and $y$.
- $F$ is the DP matrix; $s$ is the substitution matrix; $d$ is the linear gap penalty.

\[
F(0,0) = 0
\]

\[
F(i,j) = \max\left\{\begin{array}{c}
F(i-1,j-1) + s(x_i, y_j) \\
F(i-1,j) + d \\
F(i,j-1) + d
\end{array}\right.
\]

“Align” observed sequence to state sequence:

- $F(1,j-1)a(\pi_1, \pi_i)e(x_j, \pi_i)$
- $F(2,j-1)a(\pi_2, \pi_i)e(x_j, \pi_i)$
- etc.
Dynamic programming to find the best path for Needleman-Wunsch and Viterbi

DP in equation form

- Align sequence $x$ and $y$.
- $F$ is the DP matrix; $s$ is the substitution matrix; $d$ is the linear gap penalty.

$$F(0,0) = 0$$
$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) + d \\ F(i, j-1) + d \end{cases}$$

“Align” observed sequence to state sequence

- $\pi_i$: A-rich
  - $0.4 \rightarrow 0.288 \rightarrow \cdots \rightarrow \cdots \rightarrow \cdots \rightarrow 0.00001$
- $\pi_i$: T-rich
  - $0.1 \rightarrow \cdots \rightarrow \cdots \rightarrow \cdots \rightarrow \cdots \rightarrow 0.0002$

F(1,j-1)$a(\pi_1, \pi_i)e(x_j, \pi_i)$
F(2,j-1)$a(\pi_2, \pi_i)e(x_j, \pi_i)$

etc.
Dynamic programming to find the best path for Needleman-Wunsch and Viterbi

DP in equation form

- Align sequence \( x \) and \( y \).
- \( F \) is the DP matrix; \( s \) is the substitution matrix; \( d \) is the linear gap penalty.

\[
F(0, 0) = 0 \\
F(i, j) = \max \left\{ \begin{array}{ll}
F(i-1, j-1) + s(x_i, y_j) \\
F(i-1, j) + d \\
F(i, j-1) + d
\end{array} \right.
\]

F(i, j) = max \[
\left\{ \begin{array}{l}
F(1, j-1)a(\pi_1, \pi_i)e(x_j, \pi_i) \\
F(2, j-1)a(\pi_2, \pi_i)e(x_j, \pi_i)
\end{array} \right.
\]

etc.

• “Align” observed sequence to state sequence
For forward-backward, we account for all paths instead of just the best path.

- What’s the probability that the $T$ at position 3 was emitted by the T-rich state?
- What’s the probability that any path goes through the T-rich state at the third position?
- Combine all paths that pass through that position/state pair.

$$P(\pi_i = k | x) = \frac{P(x, \pi_i = k)}{P(x)}$$

$$P(x, \pi_i = k) = \sum_{\pi_i = k} P(\pi | x)$$
Programming
Reminder: Tons of resources online for extra programming practice

• I still recommend this one:
  • http://interactivepython.org/runestone/static/thinkcspy/index.html

• You can use the help() function to learn about what other functions do, what arguments they need, etc.:
  
  >>> help(len)
  >>> help(open)
  >>> my_list = []
  >>> help(my_list.append)
A quick note on some useful list functions

```python
>>> my_list = [3,1,2]
>>> my_list[2]
1
>>> my_list[2:4]
[2, 3]
>>> my_list[1:]
[1, 2, 3]
>>> my_list[-1]
2

>>> my_list.append(4)
>>> my_list.remove(4)
>>> my_list.extend([4,5,6])
#compare with .append([4,5,6]

>>> my_list.sort()
>>> my_list.sort(reverse = True)

Both!
```
Structuring a program, keeping your code organized

- Remember last week we talked about how you can import functions from another script

**intersection_function.py:**

```python
def my_intersection(list1, list2):
    new_list = []
    for x in list1:
        if x in list2:
            new_list.append(x)
    return new_list
```

**calc_intersections.py:**

```python
#This line imports all function definitions from the file
#intersection_function.py
from intersection_function import *

#This runs all the code in intersection_function.py!
list_num = my_intersection([1,3,5,21],[5,4,19,21])
print list_num
```
We can put code to test or apply defined functions in its own section

Recall HW4

from __future__ import division

def euclidean_distance(point1, point2):
    #Calculate and return Euclidean distance here
    return(dist)

def manhattan_distance(point1, point2):
    #Calculate and return Manhattan distance here
    return sum_dist
Recursion

• Doug mentioned on Friday that the Forward-Backward algorithm is a *recursive* algorithm

• What does that mean?
Here's a puzzle: how to calculate the sum of a list (of any length) without for and while loops?

```python
def sumList(list1):
    # List sum calculation here

my_list = [0, 5, 3, 4, 8]
```

Hint: One way to show this mathematically

\[ \text{sum} = \left(0 + \left(5 + \left(3 + \left(4 + (8)\right)\right)\right)\right) \]
Recursively calculating the sum of a list

```python
def sumList(list1):
    if len(list1) > 1:
        return list1[0] + listsum(list1[1:]):
    else:
        return list1[0]
```

A *recursive* algorithm refers to itself – it calls itself iteratively *until reaching a base case*
In what order does our sum_list function actually run?

```
sum(1,3,5,7,9) = 1 +
sum(3,5,7,9) = 3 +
sum(5,7,9) = 5 +
sum(7,9) = 7 +
sum(9) = 9

25 = sum(1,3,5,7,9) = 1 + 24
sum(3,5,7,9) = 3 + 21
sum(5,7,9) = 5 + 16
sum(7,9) = 7 + 9
sum(9) = 9
```
A bad computer scientist joke

Did you mean: recursion

What's wrong with this recursive "algorithm"?
Write a recursive function to calculate a factorial

```python
def factorial(num):
    if # something:
        # recursion
    else:
        # base case
    return prod
```
Write a recursive function to calculate a factorial

def factorial(num):
    if num > 1:
        prod = num*factorial(num - 1)
    else:
        prod = num
    return prod
In what way(s) is the forward-backward algorithm recursive?
We build on all the forward-backward probabilities

\[
P(\pi_i = k \mid x) = \frac{P(x, \pi_i = k)}{P(x)} = \frac{f_{k,i} \ast b_{k,i}}{P(x)}
\]

\[
f_{k,i} = e_k(x_i) \sum_l f_{l,i-1} a_{lk}
\]

\[
b_{k,i} = \sum_l e_l(x_{i+1}) b_{l,i+1} a_{kl}
\]
Generating random numbers in Python

What are some situations where you’d want to generate random numbers?

In-class examples?

• Generating random sequences to create null distribution for sequence alignment
• A Markov chain that changes states probabilistically
random() returns a uniformly distributed random value between 0 and 1

- How can you convert this into a random coin flip with heads or tails?

```python
import random
r = random.random()
print r
0.261256363123
```
random() returns a uniformly distributed random value from [0,1)

- How can you convert this into a random coin flip with heads or tails?
- Throw a dart, call heads if dart lands between 0 and 0.5, tails if between 0.5 and 1
random() returns a uniformly distributed random value between 0 and 1

• How can you convert this into a random coin flip with heads or tails?

• Throw a dart, call heads if dart lands between 0 and 0.5, tails if between 0.5 and 1
Exercise: write a function to simulate a coin flip using random()

```python
import random

# return 'heads' or 'tails' with 50/50 odds
def coinflip():
```
Exercise: write a function to simulate a coin flip using random()

import random

# return heads or tails
def coinflip():
    v = random()
    if f > 0.5:
        return 'Tails'
    else:
        return 'Heads'
random() returns a uniformly distributed random value between 0 and 1

- How can you convert this into a die roll?
Exercise: write a function to simulate a die roll using random()

import random
# return 1,2,3,4,5, or 6 with equal odds
def dieroll():
Randomly shuffling a sequence of letters

How would you generate a random permutation of this sequence?

ATCGTCCTTAAGGATTACCATTGTCCTAGA
Randomly shuffling a sequence of letters

How would you generate a random permutation of this sequence?

ATCGTCCTTAAGGATTACCATTTGGCCTAGA