Biological Computation 20.181

Homework 5

In the last homework, you calculated the parsimony score for a particular tree and a set of sequences. In this assignment, you will write Python code to evaluate the Maximum Likelihood (ML) score given a tree and associated sequences. You will be provided with both the tree topology (including branch lengths) and the sequences of leaf nodes.

The Evolutionary Model. In general, a model of DNA sequence evolution gives the probability that a nucleotide, x, will change to another nucleotide, y (which may or may not be equal to x), over a certain period of time (actually over a certain distance, since evolutionary rates can vary from branch to branch).

In the Jukes-Cantor model of evolution, the probability of mutation to any of the three different nucleotides during a short time interval dt is a. Now consider the probability of a site being a certain base at time t, P(t). What is the probability of the site being the same nucleotide at time t + dt, P(t+dt)?

Certainly the probability is related to P(t) in some way, but what else? There is some chance that even if the base was correct at time t, it has mutated away since then (3*a*dt * P(t) will have changed -- this is the mutation rate times the fraction with the correct base at time t). But there will be some net gain from other bases that mutate to the correct base (a*dt * (1-P(t)) will change to the correct base -- this is the rate of change to a single base times the fraction of incorrect bases). Now write out the complete formula for P(t+dt).

Can you rearrange this formula to express the derivative of P(t) as a function of P(t)? Solve this first order differential equation for P(t). There are two possible initial conditions: either the base started out with the correct sequence, P(0)=1; or the base started out with a different sequence, P(0)=0. As t grows very large, of course, the probability that it will be any particular nucleotide is 1/4. Use these constraints to derive an expression for the probability that a base will have changed its sequence or not after a period of time t.

Coding the model. Add your result above to the codebase provided. It should have the following form:

```
def evoModel(x,y,distance):
#this function returns the probability of a change in sequence from
#x->y given an evolutionary distance
  import math #the math.exp() function may be useful
#
#...insert code here...
#
```

Your function should take two characters x and y, and a distance between them. The distance corresponds to rate*time in the above derivation, or a*t.

The Maximum Likelihood calculation. You will write a function that calculates the likelihood of a single position in your sequence alignment at a time. It will take the following form:

```
def ml(tree,pos):
#this function returns a dictionary containing the likelihood
#of each of the characters ['A','C','G','T']

  if tree['name'] != 'internal':
    likelihood = {}
```

```
    for n in ['A','C','G','T']:
      if [n] == tree['data'][pos]:
        likelihood[n] = 1
      else:
        likelihood[n] = 0
    return likelihood
#
#...insert code here...
#
```

Your function should return a dictionary with likelihood scores associated with each of the possible values (A,C,G,T) at the parent node. The stop case is given as an example. You will need to call the evoModel from above to calculate the probability of the two branches to the children.

You will need to consider each of the possible nucleotides at the parent node **and** the two children nodes -- a total of 64 different cases. In our Sankoff downpass algorithm, we did this and returned the scenario with the best score, but here we will add the likelihoods from all possibilities as we did in class. In addition, the simple Sankoff downpass algorithm we outlined in class did not do a very good job of reducing the number of redundant calculations (think about our two versions of the Fibonacci problem). Try to make sure your ml function visits each node only once.