AS.020.674 Spring 2017

Lab 11

Due May 5 in lab. Submit your answers as either a tgz file containing the python scripts and associated figures or as a single working Jupyter notebook file.

In [ ]:
```
# Standard iPython notebook preamble.
%matplotlib inline
from __future__ import division
import matplotlib
import numpy as np
from pylab import *
matplotlib.rcParams.update({"axes.formatter.limits": (-3,3)})
plotStyles={"markersize":10,"markeredgewidth":2.0,"linewidth":2.0}
```

Introduction.

In this lab we will explore models of sequence evolution. We will use a model to construct a set of hypothetical related sequences. Then we will use online alignment servers to see how well they are able to determine the relationships between the sequences we generated. This will give you an idea of the accuracy of the various alignment tools.

The first thing we need, is a similarity matrix. The file blosum50.txt contains a 21x21 similarity matrix where the first row and column in the file contains the single letter amino acid code for the entry or - for a gap. We would like to use the similarty matrix as a dictionary, so we will load it ourselves instead of using an external package like Numpy or Pandas.

In [ ]:
```
blosum50={}
f=open("blosum50.txt")
aa_codes=f.readline().split()
for line in f:
    values=line.split()
    row=values[0]
    values=values[1:]
    for index in range(0,len(aa_codes)):
        blosum50[row,aa_codes[index]]=float(values[index])
f.close()
```

Read through and try to understand the code above, it will help if you manually open the blosum50.txt file in a text editor and see how it is formatted. The for loop executes once for each item in a particular collection. In this case the loop is executed one time for each line in the file. The current line gets stored in the line variable.

The **blosum50** dictionary contains the score for each pair of amino acids. Check some of the scores in the dict by printing out their values:

In [ ]:
```
print blosum50['A','A']
print blosum50['A','C']
print blosum50['A','-']
```

Calculating transition probabilities.

We want to generate random sequences that are reasonable given a particular scoring matrix. The score matrix gives us the log-odds of a pair being seen in an alignment:

$$BLOSUM50(a,b) = 3 \cdot s(a,b) = 3 \cdot log_2[\frac{p(a,b)}{q(a) \cdot q(b)}]$$

Here, BLOSUM50(a,b) is the value of the pair a and b from the BLOSUM50 matrix, p(a,b) is the probability of amino acid a and b appearing together in a column, q(a) and q(b) are the absolute probabilities of finding amino acids a and b anywhere in the alignment. The constant 3 is a scaling constant to bring the values into a suitable range for representing the scores as integers and varies between the different BLOSUM matrices.

To generate a random sequence, we first need to calculate:

$$p(a,b) = q(a) * q(b) * 2^{(BLOSUM50(a,b)/3)}$$

We know the score from the matrix. We could estimate q(a) and q(b) from sequence databases, but for now we just assume all amino acids occur with equal frequency. For our purposes, a gap is counted as a disticnt amino acid.

Problem 1

1) Write a python script to create a new dictionary called **prob_ab** containing the *probabilities* of seeing every pair of amino acids, using the above formula and the **blosum50** scoring matrix we loaded from the file. You may also need the **aa_codes** list we created when loading the file.

2) What will be one possible effect on a generated alignment of our assumption about q(a) and q(b)?

Probability normalization.

Since we are using different q(a) and q(b) values then when the scores were originally generated originally, only the relative probabilities will be correct. This means that the probabilities will not sum to one. They therefore need to be renormalized so that the sum of all the probabilities is one. The following code will perform the renormalization.

In [ ]:
```
prob_sum=0.0
for key, value in prob_ab.items():
    prob_sum += value
for key, value in prob_ab.items():
    prob_ab[key] /= prob_sum
```

Now, we need to compute a transition matrix that contains the probability of changing from one amino acid to the other. Each row of the **prob_ab** matrix contains the relative probability of changing to abother amino acid, but we need to modify the matrix so that each row is an absolute probability, i.e., each row must sum to one. The following code will create the transition matrix **trans_ab**.

In [ ]:
```
trans_ab={}
for a in aa_codes:
    row_sum=0.0
    for b in aa_codes:
        row_sum += prob_ab[a,b]
    for b in aa_codes:
        trans_ab[a,b] = prob_ab[a,b]/row_sum
```

Problem 2

1) Write a python script that verifies that the probabilities in **prob_ab** have been normalized.

2) Write a python script that verifies that the transition probabilities in **trans_ab** have been renormalized, i.e., that prob_ab['A','A']+prob_ab['A','C']+... = 1.0, and that prob_ab['C','A']+prob_ab['C','C']+... = 1.0, etc.

Generating random sequences.

The dictionary **trans_ab** now contains the probability of transitioning from the first amino acid to the second amino acid, i.e., trans_ab['A','C'] gives the probability of mutating from an A to a C. Likewise, trans_ab['A','A'] gives the probability of an A remaining an A. Also, trans_ab['A','-'] gives the probability that an A will be deleted (replaced by a gap). We will use these probabilities to generate a random sequence. We will start from a master sequence:

In [ ]:
```
sequence=list("CGSWVMKEHLVRPYLNMKGLRGIQQFYKYCITYQNDHMKPMTVREPGPQTIHGACYV
NRKFTNQLPGWFGQKKGQLNVNVPNGWKADKWCVSFAPTNPSV")
```

First, we create a new FASTA file and write the original sequence into it. Each sequence is recorded in the FASTA file using two lines. The first line contains the > character followed by the name of the sequence. The second line contains the sequence data. In real FASTA files the sequence data can be split across multiple lines, but here we will not do so.

In [ ]:
```
f = open('seqs.fasta','w')
f.write(">original\n")
for r in sequence:
    f.write(r)
f.write("\n")
f.close()
```

Problem 3

1) Define a function named **mutate** that takes an amino acid character and returns a random amino acid (either a mutation, a deletion, or the original amino acid) distributed according to the transition probabilities in the **trans_ab** matrix. If a deletion is chosen, i.e., the amino acid transitions to a "-" character, the function should return the empty string "".

2) Write a python script that uses your **mutate** function to generate 100 random sequences based on original and appends them into the FASTA file.

Hint 1: You can use the command "f = open('seqs.fasta','a')" to open an existing file and append to its contents.

Hint 2: The name of each sequence in the FASTA file must be unique.

Problem 4

Use an alignment server to align the generated sequences. Usually you can just copy and paste the contents of the FASTA file into the server webpage.

Here are a few suggested servers to try:

https://www.ebi.ac.uk/Tools/msa/clustalo/ (https://www.ebi.ac.uk/Tools/msa/clustalo/)

https://www.ebi.ac.uk/Tools/msa/muscle/ (https://www.ebi.ac.uk/Tools/msa/muscle/)

https://www.ebi.ac.uk/Tools/msa/tcoffee/ (https://www.ebi.ac.uk/Tools/msa/tcoffee/)

http://mafft.cbrc.jp/alignment/server/ (http://mafft.cbrc.jp/alignment/server/)

Look at the aligned output. How well do you think it did? What parts did it get right and what parts did it get wrong? Briefly explain why you think this is a fair or an unfair test of an alignment server's accuracy? (Paste a few representative alignments into your answer.)