SCIE2100 Practical 5, Week 8

Phylogenetics

Aim

This practical will introduce phylogenetic analysis. You will develop both practical and theoretical skills, and gain hands-on experience relevant to the manipulation of sequence data, and analysis of their evolutionary relationship.

Files

For this practical you will need this guide, *sequence.py*, with all related files used in previous practicals, a new Python module *phylo.py* and a Python script *evodist.py*. In addition, the practical will reference a set of data files: *cyp_mouse.aln*, *cyp1a1.aln*, *cyp1a1.tree* and *28S_vertebrates.aln*.

Alignment and evolutionary distances

"Good" sequence alignments can indicate evolutionary relationships between sequences at the level of individual positions, i.e. that they originate from the same ancestral sequence. Insertions and deletions are indicated through the introduction of gaps. Alignments are valid for any sequence class (e.g. DNA or protein).

To investigate the evolutionary history of a set of sequences, you can measure evolutionary distance between sequences when placed in an alignment. With a measure in hand, it is possible to construct phylogenetic trees from a multiple sequence alignment. You will do both below.

The simplest way to measure the distance between two sequences is to calculate their fractional difference: the ratio of aligned non-identical symbols to their length. For two aligned sequences, computing the number of aligned non-identical symbols is a simple calculation: for each column in the alignment that is without a gap, check whether the two sequences have a matching symbol.

The Alignment class in *sequence.py* contains an incomplete function for calculating distances: calcDistances. It takes a measure argument, which can be any of the five different evolutionary distance measures (sometimes packaged as *evolutionary models*) described in Zvelebil and Baum (p267-276; fractional alignment difference—the p-distance, Poisson correct distance, Gamma distance, JC and Kimura). We can use any to measure the distances between all the pairs of sequences in a multiple sequence alignment. Below is an implementation of the p-distance (which you find in *sequence.py*):

```
distmat = numpy.zeros((len(self.seqs), len(self.seqs)))
# Loop through each pair of sequences
for i in range(len(self.seqs)):
    for j in range(i + 1, len(self.seqs)):
        seqA = self.seqs[i]; seqB = self.seqs[j]
        L = 0; D = 0
        for k in range(self.alignlen):
            if seqA[k] != '-' and seqB[k] != '-':
                L += 1
                if seqA[k] != seqB[k]:
```

```
            D += 1
    p = float(D)/L
    distmat[i, j] = distmat[j, i] = p
```

This code creates a distance matrix with an entry for all possible pairs of (aligned) sequences. Later we will try to make some sense of distances, but first we will use the distance matrix for protein sequences.

1. Open *cyp1a1.aln*. Determine the distances between all sequences and plot them as a heatmap like in the example code below (found in *evodist.py*). The identifiers in the file are Uniprot identifiers and they are composed of two parts: a gene identifier-like prefix, and a species identifier-like suffix. All of the proteins are of from the same gene family (namely cytochrome P450 class 1A1). Just so you know: the sequences are incomplete protein, and form part of a high-confidence (and non-gapped) alignment.

   ```
   import numpy as np
   import matplotlib.pyplot as plt
   aln = readClustalFile('cyp1a1.aln', Protein_Alphabet)
   d = aln.calcDistances('fractional')
   fig, ax = plt.subplots()
   ax.imshow(d, plt.cm.gray, interpolation='nearest')
   plt.yticks(np.arange(len(aln)), [s.name for s in aln])
   plt.title('Distances')
   plt.show()
   ```

   Submit (Q1): Print the map and explain what you see in terms of species organisation. (Do they fall into groups? Are the groups expected?)

2. Open *cyp_mouse.aln*. Similar story: determine the distances between all sequences and plot them as a heatmap. The identifiers in the file are again Uniprot identifiers, but this time you are interested in the gene identifier part.

   Submit (Q2): Print the map and explain what you see in terms of gene names. Answer the questions: Which families of cytochrome P450 seem to belong together? (Confirm observations made by looking at the map by using UniprotKB to check what functions they are involved with.)

Phylogenetic tree

The representation above implies groupings of sequences, but does not reveal their evolutionary origin clearly. Phylogenetic trees are used to represent and visualise the ancestral relationship between multiple genotypes. (Note that I did not say “species” or “genes” because we can use these trees for either.) The evolutionary distances between orthologous sequences can be used to construct a phylogenetic tree.

A distance matrix like that above is viewed as ‘additive’, that is we expect that the weights along the path between any two sequences sum to their distance. There are examples and information about methods for determining the best tree in Zvelebil and Baum (p276-307) and in Jones and Pevzner (p358-378). We will use UPGMA below. First: a look at file formats.

The Newick (or New Hampshire) tree format is a way of writing a tree in a simple text format. Each split in the tree is written as the two ends of the split in brackets,

separated by a comma. For instance, a simple tree, which has a bear and raccoon split off from a dog, would be written as

```
((raccoon, bear), dog);
```

Note that the format ends with a semi-colon. The length of each branch can also be specified using colons, like so:

```
((raccoon:0.20, bear:0.07):0.01, dog:0.25);
```

(This is part of the example tree shown in Figure 7.4 in Zvelebil and Baum.)

Go to http://lh3lh3.users.sourceforge.net/jstree.shtml, and paste the Newick tree data into the text box. Click in the widget area below to upload the data. You will then see your tree appear.

phylo.py contains an implementation of a binary tree PhyloTree suited to the representation of phylogenetic trees. The method readNewick reads a file with a tree on the Newick format and constructs an instance of PhyloTree with each node represented as a PhyloNode. Each of the leaves (and internal nodes) can be labelled, enabling you to reference sequences in a separate alignment, e.g.

```
tree = readNewick('cyp1a1.tree')
aln = readClustalFile('cyp1a1.aln', Protein_Alphabet)
for seq in aln:
    node = tree.findLabel(seq.name)
    a = tree.getAncestorsOf(node)
    print seq.name, 'has-distance', node.dist, 'from ancestor', a.label
```

Several methods are available to generate trees. Most are based on evolutionary models and/or distances (one notable exception is Maximum Parsimony, which we will look at later). Some generate a single tree and present that, including UPGMA and Neighbour-joining. Others consider many different tree topologies before presenting one tree or an aggregate, including Maximum Parsimony, Maximum Likelihood and Bayesian estimation-based methods. *phylo.py* contains an implementation of UPGMA (runUPGMA) that takes an alignment and a distance metric (same as calcDistances).

3. Load *28S_vertebrates.aln*, which contains an alignment of sequences taken from 28S rRNA of a number of vertebrates. (Ribosomal RNA gene regions like for 28S are available throughout many kingdoms of life and are highly conserved; hence they are often used for phylogenetic analysis.)

   Construct a phylogenetic tree of the sequences, by using runUPGMA. Try the p-distance (fractional). Then try at least two other distances. Poisson corrected distance was designed to account for multiple mutations that happen more extensively at greater evolutionary distances. Confirm this by observing the differences between distances using the heatmap you used previously.

   Submit (Q3): Print the tree for at least two distance metrics and explain what you see in terms of how species are placed in relation to one another. Print the heatmap of differences and try to answer: Which distance metric makes more biological sense? (Explain technically why it should work better, and find some indication in the maps that would support this observation. **Hint**: each metric tries to correct for

something that is missing in the simpler. The correction may be more obvious in *some* cases.)

4. Load *cyp_mouse.aln* again, this time to generate a tree with the distance metric that you choose for Q2.

   Submit (Q4): Print the tree. Identify the groups that you found in the heatmap of Q2. Identify additional groups that now are clearer and explain if they make sense.

Ancestral Sequence Reconstruction

To find the ancestral sequences, given the tree is called the *small parsimony problem* by Jones and Pevzner (section 10.10) or see Zvelebil and Baum (section 8.4, p300). They describe a maximum parsimony approach to character-based tree reconstruction, meaning that a tree with sequence 'characters' assigned at each of the nodes. The standard way of solving it is based on dynamic programming (that we have seen for sequence alignment). For the curious, the *large* parsimony problem involves finding the tree that gives the best parsimony score as well.

phylo.py contains a solution to the small parsimony problem (parsimony). Note that parsimony is a method for an already existing PhyloTree. So you have to get one first. And after the tree has been created, you need to assign the alignment to it, to make sure that leaves have sequences assigned to them, like so:

```
aln = readClustalFile('cyp_mouse.aln', Protein_Alphabet)
tree.putAlignment(aln)
newickBeforeParsimony = tree.strSequences(10, 15)
tree.parsimony()
```

5. Look at the sequences in *cyp_mouse.aln* again. Reconstruct the sequence at each of the ancestral nodes given a tree generated by UPGMA. You can print out the sequences in the tree in the Newick format, using the method strSequences in PhyloTree passing the start and end columns of the alignment, like so:

   ```
   print tree.strSequences(10, 15)
   ```

   Submit (Q5): Print the tree with gene names. What is the ancestral sequence of the proteins given the tree according to Maximum Parsimony? In your report discuss technical aspects of the method, and the outcomes of the analysis. Present the tree again, this time with an interesting but short section of the alignment, and explain how maximum parsimony works using that as an example.

References

- Jones NC and Pevzner PA (2004) Clustering and Trees (Chapter 10). *An Introduction to Bioinformatics Algorithms*. MIT Press.
- Zvelebil M and Baum JO (2007) Building phylogenetic trees (Chapter 8). *Understanding Bioinformatics*. Garland.

Assessment

Respond to the problems marked with **Submit** above tasks in a **PDF or Word document** and submit it to Blackboard as an **attached file** to the assessment by the due date.