h4_phylogenetics

October 7, 2019

0.0.1 1)Build UPGMA and NJ trees for this alignment, add both trees to the report.

1 we will use the alignment obtained using prank taking into account codons

All algorithms are designed as worker subclasses of a base class TreeConstructor. All constructors have the same method build_tree that accept a MultipleSeqAlignment object to construct the tree. Currently there are two types of tree constructors: DistanceTreeConstructor and ParsimonyTreeConstructor."'

[45]:
```
from Bio.Phylo.TreeConstruction import DistanceCalculator
from Bio import AlignIO
aln = AlignIO.read('/home/sedreh/ITMO/semester3/Molecular_phylogenetic/
 ↪homework_4/data/SUP35_aln.best.fas', 'fasta')
print (aln)
```

```
SingleLetterAlphabet() alignment with 10 rows and 2460 columns
ATGTCA------------------------------GACCAA--...AAT SUP35_Kla_AB039749
ATGTCGGAGGAAGATCAAATTCAATCGCAAGGCAACGACCAAGG...---
SUP35_Agos_ATCC_10895_NM_211584
ATGTCG------------------------------GAT-----...---
SUP35_Scer_74-D694_GCA_001578265.1
ATGTCG------------------------------GAT-----...--- SUP35_Sbou_unique28_CM003560
ATGTCG------------------------------GAT-----...--- SUP35_Scer_beer078_CM005938
ATGTCG------------------------------GAT-----...--- SUP35_Spar_A12_Liti
ATGTCT------------------------------GAT-----...--- SUP35_Smik_IFO1815T_30
ATGTCT------------------------------GAT-----...---
SUP35_Sarb_H-6_chrXIII_CM001575
ATGTCA------------------------------GAT-----...--- SUP35_Skud_IFO1802T_36
ATGTCT------------------------------GAT-----...---
SUP35_Seub_CBS12357_chr_II_IV_DF968535
```

2 DistanceTreeConstructor:

The DistanceTreeConstructor has two algorithms: UPGMA (Unweighted Pair Group Method with Arithmetic Mean) and NJ (Neighbor Joining). Both algorithms construct trees based on a distance matrix "identity" is the name of the model (scoring matrix) to calculate the distance

[46]:
```
calculator = DistanceCalculator('identity')
dm = calculator.get_distance(aln)
dm
```

[46]:
```
DistanceMatrix(names=['SUP35_Kla_AB039749', 'SUP35_Agos_ATCC_10895_NM_211584',
'SUP35_Scer_74-D694_GCA_001578265.1', 'SUP35_Sbou_unique28_CM003560',
'SUP35_Scer_beer078_CM005938', 'SUP35_Spar_A12_Liti', 'SUP35_Smik_IFO1815T_30',
'SUP35_Sarb_H-6_chrXIII_CM001575', 'SUP35_Skud_IFO1802T_36',
'SUP35_Seub_CBS12357_chr_II_IV_DF968535'], matrix=[[0], [0.3825203252032521, 0],
[0.3674796747967479, 0.39634146341463417, 0], [0.3670731707317073,
0.39593495934959344, 0.0008130081300813385, 0], [0.37073170731707317,
0.40975609756097564, 0.025203252032520274, 0.025203252032520274, 0],
[0.36056910569105693, 0.3898373983739838, 0.060975609756097615,
0.060975609756097615, 0.08333333333333337, 0], [0.3556910569105691,
0.38699186991869916, 0.10284552845528461, 0.10284552845528461,
0.12276422764227646, 0.09024390243902436, 0], [0.359349593495935,
0.39593495934959344, 0.12032520325203255, 0.12032520325203255,
0.14065040650406502, 0.1077235772357723, 0.11016260162601621, 0],
[0.36626016260162597, 0.3967479674796748, 0.12479674796747964,
0.12560975609756098, 0.14349593495934965, 0.11829268292682926,
0.12276422764227646, 0.11869918699186988, 0], [0.3556910569105691,
0.40569105691056906, 0.14552845528455283, 0.14552845528455283,
0.14349593495934965, 0.13455284552845526, 0.1353658536585366,
0.12113821138211378, 0.12967479674796745, 0]])
```

[47]:
```
print(dm)
```

```
SUP35_Kla_AB039749      0
SUP35_Agos_ATCC_10895_NM_211584 0.3825203252032521      0
SUP35_Scer_74-D694_GCA_001578265.1      0.3674796747967479
0.39634146341463417     0
SUP35_Sbou_unique28_CM003560    0.3670731707317073      0.39593495934959344
0.0008130081300813385   0
SUP35_Scer_beer078_CM005938     0.37073170731707317     0.40975609756097564
0.025203252032520274    0.025203252032520274    0
SUP35_Spar_A12_Liti     0.36056910569105693     0.3898373983739838
0.060975609756097615    0.060975609756097615    0.08333333333333337     0
SUP35_Smik_IFO1815T_30  0.3556910569105691      0.38699186991869916
0.10284552845528461     0.10284552845528461     0.12276422764227646
0.09024390243902436     0
SUP35_Sarb_H-6_chrXIII_CM001575 0.359349593495935       0.39593495934959344
0.12032520325203255     0.12032520325203255     0.14065040650406502
0.1077235772357723      0.11016260162601621     0
SUP35_Skud_IFO1802T_36  0.36626016260162597     0.3967479674796748
0.12479674796747964     0.12560975609756098     0.14349593495934965
0.11829268292682926     0.12276422764227646     0.11869918699186988     0
SUP35_Seub_CBS12357_chr_II_IV_DF968535  0.3556910569105691
0.40569105691056906     0.14552845528455283     0.14552845528455283
```

```
0.14349593495934965     0.13455284552845526     0.1353658536585366
0.12113821138211378     0.12967479674796745     0
        SUP35_Kla_AB039749      SUP35_Agos_ATCC_10895_NM_211584
SUP35_Scer_74-D694_GCA_001578265.1      SUP35_Sbou_unique28_CM003560
SUP35_Scer_beer078_CM005938     SUP35_Spar_A12_Liti     SUP35_Smik_IFO1815T_30
SUP35_Sarb_H-6_chrXIII_CM001575 SUP35_Skud_IFO1802T_36
SUP35_Seub_CBS12357_chr_II_IV_DF968535
```

[18]:
```
#from Bio.Phylo.TreeConstruction import DistanceCalculator,␣
 ↪DistanceTreeConstructor
#constructor = DistanceTreeConstructor(calculator, 'nj')
#tree = constructor.build_tree(aln)
#print(tree)
```

3 use our own DistanceMatrix directly to build trees

[48]:
```
from Bio.Phylo.TreeConstruction import DistanceCalculator,␣
 ↪DistanceTreeConstructor
constructor = DistanceTreeConstructor()
tree_nj = constructor.nj(dm)
print(tree_nj)
```

```
Tree(rooted=False)
    Clade(branch_length=0, name='Inner8')
        Clade(branch_length=0.003163109756097557, name='Inner6')
            Clade(branch_length=0.06587059620596204,
name='SUP35_Seub_CBS12357_chr_II_IV_DF968535')
            Clade(branch_length=0.12356029810298101, name='Inner1')
                Clade(branch_length=0.17411077235772357,
name='SUP35_Kla_AB039749')
                Clade(branch_length=0.2084095528455285,
name='SUP35_Agos_ATCC_10895_NM_211584')
        Clade(branch_length=0.002223069105691068, name='Inner7')
            Clade(branch_length=0.054026930894308905,
name='SUP35_Sarb_H-6_chrXIII_CM001575')
            Clade(branch_length=0.00984502032520325, name='Inner5')
                Clade(branch_length=0.047522865853658544,
name='SUP35_Smik_IFO1815T_30')
                Clade(branch_length=0.017924288617886155, name='Inner4')
                    Clade(branch_length=0.027865853658536582,
name='SUP35_Spar_A12_Liti')
                    Clade(branch_length=0.031686991869918774, name='Inner3')
                        Clade(branch_length=0.01988482384823846,
name='SUP35_Scer_beer078_CM005938')
                        Clade(branch_length=0.004911924119241143, name='Inner2')
                            Clade(branch_length=0.00043554006968642145,
```

```
name='SUP35_Sbou_unique28_CM003560')
                            Clade(branch_length=0.0003774680603949171,
name='SUP35_Scer_74-D694_GCA_001578265.1')
        Clade(branch_length=0.06208079268292681, name='SUP35_Skud_IFO1802T_36')
```

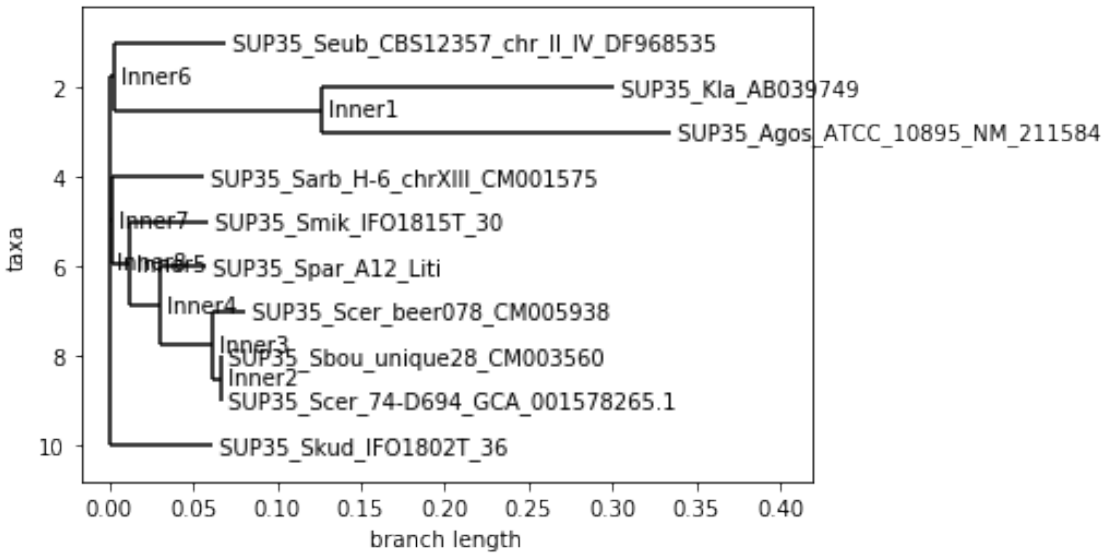
4 we can write the tree in xml file

[55]:
```
from Bio import Phylo
Phylo.write([tree_nj], 'tree-nj.xml', 'phyloxml')
```

[55]:
```
1
```

5 we can draw the tree

[56]:
```
import pylab
Phylo.draw(tree_nj)
pylab.savefig('tree_nj.png')
```




```
<Figure size 432x288 with 0 Axes>
```

[57]:
```
tree_upgma = constructor.upgma(dm)
print(tree_upgma)
```

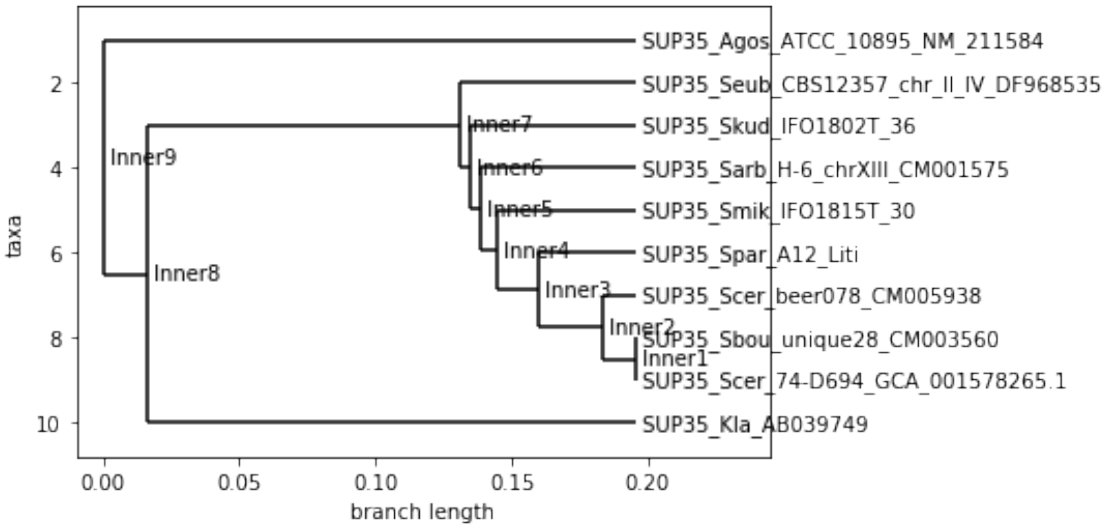
```
Tree(rooted=True)
    Clade(branch_length=0, name='Inner9')
```

```
        Clade(branch_length=0.19575155614837397,
name='SUP35_Agos_ATCC_10895_NM_211584')
        Clade(branch_length=0.01607199568089429, name='Inner8')
            Clade(branch_length=0.11493743648373986, name='Inner7')
                Clade(branch_length=0.06474212398373982,
name='SUP35_Seub_CBS12357_chr_II_IV_DF968535')
                Clade(branch_length=0.003931656504065031, name='Inner6')
                    Clade(branch_length=0.06081046747967479,
name='SUP35_Skud_IFO1802T_36')
                    Clade(branch_length=0.003493394308943104, name='Inner5')
                        Clade(branch_length=0.05731707317073169,
name='SUP35_Sarb_H-6_chrXIII_CM001575')
                        Clade(branch_length=0.006554878048780466, name='Inner4')
                            Clade(branch_length=0.05076219512195122,
name='SUP35_Smik_IFO1815T_30')
                            Clade(branch_length=0.014684959349593477,
name='Inner3')
                                Clade(branch_length=0.036077235772357746,
name='SUP35_Spar_A12_Liti')
                                Clade(branch_length=0.02347560975609761,
name='Inner2')
                                    Clade(branch_length=0.012601626016260137,
name='SUP35_Scer_beer078_CM005938')
                                    Clade(branch_length=0.012195121951219468,
name='Inner1')
Clade(branch_length=0.00040650406504066927, name='SUP35_Sbou_unique28_CM003560')
Clade(branch_length=0.00040650406504066927,
name='SUP35_Scer_74-D694_GCA_001578265.1')
            Clade(branch_length=0.17967956046747968, name='SUP35_Kla_AB039749')
```

[58]:
```
Phylo.write([tree_upgma], 'tree-upgma.xml', 'phyloxml')
```

[58]:
```
1
```

[60]:
```
import pylab
Phylo.draw(tree_upgma)
pylab.savefig('tree-upgma.png')
```




```
<Figure size 432x288 with 0 Axes>
```

some explanation to understand UPGMA and NJ trees.

6 The most important practical issues: UPGMA provides rooted tree as a result, while NJ unrooted, and you have to take care proper rooting the NJ tree afterward.

Also, UPGMA is regarded as an unreliable method, so I would prefer to use UPGMA. Bootstrapping is a re-sampling method used in phylogenetics to estimate the reliability (or by other opinions the peroduceability) of individual branching points of a tree: As a first approach, bootstrap values tell you how much you can "trust" a branch on a phylogenetic tree. The bigger the better. Values over 0.9 (90%) are usually OK. (But that depends on your data and research question too.) Values below 0.6 (60%) are usually no-go. If data is sparse or very noisy, bootstrap values will be low.

simplely diff: UPGMA: equal rate among branches >> equal depth from root NJ: allow diff rate among branches >> diff depth from root

similar : both distance based method

Clustering methods such as the unweighted-pair group method with arithmetic means (UPGMA) use a sequential clustering algorithm. A tree is built in a stepwise manner, by grouping sequences or groups of sequences – usually referred to as operational taxonomic units (OTUs) that are most similar to each other; that is, for which the genetic distance is the smallest. When two OTUs are grouped, they are treated as a new single OTU. From the new group of OTUs, the pair for which the similarity is highest is again identified, and so on, until only two OTUs are left.

6.0.1 2)Do they support the choice of Agos + Kla as an external group?

outgroup: a sequence for which it is "known" that all the other sequences are more closely related among themselves than to the outgroup. Then, we know that the ancestral sequence lies on the evolutionary path between the outgroup and all the other sequences.

A)UPGMA- this is the simplest method for constructing trees, assumes the same evolutionary speed for all lineages. all leaves have the same distance from the root (creates ultrametric tree)

B)Neighbor-joining- taking the two closest nodes of the tree and defines them as neighbors; you keep doing this until all of the nodes have been paired together

UPGMA is aaplied to phylogenetics to construct ultrametric trees. Ultrametricityis satisfied when two of the three distances under consideration are equal and as large (or larger) as the third one. Ultrametric trees are rooted trees in which all the end nodes are equidistant from the root of the tree, which is only possible by assuming a molecular clock.

But NJ trees do not assume a molecular clock and they are unrooted tree. One of the rooting technique consists of including an outgroup in the data set and placing the root at the midpoint of the branch that connects the outgroup with the rest (ingroup) of the taxa.

Because the polarity (direction) of character change can be determined only on a rooted phylogeny, the choice of outgroup is essential for understanding the evolution of traits along a phylogeny

6.0.2 3) Model selection: To select appropriate model for our data, we can use jmodeltest (https://github.com/ddarriba/jmodeltest2, web interface is down right now), or ModelFinder (part of IQ-Tree, http://iqtree.cibiv.univie.ac.at/ , choose “Model selection” tab). Which model works best for the data?

I used jmodeltest and Found Best-fit model according to BIC: TIM3+F+I+G4.

PartitionFinder2 is a program for selecting best-fit partitioning schemes and models of evolution for nucleotide, amino acid, and morphology alignments.

6.0.3 4.1 Prepare the alignment for analysis in PartitionFinder

I did it in R with the code fastaobject<-seqinr::read.fasta("/home/sedreh/ITMO/semester3/Molecular_phylogen
ape::write.dna(fastaobject, "phyfile.phy", nbcol=1,colsep="", colw=1000000)

6.0.4 4.2 Based on a configuration file template in the folder with the lesson materials, create a configuration file that forces PartitionFinder to analyze the entire sequence as a whole.

Based on the manuall I uncommented following parts:

alignment = "SUP35_aln.best.phy"
branchlengths = linked;
models = all;
model_selection = bic;
And [data_blocks]

Does the suggested model remains the same? After running in the folder of analysis I found best_scheme file that explain about best model! As I understand I have GTR+I+G as a best model:

in number 3 the best model was TIM3+F+I+G4! so it changed!

6.0.5 4.3 Based on the same template, create a configuration file that compares four partition schemes:

the entire sequence (without partition);
two separate (consecutive) domains;
each of the three positions of the codons individually for the entire sequence;
each of the three codon positions individually in two separate domains.
Run the PartitionFinder with this configuration file (this is one run, not 4) on the same alignment. Which data partitioning scheme is the best?
for this part I need to talk with you!

[ ]: