bio-hw3-q3b

November 26, 2017

1 Fundamentals of Bioinformatics

1.1 Homework Assignment 3

1.1.1 Sepehr Torabparhiz

1.1.2 93100774

1.2 Q3 part B

First few cells contain codes for Q3 part A. Scroll down to see the codes for part B.

In [1]:
```
from math import inf
from Bio import SeqIO, pairwise2 # BioPython Library
```

In [2]:
```
def find_max_pair(sim_list):
    max_sim = -inf
    for i in range(len(sim_list)):
        for j in range(len(sim_list)):
            if i != j:
                if max_sim < sim_list[i][j]:
                    max_sim = sim_list[i][j]
                    max_coords = [i, j]

#    print('min nodes')
    return max_coords, max_sim
```

In [3]:
```
def upgma(similarity_list, node_names):
    while len(similarity_list) > 1:
        max_coords, max_sim = find_max_pair(similarity_list)

        # Update scores
        for i in range(len(similarity_list)):
            if not(similarity_list[max_coords[0]][i] == -inf or similarity_list[max_coor
                new_score = (similarity_list[max_coords[0]][i] + similarity_list[max_coo
                similarity_list[max_coords[0]][i] = new_score
                similarity_list[i][max_coords[0]] = new_score

        # Remove row and column the second one in min pair
```

```
            similarity_list.pop(max_coords[1])
            for row in similarity_list:
                row.pop(max_coords[1])
            # Change the name list
            node1 = node_names[max_coords[0]]
            node2 = node_names[max_coords[1]]
            node_names[max_coords[0]] = '(' + node_names[max_coords[0]] + ',' + node2 + ')'
            node_names.pop(max_coords[1])

            merge_set_name = '(' + node1 + ',' + node2 + ')'
            left_right_similarity = int(abs(max_sim))
            print(str(left_right_similarity) + ' ' + merge_set_name)
```

```
In [4]: def process_input():
            n = int(input())

            node_names = []
            for i in range(n):
                node_names.append(input())

            similarity_list = []
            for i in range(n):
                similarity_list.append(input())

            return node_names, similarity_list
```

1.2.1 As mentioned in the assignment PDF, I found the mitochondrial 12S rRNA for each species from NCBI which you can see below. I also saved each sequence as a FASTA file so I could read them with BioPython

```
In [5]: # mitochondrial 12S rRNA
        brown_bear = 'TAAAGGTTTGGTCCTAGCCTTCCCATTAGCTACTAACAAGATTACACATGTAAGTCTCCGCGCTCCAGTGAAAA
        black_bear = 'CAAAGGTTTGGTCCTGGCCTTCCTATTAGCTGCTAACAAGATTACACATGTAAGTCTCCGCGCCCCAGTGAAAA
        racoon = 'TAAAGGTTTGGTCCTGGCCTTCCTATTAGTTCTTGACAAATTTACACATGCAAGTCTCCACATCCCAGTGAAATATGC
        red_panda = 'CAAAGGTTTGGTCCTAGCCTTCCCGTTAGTTCTTAATAAAATTACACATGCAAGTATCTACACCCCAGTGAAAAT
        giant_panda = 'TAAAGGTTTGGTCCTAGCCTTCCTATTAGCCATTAACAAGATTACACATGTAAGTCTCCACGCTCCAGTGAAA
```

1.2.2 A similarity matrix is generated with zero for each entry

```
In [6]: sim_mat = [[0 for i in range(5)] for j in range(5)]
        sim_mat

Out[6]: [[0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0]]
```

1.2.3 Reading Inputs

Each sequence is read from the files and then is stored in *seq_list*. A list of names is also created to be given to Q3 part A code.

```
In [7]: brown = SeqIO.read('brown-bear.faa.txt', 'fasta')
        black = SeqIO.read('black-bear.faa.txt', 'fasta')
        racoon = SeqIO.read('racoon.faa.txt', 'fasta')
        red = SeqIO.read('red-panda.faa.txt', 'fasta')
        giant = SeqIO.read('giant-panda.faa.txt', 'fasta')
        names = ['brown-bear', 'black-bear', 'racoon', 'red-panda', 'giant-panda']
        seq_list = [brown, black, racoon, red, giant]
```

1.3 Filling the Similarity Matrix

1.3.1 Choosing the Right Alignment Method

I use global alignment to align sequences. The reason I choose global alignment instead of local alignment was that we want to compare entire sequences and these input sequences are similar both in length, function and content as we are compaing same rRNA in different species. We know that rRNAs are highly conserved among different species.

1.3.2 BioPython's Pairwise Alignment

I use BioPython's *pairwise2.align.globalxs* function to align each two sequences. The two letters after *global* in *globalxs* specify match score and gap penalties respectively. *x* means each match has a score of 1 while a mismatch has a score of zero. *s* means open and extend gap penalties are the same in both sequences. The two -1 arguments given to the *pairwise2.align.globalxs* signify that open and extend penalties are both -1.

pairwise2.align.globalxs returns a list of optimal alignments, each containing the aligned sequences, the score, the start and end of alignment position(which are zero and length of alignment in global alignments). We only need the score of one of the computed alignments, which is why *[0][2]* subscript is added after the alignment function.

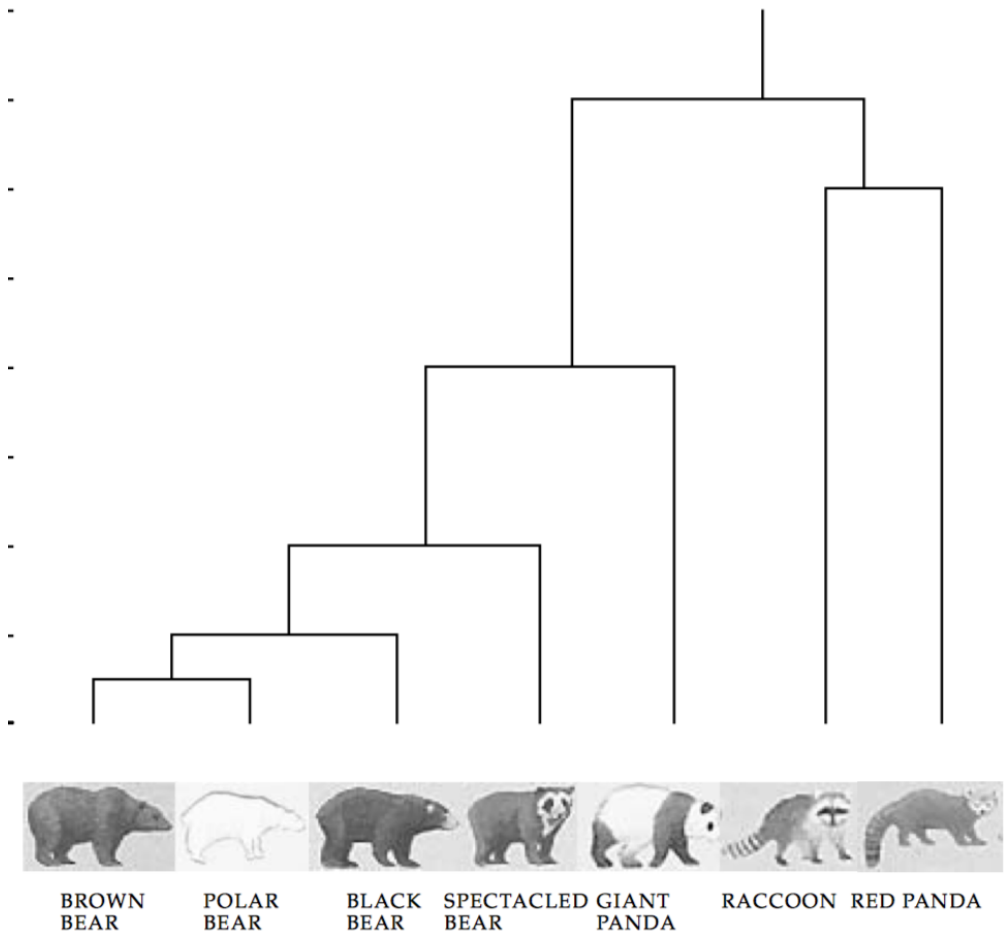
So I fill the similarity matrix by finding alignment score for each pair of rRNA sequences.

```
In [8]: for i in range(5):
            for j in range(5):
                if i != j:
                    sim_mat[i][j] = pairwise2.align.globalxs(seq_list[i].seq, seq_list[j].seq, -

        sim_mat

Out[8]: [[0, 912.0, 792.0, 800.0, 842.0],
         [912.0, 0, 791.0, 802.0, 837.0],
         [792.0, 791.0, 0, 821.0, 799.0],
         [800.0, 802.0, 821.0, 0, 804.0],
         [842.0, 837.0, 799.0, 804.0, 0]]
```

Last but not least I give the similarity matrix and the name list to the UPGMA algorithm I implemented in part A of question 3.



Phylogeny Tree

As we can see the final phylogeny tree computed by the algorithm is correct. Red panda and racoons are closer to each other. Brown and black bears are close too. And giant pandas are a closer specie to bears than they are to red pandas.

```
In [9]: upgma(sim_mat, names)

912 (brown-bear,black-bear)
839 ((brown-bear,black-bear),giant-panda)
821 (racoon,red-panda)
798 (((brown-bear,black-bear),giant-panda),(racoon,red-panda))
```

bio-hw3-q3c

November 26, 2017

1 Fundamentals of Bioinformatics

1.1 Homework Assignment 3

1.1.1 Sepehr Torabparhiz

1.1.2 93100774

1.2 Q3 part C

In [1]:
```
from Bio import SeqIO, pairwise2 # BioPython Library
```

In [2]:
```
sim_mat = [[0 for i in range(7)] for j in range(7)]
blosum_sim_mat = [[0 for i in range(7)] for j in range(7)]

sim_mat
```

Out[2]:
```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

In [3]:
```
mallard = SeqIO.read('mallard.txt', 'fasta')
human = SeqIO.read('human.txt', 'fasta')
chick = SeqIO.read('chick.txt', 'fasta')
mouse = SeqIO.read('mouse.txt', 'fasta')
rabbit = SeqIO.read('rabbit.txt', 'fasta')
dog = SeqIO.read('dog.txt', 'fasta')
gorilla = SeqIO.read('gorilla.txt', 'fasta')

names = ['chick', 'mallard', 'dog', 'human', 'gorilla', 'mouse', 'rabbit']
seq_list = [chick, mallard, dog, human, gorilla, mouse, rabbit]
```

In [4]:
```
# Checking the length of different sequences.
for specie_tmc1_seq in seq_list:
    print(len(specie_tmc1_seq))
```

```
847
669
758
760
761
757
757
```

1.3 Part 2: Alignment

Same as Q3 part b, I use global alignment. Again all sequences are quite similar in length and function(They all help these species to hear) as we are comparing the same protein in different animals, also we want to align the entire sequences together, in contrast to checking a small sequence against a large one. So global alignment is used again.

Also I found out that I can give *pairwise2* a *score_only* argument that works much faster.

```
In [5]: for i in range(7):
            for j in range(7):
                if i != j:
                    sim_mat[i][j] = pairwise2.align.globalxs(seq_list[i].seq, seq_list[j].seq, -

        sim_mat

Out[5]: [[0, 402.0, 392.0, 396.0, 394.0, 380.0, 387.0],
         [402.0, 0, 348.0, 350.0, 347.0, 351.0, 347.0],
         [392.0, 348.0, 0, 728.0, 732.0, 714.0, 724.0],
         [396.0, 350.0, 728.0, 0, 747.0, 718.0, 730.0],
         [394.0, 347.0, 732.0, 747.0, 0, 713.0, 727.0],
         [380.0, 351.0, 714.0, 718.0, 713.0, 0, 727.0],
         [387.0, 347.0, 724.0, 730.0, 727.0, 727.0, 0]]
```

1.4 Part 3: BLOSUM

BLOSUM is substitution matrix that for different pairs of amino acids, gives different mismatch scores. The reason why all possible pairs do not have the same score is that amino acids can be categorized into groups of similar attributes. So when comparing a pair of amino acids in two protein sequences, if the two amino acids belong to the same group, we expect less signifcant differences between two species compared to the time when one amino acid in one sequence has mutated to an amino acid whose properties differ greatly from the original amino acid.

Therefore, conservative changes in amino acids receive poistive scores, while one amino acid changing to another amino acis that is biochemically very different is given a negative score in BLOSUM matrices.

In building BLOSUMs, a set of sequences and their alignments are considered. As explained in a PMC article about Steve and Jorja Henikof's creation of BLOSUMs: "they identified conserved BLOCKS, or ungapped patches of conserved sequences, in sets of proteins that were potentially very distantly related. They then counted the amino-acid replacements within these blocks, using a percent identity threshold to exclude closely and more moderately related sequences." They also

showed that BLOSUM62 generally performs better and since then it became the default substitution matrix in many applications.

As it can be seen below, we are using BLOSUM62 which is available in BioPython. There are different BLOSUM matrices that have different numebrs in their name like BLOSUM80 or BLOSUM45. This number means that for example BLOSUM62 matrix was built using sequences that were less than 62% similar.

```
In [6]: from Bio.SubsMat.MatrixInfo import blosum62
```

1.5 Part 4: Choosing the Right Penalty Costs

Again BioPython's *pairwise2.align* is used. Two things are differnt from last time. First, I'm using *globalds* instead of *globalxs*. *d* means that a dictionary will be used to score matches and mismatches for different characters. This dictionary is the blosum62.

Also gap open and gap extension penalties are different. I have chosen -10 for open gap penalty and -0.5 for gap extension penalty. The reasoning behind these penalties is that better alignments are produced by highly penalizing for opening new gaps but reducing the score a little for extending an existing gap. This causes the aligner to match bigger gapless substrings of the sequences.

```
In [7]: for i in range(7):
            for j in range(7):
                if i != j:
                    blosum_sim_mat[i][j] = pairwise2.align.globalds(seq_list[i].seq, seq_list[j]

        blosum_sim_mat

Out[7]: [[0, 2983.0, 2650.5, 2655.0, 2645.0, 2604.0, 2623.5],
         [2983.0, 0, 2318.0, 2326.0, 2313.5, 2285.0, 2302.5],
         [2650.5, 2318.0, 0, 3857.0, 3888.0, 3817.5, 3833.0],
         [2655.0, 2326.0, 3857.0, 0, 3938.0, 3824.5, 3868.0],
         [2645.0, 2313.5, 3888.0, 3938.0, 0, 3805.5, 3860.0],
         [2604.0, 2285.0, 3817.5, 3824.5, 3805.5, 0, 3871.5],
         [2623.5, 2302.5, 3833.0, 3868.0, 3860.0, 3871.5, 0]]
```

1.6 Part 5: Phylogeny Tree

As we did in Q3 part B, I again use the UPGMA code to find the tree. As it can be seen the final result is:

((chick,mallard),((dog,(human,gorilla)),(mouse,rabbit))) This does intuitively make sense. Chickens and mallards(ducks) are close relatives. Same goes for mouse and rabbits, and humans and gorillas. It is also logical to see dogs are closer relatives to humans than chickens or rabbits.

```
In [8]: from q3a_upgma import upgma

In [9]: upgma(sim_mat, names)
```

```
747 (human,gorilla)
730 (dog,(human,gorilla))
727 (mouse,rabbit)
720 ((dog,(human,gorilla)),(mouse,rabbit))
402 (chick,mallard)
368 ((chick,mallard),((dog,(human,gorilla)),(mouse,rabbit)))
```