12/12/2021

Bioinformatics Programming Project: Coding the UPGMA Algorithm for Building Phylogenetic Trees

Collin Barraugh[1,2]

I. Algorithm Description & Understanding

I chose to code the Unweighted Pair Group Method using Arithmetic Averages (UPGMA) algorithm for the bioinformatics programming project. This algorithm is used to construct phylogenetic trees that establish an evolutionary hierarchy for species or genes [1]. There are three main types of methods that can build these trees: distances-based, parsimony-based, and maximum-likelihood-based methods. UPGMA falls into the distance-based category of these tree-building algorithms.

The UPGMA algorithm begins by taking in as its input a matrix of pairwise distances between all species or genes being considered. The algorithm then performs iterations in which it:

1. Finds the shortest distance between two pairs.
2. Merges the corresponding pair of species or genes into a single group.
3. Calculates the depth of the branch joining the pair by halving their pairwise distance.
4. Recomputes matrix of pairwise distances treating the joined pair as a new entity.

UPGMA will repeat these iterations until all species or genes are joined together into a single group.

This algorithm is especially useful from a bioinformatics perspective as it can be used to identify homologs, orthologs, and paralogs. Homologs are genetic sequences in different species that originated from a common ancestor species [2]. Orthologs are homologs that result from a speciation event but have their main function conserved. Paralogs are just duplicate genes in a species. Identifying these kinds of genetic similarity requires evolutionary information to build genetic lineages. UPGMA is powerful in that it can provide this evolutionary information just through utilizing pairwise sequence alignment as input distances.

II. Implementation

A. Development Specifications

I implemented my algorithm using the Python 3.7 programming language. My development environment was built on the MacOS X Big Sur v11.6.1 operating system. My entire repository of code is hosted on GitHub at https://github.com/collinbarraugh/Bioinformatics_Class.

B. Dependencies

All necessary dependencies are listed in the environment.yml file. They will be installed upon the creation of the bioinformatics_class virtual environment. The dependencies include pandas and numpy for data manipulation, matplotlib and scipy for data visualization, jupyter for step-by-step execution of the algorithm, pytest for running the test cases, and ipykernel and pip for package installation and virtual environment management.

1. Colorado School of Mines
2. National Renewable Energy Laboratory

C. Procedure/Commands to Execute Code

All instructions for cloning my GitHub repository and running executables and test cases are in the repository's README. They are as follows:

Installation

To install the dependencies in the .yml file:

`conda env create`

`conda activate bioinformatics_class`

`python setup.py develop`

Install algorithm executables with pip:

`pip install --editable=git+https://github.com/collinbarraugh/Bioinformatics_Class.git`

If you edit the .yml file with new packages to install, just use:

`conda env update`

Testing

Verify your installation is correct and execute test cases by running:

`pytest -s -v tests/test.py`

D. Approach

To implement the algorithm, my function upgma_full takes in two inputs: a matrix of pairwise distances and a list of single character labels that correspond to the species/genes in each row/column. The matrix of pairwise distances is symmetric across a diagonal of -1 scalars. That function then creates a loop that runs another function, upgma_iteration, until the matrix of pairwise distances is a 1x1 matrix. After it runs each iteration of the algorithm, it saves a list the updated stepwise groupings of entities, the depth associated with the branch of the newest grouping, the dimension of the new pairwise distance matrix, and the index of the original pairwise distance matrix that was cut due to the new grouping.

The upgma_iteration function that is called by upgma-full is where most of the magic happens. upgma_iteration takes in the updated pairwise distance matrix, the original pairwise distance matrix, a list of all indices that have been cut from prior pairwise distance matrices, as well as a list of the stepwise groupings of character codes. The function first checks go make sure that the updated pairwise distance matrix is square and has a dimension greater than 1x1. It then parses the lower triangular matrix of the pairwise distances and finds the smallest distance. It stores the indices of the smallest distances to mark the individual or groups of species or genes that are in the new pair.

It then appends the new grouping to the list of stepwise groupings. The function then calculates the mean pairwise distances between the new grouping and the pre-existing individuals/ groupings in the matrix. Computing this mean was the crux of the algorithm. It was done by taking column slices of the matrix corresponding to the indices of the new pair, setting the smallest distance to -1 and then taking the average across values in both slices. Note that it is imperative for the input matrix to be symmetric across a diagonal of -1s for this to work.

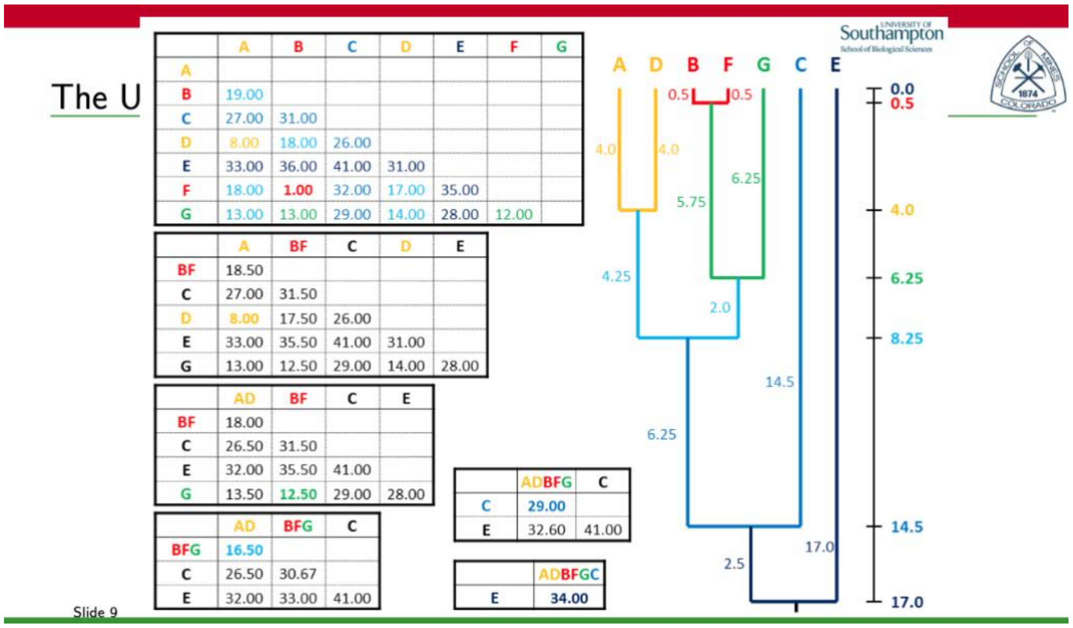
To construct the new matrix of pairwise distances, I then:

1. Replace the slice at the smaller index in the pair with the mean pairwise distance slice.
2. And delete the row and column corresponding to the higher index in the pair.

Upon construction of the new matrix, the iteration is completed, and I save the new matrix, new depth, original matrix, index of the column/row was deleted, and then the list of stepwise groupings of the entities. These iterations execute until the last grouping is made and the algorithm is complete.

III. Verification

For my test cases, I used the UPGMA walk-through example from our in-class lecture [1] to verify the output of my algorithm.



The U

	A	B	C	D	E	F	G
A							
B	19.00						
C	27.00	31.00					
D	8.00	18.00	26.00				
E	33.00	36.00	41.00	31.00			
F	18.00	1.00	32.00	17.00	35.00		
G	13.00	13.00	29.00	14.00	28.00	12.00	

	A	BF	C	D	E
BF	18.50				
C	27.00	31.50			
D	8.00	17.50	26.00		
E	33.00	35.50	41.00	31.00	
G	13.00	12.50	29.00	14.00	28.00

	AD	BF	C	E
BF	18.00			
C	26.50	31.50		
E	32.00	35.50	41.00	
G	13.50	12.50	29.00	28.00

	AD	BFG	C
BFG	16.50		
C	26.50	30.67	
E	32.00	33.00	41.00

	ADBFG	C
C	29.00	
E	32.60	41.00

	ADBFGC
E	34.00

Figure 1. The results of the class walk-through example of UPGMA that my model correctly replicated.

1. Colorado School of Mines
2. National Renewable Energy Laboratory

My test functions verify that the groupings are correct after each iteration of the algorithm. It also checks to make sure that all the branch depths for new pairings are correct as well. In the Jupyter Notebook version, you can also print out the new_matrix that results from each iteration to verify that the pairwise distance matrix is being updated correctly.

IV. Results/Analysis

I found that my algorithm as I have coded from scratch reproduces the results of the class walk-through exactly. This was supported my implementation of the algorithm. For future steps, I could also add a visual verification where I convert my outputs into a dendrogram. It would then be possible to validate the tree diagram against that created by a verified bioinformatics tool such as BioPython which has the Phylo.TreeConstruction package. As my results stand, I would expect that validation would confirm the accuracy of my algorithm as well.

IV. References

[1] Wang, H. *Lecture 08: Phylogenetic Analyses*. (2021, September 23). Department of Computer Science, Colorado School of Mines

[2] *Homologs, Orthologs, and Paralogs*. (2021, January 3). https://bio.libretexts.org/@go/page/9312

1. Colorado School of Mines
2. National Renewable Energy Laboratory