Practical 2: Perfect Phylogeny

Sara Montese

25/09/2023

1. Python script to extract the segregating sites from the sequences into a binary matrix.

```
import numpy as np
from Bio import AlignIO
import pandas as pd

input_file = "./sequences.fa"
alignment = AlignIO.read(input_file, "fasta") #read fasta file
df = pd.DataFrame(alignment)

#calculate the number of unique values in each column of the df
#and drop the columns which only have a single unique value:
nunique = df.nunique()
nonunique_cols_to_drop = nunique[nunique == 1].index
df.drop(nonunique_cols_to_drop, axis=1, inplace=True)

#Remove columns containing dots by checking selecting only
#columns which have allowed values
allowed_vals=['A', 'C', 'T', 'G']
genomic_sequences = df[ df.isin(allowed_vals) ].dropna(axis=1)


#merge content of columns in dataframe
new_df =genomic_sequences↩
   .apply(lambda row: ''.join(map(str, row)), axis=1)

#convert to binary representation
genomes = []    # a list of full one line genomes
length = 0      # a length of single genome
base = []↩
        #search for the first genome -oldest one without mutations
for line in new_df:
    genomes.append(line)

    if not length:
        length = len(genomes[0])
        for i in range(length):
            base.append({"A":0, "T":0, "G":0, "C":0})
    for i, letter in enumerate(line):
    #count how many times a letter appears in each column
        if letter == "A":
            base[i]["A"]+=1
        if letter == "T":
            base[i]["T"]+=1
        if letter == "G":
            base[i]["G"]+=1
        if letter == "C":
            base[i]["C"]+=1
genome_amount = len(genomes)
```

```
#how many times each gene had the most frequent oligonucleotide
gen_count= [0]*genome_amount
impact_letter_num_list = []
for i, dict in enumerate(base):
    #letter most often in the sequences at this column
    frequent = max(dict, key=dict.get)
     #for each genome give it one
    for j, genome in enumerate(genomes):
            if(genome[i]==frequent):
                #how many times↩
                    each gene had the most frequent oligonucleotide
                 gen_count[j]+=1
    impact_letter_num_list.append(i)
max_count = 0
max_count_iterator = None
for i, g in enumerate(gen_count):
    if g>max_count:
        max_count = g
        max_count_iterator = i
#the genome with the highest count, the not mutated one
long_base_seq=genomes[max_count_iterator]
base_seq =""
for i, letter in enumerate(long_base_seq):
    if(i in impact_letter_num_list):
        base_seq+=letter

meaning_genoms = []
for j, genome in enumerate(genomes):
    new_genome=''
    for i, (↩
       num, b) in enumerate(zip(impact_letter_num_list, base_seq)):
        if(genome[num]==base_seq[i]):
            new_genome+='0'
        else:
            new_genome+='1'
    meaning_genoms.append(str(new_genome))
base_seq = meaning_genoms[max_count_iterator]
print("There are", len(meaning_genoms), "sequences.")
print("The sequences have", len(base_seq), "segregating sites." )
print("Binary matrix of segregating sites:")
for j, genome in enumerate(meaning_genoms):
    print(genome)
    if(j == max_count_iterator):
        if('1' in genome):
            print("ERROR")
        else:
            print("The base sequence consists of only 0 - CORRECT")
```

2. How many genomic sequences are there? There are 11 genomic sequences.
3. How many segregating sites do they have? There are 44 segregating sites.
4. Python script to determine whether there is a perfect phylogeny for the segregating sites of the sequences

```
import numpy as np

# Convert binary strings to a matrix and then np.array
```

```
matrix↩
    = [[int(bit) for bit in binary] for binary in meaning_genoms]
np_matrix = np.array(matrix)

# Transpose the matrix to work with rows
transposed_matrix = np_matrix.T

# Calculate the sum of each row
row_sums = [(i, sum(transposed_matrix↩
   [i])) for i in range(transposed_matrix.shape[0])]

# Sort the rows in decreasing order of sums
sorted_rows = sorted(row_sums, key=lambda x: x[1], reverse=True)

# Create a new matrix using the sorted rows AND
# Transpose the sorted matrix back to its original form
sorted_matrix↩
    = np.array([transposed_matrix[i] for i, _ in sorted_rows]).T
num_columns = len(sorted_matrix[0])

relationships = {}
is_perfect = True
# Compare all pairs of columns
for i in range(num_columns):
    for j in range(i + 1, num_columns):
        Oi = [row[i] for row in matrix]
        Oj = [row[j] for row in matrix]

        intersection_empty↩
            = all(x & y == 0 for x, y in zip(Oi, Oj))
        oi_subset_oj = all(x <= y for x, y in zip(Oi, Oj))
        oj_subset_oi = all(x >= y for x, y in zip(Oi, Oj))

        if intersection_empty or oi_subset_oj or oj_subset_oi:
            pass
        else:
            is_perfect = False

if is_perfect == True:
    print("it's a perfect phylogeny")
else:
    print("it's not a perfect phylogeny")
```

5. What is the running time of your script, as a function of the number n of genomic sequences and the number m of segregating sites? To solve the perfect phylogeny problem for a binary matrix M, we implemented a algorithm with a running time of $O(nm^2)$, where n is the number of genomic sequences (11) and m is the number of segregating sites (44).
6. What is the best possible running time of an algorithm to solve the perfect phylogeny problem? The best algorithm to solve the perfect phylogeny problem for a binary matrix M of n genomic sequences and m segregating sites has a running time of $O(nm)$.