# 软件著作权-源码

#!/usr/bin/python3

import numpy as np

from pandas.core.frame import DataFrame

from multiprocessing import Pool

from Bio import SeqIO

import os, time, random

import argparse

import sys

'''

解析参数

'''

parser = argparse.ArgumentParser(description="Options for cp\_alignment.py",

add\_help=True)

additional = parser.add\_argument\_group("additional arguments")

additional.add\_argument('-p', '--proportion', action="store", type=float,

default=0.5, metavar='\b', help='''The proportion of

missing data allowed in each site, default = 0.2''')

additional.add\_argument('-n', '--num\_cpu', action="store", type=int, default=12,

metavar='\b', help='''The maximum number of CPUs that

this script can be used (Two usable CPUs means that the

script will analyze two matrices at the same time),

default = 12''')

args = parser.parse\_args()

proportion = args.proportion

th = args.num\_cpu

if proportion == 0: #该参数会作为被除数使用，故不能为零

proportion = 0.0001

def get\_file\_list(ext):

'''

函数get\_file\_list:

获取当前文件夹中符合目标扩展名的文件

输入:

当前工作文件夹中的所有文件名名称(脚本自动获取)

输出:

file\_name：扩展名符合的文件的名称列表

'''

file\_name = []

ext\_len = len(ext)

for each in os.listdir(os.getcwd()):

if ext == each[-ext\_len:]:

file\_name.append(each)

return file\_name

def get\_th\_list(file\_name):

'''

函数 get\_th\_list：

将所有文件平均的分给各个进程

输入：

file\_name: 需要进行分析的所有文件的名称列表

输出：

th\_list：列表，列表中的各个元素为各个进程所应该处理的文件

'''

th\_list = []

for each\_num in range(th):

th\_list.append([])

print("本程序共使用 " + str(th) + " 个进程")

n = 0

for each\_file in file\_name:

th\_list[n].append(each\_file)

if n == th - 1:

n = 0

else:

n = n + 1

return th\_list

def if80to1(fasta\_name):

'''

函数if80to1:

如果fasta序列是80列换行的话则修改成不换行

输入：

待处理的fasta文件名称

输出：

如果原始fasta中的序列是80行换行的则将所有序列集中至一行。并写出一个新的

后缀为“.fa”格式的文件

'''

tmp\_len\_list = []

for each\_record in SeqIO.parse(fasta\_name, "fasta"):

tmp\_len\_list.append(len(str(each\_record.seq)))

tmp\_len\_list.sort()

with open(fasta\_name[:-3], "a") as write\_file:

for each\_record in SeqIO.parse(fasta\_name, "fasta"):

if len(str(each\_record.seq)) != tmp\_len\_list[-1]:

gap = "-"\*(tmp\_len\_list[-1] - len(str(each\_record.seq)))

each\_record.seq = each\_record.seq + gap

write\_file.write(">" + str(each\_record.id) + "\n")

write\_file.write(str(each\_record.seq) + "\n")

else:

write\_file.write(">" + str(each\_record.id) + "\n")

write\_file.write(str(each\_record.seq) + "\n")

if tmp\_len\_list[0] >= 2000:

return True

else:

return False

def split\_fasta(fasta\_name):

'''

函数split\_fasta:

将排序文件每隔2000bp分割一次

输入：

待处理的fasta文件名称

输出：

后缀为“.fa”，并且文件名中带有“.split.”字样的文件

'''

with open(fasta\_name[:-3], "r") as read\_file:

sequences = read\_file.readlines()

length = len(sequences[1]) - 1

left = 0

right = 2000

while True:

new\_name = (fasta\_name[:-6] + ".split." + str(left) + ".fa")

if right <= length:

with open(new\_name, "a") as write\_file:

for each\_line in sequences:

if each\_line[0] == ">":

write\_file.write(each\_line)

else:

write\_file.write(each\_line[left:right] + "\n")

left = left + 2000

right = right + 2000

else:

with open(new\_name, "a") as write\_file:

for each\_line in sequences:

if each\_line[0] == ">":

write\_file.write(each\_line)

else:

write\_file.write(each\_line[left:])

break

def get\_seq\_name\_list(fasta\_name):

'''

函数get\_seq\_name\_list:

获得alignment文件中各个序列的序列名称

输入：

待处理的fasta文件名称

输出：

seq\_name\_list包含有所有序列名称的一个列表文件

'''

seq\_name\_list = []

with open(fasta\_name) as read\_file:

for each\_line in read\_file:

if each\_line[0] == ">":

seq\_name\_list.append(each\_line)

return seq\_name\_list

def calculate(fasta\_name, proportion):

'''

函数calculate:

计算alignment文件各位点上gap所占的比例，如果大于设定的阈值，则删除该位点

输入：

待处理的fasta文件名称

输出：

result\_list: 包含所有被删除过gap的序列的列表

'''

seq\_array\_temp\_list = []

with open(fasta\_name) as read\_file:

for each\_line in read\_file:

if each\_line[0] != ">":

if each\_line[-1] == "\n":

seq\_array\_temp\_list.append(list(each\_line[:-1]))

else:

seq\_array\_temp\_list.append(list(each\_line))

seq\_array = DataFrame(seq\_array\_temp\_list)

row = seq\_array.shape[0]

column = seq\_array.shape[1]

temp\_list = []

for each\_num in range(0,column):

gap1 = list(seq\_array[each\_num]).count("-")

gap2 = list(seq\_array[each\_num]).count("?")

gap\_num = gap1 + gap2

if gap\_num/row >= proportion:

pass

else:

temp\_list.append(list(seq\_array[each\_num]))

temp\_seq\_array = DataFrame(temp\_list)

result\_list = []

for each\_num in range(0,row):

try:

result\_list.append("".join(list(temp\_seq\_array[each\_num].values)))

except:

result\_list = []

return(result\_list)

def white2file(fasta\_name, seq\_name\_list, seq\_list):

'''

函数white2file: 将计算结果写入输出文件中

输入：

fasta\_name: 待处理的fasta文件名称；

seq\_name\_list：包含有所有序列名称的列表文件；

seq\_list：包含所有被删除过gap的序列的列表

输出：

写出一个后缀为.fas的结果文件

'''

with open(fasta\_name + "s", "a") as write\_file:

seq\_num = len(seq\_name\_list)

for num in range(0,seq\_num):

judge1 = seq\_list[num].count("-") == len(seq\_list[num])

judge2 = seq\_list[num].count("?") == len(seq\_list[num])

if judge1 or judge2:

pass

else:

write\_file.write(seq\_name\_list[num])

write\_file.write(seq\_list[num] + "\n")

def main\_get\_homo(file\_name):

'''

函数main\_get\_homo:

控制前面函数运行的函数,删掉alignment中missing site

输入：

“.fa”格式文件，该文件由preprocessing函数生成

输出：

“.fas”结果文件

'''

for fasta\_name in file\_name:

seq\_name\_list = get\_seq\_name\_list(fasta\_name)

seq\_list = calculate(fasta\_name, proportion)

if len(seq\_list) != 0: #删完gap之后删掉只剩下"-"的物种

white2file(fasta\_name, seq\_name\_list, seq\_list)

os.remove(fasta\_name)

else:

os.remove(fasta\_name)

def preprocessing(file\_name):

'''

函数preprocessing:

1：如果是80行换行的fasta，则转化为不换行的

2：如果矩阵的长度超过2000bp，则将矩阵分割成最长为2000bp的小矩阵

输入：

自动输入文件夹中“.fasta”格式文件

输出：

“.fas”格式的结果文件

'''

for fasta\_name in file\_name:

if if80to1(fasta\_name):

split\_fasta(fasta\_name)

os.remove(fasta\_name[:-3])

def concat(gene\_name\_list):

'''

函数concat:

脚本最初会将过长的矩阵分割成较小的部分进行并行处理，这里需要将这些被分割的

文件重新合并成一个文件

输入：

基因名称的列表，不含有后缀

输出：

“.fas”格式的结果文件

'''

for each\_gene in gene\_name\_list:

fasta\_file\_list = []

for each\_file\_name in os.listdir(os.getcwd()):

if each\_gene + ".split." in each\_file\_name:

fasta\_file\_list.append(each\_file\_name)

concat\_list = []

for each\_file in fasta\_file\_list:

with open(each\_file, "r") as read\_file:

for each\_line in read\_file:

if each\_line[0] == ">":

if each\_line not in concat\_list:

concat\_list.append(each\_line)

n = 0

for each\_file in fasta\_file\_list:

fasta\_dict = SeqIO.to\_dict(SeqIO.parse(each\_file, "fasta"))

for each\_len in fasta\_dict:

seq\_len = len(fasta\_dict[each\_len].seq)

break

for index, each\_species in enumerate(concat\_list):

if each\_species.split("\n")[0][1:] in fasta\_dict:

str1 = str(fasta\_dict[each\_species.split("\n")[0][1:]].seq)

concat\_list[index] = concat\_list[index] + str1

else:

str2 = "?"\*seq\_len

concat\_list[index] = concat\_list[index] + str2

os.remove(each\_file)

with open(each\_gene + ".fas", "a") as write\_file:

for each\_line in concat\_list:

write\_file.write(each\_line + "\n")

'''

预处理文件：

1：如果是80行换行的fasta，则转化为不换行的

2：如果矩阵的长度超过2000bp，则将矩阵分割成最长为2000bp的小矩阵

'''

file\_name = get\_file\_list(".fasta")

th\_list = get\_th\_list(file\_name)

p = Pool(th)

for i in range(th):

p.apply\_async(preprocessing,(th\_list[i],))

p.close() # 关闭进程池，关闭后po不再接收新的请求

p.join() # 等待po中所有子进程执行完成，再执行下面的代码

'''

并行删除alignment格式文件中的missing site

'''

file\_name = get\_file\_list(".fa")

th\_list = get\_th\_list(file\_name)

p = Pool(th)

for i in range(th):

p.apply\_async(main\_get\_homo,(th\_list[i],))

p.close() # 关闭进程池，关闭后po不再接收新的请求

p.join() # 等待po中所有子进程执行完成，再执行下面的代码

'''

合并最开始被分成小片段的矩阵

'''

gene\_name\_list = []

for each\_file\_name in os.listdir(os.getcwd()):

if ".split." in each\_file\_name:

gene\_name = each\_file\_name.split(".")[0]

if gene\_name not in gene\_name\_list:

gene\_name\_list.append(gene\_name)

th\_list = get\_th\_list(gene\_name\_list)

p = Pool(th)

for i in range(th):

p.apply\_async(concat,(th\_list[i],))

p.close() # 关闭进程池，关闭后po不再接收新的请求

p.join() # 等待po中所有子进程执行完成，再执行下面的代码