MAIN CLASS (FI-DE)

import math

import numpy as np

import random

import pandas as pd

from statistics import mean

def DE(biclusters, biclu, best):

dimen = len(biclusters)

F = 0.2

CR = 0.5

abc = random.sample(biclusters,2)

q = random.randint(1,dimen)

for k in range(dimen):

RK = random.uniform(0,1)

if RK<CR or k==q:

mutate = np.array(best) + F\*(np.array(abc[0])-np.array(abc[1]))

mutate\_round = roundoff(mutate).tolist()

new\_bic = [round(x) for x in mutate\_round]

else:

new\_bic = biclu

return new\_bic

def hamming(biclu\_1, biclu\_2):

diff = 0

for ch1, ch2 in zip(biclu\_1, biclu\_2):

if ch1 != ch2:

diff += 1

return diff

def volume(biclusters,n):

volumes = []

for line in biclusters:

vol = line[:n].count(1)\*line[n:].count(1)

volumes.append(vol)

return volumes

def roundoff(bics):

for j, item in enumerate(bics):

if item < 0.95:

bics[j] = 0

else:

bics[j] = 1

return bics

def MSR(bic):

bicluster\_residue = []

aiJ\_list = []

aIj\_list = []

#calculate the mean of each row in the bicluster

for row in bic:

row\_mean = mean(row)

aiJ\_list.append(row\_mean)

#calculate the mean of each column in the bicluster

bicluster\_transpose = np.array(bic).T.tolist()

for column in bicluster\_transpose:

column\_mean = mean(column)

aIj\_list.append(column\_mean)

aIJ = mean(aiJ\_list) #mean of the bicluster

#calculate the residue of each element in the bicluster

for i, row in enumerate(bic):

row\_residue = []

for j, item in enumerate(row):

res = item + aIJ - aiJ\_list[i] - aIj\_list[j]

row\_residue.append(res)

bicluster\_residue.append(row\_residue)

#calculate the MSR of the bicluster

residue\_sum = 0.0

I = len(bicluster\_residue)

J = len(bicluster\_residue[0])

for row in bicluster\_residue:

row\_sum = 0.0

for item in row:

row\_sum = row\_sum + item\*\*2

residue\_sum = residue\_sum + row\_sum

msr = residue\_sum/(I\*J)

#calculate the variance of the bicluster

variance\_sum = 0.0

for i, row in enumerate(bic):

sum\_vari = 0.0

for item in row:

sum\_vari += (item - aiJ\_list[i])\*\*2

row\_variance = sum\_vari/J

variance\_sum += row\_variance

bi\_variance = variance\_sum/I

#calculate the bicluster fitness

if bi\_variance == 0:

bi\_fitness = msr

else:

bi\_fitness = msr + 1/bi\_variance

return msr, bi\_variance, bi\_fitness

def extract\_bicluster(input\_data, new\_bic):

gindex = [i for i, item in enumerate(new\_bic[:15380]) if item==1]

cindex = [i for i, item in enumerate(new\_bic[15380:]) if item==1]

bic = []

for item in gindex:

row = []

for line in cindex:

row.append(input\_data[item][line])

bic.append(row)

return bic

#exclude the same biclusters using MSR

def exclude\_same\_biclusters(MSRs, var, fit, biclusters):

for i, item1 in enumerate(MSRs):

for j, item2 in enumerate(MSRs):

if item1 == item2:

if biclusters[i] == biclusters[j]:

del(biclusters[j])

del(MSRs[j])

del(var[j])

del(fit[j])

return MSRs, biclusters

def sort\_biclusters(MSRs, var, fit, biclusters):

zipped\_pairs = zip(MSRs, var, fit, biclusters)

sorted\_pairs = sorted(zipped\_pairs)

tuple\_pairs = zip(\*sorted\_pairs)

sorted\_msr, sorted\_var, sorted\_fit, sorted\_bic = [list(tuple) for tuple in tuple\_pairs]

return sorted\_msr, sorted\_var, sorted\_fit, sorted\_bic

def annd(f,bic1):

dim = len(bic1)

and\_result = []

for i in range(dim):

if bic1[i] == 1 and f == 1:

and\_result.append(1)

else:

and\_result.append(0)

return and\_result

def xor(bic1,bic2):

dim = len(bic1)

xor\_result = []

for i in range(dim):

if bic1[i]==bic2[i]:

xor\_result.append(0)

else:

xor\_result.append(1)

return xor\_result

def union(bic1,bic2):

dim = len(bic1)

union\_result = []

for j in range(dim):

if bic1[j] == 0 and bic2[j] == 0:

union\_result.append(0)

else:

union\_result.append(1)

return union\_result

def index(biclus):

biclusters\_gene\_cond\_count = []

for item in biclus:

gene\_cond\_count = []

gene\_count = item[:15380].count(1)

cond\_count = item[15380:].count(1)

gene\_cond\_count.append(gene\_count)

gene\_cond\_count.append(cond\_count)

biclusters\_gene\_cond\_count.append(gene\_cond\_count)

return biclusters\_gene\_cond\_count

#reading encoded bicluster file and normalized input data

df = pd.read\_csv(r'C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/output/encoded\_schisto.csv', header=None)

df2 = pd.read\_csv(r'C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/output/norm\_input\_file\_schisto.csv', header=None)

df.dropna(axis=0, how='all',inplace=True)

df2.dropna(axis=0, how='all',inplace=True)

biclust = df.values.tolist()

input\_data = df2.values.tolist()

biclusters = []

for line in biclust:

biclusters.append([int(x) for x in line])

#calculate MSR, gene variance and fitness of all seeds

MSRs = []

var = []

fit = []

for item in biclusters:

seed\_msr = extract\_bicluster(input\_data, item)

MSRs.append(MSR(seed\_msr)[0])

var.append(MSR(seed\_msr)[1])

fit.append(MSR(seed\_msr)[2])

print(len(MSRs))

MSRs, biclusters = exclude\_same\_biclusters(MSRs, var, fit, biclusters)

print()

print(len(MSRs))

MSRs, var, fit, biclusters = sort\_biclusters(MSRs, var, fit, biclusters)

biclusters\_dimen = len(biclusters)

print(index(biclusters))

print(MSRs)

beta = 1.0

gamma = 5.0

alpha = 0.5

F = 1

#abc = random.sample(biclusters,2)

best\_index = MSRs.index(min(MSRs))

best = biclusters[best\_index]

for l in range(10):

print('Epoch: ',l)

for i in range(biclusters\_dimen):

ab = random.sample(biclusters,2)

'''if i == biclusters\_dimen-1:

xor\_res = xor(biclusters[0],biclusters[i])

else:'''

xor\_res = xor(ab[0],ab[1])

temp\_bic = union(best, xor\_res)

new\_bic = union(biclusters[i],temp\_bic)

new\_bic\_data = extract\_bicluster(input\_data, new\_bic)

new\_bic\_MSR = MSR(new\_bic\_data)[0]

new\_bic\_var = MSR(new\_bic\_data)[1]

new\_bic\_fit = MSR(new\_bic\_data)[2]

biclusters[i] = new\_bic

MSRs[i] = new\_bic\_MSR

var[i] = new\_bic\_var

fit[i] = new\_bic\_fit

MSRs, var, fit, biclusters = sort\_biclusters(MSRs, var, fit, biclusters)

''' for j in range(biclusters\_dimen):

if MSRs[i] < MSRs[j]:

#vol = biclusters[i].count(1)

dist = hamming(biclusters[i],biclusters[j])

attract = beta\*math.exp(-gamma\*dist\*\*2)

move = np.array(biclusters[i]) + attract\*(np.array(biclusters[j]) - np.array(biclusters[i])) + alpha\*(random.random() - 0.5)

xor\_res = xor(biclusters[j],biclusters[i])

#and\_res = annd(F, xor\_res)

new\_bic = union(best, xor\_res)

#new\_bic = union(biclusters[i], mutate)

new\_bic\_data = extract\_bicluster(input\_data, new\_bic)

geneindex = [i for i, item in enumerate(new\_bic[:721]) if item==1]

condindex = [i for i, item in enumerate(new\_bic[721:]) if item==1]

if 1 not in geneindex or 1 not in condindex:

continue

new\_bic\_MSR = MSR(new\_bic\_data)

#new\_vol = new\_bic.count(1)

#if new\_bic\_MSR < 300:

biclusters[i] = new\_bic

MSRs[i] = new\_bic\_MSR

else:

new\_bic = DE(biclusters,biclusters[i],best)

geneindex = [i for i, item in enumerate(new\_bic[:721]) if item==1]

condindex = [i for i, item in enumerate(new\_bic[721:]) if item==1]

if 1 not in geneindex or 1 not in condindex:

continue

new\_bic\_data = extract\_bicluster(input\_data, new\_bic)

new\_bic\_MSR = MSR(new\_bic\_data)

if new\_bic\_MSR > 300:

continue

else:

biclusters[i] = new\_bic

MSRs[i] =new\_bic\_MSR'''

print(index(biclusters))

print(MSRs)

print(var[:15])

#print(fit)

biclusters\_index = []

for line in biclusters:

biclu\_index = []

biclu\_index.append([x for x,item in enumerate(line[:15380]) if item==1])

biclu\_index.append([x for x,item in enumerate(line[15380:]) if item==1])

biclusters\_index.append(biclu\_index)

print(biclusters\_index[:15])

Extract\_gene names

import csv

path = r'C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/data/schisto\_genenames.csv'

with open(path) as f:

gene\_names = csv.reader(f, delimiter = '\t')

gnames = [line for sublist in gene\_names for line in sublist]

path1 = r'C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/output/bicluster\_gene\_index\_schisto.txt'

input\_file = open(path1)

bicluster\_index = []

for line in input\_file:

gene\_line = line.split(',')

bicluster\_index.append([int(i) for i in gene\_line])

bicluster\_genes = []

for line in bicluster\_index:

each\_bicluster\_genes = []

for item in line:

each\_bicluster\_genes.append(gnames[item])

bicluster\_genes.append(each\_bicluster\_genes)

for line in bicluster\_genes:

print(line)

print()

Clustering. Py

import csv

import math

import numpy as np

from nltk.cluster import KMeansClusterer, cosine\_distance

from statistics import mean

#function to recluster bicluster seed that has # of genes>15

def recluster\_gene(pos,gene\_data):

biclut\_array = [gene\_data[x] for x in pos]

gene\_clusterer = KMeansClusterer(2, cosine\_distance)

gene\_clusters = gene\_clusterer.cluster(biclut\_array, True, trace = True)

group1 = []

group2 =[]

for i, item in enumerate(gene\_clusters):

if item ==0:

group1.append(pos[i])

else:

group2.append(pos[i])

return group1,group2

#function to recluster bicluster seed that has # of conditions>5

def recluster\_condition(pos,cond\_data):

biclut\_array = [cond\_data[x] for x in pos]

condition\_clusterer = KMeansClusterer(2, cosine\_distance)

condition\_clusters = condition\_clusterer.cluster(biclut\_array, True, trace = True)

group1 = []

group2 =[]

for i, item in enumerate(condition\_clusters):

if item ==0:

group1.append(pos[i])

else:

group2.append(pos[i])

return group1,group2

#open input file

in\_file = []

path = r'C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/data/schisto.csv'

with open(path) as f:

input\_file = csv.reader(f, delimiter = '\t')

for line in input\_file:

gene\_line = line[0].split(",")

in\_file.append([float(i) for i in gene\_line])

#transformation of raw expression values with log function

normalized\_file = []

for line in in\_file:

ll=[]

for item in line:

if item <=0:

norm = 0

ll.append(norm)

else:

norm = 100\*(math.log(10\*\*5\*item))

ll.append(norm)

normalized\_file.append(ll)

'''input\_file = open(path)

for line in input\_file:

gene\_line = line.split()

in\_file.append([float(i) for i in gene\_line])'''

#write normalized raw data

with open("C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/output/norm\_input\_file.csv","w") as f:

wr = csv.writer(f)

wr.writerows(in\_file)

#convert list data to array data

gene\_normalized\_array = np.array([np.array(i) for i in in\_file])

condition\_normalized\_array = np.array([np.array(i) for i in in\_file]).T

#cluster the data using gene, k=54(yeast), k=124(schisto)

gene\_clusterer = KMeansClusterer(124, cosine\_distance, repeats = 10)

gene\_clusters = gene\_clusterer.cluster(gene\_normalized\_array, True, trace = True)

#extract the genes positions for each cluster

gene\_positions = []

for k in range(124):

gene\_pos = []

for i, item in enumerate(gene\_clusters):

if item != k:

continue

else:

gene\_pos.append(i)

gene\_positions.append(gene\_pos)

#print(gene\_positions)

print(len(gene\_positions))

#recluster seeds with higher number of genes

countg = 0

while countg<len(gene\_positions):

if len(gene\_positions[countg])>80:

grp1,grp2 = recluster\_condition(gene\_positions[countg],gene\_normalized\_array)

del(gene\_positions[countg])

gene\_positions.append(grp1)

gene\_positions.append(grp2)

else:

countg +=1

#print(gene\_positions)

print(len(gene\_positions))

#cluster the data using condition, k=4(yeast), k=5(shisto)

condition\_clusterer = KMeansClusterer(5, cosine\_distance, repeats = 5)

condition\_clusters = condition\_clusterer.cluster(condition\_normalized\_array, True, trace = True)

#extract the condition positions for each cluster

condition\_positions = []

for k in range(5):

condition\_pos = []

for i, item in enumerate(condition\_clusters):

if item != k:

continue

else:

condition\_pos.append(i)

condition\_positions.append(condition\_pos)

print(condition\_positions)

print(len(condition\_positions))

#recluster seeds with higher number of conditions

countc = 0

while countc<len(condition\_positions):

if len(condition\_positions[countc])>5:

grp1,grp2 = recluster\_condition(condition\_positions[countc],condition\_normalized\_array)

del(condition\_positions[countc])

condition\_positions.append(grp1)

condition\_positions.append(grp2)

else:

countc +=1

print(condition\_positions)

print(len(condition\_positions))

#extract biclusters

biclusters = []

biclusters\_indices = []

for line in gene\_positions:

gene\_index = line

for row in condition\_positions:

each\_bicluster = []

each\_bicluster\_index = []

for item in gene\_index:

each\_row = []

for column\_index in row:

each\_row.append(gene\_normalized\_array[item,column\_index])

each\_bicluster.append(each\_row)

each\_bicluster\_index.append(gene\_index)

each\_bicluster\_index.append(row)

biclusters.append(each\_bicluster)

biclusters\_indices.append(each\_bicluster\_index)

print(len(biclusters))

#MSR function

biclusters\_MSR = []

for k, line in enumerate(biclusters):

bicluster\_residue = []

aiJ\_list = []

aIj\_list = []

#calculate the mean of each row in the bicluster

for row in line:

row\_mean = mean(row)

aiJ\_list.append(row\_mean)

#calculate the mean of each column in the bicluster

bicluster\_transpose = np.array(line).T.tolist()

for column in bicluster\_transpose:

column\_mean = mean(column)

aIj\_list.append(column\_mean)

aIJ = mean(aiJ\_list) #mean of the bicluster

#calculate the residue of each element in the bicluster

for i, row in enumerate(line):

row\_residue = []

for j, item in enumerate(row):

res = item + aIJ - aiJ\_list[i] - aIj\_list[j]

row\_residue.append(res)

bicluster\_residue.append(row\_residue)

#calculate the MSR of the bicuster

residue\_sum = 0.0

I = len(bicluster\_residue)

J = len(bicluster\_residue[0])

for row in bicluster\_residue:

row\_sum = 0.0

for item in row:

row\_sum = row\_sum + item\*\*2

residue\_sum = residue\_sum + row\_sum

MSR = residue\_sum/(I\*J)

biclusters\_MSR.append(MSR)

print(len(biclusters\_MSR))

print(sorted(biclusters\_MSR))

#remove biclusters with MSR>threshold(yeast=300, schisto=2100, lymphoma=1200)

new\_biclusters = []

new\_biclusters\_MSR = []

new\_biclusters\_indices = []

for j, item in enumerate(biclusters\_MSR):

if biclusters\_MSR[j]>=2100:

continue

else:

new\_biclusters.append(biclusters[j])

new\_biclusters\_indices.append(biclusters\_indices[j])

new\_biclusters\_MSR.append(biclusters\_MSR[j])

print()

print(len(new\_biclusters))

print(len(new\_biclusters\_MSR))

#ENCODING

data\_dimen = gene\_normalized\_array.shape

encoded\_biclusters = []

result\_file = open("C:/Users/THRIVESBIO/Documents/PD/Mrs Osuntokun/project/Biclustering/output/encoded.csv",'w')

wr = csv.writer(result\_file, dialect='excel')

for line in new\_biclusters\_indices:

g\_index = line[0]

c\_index = line[1]

gene\_code = np.zeros(data\_dimen[0], dtype = int)

condition\_code = np.zeros(data\_dimen[1], dtype = int)

for item in g\_index:

gene\_code[item] = 1

for item in c\_index:

condition\_code[item] = 1

encoded\_bi = gene\_code.tolist() + condition\_code.tolist()

encoded\_biclusters.append(encoded\_bi)

wr.writerow(encoded\_bi)

print(sorted(new\_biclusters\_MSR))

print('Encoded file and biclusters written')

Biclustering. PY

import csv

import math

import numpy as np

from nltk.cluster import KMeansClusterer, cosine\_distance

from statistics import mean

class file\_input(object):

def \_\_init\_\_(self,file\_path):

self.file\_path=file\_path

#function that extracts the gene expression value from text input file

def file\_input1(self):

path = self.file\_path

in\_file = []

input\_file = open(path)

for line in input\_file:

gene\_line = line.split()

in\_file.append(gene\_line)

numeric\_file = []

for line in in\_file:

new = [float(i) for i in line]

numeric\_file.append(new)

normalized\_array = np.array([np.array(i) for i in numeric\_file])

return normalized\_array

#function that extracts the gene expression value from csv input file

def file\_input2(self):

path = self.file\_path

in\_file = []

with open(path) as f:

input\_file = csv.reader(f, delimiter = '\t')

for line in input\_file:

gene\_line = line[0].split(",")

in\_file.append(gene\_line)

numeric\_file = []

for line in in\_file:

new = [float(i) for i in line]

numeric\_file.append(new)

normalized\_file = []

for line in numeric\_file:

ll=[]

for item in line:

if item <=0:

continue

norm = 100\*(math.log(10\*\*5\*item))

ll.append(norm)

normalized\_file.append(ll)

normalized\_array = np.array([np.array(i) for i in normalized\_file])

return normalized\_array

#function that performs the initial gene seed generation using kmeans

def gene\_seed(self, normalized\_array, n):

gene\_seed\_file = infile

no\_clusters = n

clusterer = KMeansClusterer(no\_clusters, cosine\_distance)

seed\_gene = clusterer.cluster(gene\_seed\_file, True, trace = True)

return seed\_gene

#function that performs the initial condition seed generation using kmeans

def condition\_seed(self, normalized\_array, n):

condition\_seed\_file = np.array(normalized\_array).T

no\_clusters = n

clusterer = KMeansClusterer(no\_clusters, cosine\_distance)

seed\_condition = clusterer.cluster(condition\_seed\_file, True, trace = True)

return seed\_condition

#function that extracts the genes positions for each cluster

def gene\_positions(self, gene\_clusters, n):

gene\_positions = []

for k in range(n):

gene\_pos = []

for i, item in enumerate(gene\_clusters):

if item != k:

continue

else:

gene\_pos.append(i)

gene\_positions.append(gene\_pos)

return gene\_positions

#function that reclusters seeds with higher number of genes

def recluster\_gene(gene\_positions,gene\_data):

countg = 0

while countg<len(gene\_positions):

if len(gene\_positions[countg])>15:

gene\_biclut\_array = [gene\_data[x] for x in gene\_positions[countg]]

gene\_clusterer = KMeansClusterer(2, cosine\_distance)

gene\_clusters = gene\_clusterer.cluster(gene\_biclut\_array, True, trace = True)

group1 = []

group2 = []

for i, item in enumerate(gene\_clusters):

if item ==0:

group1.append(gene\_positions[countg][i])

else:

group2.append(gene\_positions[countg][i])

del(gene\_positions[countg])

gene\_positions.append(group1)

gene\_positions.append(group2)

else:

countg +=1

#function that extracts the genes positions for each cluster

def condition\_positions(self, condition\_clusters, n):

condition\_positions = []

for k in range(n):

condition\_pos = []

for i, item in enumerate(condition\_clusters):

if item != k:

continue

else:

condition\_pos.append(i)

condition\_positions.append(condition\_pos)

return condition\_positions

#function that reclusters seeds with higher number of conditions

def recluster\_condition(condition\_positions,cond\_data):

countc = 0

while countc<len(condition\_positions):

if len(condition\_positions[countc])>5:

cond\_biclut\_array = [cond\_data[x] for x in condition\_positions[countc]]

condition\_clusterer = KMeansClusterer(2, cosine\_distance)

condition\_clusters = condition\_clusterer.cluster(cond\_biclut\_array, True, trace = True)

group1 = []

group2 = []

for i, item in enumerate(condition\_clusters):

if item ==0:

group1.append(condition\_positions[countc][i])

else:

group2.append(condition\_positions[countc][i])

del(condition\_positions[countc])

condition\_positions.append(group1)

condition\_positions.append(group2)

else:

countc +=1

#function that extracts biclusters

def extract\_biclusters(self, gene\_positions, condition\_positions, normalized\_array):

biclusters = []

biclusters\_indices = []

for line in gene\_positions:

gene\_index = line

for row in condition\_positions:

each\_bicluster = []

each\_bicluster\_index = []

for item in gene\_index:

each\_row = []

for column\_index in row:

each\_row.append(gene\_normalized\_array[item,column\_index])

each\_bicluster.append(each\_row)

each\_bicluster\_index.append(gene\_index)

each\_bicluster\_index.append(row)

biclusters.append(each\_bicluster)

biclusters\_indices.append(each\_bicluster\_index)

return biclusters, biclusters\_indices

#MSR function for each bicluster

def MSR\_function(self, biclusters):

biclusters\_MSR = []

for line in biclusters:

bicluster\_residue = []

aiJ\_list = []

aIj\_list = []

#calculate the mean of each row in the bicluster

for row in line:

row\_mean = mean(row)

aiJ\_list.append(row\_mean)

#calculate the mean of each column in the bicluster

bicluster\_transpose = np.array(line).T.tolist()

for column in bicluster\_transpose:

column\_mean = mean(column)

aIj\_list.append(column\_mean)

aIJ = mean(aiJ\_list) #mean of the bicluster

#calculate the residue of each element in the bicluster

for i, row in enumerate(line):

row\_residue = []

for j, item in enumerate(row):

res = item + aIJ - aiJ\_list[i] - aIj\_list[j]

row\_residue.append(res)

bicluster\_residue.append(row\_residue)

#calculate the MSR of the bicuster

residue\_sum = 0.0

I = len(bicluster\_residue)

J = len(bicluster\_residue[0])

for row in bicluster\_residue:

row\_sum = 0.0

for item in row:

row\_sum = row\_sum + item\*\*2

residue\_sum = residue\_sum + row\_sum

MSR = residue\_sum/(I\*J)

biclusters\_MSR.append(MSR)

return biclusters\_MSR

#function that removes biclusters with MSR>threshold(yeast=300, schisto=2100, lymphoma=1200)

def bicluster\_seeding(self, biclusters\_MSR, biclusters, biclusters\_indices):

for j, item in enumerate(biclusters\_MSR):

if biclusters\_MSR[j]>2100:

del(biclusters[j])

del(biclusters\_indices[j])

del(biclusters\_MSR[j])

return biclusters\_MSR, biclusters, biclusters\_indices

# function for ENCODING

def encoding(self, biclusters\_indices):

data\_dimen = gene\_normalized\_array.shape

encoded\_biclusters = []

result\_file = open("encoded.csv",'w')

wr = csv.writer(result\_file, dialect='excel')

for line in biclusters\_indices:

g\_index = line[0]

c\_index = line[1]

gene\_code = np.zeros(data\_dimen[0], dtype = int)

condition\_code = np.zeros(data\_dimen[1], dtype = int)

for item in g\_index:

gene\_code[item] = 1

for item in c\_index:

condition\_code[item] = 1

encoded\_bi = gene\_code.tolist() + condition\_code.tolist()

encoded\_biclusters.append(encoded\_bi)

wr.writerow(encoded\_bi)

return encoded\_biclusters