```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec  7 14:59:02 2021

@author: Utilisateur aka Kévin-Lâm
"""
# importing modules
import numpy as np
from tqdm import tqdm
import time as t
##variables
nb_class=3609
size_fault_matrix=(55,108)

t1=t.time()
def sub_bool(a,b):
    if a :
        if b :
            return 0
        else :
            return 1
    else :
        if b :
            return 1
        else :
            return 0


def norme_1(matrix_1,matrix_2):
    distance = 0
    dimensions=np.shape(matrix_1)
    if dimensions!=np.shape(matrix_2):
        return ('Pas la même taille')
    for i in range(dimensions[0]) :
        for j in range(dimensions[1]):
            distance+=sub_bool(matrix_1[i,j],matrix_2[i,j])
    return distance

def insert_sorted_list_dich(elem,L):
    a=0
    length=len(L)
    b=length
    x=(a+b)//2
    if elem[0]>=L[b-1][0]:
        L.append(elem)
        return

    while a!=b:
       if elem[0]>=L[x][0]:
           a=x+1
           x=(a+b)//2

       else:
           b=x
           x=(a+b)//2
    L.insert(a, elem)
    return
##regroup all permanent fault in the same matrix --> easier to operate on them
"""
matrix_file_opened= open('fault_matrix_file2.txt',"r")
matrix_file=matrix_file_opened.readlines()
matrix_grouped=np.zeros((size_fault_matrix[0],size_fault_matrix[1],nb_class),dtype=bool)
for k in tqdm(range(nb_class)):
    for cpt in range(size_fault_matrix[0]):
        for j in range(size_fault_matrix[1]):
            matrix_grouped[cpt,j,k]=int(matrix_file[(k+1)+k*size_fault_matrix[0]+cpt][j])

np.savez_compressed('permanent_fault',matrix=matrix_grouped)
"""
###Creation de classe :
```

```
nb_metaclass=120
avg_element_per_class=nb_class//120


# easy method: we take the first avg_element_per_class closest matrix to the first matrix
# then we do the same for the second etc.




already_classified=np.zeros((nb_class),dtype=bool)
metaclassnumber=0

matrix_permanents=np.load('permanent_fault.npz')['matrix']
List_metaclass= {}

for i in tqdm(range(nb_class)): # we create a sorted list of elements
    if not already_classified[i]:
        L_chosen=[]
        already_classified[i]=True
        metaclassnumber+=1
        print(metaclassnumber,i)
        studied_matrix=matrix_permanents[:,:,i]
        L_dist=[(0,i)]
        length_L_dist=1
        for j in range(i,nb_class):
            if not already_classified[j]:
                dist=norme_1(studied_matrix,matrix_permanents[:,:,j])
                #dist=np.linalg(studied_matrix-matrix_permanents[:,:,j])
                if length_L_dist<=avg_element_per_class :
                    insert_sorted_list_dich((dist,j),L_dist)
                    length_L_dist+=1
                elif length_L_dist>avg_element_per_class and dist<L_dist[-1][0] :
                    L_dist=L_dist[0:length_L_dist-1]
                    insert_sorted_list_dich((dist,j),L_dist)

                #print(L_dist) #a way to improve is to add element at the right place
        #so the list remain always sorted, (we can use dichotomy to add the dist at the right
        #place, it's done now : before we added all the distances in L_dist and then sort the list after

        for chosen in range(min(avg_element_per_class,len(L_dist))):
            L_chosen.append(L_dist[chosen][1])
            already_classified[L_dist[chosen][1]]=True
        List_metaclass["class%s" %metaclassnumber] = L_chosen


t2=t.time()
duration=t2-t1

#To optimize There is nb_class! distances to calculate then we can sort it ?
#and then we classify each matrix according to those distances
```