1. 執行環境: VS code
2. 程式語言: Python 3.11.5
3. 執行需求:

* 文章資料放置在./data/ 路徑之下
* 8.txt,13.txt,20.txt最終結果會放置與pa3.py同一層資料夾
* 需要pip install numpy, nltk, pandas

1. 作業處理邏輯
2. 將資料讀進，並各自做前處理(preprocessing)後算出tf與df，分別存進word\_tf與word\_df變數中

def preprocessing(file):

    file = file.lower()

    file = Tokenize(file)

    file = RemoveStopwords(file,StopWords)

    file = Porter\_Stemmmer(file)

    return file

def TermFrequency(start,end):

    word\_tf =[]

    word\_df = {}

    for i in range(start,end+1):

        i=str(i)

        print(i)

        path = f'./data/{i}.txt'

        tdf = OpenFile(path)

        tdf = preprocessing(tdf)

        tdf = [word for word in tdf if word!='']

        for word in tdf:

            if word not in word\_tf:

                temp =[]

                frequency = tdf.count(word)

                temp.append(i)

                temp.append(word)

                temp.append(frequency)

                word\_tf.append(temp)

        for word in set(tdf):

            if word in word\_df:

                word\_df[word] +=1

            else :

                word\_df[word] =1

    return word\_tf, word\_df

word\_tf, word\_df = TermFrequency(1,size)

1. 建立字典並編號，存在index\_dict變數裡

def get\_index\_dict(word\_df):

    index\_dict={}

    index =0

    for term in word\_df:

        index\_dict[term] = index

        index +=1

    return index\_dict

index\_dict = get\_index\_dict(word\_df)

1. 將文件轉換成tf-idf vector，並撰寫可以計算cosine similarity的公式

def get\_tf\_vector(word\_tf,index\_dict):

    tf\_vectors=[]

    for i in range(1,size+1): #

        tf\_vector = np.array([0] \* len(index\_dict), dtype=float)

        for row in word\_tf:

            if row[0] == str(i):

                word = row[1]

                tf\_vector[index\_dict[word]] = row[2]

            else:

                continue

        tf\_vectors.append([i, tf\_vector])

    return tf\_vectors

tf\_vectors = get\_tf\_vector(word\_tf,index\_dict)

import math

def get\_tf\_idf\_vector(tf\_vectors, word\_df, index\_dict):

    idf\_vector = np.array([0] \* len(index\_dict), dtype=float)

    for word, df in word\_df.items():

        idf = math.log(len(tf\_vectors) / df, size)

        idf\_vector[index\_dict[word]] = idf

    tf\_idf\_vectors = list()

    for tf\_vector in tf\_vectors:

        idx = tf\_vector[0]

        print(tf\_vector[1])

        tf\_idf = np.array(tf\_vector[1], dtype=float) \* np.array(idf\_vector, dtype=float)

        tf\_idf\_unit = tf\_idf / np.linalg.norm(tf\_idf)

        tf\_idf\_vectors.append([idx, tf\_idf\_unit])

    return tf\_idf\_vectors

tf\_idf\_vectors = get\_tf\_idf\_vector(tf\_vectors,word\_df,index\_dict)

doc\_vectors = np.array([each[1] for each in tf\_idf\_vectors])

def extract\_vector(doc\_id, doc\_vectors):

    return doc\_vectors[doc\_id-1]

def cosine(doc\_x, doc\_y):

    vector\_x = extract\_vector(doc\_x, doc\_vectors)

    vector\_y = extract\_vector(doc\_y, doc\_vectors)

    cosine\_sim = float(np.dot(vector\_x, vector\_y))

    return cosine\_sim

te = cosine(1,2)

1. 使用complete-link策略並按照HAC的algothrim撰寫其餘程式碼

def get\_max\_similarity(C, I, size):

    max\_sim = -1

    doc\_i, doc\_m = -1, -1

    for i in range(size):

        if I[i] != 1:

            continue

        for m in range(size):

            if I[m] == 1 and i != m:

                if max\_sim < C[i][m]:

                    max\_sim = C[i][m]

                    doc\_i, doc\_m = i, m

    return doc\_i, doc\_m

C = np.array([[cosine(x, y) for x in range(1, size+1)] for y in range(1, size+1)])

I = np.array([1]\* size)

A = []

for k in range(1,size-1):

    i, m = get\_max\_similarity(C, I, size)

    A.append([i ,m])

    for j in range(size):

        C[i][j] = min(cosine(i, j), cosine(m, j))

        C[j][i] = min(cosine(j, i), cosine(j, m))

    I[m] = 0

1. 保存結果，分別存入8.txt,13.txt,20.txt

def write\_result(hac\_dict, cluster\_num):

    with open(f"./{cluster\_num}.txt", "w") as f:

        for k, v in hac\_dict.items():

            for doc\_id in sorted(v):

                f.write(f"{doc\_id+1}\n")

            f.write("\n")

hac\_dict = {str(i) : [i] for i in range(size)}

for doc\_i, doc\_m in A:

    new\_element = hac\_dict[str(doc\_m)]

    hac\_dict.pop(str(doc\_m))

    hac\_dict[str(doc\_i)] += new\_element

    if len(hac\_dict) == 20:

        write\_result(hac\_dict, 20)

    if len(hac\_dict) == 13:

        write\_result(hac\_dict, 13)

    if len(hac\_dict) == 8:

        write\_result(hac\_dict, 8)