R09725049 吳延東 pa2

檔案內所 import package:

1. nltk.stem import PorterStemmer

2. os

3. pandas

4. numpy

Class of Preprocess is part of pa1

In [1]:
```python
from nltk.stem import PorterStemmer

class Preprocess():
    ## this is for each document
    def __init__(self):
        self.inittoken_list = []
        self.http_dic = {"ALL": []}
        self.number_removed_list = []
        self.number_dic = {"ALL":[]} ## recorded in init order
        self.stemmed_list = []
        self.punctuation_list = [".", "'", '"', "?", ",", ")", "(", "@", "%", "$", "*",
                                  "-", "_", "/", "!", "#", "^", "&", "`", ":", ";"]
        self.poter = PorterStemmer()
        self.stopward_list = []

        self.stopward_dic = {"ALL":[]}
        init_stopward_list = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
"you'd", 'your', 'yours', 'yourself', 'yourselves',
                              'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'its
elf', 'they', 'them', 'their', 'theirs', 'themselves',
                              'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'w
as', 'were', 'be', 'been', 'being',
                              'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'i
f', 'or', 'because', 'as', 'until', 'while', 'of', 'at',
                              'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'afte
r', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
                              'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'wher
e', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
                              'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
'too', 'very', 's', 't', 'can', 'will', 'just',
                              'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't",
                              'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
'mightn', "mightn't", 'mustn', "mustn't", 'needn',
                              "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
"won't", 'wouldn', "wouldn't"]
        self.stopward_adding( init_stopward_list)
        self.stopwarded_list = []
        self.word_dic = {} ## format:{term:[index_list]}
        return None

    def read_file(self, storage_place):
        #print(1)
        if len(self.inittoken_list) != 0:
            return "you have already put some data in here"
        ## vertify type of input
        if not isinstance(storage_place, str):
            print("you should input where you store your document in string type.")
            return False
        ## make document in to a list of list of strings, seperated in lines
        storage_place = storage_place.strip("/")
        document_list = open(storage_place, 'rt').readlines()

        ## make document into a single list of string
        valid_index = 0
        for line in document_list:
            start_flag = 0
            for stop_flag in range( len(line)):
                valid_flag = False
                if line[ stop_flag] == ' ' :
                    word = line[ start_flag : stop_flag]
                    valid_flag = self.preprocess_word(word, valid_index)
                    start_flag = stop_flag + 1
                if line[-1] == '.' and stop_flag == len(line)-1:
                    # check the last word for each line
                    word = line[ start_flag : -1]
                    valid_flag = self.preprocess_word(word, valid_index)

                if valid_flag:
                    ## flag is true if word is valid
                    valid_index = valid_index + 1

        return True

    def preprocess_word(self, word, valid_index):
        ## 應在這裡把 字串list、字典 建好
        if self.http_remove(word):
            return False
        if self.number_remove(word):
            return False
        #self.minus_split( voca_index)
        pun_removed = self.punctuation_remove(word)
        if self.len_filter(pun_removed):
            return False
        stemmed =  self.stemming(pun_removed)
        if self.stopwording(stemmed):
            return False
        self.word_dic_create(stemmed, valid_index)

        return True

    def http_remove(self, word):
        ## true if this word is a website address, and add it into http_dic
        flag = 0

        if "http" == word[:4] or "www" == word[: 3]:
        ## first 4 chars in word == http, or first 3 chars in word == www
            flag = 1

        self.http_index = 0

        if flag:
            if not word in self.http_dic:
                self.http_dic["ALL"].append(word)
                self.http_dic[ word] = []
            self.http_dic[ word].append(self.http_index)
            self.http_index = self.http_index + 1
            return True
        return False

    def number_remove(self, word):
        ## true if 're is number in the word, and add it into number_dic
        flag = 0
        for char in word:
            if ord(char) < 58 and ord(char) > 47:
            ## ASCII for numbers : 48~57
                flag = 1
                break
        self.number_index = 0
        if flag:
            if not word in self.number_dic:
                self.number_dic["ALL"].append(word)
                self.number_dic[ word] = []
            self.number_dic[ word].append(self.number_index)
            self.number_index = self.number_index + 1
            return True
        return False

    def punctuation_remove(self, word):
        for pun in self.punctuation_list:
            if pun in word :
                word = word.replace(pun, '')
        return word

    def len_filter(self, word):
        if len(word) < 3:
            return True
        return False

    def stemming(self, word):
        stemmed = self.poter.stem( word)
        return stemmed

    def stopwording(self, word):
        ## true if the word is stopword
        if word in self.stopward_dic:
            #self.stopward_dic['ALL'].append( voca_index)
            #self.stopward_dic[ dest_document[voca_index]].append(voca_index)
            return True

        ## add normal word into stopworded_list
        self.stopwarded_list.append(word)
        return False

    def word_dic_create(self, word, index):
        if word in self.word_dic:
            self.word_dic[word].append(index)
        else:
            self.word_dic[word] = [index]

        return True

    def minus_split(self):####### INCOMPLEPE #######
        for voca_index in  range( len( self.inittoken_list)):
            if "-" in self.inittoken_list[voca_index]:
                temp = self.inittoken_list[voca_index].split("-")
                self.inittoken_list.append(temp)
                self.inittoken_list[voca_index] = self.inittoken_list[voca_index].replace("-", "")
        return None


    def stopward_adding(self, new_ward_list):
        ## check for type of list
        if not isinstance(new_ward_list, list):
            print("want a list. in stopward_adding")
            return False
        for stopward in new_ward_list:
            ## check for type of each ward in list
            if not isinstance(stopward, str):
                print("want a list of string. in stopward_adding")
                return False
            ## stem and add
            stemmed_stopward = self.poter.stem( stopward)
            if not stemmed_stopward in self.stopward_dic:
                self.stopward_list.append( stemmed_stopward)
                self.stopward_dic.update({stemmed_stopward: []})
        #self.stopward_flag[0] = self.stopward_flag[0] +1
        return 0

    def punctuation_adding(self, new_pun):

        return 0

    def save_result(self):
        with open("R09725049_result.txt" , "w") as text_file:
            text_file.write(str(self.stopwarded_list))
        return "file saved"
```

pa2

In [2]:
```python
import os
import pandas as pd
import numpy as np

class Dictionary():
    ## this is for whole file
    def __init__(self):
        self.preprocess_list = []
        return None

    def preprocess_all_file(self, pre_path):
        if not isinstance(pre_path, str):
            print("you should input where you store your document in string type.")
            return False
        ## make document in to a list of list of strings, seperated in lines
        pre_path = pre_path.strip("/")

        ## get the list of document in pre_path
        all_file_list = os.listdir(pre_path)

        ## preprocess all document, and store Class of Preprocess in preprocess_list
        for docu_index in range(len(all_file_list)):
            self.preprocess_list.append(Preprocess())
            self.preprocess_list[docu_index].read_file(pre_path + "/" +  str(docu_index +1) + ".txt")
        return None

    ################################ PART 1 ###########################
    ################################ PART 1 ###########################
    ################################ PART 1 ###########################
    ################################ PART 1 ###########################
    ################################ PART 1 ###########################


    def document_frequency(self):
        ### .isin() is too slow, so we use .in() with dic instead
        ## 1. construct a dictionary and DataFrame of terms
        ## 2. sort the term of the DataFrame in ascending order, reorder the index in dictionary
        ## 3. append t_index into DataFrame

        self.term_dic = {}
        self.docu_freq_df = pd.DataFrame(columns = ["term", "df", "exsist_docu_list"])
        self.docu_freq_df.astype({'df': 'int32'}).dtypes

        ## 1
        docu_index = 0
        term_index = 0
        for document in self.preprocess_list:
            for term in document.word_dic:
                ## put term into term_dic, and append it into docu_freq_df

                if term in self.term_dic:
                    ## real index is in term_index_list[0], which .to_list() return a list
                    ## raise df by 1, and append docu_index into exsist_docu_list
                    self.docu_freq_df.at[ self.term_dic[term], "df"] += 1
                    self.docu_freq_df.at[ self.term_dic[term], "exsist_docu_list"].append(docu_index)

                else:
                    self.docu_freq_df = self.docu_freq_df.append(pd.DataFrame([[term, 1, [docu_index]]],
                                                                              columns = ["term", "df", "exsist_docu_list"]),
                                                                 ignore_index=True)
                    self.term_dic[term] = term_index ## this value will be index of term in dataframe
                    term_index += 1
            docu_index += 1

        ## 2
        self.docu_freq_df = self.docu_freq_df.sort_values(by=['term']).reset_index().drop(["index"], axis= 1)

        for index in range(self.docu_freq_df.shape[0]):
            ## have to revalue term_dic to fit term_index of docu_freq_df
            self.term_dic[self.docu_freq_df.loc[index, "term"]] = index

        ## 3
        t_index = pd.DataFrame( np.arange(self.docu_freq_df.shape[0]), columns=["t_index"])
        self.docu_freq_df = pd.concat([t_index, self.docu_freq_df], axis = 1)

        return None

    def save_document_frequency(self):
        with open("dictionary.txt" , "w") as text_file:
            text_file.write(self.docu_freq_df.loc[:,"t_index": "df"].to_string(index= False))
            ## output only "t_index", "term", "df" column
        return "file saved"

    ################################ PART 2 ###########################
    ################################ PART 2 ###########################
    ################################ PART 2 ###########################
    ################################ PART 2 ###########################
    ################################ PART 2 ###########################


    def docu_tf_idf(self):
        ## docu_tf_idf_list : list of ndarray in order of docu_index
        self.docu_tf_idf_list = []
        for docu_index in range(len(self.preprocess_list)):

            first = 1
            for term in self.preprocess_list[docu_index].word_dic :
                if first:
                    ## ndarray format: [t_index, tf_idf]
                    term_nparray = np.array([[int(self.term_dic[term]), self.tf_idf(term, docu_index)]])
                    first = 0
                else:
                    term_nparray = np.append(term_nparray, [[int(self.term_dic[term]), self.tf_idf(term, docu_index)]], axis=0)
            ## sort the ndarry by column 0
            term_nparray = term_nparray[np.argsort(term_nparray[:, 0])]
            self.docu_tf_idf_list.append(term_nparray)
        return None

    def tf_idf(self, term, docu_index):
        tf = self.term_frequency(term, docu_index)
        idf = self.inverse_document_frequency(term)
        tf_idf = tf * idf
        return tf_idf

    def term_frequency(self, term, docu_index):
        tf = len(self.preprocess_list[docu_index].word_dic[term])
        return tf

    def inverse_document_frequency(self, term):
        term_index = self.term_dic[term]
        idf = np.log10(len(self.preprocess_list) / self.docu_freq_df.loc[term_index, "df"])
        return idf

    def save_tf_idf(self, docu_index):
        head = np.array([[self.docu_tf_idf_list[docu_index].shape[0] #F
                          , ""], ["t_index", "tf_idf"]])
        whole = np.concatenate((head, self.docu_tf_idf_list[docu_index]), axis = 0)
        np.savetxt(str(docu_index) + '.txt', whole, delimiter='\t', fmt = "%s")
        return "file saved"

    ################################ PART 3 ###########################
    ################################ PART 3 ###########################
    ################################ PART 3 ###########################
    ################################ PART 3 ###########################
    ################################ PART 3 ###########################

    def tf_idf_matrix_full(self):
        ## shape of matrix: [term_i, docu_i], init by 0
        self.tf_idf_matrix = np.zeros(shape = (self.docu_freq_df.shape[0] , len(self.preprocess_list)))

        for docu_index in range(len(self.preprocess_list)):
            for index in range(self.docu_tf_idf_list[docu_index].shape[0]):
                term_index = int(self.docu_tf_idf_list[docu_index][index, 0])
                ## replace 0 with tf_idf value in each term(row) by document(column)
                self.tf_idf_matrix[term_index, docu_index] = self.docu_tf_idf_list[docu_index][index, 1]
        return None

    def cos_similarity(self, docu_index_1, docu_index_2):

        ## get vactor of each document
        docu_1_vec = self.tf_idf_matrix[:, docu_index_1]
        docu_2_vec = self.tf_idf_matrix[:, docu_index_2]

        ## Dot and norm
        dot = sum(a*b for a, b in zip(docu_1_vec, docu_2_vec))
        norm_a = sum(a*a for a in docu_1_vec) ** 0.5
        norm_b = sum(b*b for b in docu_2_vec) ** 0.5

        ## Cosine similarity
        similarity = dot / (norm_a*norm_b)

        return similarity
```

In [3]:
```python
pa2 = Dictionary()
pa2.preprocess_all_file("D:\Desktop\IR\PA2\IRTM")
```

part 1

In [4]:
```python
pa2.document_frequency()
pa2.save_document_frequency()
```

Out[4]: 'file saved'

part 2

In [5]:
```python
pa2.docu_tf_idf()
pa2.save_tf_idf(0)
```

Out[5]: 'file saved'

In [6]:
```python
pa2.tf_idf_matrix_full()
```

part 3 : cosine similarity between document 1 and 2

In [7]:
```python
print(pa2.cos_similarity(0, 1))
```

0.19983854846589438