**程式的原始介紹網頁:**

<https://spaces.ac.cn/archives/3491>

**程式代碼 :**

import numpy as np # 引入 numpy 庫，用於數據處理

import pandas as pd # 引入 pandas 庫，用於數據分析

import re # 引入 re 庫，用於正則表達式操作

from numpy import log # 從 numpy 庫中引入 log 函數，用於計算信息熵

import os # 引入 os 庫，用於操作系統路徑

# 詢問使用者來源文件的路徑

file\_path = input("Enter the path of the source file: ")

# 獲取用戶輸入的文件路徑

# 詢問使用者詞語的最小和最大長度

min\_len = int(input("Enter the minimum length of words: ")) # 獲取用戶輸入的詞語最小長度

max\_len = int(input("Enter the maximum length of words: ")) # 獲取用戶輸入的詞語最大長度

# 詢問使用者 min\_count，min\_support 和 min\_s

"""

min\_count 錄取詞語最小出現次數

min\_support 錄取詞語最低支持度，1代表著隨機組合

min\_s 錄取詞語最低信息熵，越大說明越有可能獨立成詞

"""

min\_count = int(input("Enter the minimum count: ")) # 獲取用戶輸入的詞語最小出現次數

min\_support = int(input("Enter the minimum support: ")) # 獲取用戶輸入的詞語最低支持度

min\_s = int(input("Enter the minimum entropy: ")) # 獲取用戶輸入的詞語最低信息熵

f = open(file\_path, 'r', encoding='utf-8') # 以讀取模式打開用戶指定的文件

s = f.read() # 讀取文件內容為一個字符串

# 定義要去掉的標點字

drop\_dict = [u'，', u'\n', u'。', u'、', u'：', u'(', u')', u'[', u']', u'.', u',', u' ',

u'\u3000', u'”', u'“', u'？', u'?', u'！', u'‘', u'’', u'…', u'「', u'」']

for i in drop\_dict: # 對每一個要去掉的標點字

s = s.replace(i, '') # 在字符串中替換掉該標點字

# 為了方便調用，自定義了一個正則表達式的詞典

myre = {i:f'({"."\*i})' for i in range(min\_len, max\_len+1)}

t=[] # 初始化一個空列表，用於保存結果

t.append(pd.Series(list(s)).value\_counts()) # 對字符串進行逐字統計

tsum = t[0].sum() # 統計總字數

rt = [] # 初始化一個空列表，用於保存結果

for m in range(min\_len, max\_len+1): # 對於每一個詞語長度

print(u'正在生成%s字詞...'%m) # 輸出提示信息

t.append([]) # 在結果列表中添加一個空列表

for i in range(m): # 對於每一個位置

t[m-1] = t[m-1] + re.findall(myre[m], s[i:]) # 生成所有可能的m字詞

t[m-1] = pd.Series(t[m-1]).value\_counts() # 對詞語進行逐詞統計

t[m-1] = t[m-1][t[m-1] > min\_count] # 對詞語進行最小次數篩選

tt = t[m-1][:] # 複製一份統計結果

for k in range(m-1): # 對於每一個位置

qq = np.array(list(map(lambda ms: tsum\*t[m-1][ms]/t[m-2-k][ms[:m-1-k]]/t[k][ms[m-1-k:]], tt.index))) > min\_support # 進行最小支持度篩選

tt = tt[qq] # 更新統計結果

rt.append(tt.index) # 將結果添加到結果列表中

def cal\_S(sl): # 定義信息熵計算函數

return -((sl/sl.sum()).apply(log)\*sl/sl.sum()).sum()

for i in range(min\_len, max\_len+1): # 對於每一個詞語長度

print(u'正在進行%s字詞的最大熵篩選(%s)...'%(i, len(rt[i-2])))

# 輸出提示信息

pp = [] # 初始化一個空列表，用於保存所有的左右鄰結果

for j in range(i+2): # 對於每一個位置

pp = pp + re.findall('(.)%s(.)'%myre[i], s[j:])

# 生成所有可能的左右鄰結果

pp = pd.DataFrame(pp).set\_index(1).sort\_index()

# 將結果轉換為數據框並排序

index = np.sort(np.intersect1d(rt[i-2], pp.index))

# 進行交集操作

# 下面兩句分別是左鄰和右鄰信息熵篩選

index = index[np.array(list(map(lambda s: cal\_S(pd.Series(pp[0][s]).value\_counts()), index))) > min\_s]

rt[i-2] = index[np.array(list(map(lambda s: cal\_S(pd.Series(pp[2][s]).value\_counts()), index))) > min\_s]

# 下面都是輸出前處理

for i in range(len(rt)):

t[i+1] = t[i+1][rt[i]]

t[i+1] = t[i+1].sort\_values(ascending = False)

# 獲取源文件的目錄

dir\_path = os.path.dirname(file\_path)

# 保存結果並輸出

with pd.ExcelWriter(os.path.join(dir\_path, f'result\_entropy{min\_s}\_min\_count{min\_count}\_min\_support{min\_support}.xlsx')) as writer:

for i in range(len(t[1:])):

df = pd.DataFrame(t[i+1])

df.columns = ['Words count']

df.index.name = 'Words'

df.to\_excel(writer, sheet\_name=f'{i+2} character words')

print(f"Results saved to 'result\_entropy{min\_s}\_min\_count{min\_count}\_min\_support{min\_support}.xlsx' in the same directory as the source file.")

這個腳本是一種基於信息熵概念的中文詞語分割算法。它的設計目的是在文檔中識別有意義的中文詞語。以下是逐步解釋：

1. **輸入參數**：腳本首先向用戶詢問幾個輸入參數：
   * file\_path：包含要分析的文本的源文件的路徑。
   * min\_len 和 max\_len：要考慮的詞語的最小和最大長度。
   * min\_count：詞語應出現的最小次數。
   * min\_support：最小支持度，這是一種衡量詞語出現次數與字符隨機組合次數的比例。
   * min\_s：最小熵，這是衡量詞語的隨機性的指標。熵越高，該詞語越可能是一個獨立的詞。
2. **預處理**：腳本從文件中讀取文本並去除標點符號和數字。
3. **生成詞語**：對於從 min\_len 到 max\_len 的每個可能的詞語長度，腳本生成所有可能的該長度的詞語。
4. **篩選詞語**：然後腳本篩選出不滿足 min\_count 和 min\_support 標準的詞語。
5. **計算熵**：對於每個剩餘的詞語，腳本通過查看文本中詞語前後出現的字符來計算左熵和右熵。熵是一種衡量隨機性的指標；熵越高，該詞語越可能是一個獨立的詞。
6. **熵篩選**：腳本篩選出不滿足 min\_s 熵標準的詞語。
7. **輸出**：最後，腳本將識別出的詞語及其計數保存到一個 Excel 文件中。

這個腳本是一種簡單而有效的進行中文詞語分割的方式。它基於一個想法，即一個真實的詞語比隨機的字符組合更可能出現在文本中，並且在該詞語前後有特定的字符出現。使用熵為此添加了一種概率性方面，使得該算法更為強大。然而，請注意，這是一種基本的算法，可能無法完美地適用於所有文本，特別是對於有許多罕見或新詞語的文本。進階的算法可能使用機器學習技術和大型語言模型以獲得更好的性能。

"""

4. 篩選詞語：然後腳本篩選出不滿足 min\_count 和 min\_support 標準的詞語。

5.計算熵：對於每個剩餘的詞語，腳本通過查看文本中詞語前後出現的字符來計算左熵和右熵。熵是一種衡量隨機性的指標；熵越高，該詞語越可能是一個獨立的詞。

"""

請把這兩點儘量詳細解釋一下 :

當然，我會詳細解釋這兩點：

1. **篩選詞語**：在生成所有可能的詞語之後，腳本會進行篩選，只保留那些出現次數超過 min\_count 的詞語。這是為了確保我們關注的詞語在文本中有足夠的出現次數，從而使其統計特性更為穩定。此外，腳本還會檢查詞語的支持度是否超過 min\_support。支持度是詞語出現的次數與詞語由隨機字符組成的預期次數的比值。如果一個詞語的支持度高，那麼這個詞語不太可能是隨機字符的組合。
2. **計算熵**：對於每個剩餘的詞語，腳本會計算其左熵和右熵。這裡的左熵和右熵是指詞語左側和右側字符的熵。熵是一種衡量隨機性的指標，如果一個詞語的左熵和右熵都很高，那麼我們可以認為這個詞語的左側和右側字符分布很均勻，換句話說，這個詞語可以和很多其他的詞語組合在一起，因此這個詞語很可能是一個獨立的詞。腳本會根據 min\_s 這個閾值來篩選出熵大於這個閾值的詞語。

"支持度是詞語出現的次數與詞語由 "隨機字符組成" 的預期次數的比值" 這裏的 "隨機字符組成" , 是什麼意思 ?

“隨機字符組成” 是指詞語由任意字符組成的情況。在這裡，我們假設所有的字符組合都是等可能的，也就是說，每個字符出現的機會都是相同的。因此，一個詞語由隨機字符組成的預期次數，就是在這種等可能的情況下，這個詞語出現的次數。

然而，在實際的語言中，並非所有的字符組合都會出現，並且即使是出現的字符組合，其出現的頻率也會有所不同。因此，如果一個詞語在文本中出現的次數遠超過了由隨機字符組成的預期次數，那麼我們可以認為這個詞語不是隨機出現的，而是有一定的意義。這就是支持度的概念。希望這個解釋能對您有所幫助！如果您還有其他問題，請隨時提問。我會很樂意為您解答。

**對於上面的解釋, 可否給我一個例子**

讓我們以一個簡單的例子來說明 “隨機字符組成” 和 “支持度” 的概念。

假設我們有一個只包含兩個字符的語言，這兩個字符是 “A” 和 “B”。在這種情況下，如果我們考慮所有可能的兩字符組合，那麼我們有四種可能的組合：“AA”，“AB”，“BA”和“BB”。如果這些組合都是等可能的，那麼我們預期每種組合在文本中出現的次數應該是相同的。

然而，如果我們觀察一個實際的文本，例如 “ABABABABA”，我們會發現 “AB” 的出現次數遠超過了其他的組合。在這種情況下，“AB” 的支持度就會很高，因為它的實際出現次數遠超過了隨機情況下的預期出現次數。這就意味著 “AB” 不太可能是隨機出現的，而更有可能是一個有意義的詞語。

"計算熵：對於每個剩餘的詞語，腳本會計算其左熵和右熵。這裡的左熵和右熵是指詞語左側和右側字符的熵。熵是一種衡量隨機性的指標，如果一個詞語的左熵和右熵都很高，那麼我們可以認為這個詞語的左側和右側字符分布很均勻，換句話說，這個詞語可以和很多其他的詞語組合在一起，因此這個詞語很可能是一個獨立的詞。腳本會根據 min\_s 這個閾值來篩選出熵大於這個閾值的詞語。"

**對於上面的解釋, 可否給我一個例子:**

讓我們以一個簡單的例子來說明 “左熵” 和 “右熵” 的概念。

假設我們有一個詞語 “學習”，並且我們有以下的句子：

1. 我喜歡學習數學。
2. 他正在學習英語。
3. 我們都在學習編程。

在這些句子中，“學習” 的左側字符有 “歡”、“在” 和 “在”，右側字符有 “數”、“英”和“編”。我們可以看到，左側和右側的字符分布相對均勻，沒有哪個字符特別常見。因此，“學習” 的左熵和右熵都會相對較高，這表明 “學習” 很可能是一個獨立的詞。

然而，如果我們考慮一個不太可能是獨立詞的字符組合，例如 “在學”，我們可能會發現它的左側和右側字符分布不均勻。例如，如果 “在學” 總是出現在 “我們都在學習” 這樣的句子中，那麼它的左側字符總是 “都”，右側字符總是 “習”。在這種情況下，“在學” 的左熵和右熵都會較低，這表明 “在學” 不太可能是一個獨立的詞。