compute_similarity

2023 年 2 月 18 日

1 计算字母相似度

1.1 介绍

这个文件中我们进行了三个方面的工作

1. 利用开源的字典计算两个字母的相关程度

计算方法为：在字典数据集中，统计两两字母相邻的组合出现次数，然后计算当个字母分别在前面所得组合的出现次数，用两个字母的出现次数向量进行余弦相似度计算。

2. 定义了一种计算一个单词的正常性的方法

- 通过计算一个单词的字母的相似度，得到一个相似度矩阵
- 将给定单词的字母与下一个字母的相似度依次相加，（由于我们的单词是定长的，所以一共是四个相似度相加即可）
- 如此得到一个相似度之和，我们可以知道，一定程度上，一共单词前后两个字母的相似度越高，那么这个单词的正常性越高。

3. 对原数据进行处理（参考了这个数据 https://bert.org/assets/posts/wordle/words.json）

- 去除异常值（改掉了四个单词，可以说是参考了网上的那天给的具体的正确单词值）
- 对于每个单词，计算其正常性，并将其加入到原数据中（为后续神经网络做准备）

1.2 利用开源的字典计算两个字母的相关程度

[1]:
```
# 导入 numpy 库
import numpy as np
```

[2]:
```
# 打开英文字典文件
with open("words_clean.txt", "r") as words:
    # 创建一个空字典来存储每个字母出现的次数
    letter_count = {}
```

```
    # 创建一个空字典来存储每对相邻字母出现的次数
    pair_count = {}

    # 遍历每个单词
    for word in words:
        # 去掉单词末尾的换行符
        word = word.strip()
        # 遍历单词中的每个字母
        for i in range(len(word)):
            # 获取当前字母
            letter = word[i]
            # 如果字母不在 letter_count 中，就把它加入，并初始化为 0
            if letter not in letter_count:
                letter_count[letter] = 0
            # 把当前字母出现的次数加 1
            letter_count[letter] += 1

            # 如果不是最后一个字母，就获取下一个字母，并组成一对相邻字母
            if i < len(word) - 1:
                next_letter = word[i+1]
                pair = letter + next_letter
                # 如果相邻字母不在 pair_count 中，就把它加入，并初始化为 0
                if pair not in pair_count:
                    pair_count[pair] = 0
                # 把当前相邻字母出现的次数加 1
                pair_count[pair] += 1
```

[3]:
```
# 显示 letter_count 中的前 10 个 item
print("letter_count:\n", list(letter_count.items())[:10])
# 显示 pair_count 中的前 10 个 item
print("pair_count:\n", list(pair_count.items())[:10])
```

```
letter_count:
 [('a', 330309), ('l', 216916), ('s', 273315), ('b', 71757), ('e', 412371),
('r', 269353), ('g', 90238), ('c', 164370), ('h', 103087), ('n', 275977)]
pair_count:
 [('aa', 463), ('al', 43636), ('as', 18572), ('ab', 14013), ('be', 10451),
```

```
('er', 72275), ('rg', 4129), ('ac', 19190), ('ch', 22188), ('he', 20824)]
```

[4]:
```python
# 创建一个空列表来存储所有可能的 26 个英文字母组合（共有 26*26=676 种）
letter_combinations = []
# 遍历所有可能的第一个字母（从 a 到 z）
for first_letter in "abcdefghijklmnopqrstuvwxyz":
    # 遍历所有可能的第二个字母（从 a 到 z）
    for second_letter in "abcdefghijklmnopqrstuvwxyz":
        # 组合两个字母，并添加到列表中
        combination = first_letter + second_letter
        letter_combinations.append(combination)
```

[5]:
```python
# 显示 letter_combinations 中的前 10 个 item
print("letter_combinations:\n", letter_combinations[:10])
print(letter_combinations[26:36])
print(letter_combinations[52:62])
print(letter_combinations[78:88])
print(letter_combinations[104:114])
print(letter_combinations[130:140])
print(letter_combinations[156:166])
print(letter_combinations[182:192])
print(letter_combinations[208:218])
print(letter_combinations[234:244])
```

```
letter_combinations:
 ['aa', 'ab', 'ac', 'ad', 'ae', 'af', 'ag', 'ah', 'ai', 'aj']
['ba', 'bb', 'bc', 'bd', 'be', 'bf', 'bg', 'bh', 'bi', 'bj']
['ca', 'cb', 'cc', 'cd', 'ce', 'cf', 'cg', 'ch', 'ci', 'cj']
['da', 'db', 'dc', 'dd', 'de', 'df', 'dg', 'dh', 'di', 'dj']
['ea', 'eb', 'ec', 'ed', 'ee', 'ef', 'eg', 'eh', 'ei', 'ej']
['fa', 'fb', 'fc', 'fd', 'fe', 'ff', 'fg', 'fh', 'fi', 'fj']
['ga', 'gb', 'gc', 'gd', 'ge', 'gf', 'gg', 'gh', 'gi', 'gj']
['ha', 'hb', 'hc', 'hd', 'he', 'hf', 'hg', 'hh', 'hi', 'hj']
['ia', 'ib', 'ic', 'id', 'ie', 'if', 'ig', 'ih', 'ii', 'ij']
['ja', 'jb', 'jc', 'jd', 'je', 'jf', 'jg', 'jh', 'ji', 'jj']
```

[6]:
```
# 创建一个空列表来存储每对相邻字母出现的次数向量（共有 676 个元素：26*26=676）
pair_vectors = []
# 遍历所有可能的相邻字母组合（从 aa 到 zz）
for combination in letter_combinations:
    # 如果相邻字母在 pair_count 中，就获取它出现的次数，否则默认为 0
    count = pair_count.get(combination, 0)
    # 把次数添加到向量中（作为一个元素）
    pair_vectors.append(count)
```

[7]:
```
len(pair_vectors)
```

[7]: 676

[8]:
```
# 打印 pair_vectors，我们可以看到它是一个一维向量，共有 676 个元素
print("pair_vectors:\n", np.array(pair_vectors).reshape(26, 26)[:10, :10])
```

```
pair_vectors:
 [[  463 14013 19190 10757  6575  2441  9259  2132  7452   411]
 [ 9945  1907   434   479 10451   159    96   237  9578   275]
 [24803    58  2719   129 15772    56    54 22188 12195     4]
 [10191   449   277  2113 24078   404  1184   595 20452   322]
 [16171  3474 13839 32075  8302  4032  4621  1710  5268   438]
 [ 4058    58    73    63  5661  3453    35    54  7519    13]
 [ 8798   256    74   176 13072   145  2309  3266  9080    17]
 [16501   373   154   181 20824   294    95   132 16152    26]
 [22491  5033 37513 15228 15426  6499  7657   501   630   190]
 [ 1842     5    21    18  1344     5     5    12   498     9]]
```

这里就得到了一个 pairs 的 map

[9]:
```
# 创建一个空列表来存储每个单独字母出现的次数向量（共有 26 个向量，每个向量有 676 个元素）
letter_vectors = []
alphabet = "abcdefghijklmnopqrstuvwxyz"
# 遍历所有可能的单独字母（从 a 到 z）
for letter in alphabet:
    # 创建一个空列表来存储当前单独字母出现的次数向量（共有 676 个元素）
    vector = []
    # 遍历所有可能的相邻字母组合（从 aa 到 zz）
```

```
    for combination in letter_combinations:
        # 如果当前单独字母是相邻组合中的第一个或第二个，就获取它出现的次数，否则默认为 0
        if letter in combination:
            count = pair_count.get(combination, 0)
        else:
            count = 0
        # 把次数添加到向量中（作为一个元素）
        vector.append(count)

    # 把当前单独字符出现的次数向量添加到列表中
    letter_vectors.append(vector)
```

[11]:
```
np.corrcoef(letter_vectors[0], letter_vectors[1])
```

[11]:
```
array([[1.        , 0.05158347],
       [0.05158347, 1.        ]])
```

[12]:
```
# 创建一个空列表来存储相关系数矩阵（共有 26*26=676 个元素，每个元素是两个字符之间的相关系数）
correlation_matrix = []
# 遍历所有可能的第一个单独字母（从 a 到 z）
for i in range(26):
    # 获取当前单独字母出现的次数向量
    letter_vector_1 = letter_vectors[i]
    # 遍历所有可能的第二个单独字母（从 a 到 z）
    for j in range(26):
        # 获取当前单独字母出现的次数向量
        letter_vector_2 = letter_vectors[j]
        # 用 numpy 库中的 corrcoef() 函数来计算两个向量之间的相关系数
        correlation = np.corrcoef(letter_vector_1, letter_vector_2)[0][1]
        # 把相关系数添加到矩阵中（作为一个元素）
        correlation_matrix.append(correlation)
```

[54]:
```
# 打印相关系数矩阵 保留两位小数
print("correlation_matrix:\n", np.array(correlation_matrix).reshape(26, 26).
 ↪round(3)[:10,:10])
```

```
correlation_matrix:
 [[ 1.     0.052  0.088  0.007 -0.02  -0.019  0.007  0.018 -0.002 -0.014]
 [ 0.052  1.    -0.023 -0.02  -0.002 -0.023 -0.02  -0.021  0.    -0.017]
 [ 0.088 -0.023  1.    -0.02   0.013 -0.024 -0.02   0.117  0.138 -0.018]
 [ 0.007 -0.02  -0.02   1.     0.171 -0.021 -0.017 -0.018  0.063 -0.015]
 [-0.02  -0.002  0.013  0.171  1.    -0.01   0.006  0.031 -0.023 -0.018]
 [-0.019 -0.023 -0.024 -0.021 -0.01   1.    -0.02  -0.021  0.015 -0.018]
 [ 0.007 -0.02  -0.02  -0.017  0.006 -0.02   1.    -0.013  0.    -0.015]
 [ 0.018 -0.021  0.117 -0.018  0.031 -0.021 -0.013  1.     0.012 -0.016]
 [-0.002  0.     0.138  0.063 -0.023  0.015  0.     0.012  1.    -0.021]
 [-0.014 -0.017 -0.018 -0.015 -0.018 -0.018 -0.015 -0.016 -0.021  1.   ]]
```

[14]:
```
corr_matrix = np.array(correlation_matrix).reshape(26, 26)
```

[15]:
```
def alpha2idx(alpha):
    return ord(alpha) - ord('a')
```

[16]:
```
alpha2idx('z')
```

[16]: 25

1.3 定义了一种计算一个单词的正常性的方法

[37]:
```
corr_matrix.shape
```

[37]: (26, 26)

[55]:
```
# corr_matrix[1,26]
```

[17]:
```
def get_normal_value(test_word):
    normal_value = 0
    # 用相关系数矩阵去衡量 test_word 的怪异程度
    # 衡量方法为，从第一个字母开始，计算每个字母与其后续字母的相关系数，加在
normal_value 上，以此衡量该值
    for i in range(len(test_word)-1):
        normal_value +=␣
 ↪corr_matrix[alpha2idx(test_word[i]),alpha2idx(test_word[i+1])]
    return normal_value
test_word = "ooooo"
```

```
print(test_word + " normal value is " + str(get_normal_value(test_word)))
```

```
ooooo normal value is 3.9999999999999996
```

1.4 对原数据进行处理

[18]:
```
import pandas as pd
# 读入 Problem_C_Data_Wordle.xlsx 文件
df = pd.read_excel("Problem_C_Data_Wordle.xlsx")
```

[19]:
```
# 将第一行作为列名
df.columns = df.iloc[0]
# 删除第一行
df = df.drop(0)
# 删除空列
df = df.dropna(axis=1, how='all')
# 重置索引
df = df.reset_index(drop=True)
# 1.2 查看数据
df.head()
```

[19]:

0	Date	Contest number	Word	Number of reported results	Number in hard mode	1 try	2 tries	3 tries	4 tries	5 tries	6 tries	7 or more tries (X)
0	2022-12-31 00:00:00	560	manly	20380	1899	0	2	17	37	29	12	2
1	2022-12-30 00:00:00	559	molar	21204	1973	0	4	21	38	26	9	1
2	2022-12-29 00:00:00	558	havoc	20001	1919	0	2	16	38	30	12	
3	2022-12-28 00:00:00	557	impel	20160	1937	0	3	21	40	25	9	
4	2022-12-27 00:00:00	556	condo	20879	2012	0	2	17	35	29	14	

```
2                    2
3                    1
4                    3
```

去除异常值

[20]:
```
# 找出字符串长度不等于 5 的行
print("总共有" + str(len(df[df['Word'].str.len() != 5])) + "行字符串长度不等于
5")
df[df['Word'].str.len() != 5]
```

```
总共有 4 行字符串长度不等于 5
```

[20]:
```
0                   Date Contest number    Word Number of  reported results  \
15   2022-12-16 00:00:00            545  rprobe                      22853
35   2022-11-26 00:00:00            525   clen                       26381
246  2022-04-29 00:00:00            314   tash                      106652
353  2022-01-12 00:00:00            207  favor                      137586

0   Number in hard mode 1 try 2 tries 3 tries 4 tries 5 tries 6 tries  \
15                 2160     0       6      24      32      24      11
35                 2424     1      17      36      31      12       3
246                7001     2      19      34      27      13       4
353                3073     1       4      15      26      29      21

0   7 or more tries (X)
15                    3
35                    0
246                   1
353                   4
```

[21]:
```
# 我们在网址上找到了那天的正确答案
df.loc[15, 'Word'] = 'probe'
df.loc[35, 'Word'] = 'clean'
df.loc[246, 'Word'] = 'stash'
df.loc[353, 'Word'] = 'favor'
```

[22]:
```
print("总共有" + str(len(df[df['Word'].str.len() != 5])) + "行字符串长度不等于5")
```

```
总共有 0 行字符串长度不等于 5
```

[50]:
```
# 找出字符串中包含非字母字符的行
print("总共有" + str(len(df[df['Word'].str.contains('[^a-zA-Z]')])) + "行字符串中包含非字母字符")
df[df['Word'].str.contains('[^a-zA-Z]')]
```

```
总共有 1 行字符串中包含非字母字符
```

[50]:
```
0                  Date Contest number   Word Number of  reported results  \
20  2022-12-11 00:00:00            540  naïve                        21947

0  Number in hard mode 1 try 2 tries 3 tries 4 tries 5 tries 6 tries  \
20                2075     1       7      24      32      24      11

0  7 or more tries (X)
20                   1
```

[51]:
```
df.loc[20, 'Word'] = 'naive'
```

1.5 计算正常性并加入到原数据中

[52]:
```
# 用 apply() 函数来对每一行的 Word 列进行计算
df['normal_value'] = df['Word'].apply(get_normal_value)
```

[53]:
```
# 保存为 xlsx 文件
df.to_excel("Problem_C_Data_Wordle_new.xlsx", index=False)
```