

数据科学期末报告

王昊旻 211250107 丁晟元 211250097 赵政杰 211250109

Part1 问题研究

1.1 问题回顾

在政务工作领域，目前，每一项服务背后都依托于不同的数据支撑，但由于不同地区、不同政府部门对于数据的标准不同，也就导致了同一数据有不同的表现形式，因而如何高效精准地将其进行匹配归类是一个亟待解决的问题。

而这一问题本质上是一个文本匹配问题。文本匹配是自然语言处理中的一个核心问题，很多自然语言处理的任务都可以抽象成文本匹配问题，例如信息检索可以归结成查询项和文档的匹配，问答系统可以归结为问题和候选答案的匹配，对话系统可以归结为对话和回复的匹配。针对不同的任务选取合适的匹配模型，提高匹配的准确率和速度成为自然语言处理任务的重要挑战。而这一问题正是让我们在政务领域下思考文本匹配的实现。

1.2 初步认识

文本匹配问题是 NLP 中一个并不陌生的领域，但是从需要解决的领域（政务工作领域）来看，由于专门语料库的缺乏，即该领域相关研究并不丰富，使用 word2vec 和 Bert 等有一定的困难。从文本匹配更细化的角度来看，该问题的匹配对象为字段，并不直接是简单的词匹配和较为复杂的句匹配，可以说介于词匹配和句匹配之间。同时，文本中包含如“姓名”与“名字”等并非纯字面上的匹配，而是涉及到语义层面的匹配，这也一定程度上增加了文本匹配的难度。对此，我们的初步想法是采用 SequenceMatcher 方法和词向量结合的方法。要实现的目标是将 excel 表的 C 列与 D 列相匹配。

1.3 探索过程

1.3.1 小组分工

人数：3 人

成员及分工：

王昊旻	211250107@smail.nju.edu.cn	模型训练，数据元的处理和加载
丁晟元	211250097@smail.nju.edu.cn	模型加载，词向量和余弦相似度的计算
赵政杰	211250109@smail.nju.edu.cn	数据处理，准确率的计算

1.3.2 研究思路

基于数据的特殊性，即语义匹配和字面义匹配结合的特点，我们决定采用 Python 标准库中的 SequenceMatcher 方法和 Word2vec 词向量模型相结合的方法进行研究。SequenceMatcher 方法基本思想是找到不包含“junk”元素的最长连续匹配子序列（LCS），相当于从字面上检验相似度而不考虑语义。而 Word2vec 更多的是从语义角度来计算相似度，这也是我们的重要方向。于是我们决定将两种方法相结合，提高匹配的准确率。对于 Word2vec 所需的语料库问题，由于目前时间有限，我们决定采用腾讯提供的已训练好的模型（200 维）。

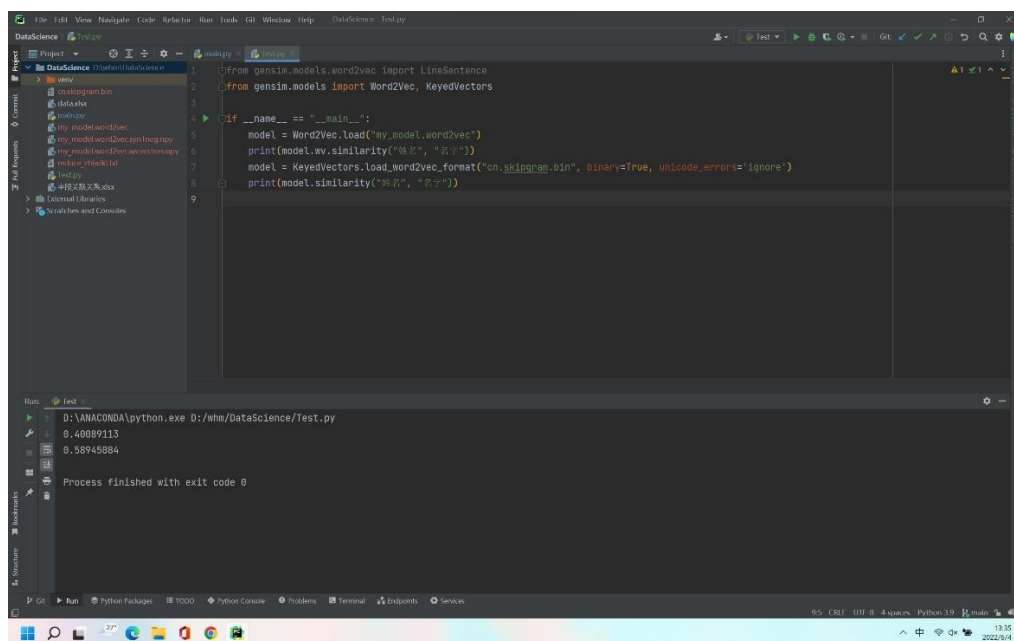
1.3.3 研究方法（重点）

一、模型的训练与选择

起初，我们曾希望用维基百科的中文语料库来训练我们的模型，但是第一次训练出来的模型的准确率比较差，如“姓名”和“名字”的相似度只有 0.40089，究其原因我认为可能是第一次我们没有采用负采样，并且我们的 epoch 只有 5 代。

由于是第一次进行模型的训练，我确实经验缺乏，而且对于参数的理解也不够深入。第二次训练时我尝试将 negative 设为了 20，并将 epoch 改为 10，如果这个模型能成功构建，效果应该是会比第一个好很多的，但是，我的笔记本算力

显然不足以支撑我的训练了。CPU 占用率达到了 100%，20 分钟过去了才完成了第一代处理。这时候我们经过讨论，决定改用别人已经训练好的模型，因为虽然维基中文语料库很大，但是对政府相关术语的针对性不强，而且加载这样一个模型也是非常费时的。我们测试了从 GitHub 上找到的前人用维基中文语料库训练的模型，发现用这么大的语料准确性确实是一般的。如下图



```
1 from gensim.models.word2vec import LineSentence
2 from gensim.models import Word2Vec, KeyedVectors
3
4 if __name__ == '__main__':
5     model = Word2Vec.load("my_model.word2vec")
6     print(model.wv.similarity("姓名", "名字"))
7     model = KeyedVectors.load_word2vec_format('cn.skipgram.bin', binary=True, unicode_errors='ignore')
8     print(model.similarity("姓名", "名字"))
9
```

Run: D:\ANACONDA\python.exe D:/hnm/DataScience/Test.py

0.4089113
0.58945884

Process finished with exit code 0

第一个是我们训练的模型的相似度，第二个是别人用维基语料库训练出来的模型的相似度。可以发现，使用这样大的语料库，不排除有导致很多的词汇的余弦相似度降低的风险。因此，我们决定选一些大小适中的语料库训练的模型来尝试。我们目前使用的模型是腾讯提供的 200 维，20 万字的语料库训练的模型，比我们中期所使用的模型要更好，更准确。因此我们最终选择使用了这个模型。鉴于时间有限，我们未能完成预定计划中爬取相关政务领域的内容来训练出与政务领域高度相关的模型。

另外，对于 txt 文件，由于每次重载都需要花费大量的时间，出于加载速度的考虑，我们将模型的 txt 文件转化为了二进制文件以提高读入速度。

```

"""
# tencent 预训练的词向量文件路径
vec_path = "tencent-ailab-embedding-zh-d100-v0.2.0-s.txt"
# 加载词向量文件
wv_from_text = gensim.models.KeyedVectors.load_word2vec_format(vec_path, binary=False)
# 如果每次都上面的方法加载，速度非常慢，可以将词向量文件保存成bin文件，以后就加载bin文件，速度会变快
wv_from_text.init_sims(replace=True)
wv_from_text.save(vec_path.replace(".txt", ".bin"))
"""

```

二、数据元的加载和处理

1.对标准数据元的处理

我们将助教给我们的 excel 表格的“D”列视为标准数据元。我们定义了 RenewDict 类来实现数据元的动态加载与更新，思路如下：

(1) 我们类比句子相似度比较的方法，将关键字提取步骤简化为对字段进行分词，然后计算其平均词向量。即对每一个数据元进行 jieba 分词，寻找到每一个分词的词向量，再计算平均词向量，将其视为唯一标识该数据元的“词向量”。

(2) 计算完的所有标准数据元的词向量会以字典的形式返回，即每个匹配字段都会有一个唯一的词向量，这就完成了对数据元的去重（字典的键不能重复）。

由于某些词汇在模型的语料库中没有，会生成元素均为 0 的 array 数组，因此采用 np.any()方法完成删除。

代码如下：

```

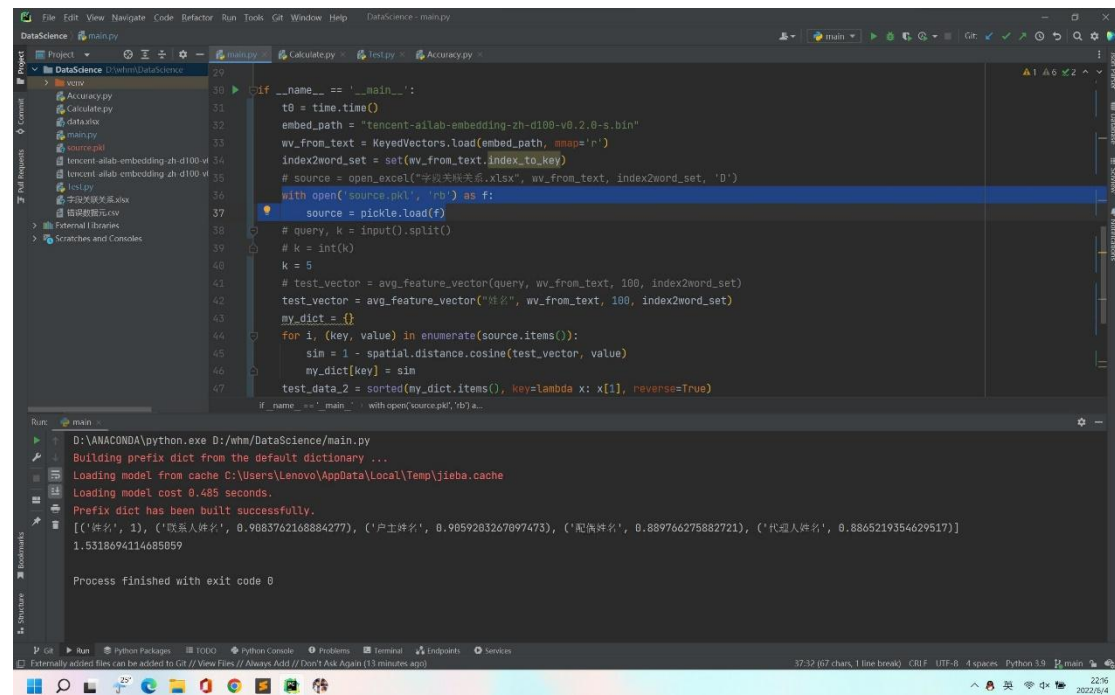
def avg_feature_vector(self, sentence, model, num_features, index2word_set):
    words = jieba.lcut(sentence)
    feature_vec = np.zeros((num_features,), dtype='float32')
    n_words = 0
    for word in words:
        if word in index2word_set:
            n_words += 1
            feature_vec = np.add(feature_vec, model[word])
    if n_words > 0:
        feature_vec = np.divide(feature_vec, n_words)
    return feature_vec

def open_excel(self, path, model, set_, line_name):
    f = pd.read_excel(path, sheet_name=1, usecols=line_name)
    data = {}
    for line in range(len(f)):
        this_str = format(f.loc[line].values[0])
        vector = self.avg_feature_vector(sentence=this_str, model=model, num_features=200, index2word_set=set_)
        if np.any(vector):
            data[this_str] = vector
    return data

```

(3) 动态加载的实现：首先，为了减少每次加载的时间，对于标准数据元词向

量的字典，我们将其生成为 pkl 文件，每次静态加载字典，这样子可以减少 1s 左右的耗时。两者对比结果如下：

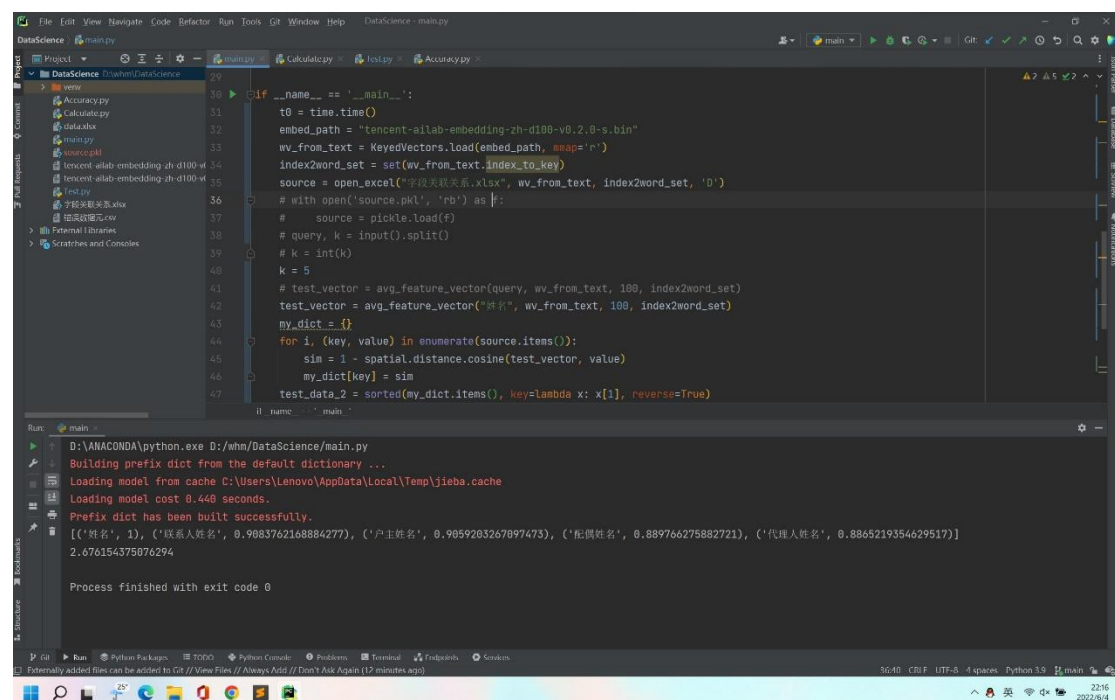


```
29
30 if __name__ == '__main__':
31     t0 = time.time()
32     embed_path = "tencent-ai/label-embedding-zh-d100-v0.2.0-s.bin"
33     ww_from_text = KeyedVectors.load(embed_path, mmap='r')
34     index2word_set = set(ww_from_text.index_to_key)
35     # source = open_excel("字段关联关系.xlsx", ww_from_text, index2word_set, 'D')
36     # with open('source.pkl', 'rb') as f:
37     #     source = pickle.load(f)
38     # query, k = input().split()
39     # k = int(k)
40     k = 5
41     # test_vector = avg_feature_vector(query, ww_from_text, 100, index2word_set)
42     test_vector = avg_feature_vector("姓名", ww_from_text, 100, index2word_set)
43     my_dict = {}
44     for i, (key, value) in enumerate(source.items()):
45         sim = 1 - spatial.distance.cosine(test_vector, value)
46         my_dict[key] = sim
47     test_data_2 = sorted(my_dict.items(), key=lambda x: x[1], reverse=True)
48     if __name__ == '__main__': with open('source.pkl', 'rb') as f:
49         source = pickle.load(f)
```

Running the script in a Jupyter Notebook environment. The output shows the following:

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Lenovo\AppData\Local\Temp\jieba.cache
Loading model cost 0.485 seconds.
Prefix dict has been built successfully.
[('姓名', 1), ('关联人姓名', 0.9083762168884277), ('户主姓名', 0.9059203267097473), ('配偶姓名', 0.889766275882721), ('代理人姓名', 0.8865219354629517)]
1.5318694114685059

Process finished with exit code 0
```



```
29
30 if __name__ == '__main__':
31     t0 = time.time()
32     embed_path = "tencent-ai/label-embedding-zh-d100-v0.2.0-s.bin"
33     ww_from_text = KeyedVectors.load(embed_path, mmap='r')
34     index2word_set = set(ww_from_text.index_to_key)
35     source = open_excel("字段关联关系.xlsx", ww_from_text, index2word_set, 'D')
36     # with open('source.pkl', 'rb') as f:
37     #     source = pickle.load(f)
38     # query, k = input().split()
39     # k = int(k)
40     k = 5
41     # test_vector = avg_feature_vector(query, ww_from_text, 100, index2word_set)
42     test_vector = avg_feature_vector("姓名", ww_from_text, 100, index2word_set)
43     my_dict = {}
44     for i, (key, value) in enumerate(source.items()):
45         sim = 1 - spatial.distance.cosine(test_vector, value)
46         my_dict[key] = sim
47     test_data_2 = sorted(my_dict.items(), key=lambda x: x[1], reverse=True)
48     if __name__ == '__main__':
49         with open('source.pkl', 'rb') as f:
50             source = pickle.load(f)
```

Running the script in a Jupyter Notebook environment. The output shows the following:

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\Lenovo\AppData\Local\Temp\jieba.cache
Loading model cost 0.440 seconds.
Prefix dict has been built successfully.
[('姓名', 1), ('关联人姓名', 0.9083762168884277), ('户主姓名', 0.9059203267097473), ('配偶姓名', 0.889766275882721), ('代理人姓名', 0.8865219354629517)]
2.676154375076294

Process finished with exit code 0
```

同时，我们实现了 renew 方法，这样每次在标准数据元发生变化的时候就可以

通过调用此方法来完成 pkl 文件的更新，从而实现动态加载。

代码如下：

```
def renew(self):
    wv_from_text = KeyedVectors.load(self.embed_path, mmap='r')
    index2word_set = set(wv_from_text.index_to_key)
    source = self.open_excel("字段关联关系.xlsx", wv_from_text, index2word_set, 'D')
    with open('source.pkl', 'wb') as f:
        pickle.dump(source, f)
    print("Renew done!")
```

(4) SequenceMatcher 方法的实现代码

```
def get_equal_rate_1(str1, str2):
    return difflib.SequenceMatcher(None, str1, str2).quick_ratio()

def findTopk_SM(query_str, topk):
    k = topk
    f = pd.read_excel("字段关联关系.xlsx", sheet_name=0, usecols='B')
    size = len(f)
    # query是待匹配的字符串
    query = query_str
    similarity_dict = {}
    for line in range(size):
        b_str = format(f.loc[line].values[0])
        # similarity_list.append(get_equal_rate_1(query, b_str))
        similarity_dict[b_str] = get_equal_rate_1(query, b_str)
    res = sorted(similarity_dict.items(), key=lambda x: x[1], reverse=True)
    print(res[0:k])

def findTop1_SM(query_str):
    f = pd.read_excel("字段关联关系.xlsx", sheet_name=0, usecols='B')
    size = len(f)
    # query是待匹配的字符串
    query = query_str
    similarity_dict = {}
    for line in range(size):
        b_str = format(f.loc[line].values[0])
        similarity_dict[b_str] = get_equal_rate_1(query, b_str)
    res = sorted(similarity_dict.items(), key=lambda x: x[1], reverse=True)
    return res[0][0]
```

2.对待匹配字段的处理

在 main 的主循环里，我们可以完成待匹配字段和 topk 的字段的输入。对于 Word2vec 方法，程序将会遍历我们的字典，计算它们的余弦相似度（调用

spatial.distance.cosine()函数), 返回另一个字典, 在排序后即可完成 topk 的输出, 同时给出相似度; 对于 SequenceMatcher 方法, 直接调用 SM 中的方法, 也会给出 topk 的输出与相似度。

同时, 在循环里可以向标准数据元中增加数据 (输入 add 指令), 调用 RenewDict 的 renew 方法完成 pk1 文件的更新, 这就成功地实现了动态加载。

While 代码如下:

```
while True:
    query = input()
    if query == "quit":
        break
    if query == "add":
        # 输入要增加的数据, 四列的数据元之间以空格隔开
        data = input().split()
        target = [(data[0], data[1], data[2], data[3])]
        df = pd.DataFrame(target, columns=['机构名称', '表中文名', '字段中文(可忽略)', '数据元']) # 列表数据转为数据框
        df1 = pd.DataFrame(pd.read_excel(filename, sheet_name=1)) # 读取原数据文件和表
        book = load_workbook(filename)
        writer = pd.ExcelWriter(filename, engine='openpyxl') # 必须先load_workbook再writer
        writer.book = book
        writer.sheets = dict((ws.title, ws) for ws in book.worksheets)
        df_rows = df1.shape[0] # 获取原数据的行数
        df.to_excel(writer, sheet_name="关联关系", startrow=df_rows + 1, index=False,
                    header=False) # 将数据写入excel中的关联关系表, 从第一个空行开始写
        writer.save() # 保存
        renew.renew()
        continue
    k = int(input())
    test_vector = avg_feature_vector(query, vv_from_text, 200, index2word_set)
    my_dict = {}
    for i, (key, value) in enumerate(source.items()):
        sim = 1 - spatial.distance.cosine(test_vector, value)
        my_dict[key] = sim
    test_data_2 = sorted(my_dict.items(), key=lambda x: x[1], reverse=True)
    print("Using Method 1 : Average feature vector")
    print(test_data_2[0:k])
    print("Using Method 2 : SequenceMatcher")
    # findTopk_SM(query, k)
    findTopk_SM(query, k)
```

经过多次实验, 我们确定每一个数据元的匹配速度在 1 到 2 秒内会完成, 完美达成 5 秒以内的要求。输出结果如下:

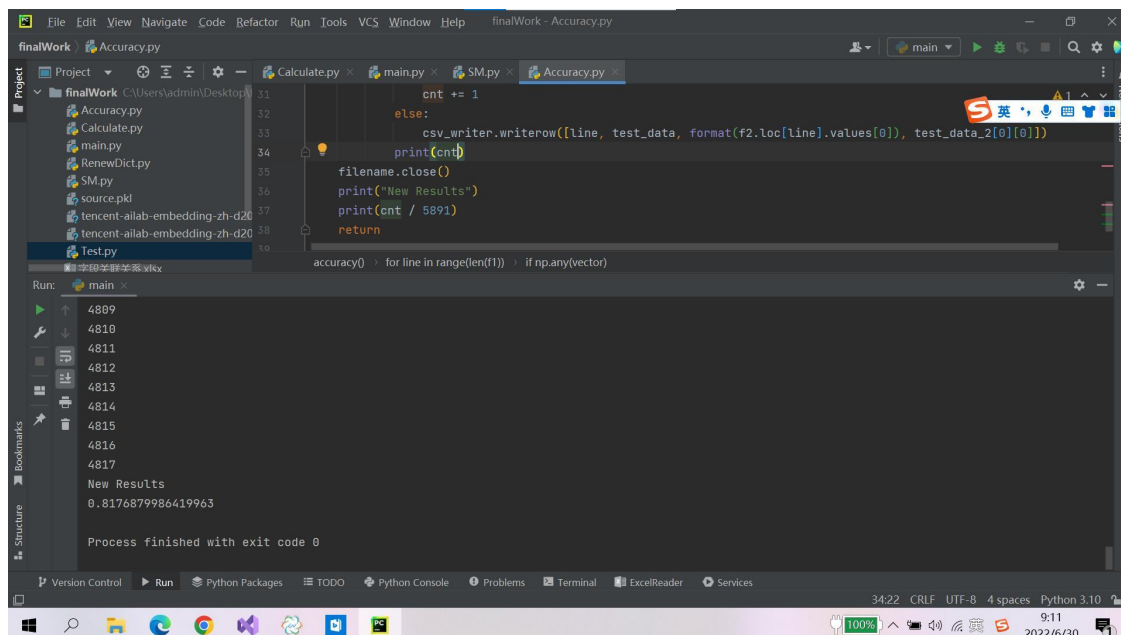
```
E:\Anaconda\python.exe D:/Codes/Data/FinalWork/main.py
采矿人姓名
5
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\HARKTI-1\AppData\Local\Temp\jieba.cache
Loading model cost 0.500 seconds.
Prefix dict has been built successfully.
Using Method 1 : Average feature vector
[('采矿人姓名', 0.8838536739349365), ('开户人姓名', 0.8557424545288086), ('原采矿权人姓名', 0.8551661372184753), ('法定代表人姓名', 0.8421199917793274), ('投诉人姓名', 0.8387596011161804)]
Using Method 2 : SequenceMatcher
[('采矿人姓名', 0.9090909090909091), ('原采矿权人姓名', 0.8333333333333334), ('探矿权人姓名', 0.7272727272727273), ('采矿权名称', 0.7272727272727273), ('病人姓名', 0.6666666666666666)]
```

三、准确率检验

我们将字段关联关系表中的 5891 项待匹配字段一一读入, 并将两个方法返回的第一个 topk 项与相对应的匹配项作比较。如图, 最终得到的结果为 4817

项是直接命中的，即当前算法直接召回且第一个就是正确答案的精度为 0.818。

这比中期只是使用单一方法多出 604 项，精度提高 14%，说明两种方法结合的结果是有效的。



The screenshot shows an IDE window titled 'finalWork - Accuracy.py'. The code in the editor is as follows:

```
11 cnt += 1
12 else:
13     csv_writer.writerow([line, test_data, format(f2.loc[line].values[0]), test_data_2[0][0]])
14     print(cnt)
15     filename.close()
16     print("New Results")
17     print(cnt / 5891)
18     return
19 accuracy() for line in range(len(f1)) if np.any(vector)
```

The console output at the bottom shows the following sequence of values:

```
4809
4810
4811
4812
4813
4814
4815
4816
4817
New Results
0.8176879986419963
Process finished with exit code 0
```

1.3.4 代码开源地址: <https://github.com/marktiwnzhao/DataScience>

Part2 目前的问题

1. 由于时间有限，我们未能利用爬虫等技术大量爬取政府网站的数据，构建自己的语料库，没有训练出我们自己的模型；同时，两种方法的结合也有所欠缺，还需一定的数据处理技术来进行优化。
2. 就算法角度而言，我们目前的主要内容还是遍历字典，其实从余弦相似度计算公式角度出发，也许我们并不需要将字典中每一个 key 的 value 都拿出来算一个相似度。后续我们希望在查找层面提升我们的算法，提高匹配速度。
3. 就三个附加点而言，动态加载的功能我们已经实现了，在算法层面，两种方法的结合与 pkl 文件等算法，已经在准确率和速度方面有了不错的表现。可

惜时间有限，未能实现多对一的匹配。

Part3 对这门课的意见

1. 很感谢颜昌粤助教对我们之前的邮件中问题的解答，及时纠正了我们的研究方向，让我们从文本扩增转向文本匹配，并且中期项目中基本上完成了文本匹配的基本内容了。总体来说，我们还是觉得老师和助教们和我们的信息差有一些大，在中期考核文档之前实际上我们的研究方向一直是不太明确的。
2. 虽然过程磕磕绊绊，但我们也确实领悟到了一些数据分析和深度学习技术的乐趣，后续有机会还是希望可以与助教、老师做更深入的了解。

Part4 想对老师说的话

1. 老师上课讲的很多内容很高深，我们的基础数学知识并没有达到那样的高度，三小时听下来可能都没能很好的理解。
2. 老师对数据科学的思想的讲述让我们受益匪浅，虽然有些思想在本次大作业中未能体现，但还是觉得收获多多。