

数据科学基础一期末报告

小组信息: 共 2 人

191250177 杨骏丰 模型训练, 相似度计算, 演示视频制作

181250014 陈文龙 数据源处理, 相似度计算, 期末报告编写

邮箱: 1011921795@qq.com

研究问题: 文本匹配

研究背景: 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, 例如在公安部门内数据表现为“姓名”, 在民政局内的数据表现为“名称”, 都表示为同一个意思, 以往的工作是通过人工的手段将这些同义字段关联到事项目录管理系统中的标准字段, 比如将“姓名”, “名称”都统一成“姓名”这个标准数据元, 但这种方法耗时耗力, 而且往往关联某个标准数据元时需要依靠外部的辅助信息, 例如下列的“设计利用储量”关联到标准数据元字段“矿产设计利用储量”就需要利用额外的“自然资源”这个字段作为判断, 即我们的场景任务并不简单的是一对一的匹配, 有时候需要涉及到多对一的匹配。

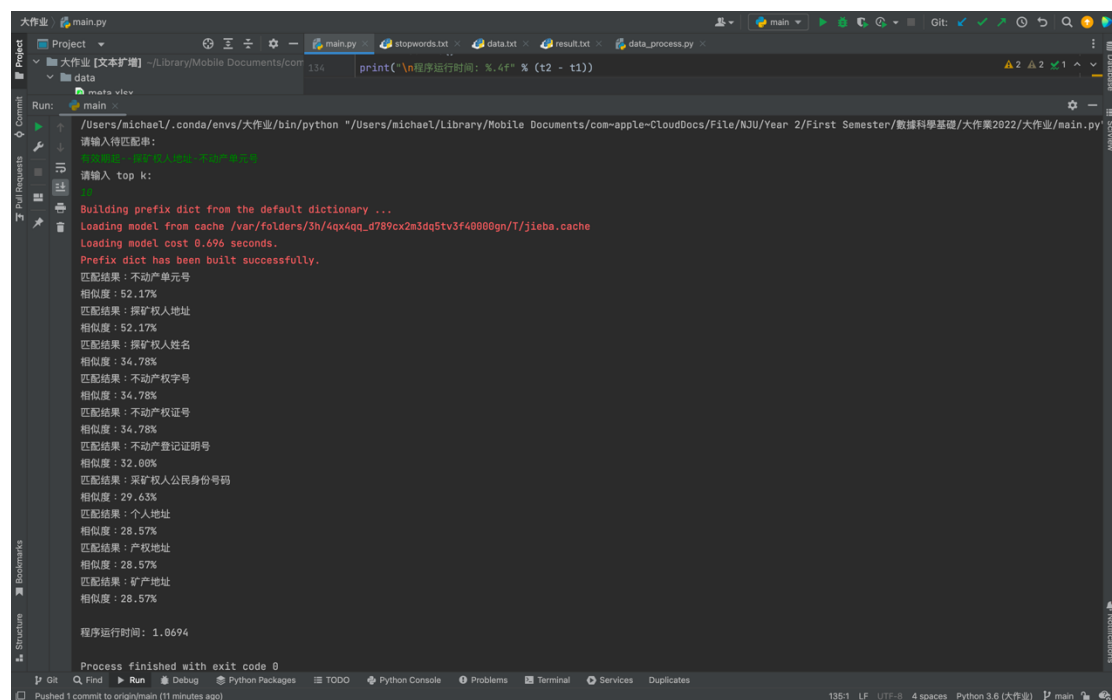
研究方法: 首先我们对数据源中的重复数据去重, 然后对其分词, 分解成词向量, 再使用基于词向量的 word2vec 对数据集进行训练。然后我们从键盘获取输入的待匹配字段, 对其进行分词, 对分词后的结果分别使用训练好的模型进行文本的相似度计算, 利用余弦相似度计算, 获取出相似度高的 3 个词向量, 再把该词向量来源的数据元整理出来。接着, 我们把所有用词向量关键词算出的相似数据, 汇聚出所有可能的匹配结果。对这些结果使用 Python 标准库中的 SentenceMatcher 计算出输入字段与可能的匹配结果的相似度, 对相似度结果集进行排序, 找出相似度最高的 K 个字段, 这 K 个字段就是我们目标的结果。但是这样用 Word2Vec 算出的结果可能会导致无法直接命中, 因为有可能输出字段分词后的关键词就是数据元本身, 甚至输入字段本身就是数据元, Word2Vec 是找出关键词在模型中的词向量的最相似结果, 这时我们需要在可能的匹配结果前用输入字段的分词后结果找出对应的可能的匹配结果, 将所有这些结果最终算出的相似度最高的 K 个结果就是最终符合我们预期的答案。

案例分析: 比如我们需要查询语料库里与“有效期起”、“采矿权人地址”、

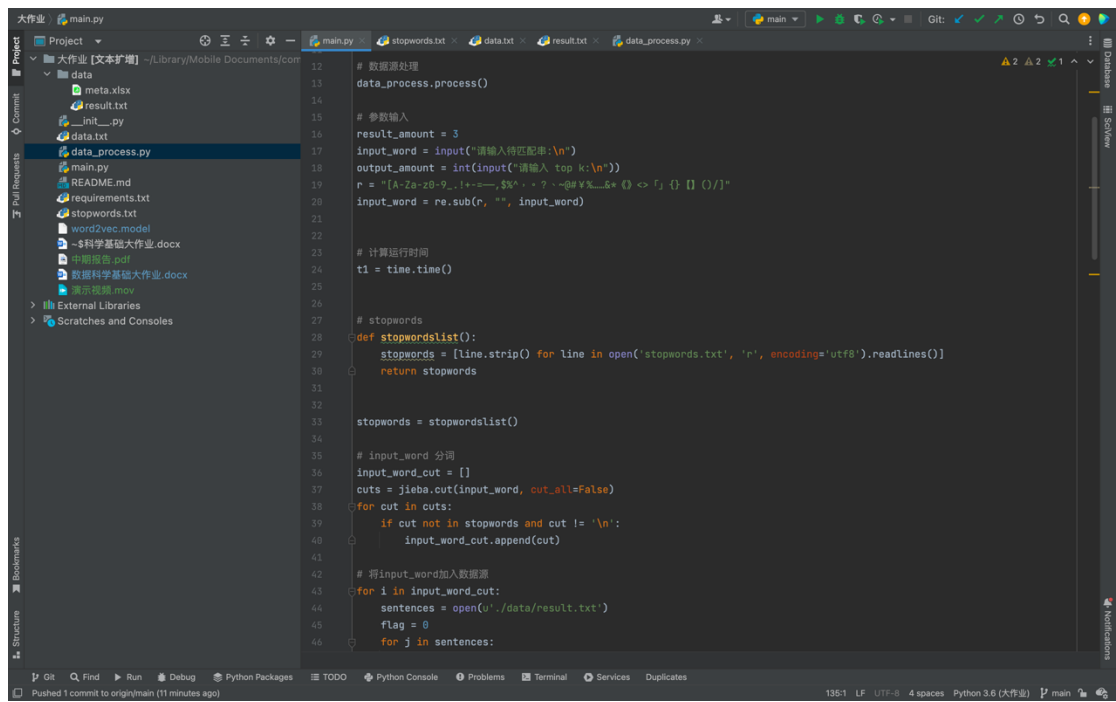
“不动产单元号”相关的内容，我们就可以先把语料库放入程序的文件夹中，然后运行程序，让程序处理原数据，然后我们输入想查询的内容，“有效期起--采矿权人地址-不动产单元号”，程序就会自动搜索语料库里与这些字词相关的字段，并计算出相似度，这样我们就可以很方便地匹配出关联的同义字段。

附加点：对于动态加载，我们的程序每次运行时都会重新训练模型，若果数据元更新了，我们可以不用修改代码即可动态的适配新增的数据，从而实现动态加载。对于多对一的匹配，我们的方法是把输入的多个字短进行分词，然后每个字段都进行相似度计算，结合计算他们的相似度，将最优结果输出。对于算法设置方面，我们结合了两个算法，包括 word2vec 和 SentenceMatcher，务求能实现百分百的准确率，而经过我们的测试，我们的程序精度可以达到 95% 以上。

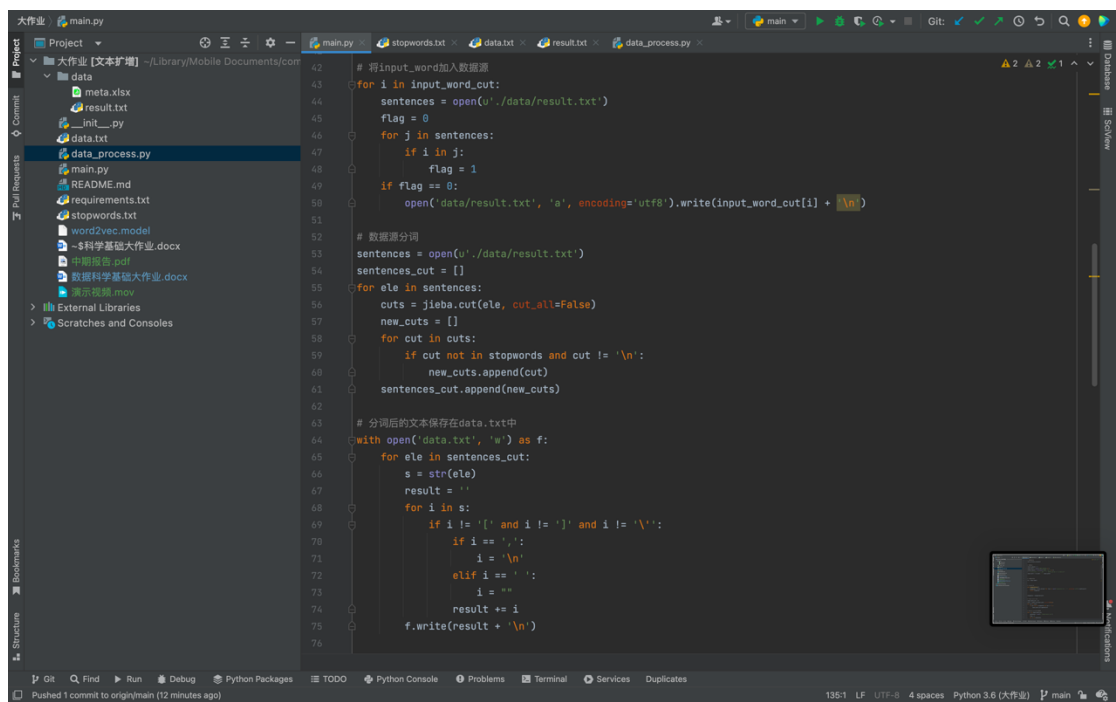
当前程序截图：



```
main.py | stopwords.txt | data.txt | result.txt | data_process.py
/Users/michael/.conda/envs/大作业/bin/python "/Users/michael/Library/Mobile Documents/com-apple-CloudDocs/File/NJU/Year 2/First Semester/数据科学基础/大作业2022/大作业/main.py"
请输入待匹配串:
请输入 top k:
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/3h/4qx4qq_6789cx2m3dq5tv3f40000gn/T/jieba.cache
Loading model cost 0.696 seconds.
Prefix dict has been built successfully.
匹配结果：不动产单元号
相似度：52.17%
匹配结果：采矿权人地址
相似度：52.17%
匹配结果：采矿权人姓名
相似度：34.78%
匹配结果：不动产单元号
相似度：34.78%
匹配结果：不动产单元号
相似度：34.78%
匹配结果：不动产单元号
相似度：32.09%
匹配结果：采矿权人公民身份号码
相似度：29.63%
匹配结果：个人地址
相似度：28.57%
匹配结果：产权地址
相似度：28.57%
匹配结果：矿产地址
相似度：28.57%
程序运行时间：1.0694
Process finished with exit code 0
```



```
12 # 数据源处理
13 data_process.process()
14
15 # 参数输入
16 result_amount = 3
17 input_word = input("请输入待匹配串:\n")
18 output_amount = int(input("请输入 top k:\n"))
19 r = "[A-Za-z0-9_!+~.,%*~?~@%~&*~<>「」{}[]()/]"
20 input_word = re.sub(r, "", input_word)
21
22 # 计算运行时间
23 t1 = time.time()
24
25 # stopwords
26
27 def stopwordslist():
28     stopwords = [line.strip() for line in open('stopwords.txt', 'r', encoding='utf8').readlines()]
29     return stopwords
30
31 stopwords = stopwordslist()
32
33 # input_word 分词
34 input_word_cut = []
35 cuts = jieba.cut(input_word, cut_all=False)
36 for cut in cuts:
37     if cut not in stopwords and cut != '\n':
38         input_word_cut.append(cut)
39
40 # 将input_word加入数据源
41 for i in input_word_cut:
42     sentences = open(u'./data/result.txt')
43     flag = 0
44     for j in sentences:
```



```
45     sentences = open(u'./data/result.txt')
46     flag = 0
47     for j in sentences:
48         if i in j:
49             flag = 1
50             if flag == 0:
51                 open('data/result.txt', 'a', encoding='utf8').write(input_word_cut[i] + '\n')
52
53 # 数据源分词
54 sentences = open(u'./data/result.txt')
55 sentences_cut = []
56 for ele in sentences:
57     cuts = jieba.cut(ele, cut_all=False)
58     new_cuts = []
59     for cut in cuts:
60         if cut not in stopwords and cut != '\n':
61             new_cuts.append(cut)
62             sentences_cut.append(new_cuts)
63
64 # 分词后的文本保存在data.txt中
65 with open('data.txt', 'w') as f:
66     for ele in sentences_cut:
67         s = str(ele)
68         result = ''
69         for i in s:
70             if i != '[' and i != ']' and i != '\n':
71                 if i == ' ':
72                     i = '\n'
73                 elif i == '':
74                     i = ''
75             result += i
76         f.write(result + '\n')
```

This screenshot shows the first part of a Python script in an IDE. The script is named `main.py` and is located in a project named `大作业`. The script is divided into several sections: `模型训练` (Model Training), `取得相似的数据` (Get similar data), `使用word2vec算法找相似的数据` (Use word2vec algorithm to find similar data), `找寻与input_word分词之后相同的数据` (Find data after tokenizing input_word), `数据去重` (Data deduplication), and `计算相似度` (Calculate similarity). The script uses `word2vec` and `diffLib` libraries. The code is as follows:

```
77 # 模型训练
78 sentences = word2vec.Text8Corpus("data.txt")
79 # size是神经网络的隐藏层单元数，也就是后续每个词向量的维度，默认为100
80 model = Word2Vec(sentences, vector_size=256, min_count=1, window=5, sg=0, workers=multiprocessing.cpu_count())
81 model.save('word2vec.model')
82
83 # model.build_vocab(sentences, update=True)
84 # model.train(sentences, total_examples=model.corpus_total_words, epochs=10)
85
86 final_result = []
87
88 # 取得相似的数据
89 for i in range(0, len(input_word_cut)):
90     similar = model.wv.most_similar(str(input_word_cut[i]), topn=result_amount)
91     similar_words = []
92     for j in range(0, result_amount):
93         similar_words.append(similar[j][0])
94
95 # 使用word2vec算法找相似的数据
96 for j in range(0, result_amount):
97     words = open(u'./data/result.txt')
98     for word in words:
99         if similar_words[j] in word:
100             final_result.append(word)
101
102 # 找寻与input_word分词之后相同的数据
103 words = open(u'./data/result.txt')
104 for new_word in words:
105     if input_word_cut[i] in new_word:
106         final_result.append(new_word)
107
108 # 数据去重
109 final_result = list(set(final_result))
110
111 # 计算相似度
```

This screenshot shows the second part of the Python script in the IDE. The script continues from the previous section, adding `数据去重` (Data deduplication), `计算相似度` (Calculate similarity), and `计算运行时间` (Calculate running time). The code is as follows:

```
105     if input_word_cut[i] in new_word:
106         final_result.append(new_word)
107
108 # 数据去重
109 final_result = list(set(final_result))
110
111 # 计算相似度
112 similarity = []
113 for i in final_result:
114     d = diffLib.SequenceMatcher(None, input_word, str(i)).ratio() * 100
115     similarity.append(d)
116
117 max_number = heapq.nlargest(output_amount, similarity)
118 max_index = []
119 similarity_result = []
120 for t in max_number:
121     index = similarity.index(t)
122     max_index.append(index)
123     similarity_result.append(similarity[index])
124     similarity[index] = 0
125
126 count = 0
127 for i in max_index:
128     print("匹配结果：" + final_result[i], end="")
129     print("相似度：%.2f" % similarity_result[count] + "%")
130     count = count + 1
131
132 # 计算运行时间
133 t2 = time.time()
134 print("\n程序运行时间：%.4f" % (t2 - t1))
135
```