数据科学中期报告

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归类是一个亟待解决的问题。

而这一问题无疑可以抽象成文本匹配来进行解决。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，提高匹配的准确率成为自然语言处理任务的重要挑战。

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，对于使用 word2vec 和 Bert 等方法造成了一定的困难。从文本匹配更细化的角度来看，该问题并不属于词匹配，句匹配，短文本匹配和长文本匹配中的一种，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。

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

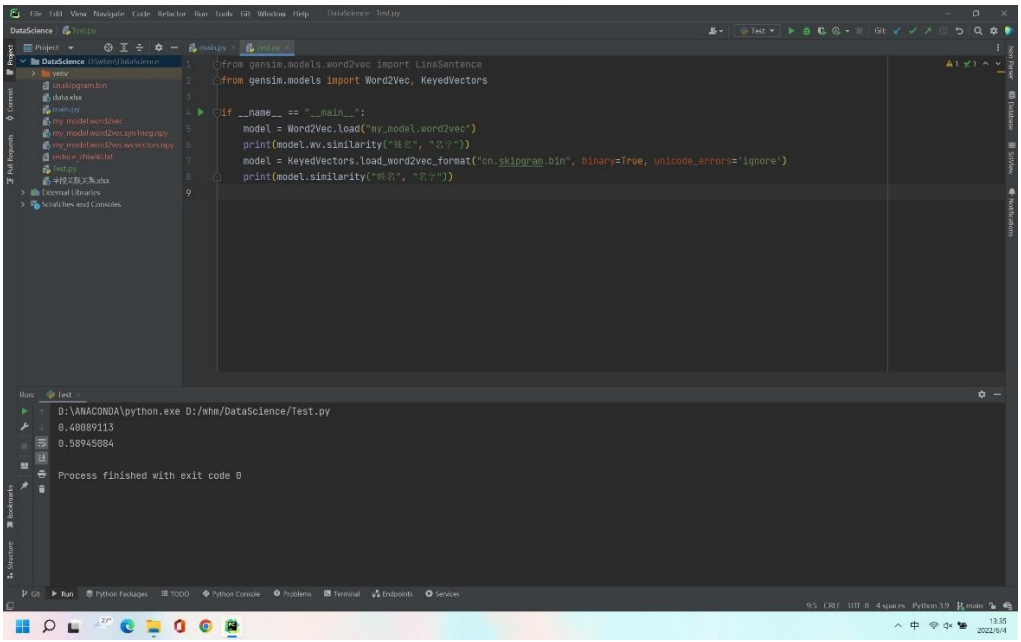
基于数据的特殊性，即语义匹配和字面义匹配结合的特点，我们决定采用Python标准库中的SequenceMatcher方法和Word2vec词向量模型相结合的方法进行研究。由于SequenceMatch方法相对简易，考虑后期再结合使用，目前专攻于word2vec方法的实现。对于语料库问题，由于目前时间有限和笔记本算力的缺乏，我们目前暂时先采用腾讯提供的已训练好的模型，后期考虑采用爬虫等方式，搜集相关性更强的语料对模型进行训练以达到更好的效果，同时，决定采用word2vec的CBOW训练模式进行训练。

1.3.3 研究方法

一、模型的训练与选择

如上一部分所说，目前我们只集中于Word2vec词向量模型实现我们的文本匹配。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



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云上提供的 100 维，20 万字的语料库训练的模型，其“姓名”和“名字”的相似度为 0.7451134，已经算是比较理想了。因此在中期作品中我们暂时选择使用了这个模型。在后期中我们计划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，以提高我们的匹配精度。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。

```
"""
    # 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)
    # 如果每次都用上面的方法加载，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、数据元的处理和加载

我们选择了将助教给我们的 excel 表格的“D”列——标准数据元导入生成一个列表存储我们的词向量，在此过程中我们使用了 jieba 分词（后续会考虑以更合理更准确的方式进行分词上的优化）。同时，为了便于后续进行字段的相似度比较，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。

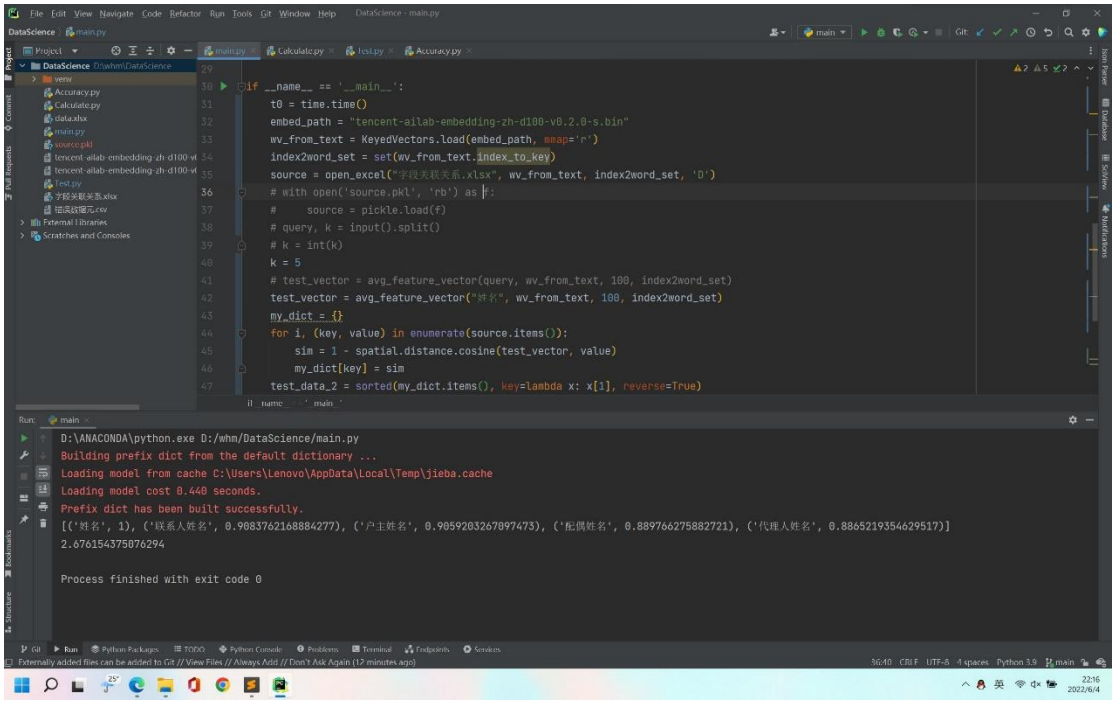
```
def avg_feature_vector(sentence, model, num_features, index2word_set):
    words = jieba.lcut(sentence)
    feature_vec = np.zeros((num_features,), dtype='float32')
    n_words = 0
    for word in words:
        if word in index2word_set:
            n_words += 1
            feature_vec = np.add(feature_vec,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

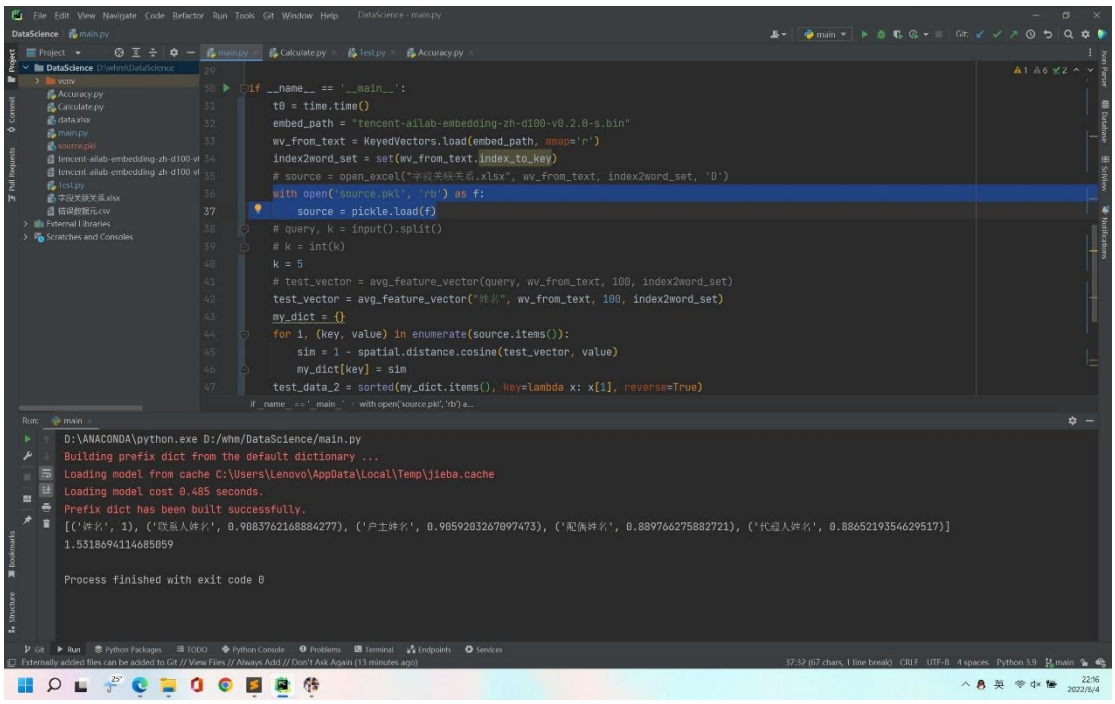
```
def open_excel(path, model, set_, line_name):
    f = pd.read_excel(path, sheet_name=1, usecols=line_name)
    data = {}
    for line in range(len(f)):
        this_str = format(f.loc[line].values[0])
        vector = avg_feature_vector(sentence=this_str, model=model, num_features=100, index2word_set=set_)
        if np.any(vector):
            data[this_str] = vector
    return data
```

计算完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，以方便后续的余弦相似度的计算和查找。余弦相似度直接调用 spatial.distance.cosine()函数进行计算，如下所示：

```
embed_path = "tencent-ailab-embedding-zh-d100-v0.2.0-s.bin"
wv_from_text = KeyedVectors.load(embed_path, mmap='r')
index2word_set = set(wv_from_text.index_to_key)
print(avg_feature_vector('侨居国', model=wv_from_text, num_features=100, index2word_set=index2word_set))
s1_afv = avg_feature_vector('侨居国', model=wv_from_text, num_features=100, index2word_set=index2word_set)
s2_afv = avg_feature_vector('收养人姓名', model=wv_from_text, num_features=100, index2word_set=index2word_set)
sim = 1 - spatial.distance.cosine(s1_afv, s2_afv)
print(sim)
```

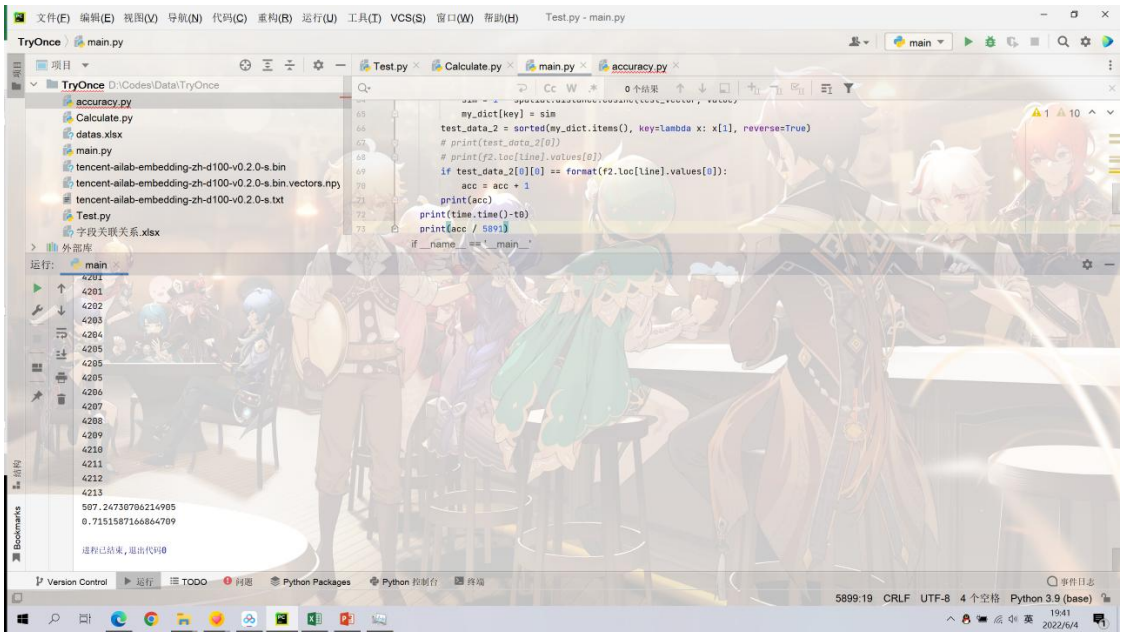
由于某些词汇在模型的语料库中没有，会生成元素均为 0 的 array 数组，因此采用 np.any()方法完成删除，总共删除了 8 个数据元（后期也会考虑这些因为不在词库里而被剔除的数据元的处理方法）。之后我们可以完成待匹配字段和 topk 的字段的输入，并遍历我们的字典，计算它们的余弦相似度，返回另一个，在排序后即可完成输出。此外，为了减少每次加载的时间，对于标准数据元词向量的字典，我将其生成为 pkl 文件，每次静态的加载字典，这样子可以减少 1s 左右的耗时。两者对比结果如下：





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，并将返回的第一个 topk 项与相对应的关系作比较。如图，最终得到的结果为 4213 项是直接命中的，运行总耗时 507 秒，即当前算法直接召回且第一个就是正确答案的精度为 0.715.在后期，我们将对训练得到的词向量库与算法做进一步的改进与提高，预计最后直接召回的精度将会更高。同时，我们将会计算包含非第一项的精度。



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

Part2 目前的问题

1. 由于本学期疫情等错综复杂的原因，和老师、助教们的交流存在信息差，以至于我们以为中期报告并不需要实现太多的内容，所以我们确实存在着对 word2vec 的理解认知不充分的情况。我们的语料库本身也不是特别有针对性，这也导致了我们的中期作品的准确率并不是特别的高，虽然可能人工输入的结果匹配效果看上去很好，但总体的效果经过测试还是不太满意的。后期我们希望可以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，并且在分词环节也不单单利用 jieba，可能需要引入正则表达式和 SequenceMatcher 方法，提高分词的精度，以此训练我们的模型。
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，算法设置是我们下一步需要着重关注的方向。此外，如果时间充沛，我们还将实现多对一的匹配。

Part3 对这门课的意见

1. 希望对作业的讲解能更加细致一点，最好能了解到同学的痛处和难处究竟在哪（或许可以考虑不定时发放调查问卷的方式？）
2. 很感谢颜昌粤助教对我们之前的邮件中问题的解答，虽然可能后期我们也会

涉及到部分文本扩增的内容，但是中期项目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。

3. 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更多的了解的。

Part4 想对老师说的话

1. 后续多期录屏时的声音不太听的清楚，总是一段呜噜呜噜的

2. 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。