**基于深度学习的自然语言处理**

**第1次实验报告**

**一、实验名称**

使用word2vec和Fasttext训练中文词向量

**二、实验日期**

2020.09.14 – 2020.09.27

**三、实验目的**

（1）熟悉这两种训练词向量方法的内部原理

（2）使用这两种方法训练词向量，并在测评任务上测试词向量的精度

**四、实验数据**

维基百科上的中文数据

**五、实验步骤及结果**

**5.1 word2vec**

（1）使用维基百科下载中文语料库

下载位置：<https://dumps.wikimedia.org/zhwiki/20200820/>

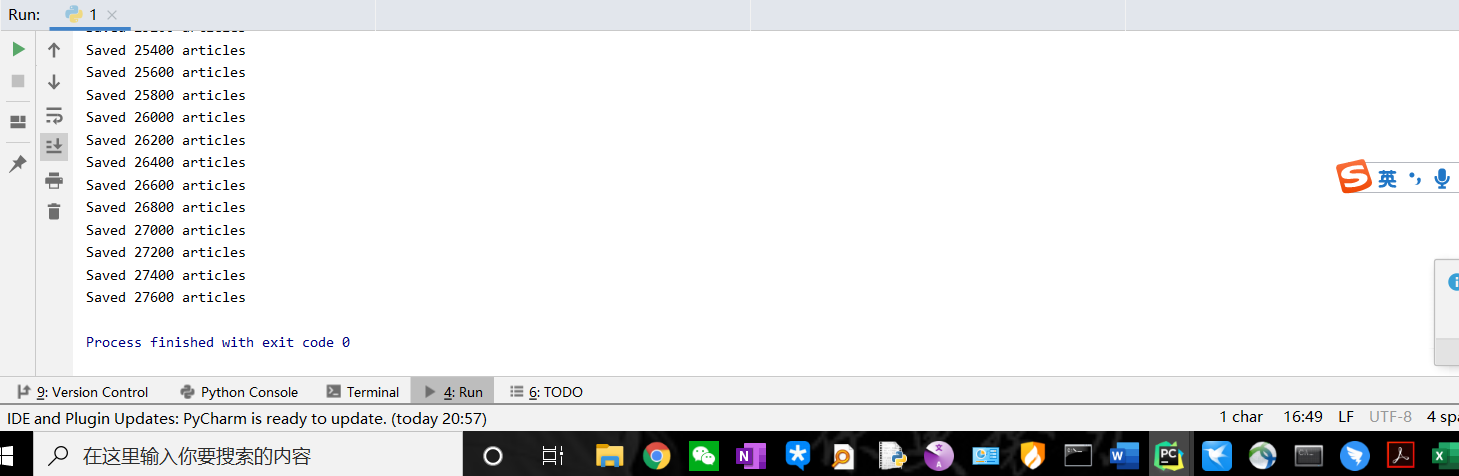


zhwiki-20200820-pages-articles-multistream1.xml-p1p162886.bz2

（2）用python wikipedia extractor抽取维基百科的内容（运行1.py）

（写出运行流程、关键代码理解和运行结果截图）

使用gensim包中维基百科的WikiCorpus方法从下载的xml文件中读预料，如wiki.get\_texts()获取到所有文章的集合，text则是每篇文章里的所有句子构成的list，如['歐幾里得', '西元前三世紀的古希臘數學家', '現在被認為是幾何之父', '此畫為拉斐爾的作品'...]其中还有一些英语，temp\_sentence提取每句句子，然后利用Converter先将繁体中文转为简体中文，再对每句句子使用结巴分词并将分词结果导入到l，把所有这些分词结果导入到txt文件中，每个单次以空格分隔每篇文章以回车分割，文件内容大致如下【欧几里得 西元前 三 世纪 的 古希腊 数学家。。。】。代码实际为了较少地访问文件系统，使用到了列表作为临时存储，以一篇文章为单位导入到txt文件中。



（3）安装包、调试代码跑通整个模型使用Gensim进行词向量训练（运行2.py）

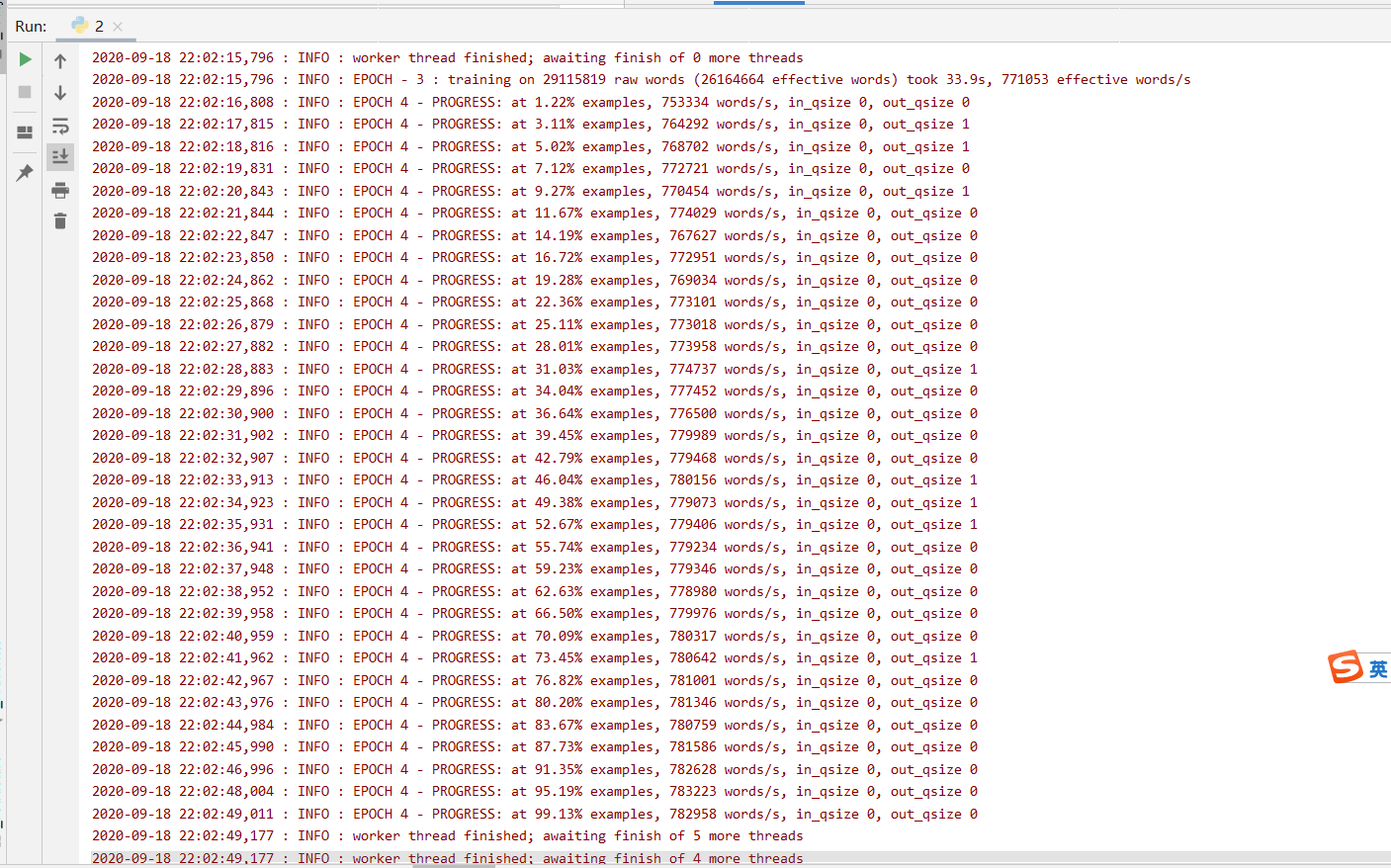
pip install gensim

（写出运行流程、关键代码理解和运行结果截图）（注gensim不支持gpu）

由于word2vec包需要已经分词好了的列表或者文件，所以在上部对文件进行解压并分词。

对于文件形式的分词结果word2vec需要接受一个迭代器，而对于单个文件预料，如果一行对应一个句子并已经分词好了，则可以直接使用LineSentence把txt文件转为所需要的格式（加载语料）。之后设置参数，并训练语料，模型的结果及每个词的词向量写入到文件中。参数解释如下

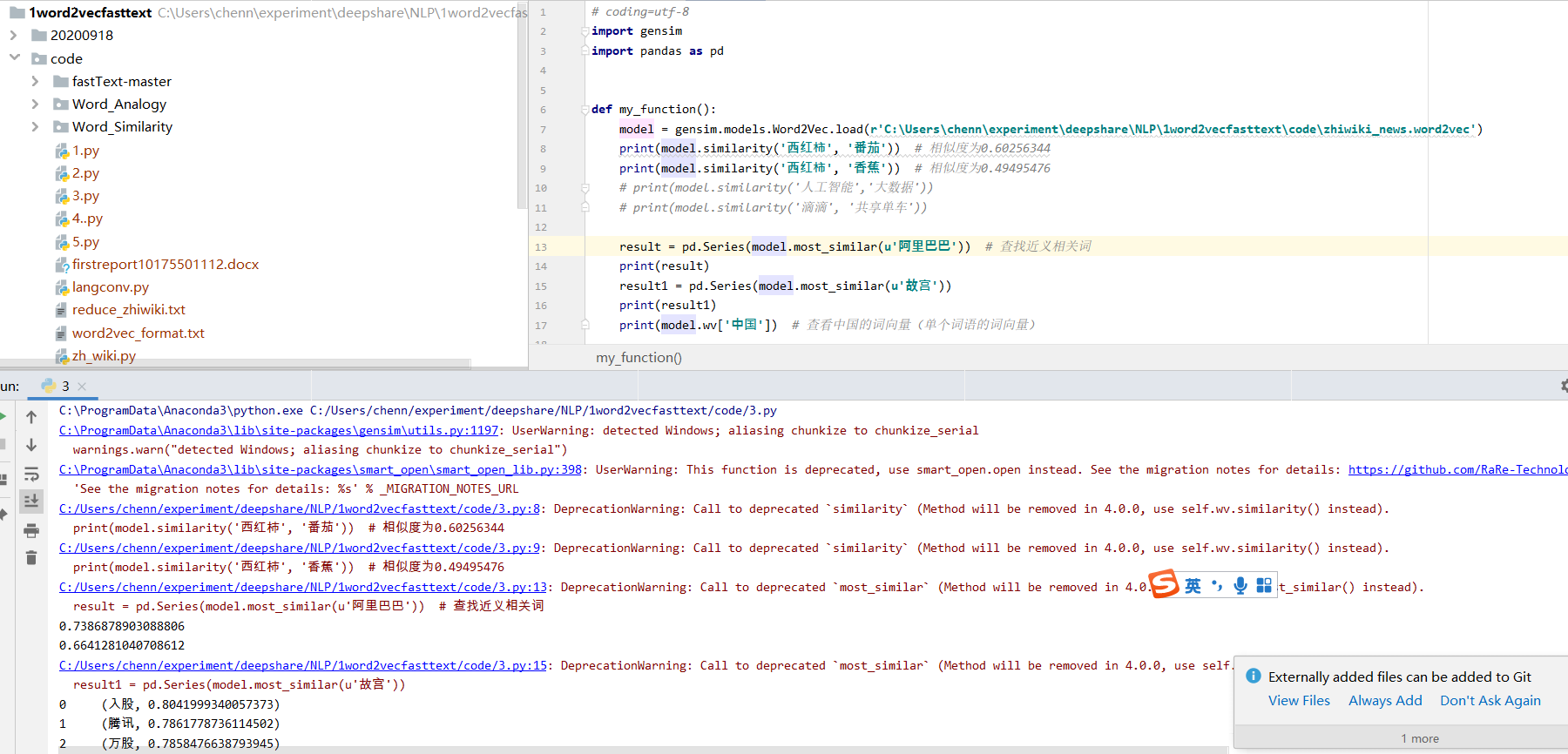
1. Word2Vec(sentences=None,  #sentences可以是分词列表，也可以是大语料
2. size=100,#特征向量的维度
3. alpha=0.025,#学习率
4. window=5,#一个句子内，当前词和预测词之间的最大距离
5. min\_count=5,#最低词频
6. max\_vocab\_size=None,#
7. sample=0.001, #随机下采样的阈值
8. seed=1,#随机数种子
9. workers=3,#进程数
10. min\_alpha=0.0001,#学习率下降的最小值
11. sg=0, #训练算法的选择，sg=1，采用skip-gram，sg=0，采用CBOW
12. hs=0,# hs=1,采用hierarchica·softmax，hs=10,采用negative sampling
13. negative=5,#这个值大于0，使用negative sampling去掉'noise words'的个数（通常设置5-20）；为0，不使用negative sampling
14. cbow\_mean=1,#为0，使用词向量的和，为1，使用均值；只适用于cbow的情况
15. iter = 5,#迭代次数
16. null\_word = 0,
17. trim\_rule = None, #裁剪词汇规则，使用None（会使用最小min\_count）
18. sorted\_vocab = 1,#对词汇降序排序
19. batch\_words = 10000,#训练时，每一批次的单词数量
20. compute\_loss = False,
21. callbacks = ())



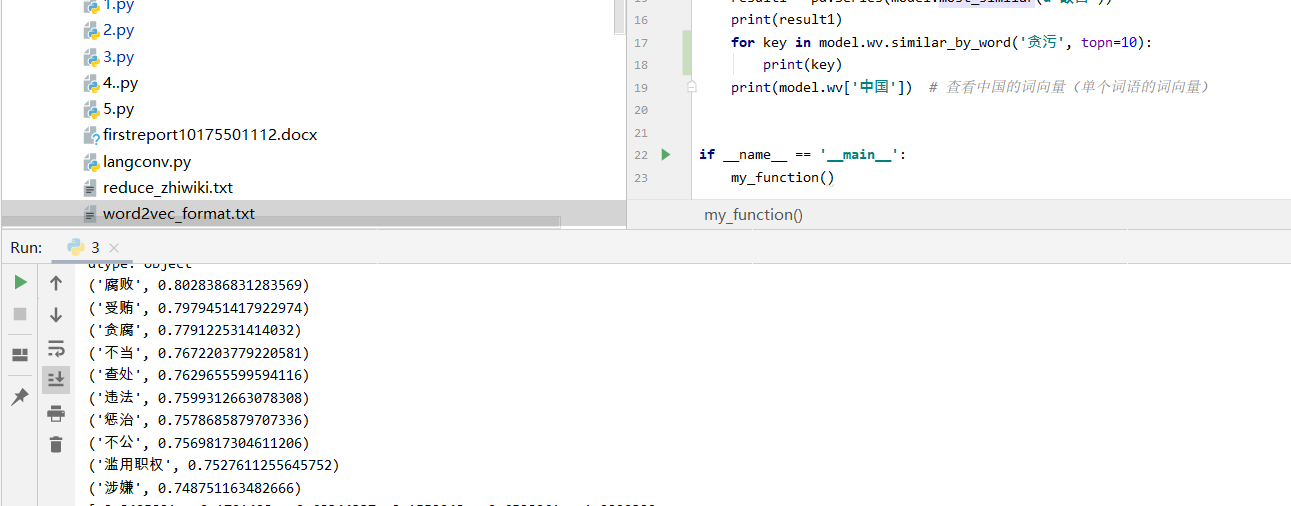


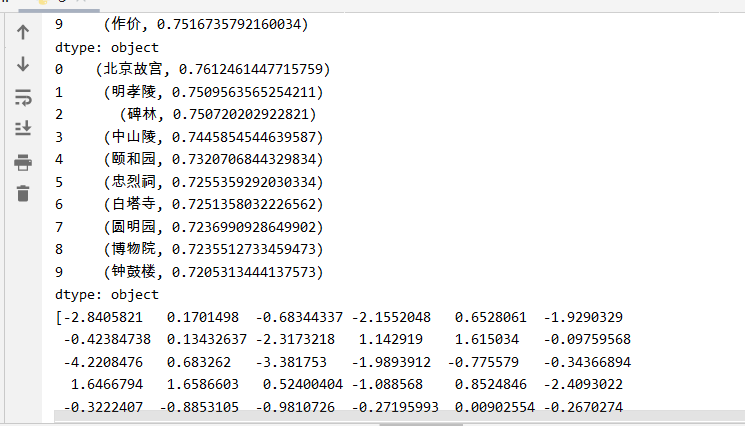
（4）测试同义词，找几个单词，看下效果（运行3.py）

（写出运行流程、关键代码理解和运行结果截图）



可以看出在该参数对应模型下西红柿番茄的相似度为0.73869，西红柿香蕉的相似度为0.66413，与阿里巴巴最相近的词是腾讯，与故宫最相近的词是北京故宫，与贪污最相似的词为腐败（使用similar\_by\_word）并且查看了中国这个单词的词向量。





（5）使用训练好的词向量测试词相似度任务的精度

在Word \_Similarity文件夹下运行 ：

python -u word\_similarity.py --vector word2vec\_format.txt --similarity wordsim-240.txt

（--vector 向量文件位置）

python -u word\_similarity.py --vector word2vec\_format.txt --similarity wordsim-297.txt

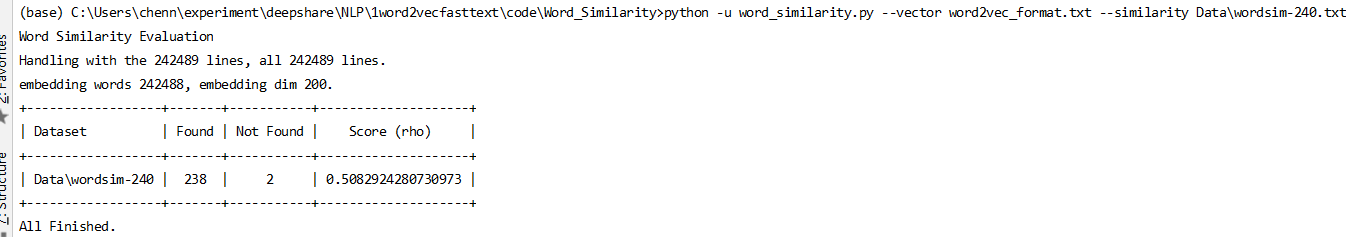
（--vector 向量文件位置）

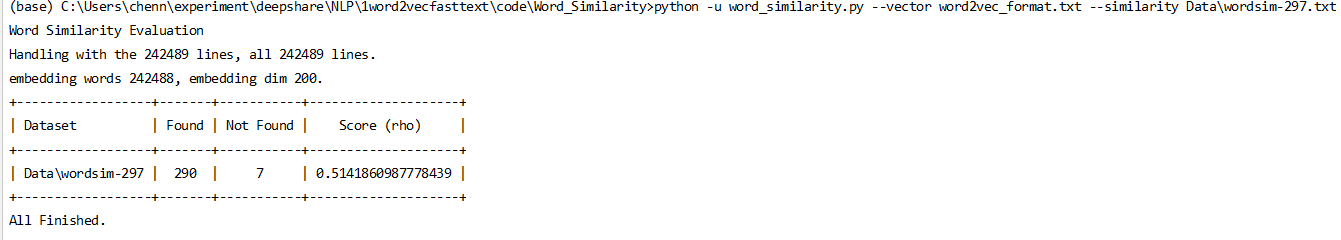
（写出运行流程、关键代码理解和运行结果截图）

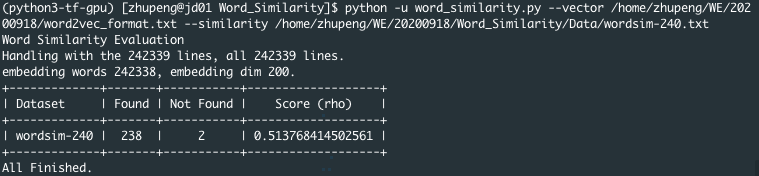
这是以相似度来评价word2vec训练情况的好坏

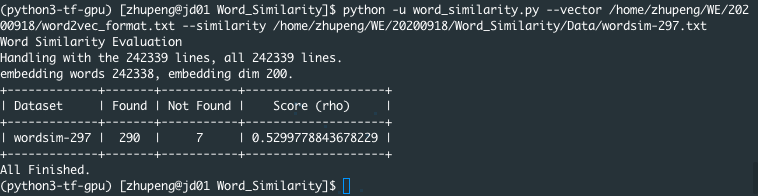
240与297的两个文件为需要测试的相似度文件（240与297对应测试集个数），若模型预测的相似度超过对应的指标则为正确，反之则为错误。运行的时候其先进行embedding（导进来文件），再测试需要测试文件的相似度，在以下参数的情况下结果如下，

1. model = Word2Vec(LineSentence(wiki\_news), sg=0,size=200, window=5, min\_count=5, workers=6)









（6）使用训练好的词向量测试词类比任务的精度

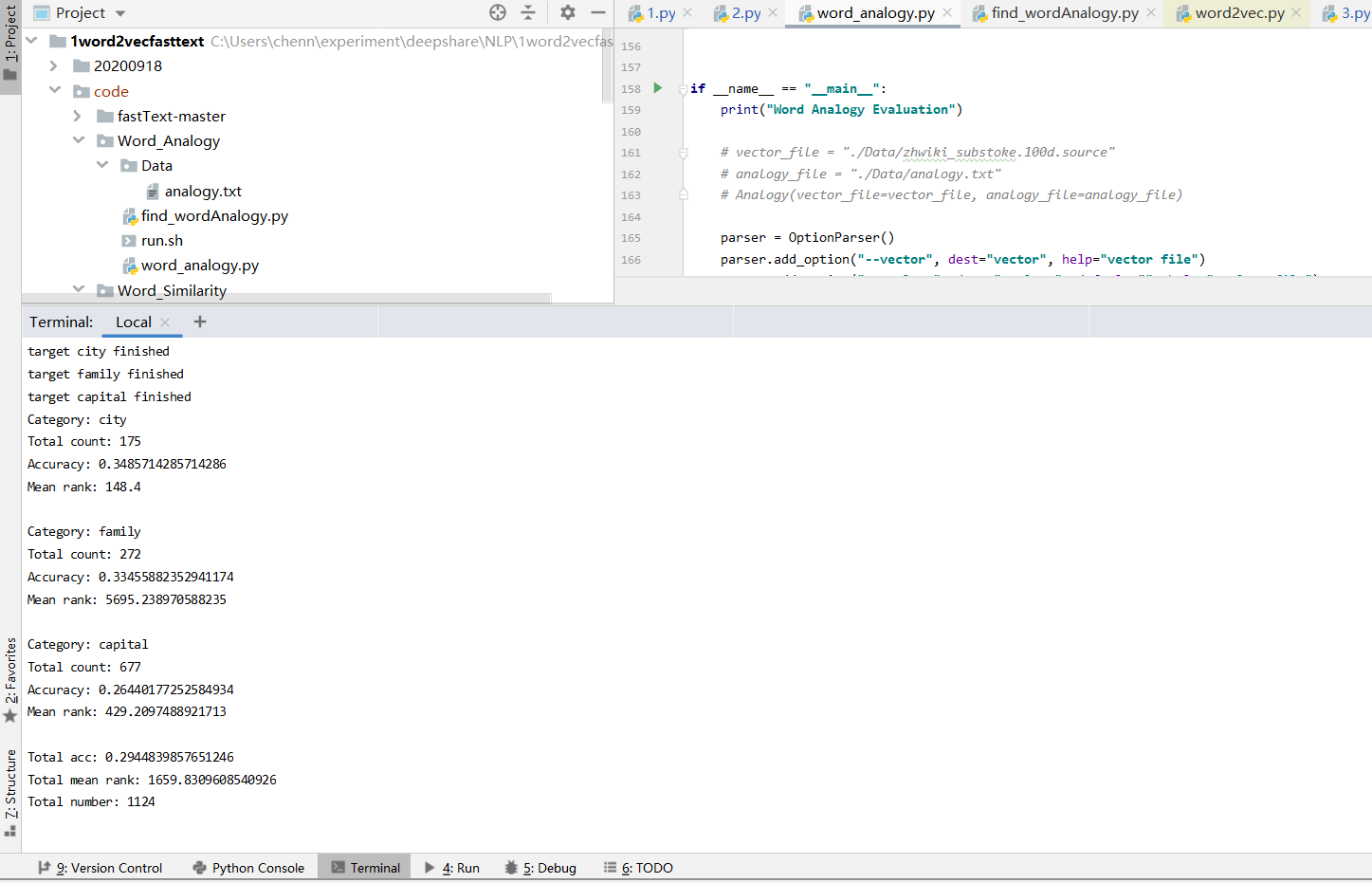
在Word\_Analogy文件夹下运行 ：

python -u word\_analogy.py --vector word2vec\_format.txt -–analogy analogy.txt

（--vector 向量文件位置）

（写出运行流程、关键代码理解和运行结果截图）

主要代码的逻辑依然是先embedding，再对文件中的每个sample进行测试，正确则+1不正确则不加，结果如下，运行约3个小时。

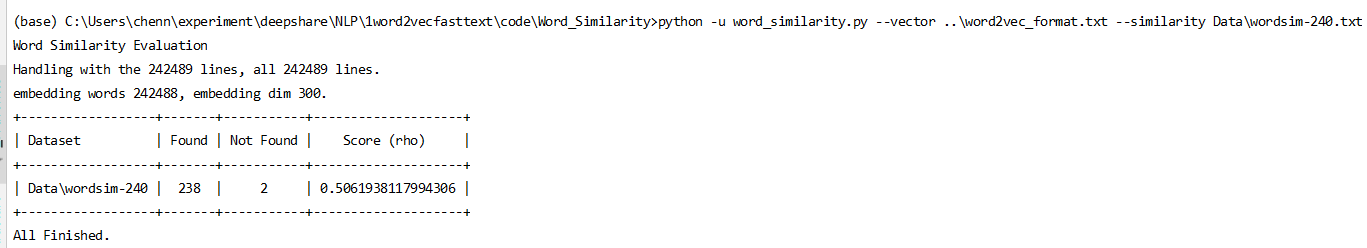


（7）调试模型中的参数重新训练词向量，提高两个任务的精度

对于

1. model = Word2Vec(LineSentence(wiki\_news), sg=0,size=300, window=5, min\_count=5, workers=12)

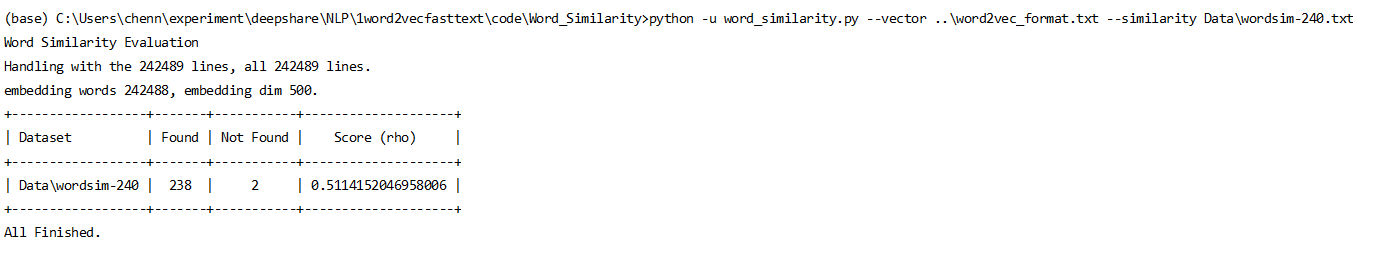
的情况，即size变大，workers=12的情况，240准确率为，运行总时长约为6分钟。



准确率不升反降，workers只是在计算上提升了时间而已，另外workers最高也就multiprocessing.cpu\_count()，8核cpu如果设置大于8也依然是=8的效果。

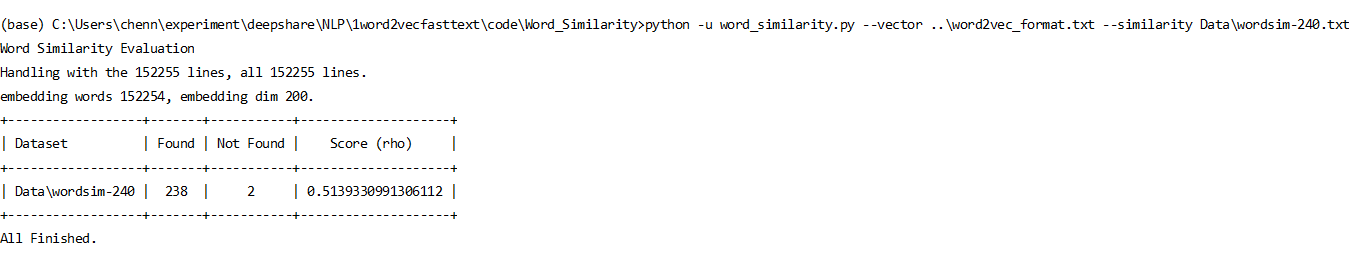
1. model = Word2Vec(LineSentence(wiki\_news), sg=0,size=500, window=5, min\_count=5, workers=20)

训练了6分钟，训练分数提升到了0.51.



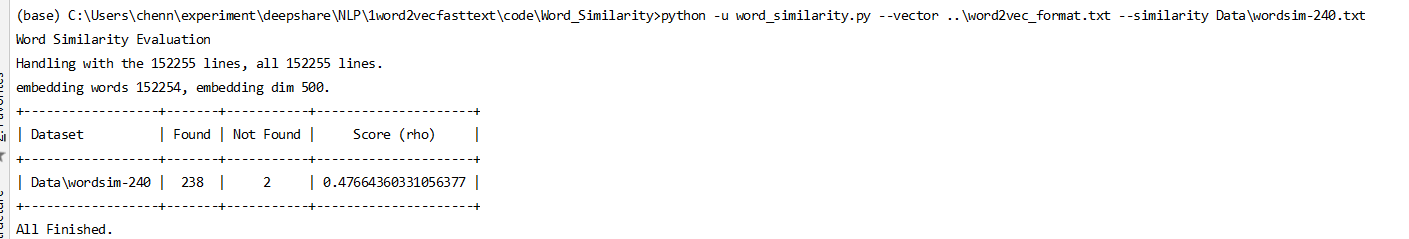
1. model = Word2Vec(LineSentence(wiki\_news), sg=0,size=200, window=5, min\_count=10, workers=8)

若提高min\_count,size不变，训练4分种，240准确率提升到0.513.

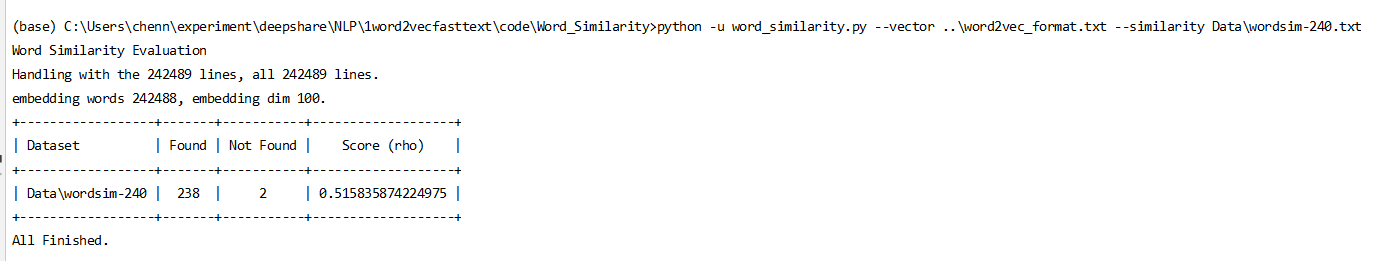


最后降低窗口数，结果准确率明显下滑（大gram追求准确度，小gram追求速度）。

1. model = Word2Vec(LineSentence(wiki\_news), sg=0,size=500, window=3, min\_count=10, workers=8)

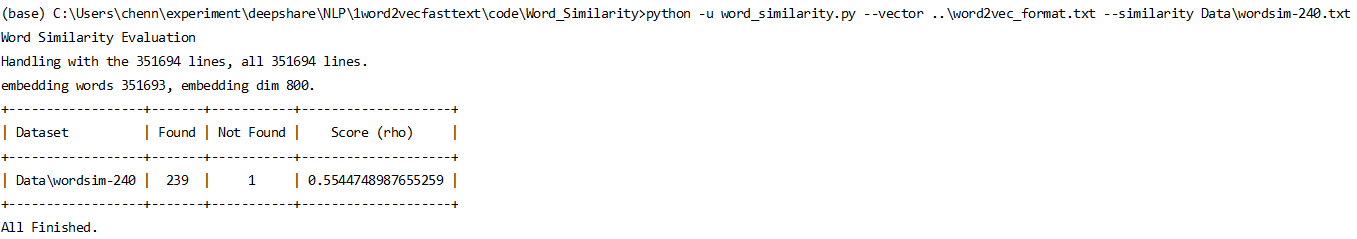


若使用默认参数，则训练五分钟，成绩为0.5158.



而若将参数都调的很高，则效果很好。

1. model = Word2Vec(LineSentence(wiki\_news), sg=0,size=800, window=12, min\_count=3, workers=8)





（写出运行流程、关键代码理解和运行结果截图）

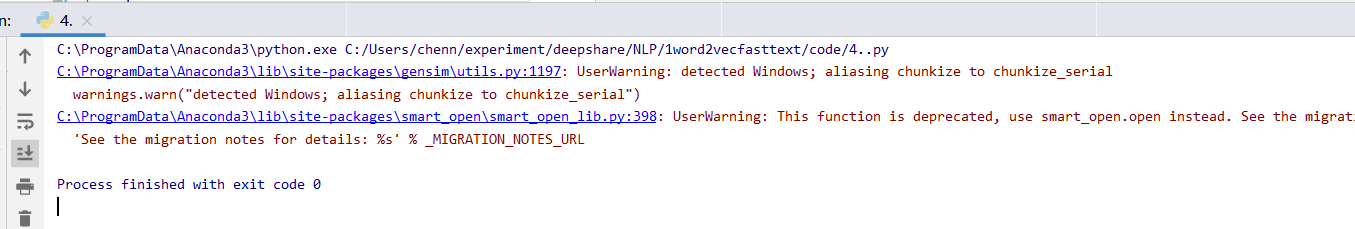
**5.2 Fasttext**

（1）安装包、调试代码跑通整个模型使用Gensim进行词向量训练（运行4.py）

cd fastText-master

pip install . （一定要在install后空一格加‘.’）

（写出运行流程、关键代码理解和运行结果截图）



训练完毕后保存模型和embedding结果。

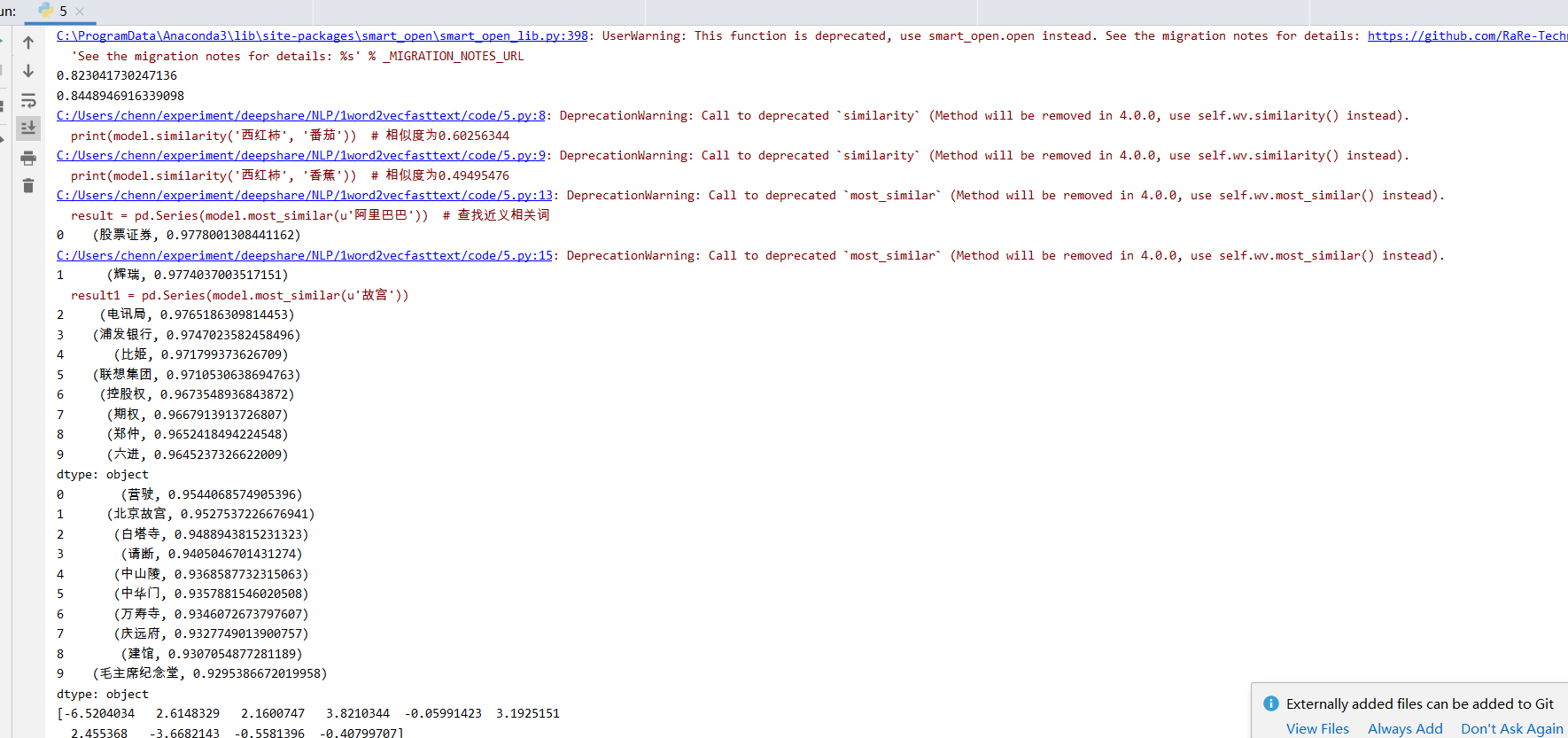
（2）测试同义词，找几个单词，看下效果（运行5.py）

（写出运行流程、关键代码理解和运行结果截图）

同义词结果如下：

西红柿番茄的相似度为0.82，西红柿香蕉的概率为0.84，与阿里巴巴最近的是(股票证券, 0.9778001308441162)，与故宫最近的是(营驶, 0.9544068574905396)。

并求得了中国的词向量。

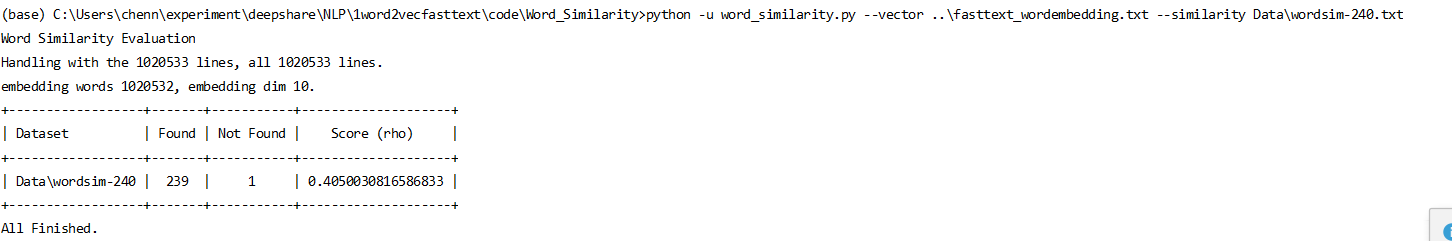


（3）使用训练好的词向量测试词相似度任务的精度

在Word \_Similarity文件夹下运行 ：

python -u word\_similarity.py --vector fasttext\_wordembedding.txt --similarity wordsim-240.txt （--vector 向量文件位置）

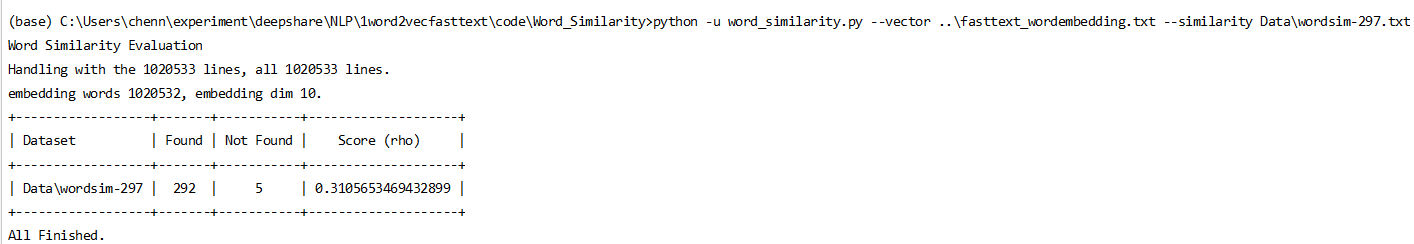
1. model = FastText(sentences,  size=10, window=3, min\_count=1, iter=10,min\_n = 3 , max\_n = 6,word\_ngrams = 0)



python -u word\_similarity.py --vector fasttext\_wordembedding.txt --similarity wordsim-297.txt （--vector 向量文件位置）

1. model = FastText(sentences,  size=10, window=3, min\_count=1, iter=10,min\_n = 3 , max\_n = 6,word\_ngrams = 0)

（写出运行流程、关键代码理解和运行结果截图）



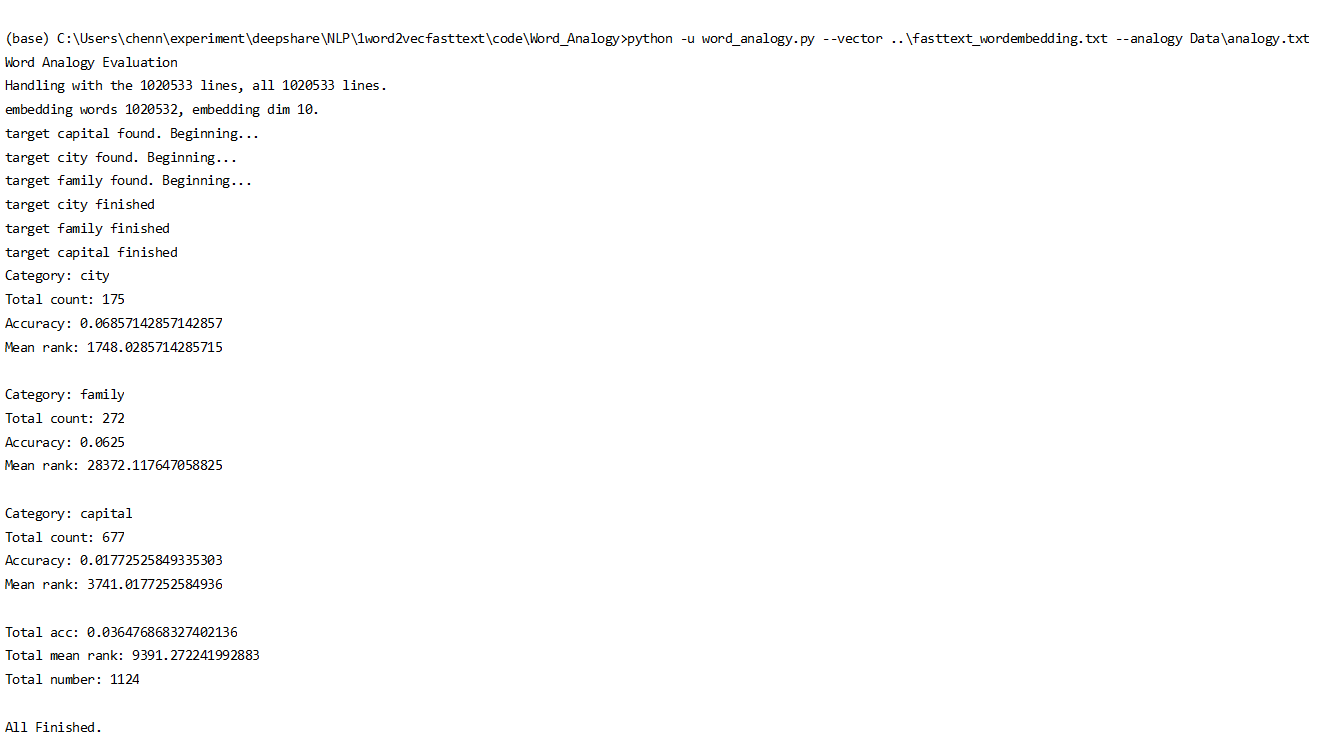
（4）使用训练好的词向量测试词类比任务的精度

在Word\_Analogy文件夹下运行 ：

python -u word\_analogy.py --vector fasttext\_wordembedding.txt -–analogy analogy.txt

（--vector 向量文件位置）

（写出运行流程、关键代码理解和运行结果截图）



可以看到准确率都不是很高。

（5）调试模型中的参数重新训练词向量，提高两个任务的精度

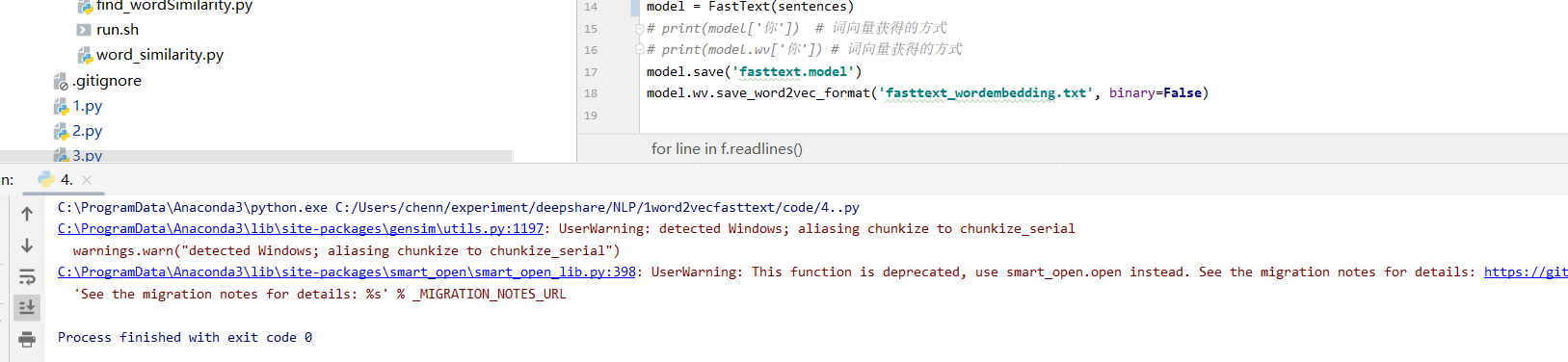


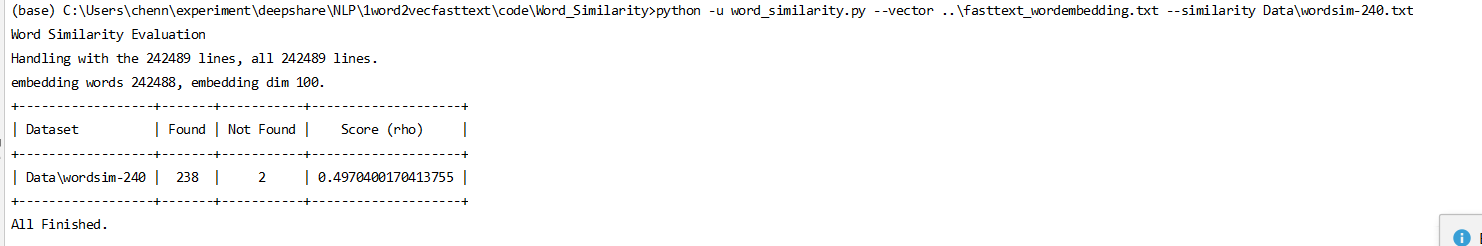
（写出运行流程、关键代码理解和运行结果截图）

1. model = FastText(sentences,  size=10, window=3, min\_count=1, iter=10,min\_n = 3 , max\_n = 6,word\_ngrams = 0)

在240数据集上准确率是0.40

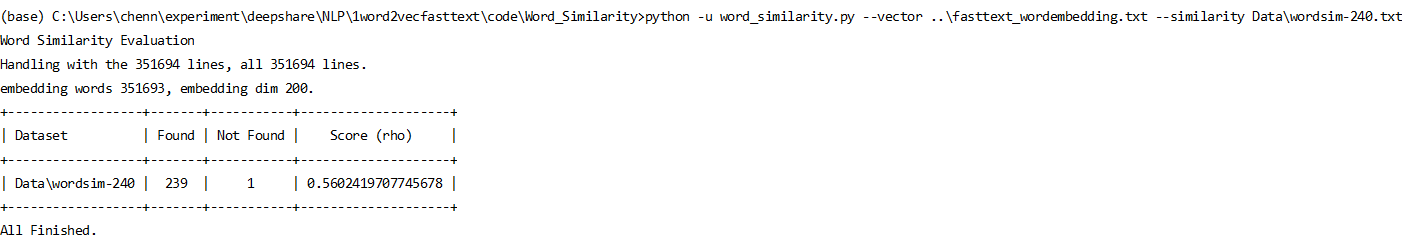
若使用默认参数自动调参，则准确率提升到0.497





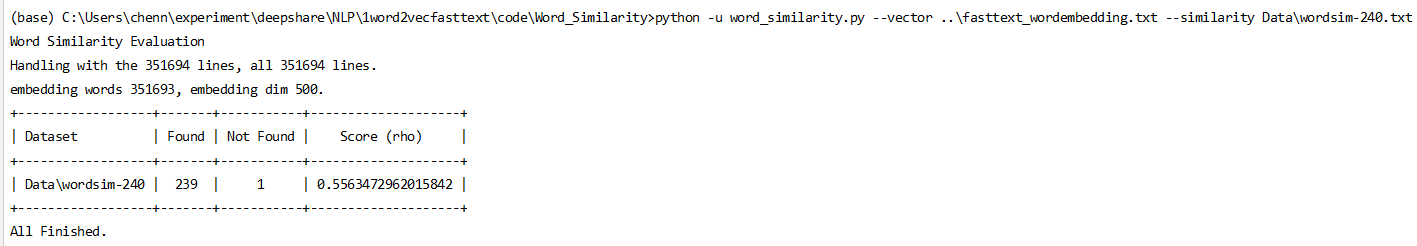
而若将参数都调高则效果显著变好，分数为0.56.

1. model = FastText(sentences, size=200, window=12, min\_count=3, iter=20,min\_n = 5 , max\_n = 10,word\_ngrams = 0)

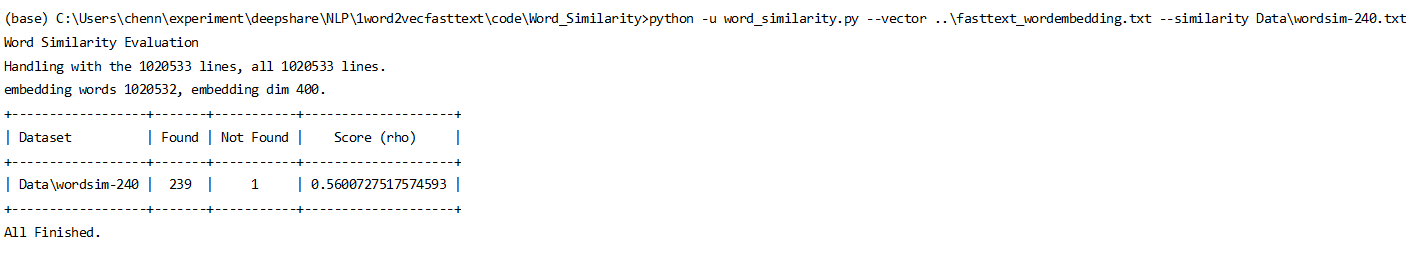


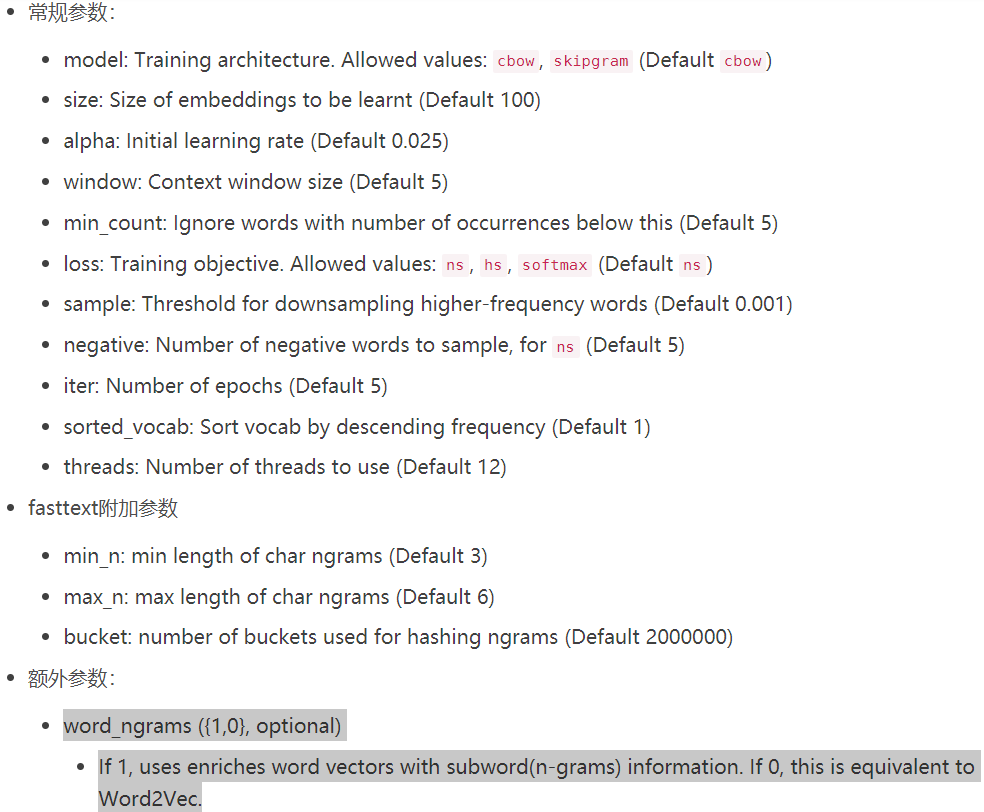
若进一步提高，效果反而会没有上面这么好，究其原因，可能是因为维度越高边际效益会递减，甚至会降低准确率。

1. model = FastText(sentences, size=500, window=15, min\_count=3, iter=30,min\_n = 5 , max\_n = 15,word\_ngrams = 0)



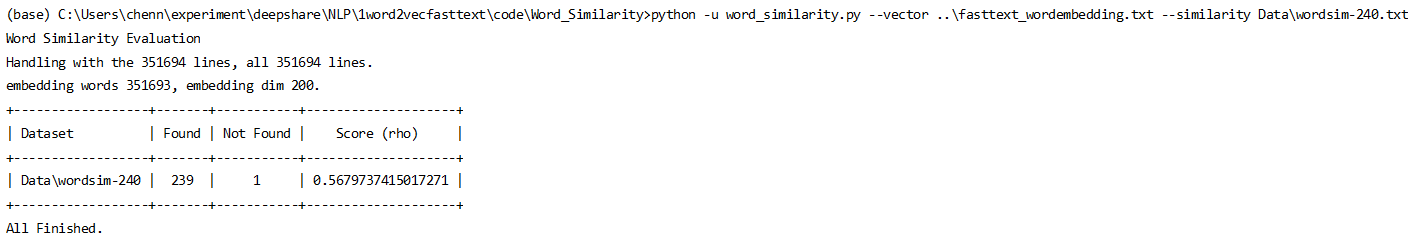
**而其实若观测文档，可以看到把wordgram设为0是无效的，使用的不是fasttext而是word2vec。**

1. model = FastText(sentences, size=400, window=64, min\_count=1, iter=20,min\_n = 3 , max\_n = 25,word\_ngrams = 1) 

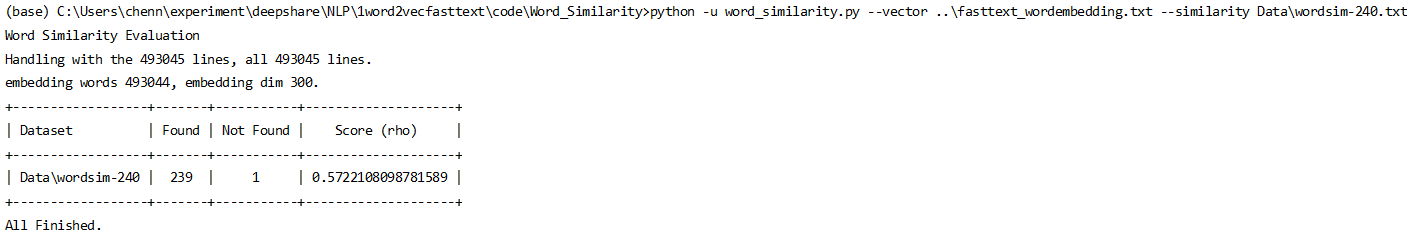


**现把wordgram设为1，分数达到0.567。**

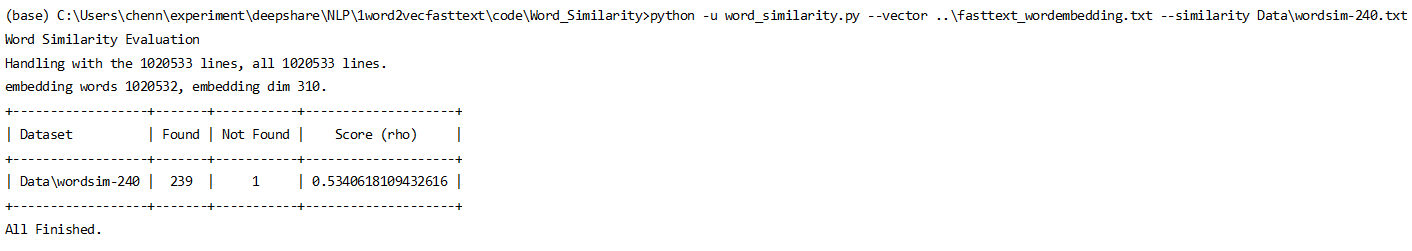
1. model = FastText(sentences, size=200, window=12, min\_count=3, iter=20,min\_n = 5 , max\_n = 10,word\_ngrams = 1)



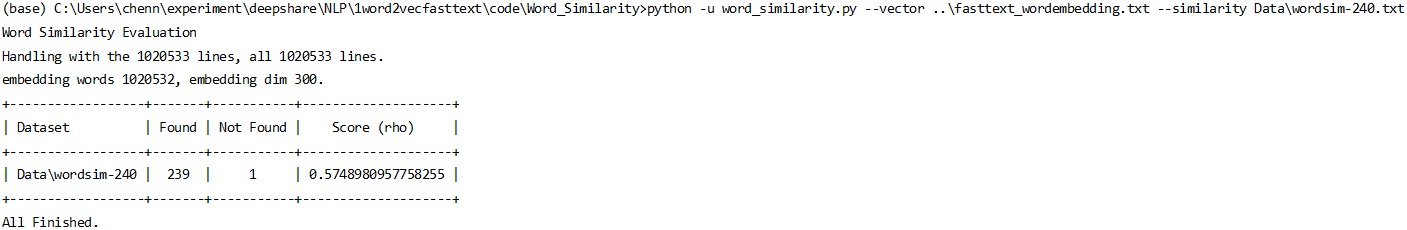
1. model = FastText(sentences, size=300, window=64, min\_count=2, iter=20,min\_n = 3 , max\_n = 15,word\_ngrams = 1)



**若使用***hierarchical softmax的***skipgram，则效果没有负采样更好。**



1. model = FastText(sentences, size=300, window=32, min\_count=1, iter=20,min\_n = 3 , max\_n = 16,word\_ngrams = 1)



**六、实验感想**

**Word2vec提出之前的语言模型是统计语言模型，用语料的频率替代概率，而因为一些词在预料中没有出现过，所以需要使用拉普拉斯平滑，将词频高的词概率降低，低得词概率增高，但这也无法解释一局句子较长的问题，这会使得参数空间过大、数据稀疏严重。于是引入了马尔科夫假设，即假设下一个词只依赖于前一个或几个词,使用k-gram，提升速度（及降低困惑度），而最以前的词的表示方式是通过onehot表示的，这样表示的词越多维数会显著变高，也无法表示词与词的关系，所以word2vec引入了分布式表示的思想，可以将词表大小从V\*V降至V\*K，并给在词与词之间可以通过余弦相似度计算词与词的相似度或关系。其实该思想是1986年Hinton提出的，而首次使用词向量的论文在2003年。实则word2vec的兴起和广泛使用是在2013年的开源word2vec及那篇基于向量空间中词表示的有效估计出现的，他不仅提出了两种新的模型结构，还通过层次softmax和负采样的方法来优化训练使得训练速度进一步加快，并可以在大规模语料上进词向量训练。而它的最重要的研究意义，是可以衡量词向量之间的相似程度（这个方法经过代码运行效果已经超过使用统计模型并使用Ngram），以及进行词类比任务，当然他也能作为预训练模型提升NLP任务的准确率，比如实体识别和文本分类等任务防止过拟合。而前者提到的衡量想死程度和词类比任务，也是作为衡量word2vec模型好坏的一种评价标准，本实验即使用了这两种方法评判最终模型的好坏。回到2003年提出的NNLM和RNNLM，NNLM结构与word2vec的CBOW有些类似，但其实通过源码可以发现，CBOW的实现过程在第二层（中间层）少了很多计算量（如省略了tanh），而在2003年的结构中，其虽然使用了加上softmax的log-linear model，但是他在计算概率时依然复杂，无法避免地要进行V次指数次乘积加法，这也使得计算复杂度明显变高，而使用层次softmax，则只需要计算不到log2V个sigmoid（将softmax计算转换成sigmoid计算 使用一般平衡二叉树是log2v个而是用Huffuman树则不到log2V个 这在运算中也可以体现出 一个要2h一个得一个晚上），然而大部分情况下依然不使用层次softmax而使用负采样，因为负采样效果更好（在之前的实验记录中，同样的参数使用负采样可以到57%准确率而使用层次softmax则只有52准确率），负采样通过增大正样本概率，减少负样本概率，舍弃多分类，提升速度，使用该方法效果显著比在计算上进行优化的层次softmax更好。而不论是使用skipgram或是cbow模型，在这两个任务上准确率都变化不大。然而在调参过程中，其实还是能明显感受到word2vec效率相对低下，在将词向量size调大的过程中，从默认参数的size开始增大调，可以看出准确率有明显提升，但是时间代价也是显著提升的，size从30提升到300-500时间上会有十几二十倍的增长（1h内到约15小时），而且在一个大型的实际的相似度测试和语义关系测试任务中，准确率也没有达到百分之七八十以上，效果只是较2013年的统计模型有了时间和准确度上的提升，并且解决了一些统计无法计算的问题（比如概率过小的精度问题）。（写到这被人打断了，诶感觉也差不多了）然而其实86年提出的分布式语言表示和03年提出的词向量模型才是我们真正方便参考借鉴和沿用的方法，之后的NLP模型也大多使用的是这种方法。**