

第二次实验报告

课程名称	内容安全实验				
学生姓名	陈曦	学号	2020302181081	指导老师	张典
专业	网络安全	班级	2020 级 3 班	实验时间	2022. 3. 20

一、实验描述

1. 基于 word2vec 实现文本分类

Word2Vec 是生成词嵌入向量的方式。词嵌入向量可以标识一个单词固定长度向量，每个维度都有含义。
通过代码实现 word2vec 训练模型，并应用模型，计算文本分类的结果以及准确度。

2. 岗位关键词提取

从招聘数据看各岗位招聘需求关键字。
根据自己的意向工作和市场需求完善自身的技术栈，增强自身的软实力和硬实力。
根据不同行业 and 不同岗位的关键词，有针对性的修改、美化、设计自身的简历。

二、实验原理

1. 基于 word2vec 实现文本分类

Word2vec, 是一群用来产生词向量的相关模型。这些模型为浅而双层的神经网络，用来训练以重新建构语言学之词文本。网络以词表现，并且需猜测相邻位置的输入词，在 word2vec 中词袋模型假设下，词的顺序是不重要的。训练完成之后，word2vec 模型可用来映射每个词到一个向量，可用来表示词对词之间的关系，该向量为神经网络之隐藏层。

本实验对目标文本产生词向量，并用 70%的词向量训练模型，判断属于何种类型文本，并用 30%的词向量来测试模型的准确度。

2. 岗位关键词提取

使用两种不同的算法提取文本中的 20 个关键词，两种算法分别是 TF-IDF 算法，

TextRank 算法。

(1) TF-IDF 算法

TF，指词频，即一个词在文中出现的次数，统计出来就是词频 TF。TF 用于评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。停用词不包括在内。

IDF，指逆文档频率，这是一个词语“权重”的度量，在词频的基础上，如果一个词在多篇文档中词频较低，也就表示这是一个比较少见的词，但在某一篇文章中却出现了很多次，则这个词 IDF 值越大，在这篇文章中的“权重”越大。所以当词越常见，IDF 越低。

计算出 TF 和 IDF 的值后，两个一乘就得到 TF-IDF，这个词的 TF-IDF 越高就表示，就表示在这篇文章中的重要性越大，越有可能就是文章的关键词。而 Python 的 scikit-learn 包下有计算 TF-IDF 的 API，我们就用这个来简单的实现抽取文章关键词。

(2) TextRank 算法

TextRank 算法是一种基于图的用于关键词抽取和文档摘要的排序算法，由谷歌的网页重要性排序算法 PageRank 算法改进而来，它利用一篇文档内部的词语间的共现信息(语义)便可以抽取关键词，它能够从一个给定的文本中抽取该文本的关键词、关键词组，并使用抽取式的自动文摘方法抽取该文本的关键词。

TextRank 算法的基本思想是将文档看作一个词的网络，该网络中的链接表示词与词之间的语义关系。

算法计算公式：

$$WS(V_i) = (1 - d) + d * \sum_{V_j \in In(V_i)} \frac{W_{ji}}{\sum_{V_k \in Out(V_j)} W_{jk}} WS(V_j)$$

其中，WS (Vi)表示句子 i 的权重，右侧的求和表示每个相邻句子对本句子的贡献程度，在单文档中，我们可以粗略地认为所有句子都是相邻的，不需要想多文档一样进行多个窗口的生成和抽取。Wji 表示两个句子的相似度 WS (Vj) 代表上次迭代出的句子 j 的权重。d 是阻尼系数，一般为 0.85。

关键词抽取是指从文本中确定一些能够描述文档含义的术语的过程。对关键词抽取而言，用于构建顶点集的文本单元可以是句子中的一个或多个字；根据这些字之间的关系（比如：在一个框中同时出现）构建边。根据任务的需要，可以使用语法过滤器（syntactic filters）对顶点集进行优化。语法过滤器的主要作用是将某一类或者某几类词性的字过滤出来作为顶点集。

三、实验步骤

1. 基于 word2vec 实现文本分类

【配置环境】

安装工具 python-Levenshtein 防止出现 warning。

```
PS D:\chenxi\code\worm2> pip install python-Levenshtein
```

安装 python-Levenshtein 成功。

```
Collecting python-Levenshtein
  Downloading python-Levenshtein-0.20.9-py3-none-any.whl (9.4 kB)
Collecting Levenshtein==0.20.9
  Downloading Levenshtein-0.20.9-cp39-cp39-win_amd64.whl (101 kB)
    |#####| 101.3/101.3 kB 364.4 kB/s eta 0:00:00
Collecting rapidfuzz<3.0.0,>=2.3.0
  Downloading rapidfuzz-2.13.7-cp39-cp39-win_amd64.whl (1.0 MB)
    |#####| 1.0/1.0 MB 791.2 kB/s eta 0:00:00
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Installing collected packages: rapidfuzz, Levenshtein, python-Levenshtein

Successfully installed Levenshtein-0.20.9 python-Levenshtein-0.20.9 rapidfuzz-2.13.7
```

检查 Jieba 工具库安装完毕。

```
Requirement already satisfied: jieba in d:\chenxi\try\ide\anaconda\lib\site-packages (0.42.1)
```

检查 Gensim 工具库安装完毕。

```
Requirement already satisfied: gensim in d:\chenxi\try\ide\anaconda\lib\site-packages (4.2.0)
```

检查 Numpy 工具库安装完毕。

```
Requirement already satisfied: numpy in d:\chenxi\try\ide\anaconda\lib\site-packages (1.20.3)
```

【运行代码】

导入工具库。

需要在 import torch 之前添加 os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"，否则会报错。导入 os, re, sys, torch, numpy 库，以及 word2vec 作为模型需要的库，jieba 分词库，生成统计图的 matplotlib 工具，tqdm 和 pickle 工具。

```
1. import os
```

```

2. os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
3. import re
4. import sys
5. import jieba
6. import torch
7. from gensim.models import Word2Vec, word2vec
8. import numpy as np
9. import jieba.analyse
10. import matplotlib.pyplot as plt
11. from tqdm import tqdm
12. import pickle

```

使用下面两条语句，第一句为用来正常显示中文标签，第二句为正常显示负号。

```

1. plt.rcParams['font.sans-serif'] = ['SimHei']
2. plt.rcParams['axes.unicode_minus'] = False

```

固定随机数。设置文件路径“data”，设置所有词的列表 all_word_list。

```

1. np.random.seed(100)
2. torch.cuda.manual_seed(100)
3. sys.stdout.flush()
4. nFile = 200
5. root_path = "data"
6. class_list = os.listdir(root_path)
7. all_word_list = []

```

对每一个文件，分词并添加到所有词列表中，用空格隔开词语。

```

1. for c in class_list:
2.     class_path = root_path + "/" + c
3.     file_list = os.listdir(class_path)
4.     for name in file_list:
5.         file_path = class_path + "/" + name
6.         with open(file_path, "r", encoding="utf-8") as f:
7.             txt = f.read()
8.             txt = re.sub("[\t\n]*", "", txt)
9.             word_list = jieba.analyse.texttrank(txt, topK=None, withWeight=False, allowPOS=('ns', 'n', 'vn', 'v'))
10.            all_word_list.extend(word_list)
11. result = " ".join(all_word_list)

```

生成 result.txt，将存放分词结果的长字符串存在文本文件中。

```

1. with open("result.txt", "w", encoding="utf-8") as f:
2.     f.write(result)
3. f.close()

```

加载语料，也就是文本文件 result.txt，并使用 word2vec 生成词向量，存放在 sentences 中。并训练模型，维度设置为 200，最小词频设置为 1。模型保存在 my_model.model 中。

```
1. sentences = word2vec.Text8Corpus("result.txt")
2. model = word2vec.Word2Vec(sentences, vector_size = 200, min_count
    = 1)
3. model.save("my_model.model")
```

这里使用 pickle 工具生成 all_word_list.pkl 文件。在机器学习中，我们常常需要把训练好的模型存储起来，这样在进行决策时直接将模型读出，而不需要重新训练模型，这样就大大节约了时间。Python 提供的 pickle 模块就很好地解决了这个问题，它可以序列化对象并保存到磁盘中，并在需要的时候读取出来，任何对象都可以执行序列化操作。Dump 函数的功能是将 obj 对象序列序列化存入已经打开的 file 中。

```
1. model.save("my_model.model")
2. with open("all_word_list.pkl", "wb") as f:
3.     pickle.dump(all_word_list, f)
4. f.close()
```

这里仍然使用 pickle 工具，将刚才的 all_word_list.pkl 文件中的对象序列化读出。

```
1. model = Word2Vec.load("my_model.model")
2. with open("all_word_list.pkl", 'rb') as f:
3.     all_word_list = pickle.load(f)
4. f.close()
```

进行词频统计。取 70% 的文件作为训练集，并对训练集进行词频统计。

```
1. def train_f():
2.     class_all_words = {}
3.     print("对训练集词频统计")
4.     for c in tqdm(class_list):
5.         all_words = {}
6.         class_path = root_path + "/" + c
7.         file_list = os.listdir(class_path)
8.         for name in file_list[:int(nFile * rate)]:
9.             file_path = class_path + "/" + name
10.            with open(file_path, "r", encoding="utf-8") as f:
11.                txt = f.read()
12.                txt = re.sub("[\t\n]*", "", txt)
13.                word_list = jieba.analyse.textcrank(txt, withWeight=False, allowPOS=('ns', 'n', 'vn', 'v'))
14.                for word in word_list:
15.                    if len(word) != 1:
16.                        if word in all_words.keys():
```

```

17.         all_words[word] += 1
18.     else:
19.         all_words[word] = 1
20.     class_all_words[c] = all_words
21.     with open('all_words_practice.pkl', 'wb') as f:
22.         pickle.dump(class_all_words, f)
23.     f.close()
24.     return class_all_words

```

获取每个类的平均向量。按词的数量进行倒序排列，关闭将每个词的向量映射，最后求得每个分类的平均向量。

```

1.     def average_class_vector(keyword_num):
2.         average_class_dic = {}
3.         for c in class_all_words:
4.             all_word_list = sorted(class_all_words[c].items(),
5.                                     key = lambda item: item[1], reverse=True)
6.             total = 0
7.             if keyword_num > len(all_word_list):
8.                 for t in range(len(all_word_list)):
9.                     total += model.wv[all_word_list[t][0]]
10.            else:
11.                for t in range(keyword_num):
12.                    total += model.wv[all_word_list[t][0]]
13.            average_class_dic[c] = total/keyword_num
14.        return average_class_dic

```

计算两个向量之间的余弦相似度，来判断两个文本的相似度。

```

1.     def cos_sim(vector_a, vector_b):
2.         """
3.         计算两个向量之间的余弦相似度
4.         """
5.         vector_a = np.array(vector_a)
6.         vector_b = np.array(vector_b)
7.         a_norm = np.linalg.norm(vector_a)
8.         b_norm = np.linalg.norm(vector_b)
9.         sim = np.dot(vector_a, vector_b) / (a_norm * b_norm)
10.        return sim

```

将数据和类别数据相比较，相似度最高的就是文本所属的类别。

```

1.     def predict(data):
2.         sim = {}
3.         for c in average_class_dic:
4.             sim[c] = cos_sim(data, average_class_dic[c])

```

```

5.     test_words_list = sorted(sim.items(), key = lambda item:item[
1], reverse=True)
6.     return test_words_list[0][0]

```

取 30%的文件作为测试集，并对训练集中每个文件单独进行词频统计。返回模型分类的准确率。

```

1.  def acc(keyword_num):
2.      true = 0
3.      false = 0
4.      print("Keyword_num: {}".format(keyword_num))
5.      for c in tqdm(class_list):
6.          class_path = root_path + "/" + c
7.          file_list = os.listdir(class_path)
8.          for name in file_list[int(nFile * rate):]:
9.              file_path = class_path + "/" + name
10.             print(file_path)
11.             with open((file_path), 'r', encoding="UTF-8") as f:
12.                 test_data_words = {}
13.                 txt = f.read()
14.                 txt = re.sub("[\t\n]*", "", txt)
15.                 word_list = jieba.analyse.texttrank(txt, withWeight=False, allowPOS=('ns', 'n', 'vn', 'v'))
16.                 for word in word_list:
17.                     if len(word) != 1:
18.                         if word in test_data_words.keys():
19.                             test_data_words[word] += 1
20.                         else:
21.                             test_data_words[word] = 1
22.                 test_words_list = sorted(test_data_words.items(), key =
lambda item:item[1], reverse=True)
23.                 total = 0
24.                 if keyword_num > len(test_words_list):
25.                     for t in range(len(test_words_list)):
26.                         total += model.wv[test_words_list[t][0]]
27.                 else:
28.                     for t in range(keyword_num):
29.                         total += model.wv[test_words_list[t][0]]
30.                 average_test_vector = total / keyword_num
31.                 pre = predict(average_test_vector)
32.                 if pre == c:
33.                     true += 1
34.                 else:
35.                     false += 1
36.             return true / (true + false)

```

设置参数，运行程序，画出关键词个数与准确率的关系图像。并打印准确率。

```
1.  if __name__ == '__main__':
2.      rate = 0.75
3.      keyword_num_list = [1, 3, 5, 10, 15, 20, 30, 50, 100]
4.      acc_list = []
5.      class_all_words = train_f()
6.      with open('all_words_practice.pkl', 'rb') as f:
7.          class_all_words = pickle.load(f)
8.      f.close()
9.      print("计算准确率")
10.     for keyword_num in keyword_num_list:
11.         average_class_dic = average_class_vector(keyword_num)
12.         acc_list.append(round(acc(keyword_num), 3))
13.     fig = plt.figure()
14.     ax = fig.add_subplot(1, 1, 1)
15.     ax.set_title("关键词个数与准确率的关系")
16.     ax.set_xlabel('关键词个数')
17.     ax.set_ylabel('准确率')
18.     plt.plot(keyword_num_list, acc_list, color='black', markerfacecolor='r', marker='o')
19.     print(acc_list)
20.     for a, b in zip(keyword_num_list, acc_list):
21.         plt.text(a, b, (a, b), ha='center', va='bottom')
22.     plt.show()
```

2. 岗位关键词提取

【TF-IDF 算法】

导入工具包。

NLTK 是英文分词工具。CountVectorizer 类会将文本中的词语转换为词频矩阵。而 TfidfTransformer 类用于统计每个词语的 TF-IDF 值。

```
1.  import csv
2.  import numpy as np
3.  from nltk.tokenize import word_tokenize
4.  from sklearn.feature_extraction.text import CountVectorizer
5.  from sklearn.feature_extraction.text import TfidfTransformer
```

代码入口先设置停用词为一个列表。

```
1.  if __name__ == '__main__':
```



```

2.     stopwords = ['very', 'ourselves', 'am', 'doesn', 'through', 'me',
    'against', 'up', 'just', 'her', 'ours',
3.         'couldn', 'because', 'is', 'isn', 'it', 'only', 'in', 'su
    ch', 'too', 'mustn', 'under', 'their',
4.         'if', 'to', 'my', 'himself', 'after', 'why', 'while', 'ca
    n', 'each', 'itself', 'his', 'all', 'once',
5.         'herself', 'more', 'our', 'they', 'hasn', 'on', 'ma', 'th
    em', 'its', 'where', 'did', 'll', 'you',
6.         'didn', 'nor', 'as', 'now', 'before', 'those', 'yours', '
    from', 'who', 'was', 'm', 'been', 'will',
7.         'into', 'same', 'how', 'some', 'of', 'out', 'with', 's',
    'being', 't', 'mightn', 'she', 'again', 'be',
8.         'by', 'shan', 'have', 'yourselves', 'needn', 'and', 'are
    ', 'o', 'these', 'further', 'most', 'yourself',
9.         'having', 'aren', 'here', 'he', 'were', 'but', 'this', 'm
    yself', 'own', 'we', 'so', 'i', 'does', 'both',
10.        'when', 'between', 'd', 'had', 'the', 'y', 'has', 'down',
    'off', 'than', 'haven', 'whom', 'wouldn',
11.        'should', 've', 'over', 'themselves', 'few', 'then', 'ha
    dn', 'what', 'until', 'won', 'no', 'about',
12.        'any', 'that', 'for', 'shouldn', 'don', 'do', 'there', 'd
    oing', 'an', 'or', 'ain', 'hers', 'wasn',
13.        'weren', 'above', 'a', 'at', 'your', 'theirs', 'below', '
    other', 'not', 're', 'him', 'during', 'which']

```

打开指定的文件 JobDataAnalyst.csv，读取文本内容，并用 column 列表存储第四列（row[3]），岗位描述 Description 的内容，每一条内容都是一个元素。

```

1.     with open("JobDataAnalyst.csv", "rt", encoding="utf-8") as file
    :
2.         reader = csv.reader(file)
3.         column = [row[3] for row in reader]

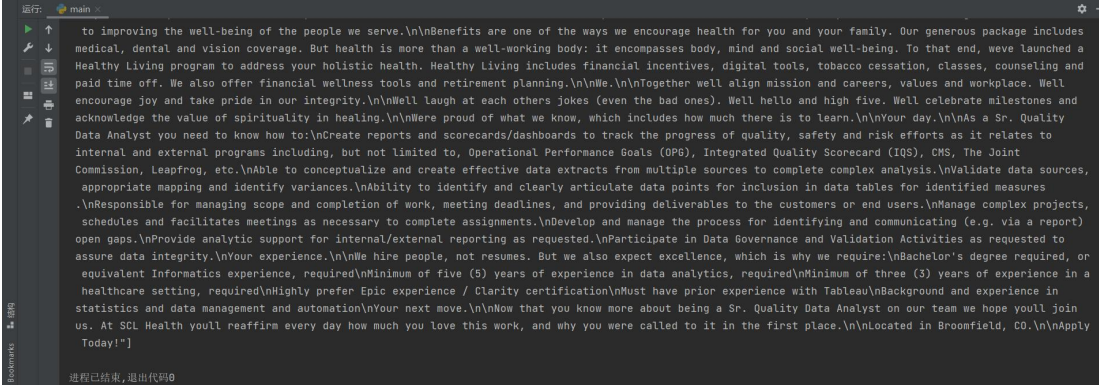
```

打印 column 的内容，可以看到岗位描述。

```

1.     print(column)

```



```

运行: main
to improving the well-being of the people we serve.\n\nBenefits are one of the ways we encourage health for you and your family. Our generous package includes medical, dental and vision coverage. But health is more than a well-working body; it encompasses body, mind and social well-being. To that end, we've launched a Healthy Living program to address your holistic health. Healthy Living includes financial incentives, digital tools, tobacco cessation, classes, counseling and paid time off. We also offer financial wellness tools and retirement planning.\n\nWe.\n\nTogether we'll align mission and careers, values and workplace. We'll encourage joy and take pride in our integrity.\n\nWe'll laugh at each other's jokes (even the bad ones). We'll hello and high five. We'll celebrate milestones and acknowledge the value of spirituality in healing.\n\nWe're proud of what we know, which includes how much there is to learn.\n\nYour day.\n\nAs a Sr. Quality Data Analyst you need to know how to:\n\nCreate reports and scorecards/dashboards to track the progress of quality, safety and risk efforts as it relates to internal and external programs including, but not limited to, Operational Performance Goals (OPG), Integrated Quality Scorecard (IQS), CMS, The Joint Commission, Leapfrog, etc.\n\nAble to conceptualize and create effective data extracts from multiple sources to complete complex analysis.\n\nValidate data sources, appropriate mapping and identify variances.\n\nAbility to identify and clearly articulate data points for inclusion in data tables for identified measures.\n\nResponsible for managing scope and completion of work, meeting deadlines, and providing deliverables to the customers or end users.\n\nManage complex projects, schedules and facilitates meetings as necessary to complete assignments.\n\nDevelop and manage the process for identifying and communicating (e.g. via a report) open gaps.\n\nProvide analytic support for internal/external reporting as requested.\n\nParticipate in Data Governance and Validation Activities as requested to assure data integrity.\n\nYour experience.\n\nWe hire people, not resumes. But we also expect excellence, which is why we require:\n\nBachelor's degree required, or equivalent Informatics experience, required\n\nMinimum of five (5) years of experience in data analytics, required\n\nMinimum of three (3) years of experience in a healthcare setting, required\n\nHighly prefer Epic experience / Clarity certification\n\nMust have prior experience with Tableau\n\nBackground and experience in statistics and data management and automation\n\nYour next move.\n\nNow that you know more about being a Sr. Quality Data Analyst on our team we hope you'll join us. At SDL Health you'll reaffirm every day how much you love this work, and why you were called to it in the first place.\n\nLocated in Broomfield, CO.\n\nApply Today!")

进程已结束,退出代码0

```

接下来需要对 column 列表中的描述进行英文分词。这里用到工具 NLTK。在使用之前，我们需要先安装 NLTK 工具包。

使用命令 `pip install nltk`。安装成功。

```
PS D:\chenxi\code\worm2.2> pip install nltk
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Requirement already satisfied: nltk in d:\chenxi\try\ide\anaconda\lib\site-packages (3.6.5)
Requirement already satisfied: click in d:\chenxi\try\ide\anaconda\lib\site-packages (from nltk) (8.0.3)
Requirement already satisfied: joblib in d:\chenxi\try\ide\anaconda\lib\site-packages (from nltk) (1.1.0)
Requirement already satisfied: regex<=2021.8.3 in d:\chenxi\try\ide\anaconda\lib\site-packages (from nltk) (2021.8.3)
Requirement already satisfied: tqdm in d:\chenxi\try\ide\anaconda\lib\site-packages (from nltk) (4.62.3)
Requirement already satisfied: colorama in d:\chenxi\try\ide\anaconda\lib\site-packages (from click->nltk) (0.4.4)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
```

再编写分词程序。需要将把所有描述分词，并将所有的分词存放在字符串变量 wordlist 中。对于 column 中的每一个描述，都分词，再用空格分开转化为字符串，最后存放在 wordlist 的后面。

处于程序需要，将 wordlist 转化为列表 wordlist2。

```
1. wordlist = ''
2.     test = []
3.     for i in range(0, len(column)):
4.         token = word_tokenize(column[i])
5.         content = ' '.join(token)
6.         wordlist = wordlist + content
7.     wordlist2 = [wordlist]
```

这时执行程序，发现 NLTK 工具无法使用，原因是找不到包。

```
LookupError:
*****
Resource punkt not found.
Please use the NLTK Downloader to obtain the resource:

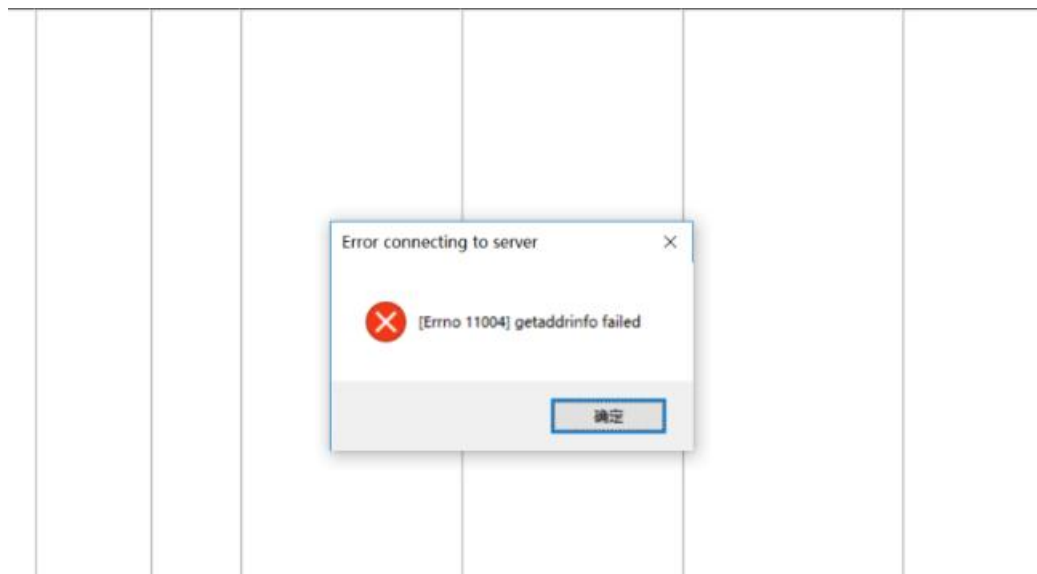
>>> import nltk
>>> nltk.download('punkt')
```

打开终端，输入 python

```
PS D:\chenxi\code\worm2.2> python
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
```

输入语句：import nltk，以及 nltk.download()，来尝试导入包。

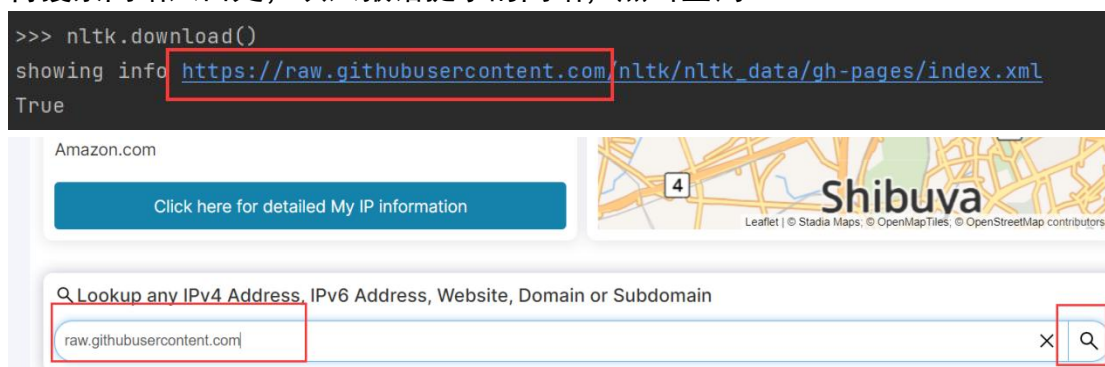
但是交互界面发生错误



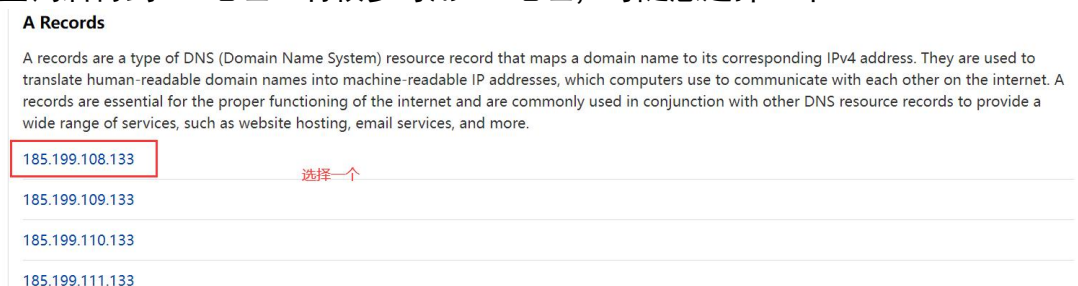
需要配置 IP 地址。翻墙后打开 ipaddress.com 网站。



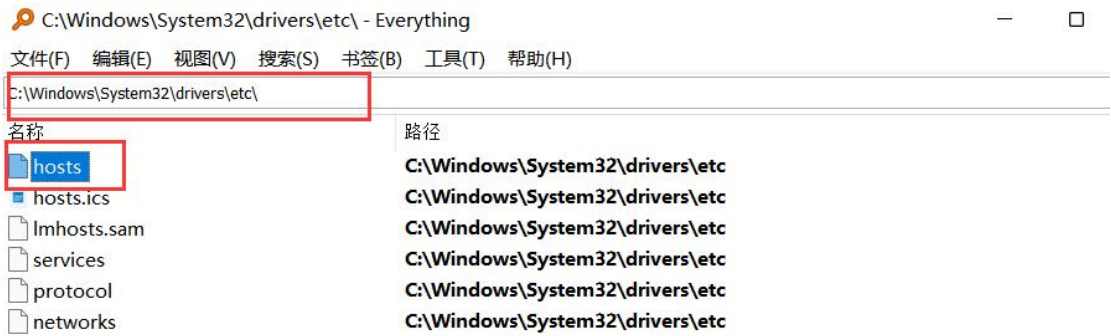
再搜索网站入口处，填入报错提示的网站，点击查询。



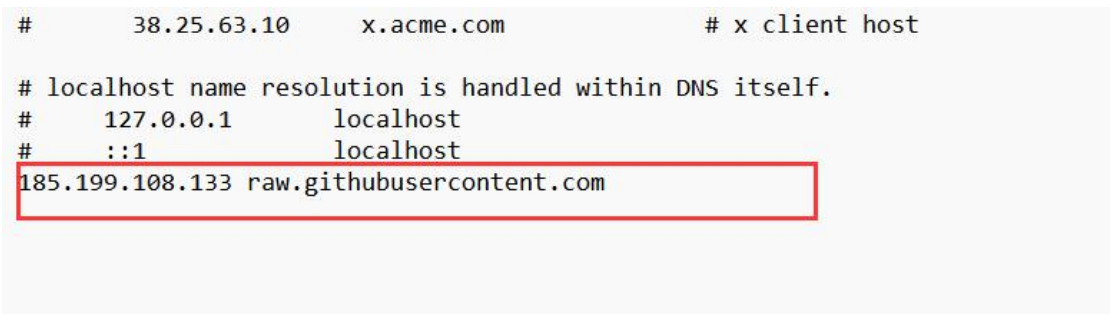
查询后得到 IP 地址。有很多可用 IP 地址，可随意选择一个。



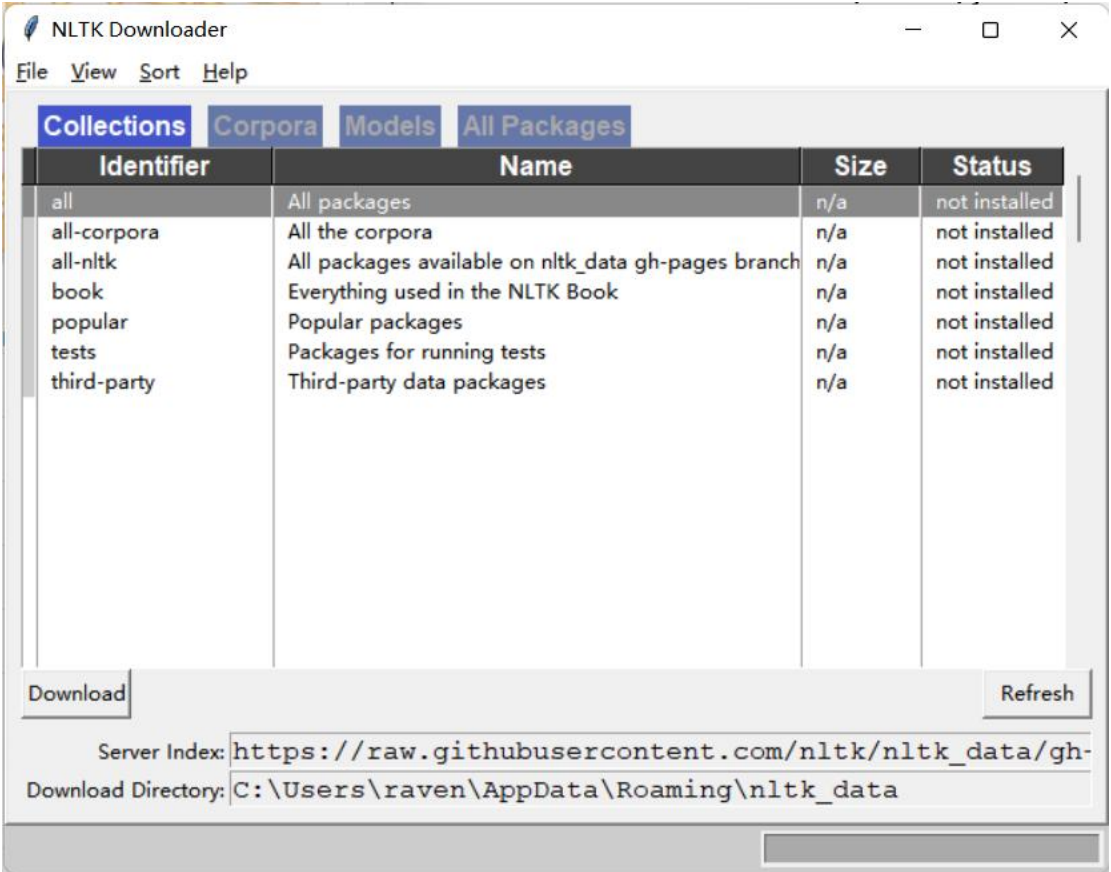
在路径 C:\Windows\System\drivers\etc\ 中找到 hosts 文件。



用记事本打开，并添加刚才查询的 IP，以及导入工具报错提示的网址。



这时再在终端使用命令 `nltk.download()` 就可以打开可视化界面。



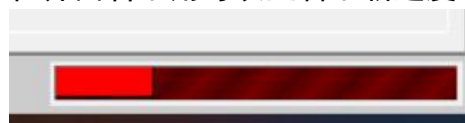
可以下载包。选择 all-nltk。

Collections Corpora Models All Packages	
Identifier	
all	All packages
all-corpora	All the corpora
all-nltk	All packages available on nltk_data gh-pages branch
book	Everything used in the NLTK Book
popular	Popular packages
tests	Packages for running tests
third-party	Third-party data packages

再下载程序运行报错提示的缺乏的包，punkt。

porter test	Porter Stemmer Test Files
punkt	Punkt Tokenizer Models
rslp	RSLP Stemmer (Removed or de Sufijos)
sample_grammars	Sample Grammars
snowball_data	Snowball Data
spanish_grammars	Grammars for Spanish
tagsets	Help on Tagsets

在界面右下角可以查看下载进度。



这时在再次运行程序，没有报错。

尝试打印 wordlist。

```
1. print(wordlist2)
```

跑出结果，是一个大列表。词和词之间用空格分开。

```
运行: main
extracts from multiple sources to complete complex analysis . Validate data sources , appropriate mapping and identify variances . Ability to identify and
clearly articulate data points for inclusion in data tables for identified measures . Responsible for managing scope and completion of work , meeting deadlines
, and providing deliverables to the customers or end users . Manage complex projects , schedules and facilitates meetings as necessary to complete assignments .
Develop and manage the process for identifying and communicating ( e.g . via a report ) open gaps . Provide analytic support for internal/external reporting as
requested . Participate in Data Governance and Validation Activities as requested to assure data integrity . Your experience . We hire people , not resumes .
But we also expect excellence , which is why we require : Bachelor 's degree required , or equivalent Informatics experience , required Minimum of five ( 5 )
years of experience in data analytics , required Minimum of three ( 3 ) years of experience in a healthcare setting , required Highly prefer Epic experience /
Clarity certification Must have prior experience with Tableau Background and experience in statistics and data management and automation Your next move . Now
that you know more about being a Sr. Quality Data Analyst on our team we hope you'll join us . At SCI Health you'll reaffirm every day how much you love this work
, and why you were called to it in the first place . Located in Broomfield , CO . Apply Today [ ]
进程已结束,退出代码0
```

使用 TF-IDF 算法，需要下载工具库。

```
PS D:\chenxi\code\worm2.2> pip install scikit-learn
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Requirement already satisfied: numpy>=1.13.3 in d:\chenxi\try\ide\anaconda\lib\site-packages (from scikit-learn) (1.20.3)
Requirement already satisfied: joblib>=0.11 in d:\chenxi\try\ide\anaconda\lib\site-packages (from scikit-learn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\chenxi\try\ide\anaconda\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: scipy>=0.19.1 in d:\chenxi\try\ide\anaconda\lib\site-packages (from scikit-learn) (1.7.1)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
```

工具下载成功之后就可以使用函数。

文本中的词语转换为词频矩阵，函数存为 `vectorizer`。

统计每个词语的 TF-IDF 值，此函数存为 `transformer`。

将 `wordlist2` 统计词频并学习词语词典，返回文档矩阵，矩阵中元素为词语出现的次数。最后再进行 TF-IDF 值统计，存在 `tfidf` 中。

设 `words` 为所有文本的关键字，获取特征整数索引到特征名称映射的数组，即文档中所有关键字的数组。

`Toarray` 为将 `tfidf` 中的对象转换为数组，存为 `weight` 中，表示权重数组，方便后续排序。

```
1. vectorizer = CountVectorizer()
2. transformer = TfidfTransformer()
3. tfidf = transformer.fit_transform(vectorizer.fit_transform(wordlist2))
4. words = vectorizer.get_feature_names()
5. weight = tfidf.toarray()
```

`i` 为循环索引，`n` 为输出的关键词的个数。因为有的关键词可能是停用词，所以如果排前 20 的词中有停用词，那么就要忽略此停用词。所以添加变量 `num` 为输出的关键词序列。如果关键词是停用词的话则 `num` 序列不变化。

当 `num` 序列小于规定的 20 时，打印关键词字符串和它的权重，并将 `num` 序列加一。如果是停用词的话就继续下一个循环。

```
1. i = 0
2. n = 20
3. num = 0
4. for (title, w) in zip(wordlist2, weight):
5.     loc = np.argsort(-w)
6.     while num < n:
7.         if words[loc[i]] not in stopwords:
8.             print('•
{}: {} {}'.format(str(num + 1), words[loc[i]], w[loc[i]]))
9.             num += 1
10.            i += 1
11.        else:
12.            i += 1
```

【TextRank 算法】

下载工具包 spacy。是一个自然语言处理工具包。

```
PS D:\chenxi\code\worm2.2> pip install spacy
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Collecting spacy
  Downloading spacy-3.5.1-cp39-cp39-win_amd64.whl (12.2 MB)
    12.2/12.2 MB 2.8 MB/s eta 0:00:00
Collecting thinc<8.2.0,>=8.1.8
```

安装成功

```
Successfully installed blis-0.7.9 catalogue-2.0.8 colorama-0.4.6 confection-0.0.4 cyemem-2.0.7 langcodes-3.3.0 mureurhash-1.0.9 pathy-0.10.1 preshed-3.0.8 pydantic-1.10.7 spacy-3.5.1 spacy-legacy-3.0.12 spacy-loggers-1.0.4 srsly-2.4.6 thinc-8.1.9 typer-0.7.0 typing-extensions-4.5.0 wasabi-1.1.1
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
```

还需要安装工具 en_core_web_sm，否则会报错。

```
PS D:\chenxi\code\worm2.2> python -m spacy download en_core_web_sm
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Collecting en-core-web-sm==3.5.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.5.0/en_core_web_sm-3.5.0.tar.gz (12.8 MB)
    12.8/12.8 MB 3.6 MB/s eta 0:00:00
```

安装成功

```
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

导入工具包。并导入停用词包。

1. `import csv`
2. `from nltk.tokenize import word_tokenize`
3. `from collections import OrderedDict`
4. `import numpy as np`
5. `import spacy`
6. `from spacy.lang.en.stop_words import STOP_WORDS`

En_core_web_sm 是英语多任务 CNN，在 OntoNotes 上训练。导入并保存为 nlp。

1. `nlp = spacy.load('en_core_web_sm')`

定义 python 类 textrank4keyword。定义设定值。

1. `class TextRank4Keyword():`
- 2.
3. `def __init__(self):`
4. `self.d = 0.85`
5. `self.min_diff = 1e-5`
6. `self.steps = 10`
7. `self.node_weight = None`

设置停用词。

```
1.     def set_stopwords(self, stopwords):
2.         for word in STOP_WORDS.union(set(stopwords)):
3.             lexeme = nlp.vocab[word]
4.             lexeme.is_stop = True
```

仅使用 candidate_POS 标签存储单词。

```
1.     def sentence_segment(self, doc, candidate_pos, lower):
2.         sentences = []
3.         for sent in doc.sents:
4.             selected_words = []
5.             for token in sent:
6.                 if token.pos_ in candidate_pos and token.is_stop is False
7.                     :
8.                         if lower is True:
9.                             selected_words.append(token.text.lower())
10.                        else:
11.                            selected_words.append(token.text)
12.            sentences.append(selected_words)
13.        return sentences
```

得到全部的分词。

```
1.     def get_vocab(self, sentences):
2.         vocab = OrderedDict()
3.         i = 0
4.         for sentence in sentences:
5.             for word in sentence:
6.                 if word not in vocab:
7.                     vocab[word] = i
8.                     i += 1
9.         return vocab
```

从窗口在句子中构建 token_pairs。

```
1.     def get_token_pairs(self, window_size, sentences):
2.         token_pairs = list()
3.         for sentence in sentences:
4.             for i, word in enumerate(sentence):
5.                 for j in range(i + 1, i + window_size):
6.                     if j >= len(sentence):
7.                         break
8.                     pair = (word, sentence[j])
9.                     if pair not in token_pairs:
10.                        token_pairs.append(pair)
11.        return token_pairs
```


对称化。

```
1. def symmetrize(self, a):
2.     return a + a.T - np.diag(a.diagonal())
```

获取创建归一化矩阵。

```
1. def get_matrix(self, vocab, token_pairs):
2.     vocab_size = len(vocab)
3.     g = np.zeros((vocab_size, vocab_size), dtype='float')
4.     for word1, word2 in token_pairs:
5.         i, j = vocab[word1], vocab[word2]
6.         g[i][j] = 1
7.     g = self.symmetrize(g)
8.     norm = np.sum(g, axis=0)
9.     g_norm = np.divide(g, norm, where=norm != 0)
10.    return g_norm
```

得到关键词。

```
1. def get_keywords(self, number=10):
2.     node_weight = OrderedDict(sorted(self.node_weight.items(), key=lambda t: t[1], reverse=True))
3.     for i, (key, value) in enumerate(node_weight.items()):
4.         print('• ' + str(i+1) + ' ' + key + ' : ' + str(value))
5.         if (i + 1) >= number:
6.             break
1.
```

将上面函数构成分析的流程。

```
1. def analyze(self, text,
2.             candidate_pos=['NOUN', 'PROPN'],
3.             window_size=4, lower=False, stopwords=list()):
4.     self.set_stopwords(stopwords)
5.     doc = nlp(text)
6.     sentences = self.sentence_segment(doc, candidate_pos, lower)
7.     vocab = self.get_vocab(sentences)
8.     token_pairs = self.get_token_pairs(window_size, sentences)
9.     g = self.get_matrix(vocab, token_pairs)
10.    pr = np.array([1] * len(vocab))
11.    previous_pr = 0
12.    for epoch in range(self.steps):
13.        pr = (1 - self.d) + self.d * np.dot(g, pr)
14.        if abs(previous_pr - sum(pr)) < self.min_diff:
15.            break
16.    else:
```

```

17.         previous_pr = sum(pr)
18.         node_weight = dict()
19.         for word, index in vocab.items():
20.             node_weight[word] = pr[index]
21.         self.node_weight = node_weight

```

函数执行入口。将目标文件打开并且读取内容。和算法 1 一样。不过使用的是字符串变量 wordlist。

```

1.  if __name__ == '__main__':
2.      with open("JobDataAnalyst.csv", "rt", encoding="utf-8") as file
3.          :
4.              reader = csv.reader(file)
5.              column = [row[3] for row in reader]
6.
7.              wordlist = ''
8.              test = []
9.              for i in range(0, len(column)):
10.                 token = word_tokenize(column[i])
11.                 content = ' '.join(token)
12.                 wordlist = wordlist + content

```

为了防止相同的关键词只是字母大小写有区别，把文本全部转化为小写。再利用 analyze 函数分析文本，并依次打印关键词字符串和权重。

```

1.         wordlist = wordlist.lower()
2.         tr4w = TextRank4Keyword()
3.         tr4w.analyze(wordlist, candidate_pos=['NOUN', 'PROPN'], window
4.             _size=4, lower=False)
5.         tr4w.get_keywords(20)

```

运行之后会发现因字符串太长而无法运行。这时我们修改一下配置。在 Web.config 的<configuration></configuration>节点下,添加以下配置即可。

```

1.  <system.web.extensions>
2.      <scripting>
3.          <webServices>
4.              <jsonSerialization maxJsonLength = "999999999">
5.                  </jsonSerialization>
6.              </webServices>
7.          </scripting>
8.      </system.web.extensions>

```

四、实验结果

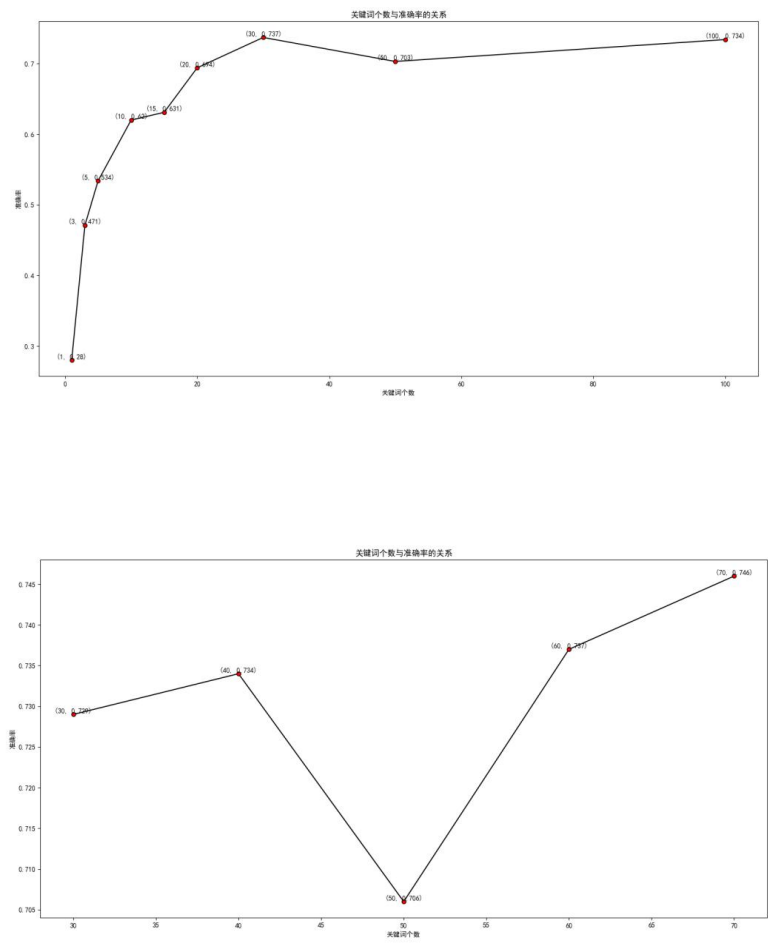
1. 基于 word2vec 实现文本分类

运行成功。

```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/worm2/main.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\raven\AppData\Local\Temp\jieba.cache
Loading model cost 0.493 seconds.
Prefix dict has been built successfully.
对训练集词频统计
100%|██████████| 7/7 [00:25<00:00, 3.61s/it]
0%|          | 0/7 [00:00<?, ?it/s]计算准确率
Keyword_num: 30
data/家居/54.txt
data/家居/55.txt
```

```
100%|██████████| 7/7 [00:09<00:00, 1.32s/it]
[0.28, 0.471, 0.534, 0.62, 0.631, 0.694, 0.737, 0.703, 0.734]
进程已结束,退出代码0
```

使用两个不同参数的结果。



2. 岗位关键词提取

【TF-IDF 算法】

运行成功，打印结果。

```
•1: data 0.2682500679287533
•2: experience 0.10626034155710774
•3: business 0.07899374786349535
•4: work 0.060290586219619365
•5: skills 0.05095577452049417
•6: analysis 0.04345438691556842
•7: team 0.0418669099559567
•8: management 0.041352656856364176
•9: analyst 0.039608668083833
•10: ability 0.03850190597818822
•11: years 0.034164292877277354
•12: information 0.030844006560343005
•13: knowledge 0.03055334176492114
•14: support 0.029972012174077415
•15: requirements 0.029580732641778757
•16: strong 0.029122376618228897
•17: reporting 0.028339817553631574
•18: job 0.02793735860612438
•19: analytics 0.027702590886745183
•20: required 0.02762433498028545
```

进程已结束,退出代码0

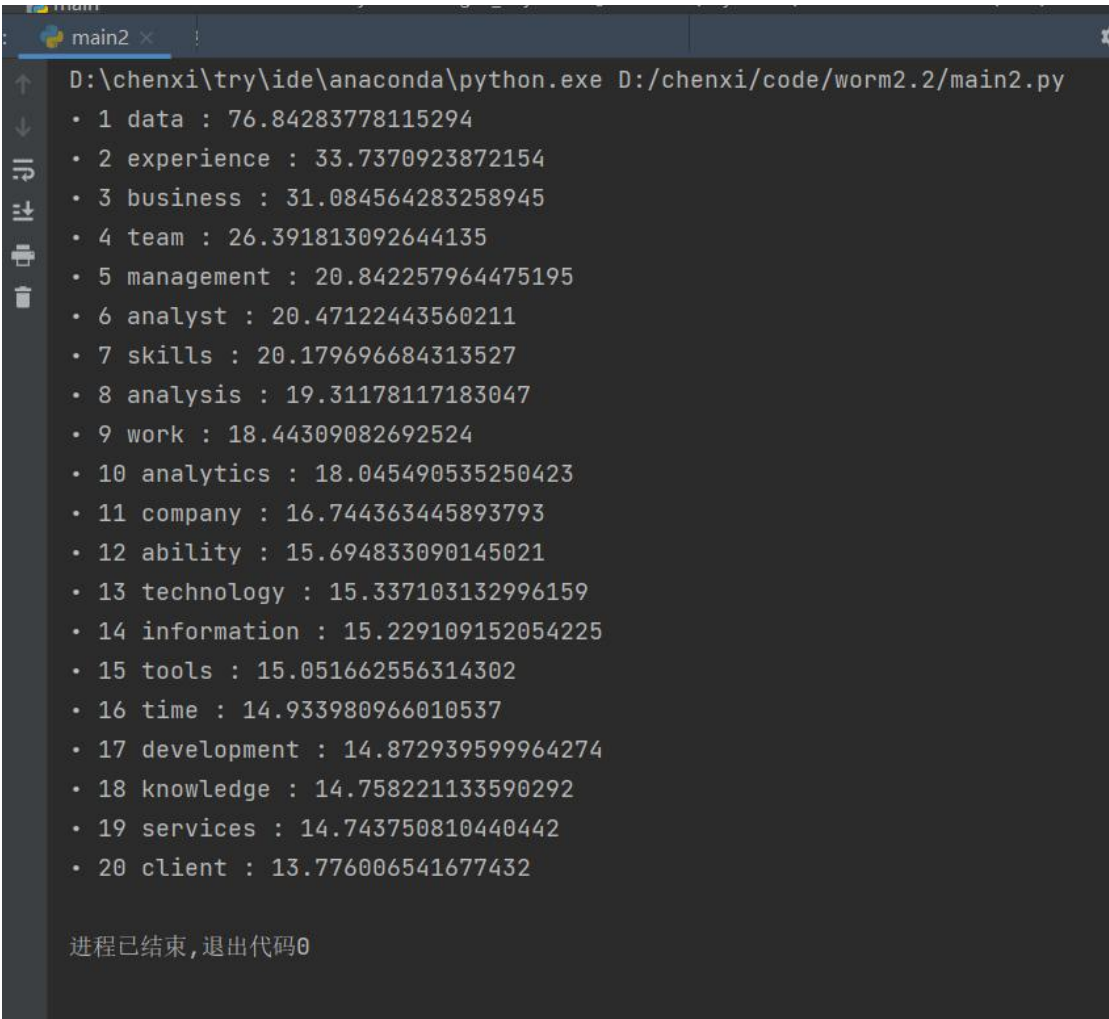
【TextRank 算法】

改小写之前。同一个单词只是大小写的区别却被分开分析。比如“data”和“Data”。

```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/worm2.2/main2.py
•data : 77.50516116266292
•Data : 42.726744538396424
•business : 32.08130571120845
•team : 29.982617079601177
•experience : 28.91916613461971
•Experience : 24.742188715509915
•analysis : 21.246197247057783
•Analyst : 20.999806764727744
•management : 20.75370505408988
•skills : 20.086565299244405
•work : 18.826124160326135
•tools : 16.43138176499348
•development : 16.328313945872694
•analytics : 15.550560607656676
•SQL : 15.162711051977391
•New : 14.731128312614635
•time : 14.68183307535529
•clients : 14.637213816364557
•teams : 14.287328494016897
•reporting : 14.130653464081302
•knowledge : 13.882854564409229
•role : 13.875538056650715
```

进程已结束,退出代码0

将所有字符改成小写之后，消除了这种情况。再加上序号。



```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/worm2.2/main2.py
• 1 data : 76.84283778115294
• 2 experience : 33.7370923872154
• 3 business : 31.084564283258945
• 4 team : 26.391813092644135
• 5 management : 20.842257964475195
• 6 analyst : 20.47122443560211
• 7 skills : 20.179696684313527
• 8 analysis : 19.31178117183047
• 9 work : 18.44309082692524
• 10 analytics : 18.045490535250423
• 11 company : 16.744363445893793
• 12 ability : 15.694833090145021
• 13 technology : 15.337103132996159
• 14 information : 15.229109152054225
• 15 tools : 15.051662556314302
• 16 time : 14.933980966010537
• 17 development : 14.872939599964274
• 18 knowledge : 14.758221133590292
• 19 services : 14.743750810440442
• 20 client : 13.776006541677432

进程已结束,退出代码0
```

【两算法优劣比较】

- (1) TF-IDF 算法花费的时间要小于 TextRank，效率比较高。
- (2) TF-IDF 算法会将不同大小写的同一个单词归为一个单词，如“data”和“Data”。但 TextRank 算法并不会。
- (3) 20 个关键词比较：

序号	TF-IDF 算法	TextRank 算法
1	data	data
2	experience	experience
3	business	business
4	work	team

序号	TF-IDF 算法	TextRank 算法
5	skills	management
6	analysis	analyst
7	team	skills
8	management	analysis
9	analyst	work
10	ability	analytics
11	years	company
12	information	ability
13	knowledge	technology
14	support	information
15	requirements	tools
16	strong	time
17	reporting	development
18	job	knowledge
19	analytics	services
20	required	client

可以看出，TF-IDF 生成关键词的质量要高于 TextRank 算法。

看排名前 20 的关键词。TF-IDF 算法中，有如，work, years, information, strong, job，不能很明显地体现岗位招聘需求。而这些词，TextRank 的排名要比 TF-IDF 要低，或者根本没有上榜。当然 company 这种岗位需求特征弱的词，在 TextRank 中有但是 TF-IDF 没有，不过综合来看还是 TextRank 算法输出的关键词质量更好。

（4）可能因为一些地域之类的关系，TF-IDF 的环境配置要比 TextRank 麻烦许多。TF-IDF 需要的工具包安装界面无法正常打开，好多人都有这个问题。需要查询 IP 地址并添加到配置文件中。而 TextRank 需要的工具包在 pycharm 即可顺利下载。

【招聘岗位需求分析】

通过关键词，我们可以分析出：

岗位需要经验丰富的人才，擅长数据信息分析和科技业务处理，擅长实用工具；
可以团队协作，最好技术能力强。擅长管理，善于学习，知识丰厚者优先。