

杭州电子科技大学

《自然语言处理》

**实验一报告**

|  |  |
| --- | --- |
| **任 务** | 文本信息检索系统 |
| **学 院** | 卓越学院 |
| **专 业** | 人工智能与智慧健康 |
| **学 号** | 20063217 |
| **学生姓名** | 姜习武 |

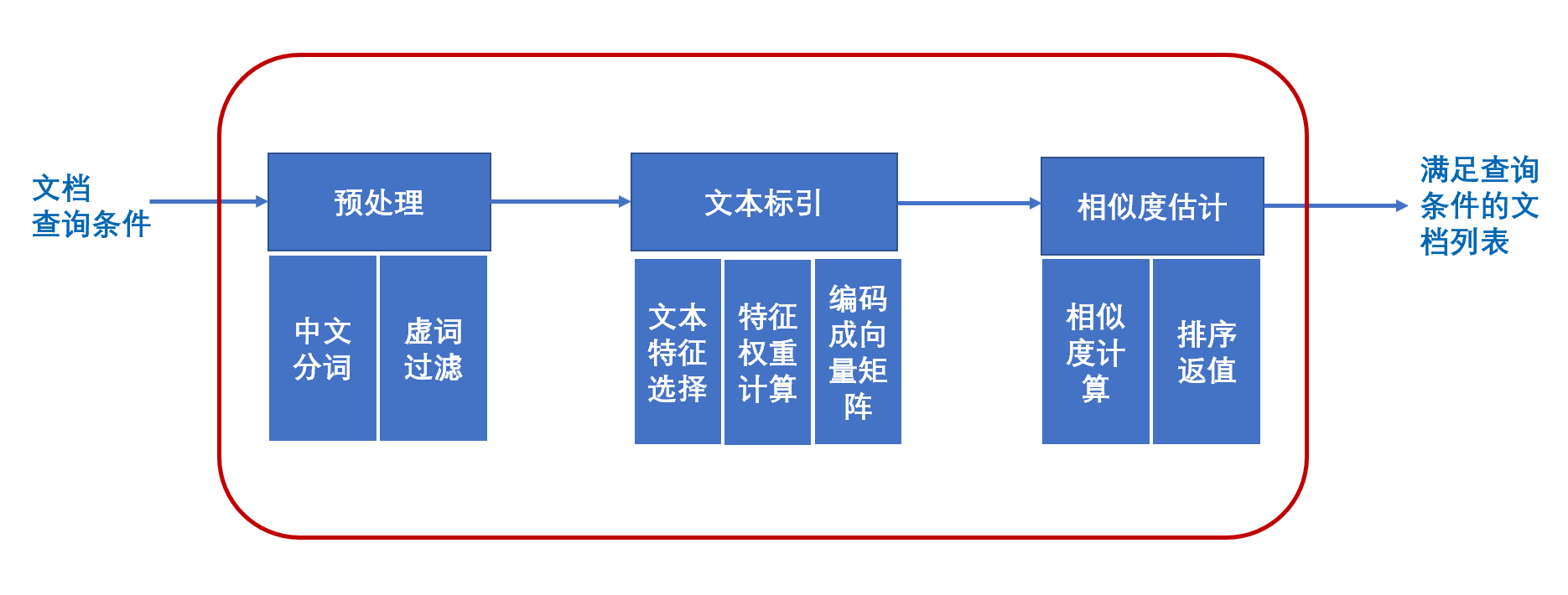
2023年 4月

# 实验背景介绍

## 1.实验目的

从给定的测试文件中按行对句子进行分词，然后对这些分词后的关键字与训练文档进行相似度匹配，然后对文本进行排序，从已有的若干篇文本中找出最相关的1篇或多篇文本。

## 2.实验流程



## 3.理论介绍

文本检索（text retrieve）的常用策略是：用一个ranking function根据搜索词对所有文本进行排序，选取前n个，就如百度搜索类似。

显然，ranking function（排序函数）是决定检索效果最重要的因素。根据课程内容，我选用了**TF-IDF**计算值作为计算特征权重计算。其仅用到了一些基础的统计和文本处理的方法，并不是特别高深。

TF-IDF（term frequency–inverse document frequency，词频-逆向文件频率）是一种用于信息检索（information retrieval）与文本挖掘（text mining）的常用加权技术。

TF-IDF是一种统计方法，用以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。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

TF-IDF的主要思想是：如果某个单词在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

### （1）TF是词频(Term Frequency)

词频（TF）表示词条（关键字）在文本中出现的频率。这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件。

公式： 即：

其中是该词在文件中出现的次数，分母则是文件中所有词汇出现的次数总和；

### （2） IDF是逆向文件频率(Inverse Document Frequency)

逆向文件频率 (IDF) ：某一特定词语的IDF，可以由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到。

如果包含词条t的文档越少, IDF越大，则说明词条具有很好的类别区分能力。

,分母+1是为了避免其为0

### （3）TF-IDF实际上是：TF \* IDF

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。



TF-IDF算法非常容易理解，并且很容易实现，但是其简单结构并没有考虑词语的语义信息，无法处理一词多义与一义多词的情况。

# 方案设计与实验说明

（1）首先我们构建停用词表与虚词过滤表：

(停用词表下载连接：<https://github.com/goto456/stopwords>)

1. # 构建停用词表

2. stop\_words = './cn\_stopwords.txt'

3. stopwords = codecs.open(stop\_words, 'r', encoding='utf8').readlines()

4. stopwords = [w.strip() for w in stopwords]

5. # jieba分词后的停用词性

6. # [标点符号、连词、助词、副词、介词、时语素、'的'、数词、方位词、代词]

7. stop\_flag = ['x', 'c', 'u', 'd', 'p', 't', 'uj', 'm', 'f', 'r']

8. # 进行分词和虚拟此过滤

（2）然后我们对每篇文章进行分词、并且过滤停用词

1. def tokenization(filename):

2.     result = []

3.     with open(filename, 'r', encoding='gb18030', errors='ignore') as f:

# TODO: 编码！

4.         text = f.read()

5.         words = pseg.cut(text)  # 分词

6.     for word, flag in words:

7.         if flag not in stop\_flag and word not in stopwords:

8.             result.append(word)  # 虚词过滤

9. return result

10.

11. # 测试集分词

12. def tokenization\_test(raw):

13.     result = []

14.     words = pseg.cut(raw)  # 分词

15.     for word, flag in words:

16.         if flag not in stop\_flag and word not in stopwords:

17.             result.append(word)  # 虚词过滤

18.     return result

（3）对目录下的所有训练集进行处理并构建成字典为后期做准备

1. corpus = []

2. rootname = 'C:\\Users\\kieran\\Desktop\\项目代码\\Natural\_language\_Process\\NLP实验1\\中文信息检索任务数据集\\训练集'

3. filenames = []  # 存放所有文件名

4. file\_dict = {}  # 存放所有子文件下的文件名

5.

6. for root, dirs, files in os.walk(rootname):

7.     filenames.append(files)

8.     file\_dict[root] = files

9.

10. print(file\_dict) # 得到子文件夹

11. # 遍历所有文件，拼接路径

12. for k, v in file\_dict.items():

13.     for i in v:

14.         file\_name = os.path.join(k, i)

15.         # print(file\_name)

16.         corpus.append(tokenization(file\_name))

17. dictionary = corpora.Dictionary(corpus)

从上述代码中的得到结果可知：

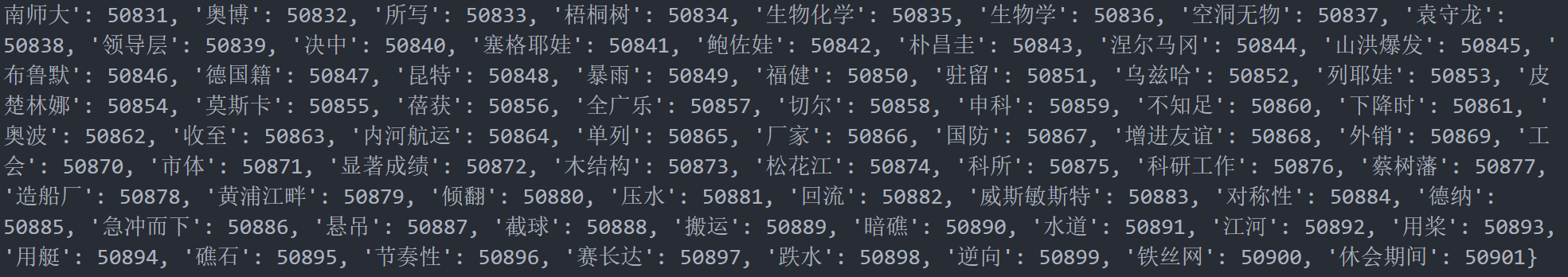
Building prefix dict from the default dictionary ...

Loading model from cache C:\Users\kieran\AppData\Local\Temp\jieba.cache

Loading model cost 0.571 seconds.

Prefix dict has been built successfully.

* 训练集文本个数：len(corpus)=4411
* 单词与编号之间的映射关系：dictionary.token2id



从上述结果中可以看出对所有文本进行分词整合后单词高达len(dictionary))=50902个！每一个单词对应一个key值。

（4）建立词袋模型，输出第一篇文本按词频排序的前5个词

1. #上面这些步骤，我们利用gensim.corpora.dictionary.Dictionary类为每个出现在语料库中的单词分配了一个独一无二的整数编号id,这个操作收集了单词计数及其他相关的统计信息。

2. # 列出1(0).txt文件中排名前5的words

3. doc\_vectors = [dictionary.doc2bow(text) for text in corpus]

4. vec1 = doc\_vectors[0]

5. vec1\_sorted = sorted(vec1, key=lambda x\_y:x\_y[1], reverse=True)

6. print(len(vec1\_sorted))

7. for term, freq in vec1\_sorted[:5]:

8.     print(dictionary[term])

此时文档1的排序后单词个数为：len(vec1\_sorted)=91

排名前五的单词分别为：日本 年薪 职棒 最高 选手

（5）建立TF-IDF的权重计算函数

1. class TF\_IDF(object):

2.     def \_\_init\_\_(self, corpus):

3.         self.f = []

4.         self.df = {}

5.         self.idf={} # 计算tf值

6.         self.tf = {}  # 计算idf值

7.         self.corpus = corpus

8.         self.corpus\_size = len(corpus)

9.

10.     def initialize(self):

11.         for document in corpus:

12.             frequencies = {}

13.             for word in document:

14.                 if word not in frequencies:

15.                     frequencies[word] = 0

16.                 frequencies[word] += 1

17.             for word in frequencies:

18.                 self.tf[word] = frequencies[word]/len(frequencies)

19.             self.f.append(frequencies)

20.             for word, freq in iteritems(frequencies):

21.                 if word not in self.df:

22.                     self.df[word] = 0

23.                 self.df[word] += 1  # 计算包含word的文档数

24.

25.         for word, freq in iteritems(self.df): # freq为包含了word的文档个数

26.             self.idf[word] = math.log(self.corpus\_size) - math.log(freq+1.0) # IDF

27.

28.     def get\_tfidf(self,document,index):

29.         tfidf = 0

30.         for word in document:

31.             idf = self.idf[word]

32.             tfidf += idf \* (self.f[index][word] / len(self.f[index]))

33.         return tfidf

34.

35.     def get\_tfidfs(self, document):

36.         tfidfs = []

37.         for index in range(self.corpus\_size):

38.             tfidf = self.get\_tfidf(document, index)

39.             tfidfs.append(tfidf)

40.         return tfidfs

41.

42. def tfidfs\_weigit(corpus):

43.     TF\_IDF\_Moel = TF\_IDF(corpus)

44.

45.     weights = []

46.     for doc in corpus:

47.         scores = TF\_IDF\_Moel.get\_tfidfs(doc)

48.         weights.append(scores)

49.

50.     return weights

（6）引入测试集进行测试

1. test\_path = './中文信息检索任务数据集/测试集.txt'

2.     with open(test\_path, 'r', encoding='gb18030', errors='ignore') as f:

3.         for line in f:

4.             temp = tokenization\_test(line.strip())

5.             query.append(temp)

6.     scores\_all=[]

7.     idx\_all=[]

8.     fname\_all=[]

9.     for sentence in query:

10.         scores = TfidfModel.get\_scores(sentence)

11.         # scores = TF\_IDF.get\_tfidfs(sentence)

12.         # scores.sort(reverse=True)

13.         scores\_all.append(scores)

14.         idx = scores.index(max(scores))

15.         idx\_all.append(idx)

16.         file = list(chain.from\_iterable(filenames))

17.         fname = file[idx]

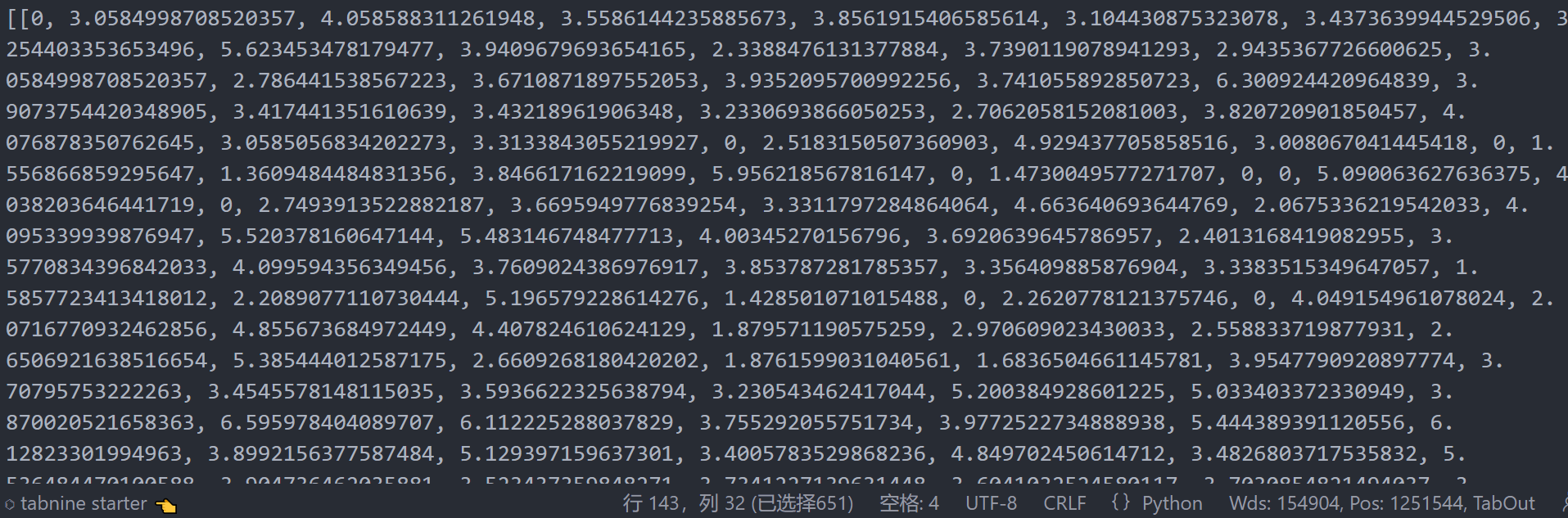
18.         fname\_all.append(fname)

最终结果突出显示为：

* 测试集分行结果：

[['中国', '女曲', '击败', '韩国', '圆梦'], ['女垒', '锦标赛', '会徽', '吉祥物', '丫丫', '南京', '亮相'], ['男子', '中国队', '双保险', '失灵', '加澳', '抢得', '冠亚军'], ['中国', '体操', '悉尼', '辉煌', '遗憾'], ['棒球', '联赛', '首回合', '沪', '闷热', '击溃', '猛虎'], ['老帅', '盛赞', '国奥', '小将', '具备', '亚洲', '顶级', '水平'], ['全国', '山地车', '冠军赛', '马艳萍', '获', '女子', '越野赛', '桂冠'], ['摩托罗拉', '世界', '名校', '赛艇', '对抗赛'], ['网球', '女单', '开赛', '塞莱斯', '首战', '轻取', '对手'], ['NBA', '湖人', '主帅', '杰克逊', '支持', '科比']]

* 由于数据量的关系，这里仅仅展示第一篇文章的权重计算结果：

第一篇文章与训练集4411篇文章的权重计算值大致如下：

* 通过节省计算量，此处我们选用匹配度最相关的一个文本，其对应的文档顺序下标idx\_all为：

[377, 3886, 2637, 1904, 21, 3330, 3060, 984, 2527, 3775]

对应的文档名字：

['11 (20).txt', '6 (113).txt', '23 (138).txt', '20 (84).txt', '1 (117).txt', '26 (72).txt', '25 (63).txt', '14 (88).txt', '22 (4).txt', '5 (162).txt']