华 北 水 利 水 电 大 学

自 然 语 言 处 理

实 验 报 告

20 22 ——20 23 学年

第 一 学期

实验报告序号：实验报告（一）

实 验 名 称：关键词提取与可视化的实现

学生专业班级：人工智能 2020185

学 生 姓 名：高树林

学　　号：202018526

专　　业：人 工 智 能

一、实验目的

1.掌握关键词抽取的基本流程。

2.掌握 TF-IDF 和 TextRank 等两种关键词抽取方法。

3.能够基于 WordCloud 工具或 Pyecharts 工具对抽取到的关键词进行可视化展示。

二、实验内容

给定待抽取关键词的文档的集合 corpus4keywords.txt 和停用词的集合 stopwords.txt，采用 TF-IDF 和 TextRank 等两种关键词抽取方法抽取各个文档中的关键词，并对最后 3 个文档的关键词抽取结果进行可视化。

三、实验要求

数据选取要求：从数据集中随机选取 100 篇文档；

四、问题分析

4.1 采用 TF-IDF 进行关键词抽取方法的实现流程分析

画出流程图如下图所示。

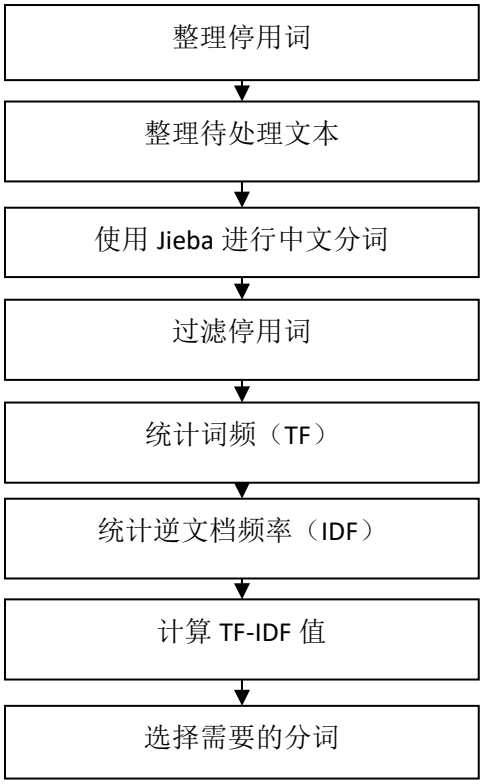

图 1 TF-IDF 算法流程图

流程介绍：首先整理出停用词，以便后期可以直接调用，最好将其放在一个列表里

面，这样后期就可以用 in 关键字来判断当前词是否是停用词，进而考虑要不要将其加入到其对应字典里。

其次再整理文档，由于给定的文档中每一行就是一个自然段，因此可以用 readlines 方法来获取其每一行的内容，但是需要注意的是每一行有特殊空格（Tab 键、换行键等等），需要将其处理掉。

之后对预处理后的文档进行分词，分词工具是 jieba，利用 jieba 将词分出来之后将每句话中的分词存在一个列表里，在加入之前需要判断当前词是否在停用词里面，如果不在，可以加入，如果在，就不能加入。

统计词频的时候需要遍历现有文档的所有元素，此时用字典做存储变量，这样做的好处是直接可以用键来匹配当前分词，方便加入和以后的查找引用。其原理是：如果当前分词不在字典的键里面说明该词是第一次被遍历，它是第一个，因此将其对应的置为 1，若当前分词在字典里，则当前分词作为键对应的值加 1。

计算逆文档频率需要计算每个词在出现的文档数，因此需要针对每一个文档去掉重复的词，之后遍历所有文档，其原理和统计词频类似。

根据 TF 值和 IDF 值来计算 TF-IDF 的值，计算之后将其排序，最后获取前 100 个文档的分词和他们的 TF-IDF 值。并以此生成词云图。

4.2 采用 TextRank 算法进行关键词抽取方法的实现流程分析

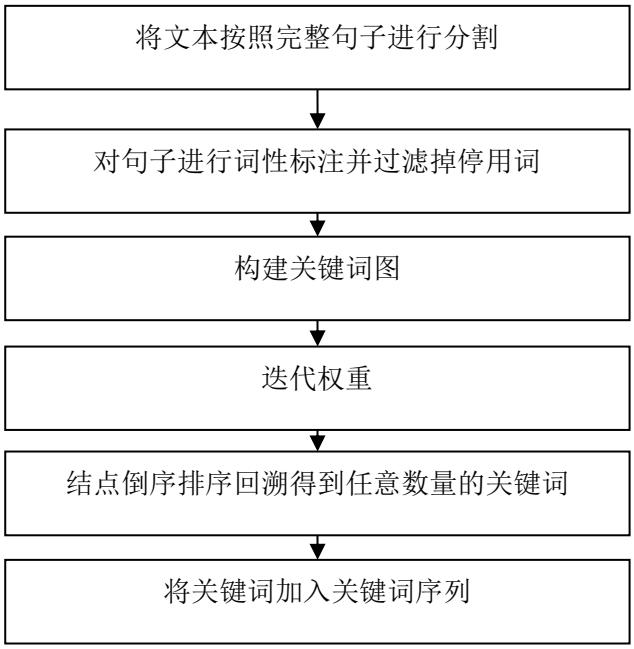

图 2　TextRand 算法流程图

关键词抽取的任务就是从一段给定的文本中自动抽取出若干有意义的词语或词组。

TextRank 算法是利用局部词汇之间关系（共现窗口）对后续关键词进行排序，直接从文本本身抽取。其主要步骤如下：

首先，需要对句子进行语义分割，这个一般用到 jieba 库中的 cut 方法。

其次，对于每个在文档中的句子，需要对其进行词性标注处理，此时也需要过滤掉停用词，只保留指定词性的词语，如名词、动词、形容词、副词等等。

然后以上一步中的关键词为候选关键词组成结点集，采用共现关系构造任意两点之间的边，两个结点之间存在边仅当他们对应的词汇在长度为 K 的窗口中出现，K 表示最多共线词语数量。

根据上述公式迭代传播点的权重，直至收敛或者达到要求。之后对节点进行排序，得到最重要的 10 个词语，作为关键词并将它们加到关键词列表中。关键词列表中的元素为当前分词和该分词的权重值。为了防止运算量过大，一般会将权重归一化处理，这样不仅快，还简化了运算。

五、关键代码

5.1 采用 TF-IDF 进行关键词抽取方法的步骤描述和核心

步骤 1：先计算出该当前的 TF 值，TF 值的计算方法为：

TF（词频） = 某词在文章中出现的次数 / 文章中的总词数

因此需要首先统计文章的所有词数（已过滤掉停用词），其次对每个词进行遍历，若该词不在字典里面，就把该词对应的值设为 1，如果在，就将其对应的值加 1。当遍历结束后，各个词的频率也就全都成为该词作为键对应的值了。当引用该值的时候只需要使用字典名[键名]就能引用到该词在文章中出现的次数了，文章的总词数可以用遍历。核心代码为：

```
def tf(filter_sent):
    # 统计 TF 值
    tf_dict = {}
    for word in filter_sent:
        if word not in tf_dict:
            tf_dict[word] = 1#如果当前词不在字典中，说明是第一次出现该词
        else:
            tf_dict[word] += 1#不是第一次出现该词就直接加 1
    for word in tf_dict:
        tf_dict[word] = tf_dict[word] / len(filter_sent)#求词频
    return tf_dict
```

步骤 2：计算出该当前的 IDF 值，IDF 值的计算公式不唯一，但是一般方法为：

IDF = log(语料库出现的文章总数 / 包含该词的文章总数+1)

因此需要计算语料库中出现该词的所有文章数，和所有的文章数，所有文章数比较好求，只需要判断 documents 的元素个数即可，也就是文本的行数。对于出现该词的文章数可以通过遍历，遍历之前要先把每个列表元素里面的列表转为集合，达到去重的效果，这样一来，每个元素中最多只包含一个当前分词。按照步骤 1 中的统计方法就能获取包含每个词的文章数，也即包含该词的集合总数。当引用时，仍只需要使用字典名[键名]就能引用到出现该词的文章数了。核心代码为：

```
def idf(documents):
    # 统计 IDF 值
    idf_dict = {}
    for doc in documents:
        for word in set(doc): # 集合去重
            if word not in idf_dict:
                idf_dict[word] = 1
            else:
                idf_dict[word] += 1
    return idf_dict
```

步骤 3：计算 IF-IDF 的值，其计算公式为：

TF-IDF = TF * IDF

值越大表示该词在文章中越重要。其实现时利用 for 循环，将键为当前分词的 tf_dir 的值和键为当前分词 log(文本长度/(idf_dir)) + 1 的值相运算并赋给 tf_idf_dir。这样一来 tf_idf_dir[当前分词]的结果就是该分词所对应的 tf-idf 值。根据这个值排序，可以取前任意数目的词。其代码如下：

```
for filter_sent in documents:
    tf_idf_dict = {}
    tf_dict=tf(filter_sent)
    for word in tf_dict:
        tf_idf_dict[word] = tf_dict[word] * math.log(len(documents) / (idf_dict[word]))
    # 提取前 10 个关键词
    topk = 10
    sort_kwords=sorted(tf_idf_dict.items(), key=operator.itemgetter(1),reverse=True)
    kword_list=[kword for kword,value in sort_kwords[:topk]]
```

步骤 4：可视化实现。利用 pyecharts 的 WordCould 模块需要传入的数据为列表类型，其中列表中的元素是一个列表，每个二级列表代表一段文本中除停用词以外的所有词。二级列表中的元素是每个除停用词外的分词以及他的 tf-idf 值，用这个值的大小来衡量该分词的重要性，反应在词云图上就是词云图上的字体大小。其中 pyecharts 可以自定

义图片，利用其内置的 mask_image 参数来传入。但是要求是被词覆盖的地方必须是黑色的，这个我目前还未搞懂是什么原因，可能是 mask 参数传入的就是要“遮住”黑色部分的内容。详细代码如下：

```
data = tf_idf(documents)
(
    WordCloud() .add(series_name='TF-IDF',
            data_pair=data[-3],   # 要将哪一句话生成词云
            word_size_range=[16, 66],  # 字体大小
            mask_image='.//img.png'  # 在这里设置背景的路径
            ) .set_global_opts(
           title_opts=options.TitleOpts(
           pos_left='40%',
             title='TF-IDF',  # 词云图的名字
             title_textstyle_opts=options.TextStyleOpts(font_size=33) ),
         tooltip_opts=options.TooltipOpts(is_show=True), )
         .render('TF-IDF_Vision.html'))
```

5.2 采用 TextRank 算法进行关键词抽取方法的步骤和核心代码

利用传统的算法无法直接对词性进行标注，因此需要借助 jieba 库中的 jieba.analyse.extract_tags 库来直接对句子进行分词并解析。其原理是调用接口，从本地 jieba 库中的模型进行识别并返回结果。利用封装提供的参数可以选择句子、权重最高的前 K 个词，需要哪些词，是否需要传入每个词的权重等等都能定义。具体代码如下：

```
def TextRand():
    keywords = []
    for i in text:
        keywords.append(jieba.analyse.extract_tags(sentence=i,topK=10,allowPOS=('n','ns','nr','nt','nz'),withWeight=True))
    return keywords
```

其可视化的实现和利用 TF-IDF 的代码类似，就不在此赘述了。代码如下：

六．实验结果与分析

6.1 利用 TF-IDF 算法制作取最后三个文本的词云图和底图如下图 3 所示：

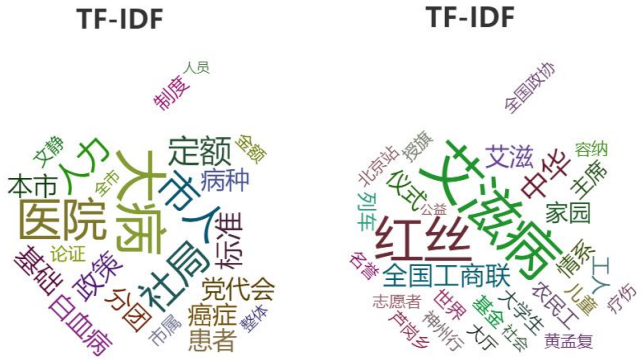

图 3 利用 TF-IDF 算法得出的词云图和底图

TF-IDF 算法是一种统计方法，用以评估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（TF），但同时会随着它在语料库中出现的频率成反比下降（IDF）。

如果某个词或短语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。TF-IDF 实际上就是 TF*IDF，其中 TF 表示词条在文章中出现的频率。IDF 的主要思想就是，如果包含某个词 Word 的文档越少，则这个词的区分度就越大，也就是 IDF 越大。对于如何获取一篇文章的关键词，我们可以计算这边文章出现的所有名词的 TF-IDF，TF-IDF 越大，则说明这个名词对这篇文章的区分度就越高，取 TF-IDF 值较大的几个词，就可以当做这篇文章的关键词。

6.2 利用 TextRand 算法制作取最后三个文本的词云图和底图如下图所示：

图 4 利用 TextRand 算法得出的词云图和底图

TextRank 算法是一种用于文本的基于图的排序算法。其基本思想来源于谷歌的 PageRank 算法, 通过把文本分割成若干组成单元(单词、句子)并建立图模型, 利用投票机制对文本中的重要成分进行排序, 仅利用单篇文档本身的信息即可实现关键词提取、文摘。和 LDA、HMM 等模型不同, TextRank 不需要事先对多篇文档进行学习训练, 因其简洁有效而得到广泛应用。

总结：

TF-IDF 算法和 TextRand 算法操作简单，对于简单文本和短文本的处理具有较高正确率和简洁高效的优点。但是 TF-IDF 算法和 TextRand 算法均严重依赖分词的结果（代码中都必须先使用 jieba 库处理这一点就能看出其对分词的严重依赖性），因此其结果收到分词结果的严重影响。如分词结果出现细微偏差，将会造成关键词抽取的准确率和召回率有极大的偏差。此外，TextRand 算法虽然考虑到了词之间的关系，但是还是会倾向于将高频词作为关键词，这一点比不上 TF-IDF 算法。此外，TextRand 算法需要每次更新迭代，因此提取的速度慢。从本质上来讲这两种方法都是基于词频统计，对于某些主题词或者关键词隐藏或者极少出现的文章，这两种算法就不再适用。