文本检索实验报告

姓名：吴悦欣

一、 概述

a) 任务目标

实现英文文本检索：给定一个英文文本数据集，用户在图形界面输入检索关键词，服务器进行计算，返回给用户排序后的检索结果。

b) 实现方案

从文本数据集构造检索词典，通过 tf-idf 构造文章和词语的特征向量，与检索词以及其相似词进行匹配得到初步排序的结果，之后通过 HITS 算法扩大和优化排序结果。

二、 具体实现与评价

a) 语料预处理

需要处理文章的标题和主体两部分。首先将文本中非字母的特殊字符去除，之后使用 lemmatizer 进行词形还原，然后将文本中的停用词去除，将低频词汇（这里选择 threshold 为 10）删去后得到预处理的结果。

需要注意的是这里的词形还原不能直接调用 lemmatizer.lemmatize，因为这一函数默认所有的词语都是名词；而应该对每篇文章使用 pos_tag 函数统一处理赋予每个单词初始分类后，对每个单词单独进行处理。

```
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return None
```

而在观察最终形成的 vocab.txt 文件的时候，可以明显的发现有一部分词语显然还没有还原（是过去式），经过仔细思考我的结论是因为我一开始将对停用词的处理放在了还原前面，导致语句的结构发生很大的变化，词语的词性更难被确定，因此在最终的实现中，我交换了二者的位置，词形的还原的情况比较好。

b) 其他预处理

使用预处理完毕的语料，得到词典，统计每个文章和标题中包含单词的数量。计算 tf-idf，假设 tf-idf 返回的 weight 矩阵每一行是一篇文章包含的每个单词的数量，那么可以根据列向量作为单词的特征向量，行向量作为文章的特征向量，使用 pca 降维进一步提取二者的特征，之后使用余弦进行相似度的计算，得到相关性矩阵。对于单词可以进一步得到相似词，以供后续排序改进使用。

在这里对 tf 的计算中，我最终还对每一列除了一个文章的总词数，因为有些文章内容很长，那么它们的 tf 值自然会很大。但同时考虑到文章长，其内容的相关性不一定会被削弱（因为可以覆盖的内容更多，相关词汇出现更多也可以表明其与检索词相关），我还对比了在对长度取 log 对数除、直接除、不除的结果，通过聚类的评价方式，最终还是直接除以对数的原始长度的聚类效果更好（即文章向量更好），此时 pca 降维采用的维度是 1600。以上数值对比如下：

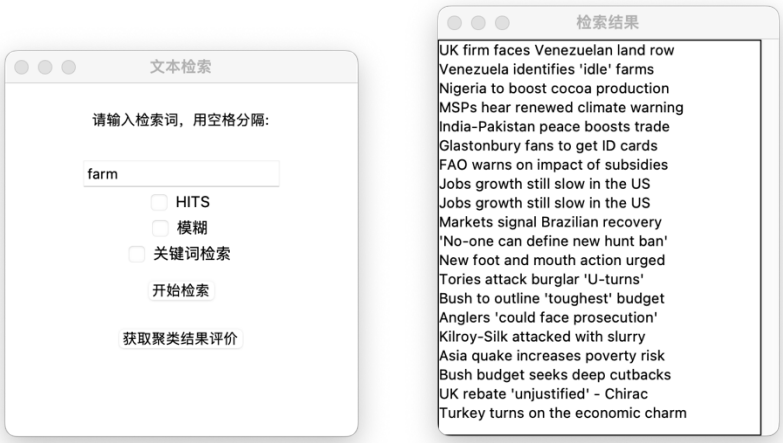
	Purity	F	NMI
/log(len)	0.64	0.91	0.56
/len	0.51	0.82	0.45
不处理的 tf	0.37	0.72	0.29

同时在调整 pca 降维的维度时发现，维度和聚类效果指标并不存在正相关、负相关性，并不那么稳定。

c) 初始检索排序

对于初始检索结果的计算，最初的想法是根据关键词，查看每篇文章内容中出现该关键词的次数、标题出现关键词的次数加权进行打分，而词频信息在进行 tf-idf 计算中的 tf 值直接就可以表示词频。又进一步考虑到 tf-idf 中 idf 的使用本身就对词语自身的特殊性、代表性进行了考量，综合上述想法，我认为直接使用 tf-idf 中的每个文章的向量，与查询词的 one-hot 编码进行点乘后就可以得到较好的评分。因此初始排序的结果就是对检索词进行 one-hot 编码和 tf-idf、每个文章的特征向量点乘后加和作为初始分数，取排名最前面的 20 个分数非 0 的结果返回给客户端。

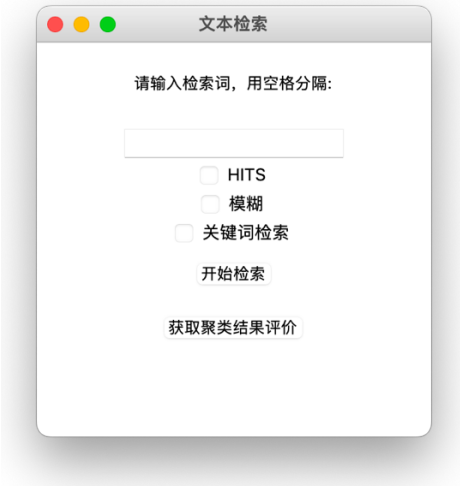
实际上初始检索排序返回的结果就很好了。



d) 对交互界面的调整

由于增加了可以使用 HITS 算法、模糊匹配、关键词检索等方式，我在交互界面增加了 checkbutton 对象，可以自行勾选并进行组合，而查询的类型会在客户端和服务端的通信中作为列表的第一项发送。

同时为了显示的给出对文章向量的评估——即聚类结果的评价。增设了获取评价指标的按钮。将会显示三个指标，分别是 Purity、F-score 和 NMI。



e) C/S 通信

客户端和服务器的通信，通过 socket 编程就可以实现，这里也实现了多线程（即每来一个 client 就新建一个专门的套接字为它服务，新建一个线程）。

其中需要关注的是 list 类型的数据的传输需要 json 编码。由于直接传送文章题目和内容给查询的客户端存在内容过长容易出错的问题，所以最终实现的是：客户端给服务器发送形如[type_num, 'word1', 'word2',…]的列表，服务器返回给客户端[num1,num2…]的文章索引列表（在查询聚类评价指标的时候返回[('Purity', purity)]…)形式的列表）。

```
def encode_data(data):
    return bytes((json.dumps(data)).encode('utf-8'))

def decode_data(data):
    return json.loads(bytes(data).decode('utf-8'))
```

f) 聚类结果评价

聚类评价指标在 cluster 函数中，这里采用了纯度、F 值、NMI 标准化互信息三种指标，并且对于前面两个指标，为了结果的准确性，采用了 KM 匹配，为聚类结果和原先的 label 做好了最大匹配。

三类指标分别衡量的是：Purity-每个类别中分类正确的比例，F 值-类内的相关性和类间的不相关性，NMI-直接度量聚类结果和原本 label 分类的相近程度。该指标数值可以通过点击客户端的对应按钮获取，值与在 pca 中保留的维度有关（因为使用的降维后的向量进行计算和后续作为特征值使用）。

g) HITS 算法

HITS 算法的目的在于通过文章之间的相关性，使得相关的文章之间形成“相互推荐”的机制，从而使得查询结果的泛化性更好。

在这里，HITS 算法中的邻接矩阵，我使用了降维后的初筛中分数为正数的文章特征向量的余弦相似度的矩阵，通过不断迭代使其收敛（其实等价于迭代对邻接矩阵进行矩阵乘法，得到邻接矩阵的特征矩阵，再用它点乘最初排序打分结果）。为了防止迭代时间太长，统一设置了迭代次数是 1000。

最终使用 HITS 的时候是直接通过 HITS 得到的向量但是实际运行效果并不好：一个在于重复的结果变多多（原本的数据集中存在重复的文本，在不断的迭代到收敛的过程中，这些相关性强的文本必然会收敛到基本一致的值）；第二点是返回的结果顺序并不理想，我认为这一点是一

些文章与该检索词相关的同时，和其他内容的相关性也比较强，在迭代的过程中不断引入了很多无关内容导致的。

h) 模糊匹配

模糊匹配是将检索词的相似词纳入考量，而相似词的筛选，在这里提前进行了计算和存储，匹配的时候直接将检索词的相似词都作为检索词的一部分，乘上一定的削弱的系数进行计算。

模糊匹配最终的影响不太明显，因为多数单词的相关性来源于：它们常常出现在一篇文章中，因此最终的检索结果没有太大的变化

i) 关键词检索

在每次运行服务器端服务的程序之前，服务器会为每一篇文章生成至少五个关键词，其中包含所有标题中的词——因为标题中的词通常是对文章内容的概括，因此更加关键和重要。剩余关键词的筛选则是根据文章特征向量这一词语所占的比重。在关键词与检索词匹配的时候，我们会希望直接返回这一文章，因为可以直接的认为这一文章和检索是直接相关的——需要在打分的时候给这一部分加上比较大的权重。

不过实际上在运行的时候，关键词检索和最初的打分基本差不多，因为关键词的筛选要求了其在 tf-idf 作为特征的向量中的值足够大，那么在最初的计算中，它自然的也会带给文章比较高的分数。

j) 对结果的简单评价

在最后增加了对返回结果中同一类的最大占比，作为一个检索结果

一致性的简单评价。在客户端的界面下方显示。

三、 加分项实现情况

在本次实验中，再加分项上主要实现了关键词检索，并且在客户端检索上增加了更多的选择，虽然每一个的结果并不那么满意，但都得到了初步的实现和一些有益的结论。

四、 提交文件结构

a) packages.py：import 一些必要的包，定义了 socket 传输数据的转换函数和聚类评价指标用到的函数/KM 算法
b) pre_process.py：包含了在预处理阶段（语料处理）需要调用的函数
c) server.py：服务器运行使用的文件，不需要参数，包含了大多数的计算过程
d) client.py：客户端运行使用的文件，不需要参数

五、 总结

在本次实验中，从语料的预处理、到中间的特征向量的构造和评价指标的构造，让我对自然语言处理整体的流程有了更具体深刻的体会。从结果上和过程中都可以发现，这一实验的结果非常依赖于语料库的质量（比如语料库里有很多关于球赛的新闻）。如果有更充足的时间，神经网络的尝试和传统方法的对比会是一个很好的方向！