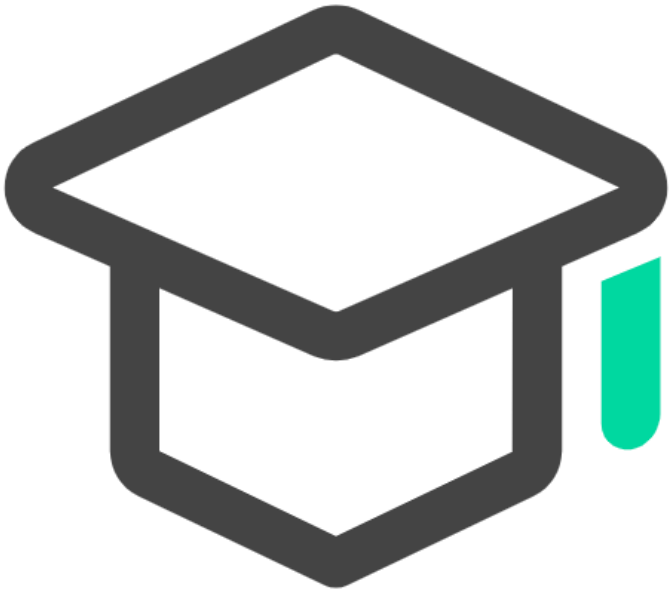

信息与知识获取作业二

于海鑫

2020/05/30

一、 作业要求

基本要求：自己动手设计实现一个信息检索系统，中、英文皆可，数据源可以自选，数据通过开源的网络爬虫获取，规模不低于 100 篇文档，进行本地存储。中文可以分词（可用开源代码），也可以不分词，直接使用字作为基本单元。英文可以直接通过空格分隔。构建基本的倒排索引文件。实现基本的向量空间检索模型的匹配算法。用户查询输入可以是自然语言字串，查询结果输出按相关度从大到小排序，列出相关度、题目、主要匹配内容、URL、日期等信息。最好能对检索结果的准确率进行人工评价。界面不做强制要求，可以是命令行，也可以是可操作的界面。提交作业报告和源代码。

扩展要求：鼓励有兴趣和有能力的同学积极尝试多媒体信息检索以及优化各模块算法，也可关注各类相关竞赛。自主开展相关文献调研与分析，完成算法评估、优化、论证创新点的过程。

二、 系统实现

A. 数据集

为了方便实现，我们选择了现有的数据集，而不是自己手动从网络中爬取数据作为被检索的文档。经过多方比较，我们选择了使用比较广泛的 BBC 数据集。其包含了 2004 年 BBC 发表的一些关于运动的新闻，我们选择了其中的一部分作为数据源，共 265 篇。

B. 构建空间向量

我们使用 sklearn 来构建文档的空间向量。sklearn 使用词袋（Bag of Words）和 TF-IDF 模型来表示文本数据，这两个模型都是 One-Hot 表示的应用，其中，词袋模型对应的就是文档向量。当前实现为了方便，我们使用词袋模型来构建空间向量，也就是说，我们只关注文档中是否出现给定的单词和单词出现频率，而舍弃文本的结构、单词出现的顺序和位置。

一般来说，对于一个文本语料库，构建词袋模型有四个步骤：

1. 文本分词：把每个文档中的文本进行分词

2. 构建词汇表：把文本分词得到的单词构建为一个词汇表，包含文本语料库中的所有单词，并对单词进行编号，假设词汇表有 n 个单词，单词编号从 0 开始，到 n-1 结束，可以把单词编号看作是单词的索引，通过单词编号可以唯一定位到该单词。
3. 词向量表示：每个单词都表示为一个 n 列的向量，在单词编号（词汇索引）位置上的列值为 1，其他列的值为 0
4. 统计频次：统计每个文档中每个单词出现的频次。

很幸运，我们使用 sklearn 即可快速构建模型，其核心代码如下：

```
from sklearn import feature_extraction
from sklearn.feature_extraction.text import CountVectorizer

def get_bag(texts):
    bag = CountVectorizer(token_pattern='\\b[A-Za-z]+\\b')
    count = bag.fit_transform(texts)
    return (bag, count)
```

CountVectorizer 即为 sklearn 提供的构建词向量模型的类别，使用它可以很方便的构建出我们的模型。

C. 构建倒排索引

所谓倒排索引，就是按照索引去反向查找文件的过程。其思路为从单词角度看文档，标识每个单词分别在那些文档中出现(文档 ID)，以及在各自的文档中每个单词分别出现了多少次（词频）及其出现位置（相对于该文档首部的偏移量）。

创建倒排索引，分为以下几步：

- 创建文档列表
- 创建倒排索引列表：对文档中数据进行分词，得到词条。对词条进行编号，以词条创建索引。然后记录下包含该词条的所有文档编号（及其它信息）。

核心代码如下：

```
from collections import defaultdict
```

```
def generate_inverse_index(text_list, bag, array):
    result = defaultdict(list)
    words = bag.get_feature_names()
    for index, value in enumerate(text_list):
        for i, word in enumerate(words):
            if array[index][i] != 0:
                position_list = [m.span() for m in re.findi
ter(
                    r'\b' + word + r'\b', value)]
                result[word].append((index, array[index][i]
, position_list))
    return result
```

D. 评分机制

直接使用相似度作为评分标准是可行的，但是我们可能会因此忽略掉真正的信息。例如，当我们使用“the best”去搜索时，我们很可能是关注于 best 而非最常见的冠词 the。此时显然把这两个单词的权重设置为一致的就不合理。因此，我们引入新的基于 Apache Lucene 的评分公式的评分机制。即使用 TF-IDF 来控制词汇的权重。具体公式为：

Lucene 实际评分公式

现在让我们看看 Lucene 实际使用的评分公式：

$$score(q, d) = coord(q, d) * queryNorm(q) * \sum_{t \text{ in } q} (tf(t \text{ in } d) * idf(t)^2 * boost(t) * norm(t, d))$$

也许你已经看到了，得分公式是一个关于查询 q 和文档 d 的函数，有两个因子 coord 和 queryNorm 并不直接依赖查询词项，而是与查询词项的一个求和公式相乘。

求和公式中每个加数由以下因子连乘所得：词频、逆文档频率、词项权重、范数。范数就是之前提到的长度范数。

这个公式听起来很复杂。请别担心，你并不用记住所有的细节，而只需意识到哪些因素是跟评分有关即可。从前面的公式我们可以导出一些基本规则：

- 越多罕见的词项被匹配上，文档得分越高。
- 文档字段越短（包含更少的词项），文档得分越高。
- 权重越高（不论是索引期还是查询期赋予的权重值），文档得分越高。

正如你所见，Lucene 将最高得分赋予同时满足以下条件的文档：包含多个罕见词项，词项所在字段较短（该字段索引了较少的词项）。该公式更“喜欢”包含罕见词项的文档。

我们使用的为简化的版本。

三、 结果展示

A. 搜索出单个项目

```
C:\Users\name1\Anaconda3\python.exe D:/PycharmProjects/INFO1/main.py
Loading data...
Vectorizing...
Generating index...
Done. Type to search now.
> jump
file: 254.txt
head: Worthington praises classy Villa
freq: 1
rank: 100.0
similarity: 0.03857583749052298
> ...hink it comes down to the quality," he said. "The jump from the first division into this level is a ...

n to next, q to quit, g to recommend, d to not recommend
```

B. 搜索出多个项目，通过按键进入下一个

```
Loading data...
Vectorizing...
Generating index...
Done. Type to search now.
> football
file: 224.txt
head: 2004 - A year to remember
freq: 7
rank: 3.553299492385787
similarity: 0.0935163889904661
> ...2004 - A year to remember

Club football in South America continues to suffer from...
> ... most decisive equalising goals in the history of football.

Argentina thought they had the game won...
> ... they amazingly failed to qualify for the Olympic football tournament. The gold medal is the only ti...
> ...the real highlights of the year in South American football could only come from the World Cup qualif...
> ...gal. For 90 minutes Brazil was the capital of the football world. Ronaldo obviously felt the sense o...
> ... involving all of the fundamentals of well-played football. The ball moved quickly around the field,...
> ...ualifiers scheduled for next year, South American football should come up with much more to celebrat...

n to next, q to quit, g to recommend, d to not recommend
> n
file: 055.txt
head: Legendary Dutch boss Michels dies
```

四、 可行的优化与展望

实际上，我们当前的实现是非常简单，甚至可以说得上事简陋的。这主要是因为这学期的作业量实在太多，没法将时间花在打磨实现上。下面我将简要列出可能的优化方式：

1. 引入单词词典，加速单词的查找。当前我们使用的为 Python 内部的字典来实现索引的查找，这对于小型的数据可行，但是对于实际应用的极大的数据集来说，几乎是不可

能实现的，因此我们需要引入一些持久化的结构来保证不受内存大小限制。当数据存放在外面时我们就应该注意到应该使用一些词典之类的方式来加速访问。

2. 允许动态更新数据集，正常的搜索引擎实际上是可以加入新的文档的，我们当前的每次启动时生成索引的方式显然不合理，应该加入动态更新的机制，例如 Lucene 的实现效果。