# 简易信息检索系统实现

## ——通过向量空间模型实现查询排序

刘子晨 1610914

Abstract

本实验在《简易信息检索系统实现——通过倒排索引实现布尔检索》[2]的基础上进行。

在上次实验，我们已经实现为存有一定量歌曲的数据库建立倒排索引并实现布尔检索，在本次实验中，我们需要利用已经建立好的倒排索引，为该数据库中的每首歌曲建立向量空间模型，并依据向量空间模型进行检索和检索内容的排序。

本实验数据库采用 MySQL，后端使用 Python 的 Web 框架 Bottle，前端使用 Vue.js 框架进行展示。

线上地址为：http://47.95.200.233/search-homework/

你所需要的源码、介绍、展示、实验数据，均可在线上地址进行查看和下载。

## 1. 引言

向量空间模型是一个把文本文件表示为标识符（比如索引）向量的代数模型。它应用于信息过滤、信息检索、索引以及相关排序。SMART 是第一个使用这个模型的信息检索系统。[1]

在本实验之前，我们已经为数据库中的数据建立了倒排索引，倒排索引表为每个出现的词项创建了 ID 和出现歌曲目录，我们将在此基础上，为数据库中的歌曲建立权重矩阵；查询内容也将用该倒排索引表中的词典建立向量矩阵。

由于数据库中每条数据需要检索的只有两个属性，所以分别对其建立权重矩阵，通过向量空间模型的方法为其计算权重并进行排序。

与倒排索引不同，在向量空间模型中，每个歌曲的歌名与歌词具有更为紧密的关系，每条数据具有两个属性这个特点使得我们需要对不同的属性分配不同的权重，以获得更好的查询排序，从而使用户获取更好的查询体验。

## 2. 实验环境与数据集介绍

实验环境与数据集同《简易信息检索系统实现——通过倒排索引实现布尔检索》[2]。

本实验数据库采用 MySQL。Web 框架采用 MVC 架构和前后端分离技术，后端使用 Python 的 Web 框架 Bottle，前端使用 Vue.js 框架进行展示。

本实验代码运行于阿里云服务器，服务器采用 LAMP 环境（Ubuntu16.04 Apache PHP5.5），并安装有 Python3.6，python 包：bottle、jieba、json、pymysql。以上 python 包均可通过 pip3 进行安装。

实验数据来源于网络，包含歌曲名称、歌词和 ID，我将其通过 python 脚本导入数据库，保存歌曲的表总共有 4 个属性：（歌曲 ID，歌名，歌词，倒排索引 ID）。总共包括中文歌 99 首，编码方式为 UTF-8。实验数据可以在线上查看和下载 json 格式文件。

## 3. 建立权重矩阵

在本实验为数据库中的两个属性，歌词和歌名分别建立了权重矩阵表，两个表的实现方式相同，所以在介绍时将以歌词为例进行介绍，为歌名所建权重矩阵不再进行描述。

实现的整个过程非常复杂，在实验过程中，我通过多个模块实现，下面我将分模块讲述我的实现思路。

**读取数据库信息**

该模块在《简易信息检索系统实现——通过倒排索引实现布尔检索》 [2]已有详细描述。

**处理文本信息**

首先我们创建 song 类，在该类中进行文本信息的处理，这只是一个处理文本信息的类，不会进行任何数据库操作，所以在构造时，需要将文本的内容提供给它。

```
class song:
    def __init__(self, id, text):
        self.id = id
        self.lyric = text
```

由于本实验的数据集是中文歌，所以歌词信息处理需要用到中文分词工具，本实验所用的分词工具为 python 的 jieba 包，调用 jieba 中 cut_for_search 函数将文本分解为词组：

```
#用jieba将文本分解
def getwords(self):
 return jieba.cut_for_search(self.lyric)
```

由于 jieba 只是将文本尽行分解处理，并不会进行计数或者去除停用词，所以需要代码对于 jieba 返回的列表再次进行处理，得到我们需要的词项频率 (tf)：

```
def get_tf(self):
    tf = Counter()
    words = self.getwords()
    for word in words:
        if word in stopwords:
            continue
        tf[word] += 1
    return tf
```

利用 python 中的 Counter 类，我们很容易的将文本中非停用词的词项频率计算了出来。

这些就是我们仅通过文本信息能做到的全部事情，计算文档频率等操作都需要查找数据库，获取到倒排索引相关信息后才能进行计算。

**权重矩阵结构**

权重矩阵本身的结构应为一个二维表，横轴文本 id，纵轴词项 id，表中填写权重值。图 1 [3]展示了这种情况，从中我们可以清楚的看出，有大量表格项内容为 0，而在数据量大量增多的真实情况下，这种现象出现的会更加频繁，从而造成大量的空间浪费。因此，这种结构是非常不可取的。

| | Anthony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth ... |
|---|---|---|---|---|---|---|
| ANTHONY | 5.25 | 3.18 | 0.0 | 0.0 | 0.0 | 0.35 |
| BRUTUS | 1.21 | 6.10 | 0.0 | 1.0 | 0.0 | 0.0 |
| CAESAR | 8.59 | 2.54 | 0.0 | 1.51 | 0.25 | 0.0 |
| CALPURNIA | 0.0 | 1.54 | 0.0 | 0.0 | 0.0 | 0.0 |
| CLEOPATRA | 2.85 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| MERCY | 1.51 | 0.0 | 1.90 | 0.12 | 5.25 | 0.88 |
| WORSER | 1.37 | 0.0 | 0.11 | 4.15 | 0.25 | 1.95 |
| ... | | | | | | |

图 1. 一个简单的权重矩阵表

为处理该问题，我们采取这种情况下通常采取的措施——哈希。本实验以文档 ID 为纵轴，为每个文档建立一个列表，列表中存储该文档中出现的词项的 ID 和该词项的权重。所以下面需要处理的主要有两个问题，一是计算每个文档中各个词项的权重值，二是将其处理成为我们需要的格式。问题二处理方式非常简单且基础，因此不对其进行叙述。

**计算权重** (tf-idf)

本实验建立向量空间所用权重计算方法为 tf-idf。

这是信息检索中最出名的权重计算方法，该权重数学实现方法：

$$W_t, d = (1 + logtf_{t,d}) * logN/df_t \tag{1}$$

其中 tf_t,d 表示 t 在文档 d 中出现的次数,df_t 表示出现 t 的文档数目。

该权重计算的代码实现：

```
def get_tf_idf(self):
    words = self.song.getwordlist()
    for word in words:
      data = InvertedIndex.get_word_info(word)
      if data['num'] != 0:
        self.lyric_tf_idf[data['id']] = \
          (1+math.log10(self.tf[word]))\
         *math.log10(num_of_song/data['num'])
```

通过函数 getwordlist() 可以获取到该文档中去除停用词、去重的词项列表。get_word_info() 函数可以

获取到提供的词项的各项信息，返回值中下标 num 项代表词项在文档集中出现次数，id 项代表词项在倒排索引中的 ID。

获取了权重之后，计算欧氏长度也非常简单，不在陈述。

## 4. 向量空间下检索

同倒排索引类似，本实验中也将为所有歌曲建立的权重矩阵保存在了数据库，方便查找与保存。

利用为歌曲建立权重矩阵相同的方法，我们也可以得到搜索内容的各个词项的 tf-idf 权重。那么我们目前所需要解决的问题便是通过权重计算该查询内容与数据库中各个歌曲的相似度。

### 计算余弦相似度

相似度的计算方法采用余弦相似度计算方式：

$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

图 2. 相似度计算方法

相似度计算的代码实现：

```
def get_one_cos(self, sid):
    vector = VectorSpace(sid)
    res = 0
    for v in vector.w_list:
        res+=self.tfidf[v[0]]*v[1]
    return res/vector.length
```

该函数可以计算当前查询与某个歌曲 ID 的余弦相似度。通过 VectorSpace 类，可以获取到对应歌曲 ID 之前已经建立好的权重列表，而后将相同词项的权重机型乘法计算再除以该歌曲的欧氏长度即为查询与该歌曲的余弦相似度。

### 相似度排序

之前我们给出了对于两个已经建立向量空间模型的文本计算相似度的方法，而我们最后需要实现的是一个查询系统，系统在查询的基础上还需要对返回的结果根据相似度进行排序。据此看来，本实验当前完成的功能还有很大空缺。

我们目前要解决两个问题，一是返回那些结果，二是如何排序。

对于第一个问题，本实验的处理方式为将所有与查询相关的结果返回，即为将查询文本中的每个词项出现过的歌曲 ID 集合进行并集操作：

```
# 处理得出相关歌曲列表
def get_list(self, kind=0):
    words = self.song.getwords()
    for word in words:
        data = InvertedIndex.get_word_info(word, kind)
        if data['num'] != 0:
            if kind == 0:
                self.lyric_list = list(set(self.lyric_list).union(set(data['list'])))
            else:
                self.name_list = list(set(self.name_list).union(set(data['list'])))
```

图 3. 计算得出所有相关歌曲 ID 列表

在我们已经获取到所有相关歌曲 ID 列表之后，计算相似度时就不需要对于数据库中所有歌曲都匹配一遍，而只需要将列表中的歌曲计算相似度并进行排序。

```
def get_cos_list(self, kind=0):
    if kind == 0:
        the_list = self.lyric_list
    else:
        the_list = self.name_list
    for sid in the_list:
        if kind == 0:
            self.lyric_cos[sid] = self.get_one_cos(sid, kind)
        else:
            self.name_cos[sid] = self.get_one_cos(sid, kind)
```

图 4. 将歌曲 ID 列表所有歌曲与查询计算相似度

由于对于一个歌曲有歌词和歌名两个属性，所以在最终计算相似度时，需要将根据歌词计算的余弦相似度和根据歌名计算的余弦相似度进行加权，得出一个最后总的权重再进行排序。当然，我们也需要根据用户需要提供仅仅查询歌词和歌名的方法。目前本实验的对于歌名和歌曲的加权方式为

```
def get_sort(self, kind=0):
    if kind == 0:
        return counter_to_list(self.lyric_cos)
    elif kind == 1:
        return counter_to_list(self.name_cos)
    w_list = Counter()
    keys = set(self.lyric_cos).union(self.name_cos)
    for key in keys:
        w_list[key] = self.name_cos[key]*name_weight + self.lyric_cos[key]
    return counter_to_list(w_list)
```

图 5. 根据需求获取相似度排序列表

counter_to_list() 函数可以将 counter 类数据排序并返回 list 格式的数据。其具体实现方法如下：

```
def counter_to_list(co):
    res = {"id_list": [], "w_list": []}
    for sid in list(co.most_common()):
        res["id_list"].append(sid[0])
        res["w_list"].append(sid[1])
    return res
```

至此，我们获取了与用户查询相似的歌曲 ID 列表且该列表已经根据相似度进行了排序，接下来我们只需要将这些 ID 的歌曲返回给用户即可。

## 5. 总结

在本次实验中，我明显能够感受出来作业 1 带给我的提升，这次实验的内容相对于作业 1 的难度是上了一个档次的，思考的内容明显增多，但是在实现过程中，我遇到的困难却比作业 1 变少了，我认为这是源于在做作业 1，我熟悉了建立检索系统的技术方法。

本次实验由于有了作业 1 的倒排索引的基础，构建向量空间模型也轻松了许多。同时能明显感受出来向量空间模型和布尔检索之间计算量的巨大差别。构建过程明显慢了许多，这还是我们当前系统中歌曲数量有限的情况。

本次实验至此圆满完成，在实验过程中，我感受到了工程与理论的差别，并提升了自己将理论转化为实际的能力，同时对于信息检索系统有了更加深入的认识。

## 参考文献


[1] Wikipedia. 向量空间模型, 2018. [Online; accessed 10-August-2017]. 1

[2] 刘子晨. 简易信息检索系统实现——通过倒排索引实现布尔检索. 2018. 1, 2

[3] 温延龙. 第 4 讲文档评分、词项权重计算及向量空间模型. 2018. 2