中山大學
SUN YAT-SEN UNIVERSITY

《信息检索》课程大作业

题目 Title：TF-IDF 算法应用

院 系
School (Department)：数据科学与计算机学院

专 业
Major：软件工程

学生姓名
Student Name：陈贤鹏

学 号
Student No.：16340038

时 间
Submitted Time.：2020/01/06

中山大学 • 中国广州

【摘　要】

利用 TF-IDF 算法，从一个目录下的所有文档中进行检索，得到一个相关性由高到低的文档序列

【关键词】：TF-IDF

目　录

目录

第一章 概述/引言

1.1 背景

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 加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级。

1.2 问题的描述

根据所给关键字找出相关性最高的文档

1.3 本文的工作

给出一个 key（检索关键字），对一个目录下（包括子目录）的所有文档进行检索，依据相关度由高到低排序，返回一个文档的序列

1.4 论文结构简介

本文第一章主要是阐述研究 TF-IDF 算法应用的背景和意义以及本文所要大致工作内容；第二章主要综述本文在完成 TF-IDF 算法应用的过程中所作的相关工作以及选择的考量；第三章将会介绍我在解决问题过程中所作的工作以及所提出的方法和实现算法的步骤；第四章是实验结果的评估；第五章是对本文的总结以及对接下来研究方向的展望；最后是参考文献以及致谢内容。

第二章 相关工作综述

2.1 读取目录下的所有文档

其他类型文档的读取方式与.txt 一致，为了简单起见，所有的文档都是.txt 文档。读取该目录以及子目录下的所有.txt 文档

2.2 对读取的文档的内容进行分析，分词，统计

文档的读取使用 Java 的文件字节流读取，文档内容采用本学期第二次作业的文档案列

- D_1: jack and jill went up the hill
- D_2: to fetch a pail of water
- D_3: jack fell down and broke his crown
- D_4: and jill came tumbling after
- D_5: up jack got and home did trot
- D_6: as fast as he could caper
- D_7: to old dame dob who patched his nob
- D_8: with vinegar and brown paper

分词器是使用 IK 分词器，能对中文和英文进行分词，本次文档为英文文档，按照空格和标点符号分词

对分词结果进行统计，并计算 tf，idf，tf_idf

Term frequency(tf): tf(t,d) = 该文档中关键词频数/该文档总词数

idft is an inverse measure of the informativeness of term t.

idf 计算：idf(term) = $\log_{10}$(文档总数目/(包含关键词的文档数目+1)),加一是为了防止包含关键词的文档数为 0 而导致分母为零

tf_idf(t, d) = tf(t, d) * idf (term)，tf_idf 是一个综合性的评判指标，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。

2.3 根据 tf_idf 的值由高到低返回文档序列

先前统计出的 tf,idf,tf_idf 存在 Map 中，我们从 Map 中取出相应 value 进行排序，然后得到相应的 key 也就是文档的序列

第三章 完成 TF-IDF 算法应用的方法

3.1 读取目录下的所有文档

获取文档路径：获取该目录以及其子目录下所有文档的路径

```
public List<String> getAllFile(File file) {
    File[] fs = file.listFiles();
    List<String> allpath = new ArrayList<>();
    for(File f:fs){
        if(f.isDirectory())       //若是目录，则递归获取该目录下的文件路径
            getAllFile(f);
        if(f.isFile()) {              //若是文件，获取其路径
            allpath.add(f.toString());
            //System.out.println(f);
        }
    }
    return allpath;
}
```

这时可获得所有文档的数目用于计算 idf

3.2 对读取的文档的内容进行分析，分词，统计

分词：将文档中的文本分词词语

```
//创建分词对象
Analyzer anal=new IKAnalyzer(true);
StringReader reader=new StringReader(text);
//分词
TokenStream ts=anal.tokenStream("", reader);
CharTermAttribute term=ts.getAttribute(CharTermAttribute.class);
//遍历分词数据
```

统计词频：

```
public TreeMap<String, Integer> displayWordCount(String fileName) {
        //以字符流的形式统计
        TreeMap<String, Integer> tm = new TreeMap<String, Integer>();
        try {
            //读取文件
            FileReader fileReader = new FileReader(fileName);
            //使用流的方式读取内容
            BufferedReader reader = new BufferedReader(fileReader);
            //使用 TreeMap，它会自动将结果按照字典的顺序排序

            String readLine = null;
            while((readLine = reader.readLine()) != null){
                //将字母排序为小写
                readLine = readLine.toLowerCase();
                //过滤出只含有字母的字段
                String[] str = readLine.split("[\\s]+");
                //过滤掉所有的空格,“+”代表多个的意思。
                for (int i = 0; i < str.length; i++) {//循环统计出现次数
                    String word = str[i].trim();
                    if (tm.containsKey(word)) {
                        tm.put(word, tm.get(word) + 1);
                    } else {
                        tm.put(word, 1);
                    }
                }
            }
        ……
```

3.3 根据 tf_idf 的值由高到低返回文档序列

计算 tf，idf，tf_idf：

```
this.idf = (log(Double.valueOf(allfile) / Double.valueOf(hasfile + 1))) / log(10);
……
if (allword != 0){
        this.tf = Double.valueOf(key_times) / Double.valueOf(allword);
        file_tfidf.put(allpath.get(i), this.tf); //这时第二列存的是 tf，还不是 tf_idf
}
……
this.tf_idf = this.idf * value_t;
file_tfidf.put(key_t, this.tf_idf); //这时第二列存的是 tf_idf
```

将 Map 中数据转存到数组排序:

排序：排好序后再存回 Map 中，这时 Map 中的序列正是所需的与检索的关键词相关度由高到低的序列

```
Iterator<Map.Entry<String, Double>> it = unsortedMap.entrySet().iterator();
int index = 0;
while(it.hasNext()) {
        Map.Entry<String, Double> entry = it.next();
        String key_t = entry.getKey();
        Double value_t = entry.getValue();
        keystring[index] = entry.getKey();
        value[index] = entry.getValue();
        index++;
}
index = 0;
```

```
for (int i = 0; i < len - 1; i++) {
        for (int j = i + 1; j < len; j++) {
                if (value[i] < value[j]) {
                        Double temp = value[i];
                        String temp_s = keystring[i];
                        value[i] = value[j];
                        keystring[i] = keystring[j];
                        value[j] = temp;
                        keystring[j] = temp_s;
                }
```

第四章 实验结果与分析

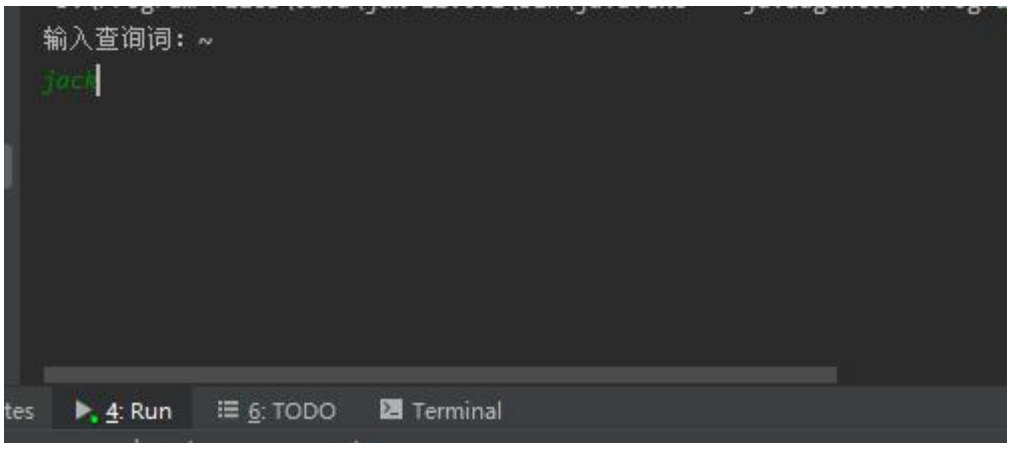
4.1 数据集

数据集是.txt 文档，本次使用第二次作业案例文档

- D_1: jack and jill went up the hill
- D_2: to fetch a pail of water
- D_3: jack fell down and broke his crown
- D_4: and jill came tumbling after
- D_5: up jack got and home did trot
- D_6: as fast as he could caper
- D_7: to old dame dob who patched his nob
- D_8: with vinegar and brown paper

4.2 结果

输入查询词“jack”：



查看结果：第一列是文档的路径，视作文档的 DocId，第二列是计算出的 tf_idf

值，与第二次作业中手算结果一致

```
输出检索结果：
文件Id    tf_idf值
D:\Learning_cxp\IR_resource\text1.txt    0.04300428509485445
D:\Learning_cxp\IR_resource\text3.txt    0.04300428509485445
D:\Learning_cxp\IR_resource\text5.txt    0.04300428509485445
D:\Learning_cxp\IR_resource\text4.txt    0.0
D:\Learning_cxp\IR_resource\text2.txt    0.0
D:\Learning_cxp\IR_resource\text6.txt    0.0
D:\Learning_cxp\IR_resource\text7.txt    0.0
D:\Learning_cxp\IR_resource\text8.txt    0.0

Process finished with exit code 0
```

查询“jill”:

```
输入查询词：~
jill
查询"jill"中……
```

查看结果：也是 tf_idf 值最高的 D4 排最前面

```
输出检索结果：
文件Id    tf_idf值
D:\Learning_cxp\IR_resource\text4.txt    0.08519374645445622
D:\Learning_cxp\IR_resource\text1.txt    0.060852676038897296
D:\Learning_cxp\IR_resource\text3.txt    0.0
D:\Learning_cxp\IR_resource\text2.txt    0.0
D:\Learning_cxp\IR_resource\text5.txt    0.0
D:\Learning_cxp\IR_resource\text6.txt    0.0
D:\Learning_cxp\IR_resource\text7.txt    0.0
D:\Learning_cxp\IR_resource\text8.txt    0.0
```

分析：

虽然文档库比较小，但是功能都是齐全的，包括读取全部的文档，词频统计，以及排序。用作业的案例来测试，比较容易看出结果的正确性。这两次测试得出的序列与作业手算是一致的，测试通过。

本次项目代码已经上传 GitHub：https://github.com/chenxp38/IR_BHW.git

第五章 总结与展望

本文中 TF-IDF 采用文本逆频率 IDF 对 TF 值加权取权值大的作为关键词，TF-IDF 算法实现简单快速，并且很实用，可以有效的对文档的相关性进行排序，但是仍有许多不足之处：

（1）没有考虑特征词的位置因素对文本的区分度，词条出现在文档的不同位置时，对区分度的贡献大小是不一样的。

（2）按照传统 TF-IDF，往往一些生僻词的 IDF(反文档频率)会比较高、因此这些生僻词常会被误认为是文档关键词。

（3)传统 TF-IDF 中的 IDF 部分只考虑了特征词与它出现的文本数之间的关系，而忽略了特征项在一个类别中不同的类别间的分布情况。

（4）对于文档中出现次数较少的重要人名、地名信息提取效果不佳。

对此的改进算法有 TF-IWF 算法。另外对于本应用的功能拓展，可以同时提取文档标题作文检索目标，并且对于标题给与更高的权重，让关键词处于标题位置的文档更加靠前。

参考文献：

[1]. Kang Liu,Peiyuan Qiu,Song Gao,Feng Lu,Jincheng Jiang,Ling Yin. Investigating urban metro stations as cognitive places in cities using points of interest[J]. Cities,2020,97.