

一 题目

试按 tf-idf 在剔除一些常用词后给出文本中术语的统计算法和程序,并按降序进行排序。

二 主要思想

TF 意思是词频(Term Frequency), IDF 意思是逆向文件频率(Inverse Document Frequency)。

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。

TFIDF 实际上是: $TF * IDF$ 。

TF 表示词条在文档 d 中出现的频率。

IDF 的主要思想是: 如果包含词条 t 的文档越少, 也就是 n 越小, IDF 越大, 则说明词条 t 具有很好的类别区分能力。

tfidf 公式:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

分子是该词在文件中的出现次数, 而分母则是在文件中所有字词的出现次数之和。

$$idf_i = \log \frac{|D|}{|\{j: t_i \in d_j\}|}$$

由总文件数目除以包含该词语之文件的数目, 再将得到的商取对数得到。

分子: $|D|$: 语料库中的文件总数。

分母: 包含词语的文件数目 (即的文件数目) 如果该词语不在语料库中, 就会导致分母为零, 因此一般情况下使用 $1 + |\{d \in D: t \in d\}|$ 作为分母。如下:

$$idf_i = \log \frac{|D|}{1 + |\{d \in D: t \in d\}|}$$

总公式为:

$$tfidf_{i,j} = tf_{i,j} * idf_i$$

三 算法

备注：没有大规模数据集，只能小规模测试，总词典为所有文件遍历一遍所得。

监听文件夹，获取所有文件，以map形式存每个文件的tfidf值，key为文件名，value为tfidf值。另外以tfMap与idfMap保存每个文件的tf值与idf值，最后计算tfidf。先遍历文件夹，得到文件名，初始化tfidfMap，tfMap，idfMap，wordNumMap。

计算tf值，map结构为Map <String,Map<String,Integer>> tfMap，外部map key String为文件名，value为文件里的每个词的词频map，内部map key为文件中词，value为出现次数。
以外部map的key读取文件，获取文件字符串，ICTCLA分词，然后对分词的length()即为wordNumMap，记录每个单词数。

根据tfMap，生成idfMap，idfMap的结构仍跟tfMap一样，Map <String,Map<String,Integer>>idfMap，内部map value 为该词在所有文件中出现的文件数。
两层循环得到idfMap，外层循环为idfMap的文件map，内层循环为每一个文件的tfMap，如果外循环的key在内循环某文件出现过，则value++，如此得到idfMap。

根据tfMap及idfMap生成tfidfMap，只需根据公式进行计算得到结果保存即可。

降序排序并输出tfidf值，将tfidfMap中的每个内层map转化为list，自带函数排序，输出即可，设置一个阈值，小于此值的都为常用词，break循环即可。

四 程序

//分词部分代码略，调用开源分词工具

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
```

```

import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import code.NlpirTest;

public class Main {

    static Map <String, Map <String, Integer>> tfMap = new HashMap<String,
Map <String, Integer>>());
    static Map <String, Integer > wordNumMap = new HashMap<String ,
Integer>());
    static Map <String, Map <String, Integer>> idfMap = new HashMap<String,
Map <String, Integer>>());
    static Map <String, Map <String, Double>> tfidfMap = new
HashMap<String, Map <String, Double>>());
    static int fileNum=0;

    public static void main(String[] args){

        //遍历文件夹，生成所有文件的map
        String path = "E:/testFile"; // 路径
        File f = new File(path);
        if (!f.exists()) {
            System.out.println(path + " not exists");
            return;
        }
        File fa[] = f.listFiles();
        fileNum=fa.length;
        for (int i = 0; i < fa.length; i++) {
            File fs = fa[i];
            if (!fs.isDirectory()) {
                tfMap.put(fs.getName(),null);
                idfMap.put(fs.getName(),null);
                wordNumMap.put(fs.getName(),0);
                tfidfMap.put(fs.getName(),null);
            } else {
                System.out.println(fs.getName() + " [目录]");
            }
        }
    }
}

```

```

String tmpFileString="";
String sOutputString="";
Map <String, Integer> tmpMap=new HashMap<String, Integer>();
Map <String, Integer> tmpMap1=new HashMap<String, Integer>();
//计算每个文件的tf值
for (String key : tfMap.keySet()) {
    //读文件生成文件字符串
    tmpFileString=readFileByLines("E:/testFile/"+key);
    tmpMap=new HashMap<String, Integer>();
    tmpMap1=new HashMap<String, Integer>();
    //每个文件分词把tfMap生成
    try {
        //System.out.println(tmpFileString);
        sOutputString = NlpTest.nlpir(tmpFileString);
        wordNumMap.put(key, sOutputString.length());
        System.out.println(sOutputString);
        String[] strs=sOutputString.split("/([a-z]*)\\d*\\s");

        for(int i=0,len=strs.length;i<len;i++){
            //System.out.println(strs[i].toString());
            if(tmpMap.containsKey(strs[i].toString())){
                tmpMap.put(strs[i].toString(),
tmpMap.get(strs[i].toString()+1);
            }else{
                tmpMap.put(strs[i].toString(),1);
            }
            tmpMap1.put(strs[i].toString(),0);
        }
        tfMap.put(key,tmpMap);
        idfMap.put(key, tmpMap1);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
System.out.println(tfMap);
System.out.println(tfMap);

//根据tfMap生成idfMap
for (String key : tfMap.keySet()) {
    //System.out.println(tmpMap.get("fsad"));
    for (String key1 : tfMap.keySet()) {
        //System.out.println(tmpMap.get("fsad"));
        tmpMap1=idfMap.get(key);
    }
}

```

```

        tmpMap=tfMap.get(key1);
        for(String key2:tmpMap1.keySet()){
            //System.out.println(key2);
            if(tmpMap.containsKey(key2)){
                tmpMap1.put(key2, tmpMap1.get(key2)+1);
            }
        }
        idfMap.put(key, tmpMap1);
    }
}
System.out.println(wordNumMap);

//根据tfMap, idfMap生成tfIdfMap
Map <String, Double> tmpMap2=new HashMap<String, Double>();
for (String key : tfMap.keySet()){
    tmpMap=tfMap.get(key);
    tmpMap1=idfMap.get(key);
    tmpMap2=new HashMap<String, Double>();
    for(String key1:tmpMap.keySet()){
        //System.out.println(tmpMap.get(key1));
        //System.out.println(key1+tfMapaLL.get(key1));
        //tf值
        double tfValue=tmpMap.get(key1)*1.0/wordNumMap.get(key)*1.0;
        //idf值
        double idfValue=Math.Log(fileNum*1.0/(1+tmpMap1.get(key1))*1.0);
        //System.out.println(tmpMap1.get(key1));
        //System.out.println(tfValue+" "+idfValue);
        tmpMap2.put(key1,tfValue*idfValue);
    }
    tfidfMap.put(key, tmpMap2);
}

System.out.println(tfidfMap);

//排序并输出 tfidf
List<Map.Entry<String,Double>> list;
for (String key : tfidfMap.keySet()){
    System.out.println(key);
    tmpMap2=tfidfMap.get(key);
    list = new
ArrayList<Map.Entry<String,Double>>(tmpMap2.entrySet());
    Collections.sort(list,new Comparator<Map.Entry<String,Double>>() {
        //升序排序
        public int compare(Entry<String, Double> o1,

```

```

        Entry<String, Double> o2) {
            return o2.getValue().compareTo(o1.getValue());
        }
    });

    for(Entry<String, Double> mapping:list){
        // if(mapping.getValue()>0.3){

System.out.println(mapping.getKey()+":"+mapping.getValue());
        // }else{
        //     break;
        // }
    }
}

private static String readFileByLines(String fileName) {
    File file = new File(fileName);
    String fileString="";
    BufferedReader reader = null;
    try {
        //System.out.println("以行为单位读取文件内容，一次读一整行：");
        reader = new BufferedReader(new FileReader(file));
        String tempString = null;
        int line = 1;
        // 一次读入一行，直到读入null为文件结束
        while ((tempString = reader.readLine()) != null) {
            // 显示行号
            fileString+=tempString;
            //System.out.println("line " + line + ": " + tempString);
            line++;
        }
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e1) {
            }
        }
    }
    return fileString;
}

```

}

}

五 数据集

在搜狗下载的新闻数据集。

此电脑 > 文档 (E:) > testFile					搜索"testFile"
名称	修改日期	类型	大小		
10.txt	2017/10/15 21:49	文本文档	7 KB		
11.txt	2017/10/15 21:49	文本文档	1 KB		
12.txt	2017/10/15 21:49	文本文档	3 KB		
13.txt	2017/10/15 21:49	文本文档	2 KB		
14.txt	2017/10/15 21:49	文本文档	3 KB		
15.txt	2017/10/15 21:49	文本文档	1 KB		
16.txt	2017/10/15 21:49	文本文档	1 KB		
17.txt	2017/10/15 21:50	文本文档	8 KB		
18.txt	2017/10/15 21:50	文本文档	8 KB		
19.txt	2017/10/15 21:50	文本文档	1 KB		
20.txt	2017/10/15 21:50	文本文档	9 KB		

六 测试结果

16.txt
占:0.008972358380202237
医疗:0.006729268785151679
人口:0.005324215324623578
全国:0.0051287486215668195
城市:0.0051287486215668195
试点:0.004486179190101119
卫生:0.004486179190101119
农村:0.004486179190101119
病:0.004486179190101119
80%:0.004486179190101119
20%:0.004486179190101119
因:0.004486179190101119
占:0.004486179190101119

15.txt

保险:0.014342832506813674

贷:0.01229385643441172

险:0.01229385643441172

车:0.009369829212477844

还款:0.00614692821720586

逾期:0.004097952144803907

赔偿:0.004097952144803907

退出:0.004097952144803907

贷款:0.0036475994411483653

由:0.0024317329607655767

银行:0.002185585830181426

>

<