**大数据处理期末课程设计**

系别：计算机科学与技术

组号：2019st07

组员：161220106 施雨杰，161220011 陈嘉言

161220012 陈剑豪，161220123 王荆涛

|  |  |
| --- | --- |
| **实验选题** | ——“金庸的江湖” |
| **主要内容** | 1. 借助Ansj\_Seg工具的中文分词与人名提取。 2. 基于MapReduce计算框架的人物关系分析。（具体包含关系权值归一化，人物PageRank值计算，标签传播……） 3. 基于Spark的对上述功能的复现。 4. 借助Gephi工具的结果图示化呈现。 |

|  |  |  |
| --- | --- | --- |
| 计算模型 | **MapReduce**（必做部分） | **Spark**（选做部分） |
| 基础架构 | hadoop（版本：2.7.1）  https://ss1.baidu.com/6ONXsjip0QIZ8tyhnq/it/u=1523529781,231372822&fm=58&bpow=563&bpoh=447 | spark（版本：2.4.3） |
| 编程语言 | Java | Python |
| 开发环境（IDE） | IntelliJ IDEA | Pycharm |

|  |  |
| --- | --- |
| 组内分工 |  |

**备注格式：**

实验中用到的指令用黄色背景标出

截图中的重要信息用红色椭圆或方框标出

**Content**

1. [说明](#〇、说明)

一、基于MapReduce计算框架的数据处理

task1：[中文分词与人名提取](#task1：中文分词与人名提取)

task2：[人物共现关系及次数统计](#task2：人物共现关系及次数统计)

task3：[关系权值归一化](#task3：关系权值归一化)

task4：人物PageRank值计算

task5：人物关系图标签传播（Label Passing）

task6：任务4、5结果的排序输出

task7：（拓展）任务5所得标签与实际书名的自动化匹配

……

（提取人物出场段落&标签传播正确率检验）

（代码整合与集群运行，附效果截图）】

二、基于Spark计算框架的复现

（1）实现思路简述

（2）性能优化分析

三、数据处理结果的图示化呈现

（1）Gephi工具的安装

（2）数据文件的生成及导入

（3）关系图效果展示（附模型生成视频？简单分析）

四、………

**〇、说明**

本文系“大数据处理综合实验”——期末课程设计实验报告，任务选题为“金庸的江湖”。主要内容是：以金庸十五本武侠小说文本为原始数据，从中提取中指定的人物名，建立人物之间的共现关系，通过计算PR值、计算标签传播等方式发掘其中隐含的信息，并最终通过图示化工具予以清晰呈现。

上述整个数据处理流程都已通过MapReduce和Spark两种计算框架分别实现。由于MapReduce为本课程的主要内容，实验报告的重点也将放在这一部分（具体参见第二章）。Spark实现思路只简单介绍，但会对其相对于MP的性能优化进行简要分析（参见第三章）。

在分任务描述每个MapReduce的task具体实现时，尽量遵循：任务目的 + 输入输出样例 + 算法描述（Setup+Map+Reduce，伪代码or代码） + 实际运行结果（截图）

的格式，尽可能详细的说明每个子任务的实现思路。

在完成课程实验必做内容（共现关系提取、PageRank计算…）和明确选做内容（标签传播、spark复现…）基础上，本小组另外增加了一些相关的拓展内容：…………

最后，由于本实验报告具有备忘的性质，在一些具体操作上描述非常细致，这些内容对于审阅者而言可能是冗长且多余的。本文中会在合适的地方设置跳转链接，帮助审阅者跳过琐碎的操作细节，直接查看最终呈现结果。

[↑返回目录](#Content)

**一、基于MapReduce计算框架的数据处理**

**task1：中文分词与人名提取**

**（1）任务目的：**

借助Ansj\_Seg分词工具，基于给定的namelist.txt人名列表文件，建立关于金庸小说人物名的词库。以15本金庸小说作为输入，进行中文分词，提取出其中指定的人物名。

**（2）输入输出样例：**

（Input：小说中的一段文本）

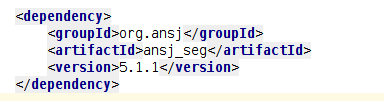
王处一道：“你六位师父既然尚未明言，我也不便代说。”他曾听丘处机说起过前后的原委，对江南六怪的义举心下好生相敬。他和马钰是一般的心思，也盼江南六怪获胜，不过他是师弟，却不便明劝丘师哥相让，今日见了郭靖的为人，暗自思量如何助他一臂之力，却又不能挫折丘师哥的威名，决意届时赶到嘉兴，相机行事，从中调处。

（Output：其中出现的（明确在词库中出现过的）人物名）

王处一 丘处机 马钰 郭靖

**（3）相关配置：**

本实验的MapReduce部分统一使用Intellij IDEA下的Maven项目管理。在项目的pom.xml配置文件<dependencies>……</dependencies>中添加如下依赖：



其中版本一栏选择“5.1.1”。

点击“Import Changes”自动完成配置。



在java源文件开头处添加以下几条依赖：

import org.ansj.domain.Result;

import org.ansj.domain.Term;

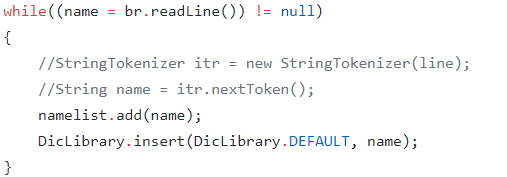
import org.ansj.library.DicLibrary;

import org.ansj.splitWord.analysis.ToAnalysis;

**（4）算法描述：**

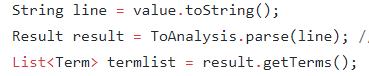
Setup()阶段：

建立人名词库。从助教提供的namelist.txt文件中逐行读取人物名（name），调用ansj库中的“DicLibrary”功能，将读出的name加入词库。同时把name加入一个全局列表namelist，供之后分词时过滤筛选。

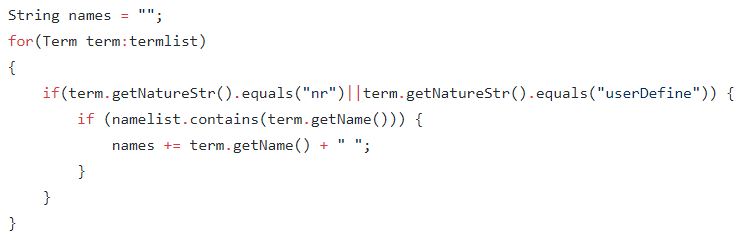


Map()阶段：

对输入的每一行文本，调用ansj库的“ToAnalysis”功能，完成中文分词，并将结果暂存入名为termlist的列表。



遍历termlist列表，筛选出其中可能是人物名的词（词性为“nr”或“userDefine”）。如果这些词确定在之前建立的namelist列表中出现过，则将其写入字符串names。



词库自行识别的人名

setup阶段添加的人名

（\*\*\*这里之所以要重复检查每个term是否在namelist列表中，是因为某些被词库自行识别出的“人名”可能并没有在namelist.txt中出现过。仅凭词性不能完全确定选出的人名是否应该加入输出，必须通过namelist列表进行额外的一轮过滤。）

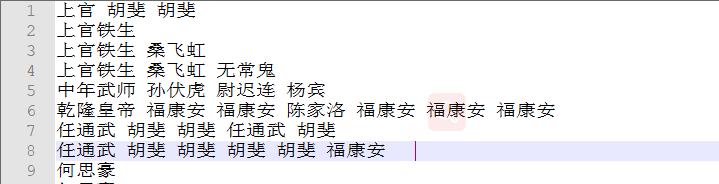
最后，将所得的（非空的）names文本发出即可。

Reduce()阶段：

task1的reduce阶段不作任何处理。直接将从mapper端传来的names文本写入输出文件。

**（5）运行结果**

这是小说《飞狐外传》分词提取结果的开头部分。



**task2：人物共现关系及次数统计**

**（1）任务目的：**

基于单词同现算法对任务一得到的人名文件进行统计分析。在人物同现分析中，如果两个人在原文的同一段落出现，则认为两个人发生了一次同现关系。通过对这些关系逐一进行统计，得到一份可以大致反应人物关系亲密度的人物同现记录文件。

**（2）输入输出样例：**

输入：任务1的输出，即每行均只由若干人名构成的文本文件，如：

狄云 戚芳 戚芳 戚长发 卜垣

戚芳 卜垣 卜垣

输出：每本小说文件的人物同现次数，如：

狄云 戚芳 1

狄云 戚长发 1

狄云 卜垣 1

戚芳 狄云 1

戚芳 戚长发 1

戚芳 卜垣 2

戚长发 狄云 1

戚长发 戚芳1

戚长发 卜垣 1

卜垣 狄云 1

卜垣 戚芳 2

卜垣 戚长发 1

**（3）算法描述：**

Map()阶段：

Map阶段的流程分为两部分，

1. 对人名列表进行去重处理

2. 对去重的人名列表遍历，从而得到所有人名二元组，将其发送给后续程序进行处理

在本次实验的单词同现计算中，同一段落中的两个人名A、B无论出现多少次，都只对其进行一次记录，因此在拿到一条写有本段落全部人名的信息时，首先要对该段落的人名进行去重处理，使得最后参与记录同现次数的人名均不相同。

String line = value.toString();

StringTokenizer itr = new StringTokenizer(line);

ArrayList<String> names = new ArrayList<>();

for(; itr.hasMoreTokens();) {

String name = itr.nextToken();

if(!names.contains(name)) {

names.add(name);

}

}

在进行人名去重并得到最终的人名列表ArrayList之后，对该表进行遍历以找到所有在列表中任取两个元素可以得到的全部组合，共计n(n-1)/2种，对每种组合中的人名A，B，分别发送[(A,B), 1], [(B,A), 1]两种同现记录。

int len = names.size();

for(int i = 0; i < len; i++) {

for(int j = i+1; j < len; j++){

Text word1 = new Text();

Text word2 = new Text();

word1.set(names.get(i)+" "+names.get(j));

word2.set(names.get(j)+" "+names.get(i));

context.write(word1, new IntWritable(1));

context.write(word2, new IntWritable(1));

}

}

Combiner阶段：

Combiner阶段的任务很简单，主要是对map端分析产生的大量key相同，value均为1的键值对进行初步整合，统计得到同一key的value总数后，重新按输入数据的组织形式发送给reduce端。

int sum = 0;

for(IntWritable val:values) {

sum += val.get();

}

result.set(sum);

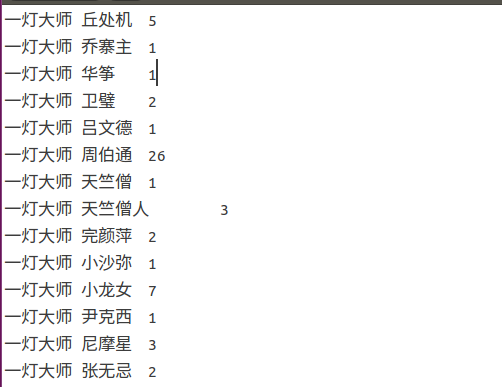
context.write(key, result);

Reduce阶段

Reduce阶段的任务和combiner阶段完全相同，在此不再赘述。专门额外添加combiner函数参与程序运行的目的在于进一步提高mapreduce运行过程中的数据传输速度，减少数据大量积压带来的通信开销。

**（5）运行结果**

下图是task2单词同现统计结束后得到输出文件的部分结果截图。



**task3：关系权值归一化**

**（1）任务目的：**

将任务2获得的人物共现关系统计数据进一步转化为用邻接表表示的，描述人物之间关系的关系图。在关系图中，人物是顶点，人物之间的关系是边，它通过共现关系确定；而边的权值则由统计得到的人物共现次数决定。为了方便后续任务的分析，在建立了该图以后，还需要对共现次数进行归一化处理，即将共现次数转化为共现概率。

**（2）输入输出样例：**

输入：任务2的输出，如：

狄云 戚芳 1

狄云 戚长发 1

狄云 卜垣 1

戚芳 狄云 1

戚芳 戚长发 1

戚芳 卜垣 2

戚长发 狄云 1

戚长发 戚芳1

戚长发 卜垣 1

卜垣 狄云 1

卜垣 戚芳 2

卜垣 戚长发 1

输出：人物关系图归一化邻接表

狄云 戚芳 0.33333|戚长发 0.333333|卜垣 0.333333

戚芳 狄云 0.25 |戚长发 0.25|卜垣 0.5

戚长发 狄云 0.33333|戚芳 0.333333|卜垣 0.333333

卜垣 狄云 0.25|戚芳 0.5|戚长发 0.25

**（3）算法描述：**

Map阶段：

Map阶段所能获得的输入是形如[(A, B), 1] 这样的由task2分析得到的人物同现数据，其关键词由两个人名组成，而在本任务中，关键词均由一个人名构成，因此为了方便起见，在map阶段对task2的键值组合进行修正，将其由[(A, B), 1]变更为[A，(B,1)]的形式

String line = value.toString();

StringTokenizer itr = new StringTokenizer(line);

String name1, name2, counts;

name1 = itr.nextToken();

name2 = itr.nextToken();

counts = itr.nextToken();

Text word1 = new Text();

Text word2 = new Text();

word1.set(name1);

word2.set(name2 + ","+counts);

context.write(word1, word2);

Combiner阶段：

该阶段主要负责将map阶段产生的新的键值对进行整合、重组与计数，将若干[A，(B,1)]形的新键值对进一步重新整合为[A，[(B1,1)，(B2,1)，(B3,1)；3]]的形式发送给reduce端进行最终处理。

在新的键值对组合中，关键词仍然为某个人名，其值则变为了该人名对应的全部同现关系中的另一个人及出现次数的组合；同时，在此过程中统计得到的关键词所对应的全部同现关系出现次数总和也将被附加在值的最后。

int sum = 0;

StringBuilder out = new StringBuilder();

for(Text val:values) {

String[] value = val.toString().split(",");

sum += Integer.valueOf(value[1]); //统计同现关系出现总数

if(values.iterator().hasNext()) {

out.append(value[0] + " " + value[1] + "|"); //将map阶段重组得到的键值对中

} 的值拼接起来

else {

out.append(value[0] + " " + value[1]);

}

}

out.append("," + sum); //将同现关系出现总数附加在新值

context.write(key, new Text(out.toString())); 的最后

Recude阶段：

Reduce阶段的任务分为如下部分，

1. 对combiner阶段得到的键值对做最终的合并与信息提取

2. 根据前一步提取计算得到的同现关系总数对每个同现关系做归一化处理

由上文可知，在combiner阶段输出的键值对是如下形式的：

Key：name，value：name1 Num1 | name2 Num2 …… | name\_n Num\_n，number

其中，number代表了以name为关键词的人名部分同现关系的总和。在reduce阶段的第一部分，面对key相同的value列表下若干不同的value值，首先要提取出number并计算总和，其次要将剔除了number后的value做最终的拼接。

int sum = 0;

StringBuilder temp = new StringBuilder();

for(Text val:values) {

String[] value = val.toString().split(","); //将同现次数之和与(人名，数字)串分离

sum += Integer.valueOf(value[1]); //统计同现关系总和

if(values.iterator().hasNext()) { //将对应key下的所有(人名，数字)串合并

temp.append(value[0] + "|");

}

else {

temp.append(value[0]);

}

}

在最终将某个key的全部(人名，数字)串合并，并统计得到了同现次数总和后，修改每个(人名，数字)对中的数字，使之归一化即可。

String[] valStrings = temp.toString().split("\\|"); //将(人名，数字)对分离

StringBuilder out = new StringBuilder();

for(int i = 0; i < valStrings.length; i++) {

String[] tempVal = valStrings[i].split(" ");

if(i == 0) { //提取(人名，数字)对中的数字并计算归一化后的新值，

out.append(tempVal[0] + " " + (double)Integer.valueOf(tempVal[1])/sum);

}

else {

out.append("|"+tempVal[0] + " " + (double)Integer.valueOf(tempVal[1])/sum);

}

}

context.write(key, new Text(out.toString()));

**（5）运行结果**

以下是输出结果文件的部分截图

