# 哈尔滨工业大学

## <<大数据分析>>

## 实验报告之一

**(2021 年度春季学期)**

| | |
|---|---|
| 姓名: | 卢兑玑 |
| 学号: | L170300901 |
| 学院: | 计算机学院 |
| 教师: | |

# 实验一 数据预处理

## 一、实验目的

掌握数据预处理的步骤和方法，包括数据抽样、数据过滤、数据标准化和归一化、数据清洗。理解数据预处理各个步骤在大数据环境下的实现方式。

## 二、实验环境

此处请说明所使用的框架（Hadoop、Spark 等）以及编程语言。

使用框架：Hadoop-2.9.2
编程语言：Java
系统：Linux（Centos7）
IDE：Eclipse

## 三、实验过程及结果

### 3.1 数据抽样

我采用了分层抽样加系统抽样的方法，首先构造了一个全局变量 `public static Map<String, Integer> NUM` 用来记录每种职业的总人数，再用一个全局变量 `public static int SUM = 0;` 记录所有职业的总人数。

Map 阶段：
在 map 阶段，统计不同职业以及它的总人数并把其以 key-value 对的形式写入 NUM 中，然后在 context.write 过程中把这一行数据中的职业作为 key，把这一行数据（元组）作为 value。
Reduce 阶段：
在 reduce 阶段，首先设定要采样的数据数，这里我把总采样数设定为 10000，那么每种职业的总采样数则如下所示：

```
int SampleSum = (int) Math.round(10000.0 * NUM.get(career)/SUM);
```

那么之后再在每种职业内部采取系统抽样的方法进行抽样，则用如下方式计算出采样间隔：

```
int tmp = (int)Math.floor(1.0 * NUM.get(career)/SampleSum);
```

于是便可通过迭代来选取相应采样点的数据进行最终输出。输出形式为`(null, new Text(data))`。

Map 函数的输入:
类型: Mapper<LongWritable, Text, Text, Text>
从左到右依次是: 文件的行数 文件的一行的内容 map 返回的 key 属性 user_career, value 值为文件的一行内容
输入:Mapper 的前两个参数
输出:Mapper 的后两个参数
操作:
将所有的数据转换成<user_career, recorder>的键值对
Reduce 函数的输入:
类型: Reducer<Text, Text, Text, Text>
从左到右依次是: 属性值 user_career,文件的一行的内容,null,文件一行的内容
输入:Reducer 的前两个参数
输出:Reducer 的后两个参数
操作:对同一个 user_career 的所有记录进行分层抽样,我通过每 100 条记录抽取 1 个记录的方式进行了抽样.

## 3.2 数据过滤

数据过滤步骤比较简单，由于已经给出了奇异值的边界，不需要再进行排序计算，所以直接比较过滤即可。
Map 阶段：
在 map 阶段当中，将 longitude 和 latitude 的边界值表示出来，然后直接进行比较：
```
if(longitude >= longiLow && longitude <= longiHigh && latitude >= latiLow && latitude <= latiHigh)
```
选取符合条件的数据元组（行），写入 map 中，将“”作为 key，将数据元组作为 value 即可。
Reduce 阶段：
在 reduce 阶段中，直接通过迭代将过滤后的数据元组输出即可。输出形式为 `(null, new Text(data))`。
Map 函数的输入:
类型: Mapper<LongWritable, Text, Text, Text>
从左到右依次是: 文件的行数 文件的一行的内容 map 返回的 key 属性 user_career, value 值为文件的一行内容
输入:Mapper 的前两个参数
输出:Mapper 的后两个参数
操作:
将所有的数据转换成<user_career, recorder>的键值对
Reduce 函数的输入:
类型: Reducer<Text, Text, Text, Text>
从左到右依次是: 属性值 user_career,文件的一行的内容,null,文件一行的内容
输入:Reducer 的前两个参数
输出:Reducer 的后两个参数
操作:对所有的同一个 user_career 的数据按照 rate 值进行从小到大排序,就可以得到所有数据的前 1%的界限值和后 1%的界限值,然后遍历所有的数据,只保留数据记录的 rate 位于界限之间的记录即可完成数据的恶意评分的过滤,同时只保留 longitude 和 latitude 位于有效范围分别

是[8.1461259, 11.1993265]和[56.5824856, 57.750511]的值域内的数据记录.

## 3.3 数据格式转换与归一化

这一项分为两个步骤，分别为转换和归一化。

先来说数据转换：
我在这里选取要把日期数据都转化为“2020-1-1”形式的，将温度数据都转化为摄氏度。构造了 DateConversion 和 TemperatureConversion 两个函数，用来把非标准化的数据转化为标准化数据。DateConversion 中首先检测是否含有“/”，对应“2020/1/1”格式，对于这种格式，将/替换为-即可；再检测是否含有“,”，对应“March 1,2020”格式，对于这种格式，分别按照“ ”和“,”进行分割，然后重新拼成标准格式即可。在 TemperatureConversion 中，将华氏温度按照公式计算成摄氏温度即可。
Map 阶段：
在 map 阶段，对于每行数据，对其进行日期转换和温度转换，然后将新数据替换老数据，即可写入 map 中，将“”作为 key，将新的数据元组作为 value 即可。
Reduce 阶段：
在 reduce 阶段中，直接通过迭代将转换后的数据元组输出即可。输出形式为 `context.write(null, new Text(line));` 。

再来说归一化：
在归一化阶段，首先设置两个全局变量 MAX 和 MIN，初始值 MAX 为 0，MIN 值为 100000。这里根据实验要求，采取（x-min）/（max-min）的归一化方法。
Map 阶段：
在 map 阶段，由于 rating 数据有缺失值，故先判断是否有“？”如果没有再进行处理：

```
if(rate > MAX) {
    MAX = rate;
}
if(rate < MIN) {
    MIN = rate;
}
```

，最终获得 rating 值的最大最小值。写入阶段将“”作为 key，将数据元组作为 value 即可（rating 为缺失值的元组也要写入）。
Reduce 阶段：
进行迭代处理数据元组，如果 rating 不是缺失值，则按照（x-min）/（max-min）公式进行计算，如果是缺失值则不处理，最终将归一化后的数据元组输出即可。输出形式为 `context.write(null, new Text(line));` 。
Map 函数的输入：
类型：Mapper<LongWritable, Text, Text, Text>
从左到右依次是：文件的行数 文件的一行的内容 map 返回的 key 属性 user_career, value 值为文件的一行内容
输入:Mapper 的前两个参数
输出:Mapper 的后两个参数
操作:
将所有的数据转换成<user_career, recorder>的键值对
Reduce 函数的输入：
类型：Reducer<Text, Text, Text, Text>

从左到右依次是: 属性值 user_career,文件的一行的内容,null,文件一行的内容
输入:Reducer 的前两个参数
输出:Reducer 的后两个参数
操作:对所有的同一个user_career 的数据,将其传入Recorder 类的构造函数中,然后构造函数会对数据项中的数据进行单位,格式的标准化,然后调用normalizeRating(Double rating)函数对 rate 的值进行归一化.归一化的公式:(rate-min)/(max-min)

## 3.4 数据清洗

**填充算法：**缺失值有 rating 和 user_income，其中 rating 近似近似依赖于 user_income、longitude、latitude 和 altitude，user_income 近似依赖于 user_nationality 和 user_career。所以要先对 user_income 值进行填充才能对 rating 值进行填充，故分为两个步骤。

先来看 user_income 步骤：
我在这里的基本思路是让 user_nationality 和 user_career 作为 map 阶段的 key，使相同职业和国籍的人分在一起，然后在 reduce 过程中迭代计算，对于缺失值的元组，他们的 income 应该取相同职业且相同国籍的人的收入的平均值，对于那些没有相同国籍相同职业的缺失值，他们的 income 取 0 值。
Map 阶段：
把 user_nationality 和 user_career 重新构成一个字符串作为 map 阶段的 key，原数据元组作为 value 即可。
Reduce 阶段：
先创建一个 list 变量，来存储相同 key 中 income 缺失的数据元组，对于不缺失的元组，记录他们的 income 总和 sum 以及他们的数量 tmp，以此来算出平均值。先迭代添加缺失值元组进入 list 中并进行 sum 和 tmp 的统计。之后，遍历 list 变量，对于每个缺失的数据元组构造新的数据元组并输出。输出形式为 `context.write(null, new Text(line));` 。
如果未缺失元祖的数量为 0，则把缺失值填补为 0，否则填为 income 平均值即可。

再来看 rating 步骤：
我在这里的思路是，对于拥有相似的 longitude、latitude、altitude 和 income 的数据元组，他们的 rating 也应该是相近的。所以可以按照 longitude、latitude、altitude 的近似值作为 key 进行分组，然后在 reduce 阶段根据处理的元组的 income 值去筛选拥有相同的 key 并且相似 income 的数据元组去求缺失值的 income。
Map 阶段：
先将数据元组的 longtitude、latitude、altitude 三个数据按照四舍五入的规则进行取整，然后将取整后的数据连接起来构造为 map 阶段的 key，将原数据元组作为 value 即可。
Reduce 阶段：
构造两个 list 变量 LostRate 和 CompleteRate，一个存储缺失 rating 值的元组，一个存储不缺失的元组。先迭代 values 一遍，将两者分别存储。然后遍历 LostRate，在其中遍历 ConpleteRate，比较遍历到的未缺失数据元组和这个缺失数据元组的 income 差值，如果绝对值小于 200，则统计其 rating 值的总和 sum 和满足条件的未缺失数据元祖的数量 tmp，以此来计算平均值。如果 tmp 是 0，则将缺失值补充为 0；如果不为 0，则补充为平均值即可。

此法，对于每个不同的缺失数据元组，可能都会有不同的填充值。

## 四、实验心得

在实验环境的搭建过程中，遇到了不少问题，比如 hadoop 如何单机部署和运行，以及 hdfs 如何启动和上传文件,这些通过仔细查阅 hadoop 的官方文档来解决。

但是每次运行的结果文件输出后，下一次运行就会遇到写出的错误，需要在每次运行开始前把上一次的运行结果删除。

通过这次实验，我熟练了数据预处理的几个基本而重要的过程以及常用的方法,进一步熟悉了 map reduce 的计算框架编程方法。