哈尔滨工业大学

<<大数据分析>>

实验报告之一

(2019 年度春季学期**)**

姓名:	姜思琪
学号:	**1160300814**
学院:	计算机学院
教师:	杨东华、王金宝

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

Windows10 下虚拟机 VMware，伪分布式 Spark 环境

三、实验过程及结果

3.1 数据抽样

使用分层抽样的方法从 D 中抽取样本， 使用 user_career 作为分层变量，然后进行分层抽样。

在 Spark 中，可以利用 sampleByKey 来进行抽样，sampleByKey 先将每一个 key 相同的聚合在一起，然后在相同的 key 之间按照指定的权重筛选，自定义一个 Map 来确定每个 key 筛选的比例，关键代码如下：

val fractions: Map[String, Double]=List(("teacher", 0.01), ("writer", 0.01), ("programmer", 0.01), ("farmer", 0.01), ("accountant",0.01), ("artist", 0.01),("Manager", 0.01),("doctor", 0.01)).toMap

sampleByKey(false,fractions,0)

输入：所有的数据记录，其中数据之间用” | ”隔开，每一条记录用换行隔开。

输出：此函数返回用 user_career 作为分层变量，抽样比例为 1%，进行分层抽样之后的所有记录。每一条数据的属性不变化。

其中，fractions 表示在每一层采样概率是 0.01 ，user_career 是表示职业的，分为"teacher", "writer", "programmer", "farmer", "accountant", "artist", "Manager", "doctor" 层，sampleByKey 中 false 表示不重复抽样，0 表示随机的 seed（seed 表示随机初始化种子）。

3.2 数据过滤

根据数据集合 D 中的属性 B 过滤包含奇异值的数据。

在样本文件 D_Sample 中统计属性 D.B 的取值分布，假设 D.B 取值排序后前 1%和最后 1%均为奇异值，获取奇异值的临界值（D.B 取值排名 1%和 99%的数值），在原始数据集合 D 中过滤掉奇异值对应的数据。

在此节内容中，分为两项：

3.2.1 属性 longitude 和 latitude 的有效范围分别为[8.1461259, 11.1993265]和[56.5824856, 57.750511]，要求过滤掉无效的数据项；

在 Spark 中，可以利用 filter()来实现过滤。定义一个函数 Tuple 来返回某条记录是否符合以上的范围：

```
def Tuple(l:(String, Double, Double, Double, String, Double, Double, String, String, String, String, Double, Boolean)): Boolean = {
    return l._2 >= 8.1461259 && l._2 <= 11.1993265 && l._3 >= 56.5824856 && l._3 <= 57.750511
  }
```

输入：分层抽样之后的每一条数据记录。

输出：返回 Boolean 值，true 表示此条数据记录在范围内，符合要求；false 表示此条记录不符合此范围，将要被过滤掉的。

再利用函数 filter 将 true 挑选出来即可。

3.2.2 假设 rating 在前 1%和最后 1%为作弊评分和恶意评分，要求过滤掉作弊评分和恶意评分数据。

在此部分中，我利用 sortByKey 函数将数据排序，定义一个变量 numall 来统计一共的记录条数，便于计算前 1%和后 1%分别为多少，再将其过滤掉。

但是，因为 rating 存在缺失的情况，所以在读入每条数据之后，先判断 rating 是否为空，如果是空值，则将 rating 值设为-1，再将其排序；过滤时，如果遇到值为-1，则按 1%的概率将其过滤掉，关键代码如下：

```
def medium(t:(Int, Double, Any), alln : Long):Boolean={
    if(t._2 == -1){//rating = null
      val randomnum = scala.util.Random.nextInt(100)
      randomnum >= 2 && randomnum <= 98
```

```
    }else{//select 4she5ru
      t._1 < (alln*0.99-0.5).toInt && t._1 > (alln*0.1+0.5).toInt
    }
  }
```

输入：(Int, Double, Any) 第一个参数表示顺序，第二个参数表示 rating 值，第三个参数表示整条记录。

输出：返回 Boolean 值，true 表示不在前 1%和后 1%，不被过滤掉；false 表示在前 1%或后 1%，过滤掉。

3.3 数据标准化和归一化

根据数据集合 D 中每个属性给定的单位，对 D_Filtered 中数据进行标准化。随后，对数据集合 D_Filtered 中的数据进行归一化。

3.3.1 属性 user_birthday 和 review_date 中日期字段随机使用 2018-03-21、2018/03/21、March 21, 2019 这三种格式；

在此部分中，我定义日期字段的标准格式是 yyyy-mm-dd，用” - ”隔开数字。我定义了日期字段标准化的函数 Userbir：

```
def Userbir(t: String): String ={
    var retu = ""
    if(t.contains("/")|| t.contains("-")){
      val sp = t.split("-|\\/")
      retu = sp(0) + "-" + sp(1) + "-" + sp(2)
    }else{
      val sp = t.split(",|\\s")
      if(sp(0).equals("January")){
        sp(0) = "01"
      }else if(sp(0).equals("February")){
        sp(0) = "02"
      }else if(sp(0).equals("March")){
        sp(0) = "03"
      }else if(sp(0).equals("April")) {
```

```
            sp(0) = "04"
        }else if(sp(0).equals("May")) {
            sp(0) = "05"
        }else if(sp(0).equals("June")){
            sp(0) = "06"
        }else if(sp(0).equals("July")){
            sp(0) = "07"
        }else if(sp(0).equals("August")){
            sp(0) = "08"
        }else if(sp(0).equals("September")){
            sp(0) = "09"
        }else if(sp(0).equals("October")){
            sp(0) = "10"
        }else if(sp(0).equals("November")){
            sp(0) = "11"
        }else if(sp(0).equals("December")){
            sp(0) = "12"
        }else{
            println("ERROR"+sp(0))
            return "ERROR"
        }
        if(sp(1).toInt < 10){
            sp(1) = "0".concat(sp(1))
        }
        retu = sp(2) + "-" + sp(0) + "-" + sp(1)
    }
    //print(retu)
    return retu
  }
```

输入：String 任何形式的日期字段

输出：String 用”–“ 间隔的形式

3.3.2 temperature 有华氏和摄氏两种，这两方面要求进行标准化。属性 rating 需要进行归一化，转换为 0~1。

通过公式摄氏温度 =（华氏温度-32）/1.8，temperature 均用摄氏温度表示。定义函数 Temp：

```
def Temp(t:String):Double={
    var retu = 0.0
    if(t.endsWith("℃")){
      retu = t.substring(0,t.length()-1).toDouble
    }else if(t.endsWith("℉")) {
      retu = (t.substring(0, t.length()-1).toDouble - 32.0)/1.8
    }
    else{
      retu = -273.0
    }
    return retu
  }
```

输入：String 温度字段

输出：Double 摄氏温度表示的字段

3.4 数据清洗

数据集合 D 中存在缺失值，其过滤之后的版本 D_Filtered 中也存在缺失值，使用一种缺失值估计的方法，对 D_Filtered 中的缺失值进行填充，并将填充后的数据集合保存在 HDFS 中。

属性 rating 和 user_income 存在少量缺失值，分为以下两部分：

3.4.1 根据先验知识，rating 近似依赖于 user_income、longitude、latitude 和 altitude（收入相近的人对同一地点的评价打分接近）；

首先，将所有未空的记录归类，将已空的记录按照经纬度和来进行连接，再

计算海拔和收入差，按照从小到大排序，取前 50 个数据求平均值，从而填充空值。关键代码如下：

```
val ratinglib = ratinttype.map(x => ((x._1._1).toInt + (x._1._2).toInt, (x._1._3, x._2)))

val rating = containnull.filter(x => x._7 < 0).map(x => ((x._2, x._3, x._4), x._12, x)).map(x => ((x._1._1).toInt + (x._1._2).toInt, (x._1._3, x._2, x._3))) .join(ratinglib).map(x => {

        ((x._2._1._1 - x._2._2._1, x._2._1._2 - x._2._2._2), (x._2._1._3, x._2._2._2))
// sort, tuple, rating sample

}).sortByKey(true).take(50).map(x => (x._2._1, x._2._2) ).groupBy(_._1).map(x => (x._1, x._2.map(x => x._2).sum/x._2.length)) .map(x => (x._1._1, x._1._2, x._1._3, x._1._4, x._1._5, x._1._6, x._2, x._1._8, x._1._9, x._1._10, x._1._11, x._1._12, x._1._13))
```

3.4.2 user_income 近似依赖于 user_nationality 和 user_career。对 rating 和 user_income 的填充可以利用这些依赖关系（同一国家同一职业的人收入接近）。

通过查询资料和与同学探讨，了解到可以采取完全聚类后取平均值的做法，数据点是离散的，我首先根据未空的数据记录根据属性 user_nationality 和 user_career 聚类，然后将含空值的记录进行匹配，找到相等的类别，用平均值进行填充。关键代码如下：

```
val incomelib = trueall.map(x => ((x._10,x._11),x)).groupByKey().map(x => (x._1,x._2.map(x => x._12).sum/x._2.map(x=>1).sum))

val income = containnull.filter(x => x._12 < 0).map(x => ((x._10,x._11),x)).join(incomelib).map( y => {
            val x = y._2._1
            (x._1, x._2, x._3, x._4, x._5, x._6, x._7, x._8, x._9, x._10, x._11, y._2._2, x._13)
        })
```

4. 思考和加分实验内容

我将 3.1-3.3 中的数据预处理尽量整合一起，在对 user_career 进行分层抽样之前，读入数据时，便进行大部分操作：对属性 longitude 和 latitude 过滤掉无效的数据项；将属性 user_birthday 和 review_date 中日期字段随机使用” - “的标准格

式；同时将属性 temperature 全改为摄氏度的形式；对属性 rating 需要进行归一化，转换为 0~1。在这之后，再去掉属性前 1%和后 1%的值，最后进行空值填充，数据清洗。整合的关键代码如下：

```
val fractions: Map[String, Double]=List(("teacher", 0.01), ("writer", 0.01),
( "programmer", 0.01),
        ("farmer", 0.01), ("accountant",0.01), ("artist", 0.01),("Manager",
0.01),("doctor", 0.01)).toMap
    val input2 = input.map(line => {
      val l = line.split("\\|")//fen ge
      var flag = true
      for(i<-0 until l.length){
        if(l(i).contains("?")){//pan duan shi fou you null
          flag = false
        }
      }
      val tuple = (l(0), l(1).toDouble, l(2).toDouble, l(3).toDouble,
Userbir(l(4)), Temp(l(5)),
        Rat(l(6)), l(7), Userbir(l(8)), l(9), l(10),
        if(l(11).contains("?")) -1.0 else l(11).toDouble, flag)
      (tuple.copy(), Tuple(tuple))
      //bu fen guo lv
    }).filter(_._2).map(x => (x._1._11,
x._1 )).sampleByKey(false,fractions,0)
      .map(x => (x._2._7,x._2)).sortByKey(false)
```

对于数据清洗的优化，我通过查询资料和询问同学得知，rating 的影响分布函数可看作三个函数：经纬度对 rating 的影响，海拔对 rating 的影响和收入对 rating 的影响。其中在地球上，根据经纬度计算距离有公式，可以调参进行优化。

已知 A(m,n)，B(x,y)经纬度，求两点的距离公式如下：

$$\Delta L = \sqrt{(R\frac{|m-x|*\pi}{180})^2 + (R*cos(\frac{m*\pi}{180})\frac{|n-y|*\pi}{180})^2}$$

其中，因为经纬度范围给出，所以我用平均值数估算值，地球半径 R=6470km。调参取值 1/100，1/70，通过测试，取 1/70 效果更好一点。

测试函数：

```
val proj_bucket.map(x => (((x._1._1-8.1461259)*70*400).toInt + ((x._1._2-56.5824856)*70).toInt, 1)).reduceByKey(_+_).foreach(println)
```

优化后的 rating 空值填充代码如下：

```
val rating = containnull.filter(x => x._7 < 0).map(x => ((x._2, x._3, x._4), x._12, x)).map(x => (((x._1._1-8.1461259)*70*400).toInt + ((x._1._2-56.5824856)*70).toInt, (x._1._3, x._2, x._3))).join(dict_fame).map(x => {
                ((x._2._1._1 - x._2._2._1, x._2._1._2 - x._2._2._2), (x._2._1._3, x._2._2._2)) // sort, tuple, rating sample
            }).sortByKey(true).take(5).map(x =>
              (x._2._1, x._2._2)
            ).groupBy(_._1).map(x => (x._1, x._2.map(x => x._2).sum / x._2.length)).map(x => (x._1._1, x._1._2, x._1._3, x._1._4, x._1._5, x._1._6, x._2, x._1._8, x._1._9, x._1._10, x._1._11, x._1._12, x._1._13))
```

四、实验心得

通过本次实验，我更熟悉里 Spark 和 scalar，它们处理大量数据上有显著的优势。而且 scalar 语句很简洁，功能也强大，对于刚入门的新手很友好。

搭配环境是一个需要摸索的过程。多次遇到版本不合适的问题或没有完整的配置教程，这里会需要许多时间。

在编写代码时，有一些函数具体用法的小区别，比如 count 的用法，表达式 _._1 代表匹配第一个属性等用法。

在数据清洗中，采用 KNN 算法和分桶办法实现。