**UPQuality设计文档**

**一 背景**

UPQuality的目标是在UserProperty数据的基础之上，对UP库中各个字段属性与基准库中的数据进行对比，检验UP库中数据的正确性

**二 UPQuality流程**

程序基于Scala编写spark运行

1）. 分别读取基准数据与UP库数据，格式为RDD[(String , String)]，形成RDD[(id , up)]结构

注：

1.1之所以up采用String格式存储，是因为up构建时未继承序列化接口，进行map等RDD操作时会报错 ”Task not serializable”，所以暂时使用String格式存储

1.2 读取UP库数据时，如果基准数据具体属于某个分区(urs, cookie, device, daid ,third , phone)，可以对其进行路径优化，减少UP库数据量）

2）. 将基准数据与UP库数据进行join，格式为RDD[(String , (String , String))] , 形成RDD(id , (up , up))结构

注：

2.1因为基准数据与UP库数据相比数据量较小，所以基准数据在前，UP库数据在后；

3）. 利用策略模式对各个Key所对应的策略方法进行构建，通过spring进行管理，格式为Map[UPKey , IStragety] , 其中IStragety是所有策略类的超类

4）. 对于同一行数据，遍历基准数据UP中的propsList，根据其中的key , source找到UP数据中相应key , source对应的value，根据Key所对应的判断规则进行判断，并计数，输出为Map[String , BigInt]格式

5）. 在driver端，对每个检测字段相应数据源的每个结果的正确率进行统计，并输出，以parquet格式存储，保留2位小数

举例：

key = UPKey.Gender

最终输出结果：{属性,数据源，正确率，正确个数}

{"key":"gender\_M\_True","source":"love","accuracy":"100.00%" , accNum: "2"}

{"key":"gender\_F\_True","source":"love","accuracy":"100.00%" , accNum: "3"}

{"key":"gender\_True","source":"love","accuracy":"100.00%" , accNum: "5"}

流程图：

基准数据

UP库数据

解析成RDD[(id , stdUp)]结构

Join操作，形成RDD(id , (stdUp , up)) 结构

解析成RDD[(id , up)]结构

对同一行数据，对基准数据和UP库数据中相应Key所对应的判断规则进行判断，并计数

在driver端，对各个检测字段的各个结果的正确率进行计算统计

**三 IStragety策略类设计**

采用策略模式对各个属性的策略类进行设计，定义抽象策略的公共实现

其中IStragety是所有策略类的超类，IStragety的输出为Map[String , BigInt]，

Map的key格式为property\_result:source，即属性（\_属性值）\_结果（ture/falsed）：数据源，输出结果为对相应的Key中结果进行计数

举例：

key = UPKey.gender

value = [F , M]

GenderStragety为继承IStragety的Gender属性的实现类，

的输出为

Map(

"Gender\_F:love" -> 3 , //表示数据源love属性Gender值为F的个数为3

"Gender\_True:love " -> 5 , //表示数据源love属性Gender总共正确的个数为5

"Gender\_M:love " -> 2 , //表示数据源love属性Gender值为M的个数为2

"Gender\_M\_True:love " -> 2 , //表示数据源love属性Gender值为M正确的个数为2

"Gender:love " -> 5 , //表示数据源love属性Gender值总共的个数为5

"Gender\_F\_True:love " -> 3 //表示数据源love属性 Gender值为F正确的个数为3

)

将策略类的管理通过strageties.xml进行配置，并通过spring进行管理。其中strageties.xml的内容如下

