第 5 章 网站转化率统计

学习目标

- 了解数据集分析，能够描述用户浏览网站页面的行为中包含的信息；
- 熟悉实现思路分析，能够描述页面单跳转化率统计的实现思路；
- 掌握实现网站转化率统计，能够编写用于实现网站转化率统计的 Spark 程序；
- 掌握运行 Spark 程序，能够将 Spark 程序提交到 YARN 集群运行。

网站转化率是一个广义的概念，表示访问网站的用户中执行期望目标行动用户与所有用户的比例。目标行动可以包括浏览商品、购买商品或用户注册等。其中，页面单跳转化率则是网站转化率的一种具体表现形式，用于衡量用户在访问一个页面后执行预期目标动作的概率。通过对页面单跳转化率的统计，可以优化页面的布局和营销策略，提高用户在网站上的深度浏览。本章将讲解如何对电商网站的用户行为数据进行分析，从而基于页面单跳转化率统计网站转化率。

5.1 数据集分析

网站转化率统计使用的数据集为 Scala 程序模拟生成的用户行为数据。数据集中的每一行数据都记录了用户浏览网站页面的行为。下面，以数据集中的一条用户行为数据为例进行详细分析，具体内容如下。

```
    {"action_time":"2023-11-06 10:41:13",
"session_id":"dfadda377e2a4733a5c9ebc0d53be6e0","page_id":10,"user_id":24}
```

从上述内容可以看出，数据集中的每一条用户行为数据都以 JSON 对象的形式存在。该对象包含多个键值对，每个键值对代表着不同的信息。下面，通过解读这些键介绍用户行为数据中各项信息的含义。

- action_time：表示用户访问网站的时间。
- session_id：表示用户每次访问网站时生成的唯一标识。
- page_id：表示用户访问网站页面的唯一标识。
- user_id：表示用户的唯一标识。

5.2 实现思路分析

用户在浏览网站页面时,若从当前正在浏览的页面 A 跳转到另一页面 B 进行浏览,则被视为用户完成了一次由页面 A 到页面 B 的单向跳转。例如,统计由页面 A 到页面 B 的页面单跳转化率,计算公式如下。

页面单跳转化率=由页面 A 到页面 B 的单向跳转总次数/页面 A 的总访问次数

下面,通过图 5-1 详细描述本项目中网站转化率统计的实现思路。

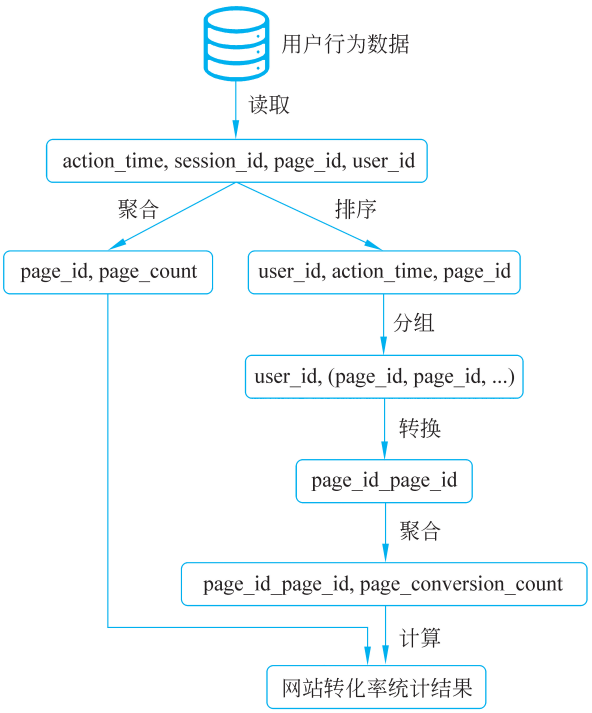

图 5-1 网站转化率统计的实现思路

针对网站转化率统计的实现思路进行如下讲解。

- 读取:读取模拟生成的用户行为数据。
- 聚合:统计每个页面被访问的次数(page_count)。
- 排序:提取用户访问网站的时间(action_time)、用户访问网站页面的唯一标识(page_id)和用户的唯一标识(user_id),并根据用户访问网站的时间进行升序排序,以获取用户访问页面的先后顺序。
- 分组:对排序结果进行分组处理,按访问的先后顺序排列每个用户访问的所有页面。
- 转换:将每个用户访问的所有页面中相邻页面的唯一标识转换为 page_id_ page_id 的形式。例如,如果相邻页面的唯一标识为 5 和 6,则转换结果为 5_6,表示由唯一标识为 5 的页面到唯一标识为 6 的页面的单向跳转。

- 聚合：统计不同页面之间单向跳转的次数(page_conversion_count)。
- 计算：将页面被访问的次数和该页面到另一个页面单向跳转的次数代入计算页面单跳转化率的公式中，计算页面单跳转化率。例如，如果唯一标识为 5 的页面被用户访问了 100 次，单向跳转 5_6 的次数为 50，那么页面单跳转化率为 50/100＝0.5，即 50%。其意义是所有访问唯一标识为 5 的页面的用户中，有 50%的用户继续浏览了唯一标识为 6 的页面。

5.3　实现网站转化率统计

5.3.1　生成用户行为数据

在项目 SparkProject 的 src/main/scala 目录中，新建了一个名为 cn.itcast.conversion 的包。在 cn.itcast.conversion 包中创建一个名为 GenerateData 的 Scala 单例对象，在该单例对象中实现模拟生成用户行为数据，具体代码如文件 5-1 所示。

文件 5-1　GenerateData.scala

```
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.{FileSystem, Path}
import org.json.JSONObject
import scala.util.Random
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
object GenerateData{
  def generateData(): Unit ={
    //指定具有 HDFS 操作权限的用户 root
    System.setProperty("HADOOP_USER_NAME","root")
    val random =new Random()
    val dateTimeFormatter =
              DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")
    //指定模拟生成用户访问网站的起始时间为 2024-03-07 00:00:00
    val startDate =LocalDateTime.of(2024, 3, 7, 0, 0, 0)
    val conf =new Configuration()
    //配置 HDFS 服务的地址
    conf.set("fs.defaultFS","hdfs://192.168.88.161:9000")
    val fs =FileSystem.get(conf)
    //指定写入用户行为数据的文件路径/page_conversion/user_conversion.json
    val outputPath =new Path("/page_conversion/user_conversion.json")
    val outputStream =fs.create(outputPath)
    for (_ <-1to 10000) {
      val randomSeconds =random.nextInt(86400)
      //模拟生成用户访问网站的时间
```

```
26        val actionTime =startDate.plusSeconds(randomSeconds.toLong)
27        //模拟用户每次访问网站时生成的唯一标识
28        val sessionId =java.util.UUID.randomUUID().toString
29        //模拟生成用户访问网站页面的唯一标识
30        val pageId =1+random.nextInt(10)
31        //模拟生成用户的唯一标识
32        val userId =1+random.nextInt(100)
33        val visitData =new JSONObject()
34        visitData.put("action_time", dateTimeFormatter.format(actionTime))
35        visitData.put("session_id", sessionId)
36        visitData.put("page_id", pageId)
37        visitData.put("user_id", userId)
38        //将模拟生成的用户行为数据写入 HDFS 的指定文件中
39        outputStream.writeBytes(visitData.toString +"\n")
40      }
41      outputStream.close()
42      fs.close()
43    }
44    def main(args: Array[String]): Unit ={
45      generateData()
46    }
47  }
```

上述代码中,第 23～40 行代码通过 for 循环模拟生成 10000 条用户行为数据,其中第 33～37 行代码,用于将模拟生成的用户行为数据调整为 JSON 对象的格式。

确保 Hadoop 集群正常启动后,运行文件 5-1。当文件 5-1 运行完成后,查看 HDFS 的目录/page_conversion 中文件 user_conversion.json 的内容。在虚拟机 Spark01 执行如下命令。

```
$hdfs dfs -cat /page_conversion/user_conversion.json
```

上述命令执行完成的效果如图 5-2 所示。

图 5-2 展示了文件 user_conversion.json 的部分数据,这些数据记录了用户浏览网站页面的行为。因此说明,成功将模拟生成的用户行为数据写入了 HDFS。

5.3.2 实现 Spark 程序

在项目 SparkProject 的包 cn.itcast.conversion 中新建一个名为 PageConversion 的 Scala 单例对象,在该单例对象中实现网站转化率统计的 Spark 程序,具体实现过程如下。

(1) 由于本项目需要通过 Spark SQL 实现网站转化率统计,所以在实现 Spark 程序之前,需要在项目 SparkProject 中添加 Spark SQL 依赖。在配置文件 pom.xml 的<dependencies>标签中添加如下内容。

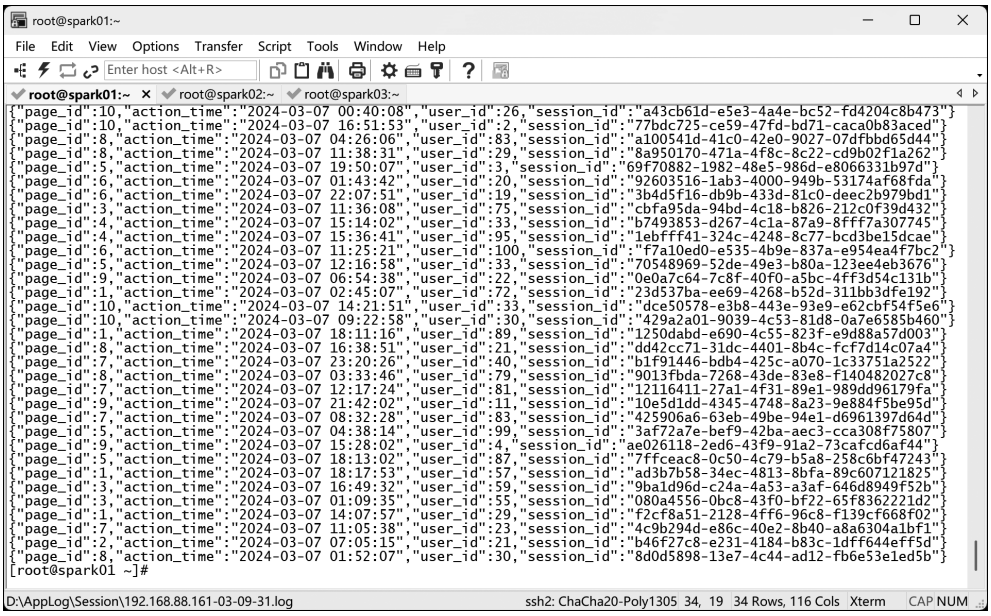

图 5-2　查看文件 user_conversion.json 的内容

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>
    <version>3.3.0</version>
</dependency>
```

上述内容添加完成后，在 IntelliJ IDEA 的 Maven 窗口中确认 Spark SQL 依赖是否存在于项目 SparkProject 中。

（2）在单例对象 PageConversion 中添加 main()方法，用于定义 Spark 程序的实现逻辑，具体代码如文件 5-2 所示。

文件 5-2　PageConversion.scala

```
package cn.itcast.conversion
object PageConversion {
  def main(args: Array[String]): Unit = {
    //实现逻辑
  }
}
```

（3）在 Spark 程序中创建 SparkSession 对象 spark，用于配置并管理 Spark 程序的执行。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
val spark = SparkSession.builder()
  .appName("PageConversion")
```

```
3   .getOrCreate()
```

上述代码指定 Spark 程序的名称为 PageConversion。

(4) 在 Spark 程序中,通过 DataFrame API 提供的 json()方法从 HDFS 中读取 JSON 格式的用户行为数据,并将读取的数据注册为临时视图 conversion_table。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
spark.read.json(args(0)).createOrReplaceTempView("conversion_table")
```

上述代码中,使用 args(0)代替用户行为数据的具体路径,以便将 Spark 程序提交到 YARN 集群运行时,可以更加灵活地通过 spark-submit 命令的参数来指定用户行为数据的具体路径。

(5) 在 Spark 程序中,通过 DataFrame API 提供的 sql()方法执行 SQL 语句,基于临时视图 conversion_table 统计每个页面被访问的次数,并将 SQL 语句的执行结果存储到 DataFrame 对象 pageVisitStatsDF 中。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
1   val pageVisitStatsDF = spark.sql(
2         "select page_id," +
3                 "count(*) as page_count " +
4         "from conversion_table " +
5         "group by page_id"
6       )
```

上述代码中的 SQL 语句使用了 group by 子句,按照字段 page_id 进行分组。对于每个分组,使用了 count()函数计算该分组中记录的数量,并将计算结果作为新的字段 page_count 返回,该字段中记录了每个页面被访问的次数。

(6) 在 Spark 程序中,通过 DataFrame API 提供的 sql()方法执行 SQL 语句,基于临时视图 conversion_table 按照用户访问网站的时间进行升序排序,并将 SQL 语句的执行结果注册为临时视图 conversion_sort_table。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
1   spark.sql(
2           "select user_id," +
3                   "action_time," +
4                   "page_id " +
5           "from conversion_table " +
6           "order by user_id,action_time")
7         .createOrReplaceTempView("conversion_sort_table")
```

上述代码中的 SQL 语句使用了 order by 子句,按照字段 user_id 和 action_time 进行升序排序,获取用户访问页面的先后顺序。

(7) 在 Spark 程序中,通过 DataFrame API 提供的 sql()方法执行 SQL 语句,基于临时视图 conversion_sort_table 合并每个用户访问的所有页面,并将 SQL 语句的执行结果存储到 RDD 对象 pageConversionRDD 中。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
val pageConversionRDD: RDD[Row] =
      spark.sql(
          "select user_id," +
            "concat_ws(',',collect_list(page_id)) as page_list " +
          "from conversion_sort_table " +
          "group by user_id").rdd
```

上述代码中的 SQL 语句使用 group by 子句,按照字段 user_id 进行分组。对于每个分组,首先,使用 collect_list()函数将字段 page_id 的值合并到一个列表。然后,使用 concat_ws()函数将列表中的每个值合并为一个逗号分隔的字符串,并将合并结果作为新的字段 page_list 返回,该字段中记录了用户访问的所有页面的唯一标识。

(8) 在 Spark 程序中,通过 flatMap 算子对 RDD 对象 pageConversionRDD 进行转换操作,并将转换操作的结果存储到 RDD 对象 rowRDD 中。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
val rowRDD: RDD[Row] =pageConversionRDD.flatMap {
  row =>
    //创建集合 list,用于存储页面唯一标识转换为单向跳转的形式
    val list: ListBuffer[Row] =new ListBuffer[Row]()
    val page: Array[String] =row.getString(1).split(",")
    for (i <-0 until page.length -1) {
      if (page(i) !=page(i +1)) {
        val pageConversionStr: String =page(i) +"_" +page(i +1)
        list +=RowFactory.create(pageConversionStr)
      }
    }
    list.iterator
}
```

上述代码中,第 5 行代码用于获取记录用户访问的所有页面的唯一标识的字符串,并通过逗号将其拆分为数组 page。第 6～11 行代码用于遍历数组 page,获取每个页面唯一标识,如果两个相邻页面的唯一标识不相等,则将它们通过字符“_”合并为字符串,并添加到集合 list 中。

(9) 在 Spark 程序中创建 StructType 对象 schema,用于指定数据结构。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
val schema: StructType =DataTypes.createStructType(
```

```
    Array(
      DataTypes.createStructField(
        "page_conversion",
        DataTypes.StringType,
        true
      )
    )
  )
```

上述代码指定的数据结构包含一个名为 page_conversion 的字段,该字段的数据类型为 String 并且允许值为空。

(10) 在 Spark 程序中,基于 StructType 对象 schema 中指定的数据结构,将 RDD 对象 rowRDD 转换为 DataFrame 对象,并将其注册为临时视图 page_conversion_table。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
spark.createDataFrame(rowRDD, schema)
      .createOrReplaceTempView("page_conversion_table")
```

(11) 在 Spark 程序中,通过 DataFrame API 提供的 sql()方法执行 SQL 语句,基于临时视图 page_conversion_table 统计不同页面之间单向跳转的次数,并将 SQL 语句的执行结果注册为临时视图 page_conversion_count_table。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
spark.sql(
    "select page_conversion," +
            "count(*) as page_conversion_count " +
    "from page_conversion_table " +
    "group by page_conversion"
).createOrReplaceTempView("page_conversion_count_table")
```

上述代码中的 SQL 语句使用了 group by 子句,按照字段 page_conversion 进行分组。对于每个分组,使用 count()函数计算该分组中记录的数量,并将计算结果作为新的字段 page_conversion_count 返回,该字段记录了不同页面之间单向跳转的次数。

(12) 在 Spark 程序中,通过 DataFrame API 提供的 sql()方法执行 SQL 语句,对临时视图 page_conversion_count_table 中字段 page_conversion 的值进行转换,并将 SQL 语句的执行结果存储到 DataFrame 对象 pageConversionCountDF 中。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
val pageConversionCountDF = spark.sql(
  "select page_conversion_count," +
          "split(page_conversion,'_')[0] as start_page," +
```

```
4              "split(page_conversion,'_')[1] as last_page " +
5    "from page_conversion_count_table"
6  )
```

上述代码中的 SQL 语句使用 split()函数，根据字符“_”将字段 page_conversion 的内容拆分为两部分，并分别将这两部分作为新的字段 start_page 和 last_page 返回。其中，字段 start_page 记录了不同页面之间单向跳转的起始页面的唯一标识；字段 last_page 记录了不同页面之间单向跳转的结束页面的唯一标识。

(13) 在 Spark 程序中，通过 join 算子对 DataFrame 对象 pageConversionCountDF 和 pageVisitStatsDF 进行左外连接操作，并将操作结果注册为临时视图 page_conversion_join。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
1  pageConversionCountDF.join(
2    pageVisitStatsDF,
3    col("start_page").equalTo(col("page_id")),
4    "left"
5  ).createOrReplaceTempView("page_conversion_join")
```

上述代码中指定的连接条件为 DataFrame 对象 pageConversionCountDF 中字段 start_page 的值等于 DataFrame 对象 pageVisitStatsDF 中字段 page_id 的值。通过对 DataFrame 对象 pageConversionCountDF 和 pageVisitStatsDF 进行左外连接，可以在临时视图 page_conversion_join 中将页面的总访问次数和该页面跳转到其他页面的单跳次数相关联，便于后续通过这两个值计算页面单跳转化率。

(14) 在 Spark 程序中，通过 DataFrame API 提供的 sql()方法执行 SQL 语句，计算页面单跳转化率，并将 SQL 语句的执行结果存储到 DataFrame 对象 resultDF 中。在单例对象 PageConversion 的 main()方法中添加如下代码。

```
1  val resultDF = spark.sql(
2    "select concat(page_id,'_',last_page) as conversion," +
3  "round(CAST(page_conversion_count AS DOUBLE)/CAST(page_count AS DOUBLE)," +
4      "5) as rage " +
5      "from page_conversion_join"
6  )
```

上述代码中的 SQL 语句，首先使用 concat()函数将字段 page_id 和 last_page 的值通过字符“_”拼接为新的字符串，并将其作为新的字段 conversion 返回，该字段中的记录为不同页面之间单向跳转的形式。然后，使用 cast()函数将字段 page_conversion_count 和 page_count 的值转换为 Double 类型，并对这两个字段的值进行除法运算，获取页面单跳转化率。最后，使用 round()函数对除法运算的结果四舍五入到小数点后五位，并将其作为新的字段 rage 返回。

5.3.3 数据持久化

通过记录历史事件、文化传统和社会变迁,有助于人们对历史的正确理解和认识,从而在思想上得到启迪和教育。上一节内容实现的 Spark 程序仅仅获取了网站转化率的统计结果。为了便于后续进行数据可视化,并确保分析结果的长期存储,需要进行数据持久化操作。本项目使用 HBase 作为数据持久化工具。接下来,分步骤讲解如何将网站转化率的统计结果存储到 HBase 的表中,具体操作步骤如下。

(1) 在单例对象 PageConversion 中定义一个 conversionToHBase()方法,该方法用于向 HBase 的表 conversion 中插入网站转化率的统计结果,具体代码如下。

```
1  def conversionToHBase(dataframe: DataFrame): Unit ={
2    //在 HBase 中创建表 conversion 并向表中添加列族 page_conversion
3    HBaseUtils.createtable("conversion","page_conversion")
4    //创建数组 column,用于指定列标识的名称
5    val column =Array("convert_page", "convert_rage")
6    dataframe.foreach {
7      row =>
8        //获取不同页面之间的单向跳转
9        val conversion =row.getString(0)
10       //获取页面单跳转化率
11       val rage =row.getDouble(1).toString
12       //创建数组 value,用于指定插入的数据
13       val value =Array(conversion, rage)
14       HBaseUtils.putsToHBase(
15         "conversion",
16         conversion +rage,
17         "page_conversion",
18         column,
19         value
20       )
21   }
22 }
```

上述代码中,第 14～20 行代码用于向 HBase 中表 conversion 的列 page_conversion:convert_page 和 page_conversion:convert_rage 插入数据,数据的内容依次为不同页面之间的单向跳转和页面单跳转化率。

(2) 在单例对象 PageConversion 的 main()方法中调用 conversionToHBase()方法并将 resultDF 作为参数传递,实现将网站转化率的统计结果插入 HBase 的表 conversion 中,具体代码如下。

```
1  try {
2    PageConversion.conversionToHBase(resultDF)
```