预处理behavior_log数据集

检查数据格式，以及对数据进行简单的统计运算(不同用户不同行为的次数)

1. 建立spark的session

In [1]:

```
import os
# 配置pyspark和spark driver运行时 使用的python解释器
JAVA_HOME = '/root/bigdata/jdk'
PYSPARK_PYTHON = '/miniconda2/envs/py365/bin/python'
# 当存在多个版本时，不指定很可能会导致出错
os.environ['PYSPARK_PYTHON'] = PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON'] = PYSPARK_PYTHON
os.environ['JAVA_HOME'] = JAVA_HOME
# 配置spark信息
from pyspark import SparkConf
from pyspark.sql import SparkSession

SPARK_APP_NAME = 'preprocessingBehaviorLog'
SPARK_URL = 'spark://192.168.58.100:7077'

conf = SparkConf()    # 创建spark config对象
config = (
    ("spark.app.name", SPARK_APP_NAME),    # 设置启动的spark的app名称，没有提供，将随机
    ("spark.executor.memory", "2g"),    # 设置该app启动时占用的内存用量，默认1g
    ("spark.master", SPARK_URL),    # spark master的地址
    ("spark.executor.cores", "2"),    # 设置spark executor使用的CPU核心数
    # 以下三项配置，可以控制执行器数量
    # ("spark.dynamicAllocation.enabled", True),
    # ("spark.dynamicAllocation.initialExecutors", 1),    # 1个执行器
    # ("spark.shuffle.service.enabled", True)
    # ('spark.sql.pivotMaxValues', '99999'),  # 当需要pivot DF，且值很多时，需要修改，
)
# 查看更详细配置及说明：https://spark.apache.org/docs/latest/configuration.html

conf.setAll(config)

# 利用config对象，创建spark session
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

2. 从hdfs上加载要处理的数据

该数据集有20G，这里仅截取其中一部分

2.1 在hdfs上查看待处理数据

In [3]:
```
!hadoop fs -ls /data
# test1.csv是behavior_log数据集截取的一部分
```

```
Found 5 items
-rw-r--r--   1 root supergroup   31286431 2020-11-05 18:06 /data/ad_feature.csv
drwxr-xr-x   - root supergroup          0 2020-11-06 15:38 /data/models
-rw-r--r--   1 root supergroup 1088060964 2020-11-05 18:06 /data/raw_sample.csv
-rw-r--r--   1 root supergroup   33812570 2020-12-12 18:39 /data/test1.csv
-rw-r--r--   1 root supergroup   24056588 2020-11-05 18:06 /data/user_profile.csv
```

2.2 将待处理数据加载到spark，查看各列数据的类型

In [5]:
```
df = spark.read.csv('/data/test1.csv',header=True)
df.show()
```

```
+------+----------+----+-----+------+
|  user|time_stamp|btag| cate| brand|
+------+----------+----+-----+------+
|558157|1493741625|  pv| 6250| 91286|
|558157|1493741626|  pv| 6250| 91286|
|558157|1493741627|  pv| 6250| 91286|
|728690|1493776998|  pv|11800| 62353|
|332634|1493809895|  pv| 1101|365477|
|857237|1493816945|  pv| 1043|110616|
|619381|1493774638|  pv|  385|428950|
|467042|1493772641|  pv| 8237|301299|
|467042|1493772644|  pv| 8237|301299|
|991528|1493780710|  pv| 7270|274795|
|991528|1493780712|  pv| 7270|274795|
|991528|1493780712|  pv| 7270|274795|
|991528|1493780712|  pv| 7270|274795|
|991528|1493780714|  pv| 7270|274795|
|991528|1493780765|  pv| 7270|274795|
|991528|1493780714|  pv| 7270|274795|
|991528|1493780765|  pv| 7270|274795|
|991528|1493780764|  pv| 7270|274795|
|991528|1493780633|  pv| 7270|274795|
|991528|1493780764|  pv| 7270|274795|
+------+----------+----+-----+------+
only showing top 20 rows
```

In [6]:
```
df.printSchema()
```

```
root
 |-- user: string (nullable = true)
 |-- time_stamp: string (nullable = true)
 |-- btag: string (nullable = true)
 |-- cate: string (nullable = true)
 |-- brand: string (nullable = true)
```

2.3 转换成可以处理的数据类型

由上步知各列都是字符串型，将用于数字处理的类型转化成int和long类型

In [7]:
```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, Long
schema = StructType([
    StructField('userId',IntegerType()),
    StructField('timestamp',LongType()),
    StructField('btag',StringType()),
    StructField('cateId',IntegerType()),
    StructField('brandId',IntegerType())
])#给每列数据 重命名+变换数据类型，相当于Row()
behavior_log_df=spark.read.csv('/data/test1.csv',header=True,schema=schema)
behavior_log_df
```

Out[7]: DataFrame[userId: int, timestamp: bigint, btag: string, cateId: int, brandId: int]

In [8]:
```
behavior_log_df.show()
```

```
+------+----------+----+------+-------+
|userId| timestamp|btag|cateId|brandId|
+------+----------+----+------+-------+
|558157|1493741625|  pv|  6250|  91286|
|558157|1493741626|  pv|  6250|  91286|
|558157|1493741627|  pv|  6250|  91286|
|728690|1493776998|  pv| 11800|  62353|
|332634|1493809895|  pv|  1101| 365477|
|857237|1493816945|  pv|  1043| 110616|
|619381|1493774638|  pv|   385| 428950|
|467042|1493772641|  pv|  8237| 301299|
|467042|1493772644|  pv|  8237| 301299|
|991528|1493780710|  pv|  7270| 274795|
|991528|1493780712|  pv|  7270| 274795|
|991528|1493780712|  pv|  7270| 274795|
|991528|1493780712|  pv|  7270| 274795|
|991528|1493780714|  pv|  7270| 274795|
|991528|1493780765|  pv|  7270| 274795|
|991528|1493780714|  pv|  7270| 274795|
|991528|1493780765|  pv|  7270| 274795|
|991528|1493780764|  pv|  7270| 274795|
|991528|1493780633|  pv|  7270| 274795|
|991528|1493780764|  pv|  7270| 274795|
+------+----------+----+------+-------+
only showing top 20 rows
```

In [9]:
```
# 再次查看数据类型是否转成我们可以处理的类型
behavior_log_df.printSchema()
```

```
root
 |-- userId: integer (nullable = true)
 |-- timestamp: long (nullable = true)
 |-- btag: string (nullable = true)
 |-- cateId: integer (nullable = true)
 |-- brandId: integer (nullable = true)
```

3. spark处理数据

3.1 分析数据集字段的类型和格式

a. 查看是否有空值
b. 查看每列数据的类型
c. 查看每列数据的类别情况

In [12]:
```
1 behavior_log_df.groupby('userId').count().show()
```

```
+-------+-----+
| userId|count|
+-------+-----+
| 668074|    7|
| 550961|    4|
| 313148|    2|
| 285987|    5|
| 369968|    1|
| 328078|    1|
| 309510|    1|
| 201031|    4|
|1097471|   15|
|1137432|    1|
| 511225|    2|
| 396208|   13|
| 913040|    9|
| 604834|    2|
| 535839|    1|
| 147711|   12|
| 304448|    4|
| 815397|    4|
| 387467|    1|
|1141195|    5|
+-------+-----+
only showing top 20 rows
```

In [10]:
```
1 # 查看总共用多少用户
2 # 第一个count()，返回的是一个dataframe，这里的count计算的是每一个分组的个数，即每个用
3 # 完整的数据集有113w个用户
4 print('查看userId的数据情况：',behavior_log_df.groupby('userId').count().count())
```

```
查看userId的数据情况： 242109
```

In [8]:
```
behavior_log_df.groupby('btag').count().show()
print('查看btag的数据情况：',behavior_log_df.groupby('btag').count().collect())#collec
```

```
+----+------+
|btag| count|
+----+------+
| buy| 13668|
| fav| 12807|
|cart| 22142|
|  pv|951383|
+----+------+
```

查看btag的数据情况： [Row(btag='buy', count=13668), Row(btag='fav', count=12807), Row(btag='cart', count=22142), Row(btag='pv', count=951383)]

In [9]:
```
print('查看cateId的数据情况',behavior_log_df.groupby('cateId').count().count())
# 商品类别
```

查看cateId的数据情况 6044

In [10]:
```
print('查看brandId的数据情况',behavior_log_df.groupby('brandId').count().count())
# 商品品牌类别
```

查看brandId的数据情况 49179

In [13]:
```
# pandas中选择某列使用df['某列的名字']，但是sparksql不能这样用，要使用sql语句，df.sele
# 可以使用df['某列的名字'].cast(某种数据类型如Longtype())
print('判断数据是否有空值：')
print('原始数据有%d行'%behavior_log_df.count())
#dropna()-某行数据有一个值是空值，就将该行删除，注意该方法不去掉str类型的null和NULL
print('去掉空值后数据有%d行'%behavior_log_df.dropna().count())
# 根据结构可知：没有空值
```

判断数据是否有空值：
原始数据有1000000行
去掉空值后数据有1000000行

3.2 统计每个用户对各类商品、各类品牌的pv、fav、cart、buy数量，建立透视表

pyspark.sql.GroupedData.pivot

pivot透视操作，把某列里的字段值转换成行并进行聚合运算

如果透视的字段中的不同属性值超过10000个，则需要设置spark.sql.pivotMaxValues，否则计算过程中会出现错误。[文档介绍 (https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=pivot#pyspark.sql.GroupedData.pivot)](https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=pivot#pyspark.sql.GroupedData.pivot)。

In [14]:
```
# 统计每个用户对各类商品的pv、fav、cart、buy数量
# 建立透视表：  | userId|cateId| pv| fav|cart| buy|
cate_count_df=behavior_log_df.groupby(behavior_log_df.userId,behavior_log_df.cateId).
# cate_count_df.show()
# tmp.where('userId=38456').show()
# 条件选择where
```

In [15]:
```
cate_count_df.printSchema()
```

```
root
 |-- userId: integer (nullable = true)
 |-- cateId: integer (nullable = true)
 |-- cv: long (nullable = true)
 |-- fav: long (nullable = true)
 |-- cart: long (nullable = true)
 |-- buy: long (nullable = true)
```

In [13]:
```
# 统计每个用户对各个品牌的pv、fav、cart、buy数量
# 建立透视表  | userId|brandId|  cv| fav|cart| buy|
brand_count_df=behavior_log_df.groupby(behavior_log_df.userId,behavior_log_df.brandId)
brand_count_df.show()
```

```
+-------+-------+----+----+----+----+
| userId|brandId|  cv| fav|cart| buy|
+-------+-------+----+----+----+----+
| 454702|  98931|null|null|null|null|
| 755246| 237649|null|null|null|null|
|  59109| 293023|null|null|null|null|
| 671713| 435218|null|null|null|null|
| 915329| 383166|null|   1|null|null|
| 161038| 247861|null|null|null|null|
|  20612| 184921|null|null|null|null|
|1078294| 113336|null|null|null|null|
| 353660|  39211|null|null|null|null|
| 726597| 102457|null|null|null|null|
| 286584| 186296|null|null|null|null|
|  60577|  93403|null|null|null|null|
|1055041| 393034|null|null|null|null|
|1056336| 352813|null|null|null|null|
| 970730| 231690|null|null|null|null|
| 743396| 310648|null|null|null|null|
| 633672| 139148|null|null|null|null|
| 921779| 168156|null|null|null|null|
| 383183| 185225|null|null|null|null|
| 187100| 238536|null|null|null|null|
+-------+-------+----+----+----+----+
only showing top 20 rows
```

3.3 将透视表先存储起来

由于运算时间比较长，所以这里先将结果存储在hdfs，供后续其他操作使用 写入数据时才开始计算

In [ ]:
```
cate_count_df.write.csv("cate_count.csv", header=True)
brand_count_df.write.csv("brand_count.csv", header=True)
```