12.实时日志行为处理

根据实时的用户日志行为做出相应的反馈：**实时更新特征、实时更新召回集**

例，用户的浏览、收藏、加购物车、购买等行为记录到日志后，可能包含如下信息：

- 时间
- 地点
- 用户Id
- 商品Id
- 类别Id
- 品牌Id
- 商品价格

以上信息，就目前我们持有的数据而言，能产生实时影响大概只有分为两种类：

- 对用户的基本信息产生影响的数据：**地点(根据当前低点定位来判断用户的消费环境/等级)、购买行为的商品价格**
- 对用户召回结果产生影响的数据：**商品的类别、品牌**

因此现在假设日志格式为："时间,地点,用户ID,商品ID,类别ID,品牌ID,商品价格"

In [1]:
```
# spark配置信息
from pyspark import SparkConf
from pyspark.sql import SparkSession

SPARK_APP_NAME = "processingOnlineData"
SPARK_URL = "yarn"

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
    ("spark.executor.memory", "2g"),    # 设置该app启动时占用的内存用量，默认1g
    ("spark.executor.cores", "2"),   # 设置spark executor使用的CPU核心数
    ("spark.executor.instances", 1)    # 设置spark executor数量，yarn时起作用
)
# 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()
```

In [7]:
```
# 注意：初次安装并运行时，由于使用了kafka，所以会自动下载一系列的依赖jar包，会耗费一定

from pyspark.streaming.kafka import KafkaUtils
from pyspark.streaming import StreamingContext

ssc = StreamingContext(spark.sparkContext, 2)

kafkaParams = {"metadata.broker.list": "192.168.58.100:9092"}
dstream = KafkaUtils.createDirectStream(ssc, ["mytopic"], kafkaParams)
```

In [6]:
```
!hadoop fs -ls /models
```

```
Found 3 items
drwxr-xr-x   - root supergroup          0 2020-12-12 21:11 /models/CTRModel_AllOneHot.o
bj
drwxr-xr-x   - root supergroup          0 2020-12-12 21:12 /models/CTRModel_Normal.obj
drwxr-xr-x   - root supergroup          0 2020-12-12 21:12 /models/userCateRatingALSMod
el.obj
```

In [9]:
```
#### 获取广告和类别的对应关系
# 从HDFS中加载广告基本信息数据，返回spark dafaframe对象
df = spark.read.csv("/data/ad_feature.csv", header=True)

# 注意：由于本数据集中存在NULL字样的数据，无法直接设置schema，只能先将NULL类型的数据处

from pyspark.sql.types import StructType, StructField, IntegerType, FloatType

# 替换掉NULL字符串，替换掉
df = df.replace("NULL", "-1")

# 更改df表结构：更改列类型和列名称
ad_feature_df = df.\
    withColumn("adgroup_id", df.adgroup_id.cast(IntegerType())).withColumnRenamed("adg
    withColumn("cate_id", df.cate_id.cast(IntegerType())).withColumnRenamed("cate_id",
    withColumn("campaign_id", df.campaign_id.cast(IntegerType())).withColumnRenamed("c
    withColumn("customer", df.customer.cast(IntegerType())).withColumnRenamed("custome
    withColumn("brand", df.brand.cast(IntegerType())).withColumnRenamed("brand", "bran
    withColumn("price", df.price.cast(FloatType()))

# 这里我们只需要adgroupId、和cateId
_ = ad_feature_df.select("adgroupId", "cateId")
# 由于这里数据集其实很少，所以我们再直接转成Pandas dataframe来处理，把数据载入内存
pdf = _.toPandas()


# 手动释放一些内存
del df
del ad_feature_df
del _
import gc
gc.collect()
```

Out[9]: 34

In [10]:
```
def m(e):
    # 当前设定日志数据格式："时间,地点,用户ID,商品ID,类别ID,品牌ID,商品价格"
    # 用逗号分割
    return e[1].split(",")

import redis
import json
import numpy as np

client1 = redis.StrictRedis(host="192.168.58.100", port=6379, db=10)
client2 = redis.StrictRedis(host="192.168.58.100", port=6379, db=9)#离线召回集也9库中

def f(rdd):
    print("foreach", rdd.collect())
    for r in rdd.collect():
        userId = r[2]
        location_level = r[1]   # 取值范围1-4

        new_user_class_level = location_level if int(location_level) in [1,2,3,4] els
        data = json.loads(client1.hget("user_features", userId))
        data["new_user_class_level"] = new_user_class_level    # 注意：该需求只是假设
        client1.hset("user_features", userId, json.dumps(data))

        cateId = r[4]
        # 用户买了哪个类的商品，就从该类商品中随机抽出50个商品，在线召回
        ad_list = pdf.where(pdf.cateId==int(cateId)).dropna().adgroupId.astype(np.int
        if ad_list.size > 0:
            # 随机抽出当前类别50个广告，进行在线召回
            ret = set(np.random.choice(ad_list, 50))
            # 更新到redis 中
            client2.sadd(userId, *ret)
```

In [11]:
```
dstream.map(m).foreachRDD(f)
```

In [12]:
```
1 ssc.start()
```

```
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
foreach []
```

In [7]:
```
1 ssc.stop()
```

In [ ]:
```
1
```