10.离线数据缓存之离线特征

目的:

缓存用户和广告的离线特征，用于实时推荐

上节已经为每个用户缓存了500个商品，这些商品是从最感兴趣的类别中随机选的，使用这些商品的离线特征利用LR算出点击率，再排序。

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

SPARK_APP_NAME = "cacheOfflineFeatures"
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
    ("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 [64]:
```
# "pid", 广告资源位，属于场景特征，也就是说，每一种广告通常是可以防止在多种资源位下的
# 因此这里对于pid，应该是由广告系统发起推荐请求时，向推荐系统明确要推荐的用户是谁，以
# 这样如果有多个资源位，那么每个资源位都会对应相应的一个推荐列表

# 需要进行缓存的特征值

feature_cols_from_ad = [
    "price"    # 来自广告基本信息中
]

# 用户特征
feature_cols_from_user = [
    "cms_group_id",
    "final_gender_code",
    "age_level",
    "shopping_level",
    "occupation",
    "pvalue_level",
    "new_user_class_level"
]
```

从HDFS中加载广告基本信息数据

In [66]:

```
_ad_feature_df = spark.read.csv("/data/ad_feature.csv", header=True)

# 更改表结构，转换为对应的数据类型
from pyspark.sql.types import StructType, StructField, IntegerType, FloatType

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

# 更改df表结构：更改列类型和列名称
ad_feature_df = _ad_feature_df.\
    withColumn("adgroup_id", _ad_feature_df.adgroup_id.cast(IntegerType())).withColumn
    withColumn("cate_id", _ad_feature_df.cate_id.cast(IntegerType())).withColumnRename
    withColumn("campaign_id", _ad_feature_df.campaign_id.cast(IntegerType())).withColu
    withColumn("customer", _ad_feature_df.customer.cast(IntegerType())).withColumnRena
    withColumn("brand", _ad_feature_df.brand.cast(IntegerType())).withColumnRenamed("b
    withColumn("price", _ad_feature_df.price.cast(FloatType()))

def foreachPartition(partition):

    import redis
    import json
    client = redis.StrictRedis(host="192.168.58.100", port=6379, db=10)

    for r in partition:
        data = {
            "price": r.price
        }
        # 转成json字符串再保存，能保证数据再次倒出来时，能有效的转换成python类型
        client.hset("ad_features", r.adgroupId, json.dumps(data))

ad_feature_df.foreachPartition(foreachPartition)
```

从HDFS加载用户基本信息数据

In [67]:
```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, Long

# 构建表结构schema对象
schema = StructType([
    StructField("userId", IntegerType()),
    StructField("cms_segid", IntegerType()),
    StructField("cms_group_id", IntegerType()),
    StructField("final_gender_code", IntegerType()),
    StructField("age_level", IntegerType()),
    StructField("pvalue_level", IntegerType()),
    StructField("shopping_level", IntegerType()),
    StructField("occupation", IntegerType()),
    StructField("new_user_class_level", IntegerType())
])
# 利用schema从hdfs加载
user_profile_df = spark.read.csv("/data/user_profile.csv", header=True, schema=schema
user_profile_df
```

Out[67]:
```
DataFrame[userId: int, cms_segid: int, cms_group_id: int, final_gender_code: int, age_l
evel: int, pvalue_level: int, shopping_level: int, occupation: int, new_user_class_leve
l: int]
```

In [68]:
```
def foreachPartition2(partition):

    import redis
    import json
    client = redis.StrictRedis(host="192.168.58.100", port=6379, db=10)

    for r in partition:
        data = {
            "cms_group_id": r.cms_group_id,
            "final_gender_code": r.final_gender_code,
            "age_level": r.age_level,
            "shopping_level": r.shopping_level,
            "occupation": r.occupation,
            "pvalue_level": r.pvalue_level,
            "new_user_class_level": r.new_user_class_level
        }
        # 转成json字符串再保存，能保证数据再次倒出来时，能有效的转换成python类型
        client.hset("user_features", r.userId, json.dumps(data))

user_profile_df.foreachPartition(foreachPartition2)
```

In [ ]:
```

```