根据用户对品牌偏好打分训练基于ALS的矩阵分解模型

注意！注意！注意！

本单元代码与前一单元代码基本一致，只不过这里是计算用户对品牌的相关偏好

但是这里的数据体量为 113w * 46w的数据量，比用户对类别数据的体量大了46倍

这里进行模型训练时，由于内存限制导致ALS模型训练异常，因此本单元代码仅限于在机器内存足够的前提下

当前测试机器：4核心8线程 + 12GB 虚拟机

In [2]:

```
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()
```

In [3]:
```
# spark ml的模型训练是基于内存的，如果数据过大，内存空间小，迭代次数过多的化，可能会造
# 设置Checkpoint的话，会把所有数据落盘，这样如果异常退出，下次重启后，可以接着上次的训
# 但该方法其实指标不治本，因为无法防止内存溢出，所以还是会报错
# 如果数据量大，应考虑的是增加内存、或限制迭代次数和训练数据量级等
spark.sparkContext.setCheckpointDir("checkPoint/")
```

In [3]:
```
!hadoop fs -ls /
```

```
Found 12 items
drwxr-xr-x   - root supergroup          0 2020-11-06 10:24 /cate_count.csv
drwxr-xr-x   - root supergroup          0 2020-12-12 19:23 /checkPoint
drwxr-xr-x   - root supergroup          0 2020-12-12 18:39 /data
drwxr-xr-x   - root supergroup          0 2021-03-10 09:46 /hbase
drwxr-xr-x   - root supergroup          0 2020-11-11 21:33 /headlines
-rw-r--r--   1 root supergroup       4358 2021-03-10 16:21 /iris.csv
drwxr-xr-x   - root supergroup          0 2020-12-23 22:31 /meiduo_mall
drwxr-xr-x   - root supergroup          0 2020-12-12 21:12 /models
drwxr-xr-x   - root supergroup          0 2020-10-30 12:35 /output
-rw-r--r--   1 root supergroup         84 2020-10-30 12:35 /test.txt
drwx------   - root supergroup          0 2020-10-30 19:26 /tmp
drwxr-xr-x   - root supergroup          0 2020-11-09 14:17 /user
```

In [4]:
```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

schema = StructType([
    StructField("userId", IntegerType()),
    StructField("brandId", IntegerType()),
    StructField("pv", IntegerType()),
    StructField("fav", IntegerType()),
    StructField("cart", IntegerType()),
    StructField("buy", IntegerType())
])
# 从hdfs加载预处理好的品牌的统计数据
brand_count_df = spark.read.csv("brand_count.csv", header=True, schema=schema)
# brand_count_df.show()
```

In [5]:
```
def process_row(r):
    # 处理每一行数据：r表示row对象

    # 偏好评分规则：
    #     m: 用户对应的行为次数
    #     该偏好权重比例，次数上限仅供参考，具体数值应根据产品业务场景权衡
    #     pv: if m<=20: score=0.2*m; else score=4
    #     fav: if m<=20: score=0.4*m; else score=8
    #     cart: if m<=20: score=0.6*m; else score=12
    #     buy: if m<=20: score=1*m; else score=20

    # 注意这里要全部设为浮点数，spark运算时对类型比较敏感，要保持数据类型都一致
    pv_count = r.pv if r.pv else 0.0
    fav_count = r.fav if r.fav else 0.0
    cart_count = r.cart if r.cart else 0.0
    buy_count = r.buy if r.buy else 0.0

    pv_score = 0.2*pv_count if pv_count<=20 else 4.0
    fav_score = 0.4*fav_count if fav_count<=20 else 8.0
    cart_score = 0.6*cart_count if cart_count<=20 else 12.0
    buy_score = 1.0*buy_count if buy_count<=20 else 20.0

    rating = pv_score + fav_score + cart_score + buy_score
    # 返回用户ID、品牌ID、用户对品牌的偏好打分
    return r.userId, r.brandId, rating
```

In [6]:
```
# 用户对品牌的打分数据
brand_rating_df = brand_count_df.rdd.map(process_row).toDF(["userId", "brandId", "rat
# brand_rating_df.show()
```

In [ ]:
```
brand_rating_df
```

基于Spark的ALS隐因子模型进行CF评分预测

ALS的意思是交替最小二乘法（Alternating Least Squares），是Spark中进行基于模型的协同过滤（model-based CF）的推荐系统算法，也是目前Spark内唯一一个推荐算法。

同SVD，它也是一种矩阵分解技术，但理论上，ALS在海量数据的处理上要优于SVD。

更多了解：pyspark.ml.recommendation.ALS (https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=vectors#module-pyspark.ml.recommendation)

注意：由于数据量巨大，因此这里不考虑基于内存的CF算法

参考：为什么Spark中只有ALS (https://www.cnblogs.com/mooba/p/6539142.html)

In [ ]:
```
# 使用pyspark中的ALS矩阵分解方法实现CF评分预测
# 文档地址：https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=
from pyspark.ml.recommendation import ALS

als = ALS(userCol='userId', itemCol='brandId', ratingCol='rating', checkpointInterval
# 利用打分数据，训练ALS模型
# 此处训练时间较长
model = als.fit(brand_rating_df)
```

In [ ]:
```
# model.recommendForAllUsers(N) 给用户推荐TOP-N个物品
model.recommendForAllUsers(3).show()
```

In [ ]:
```
# 将模型进行存储
# model.save("/models/userBrandRatingModel.obj")
```

In [4]:
```
!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 [ ]:
```
# 测试存储的模型
from pyspark.ml.recommendation import ALSModel
# 从hdfs加载模型
my_model = ALSModel.load("/models/userBrandRatingModel.obj")
my_model
```

In [ ]:
```
# model.recommendForAllUsers(N) 给用户推荐TOP-N个物品
my_model.recommendForAllUsers(3).first()
```

In [ ]:
```

```