实时推荐

网站初期，在没有排序模型情况下，可直接根据商品的相似度，找出用户当前正在发生行为的商品 的相似商品进行推荐(在线召回)

In [87]:
```
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
# 注意，如果是使用jupyter或ipython中，利用spark streaming链接kafka的话，必须加上下面语
# 同时注意：spark version>2.2.2的话，pyspark中的kafka对应模块已被遗弃，因此这里暂时只能
os.environ["PYSPARK_SUBMIT_ARGS"] = "--packages org.apache.spark:spark-streaming-kafka
# 配置spark信息
from pyspark import SparkConf
import pyspark

SPARK_APP_NAME = "meiduo_logs"
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
    ("spark.executor.memory", "2g"),    # 设置该app启动时占用的内存用量，默认1g，指一
    ("spark.master", SPARK_URL),    # spark master的地址
    ("spark.executor.cores", "2"),    # 设置spark executor使用的CPU核心数，指一台虚拟
#     ("hive.metastore.uris", "thrift://localhost:9083"),    # 配置hive元数据的访问，

    # 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，默
)
# 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
sc = pyspark.SparkContext(master=SPARK_URL, conf=conf)
```

In [88]:
```
# 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

# 第2个参数表示 程序运行间隔时间
ssc = StreamingContext(sc, 0.5)

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

注意row[1]是什么意思?

以下段代码为例:

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

# 第2个参数表示 程序运行间隔时间
ssc = StreamingContext(sc, 0.5)

kafkaParams = {"metadata.broker.list": "192.168.58.100:9092"}
dstream = KafkaUtils.createDirectStream(ssc, ["meiduo_click_trace"], kafkaParams)
def preprocessing(row):
    return row[1]
def foreachRDD(rdd):
    print("foreachRDD", rdd.collect())
dstream.map(preprocessing).foreachRDD(foreachRDD)

ssc.start()
```

无论使用row还是row[1]，结果是一样的

In [89]:
```
import re
def preprocessing(row):
    match = re.search("\
exposure_timesteamp<(?P<exposure_timesteamp>.*?)> \
exposure_loc<(?P<exposure_loc>.*?)> \
timesteamp<(?P<timesteamp>.*?)> \
behavior<(?P<behavior>.*?)> \
uid<(?P<uid>.*?)> \
sku_id<(?P<sku_id>.*?)> \
cate_id<(?P<cate_id>.*?)> \
stay_time<(?P<stay_time>.*?)>", row[1])

    result = []
    if match:
        result.append(("exposure_timesteamp", match.group("exposure_timesteamp")))
        result.append(("exposure_loc", match.group("exposure_loc")))
        result.append(("timesteamp", match.group("timesteamp")))
        result.append(("behavior", match.group("behavior")))
        result.append(("uid", match.group("uid")))
        result.append(("sku_id", match.group("sku_id")))
        result.append(("cate_id", match.group("cate_id")))
        result.append(("stay_time", match.group("stay_time")))
    return result #得到用户实时点击结果

import redis
client0 = redis.StrictRedis(db=0)    # 此前redis 0号库中已经存储了每个商品的TOP-N个相
client1 = redis.StrictRedis(db=1)

# 根据实时点击的日志->取出uid和sku_id->根据这两个id，去redis中找到对应的相似物品
def foreachRDD(rdd):
    # 网站初期，在没有排序模型情况下，可直接根据商品的相似度，找出用户当前正在发生行为
    for data in rdd.collect():
        # 你忘了吗? python 字典的生成
        # test1=[('ni',1),('hao',2),('ma',3)]
        # test2=dict(test1)
        # test2
        # {'ni': 1, 'hao': 2, 'ma': 3}
        data = dict(data)
        sku_id = data.get("sku_id")
        uid = data.get("uid")

        sim_skus = client0.zrevrange(sku_id, 0,4) # 取出最相似的前5个
        client1.sadd(uid, *sim_skus)    # 放入用户的召回集中
```

In [90]:
```
dstream.map(preprocessing).foreachRDD(foreachRDD)
```

In [91]:
```
ssc.start()
```

In [94]:
```python
# 输入一条点击流日志，比如用户1浏览(pv)了 商品sku_id=19,返回5个与sku_id最相似的商品
import logging#log: 记录
import time

def get_logger(logger_name, path, level):

    # 创建logger
    logger = logging.getLogger(logger_name)
    # level:  OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者自己定义的级别
    logger.setLevel(level)

    # 创建formatter
    # %(asctime)s: 打印日志的时间
    # %(message)s: 打印日志信息
    fmt = '%(asctime)s: %(message)s'
    datefmt = '%Y/%m/%d %H:%M:%S'
    formatter = logging.Formatter(fmt,datefmt)

    # 创建handler
    # FileHandler: writes formatted logging records to disk files
    handler = logging.FileHandler(path)
    handler.setLevel(level)

    # 添加handler和formatter 到 logger
    handler.setFormatter(formatter)
    logger.addHandler(handler)

    return logger

click_trace_logger = get_logger('click_trace','/root/workspace/3.rs_project/project2/r
                                logging.DEBUG)

# 点击流日志
exposure_timesteamp = time.time()
exposure_loc = 'detail'
timesteamp = time.time()
behavior = 'pv' # pv|浏览 cart|加入购物车 fav|喜爱 buy|购买
uid = 4
sku_id = 26
cate_id = 1
stay_time = 60
# # 假设某点击流日志记录格式如下:
click_trace_logger.info("exposure_timesteamp<%d> exposure_loc<%s> timesteamp<%d> behav
                        %(exposure_timesteamp, exposure_loc, timesteamp, behavior, ui

```

In [100]:
```python
# 你忘了吗? python 字典的生成
# test1=[('ni',1),('hao',2),('ma',3)]
# test2=dict(test1)
# test2
# {'ni': 1, 'hao': 2, 'ma': 3}
```

In [95]:
```python
ssc.stop()
```

但当网站运行一段时间后，已经收集了大量的用户行为数据以后，那么在离线处理中，就可以训练出相关排序模型(点击率预测、跳出率预测、转化率预测)。由于离线推荐主要以T+1形式推荐，因此在线推荐就还需要对用户今日的行为进行统计分析，得出用户今日的实时兴趣作为用户的实时画像，**供排序模型使用**

今天之前是一个 T单位的数据,新加一天就是（T + 1）单位的数据。