分析并预处理raw_sample数据集

经分析得：目标是点不点击(clk and nonclk)的广告；只有**广告位pid**是唯一 对点击广告 有用的特征。

依据时间戳将该数据集划分为测试集和训练集

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 = 'preprocessingRawSample'
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]:
```
df = spark.read.csv("/data/raw_sample.csv", header = True)
df.show()
df.printSchema()
```

```
+------+----------+----------+----------+------+---+
|  user|time_stamp|adgroup_id|       pid|nonclk|clk|
+------+----------+----------+----------+------+---+
|581738|1494137644|         1|430548_1007|     1|  0|
|449818|1494638778|         3|430548_1007|     1|  0|
|914836|1494650879|         4|430548_1007|     1|  0|
|914836|1494651029|         5|430548_1007|     1|  0|
|399907|1494302958|         8|430548_1007|     1|  0|
|628137|1494524935|         9|430548_1007|     1|  0|
|298139|1494462593|         9|430539_1007|     1|  0|
|775475|1494561036|         9|430548_1007|     1|  0|
|555266|1494307136|        11|430539_1007|     1|  0|
|117840|1494036743|        11|430548_1007|     1|  0|
|739815|1494115387|        11|430539_1007|     1|  0|
|623911|1494625301|        11|430548_1007|     1|  0|
|623911|1494451608|        11|430548_1007|     1|  0|
|421590|1494034144|        11|430548_1007|     1|  0|
|976358|1494156949|        13|430548_1007|     1|  0|
|286630|1494218579|        13|430539_1007|     1|  0|
|286630|1494289247|        13|430539_1007|     1|  0|
|771431|1494153867|        13|430548_1007|     1|  0|
|707120|1494220810|        13|430548_1007|     1|  0|
|530454|1494293746|        13|430548_1007|     1|  0|
+------+----------+----------+----------+------+---+
only showing top 20 rows

root
 |-- user: string (nullable = true)
 |-- time_stamp: string (nullable = true)
 |-- adgroup_id: string (nullable = true)
 |-- pid: string (nullable = true)
 |-- nonclk: string (nullable = true)
 |-- clk: string (nullable = true)
```

分析数据集字段的类型和格式

1. 查看是否有空值
2. 查看每列数据的类型
3. 查看每列数据的类别情况

In [6]:
```
print("样本数据集总条目数：", df.count())
# 约2600w
print("用户user总数：", df.groupBy("user").count().count())
# 约 114w，略多余日志数据中用户数
print("广告id adgroup_id总数：", df.groupBy("adgroup_id").count().count())
# 约85w
print("广告展示位pid情况：", df.groupBy("pid").count().collect())
# 只有两种广告展示位，占比约为六比四
print("广告点击数据情况clk：", df.groupBy("clk").count().collect())
# 点和不点比率约： 1:20
```

```
样本数据集总条目数： 26557961
用户user总数： 1141729
广告id adgroup_id总数： 846811
广告展示位pid情况： [Row(pid='430548_1007', count=16472898), Row(pid='430539_1007', cou
nt=10085063)]
广告点击数据情况clk： [Row(clk='0', count=25191905), Row(clk='1', count=1366056)]
```

使用dataframe.withColumn更改df列数据结构；使用dataframe.withColumnRenamed更改列名称

In [4]:
```
# 更改表结构，转换为对应的数据类型
from pyspark.sql.types import StructType, StructField, IntegerType, FloatType, LongT

# 打印df结构信息
df.printSchema()
# 更改df表结构：更改列类型和列名称
raw_sample_df = df.\
    withColumn("user", df.user.cast(IntegerType())).withColumnRenamed("user", "userId
    withColumn("time_stamp", df.time_stamp.cast(LongType())).withColumnRenamed("time_s
    withColumn("adgroup_id", df.adgroup_id.cast(IntegerType())).withColumnRenamed("adg
    withColumn("pid", df.pid.cast(StringType())).\
    withColumn("nonclk", df.nonclk.cast(IntegerType())).\
    withColumn("clk", df.clk.cast(IntegerType()))
raw_sample_df.printSchema()
raw_sample_df.show()
```

```
root
 |-- user: string (nullable = true)
 |-- time_stamp: string (nullable = true)
 |-- adgroup_id: string (nullable = true)
 |-- pid: string (nullable = true)
 |-- nonclk: string (nullable = true)
 |-- clk: string (nullable = true)

root
 |-- userId: integer (nullable = true)
 |-- timestamp: long (nullable = true)
 |-- adgroupId: integer (nullable = true)
 |-- pid: string (nullable = true)
 |-- nonclk: integer (nullable = true)
 |-- clk: integer (nullable = true)

+------+----------+---------+-----------+------+---+
|userId| timestamp|adgroupId|        pid|nonclk|clk|
+------+----------+---------+-----------+------+---+
|581738|1494137644|        1|430548_1007|     1|  0|
|449818|1494638778|        3|430548_1007|     1|  0|
|914836|1494650879|        4|430548_1007|     1|  0|
|914836|1494651029|        5|430548_1007|     1|  0|
|399907|1494302958|        8|430548_1007|     1|  0|
|628137|1494524935|        9|430548_1007|     1|  0|
|298139|1494462593|        9|430539_1007|     1|  0|
|775475|1494561036|        9|430548_1007|     1|  0|
|555266|1494307136|       11|430539_1007|     1|  0|
|117840|1494036743|       11|430548_1007|     1|  0|
|739815|1494115387|       11|430539_1007|     1|  0|
|623911|1494625301|       11|430548_1007|     1|  0|
|623911|1494451608|       11|430548_1007|     1|  0|
|421590|1494034144|       11|430548_1007|     1|  0|
|976358|1494156949|       13|430548_1007|     1|  0|
|286630|1494218579|       13|430539_1007|     1|  0|
|286630|1494289247|       13|430539_1007|     1|  0|
|771431|1494153867|       13|430548_1007|     1|  0|
|707120|1494220810|       13|430548_1007|     1|  0|
|530454|1494293746|       13|430548_1007|     1|  0|
+------+----------+---------+-----------+------+---+
```

```
only showing top 20 rows
```

特征选取（Feature Selection）

特征选择就是选择那些靠谱的Feature，去掉冗余的Feature，对于搜索广告，Query关键词和广告的匹配程度很重要；但对于展示广告，广告本身的历史表现，往往是最重要的Feature。

根据经验，该数据集中，只有**广告展示位pid**对比较重要，且数据不同数据之间的占比约为6:4，因此pid可以作为一个关键特征

nonclk和clk在这里是作为目标值，不做为特征

热独编码 OneHotEncode

热独编码是一种经典编码，是使用N位状态寄存器(如0和1)来对N个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候，其中只有一位有效。

假设有三组特征，分别表示年龄，城市，设备；

["男", "女"] [0,1]

["北京", "上海", "广州"] [0,1,2]

["苹果", "小米", "华为", "微软"] [0,1,2,3]

传统变化： 对每一组特征，使用枚举类型，从0开始；

["男“，”上海“，”小米“]=[0,1,1]

["女“，”北京“，”苹果“] =[1,0,0]

传统变化后的数据不是连续的，而是随机分配的，不容易应用在分类器中

而经过热独编码，数据会变成稀疏的，方便分类器处理：

["男“，”上海“，”小米“]=[1,0,0,1,0,0,1,0,0]

["女“，”北京“，”苹果“] =[0,1,1,0,0,1,0,0,0]

这样做保留了特征的多样性，但是也要注意如果数据过于稀疏(样本较少、维度过高)，其效果反而会变差

Spark中使用热独编码

注意：热编码只能对字符串类型的列数据进行处理

StringIndexer (https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=stringindexer#pyspark.ml.feature.StringIndexer)：对指定字符串列数据进行特征处理，如将性别数据“男”、“女”转化为0和1

OneHotEncoder (https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=onehotencoder#pyspark.ml.feature.OneHotEncoder)：对特征列数据，进行热编码，通常需结合StringIndexer一起使用

Pipeline (https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=pipeline#pyspark.ml.Pipeline)：让数据按顺序依次被处理，将前一次的处理结果作为下一次的输入

In [5]:
```
'''特征处理'''
'''
pid 资源位。该特征属于分类特征，只有两类取值，因此考虑进行热编码处理即可，分为是否在资
'''
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

# StringIndexer对指定字符串 列数据进行特征处理，如将性别数据“男”、“女”转化为0和1
stringindexer = StringIndexer(inputCol='pid',outputCol='pid_feature')
# OneHotEncoder对特征列数据，进行热编码，通常需结合StringIndexer一起使用
# dropLast=False 两个pid特征，onehot编码是(2,[0],[1.0])，即[0,1]和[1,0]
# dropLast=Ture 两个pid特征，onehot编码是(1,[0],[1.0])，即[1]和[0]
encoder = OneHotEncoder(dropLast = False, inputCol='pid_feature', outputCol='pid_val
pipline = Pipeline(stages=[stringindexer,encoder])
pipline_model = pipline.fit(raw_sample_df)
new_df = pipline_model.transform(raw_sample_df)# 返回pid_value是稀疏向量
new_df
```

Out[5]:
```
DataFrame[userId: int, timestamp: bigint, adgroupId: int, pid: string, nonclk: int, cl
k: int, pid_feature: double, pid_value: vector]
```

In [6]:
```
new_df.show()
```

```
+------+----------+---------+-----------+------+---+-----------+-------------+
|userId| timestamp|adgroupId|        pid|nonclk|clk|pid_feature|    pid_value|
+------+----------+---------+-----------+------+---+-----------+-------------+
|581738|1494137644|        1|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|449818|1494638778|        3|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|914836|1494650879|        4|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|914836|1494651029|        5|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|399907|1494302958|        8|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|628137|1494524935|        9|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|298139|1494462593|        9|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|775475|1494561036|        9|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|555266|1494307136|       11|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|117840|1494036743|       11|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|739815|1494115387|       11|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|623911|1494625301|       11|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|623911|1494451608|       11|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|421590|1494034144|       11|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|976358|1494156949|       13|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|286630|1494218579|       13|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|286630|1494289247|       13|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|771431|1494153867|       13|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|707120|1494220810|       13|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|530454|1494293746|       13|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
+------+----------+---------+-----------+------+---+-----------+-------------+
only showing top 20 rows
```

返回字段pid_value是一个稀疏向量类型数据

[pyspark.ml.linalg.SparseVector (https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=sparse#pyspark.ml.linalg.SparseVector)](https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=sparse#pyspark.ml.linalg.SparseVector)

In [ ]:
```
'''=====延申学习：spark中的稀疏向量=====
from pyspark.ml.linalg import SparseVector
print(SparseVector(4,[1,3],[3.0,4.0]))
print(SparseVector(4,[2,3],[3.0,4.0]).toArray())
print('*********')
print(new_df.select('pid_value').first())
print(new_df.select('pid_value').first().pid_value.toArray())
'''
'''
(4,[1,3],[3.0,4.0])
[0. 0. 3. 4.]
*********
Row(pid_value=SparseVector(2, {0: 1.0}))
[1. 0.]
'''
```

划分训练集和测试集

In [8]:
```
new_df.sort("timestamp", ascending=False).show()
```

```
+------+----------+---------+----------+------+---+-----------+-------------+
|userId| timestamp|adgroupId|       pid|nonclk|clk|pid_feature|    pid_value|
+------+----------+---------+----------+------+---+-----------+-------------+
|243671|1494691186|   600195|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|177002|1494691186|   593001|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|488527|1494691184|   687854|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
| 17054|1494691184|   742741|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|488527|1494691184|   431082|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
| 17054|1494691184|   756665|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|488527|1494691184|   494312|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|839493|1494691183|   582235|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|704223|1494691183|   624504|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|839493|1494691183|   561681|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|704223|1494691183|   675674|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|628998|1494691180|   618965|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|627200|1494691179|   782038|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|674444|1494691179|   588664|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|322244|1494691179|   820018|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|627200|1494691179|   817569|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|322244|1494691179|   735220|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|738335|1494691179|   451004|430539_1007|     1|  0|        1.0|(2,[1],[1.0])|
|627200|1494691179|   420769|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
|674444|1494691179|   427579|430548_1007|     1|  0|        0.0|(2,[0],[1.0])|
+------+----------+---------+----------+------+---+-----------+-------------+
only showing top 20 rows
```

In [9]:
```
# 本样本数据集共计8天数据
# 前七天为训练数据、最后一天为测试数据

from datetime import datetime
datetime.fromtimestamp(1494691186)
print("该时间之前的数据为训练样本，该时间以后的数据为测试样本：", datetime.fromtimesta
```

```
该时间之前的数据为训练样本，该时间以后的数据为测试样本： 2017-05-12 23:59:46
```

In [10]:
```
# `where` is an alias for :func:`filter`.
# 训练样本：
train_sample = raw_sample_df.where(raw_sample_df.timestamp<=(1494691186-24*60*60))
print("训练样本个数：")
print(train_sample.count())
# 测试样本
test_sample = raw_sample_df.filter(raw_sample_df.timestamp>(1494691186-24*60*60))
print("测试样本个数：")
print(test_sample.count())

# 注意：还需要加入广告基本特征和用户基本特征才能做程一份完整的样本数据集
```

```
训练样本个数：
23249291
测试样本个数：
3308670
```

In [ ]:
```
'''
1. user_id：脱敏过的用户ID；
2. adgroup_id：脱敏过的广告单元ID；
3. time_stamp：时间戳；
4. pid：资源位；
5. noclk：为1代表没有点击；为0代表点击；
6. clk：为0代表没有点击；为1代表点击；
'''

# 只有广告资源位pid 对是否点击 有作用
# 注意：还需要加入 广告基本特征 和 用户基本特征 才能做成 一份完整的样本数据集
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```