**KKBOX用户流失率预测项目报告**

马亚军

陈湘孔

任崇伟

[一、整体数据 1](#_Toc7378740)

[二、数据特征探索 4](#_Toc7378741)

[三、特征工程 6](#_Toc7378742)

[四、模型训练 8](#_Toc7378743)

## 一、整体数据

1. 训练集

Train : (7377418, 6)

训练集共737万个样本。

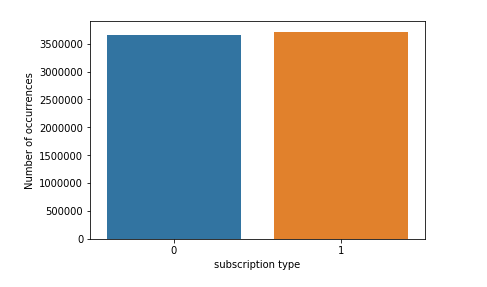
1. 查看正负样本

t = len(df\_kkbox[df\_kkbox[**'target'**]==1])  
f = len(df\_kkbox[df\_kkbox[**'target'**]==0])  
**print**(**'正样本数：'**,t)  
**print**(**'负样本数：'**,t)  
**print**(**'正负比例：'**,1.0\*t/f)

正样本数： 3714656

负样本数： 3714656

正负比例： 1.0141679967194155

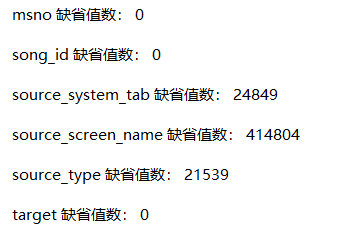


基本上正负样本比例维持在1:1，符合正负样本均衡。

1. 查看各个特征缺失值情况

*#读取数据*dtype={**"msno"**: object,**"song\_id"**: object,**"source\_system\_tab"**: object,**"source\_screen\_name"**: object,**"source\_type"**: object,**"target"**: int}

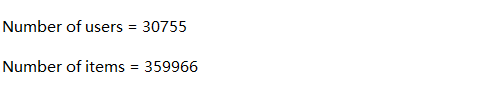
**for** key **in** dtype:  
 **print**(key,**'缺省值数：'**,len(df\_kkbox[pd.isna(df\_kkbox[key])==True]))



其中source\_screen\_name缺失值的样本数约为：41万。

1. 用户和歌曲的数量

n\_users = df\_kkbox[**'msno'**].unique().shape[0]  
n\_items = df\_kkbox[**'song\_id'**].unique().shape[0]  
**print** (**'Number of users = '** + str(n\_users) + **'\n'**+ **'Number of items = '** + str(n\_items) )



用户数：3万左右

歌曲数：359万首

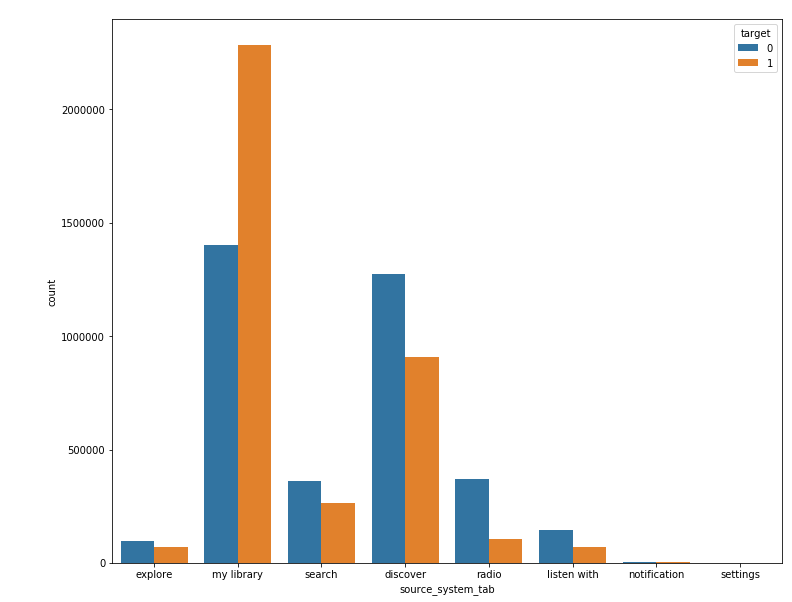
1. 训练集和测试集中的共性



训练集和测试集中共有用户2万，共有歌曲16万。对于测试集来说，大约有1万的冷启动用户，和大约17万的冷启动歌曲。那么在模型选择的过程中，个人认为协同过滤这种方式就不太适合，因为协同过滤对冷启动用户不敏感，如果要去掉训练集中这1万多冷启动用户，未免删除了太多的有用信息吧。

## 二、数据特征探索

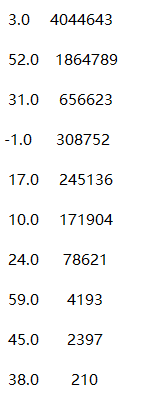
1. source\_system\_tab 事件触发类型



可以看到，notification、settings这两个特征取值基本上占据很少的比例，后期可以考虑将这两个特征值合并为一类。

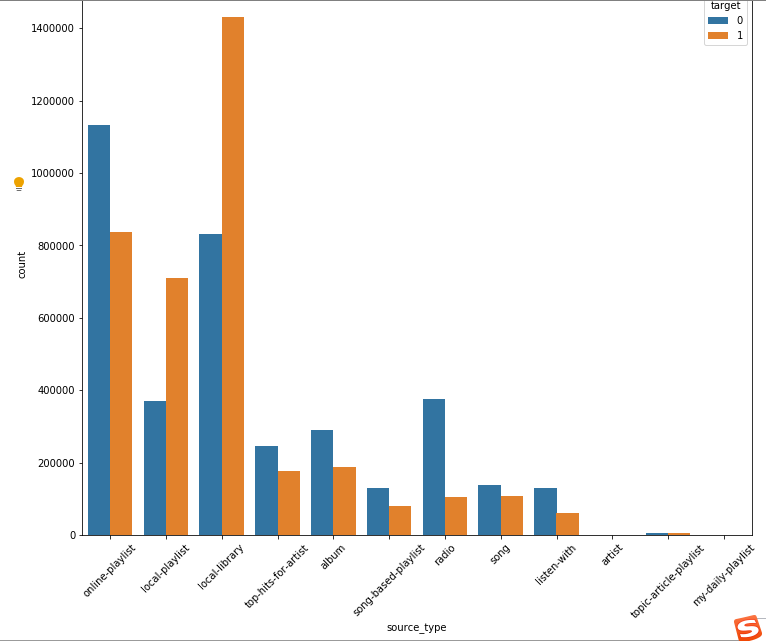
1. language 语言类型

查看语言有哪些取值：



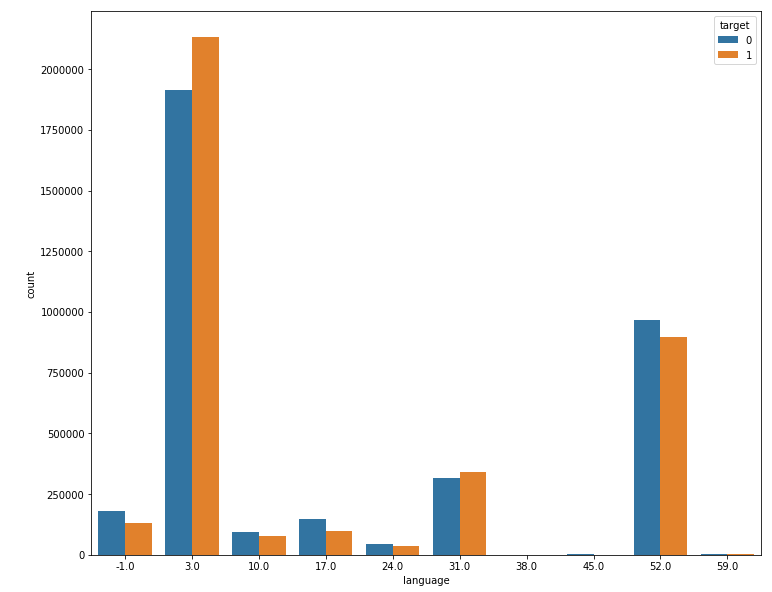
Language取值第一第二为：3.0 52.0，考虑3.0可能为英语，52.0可能是汉语。

1. source\_type 音乐入口类型



主要取值：online-playlist local-playlist local-library top-hits-for-artist album radio song listen-with

后面三个特征：artist topic-article-playlist my-daily-playlist样本取值较少，可以考虑将这三种特征取值合并为一类。



1. bd 年龄

年龄小于0的样本：

**print**(**'年龄小于0的样本'**,len(df\_train.query(**'bd<0'**)))

年龄小于0的样本 195

年龄大于100的样本：

**print**(**'年领大于100的样本有多少：'**,len(df\_train.query(**'bd>100'**)))*#6508*

年龄大于100的样本6508

## 三、特征工程

1. 将歌曲信息表合并到训练集

train = df\_train.merge(df\_songs[song\_cols], on=**'song\_id'**, how=**'left'**)

1. 取出members中的注册日期和到期日期中的年、月、日作为新的特征

members[**'registration\_year'**] = members[**'registration\_init\_time'**].apply(**lambda** x: int(str(x)[0:4]))  
members[**'registration\_month'**] = members[**'registration\_init\_time'**].apply(**lambda** x: int(str(x)[4:6]))  
members[**'registration\_date'**] = members[**'registration\_init\_time'**].apply(**lambda** x: int(str(x)[6:8]))  
  
members[**'expiration\_year'**] = members[**'expiration\_date'**].apply(**lambda** x: int(str(x)[0:4]))  
members[**'expiration\_month'**] = members[**'expiration\_date'**].apply(**lambda** x: int(str(x)[4:6]))  
members[**'expiration\_date'**] = members[**'expiration\_date'**].apply(**lambda** x: int(str(x)[6:8]))  
members = members.drop([**'registration\_init\_time'**], axis=1)

1. 取出音像制品中的年份信息，并作为新的特征song\_year

**def** isrc\_to\_year(isrc):  
 **if** type(isrc) == str:  
 **if** int(isrc[5:7]) > 17:  
 **return** 1900 + int(isrc[5:7])  
 **else**:  
 **return** 2000 + int(isrc[5:7])  
 **else**:  
 **return** np.nan  
  
songs\_extra[**'song\_year'**] = songs\_extra[**'isrc'**].apply(isrc\_to\_year)  
songs\_extra.drop([**'isrc'**, **'name'**], axis = 1, inplace = **True**)

1. 将members表合并到训练集

train = train.merge(members, on=**'msno'**, how=**'left'**)

1. 对gender（性别）采用独热编码

gender\_train=pd.get\_dummies(train[**'gender'**],drop\_first=**True**)

将编码后的新特征合并到训练集：

train=pd.concat([train,gender\_train],axis=1)

*#特征处理后，去掉无用的特征*train.drop([**'gender'**],axis=1,inplace=**True**)

1. 将年龄切分到一个范围内，方便转换为类别型特征

train[**'age\_range'**]=pd.cut(train[**'bd'**],bins=[-45,0,10,18,35,50,80,200])

combine=[train,test]  
**for** value **in** combine:  
 value.loc[(value[**'bd'**] > 0) & (value[**'bd'**] <= 10), **'age\_category'**] = 0  
 value.loc[(value[**'bd'**] > 80) & (value[**'bd'**] <= 200), **'age\_category'**] = 1  
 value.loc[(value[**'bd'**] > 50) & (value[**'bd'**] <= 80), **'age\_category'**] = 2  
 value.loc[(value[**'bd'**] > 10) & (value[**'bd'**] <= 18), **'age\_category'**] = 3  
 value.loc[(value[**'bd'**] > 35) & (value[**'bd'**] <= 50), **'age\_category'**] = 4  
 value.loc[(value[**'bd'**] > -45) & (value[**'bd'**] <= 0), **'age\_category'**] = 5  
 value.loc[(value[**'bd'**] > 18) & (value[**'bd'**] <= 35), **'age\_category'**] = 6

*#年龄、年龄范围处理完后，删除不用特征*train.drop([**'bd'**,**'age\_range'**],axis=1,inplace=**True**)  
test.drop([**'bd'**,**'age\_range'**],axis=1,inplace=**True**)

## 四、模型训练

采用LightGBM模型。为什么没有选择XGBoost呢？原因如下：

1. XGBoost在做特征分割时，需要按照特征的取值进行预排序，复杂度为O(#feature\*data)，那么内存就需要两倍的数据量。
2. XGBoost是基于Level-wise的叶子生长策略。每次特征分割时，需要遍历统一层中所有的叶子节点，计算增益，寻找增益最大的分割点。XGBoost不加区分的计算一层中所有叶子节点的增益，由于有些叶子本身增益较小，是没有必要进行特征搜索和排序的，增加了不少计算量。
3. LightGBM基于HistoGram算法，它将浮点型的特征值切分为K个整数，并生成一个宽度为k的直方图，在第一次遍历数据时，根据每一个特征取值落在直方图中的范围作为索引统计累计量，遍历一次数据后，只需要对k个桶进行搜索和排序，复杂度为O(#k\*data)
4. LightGBM是Leaf-wise的叶子生长策略。在所有的叶子中，计算增益最大的一个叶子，进行分割。同时带有深度的限制，在提高精确度的同时，防止了过拟合。
5. LightGBM是基于直方图做差。一个叶子的直方图可以由它的父节点的直方图减去它兄弟节点的直方图得到。那么，可以通过构造一个数据量较小的直方图，通过它与父节点的直方图做差，得到另一个数据量较大的直方图，计算速度提高了近2倍。

params={  
 **'boosting'**:**'gbdt'**,  
 **'objective'**:**'binary'**,  
 **'metric'**:**'auc'**,  
 **'learning\_rate'**:0.08,  
 **'num\_leaves'**:256,  
 **'max\_depth'**:10,  
 **'num\_rounds'**:1000,  
 **'max\_bin'**:256,  
 **'n\_jobs'**:-1  
}  
model=lgb.train(params=params,  
 train\_set=lgb\_train,  
 valid\_sets=lgb\_val,  
 early\_stopping\_rounds=5)

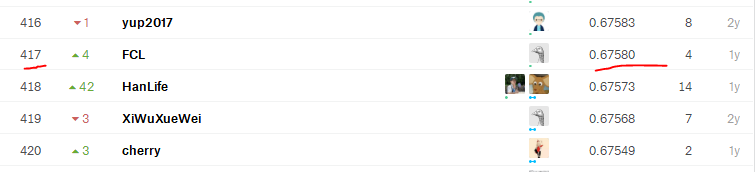
得到预测结果：

print(**'Making predictions and saving them...'**)  
p\_test = model.predict(X\_test)  
  
subm = pd.DataFrame()  
subm[**'id'**] = ids  
subm[**'target'**] = p\_test  
subm.to\_csv(**'submission.csv.gz'**, compression = **'gzip'**, index=**False**, float\_format = **'%.5f'**)  
print(**'Done!'**)

最后，提交到kaggle上得到目前最高的分数及排名：



共1031支参赛队伍，排名为：



大概排到了417名。