音乐流失预测解决方案

一、 解决方案概述

1 项目介绍与问题分析

本项目需要预测 KKBOX 用户在触发了第一个听歌曲事件后，在一个时间窗口内的重复聆听歌曲的可能性。如果用户在第一次听了歌曲后之后的一个月内重复聆听了该歌曲，则其目标被标记为 1，否则标记为 0。数据大概有 700w+。

因为 700 万数据实在太大，通过对比数据特征发现 700 万数据和前 100 万数据中位数，平均数和四分之一分位数基本相同。因此决定用前 100 万数据进行训练。用到的数据有 train，test，songs，members，song_extra_info。这几个数据。首先将 train 数据和 songs 数据根据 songid 进行融合。再将 train 数据和 members 数据根据 msno 进行融合。在融合好的数据中去除 msno 和 songid，用剩下的数据作为特征作为最终的训练数据。

2 项目总体思路

首先从数据清洗开始，介绍对缺失值的多维度处理、对离群点的剔除方法； 其次进行特征工程，包括时间特征、类别特征编码、组合特征构建等。训练算法使用了 LightGBM。

LightGBM 是个快速的、分布式的、高性能的基于决策树算法的梯度提升框架。可用于排序、分类、回归以及很多其他的机器学习任务中。相对其他算法 LightGBM 使用基于直方图的算法，拥有更快的训练速度和更高的效率；使用离散的箱子(BINS)保存并替换连续值使得训练时占用更低的内存。相比于 XGBOOST，由于它在训练时间上的缩减，使其具有够具有处理大数据的能力。并且 LightGBM 可直接处理类别型特征，会自动忽略出现少的类别。

分析项目数据可知，项目数据量较大（700w+），类别型特征较多，基于现有机器性能与服务器性能，所以最终选择 LightGBM 算法。

二、 数据清洗

1 缺失值的多维度处理

类别型特征的缺失值直接使用 Null 来替代，如歌手、作曲、性别等特征。另外还有年龄的缺失值，直接填中值处理。通过查看数据，比较多的缺失值有 sourec_screen_name ， genre_ids ， source_system_tab ， source_type ， composer，lyricist，gender 这几种。

```
In  [124]: df.isnull().sum()

Out[124]: source_system_tab          2939
          source_screen_name        46234
          source_type                2597
          target                        0
          song_length                   0
          genre_ids                 14620
          artist_name                   0
          composer                 212370
          lyricist                 410433
          language                      6
          city                          0
          bd                            0
          gender                   398406
          registered_via                0
          registration_init_time        0
          expiration_date               0
          dtype: int64
```

2 离群点剔除

在样本空间中与其他样本点的一般行为或特征不一致的点称为离群点，考虑到离群点的异常特征可能是多维度的组合，通过分析样本属性的缺失值对年龄和歌曲长度的离群点采用中值进行修改。

3 其他处理

歌手、作词、作曲、歌曲类别带分隔符的特征，使用分隔符”|”或”&”进行分割处理。需要注意的是这些类别里有很多都是带有多个歌手或者多个作词，处理方式是把他们逐个提取出来，然后按照单个歌手或者作词等出现的评率取前三个最高的作为该类特征。

三、 特征工程

1 类别型特征处理

LightGBM 中，每个类别最大支持 255 个不同类别，对于项目中语言、城市、性别、注册类型这几类，其类别较少，可直接使用。而对于歌曲类别、歌手、作词、作曲这些类型，其类别相对较多，最大可达上百万，且不是均不是唯一类别，无法直接使用；又因为数据量太大，增加太多维数对训练时间也有较大影响，所以考虑将其通过其他编码方式处理，降低维数。

类别型特征常见的处理方法有 ONE-HOT 编码、汇总编码、二进制编码、哈希编码、转成数值型特性等。

ONE-HOT 编码：种类别占据一列，对应上百万类别的特征不适用，所以未选用。

汇总编码：汇总也是由于类别太多，没有好的汇总策略，所以未选用。

二进制编码：二进制编码可以有效减少维数，百万级的数据可降维到 20~21 维。但是由于每项数据所属类别并不是唯一的，所以整体增加维数也会增大，所以留作备用方案。

哈希编码：哈希编码可以有效减少维数，而且适用于多类别的情况，对于类别较大的歌手、作词、作曲采用了哈希编码方法。

转成数值型特性：类别型特征转成数值型特征也可以有效减少特征维数，对于歌曲类别，保留了 3 种所属类别，并将类别处理成了概率的形式。

2 歌曲长度处理

歌曲长度单位为 ms，先对其转成秒；然后以 15s 为一个单位进行处理。并且大于 60 单位的歌曲都统一认为是 60，因为太长的歌曲很少，将其作为特殊处理即可。

Out[138]:

	source_system_tab	source_screen_name	source_type	target	song_length	language	city	bd	gende
count	999984.000000	999984.000000	999984.000000	999984.000000	999984.000000	999984.000000	999984.000000	999984.000000	999984.00000
mean	3.533525	8.656476	4.938052	0.691781	15.863640	4.827483	6.565416	25.390743	0.91530
std	1.524836	3.427397	1.692660	0.461758	3.863227	2.073362	6.658079	7.473615	0.83961
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000	0.00000
25%	4.000000	8.000000	4.000000	0.000000	14.000000	4.000000	0.000000	21.000000	0.00000
50%	4.000000	8.000000	5.000000	1.000000	16.000000	4.000000	4.000000	21.000000	1.00000
75%	4.000000	9.000000	6.000000	1.000000	18.000000	8.000000	15.000000	28.000000	2.00000
max	8.000000	18.000000	11.000000	1.000000	491.000000	10.000000	20.000000	74.000000	2.00000

可以看到处理完的歌曲中位数和平均值较相近，数据还是比较好的。

3 歌曲类别处理

查看了每一类大概有多少种类别；歌曲类别相对较少，但是对于类别型特征来说也是非常多的，对歌曲类别计算各每个类别出现的概率，保留了出现概率最大的三个类别，并分成三列。然后对此类别型特征转成了概率，对于那些只出现了一次的类别，他们的概率是相等的。这也就相当于把那些稀有类别归为一类，是一种处理低频特征的手段。

这一步最重要的手段是统计所有类别出现的次数。观察所有特征数据，把每一行对应的特征提取出来。并且存在“|”，或”&”分隔的类别也要分开统计。然后建立一个字典，对应每个类别出现的次数。通过这个总次数计算每个类别的频率，用这个频率作为歌曲的类别特征。提取前三个频次最高的歌曲作为最终特征。

4 歌手、作词、作曲：

观察后发现歌手、作词、作曲类别超过百万，对其进行了哈希编码；和歌曲类别不同，歌曲类别种类不会像歌手和作词一样这么多，歌手的数量是很庞大的，这里用哈希编码比较合适。

5 语言、城市、性别、注册类型

类别较少，直接进行标签编码。比如城市，语言，性别等，可以把缺失值当做控制 NULL 来处理，这样性别就有 3 类，男，女和缺失值。其他类似。

6 注册时间、到期时间

注册时间和到期时间，直接计算到当前时间的月数，对于到期时间超过 36 的统一按 36 处理。那么为了减少计算量，用到期时间减去注册时间作为一个特征。但是这样可能带来的不良后果是，比如一个用户最近才注册的，但是到期时间可能是两年后。而另一个用户是几年前注册的，到期时间可能下个月。这种情况虽然前一个用户时间段会短一点，但是他的到期时间比较后，他续订的可能就会高一点。而第二个用户虽然时间段比较长，但是他即将到期。这都有很大的不确定性。

四、 模型设计与分析

由于 LightGBM 训练速度和效果训练均相对较好，因此采用了 LightGBM 的训练方法，进行 5 折交叉验证的训练。首先在学习率 0.1 的情况下，通过 LightGBM 内置的 cv 函数得到学习器的数量是 3453 个，以 auc 作为评价标准是 0.832337227790249。接下来对各项参数进行调优。如图是调出来的参数：

```
In [15]: #调整学习率再来看一下最大树数目
params = {'boosting_type': 'gbdt',
          'objective': 'binary',
          'is_unbalance':True,
          'categorical_feature': [0,1,2,4,5,6]
           'n_jobs': 4,
          'learning_rate': 0.01,
          #'n_estimators':n_estimators_1,
          'num_leaves': 100,
          'max_depth': 7,
          'min_child_samples':120,
          'colsample_bytree': 0.8,
          'subsample': 0.9
         }
n_estimators_2 = get_n_estimators(params , X_t
F:\Anaconada3\lib\site-packages\lightgbm\basi
ed.
Please use categorical_feature argument of th
  .format(key))
best n_estimators: 10000
best cv score: 0.8187739565591702
```

之后把学习率调整为 0.01 再训练一次学习器。发现这次为 10000 个，但是结果有所下降，可能产生了过拟合。对比两个模型特征的重要性：

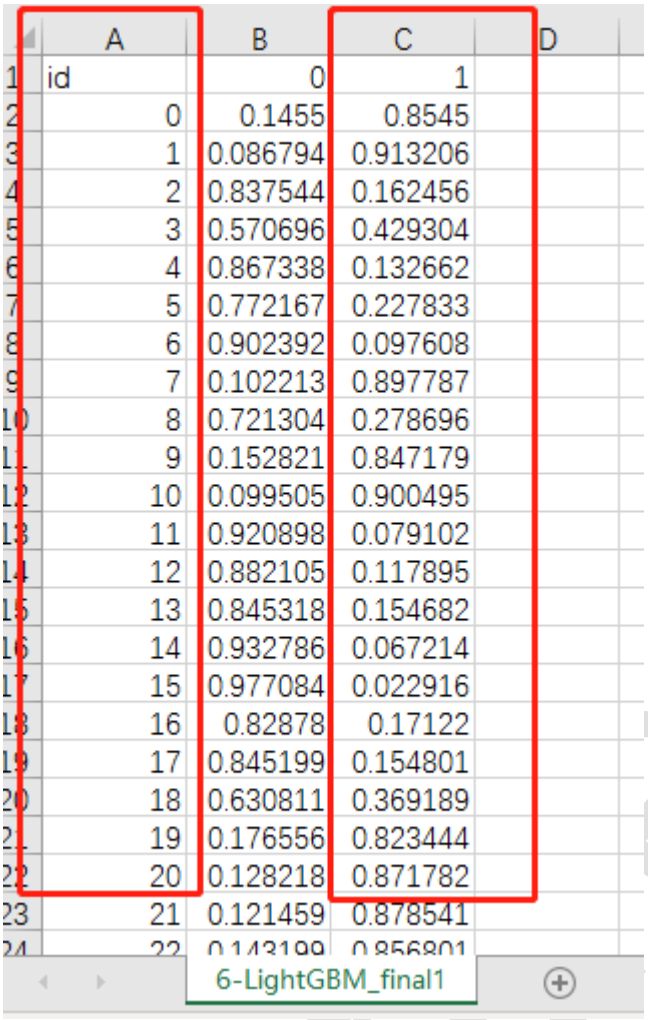
Out[16]:

	columns	importance
9	times_Mon	34485
6	bd	33551
5	city	24181
14	artist_name_hash_1	20092
26	composer_hash_3	12924
1	source_screen_name	12839
2	source_type	10980
36	lyricist_hash_3	10321
3	song_length	9005
10	genreid_0	6808
8	registered_via	6386
4	language	6257
7	gender	5513
28	composer_hash_5	5430
0	source_system_tab	3693
29	composer_hash_6	3385
35	lyricist_hash_2	2084
20	artist_name_hash_7	1604
32	composer_hash_9	1333
30	composer_hash_7	991
11	genreid_1	920
19	artist_name_hash_6	594
24	composer_hash_1	552
39	lyricist_hash_6	536
27	composer_hash_4	386

	columns	importance
6	bd	125709
9	times_Mon	109256
5	city	88087
14	artist_name_hash_1	58054
1	source_screen_name	43490
2	source_type	37077
26	composer_hash_3	36277
36	lyricist_hash_3	29091
3	song_length	23480
10	genreid_0	22429
4	language	22175
8	registered_via	21221
7	gender	18699
28	composer_hash_5	13859
0	source_system_tab	13272
29	composer_hash_6	8797
35	lyricist_hash_2	5000
20	artist_name_hash_7	3890
11	genreid_1	3165
32	composer_hash_9	2852
30	composer_hash_7	1907
24	composer_hash_1	1558
39	lyricist_hash_6	1150
19	artist_name_hash_6	1014
27	composer_hash_4	862
18	artist_name_hash_5	528
12	genreid_2	422
40	lyricist_hash_7	296
38	lyricist_hash_5	290
16	artist_name_hash_3	201

可以发现基本相似。因此采用学习率 0.1 的模型作为最终模型。

得出结果为：(文件 6-LightGBM_final1，图示如下：)



	A	B	C	D
1	id	0	1	
2	0	0.1455	0.8545	
3	1	0.086794	0.913206	
4	2	0.837544	0.162456	
5	3	0.570696	0.429304	
6	4	0.867338	0.132662	
7	5	0.772167	0.227833	
8	6	0.902392	0.097608	
9	7	0.102213	0.897787	
10	8	0.721304	0.278696	
11	9	0.152821	0.847179	
12	10	0.099505	0.900495	
13	11	0.920898	0.079102	
14	12	0.882105	0.117895	
15	13	0.845318	0.154682	
16	14	0.932786	0.067214	
17	15	0.977084	0.022916	
18	16	0.82878	0.17122	
19	17	0.845199	0.154801	
20	18	0.630811	0.369189	
21	19	0.176556	0.823444	
22	20	0.128218	0.871782	
23	21	0.121459	0.878541	
24	22	0.143199	0.856801	

6-LightGBM_final1