**CTR预估项目汇总报告**

# 1.数据初探

### 1.1数据初步印象：

|  |  |  |
| --- | --- | --- |
|  | 数量（万条） | 特征数 |
| 训练数据 | 4000 | 24 |
| 测试数据 | 450 | 23 |

由于数据量过大，电脑无法一次性读取内存，故而采用pandas中的chunksize分批次读取数据，同时注意在完成数据读取后删除相应内存。

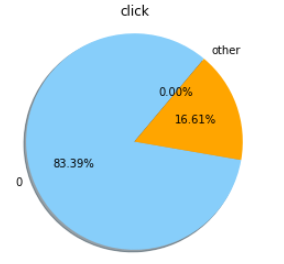
data=pd.read\_csv('D:/',chunksize=5000000)

### 1.2单个特征的数据分布观察

接下来对训练数据中剩余的23维特征中的数值分布进行观察，查看不同数值占比前二十（若有的话）的分布图。由于统计特征中各个数值的分布需要遍历n次（n=特征中不同数值的数量），故而特征数据种类越多会导致计算量飞速上升。

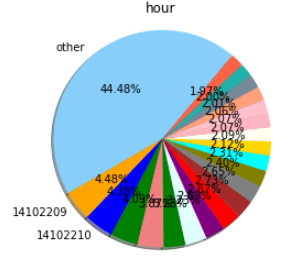
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特征名称 | 特征中数据种类 | 数据探索(万条) | 计算时间（小时） | 备注 |
| device\_id | 2359158 | 100 | 0.5 | 单核 |
| device\_ip | 5962154 | 100 | 1.5 | 单核 |
| 其他特征 | 2～7938 | 1000 | 1.5 | 单核 |

#### 1.2.1 click分布



目标click的分布不均衡，负样本的数量是正样本的5倍，故而在训练时要注意保证正负样本均衡。

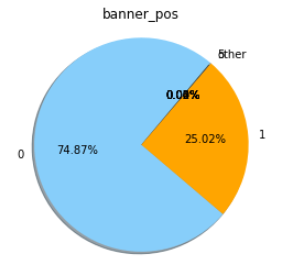
#### 1.2.2 hour分布



时间特征是连续的，由图可以发现时间基本是均匀分布的。故而在随后的特征处理中以第一此的时间为基准将时间变为数值化。

**特征处理**：同时可以额外增加两个特征，如对应的是星期几，具体对应的是早、中、晚。

#### 1.2.3 banner\_pos分布

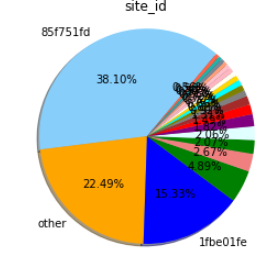


3500w训练数据的整个训练数据共7个类别，图中为1000w（约25%）的样本分布。

此表示广告位置，应该是描述广告出现在阅读设备上的区域。

**特征处理**：故而对此特征采用onehot编码进行处理。

#### 1.2.4 site\_id分布

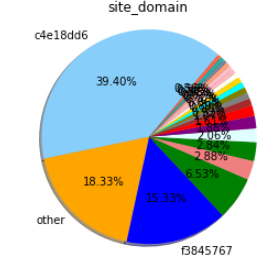


3500w训练数据的site\_id共4636个类别。上图包含了1000w数据。

此表示用户游览广告所在站点的ID，应该表示如头条、新浪等的网站。可以发现占额最大的site\_id占了38%的比例，前20项占了近78%，但于此同时其他site\_id也占了22.5%也算是不小的分了。

**特征处理**：先取前20个进行onehot编码，然后再将剩余的归为一类。若考虑进一步提高性能时，可以尝试将其展开为4636的onehot编码。

#### 1.2.5 site\_domain分布

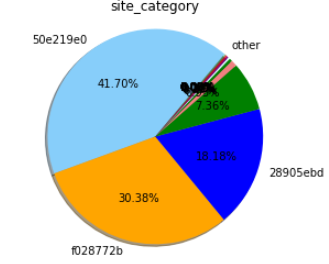


3500w训练数据的site\_domain共7506个类别。上图包含了1000w数据。

此表示用户游览平台所在站点的领域，应该表示如体育、新闻等。可以发现占额最大的site\_domain占了39.4%的比例，前20项占了近82%，但于此同时其他site\_domain也占了18.33%也算是不小的份额。

**特征处理**：先取前20个进行onehot编码，然后再将剩余的归为一类。若考虑进一步提高性能时，可以尝试将其展开为7506的onehot编码。

#### 1.2.6 site\_category分布



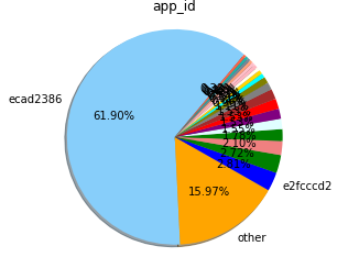
3500w训练数据的site\_category共26个类别。上图包含了1000w数据。

此表示用户游览平台所在站点的领域，应该表示如体育、新闻等。可以发现占额最大的site\_category占了41.7%的比例，前20项占了近100%，other仅占了9e-07。

。

**特征处理**：取前20个进行onehot编码，然后再将剩余的归为一类。

#### 1.2.7 app\_id分布

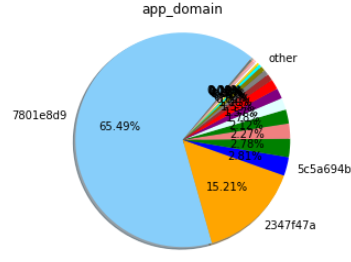


3500w训练数据的app\_id共8025个类别。上图包含了1000w数据。

此表示用户游览广告时所属的app，应该表示如头条、豆瓣等。可以发现占额最大的app\_id占了61.9%的比例，前20项占了近86%，但于此同时其他app\_id也占了16%也算是不小的份额。

**特征处理**：先取前20个进行onehot编码，然后再将剩余的归为一类。若考虑进一步提高性能时，可以尝试将其展开为8025的onehot编码。

#### 1.2.8 app\_domain分布

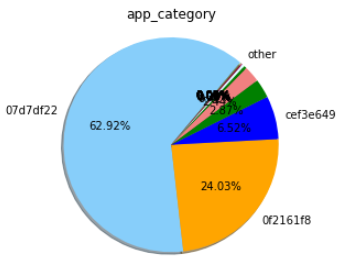


3500w训练数据的app\_domain共515个领域。上图包含了1000w数据。

此表示用户游览广告时所属的app专注的领域，应该表示如新闻、娱乐等。可以发现占额最大的app\_id占了65.5%的比例，前20项占了近99.5%，other仅占0.455%。

**特征处理**：先取前20个进行onehot编码，然后再将剩余的归为一类。若考虑进一步提高性能时，可以尝试取前50个进行的onehot编码。

#### 1.2.7 app\_category分布

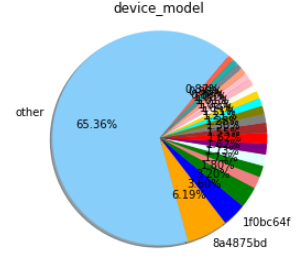


3500w训练数据共35个类别。上图包含了1000w数据。

此表示用户游览广告时所属的app专注的类别，应与app\_domain类似只是分类很少。可以发现占额最大的app\_id占了63%的比例，前20项占了近100%。

**特征处理**：先取前20个进行onehot编码，然后再将剩余的归为一类。若考虑进一步提高性能时，可以尝试将其展开为35维的onehot编码。

#### 1.2.8 device\_model分布

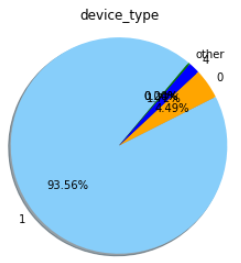


3500w训练数据共8025个类别。上图包含了1000w数据。

此表示用户游览广告所用设备的型号，？如荣耀10、6s、ipad2、R720-15IKBN？。可以发现占额最大的app\_id占了6.19%的比例，前20项仅占了近35%，分配十分均匀，也符合猜测，故而需要将整个数据集进行探索

**特征处理**：进一步探索，有必要的话将其展开为8025的onehot编码。

#### 1.2.9 device\_type分布

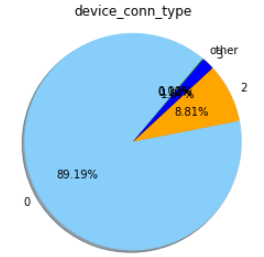


3500w训练数据共5个类别。上图包含了1000w数据。

此应表示用户游览广告所用设备的大类，应该表示如手机、电脑、pad、kindle和其他？。

**特征处理**：将其展为5维onehot编码。

#### 1.2.10 device\_conn\_type分布

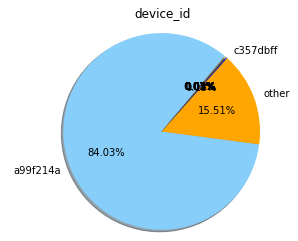


3500w训练数据的app\_id共4个类别。上图包含了1000w数据。

此表示用户游览广告时连网方式，由于其分布与device\_type非常接近，故猜想其为移动网、wifi、光纤、热点？。可以发现占额最大的app\_id占了89%的。

**特征处理**：将其展为4维onehot编码。

#### 1.2.11 device\_id分布



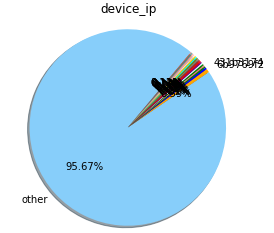
3500w训练数据共2359158个类别。上图包含了**100w**数据（83431）。

此表示设备id，**无法猜想这是什么**？可能可以反映用户所在的地理位置，如具体的基站？。

每个id号平均对应15次点击推送，是否可以将其转化热度反映区域用户的活跃程度？，准备获取全部分布后再考虑处理策略

**特征处理**：是否可以将其转化为地区热度。

#### 1.2.12 device\_ip分布



3500w训练数据共5962154个类别。上图包含了**100w**数据（313002）。

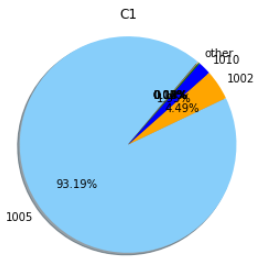
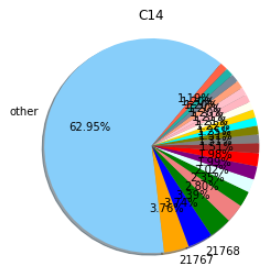
此表示设备ip，可能可以反映具体某个。

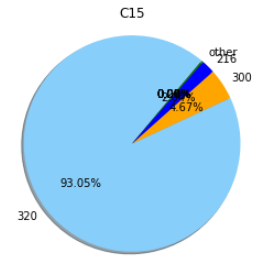
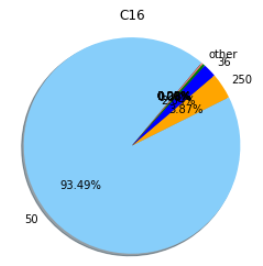
每个ip号平均对应6次点击推送，是否可以将其转化热度反映每个用户的活跃程度？，准备获取全部分布后再考虑处理策略

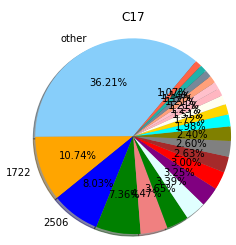
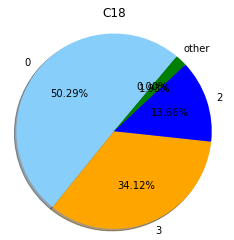
**特征处理**：是否可以将其转化为热度。

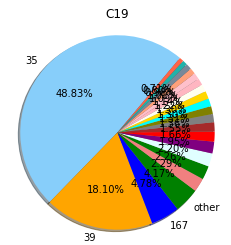
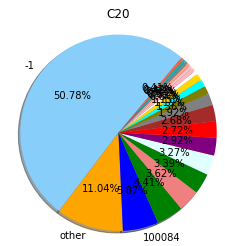
#### 1.2.13 C1,C14—C21分布

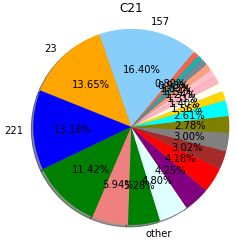
|  |  |  |  |
| --- | --- | --- | --- |
| 特征 | 训练数据特征类别数 | 前20所占比例% | 索取样本数万 |
| C1 | 7 | 100 | 1000 |
| C14 | **2569** | 31 | 1000 |
| C15 | 8 | 100 | 1000 |
| C16 | 9 | 100 | 1000 |
| C17 | **422** | 64 | 1000 |
| C18 | 4 | 100 | 1000 |
| C19 | 66 | 95 | 1000 |
| C20 | 170 | 89 | 1000 |
| C21 | 60 | 95 | 1000 |



对于此C1,C14—C21共9个特征，大致可以分为2类。C14,C17，C20一类，其分布较为平均，尝试取前50维或100维,同时怀疑C20中存在大量的负样本，不然就全部进入onehot；第二类由于所包含类别不多可以将其直接转化onehot。

### 1.3、特征之间的联合分布



由图可以发现，数值变量中device\_type和 C1，C17和C14的分布接近。目前再与click目标的关系中，未体现出明显关联，需要在特征编码完成后再进行探索。

# 2、特征工程方案与实现

个人对CTR预估的理解，用数据描述用户看到广告时的行为，不同特征代表用户不同维度的行为特征，使用机器学习可以发觉数据中蕴藏的潜在规律。 但机器学习目前无论是FFM、GBDT虽然可以对多个组合特征进行解读（如自动关联时间与用户行为特征以反应用户活跃度），但此远远不如我们直接把用户的活跃度作为特征告诉机器。

而这个简单直白根据经验提取描述用户行为特征的过程就是特征工程。

## 2.1特征工程简述

|  |  |  |
| --- | --- | --- |
|  | 数量（万条） | 特征数 |
| 训练数据 | 4200 | 24 |
| 测试数据 | 457.7 | 23 |

其中训练数据中除去click、id外共有22维原始特征，且全部是类别型特征。此特征都是描述用户游览广告时行为的特征，其反应的是那一刻的状态。

而按照我们的根据生活经验，如部分用户仅在有购物需求时才会频繁地登陆逃跑，故而购物地那1小时内登陆淘宝地次数会陡然上升，如部分用户习惯性逛淘宝，其可能对系统推荐地广告视而不见。与时间相关地这些特征都是可以反应用户地某些特定地行为。

虽然原始特征中也有hour这样的时间特征，但仅指望GBDT、FFM通过特征组合来获取用户在不同时间内地活跃度等信息并不现实。

故而在接下来地特征工程工作中，我们将特征工程分为3类。

1. 第一类是原始特征，为减少其维数，我们取后1%的数据作为other，其余用数字进行编码，仍作为类别型特征。
2. 第二类是将hour拆成，哪个小时、星期几、第几天三个特征。其中**前2个是类别**型，**后1个是数字型**。
3. 增加7个反应用户活跃度的特征，此全部是数值型特征。

## 2.2、用户活跃性特征工程

反应用户活跃的特征非常关键，但有两点首先需要确认：

1. 我们用哪个特征作为用户的唯一标识？
2. 我们用哪些特征作为广告的唯一标识？

### 2.2.1用户user\_id的选择

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 特征名称 | device\_id | device\_model | device\_id联合device\_model | device\_ip | device\_ip联合device\_model |
| 特征维数(万) | 380.7 | 11.7 | 391.4 | 1221.0 | 1771.3 |
| 维数增加(万) |  |  | 10.6 |  | 550.2 |
| 维数增加比例% |  |  | 2.8 |  | 45.1 |

在训练数据和测试数据中用户device\_id共380.8w维，将device\_id与device\_model的名称以字符串形式连接在一起进行统计，其维数共仅增加了10.7w维，数据增加仅2.85%，基本不变。

说明每个设备device\_id对应与对应的device\_model是绑定在一起的，这与我们日常使用手机、电脑的行为非常一致。故而我们可以猜测device\_id可以定位到每一个用户，而增加的部分猜测device\_id可能是对应外网设备id，若用户使用局域网的话，同一个局域网下的用户被冠以同一个device\_id。之所以仅增加10w维度，猜想茫茫人海，能聚在一个屋檐下的人并是一种缘分。

而在训练数据和测试数据中用户device\_ip共1221.05w维，将device\_ip与device\_model的名称以字符串形式连接在一起进行统计，其维数共增加了550w维，数据增加了45%，变化很大。依靠常识，上网的ip是随机分配的，每个用户很可能对应多个ip。

综上，采用device\_id联合device\_model来作为用户的识别代号。

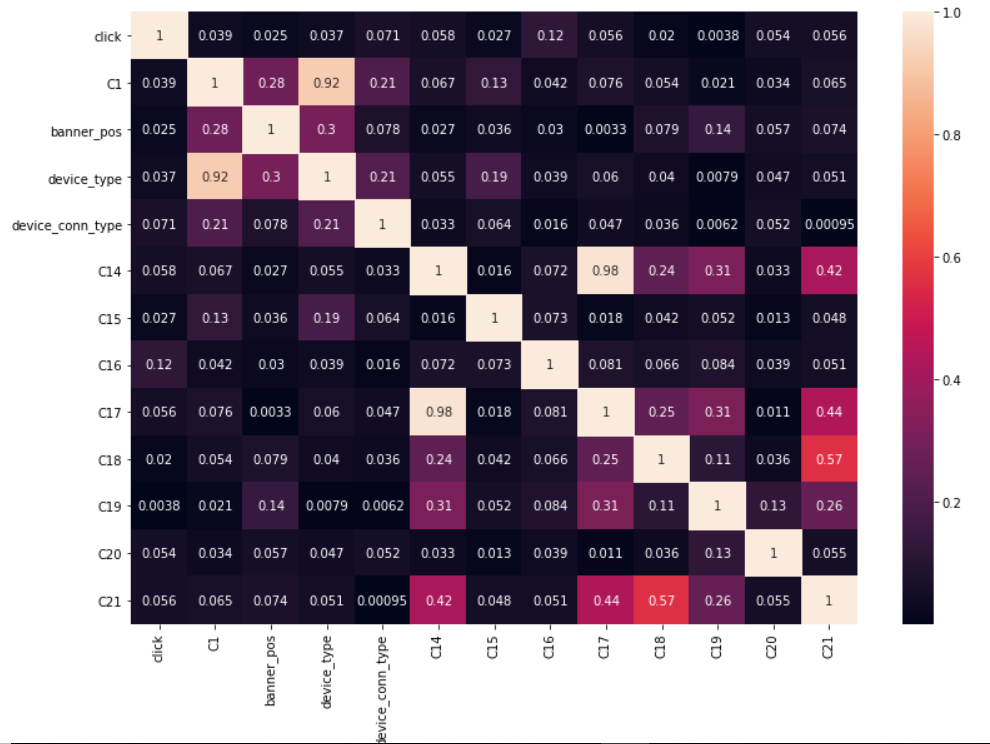
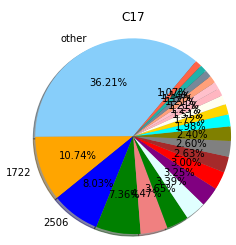
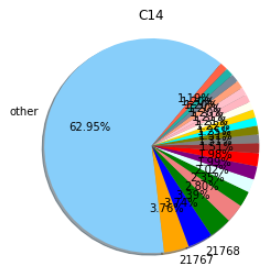
### 2.2.2 广告id选择

借鉴过往竞赛队伍经验，C14 是广告id, C17 是广告类别 id。那接下来我们看看此结论是否合理。

首先我们来看看这两个特征的数量：

|  |  |
| --- | --- |
| 特征名称 | **特征维数** |
| C14 | 2569 |
| C17 | 422 |
| 其他脱敏特征 | 4-170 |

C14和C17在在脱敏数据中间的数据维数是很高的，远高于其他特征，这与广告平台广告类别多、数量多的特点十分相符。



其次我们看看其分布这两个特征同时在热图中，相关系数达到0.97，其分布也较相似。这可以反应他们有很强的相关性与广告类别id-广告id的关系也十分类似。

故而我们将C14作为广告id，C17作为广告类别id。

## 2.3、特征工程方案制定

### 2.3.1反应用户活跃度特征的选择

1）用户游览广告频率特征（4个）

根据我们的生活经验，用户个体越频繁地阅读广告约可以代表其地行为习惯，故而我们可以统计用户该用户在过去1小时，1天中查看某广告、某类广告的次数。

2）用户活跃度特征（2个）

根据我们的生活经验，游览到广告地频率越大，可以代表一种用户的行为习惯。故而我们可以统计用户每日出现的次数、用户出现的天数。

3）用户在某个站点、app的活跃度（1个）

根据生活经验，用户若频繁地登陆如淘宝这样的app，那么应该能表现出用户的一种行为习惯，故而统计其频率也是有意义的。

### 2.3.2 特征工程方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 描述 | 特征工程 | 未onehot的特征数量 | 类别 |
| 原始特征工程 | 反应用户的行为的原始数据 | 除去device\_id、device\_ip、hour，取数据的1%作为other，其余依次用数字进行编码 | 19 | 类别 |
| 时间新增特征 | 反应用户行为发生的具体时间，如一天中的第几个小时 | 将hour特征拆分为： hour\_hours：24个小时，hour\_weekday：星期1-7，hour\_days：具体的第几天1-11 | 3 | hour\_days是数值，其余类别 |
| 用户活跃度新增特征 | user\_hour\_C14：每个用户，在小时内查看该广告的次数 | 用user\_id、hour、C14作为条件进行搜索，统计用户在每小时中游览广告次数 | 1 | 数值 |
| user\_hour\_C17：每个用户，在小时内查看该广告告的次数 | 用user\_id、hour、C17作为条件进行搜索，统计用户在每小时中游览广告次数 | 1 |
| user\_day\_C14：每个用户，在当天查看该广告的次数 | 用user\_id、hour\_days、C14作为条件进行搜索，统计用户在每天中游览广告次数 | 1 |
| user\_day\_C17：每个用户，在当天查看该类型广告的次数 | 用user\_id、hour\_days、C17作为条件进行搜索，统计用户在每天中游览广告次数 | 1 |
| user\_day\_times:每个用户，一天在此app\_id中登录的次数 | 用user\_id、hour\_day作为条件进行搜索，统计用户在每天出现的次数 | 1 |
| user\_day\_app\_id：每个用户，一天在此app\_id中登录的次数 | 用user\_id、hour\_days、app\_id作为条件进行搜索，统计用户在每天中登陆特定app | 1 |
| user\_hour\_app\_id：每个用户，每小时在此app\_id中登录的次数 | 用user\_id、hour、app\_id作为条件进行搜索，统计用户在每天中登陆特定app | 1 |
| user\_days: 每个用户，累计登陆的天数 | 在user\_day\_times的基础上以user\_id为条件统计用户累计出现天数 | 1 |

## 2.4、特征工程执行与数据利用方案

### 2.4.1特种工程实际执行原则与方案实施

由于数据量非常大，故而一方面为了最大程度上利用训练数据提高模型性能，另一方面需要在有限的时间内完成特征工程（最大效率利用手中计算资源）。

故而需结合以上两方面，将从以下2个方面找到效率与性能的最优平衡点。

1. 生成特征数据所需时间
2. 模型所能接受的特征的维度

2.4.1.1生成特征数据所需时间考虑

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 原始特征 (除device\_id、device\_ip、hour) | hour展开24个小时，星期1-7，第几天 | 反应单位小时用户活跃度特征 | 反应每天用户活跃度特征 |
| 特征维数 | 19 | 3 | 3 | 5 |
| 数据生成方式 | 1、生成19维特征，各维度名称与int类型编码对应字典。 2、遍历所有数据，通过对应字典完成特征编码 | | 1、将数据按小时，划分为240个文件。 2、用groupby统计，每个user\_id在该小时中的活跃度，生成user\_id与热度字典 3、遍历该小时数据，通过对应字典完成特征编码 | 1、将数据按天，划分为10个文件。 2、用groupby统计，每个user\_id在该天中的活跃度，生成user\_id与热度字典 3、遍历该天数据，通过对应字典完成特征编码 |
| 特征工程消耗时间h | 约24 | | 约48 | 预计240h |
| 调用资源 | 单台电脑的单核 | | 单台电脑的5个核 | - |
| 完成情况 | 完成 | 完成 | 完成 | - |

上表为，特征工程中生成各类特征所需时间，由于反应用户每天活跃度的特征太耗时间，故而本次项目放弃了此8个特征。

前后分别生成了两组，数据分别为22维特征数据，与反应用户单位小时活跃度的特征数据。并在训练时，进行了对比。

2.4.1.2从模型所能接受的特征的维度考虑

特征产生的维数越多，则onehot后所占用的内存就越大，即使采用libffm格式进行训练也会有此影响。

故而我们将总数据不同百分比的少数特征作为other，共有如下8种方案：

|  |  |  |
| --- | --- | --- |
| 序号 | other占比 | 处理后特征维数 |
| 1 | 原始 | 32000 |
| 2 | 0.01% | 2666 |
| 3 | 0.10% | 2289 |
| 4 | 0.50% | 1801 |
| 5 | 1% | **1517** |
| 6 | 2% | 1220 |
| 7 | 3% | 1043 |
| 8 | 5% | 836 |
| 9 | 10% | 561 |

故而为了最大程度上利用4000w数据，我们目前选择少于1%的数据作为other进行特征工程。

此处需进一步增加特征数量，过后会继续尝试更多维度的训练。

### 2.4.2特征工程中数据利用的原则与实施

本项目，我们将训练数据中第1-9天的数据作为训练集，共有约3500w训练数据。将第10天的数据约450w的测试数据。为了最大程度上利用我们的数据对模型进行训练同时兼顾训练的时间，我们在实际操作中遵循3点：

1. 划分出不同数量的训练、验证集。先用较小数量的数据集进行调优，后依次增加训练数据的数据量，已最大程度上提高调参的效率。
2. 同时为了保证所用训练集能最大程度上代表所有数据。我们先将训练数据按小时分成216份，然后依次平均从这216分中获取数据形成不同大小的训练、验证集。
3. 因为最终的模型是由多个模型组合而成的，如lightGBM+FFM，为了避免训练模型中用同一份数据而导致模型过拟合问题。我们对于不同数据量下的数据准备的4份，2个一组作为测试与验证集。

最终得到的不同数量训练、验证数据集如下表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据文件包含数据量 | 数据文件数量 | 每小时对应数据量 | 覆盖数据总量 |
| 100 | 4 | 4630 | 400 |
| 400 | 4 | 18519 | 1600 |
| 800 | 4 | 37037 | 3200 |
| 1200 | 2 | 55556 | 2400 |

其中包括100、400、800万数据各4分，由于数据数量有限，故而仅准备了2分1200万的数据，准备搭配400万的验证集合进行训练。

## 2.5、特征工程中的经验分享

### 2.5.1用哪个整数作为other的编码问题

关于other特征编码的工作，在整个项目中，共重复了3次。

1. 第1次将-1作为other的编码

由于lightGBM、XGboost仅能接受正整数作为输入数据，故而需更改方案

1. 第2次将999作为other的编码

在使用X-learn时，由于其自带的fm格式数据生成API是根据编码号进行onehot展开，若识别到999，则其认为该特征有999维。导致在predict时，由于训练得到的参数过于稀疏而产生不了结果，故又需更改方案。

1. 第3次将该特征已完成编码的最大整数+1，作为other的编码

如此操作，就解决了上述问题，使得模型得以正常训练。

### 2.5.2特征编码字典的获取问题。

这是一个部分数据能否代表全部数据的问题。

一开始觉得受计算机计算能力限制，准备仅采一部分数据进行特征工程。故而可以仅对参与训练数据进行统计以获得前99%特征对应的编码字典。

但实际上，若仅对部分数据进行统计，有可能出现所统计的特征分布与数据集上的总体分布相差过大，造成误差。

如app\_domain反应广告的类别共515维，若恰巧所选数据中在全部数据比例最大的维度刚好很小，被归为other，则在测试时会造成较大困扰。

### 2.5.3特征工程代码编制习惯

‘03\_特征工程执行代码.ipynb’代码的编写，因本人编码习惯恶劣经两次修改。

1、在实现过程中，为了获取数据的统计量、获取占不同百分比特征工程字典、实现字典的合并、多重字典的存读、绘图和特征工程等功能，编写了约500行的utils辅助类代码。一开始全部堆到了notebook的代码中，给团队使用者带来了很大困扰。

2、在特征工程中，由于代码块写得分散，且简洁地提供文件名修改，给团队使用亦带来了困惑。

## 2.6特征工程执行代码说明

**表.特征工程相关代码与文件**

|  |  |  |
| --- | --- | --- |
| 序号 | 特征工程相关代码与文件 | 说明 |
| 1 | 01\_特征工程数据初探.ipynb | 数据探索 |
| 2 | 02\_特征工程所需字典生成.ipynb | 获取特征工程所用字典 |
| 3 | 03\_特征工程执行代码.ipynb | 执行特征工程代码 |
| 4 | FE\_utils.py | 特征工程所需代码 |
| 5 | utils.py | 数据统计、绘图等辅助代码 |
| 6 | FE\_19.txt(包含不同比例少数特征作为other的字典) | 原始19维特征工程所需字典 |
| 7 | FE\_hour.txt | 原始hour特征工程所需字典 |

其中”**03\_特征工程执行代码.ipynb**”包括：

1、将正负样本按1：r的比例进行下采样，并可存入本地以便调用，

2、增加用户识别特征：user\_id（通过合并device\_id和device\_ip）。

3、原19维特征的特征工程

4、hour展开3个特征的特征工程

5、将数据按照小时分成240份，将other编号为999的改为已完成编码的最大整数+1。

6、分别对每小时数据，进行单位小时热度的统计

7、对处理好的特征进行下采样，形成不同大小的训练集、验证集。(训练集和测试集分开，将前9天的数据作为训练集，最后一天的作为测试集)

**Tip：FE\_19-取不同百分比数据作为other的字典，若需要可以采用其对应字典，将名字改为FE\_19.txt即可。文件说明：‘’FE\_19\_0.5other‘’为将后0.5%作为other。详细说明见下表。**

除'id','device\_id','device\_ip','hour'外，FE\_19.txt使用不同比例特征编码所产生的特征维数表

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | other占比 | 处理后特征维数 | FE\_19对应字典 |
| 1 | 原始 | 32000 | - |
| 2 | 0.01% | 2666 | FE\_19\_0.01other |
| 3 | 0.10% | 2289 | FE\_19\_0.1other |
| 4 | 0.50% | 1801 | FE\_19\_0.5other |
| 5 | 1% | 1517 | FE\_19\_1other |
| 6 | 2% | 1220 | FE\_19\_2other |
| 7 | 3% | 1043 | FE\_19\_3other |

## 2.7特征工程小结

**2.7.1对于CTR为什么需要进行特征工程：**

与图像处理不同，对于图像的深度学习之所以不需要进行特征工程是因为像素与像素之间天然存在意义。但对于CTR的结构化数据，其并不存在这种天然的联系，故而需要运用及其学习的手段将潜在规律找出来。

关于结构化数据，有以下两种模型进行处理。

FFM：此算法可以寻找两两特征之间的组合关系

GBDT：若树的深度设置为5层，理论上来说其可以帮助我们寻找至多5个特征之间的组合关系。

那么问题来了，算法既然可以做到如此，那为什么要需要人来进行特征工程嗯？

个人理解，大致有两个原因：

1. 算法虽然能对原始特征进行组合，但原始特征的组合的数量很多，让机器寻找隐层规律，需要海量的训练。而人工经验性的组合创建性的特征，可以让算法明确应该探索的方向。
2. 关于计算力的限制，若让机器机型进行组合，最好的方式是将用户id与其他特征进行组合，但如此一来对计算力要求会非常高。

# 3、单个模型训练

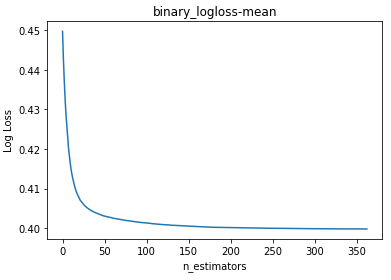
## 3.1、LightGBM的参数调优

选用400万数据，进行参数调优，并采用第10天数据作为测试数据。

采用的数据共22维特征，其中21维为类别型特种，用categorical\_feature参数进行指定。

### 3.1.1 n\_estimators调优

首先是树的数量n\_estimators，在采用默认参数下，模型在363颗树时性能不再提升。此时在训练集上的logloss为0.4001，而在测试集上的logloss为0.404.

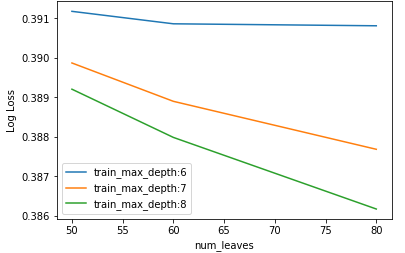
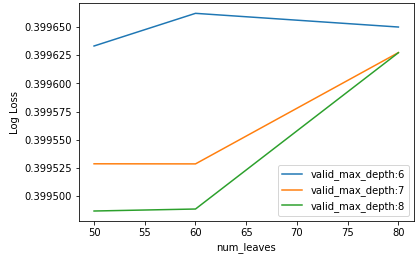


故本轮调优选择的参数，n\_estimators=363

### 3.1.2树的深度和单颗树叶子节点的数目调优

在sklearn框架下采用3折对树的max\_depth与num\_leaves进行调优。

结果显示在训练集上的logloss为0.4001，而在测试集上的logloss为0.404.

结果显示：

1）max\_depth在lightgbm中为树的最大深度限制。树的层数越深，其能组合的特征维度就越多，模型的表达精度就越高，但若深度过大则易导致过拟合。如上图所示，树的最大深度限制从6增加到了8，模型在验证集上的性能由0.39963提升到了0.39953。但同时随深度限制增加，这种增益也由于过拟合的影响越发减少。

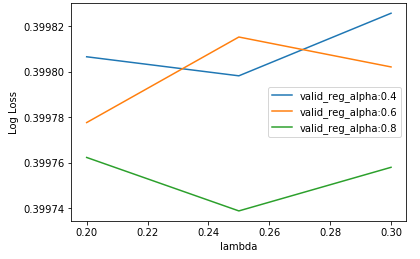
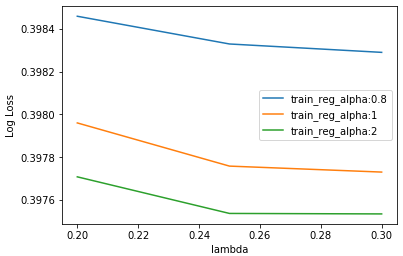
2）num\_leaves在lightgbm是单颗树叶子节点的数目。由于lightGBM是非堆成树式生长，叶子节点数目越大，树的深度越深，表现能力越强，但过大容易造成过拟合。如上图所示，随着叶子节点的增加，logloss反而会随之增大，形成过拟合。

故本轮调优选择的参数，max\_depth=8，num\_leaves=50

### 3.1.3 叶子节点数目正则系数和叶子节点分数正则系数调优

在sklearn框架下采用3折对树的reg\_alpha与reg\_lambda进行调优。

结果最优显示在训练集上的logloss为0.3978，而在测试集上的logloss为0.4028。



结果显示：

1）reg\_alpha在lightgbm中为叶子节点数目的L1正则化系数。正则系数越大，其范化能力就越强。如上图所示，reg\_alpha从0增加到了0.8，模型在验证集上的性能由0.39982提升到了0.39974。此外亦调试了reg\_alpha=1、2，最终0.8仍为最优值。

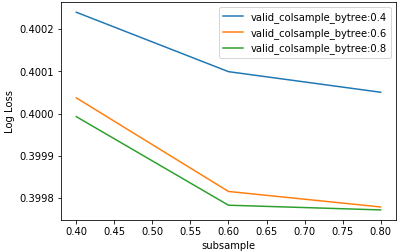
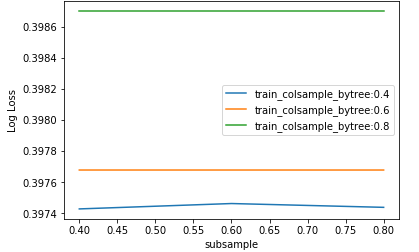
2）reg\_lambda在lightgbm是叶子节点分数的L2正则化系数。正则系数越大，其范化能力就越强。如上图所示，reg\_lambda=0.25时，模型在验证集上的性能最优。

故本轮调优选择的参数，reg\_alpha=0.8，reg\_lambda=0.25

### 3.1.4 行采样与列采样参数调优

在sklearn框架下采用3折对树的Subsample与colsample\_bytree进行调优。

结果最优显示在训练集上的logloss为0.398，而在测试集上的logloss为0.4028。



结果显示：

1）Subsample在lightgbm中为行采样比例系数。此参数越小可以增加模型的泛化能力，但过小会导致模型训练。如上图所示，Subsample从0.6增加到了0.8，模型在验证集上的性能在一直提升。但当Subsample=0.8时，此性能的增加已微乎其微。

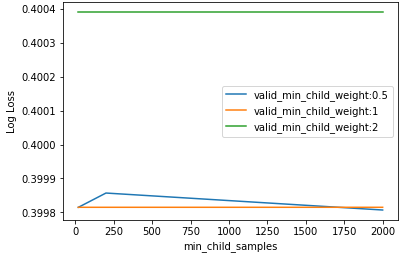
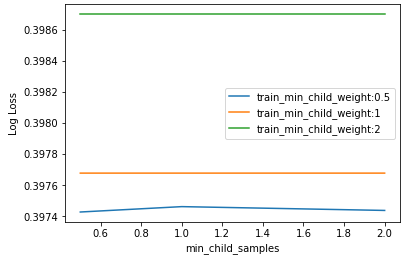
2）colsample\_bytree在lightgbm是为列采样比例系数。此参数越小可以增加模型的泛化能力，但过小会导致模型训练。如上图所示，当Subsample=0.8时，colsample\_bytree增加到0.8，对性能的提升已微乎其微。

故本轮调优选择的参数，Subsample=0.8，colsample\_bytree=0.8

### 3.1.5 叶子节点最小数据量和叶子节点最小权重

在sklearn框架下采用3折对树的min\_child\_samples与min\_child\_weight进行调优。

结果最优显示在训练集上的logloss为0.398，而在测试集上的logloss为0.4028。



结果显示：

1）min\_child\_weight在lightgbm是为叶子结点最小权重和。此参数越大可以增加模型的泛化能力，但过小易导致模型训练。如上图所示，当min\_child\_weight增加到1时，其对应性能最佳。

2）min\_child\_samples在lightgbm中为叶子节点最小数据量。此参数越大可以增加模型的泛化能力，但过小易导致模型训练。如上图所示，min\_child\_weight=1时，min\_child\_samples在20到2000的范围内，模型在验证集上的性能几乎没有变化。

故本轮调优选择的参数，min\_child\_samples=200，min\_child\_weight=1

### 3.1.6 重新优化树的数量并得到模型参数

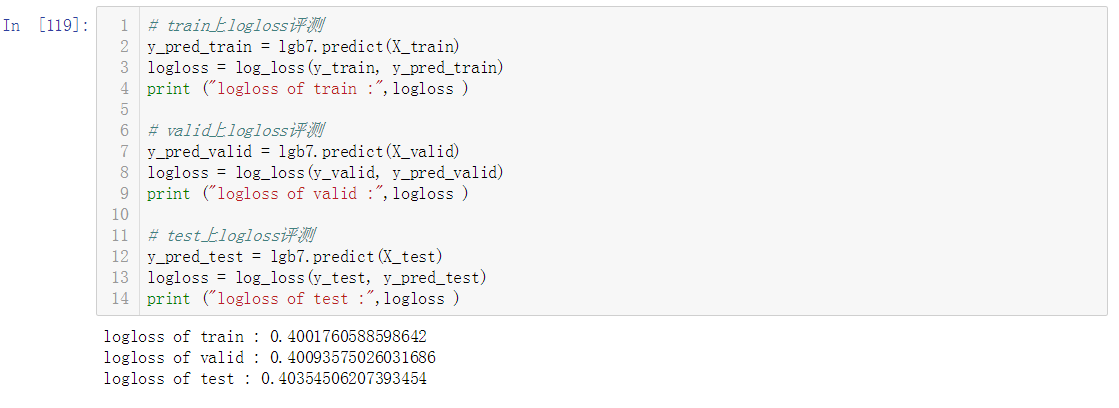
重新优化得到的n\_estimators=284，故而所有参数如下所示：

|  |  |  |
| --- | --- | --- |
| 序号 | 参数 | 数值 |
| 1 | 'learning\_rate' | 0.1 |
| 2 | 'num\_leaves' | 50 |
| 3 | 'max\_depth' | 8 |
| 4 | 'max\_bin' | 256 |
| 5 | 'subsample\_for\_bin' | 200000 |
| 6 | 'subsample' | 0.8 |
| 7 | 'colsample\_bytree' | 0.8 |
| 8 | 'reg\_alpha' | 0.8 |
| 9 | 'reg\_lambda' | 0.25 |
| 10 | 'min\_split\_gain' | 0 |
| 11 | 'min\_child\_weight' | 1 |
| 12 | 'min\_child\_samples' | 200 |

最终在训练集、验证集和测试集上的表现：

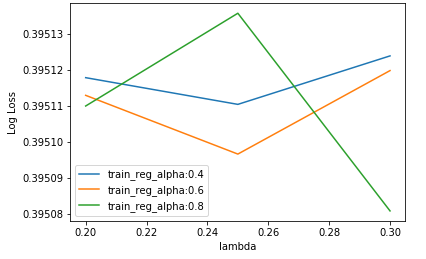
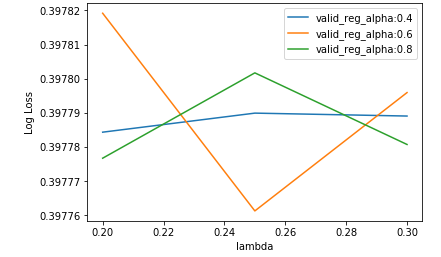
|  |  |  |  |
| --- | --- | --- | --- |
| 数据量万 | train\_logloss | valid\_logloss | test\_logloss |
| 100 | 0.3967 | 0.4019 | 0.4047 |
| 400 | 0.3992 | 0.3997 | 0.4046 |
| 800 | 0.4001 | 0.4009 | 0.4035 |

### 3.1.7训练结果



最终用800万数据在lightGBM上得到的最好结果为：训练集logloss为0.4002，验证集logloss为0.4009，测试集logloss为0.4035

### 3.1.8、引入3个反应单位用户热度特征后的模型效果

在引入关于用户热度特征后，重新对叶子节点数目正则系数和叶子节点分数正则系数调优，并确定了树的数目482。

|  |  |  |  |
| --- | --- | --- | --- |
| 项目 | train\_logloss | valid\_logloss | test\_logloss |
| 为加热度 | 0.4001 | 0.4009 | 0.4035 |
| 加热度特征 | 0.39604 | 0.3963 | 0.4017 |

### 3.1.9、小结

1）LightGBM中树的数目与正则系数对模型的性能影响最大

2）lightGBM与xgboost相比可以自动处理类别型变量，无需额外进行onehot与libff格式数据文件转化，且省内存。

3）lightGBM的速度明显比xgBoost要快，本次测试约快5倍，在参数CV调优时有时明显。

4）lightGBM中可以用ypred=bst.predict(data,num\_iteration=bst.best\_iteration)

5) gbdt生成onehot编码

若一共产生了2颗数，第一颗上3个叶子节点，第二颗上2个叶子节点，若一个样本落在第一颗的第1个，第二颗的第1个节点，则其onehot输出为：[0,1,0,1,0]

若有50颗树，每颗树深度为6，则每颗树最多产生2\*\*6个叶子节点，则最多可以产生50\*64=3200维onehot向量

若有50颗树，每颗树深度为5，则每颗树最多产生2\*\*5个叶子节点，则最多可以产生50\*32=1600维onehot向量

6）仅用lightGBM采用800万不带用户热度特征的数据训练后，该模型在测试集上可以得到logloss=0.4035的成绩。

## 3.2、FFM模型的参数调优

依次选用100万、400万、800万训练数据量对FFM模型进行调优。

对X-learn的FFM模型以下参数进行调优

|  |  |  |
| --- | --- | --- |
| 调优参数 | 参数名称 | 说明 |
| 隐层数量 | k | 隐层数量越多，模型表现力因随之增强，同时计算成本也在增加 |
| 初始化参数 | init | 权重系数的初值 |
| 是否行归一 | disableNorm() | 是否对行数据进行归一化处理，系统默认归一 |
| 是否用FTRL | opt=ftrl | 默认是adagrade可换成ftrl |
| 训练周期 | epoch | 对训练数据利用的次数 |
| 学习率 | lr | 学习率 |
| 正则系数 | lambda | 保证模型泛化能力 |

### 3.2.0 libffm数据格式说明

如上图所示为FFM所需的数据输入格式。其中从左往右解释如下：

1. 第1个表示label
2. 第2组X:Y:Z形式的数据中，X为第几个field；Y为此将全部特征维度展开为onehot，此为第几个onehot向量；Z为所取值，在本项目中全取1.

### 3.2.1 100万数据调优

选用2组100万数据分别作为训练集和验证集，进行参数调优，并采用第10天数据作为测试数据。

用自己编制的libffm数据生成def将原始数据转换为libffm格式数据。

#### 3.2.1.1 隐层数量调优调优

首先是对隐层的层数进行调优，尝试了以下4中情况。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| k | train\_logloss | valid\_logloss | test\_logloss | time(s) |
| 2 | 0.4077 | 0.4146 | 0.4107 | 61 |
| 4 | 0.4077 | 0.4145 | 0.4108 | 63 |
| 8 | 0.4077 | 0.4145 | 0.4108 | 87 |
| 16 | 0.4077 | 0.4144 | 0.411 | 141 |

结果显示：

1. 由于Xlearn利用SSE指令集对计算速度进行了优化，故而在计算时间上，k=2、4的时间复杂度是一样的，而随着k的进一步增加，所消耗时间并未明显增加。
2. 从结果上来说，随着k的增加其在验证集上的效果会有略微的提升。

综上，选择K=4作为参数设置。

#### 3.2.1.2 epoch设置

Epoch为对训练数据的利用次数。

在xlearn中自带cv函数，在设置了early\_stopping后可以如lightgbm对树的棵树一样寻找在最有的epoch。

如：

ffm\_model.xl.create()

…

ffm.cv()

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | train\_logloss | valid\_logloss | test\_logloss |
| 20 | 0.4077 | 0.4145 | 0.4107 |
| 26（设置60） | 0.4076 | 0.4142 | 0.4108 |

在100万数据集合上，当训练到26步时，模型在验证集上的性能将不再提升。

故而在本数据集上，选择epoch=26.

#### 3.2.1.3 权重初始化设置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| init | train\_logloss | valid\_logloss | test\_logloss | time(s) |
| 0.3 | 0.4077 | 0.4145 | 0.4107 | 63 |
| 0.5 | 0.4077 | 0.4145 | 0.4108 | 63 |
| 0.66-default | 0.4077 | 0.4145 | 0.4108 | 63 |

结果显示，初始化对此任务并无影响，故而采用默认值

#### 3.2.1.4 是否采用行归一化

行归一化是通过设置ffm\_model.disableNorm()规定，是否将每个样本的膜长设为1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 是否行归一 | train\_logloss | valid\_logloss | test\_logloss | time(s) |
| 是-default | 0.4077 | 0.4145 | 0.4107 | 63 |
| 否 | 0.3899 | 0.4231 | 0.4199 | 63 |

在本任务中，设置归一化可以起到很好的泛化作用，对性能有着明显的提升，故而采用默认设置，对行数据进行归一。

#### 3.2.1.5 是否采用FTRL

在xlearn模型中，最后可以用adagrade对学习率进行自动调整，也可以采用FTRL对ffm的结果再进行选择优化。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 是否用FTRL | train\_logloss | valid\_logloss | test\_logloss | time(s) |
| 是 | 0.384 | 0.444 | 0.4621 | 145 |
| 否-default | 0.4077 | 0.4145 | 0.4107 | 63 |

在本任务中，未对ftrl算法参数进行精细化调整，发现FTRL在此情况下存在较严重的过拟合。故而本任务仍采用默认的adagrade。

#### 3.2.1.6 学习率的调整

验证了不同学习率下模型的性能，详见下表。

|  |  |  |  |
| --- | --- | --- | --- |
| lr | train\_logloss | valid\_logloss | test\_logloss |
| 0.1-epoch：50 | 0.4076 | 0.4145 | 0.4109 |
| 0.2-epoch：30 | 0.4076 | 0.4142 | 0.4108 |
| 0.3-epoch：26 | 0.4076 | 0.4143 | 0.4108 |

不同学习率对模型的性能并未有明显影响，此应该是由于adagrade可以自动调整不同权重的学习比重有关。

故而本任务选择lr=0.2

#### 3.2.1.7 正则系数的调整

调整正则系数lambda可以提高模型的泛化能力，结合文档，此参数是ffm模型中最终的参数。

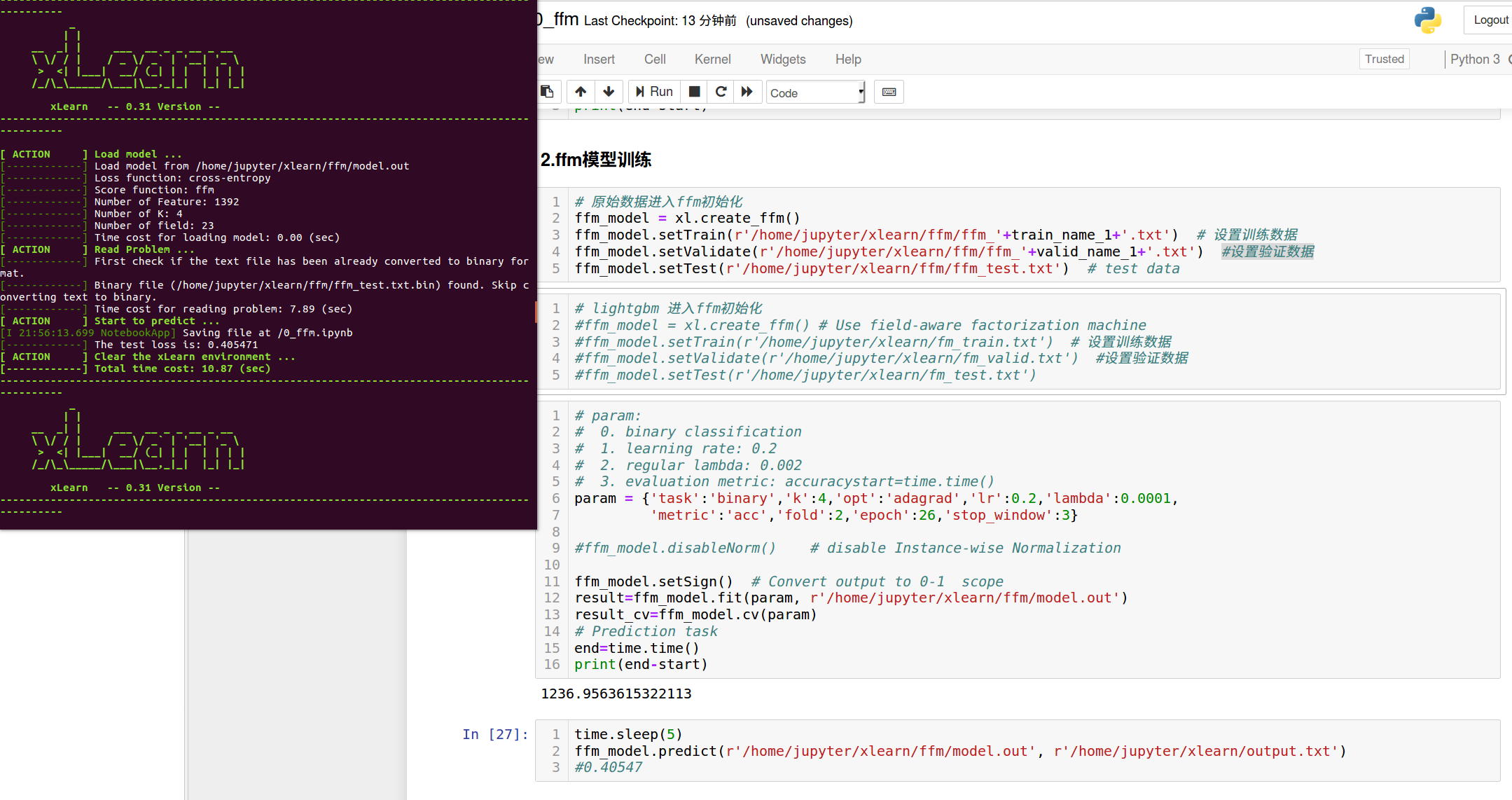
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lambda | train\_logloss | valid\_logloss | test\_logloss | time(s) |
| 0.0001 | 0.3947 | 0.4099 | 0.4053 | 70 |
| 0.0002 | 0.3987 | 0.4098 | 0.4056 | 70 |
| 0.001 | 0.4063 | 0.4134 | 0.4098 | 78 |
| 0.002 | 0.4076 | 0.4142 | 0.4108 | 63 |

结论如下：

1. lambda在本任务中确实是最重要的参数，寻优得到的参数，在验证集上的logloss比原始设置提高了近0.5%。
2. 随着lambda的降低，训练集上的logloss在不断降低，而其在验证集合上的的效果先降低再进入一个拐点。

故而本任务选择的lambda=0.0002

#### 3.2.1.8 100万数据最终效果



### 3.2.2 400万数据调优

经过上一轮的调优，已确定较多数超参数，仍需要调整的是epoch和lambda。

#### 3.2.2.1正则系数的调整

调整正则系数lambda可以提高模型的泛化能力，结合文档，此参数是ffm模型中最终的参数。

|  |  |  |  |
| --- | --- | --- | --- |
| lambda | train\_logloss | valid\_logloss | test\_logloss |
| 0.00001 | 0.3932 | 0.4097 | 0.4056 |
| 0.000025 | 0.394 | 0.4079 | 0.4046 |
| 0.00005 | 0.3952 | 0.408 | 0.4047 |
| 0.0001 | 0.3972 | 0.4076 | 0.4049 |
| 0.0002 | 0.3987 | 0.4081 | 0.4051 |

结论如下：

1. 当训练数据量从100万上升到400万时，在验证集上的logloss确实有了明显的下降，约0.2%。
2. 在采用400万数据进行训练时，对应拐点出现在lambda=0.0001处。

故而本任务选择的lambda=0.0001.

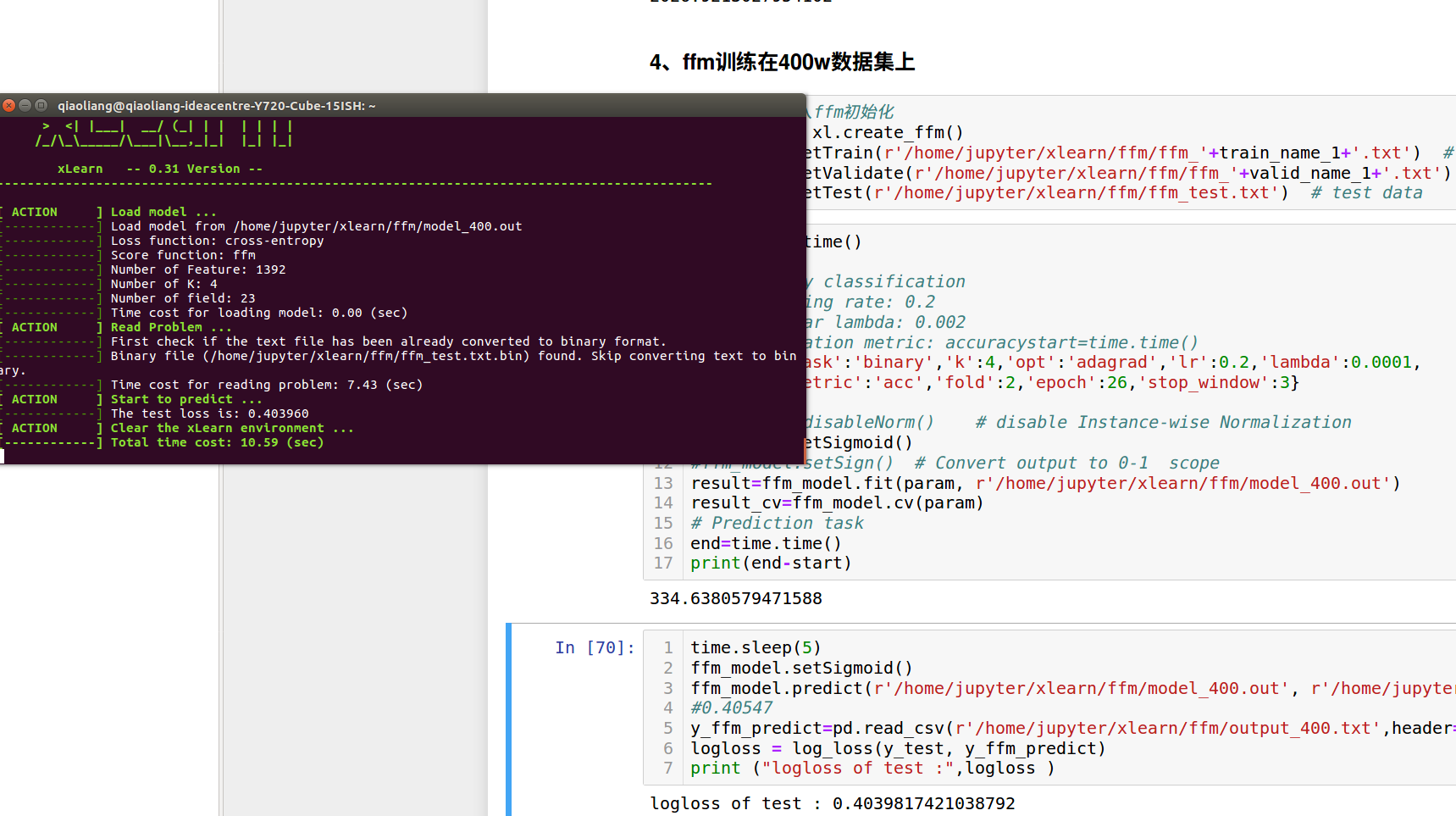
#### 3.2.2.2 epoch调整

调整epoch可以提高模型的泛化能力，结合文档，此参数是ffm模型中最终的参数。

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | train\_logloss | valid\_logloss | test\_logloss |
| 26（设置60） | 0.3972 | 0.4076 | 0.4049 |

Epoch调整后仍取26为最优值。

#### 3.2.2.3 400万数据训练结果



由于ffm模型训练时采用随机采样等，学习率自适应等，故而采用相同数据集进行训练，得到的结果却有所差别，我将结果最好的一张呈现在这。

### 3.2.2 800万数据调优

经过上一轮的调优，已确定较多数超参数，仍需要调整的是epoch和lambda。

#### 3.2.2.1正则系数的调整

调整正则系数lambda可以提高模型的泛化能力，结合文档，此参数是ffm模型中最终的参数。

|  |  |  |  |
| --- | --- | --- | --- |
| lambda | train\_logloss | valid\_logloss | test\_logloss |
| 0.00001 | 0.3949 | 0.4105 | 0.408 |
| 0.00005 | 0.3969 | 0.408 | 0.4048 |
| 0.0001 | 0.3985 | 0.4082 | 0.4053 |

结论如下：

1. 当训练数据量从400万上升到800万时，在验证集上的logloss已经几乎没有变化了。
2. 随着数据量的不断增加，lambda正则化所需要的系数就越小，这与训练数据越多，训练集越能代表全局数据有关。。

故而本任务选择的lambda=0.00005

#### 3.2.2.2 epoch调整

调整epoch可以提高模型的泛化能力，结合文档，此参数是ffm模型中最终的参数。

|  |  |  |  |
| --- | --- | --- | --- |
| epoch | train\_logloss | valid\_logloss | test\_logloss |
| 26（设置60） | 0.3972 | 0.4076 | 0.4049 |

Epoch调整后仍取26为最优值。

## 3.3、FM模型的参数调优

FM可以看作为FFM取场数量为1时的特例。本项目在100万数据集上用FFM进行训练及调参，以便于与FFM模型进行比对。

对X-learn的FM模型，其超参数的配置与FFM是一样的。而场数量的需要是通过输入格式进行区别的。

### 3.3.1 FM数据格式说明：

FM格式的数据，可以通过Xlearn自带的API进行转化，数据说明如下：

1. 第1个表示label

2）第2组X:Y形式的数据中，X为第几个特征；Y为此将此特征特征维度展开为onehot，此为该特征中第几维。

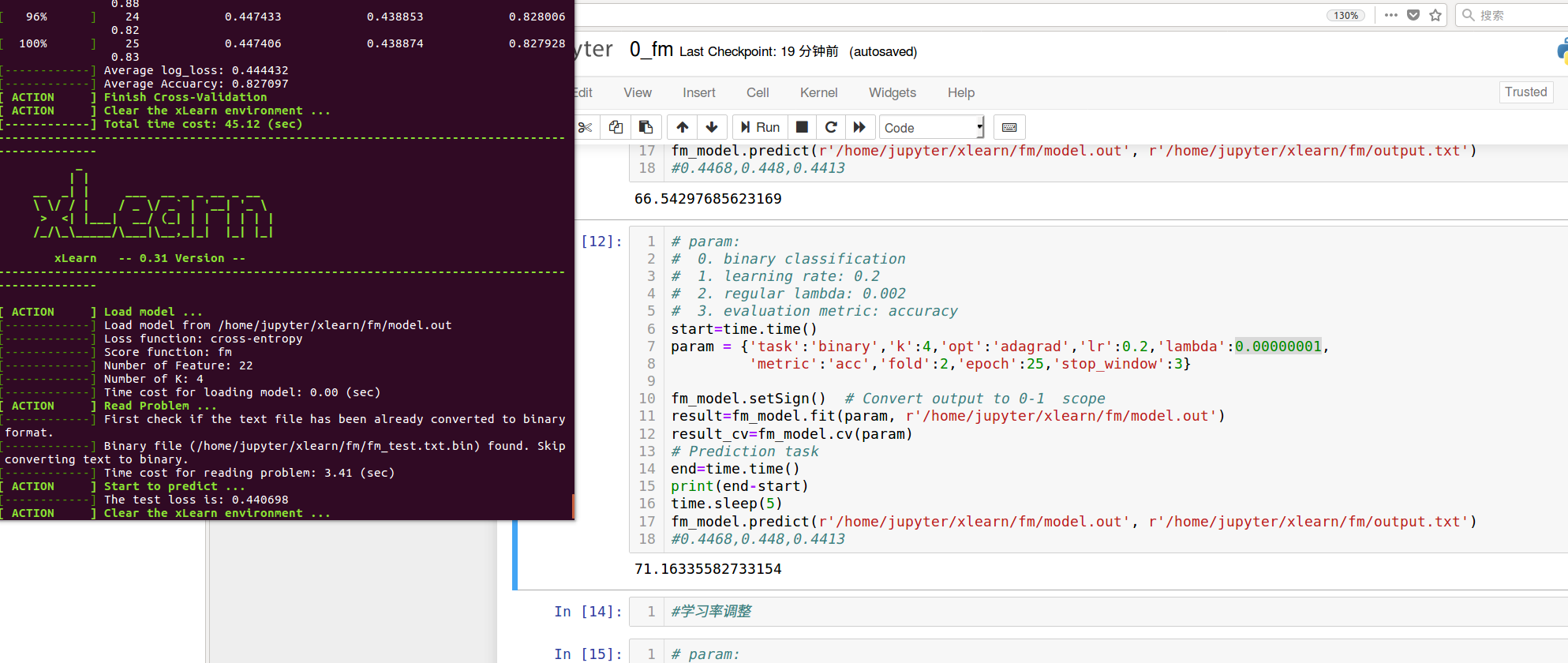
### 3.3.2 FM结果对比

采用100万数据，经过初步调参后，FM与FFM模型的训练结果对比如下所示。

|  |  |  |  |
| --- | --- | --- | --- |
|  | train\_logloss | valid\_logloss | test\_logloss |
| FFM | 0.3987 | 0.4098 | 0.4056 |
| FM | 0.4474 | 0.4388 | 0.4407 |

可以发现FM的性能在测试集、训练集和验证集上均要比FFM差很多。

### 3.3.3 FM训练结果



## 3.4、单个模型训练小结与心得

1）由于个人前3周大量时间分配到特征工程，使得后续模型训练时间过紧。但压力就是动力，由于Xlearn仅能在linux系统下运行。为了能用XLearn对FFM、FM模型进行训练。仅花了1个下午+晚上就完成了双系统的安装与环境搭建。

2）关于模型的训练，由于特征工程对于用户热度计算时间消耗大，且需兼顾模型训练。一开始的数据使用用户热度相关特征，导致在lightGBM、FFM模型上，即使加大了输入模型的数据量，预测性能也未得到明显的提升。用lightgbm在测试集得到的最好logloss=0.4035.而FFM在测试集上可以得到0.4039的成绩，却一直无法突破0.4.

3）在采用了3个与用户热度相关特征后，

# 4.模型的组合训练

## 4.1、组合模型的选择

适用于CTR预估的预测模型有如下4个。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FM | FFM | lightGBM | XGBoost |
| 使用API | X-learn | | lightGBM | XGBoost |

这4个模型分为两类，FM与FFM为一组，用于计算高维度的系数矩阵

lightGBM与XGBoost一组用于对原始特征进行自动的组合，找出对模型性能影响较大特征组合后再交于FM或FFM处理。

### 4.1.1 lightGBM与XGBoost选择

测试时采用500w数据，对模型进行训练，为了对比模型性能，我们将两个模型的叶子节点树、树的层数、树的数目均控制在同一水平。

1）使用xgboost和lightgbm的速度都很快，对于500w数据，也就是60s和10s的区别。

2）lightgbm可指定类别特征，同时对内存并未造成额外负担，故而从这个角度觉得lightgbm性能更加优良。

综上，同时由于我们组是两两一组分别使用xgboost与lightGBM，故而我选择了lightgbm。

### 4.1.2 FM与FFM的对比

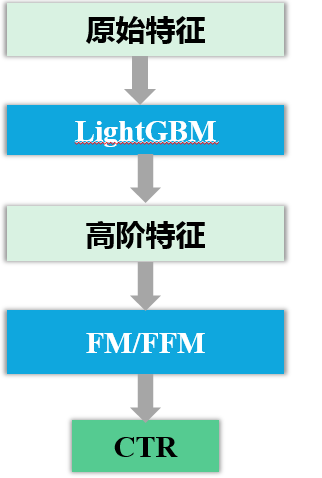
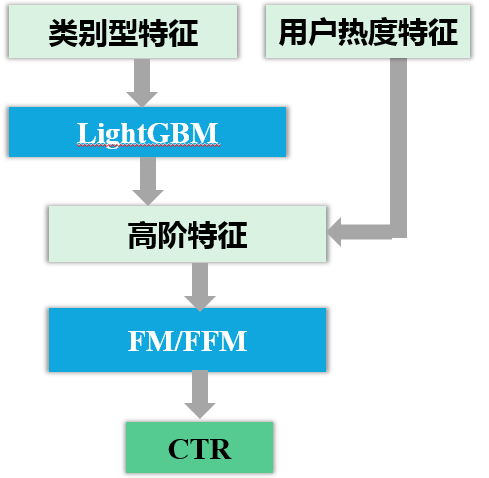
采用100万数据，经过初步调参后，FM与FFM模型的训练结果对比如下所示。

|  |  |  |  |
| --- | --- | --- | --- |
|  | train\_logloss | valid\_logloss | test\_logloss |
| FFM | 0.3987 | 0.4098 | 0.4056 |
| FM | 0.4474 | 0.4388 | 0.4407 |

可以发现FM的性能在测试集、训练集和验证集上均要比FFM差很多。

故而对于本项目选用FFM作为稀疏特征计算模型。

## 4.2、模型的组合训练方式

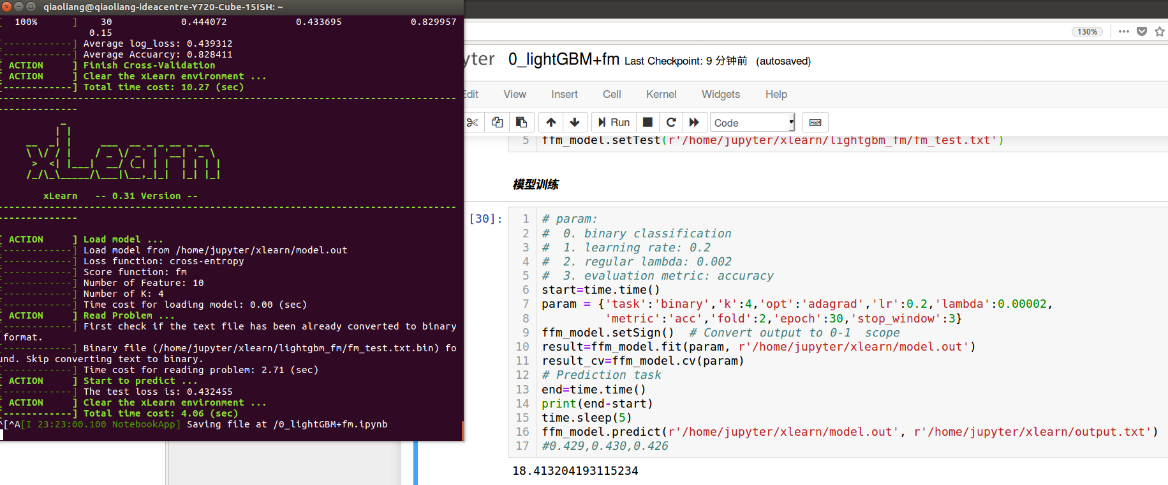
关于模型的组合大致有以上两种方式，左侧是两个模型直接对接，右侧是将连续型特征送入lightGBM训练后得到以叶子节点表示的类别型特征，然后一道送入FM或FFM。

本项目对以下三种方案分别进行测训练。

|  |  |
| --- | --- |
| 序号 | 方案 |
| 1 | lightGBM+FM |
| 2 | lightGBM+FFM |
| 3 | lightGBM处理连续特征， 与原始特征一起交于FFM进行训练 |

### 4.2.1 lightGBM+FM

lightGBM+FM的训练结果如下图所示：



上图分别为在输入100万训练数据的情况下lightGBM的计算结果，与lightGBM+FM的训练结果。

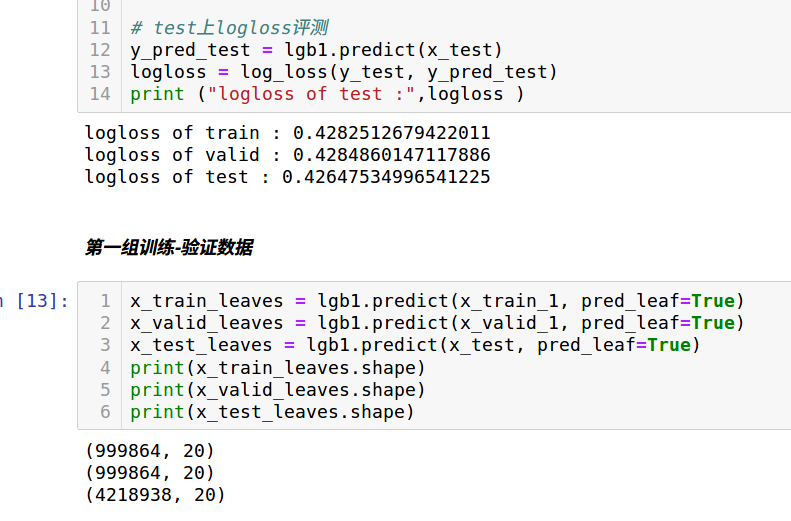
|  |  |  |  |
| --- | --- | --- | --- |
|  | train\_logloss | valid\_logloss | test\_logloss |
| lightGBM | 0.4347 | 0.435 | 0.433 |
| FM | 0.4474 | 0.4388 | 0.4407 |
| lightGBM+FM | 0.4441 | 0.4393 | 0.4324 |

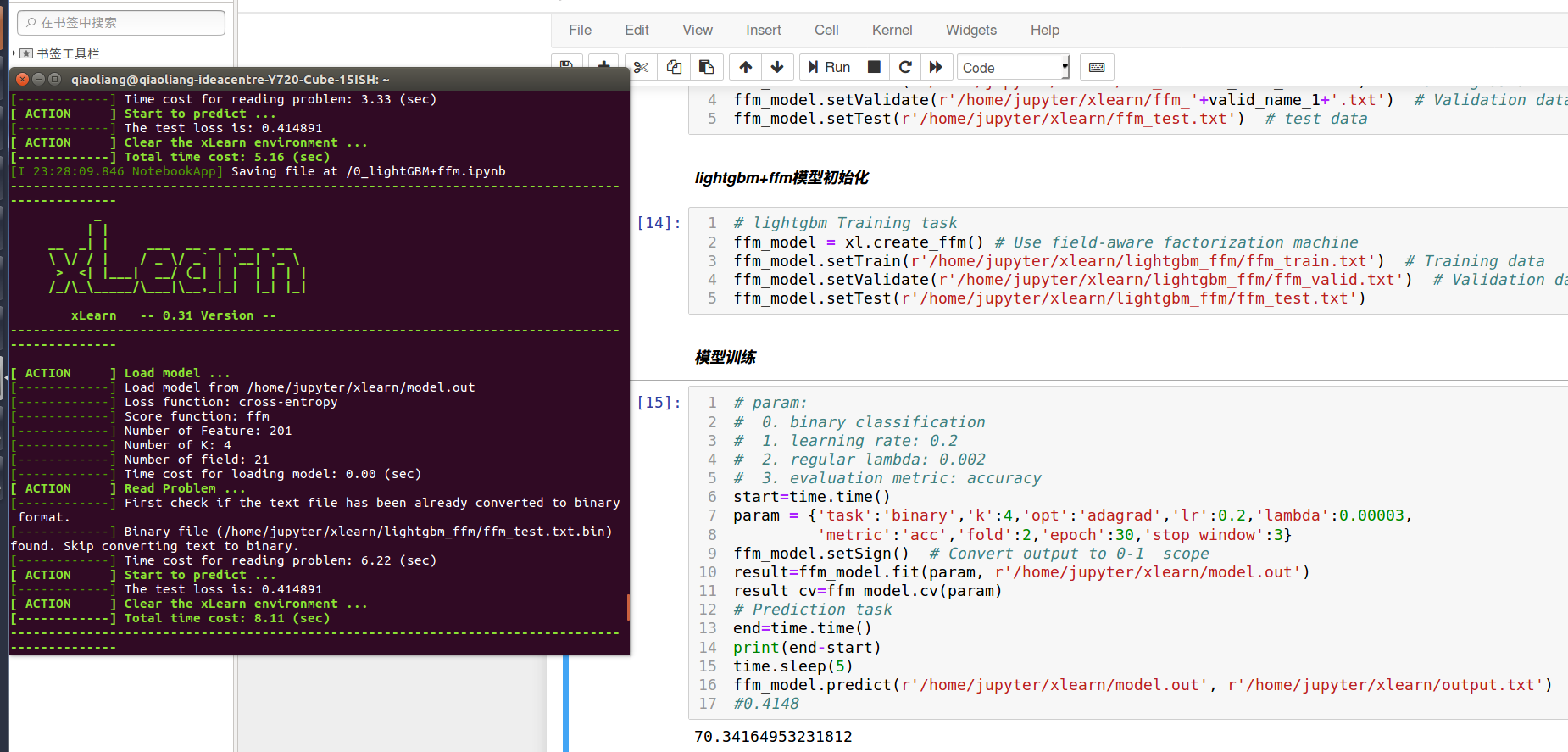
本次训练采用10棵树，叶子节点为10以保证lightGBM产生的叶子节点不出现过拟合。

但从训练效果看，如此组合后的模型相比于单独的模型，在性能上并没有明显提升。

### 4.2.2 lightGBM+FFM

lightGBM+FFM的训练结果如下图所示：





上图分别为在输入100万训练数据的情况下lightGBM的计算结果，与lightGBM+FFM的训练结果。

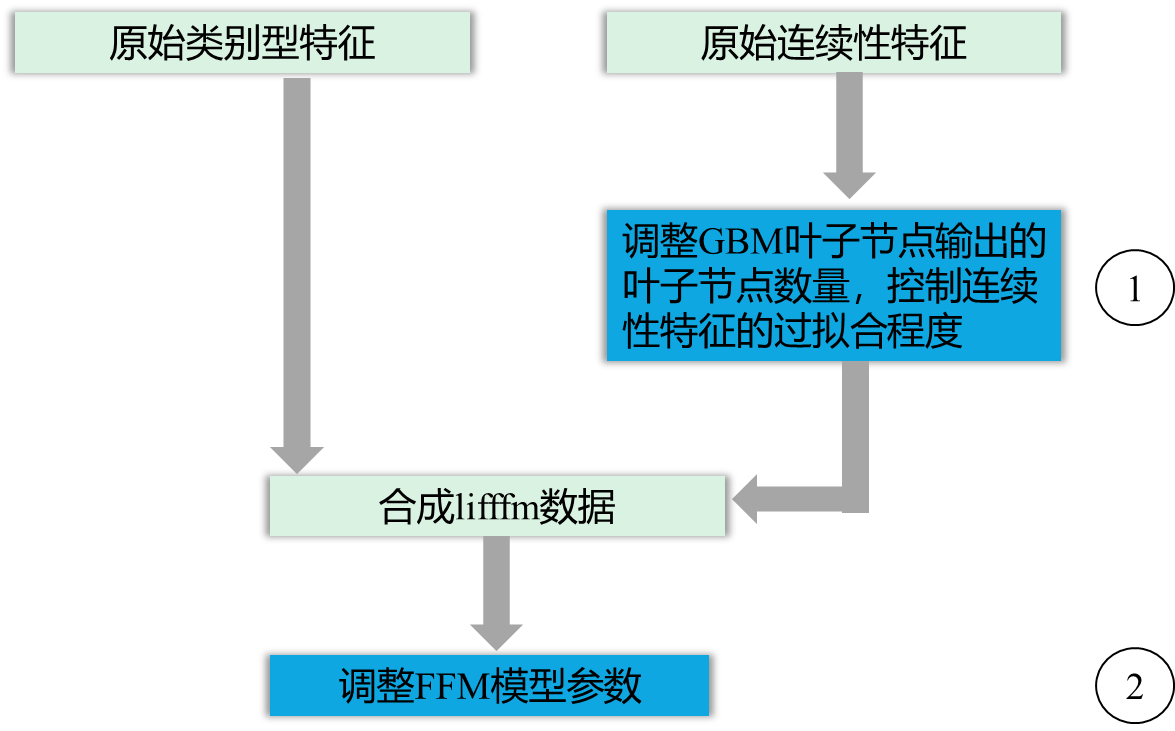
|  |  |  |  |
| --- | --- | --- | --- |
|  | train\_logloss | valid\_logloss | test\_logloss |
| lightGBM | 0.4283 | 0.4285 | 0.4265 |
| FFM | 0.3987 | 0.4098 | 0.4056 |
| lightGBM+FFM | 0.4128 | 0.4165 | 0.4149 |

本次训练采用20棵树，叶子节点为10以保证lightGBM产生的叶子节点不出现过拟合。

从训练效果看，如此组合后的模型相比于单独的lightGBM，在性能上有了明显的提升。但与单独FFM相比性能亦有明显的差异。

### 4.2.3 lightGBM处理连续性变量，与原特征一起送入FFM

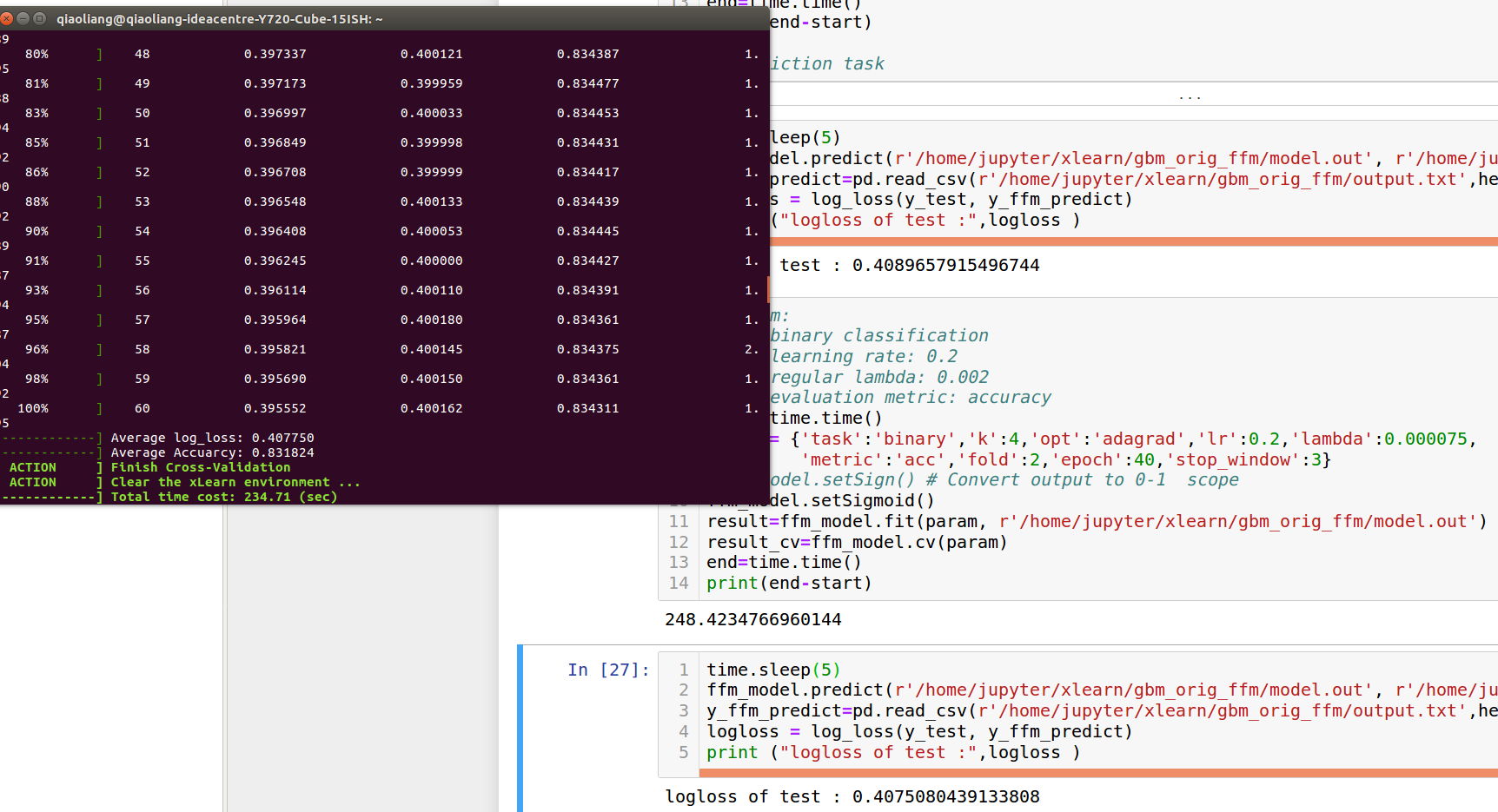
#### 2.3.1 组合模型调试方法



1. 先调试lightgb模型，控制其处理连续型特征的输出的类别型特征的复杂度。
2. 合成libffm数据后，需要对FFM模型的参数进行调优。
3. 根据结果，重新再调整第1步和第2布.

#### 4.2.3.2组合模型训练结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | **train\_logloss** | **valid\_logloss** | **test\_logloss** |
| lightGBM-3连续特征 | 0.4498 | 0.4497 | 0.4491 |
| FFM-联合特征 | 0.3942 | 0.4075 | 0.4078 |

组合模型训练的结果其目前仅用100万数据进行部分调优，应由于训练度不够，故而分数并不高。

# 5、模型性能对比

## 5.1单个模型的性能

### 5.1.1仅采用类别性特征

|  |  |  |  |
| --- | --- | --- | --- |
|  | **train\_logloss** | **valid\_logloss** | **test\_logloss** |
| FFM | 0.3972 | 0.4076 | 0.4039 |
| lightGBM | 0.4002 | 0.4009 | 0.4035 |
| lightGBM+FFM | 0.4128 | 0.4165 | 0.4149 |
| lightGBM+FM | 0.4441 | 0.4393 | 0.4324 |

仅采用类别型特征数据进行训练，可以发现lightGBM和FFM本身的性能是最好的。

而在用模型组合后，若仅输入类别型变量lightGBM与Xgboost已具有较高的预测，若其后再接FFM，性能不易提高。

故而，个人觉得lighttGBM适合直接处理类别性特征而不是作为串联的特征预处理器。同时lightGBM更适合处理连续性特征，自动将其进行分类产生类别性特征。

### 5.1.2采用反映热度的特征

|  |  |  |  |
| --- | --- | --- | --- |
|  | **train\_logloss** | **valid\_logloss** | **test\_logloss** |
| lightGBM | 0.39604 | 0.3963 | 0.4017 |
| lightGBM处理连续热度特征&原始类别送入FFM | 0.3942 | 0.4075 | 0.4078 |

由上图可以发现，增加了反映用户热度的特征后，lightGBM单个模型性能有了较大的提升。而在采用组合模型后，其测试集较原先有了一定提升，但测试集上效果还未提升，应该是受限于机器性能。

## 6、个人理解及遇到的问题汇总

### 6.1 CTR任务为什么需要进行特征工程：

与图像处理不同，对于图像的深度学习之所以不需要进行特征工程是因为像素与像素之间天然存在意义。但对于CTR的结构化数据，其并不存在这种天然的联系，故而需要运用及其学习的手段将潜在规律找出来。

关于结构化数据，有以下两种模型进行处理。

FFM：此算法可以寻找两两特征之间的组合关系

GBDT：若树的深度设置为5层，理论上来说其可以帮助我们寻找至多5个特征之间的组合关系。

那么问题来了，算法既然可以做到如此，那为什么要需要人来进行特征工程嗯？

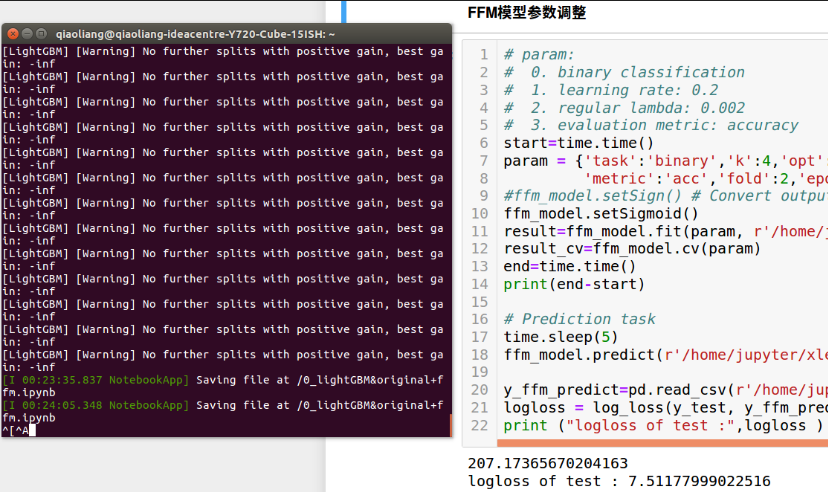
个人理解，大致有两个原因：

1） 算法虽然能对原始特征进行组合，但原始特征的组合的数量很多，让机器寻找隐层规律，需要海量的训练。而人工经验性的组合创建性的特征，可以让算法明确应该探索的方向。

2） 关于计算力的限制，若让机器机型进行组合，最好的方式是将用户id与其他特征进行组合，但如此一来对计算力要求会非常高。

### 6.2在并联的模型组合中，第几天特征hour\_days是否需作为连续性特征送入lightgbm

实际中，将表示第几天的hour\_days作为连续性变量送入lightgbm时会报错，并导致logloss异常得大。



原因是：训练集中仅有第1~9天的hour\_days而测试集中仅有第10天hour\_days。GBM是特征的组合，故而测试数据送入lightgbm后找不到对应的叶子节点，故而无法分类。

故而需要将此删除。

### 6.3 时间特征在CTR中非常重要

在实际工作中，通常将用户id与各类特征进行组合，然后送入模型进行训练。本次项目中我本准备提取的反应用户热度的特征未这个原理的简化版。但由于机器性能原因，我们目前仅生成了3个用户每小时热度的特征。相信完成所有热度特征后，模型的性能可以得到进一步的提升。

### 6.4在模型组合过程中对lightGBM的理解。

根据模型的试验结果，对于类别型变量， lightGBM与Xgboost已具有较高的预测性能，若其后串联FFM，性能不易提高。

个人觉得在模型组合时，lightGBM更适合处理连续性特征。因为lightgbm可以找到数据最优的分离点，然后进行分裂。

## 附录：

**表1、xgboost、lightGBM原生和sklearn下参数对照表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | [**xgb**](https://github.com/Jie-Yuan/DataMining/blob/master/5_PopularAlgorithm/1_Boosting/2_xgb/README.md#1-%E5%8E%9F%E7%94%9F%E6%8E%A5%E5%8F%A3) | [**lgb**](https://github.com/Jie-Yuan/DataMining/blob/master/5_PopularAlgorithm/1_Boosting/1_lgb/README.md#1-%E5%8E%9F%E7%94%9F%E6%8E%A5%E5%8F%A3) | [**xgb.sklearn**](https://github.com/Jie-Yuan/DataMining/blob/master/5_PopularAlgorithm/1_Boosting/2_xgb/README.md#2-sk%E6%8E%A5%E5%8F%A3) | [**lgb.sklearn**](https://github.com/Jie-Yuan/DataMining/blob/master/5_PopularAlgorithm/1_Boosting/1_lgb/README.md#2-sk%E6%8E%A5%E5%8F%A3) |
| 训练树的数量 | num\_boost\_round=10 | num\_boost\_round=10 | n\_estimators=10 | n\_estimators=10 |
| 树的最大深度 | max\_depth=7 | num\_leaves=2\*\*7 | max\_depth=7 | num\_leaves=2\*\*7 |
| 叶子节点停止分裂所需包含的最小样本数 hession矩阵加起来 | min\_child\_weight=5 | min\_child\_weight=5 | min\_child\_weight=5 | min\_child\_weight=5 |
| 随机从样本中抽取比例建子树 | subsample=1 | bagging\_fraction=1 | subsample=1.0 | subsample=1.0 |
| 随机从特征中抽取比例建子树 | colsample\_bytree=1.0 | feature\_fraction=1 | colsample\_bytree=1.0 | colsample\_bytree=1.0 |
| 叶子节点数目正则系数 | alpha=0 | lambda\_l1=0 | reg\_alpha=0.0 | reg\_alpha=0.0 |
| 叶子节点分数正则系数 | lambda=1 | lambda\_l2=0 | reg\_lambda=1 | reg\_lambda=0.0 |
| 可以使learningrate随着训练渐渐减小 | eta=0.1 | learning\_rate=0.1 | learning\_rate=0.1 | learning\_rate=0.1 |
|  | booster='gbtree' | boosting='gbdt' | booster='gbtree' | boosting\_type='gbdt' |
| 目标函数 | objective='binary:logistic' | application='binary' | objective=**'binary:logistic'** | objective='binary' |
| 节点分裂所需的最小损失函数下降值 | gamma=0 | min\_split\_gain=0.0 | gamma=0 | min\_split\_gain=0.0 |
| nagtive\_samples/positive\_samples 在不均衡二分类问题中，给正类样本的权值 | scale\_pos\_weight=1 | scale\_pos\_weight=1 | scale\_pos\_weight=1 | scale\_pos\_weight=1 |
| 在不均衡二分类问题中设置为true，但是使用中并没有提高模型性能 |  | is\_unbalance 默认 false |  |  |
|  | seed | bagging\_seed | random\_state=888 | random\_state=888 |
| 采用CPU的核数量 | nthread | num\_threads | n\_jobs=4 | n\_jobs=4 |
| 验证集所用评价指标 （fit函数参数，交叉验证） | eval\_metric=**'logloss'** | metric**='binary\_logloss'** | eval\_metric=**'logloss'** | eval\_metric=**'binary\_logloss'** |
| 连续多少次，在校验集上性能不再增加就停止 | early\_stopping\_rounds | early\_stopping\_rounds | early\_stopping\_rounds | early\_stopping\_rounds |
| 默认0，bagging 的频率，0意味着没有使用bagging，k意味着每k轮迭代进行一次bagging，bagging\_freq和bagging\_fraction需要配套使用 |  | bagging\_freq |  | min\_data\_in\_leaf |
| 训练好的模型输出路径默认 LightGBM\_model.txt |  | output\_model |  |  |
| 模型输入路径 |  | input\_model |  |  |
| 0,1,2意味着数据中第1，第2，第3列特征是类别特征 name:c1,c2,c3 意味着数据中特征名字c1,c2,c3 是类别特征 |  | categorical\_feature |  |  |
| 模型的输出不再是结果而是样本所掉落在树的叶子中叶子的索引 |  | predict\_leaf\_index |  |  |