Data-centric AI之数据集质量

正如之前提到的，我理解的Data-centric AI的核心之一是数据集质量。数据集的质量再如何强调都不过分，我认为在数据这个领域，数据集的质量就是第一要务。对于机器学习来说，没有高质量的数据集作为前提，模型学习不到有用的知识，也就是所谓的“垃圾进，垃圾出”。数据集的质量是个很大的话题，本文根据我在多个计算广告和推荐系统的项目中的实战经验尝试总结一下**，其实对于结构化数据建模来说，基本上下面谈到的内容都是通用的**。

项目中发现很多客户都是在发现模型效果（包括离线效果和线上效果）不理想或者在训练过程中遇到一些错误时才会去检查样本和特征，这个是不建议的做法。最建议的做法还是在开始训练之前和线上预测时就对数据集的质量进行严格检查（可能这样做的话，前期的投入是比较大的，但是只有这样，之后训练完的模型的离线效果才更可信；否则遇到问题再回朔来检查数据集的做法看似投入小，实则是定时炸弹，你永远不知道它什么时候爆炸！）。

数据集的质量包括但不限于如下的几种（**我认为以下四个是最重要的**）：

**异常样本以及异常特征的识别和处理（线下必须做，线上尽量做）；**

**Label的正确性问题（线下必须做）；**

**特征的覆盖度问题（线下必须做）；**

**样本和特征的正确性检查（线上尽可能做，线下必须做）；**

这里就每个问题在客户项目中遇到的情况来讨论一下：

1. **异常样本以及异常特征的识别和处理**：

用数据集来训练模型，我们希望送入模型的数据是干净合理的，因此**首要的任务就是把原始的数据集进行清洗，把异常样本和异常特征识别出来并做相应的处理**。

异常样本和异常特征的识别属于异常检测的范畴，经常用统计方法，数据分析或者机器学习的方法来识别异常（**三者结合起来做异常检测的效果最好**），感兴趣的同学可以参考我总结的异常检测的[文章](https://github.com/yuhuiaws/ML-study/blob/main/ML%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0%E5%85%A5%E9%97%A8/%E5%BC%82%E5%B8%B8%E6%A3%80%E6%B5%8B%E4%BB%8B%E7%BB%8D.pptx)。

异常点的归因是特征粒度即异常特征的话，经常用的处理方式就是把异常值当做缺失值处理，或者使用3-sigma上下界截断处理，最大最小值截断处理以及分位数上下界截断处理（参考Amazon SageMaker Data Wrangler中关于异常值的处理）；异常点的归因如果是样本粒度即异常样本的话，常见的方式就是从该数据集中剔除该异常样本（**剔除并不是丢弃，建议的方法是把异常样本单独存储以便之后做异常检测任务**）。

**项目中遇到的几个很典型的bad case**：

客户XXX，通过统计分析的方式（**统计训练集中的category特征的每个值的样本频次，并分析它的分布；从不同的维度比如时间维度来更细粒度的查看每个设备id的点击次数**）发现数据集中每天某个时间段的某几个设备ID会有非常多的点击数据，远远超过每个设备每小时平均点击次数。

之后通过把这些有问题的样本从数据集中剔除掉并训练模型，得到的模型的线上效果有明显提升。

客户YYY，通过样本抽查的方式发现很多样本的转化归因有问题（更好的方式是**通过业务逻辑/规则来判定每个样本的转化归因是否合理**）。

之后通过对有问题样本的重新转化归因来调整样本集并重新训练模型，离线效果和线上效果都明显提升。

客户ZZZ，进行常规的连续型特征的取值范围检查，发现数据集中的历史CTR特征的取值非常离谱，对其进行矫正后，离线模型的效果有提升。

1. **Label的正确性问题**：

**Label就是模型学习的指导/监督信号，所以保证label的正确性是显而易见并且非常重要的。**但是保证样本的label的正确性不是看起来那么容易的，在搜推广三大领域的排序任务的数据集中经常会遇到如下几种情况**：**

**转化延迟**：

这个主要是计算广告的DSP的CVR/IVR预估模型的数据集中会遇到。

在这样的情况下，要对CVR预估/IVR预估来建模的话，就需要权衡样本的正确性和时效性了。注意：在之后拿到转化延迟的数据后，记得重新给之前对应的同一个样本重新修改label（实现时每个样本可以设置唯一id，然后通过id来对齐）。

关于转化延迟问题，这里有两种选择：

**1） 完全剔除转化延迟时间窗口内的所有数据，在当前不作为训练集和验证集的一部分**：

这个方法能保证样本的label的正确性，但是因为没有近几天的数据作为训练集和验证集，因此数据的时效性差一些，可能对模型的效果有影响。（这个方法在腾讯2017年计算广告APP CVR转化率预估的决赛TOP10获奖团队的方案中，基本都采用的这个方法。）

2） **不管转化延迟问题：**

这个方法主要考虑的就是数据的时效性，不浪费最近新增的数据，但是有些真实的正样本就被wrong label为负样本了。（我接触的DSP客户基本大家都用这个方法）

思考：如果因为转化延迟导致的有问题的样本占的总的负样本的比例很少比如10%，那么是否还需要关注这个问题呢？

即使因为转化延迟导致的有问题的样本占总的负样本的比例很少比如10%，也仍然需要考虑转化延迟问题，因为这个比例的样本（本应该是正样本）可能与正样本的数量相比已经是很大了。

那是否可以对上面两个选择做个折中呢？

假设：CVR预估建模，用T+1训练方式（T取值为7天），转化延迟时间窗口是3天。那么数据集可以这样来构造：

对于验证集：离当前最近1天的所有正样本和负样本

对于训练集：验证集那天前推7天的所有正样本 + 验证集那天往前推2天的负样本做采样 + 最老的5天的负样本。（注意这里做了采样，在DSP排序任务中使用ecpm公式的时候需要校准pcvr，采样率按照采样后的负样本数量除以不采样的负样本总数量得到）

**那具体如何对负样本来进行采样呢**?

**每天按照固定采样率做等量采样**

**按照时间的权重来增加采样**

在确定整体采样比例情况下，根据总的负样本量计算出需要采样的量，然后计算每天的采样比重。需要满足的条件：离当前越近，采样比重越大。

**误点击**：

由于终端用户不小心点击item或者广告生成了点击日志。

比如在客户XXX的DSP项目中，客户的开屏广告（开屏广告就容易让人误点击）日志中发现了很多疑似的误点击。

我们这里需要合适的代码逻辑来判定当前的点击事件是否是误点击，或者通过埋点SDK来判定是否是合理的点击，如果是的话才写点击事件到日志中。

**点击进去后才发现并不是真正的感兴趣**：

这个情况在推荐系统和个性化搜索领域中比较常见。

比如在长视频/电影推荐中，某用户对某个推荐的item点击进入并观看了5秒就关闭，那么可以认为该用户对这个item并不感兴趣，因此这样的点击对应的样本的label需要标注为0而不是1（对于这个场景，可能用播放比率和播放时长做条件或的方式来打label可能更合理）。

**是否re-label导致的问题**：

使用T+1训练方式时，经常会遇到同一个设备id或同一个userid对同一个item或者广告素材在多次曝光行为下会有不同的点击行为或者转化行为。

由此带来的现象就是在比如7天的训练日志中，除了时间戳，会有其他特征完全一样的样本出现，但是前几次label为0，最后一次label为1。这种情况下，是否需要把前几次的label修改为1，也就是所谓的re-label问题（这是来自一个真实的客户XXX的问题）。

在这种情况下，建议的处理方式：

1. **不要对反复曝光的样本做re-label，也就是遵守该用户的真实点击行为或者转化行为就可以**。

这个情况在很多客户的项目中都出现过，甚至有的客户说，他们做过统计，item或者素材被点击需要平均曝光7次；转化的话需要平均曝光20次。**从这个角度来说，太严格的过滤模块不一定对业务就好**：比如对于某个用户，在召回的结果中是否应该过滤掉该用户最近三个月曝光过或者点击过的item或者广告素材。**太频繁的曝光肯定是需要过滤的（否则对终端用户不友好），这个频率需要具体根据业务来确定**。

1. **可以增加一些其他的特征来让样本有辨识度（目的是在label取值不同的情况下，让特征向量对于模型来说区别比较大）**。

比如可以利用时间戳来把一天分为24个桶作为一个离散特征；还可以把对应userid/设备id最近一段时间的安装列表以及点击列表作为强特征加入模型。

1. **特征的覆盖度问题：**

特征的覆盖度（**Category特征和连续特征都有覆盖度的问题**）尤其是训练集中的特征的覆盖度，**最终会反映到模型对该特征能了解的程度**。

理想情况下，**我们希望训练集中的每个样本都没有缺失值，并且训练集会把每个特征的所有可能的取值都能覆盖到并且每个取值出现的频次足够高并且频次有上限**（为了样本公平性），这样模型学习的才够充分，之后对验证集以及线上数据做推理才更准确。

现实中，每个特征总是或多或少会有覆盖度的问题。Category特征的覆盖度问题更复杂和常见，因此我们这里主要讨论Category特征的覆盖度问题（**连续特征通过特征缩放处理或者离散化可以把它的覆盖度问题缓解或者转移**）。导致特征覆盖度问题的经常会有如下的3种情况：

1. **特征获取困难**：

在项目的过程中，经常有很多特征的获取是比较困难的。具体又分为2种情况：

**涉及到隐私的特征**：

这类特征比如身份证号，职级，薪资水平等等是比较隐私的，一般终端用户不会提供给你，或者即使提供也可以认为不是真实的信息，基本可以忽略。

在当前个人隐私越来越重视的年代，在某些业务中，用户的行为特征也作为隐私来对待，有些用户会授权你的业务可以使用他的行为特征，有些用户不授权你使用他的行为特征，那么这个行为特征就会存在部分缺失的情况。在这种情况下，可能把这两类用户群分别单独建模更好，那个可以使用用户行为特征的模型能学习的更充分。

**非隐私特征的特征值部分缺失**：

如果某特征的部分值由于某种原因不能获得，那么就当作缺失值来处理。

如果训练集中的某特征的缺失比例很大比如超过60%，那么可以考虑剔除该特征；如果训练集中的某特征的缺失比例比较小比如小于40%，可以考虑对该特征做缺失值填充；如果训练集中的某特征的缺失比例接近50%，是选择填充缺失值还是简单剔除该特征，最好分别建模然后通过线上AB test来看效果。

对于样本粒度来说，需要逐个检查每个样本的缺失情况。对于某个样本，如果缺失的特征数量太多比如超过1半数量的特征都缺失，那么可以考虑是否可以把这个样本给去除。

在一些客户的项目中**，**发现他们尝试通过其他模型来生成一些用户的隐私特征，并且把这些特征用在下游的模型中：

有些特征比如男女性别，年龄段，是否有孩子等等这样的特征对于某些目标任务来说比较重要，但是这些特征很难从终端用户真实的获得。因此有的客户就通过规则/数据分析/机器学习的方式来根据该用户的行为来生成这些特征。

本身这个生成的特征的正确性就没有办法保证，那么把这个生成的特征放入下游的模型中，对模型是有帮助还是有负作用是需要离线评估以及线上AB test来评价的。

1. **非高基数id类特征的category特征（比如“国家”这样的特征）分布可能天然就不均衡**：

如果某些特征值的样本数很低比如少于10次，那么可以考虑特征向上合并或者把那些小类别统一归并为”Other"。

所谓的特征向上合并，指的是可以重新规划该category的类别，把细类别合并为粗类别，从而改善特征的覆盖度。

有个例外是比如约会类app，有的“城市”比如“合肥”的样本少，如果把这些小样本的城市都合并为一个other或特征向上合并，会违背了同城约会可能性更高这样的事实，这个时候可能就不要合并了。

1. **低频的高基数id类特征值被过滤**：

就是把包含长尾id的整个样本从训练集中干掉了。

对于搜推广三大领域来说，像userid/deviceid/itemid这样的特征是典型的高维稀疏特征，他们的长尾的id有很多。长尾的id常见的处理方式有如下三种：

**过滤掉低频特征的样本**：

**这个方案是我见过的客户用的最多的**。其最简单的方式就是设置一个频次阈值比如8，少于这个频次的样本就不放入当前这次训练集（下一次训练的时候可能这个id的样本的频次够了，所以低频样本不要丢掉，还是需要保存起来的）

优点：

可能会避免了模型对长尾id样本的过拟合

缺点：

需要有相应的逻辑来判断是否是长尾id，因此做线上serving时会带入额外的延迟。

**线上serving时，对于出现的长尾id还需要特殊的处理**（比如走单独的链路，不走模型）。

**使用other-id的方法**：

这个方法同样需要设置一个频次阈值，把低于这个频次的id值替换为other-id，也就是所谓的长尾id和冷启动的id共享一个other-id值。

优点：

因为other-id的存在，线上不会出现id OOV的情况。

缺点：

需要维护一个字典做长尾id到other-id的mapping，并且这个字典会根据训练集中的id的频次变化需要经常更新。

线上推理时，同样需要做这个mapping，带入额外的延迟。

**不考虑频次并使用hash trick的方法：**

对id类特征不做频次过滤，直接对id类特征做hash bucket映射。

优点：

不需要额外的字典来映射；线上不需要处理id OOV的情况；

线上处理逻辑相对比上面两种方案更简单。

缺点：

可能会有hash冲突；

对长尾的id的推理效果可能不好；

由此可以看到，**如果特征的覆盖度问题是因为低频的高基数id类特征被过滤后导致的，那么可以考虑：**

特征的频次阈值是否可以根据业务特点来重新调整；  
重新评估并权衡其他的长尾id的处理方法（可能评估完后会仍然使用低频特征被过滤的方案）。

另外一个相关的问题是，对于训练集中有的用户的样本很多，有的用户的样本很少，这样模型可能会被样本多的活跃用户所影响，如何缓解这样的问题？

Youtube电影推荐深度模型中采用的方法是**为每个用户固定样本数量上限**，尽量平等地对待每个用户，能明显提升线上效果。

1. **样本和特征的正确性检查**：

**这个看似最简单的事情，但是确实是最容易忽略，也是经常出问题的地方**。

**我见到的客户中，或多或少都会对样本和特征做一些检查，但是仍然检查力度太少**。

先不说对模型学习效果如何，就是出了问题debug也费劲（尤其是当出现奇怪的问题或者复杂的问题的时候）；而且经常会前功尽弃（比如训练到几个小时因为样本有问题报错退出，尽管通过保存checkpoint可以尽量降低伤害程度，但是仍然会浪费感情），对成本带来无谓的消耗。

因此，我建议**至少做如下的正确性检查**：

**样本级别的检查**：

比如tfrecord文件本身的检查，是否包括corrupt的样本；

我遇到过一个客户XXX在使用sagemaker pipe mode训练的时候，训练了挺长时间，后来因为某个tfrecord文件有问题导致整个训练任务失败。

**特征个数的检查**：

**检查每个样本包含的特征的个数是否是期望的个数**。

我遇到过一个客户YYY，在跑训练的时候总是遇到奇怪的报错。客户自己手动抽查了几个样本发现没有问题，后来对其中一个文件逐行肉眼检查发现有些样本的特征个数不对。

**特征类型的检查**：

**检查文件中的每个特征是否是我们期望的类型**（比如整数，浮点数，字符串等等）

**特征的取值范围检查**：

检查连续性特征是否是0，-1，NAN等特殊值；

检查连续性特征是否在期望的取值范围内；

检查离散型特征是否在期望可枚举的范围内（可通过维护一个字典来检查有哪些可枚举的离散特征）。

**单个特征的shape检查**：

**检查每个特征是否与我们期望的shape是一致的**

**多个特征的shape的对齐检查**：

比如一个安装列表的序列特征，填充后的长度是固定长度50；为了让填充的id不对训练造成影响并且不同的id对target具有不同的权重的话，一种简单方法是提供序列中每个id的weight，填充的id的weight设置为很小的一个值比如0.000001（坑：这里设置为0的话，在TF中使用weight feature column的时候会报错！另外一个坑，weight以后使用share embedding column的话，TF1.X是可以的，但是用TF2.X版本用类似的API tf.feature\_column.shared\_embeddings就会报奇怪的错误，TF2.X使用tf.compat.v1.feature\_column.shared\_embedding\_columns API则是可以的）。因此这里需要检查每个样本中的该序列特征填充的id个数和对应的weight为0.000001的个数是不是相等。

**Label取值的检查**：

**检查每个样本的label的取值是否是期望的范围**。

**建议**：

1. **每个公司对每个数据集都建立和维护一个checklist**.

这个checklist就是对这个数据集的所有的卫生检查。当在日后的迭代中发现了新的检查规则后就加入到这个list中。

1. **对样本和特征的这个卫生检查，建议不要放在训练脚本中去做，而是用一个单独的前置任务去做这个事情，不要浪费宝贵的训练资源**。

在之前的很多项目中，发现很多客户可能为了省事，喜欢把这些检查还有一些统计样本个数等的事情放在训练脚本中。这个不是建议的方式。

1. **对特征的检查，线上和线下都需要尽可能做**。

对于线上的检查来说，尤其是特征取值范围的检查，如果发现不正确，需要根据策略比如使用缺省值来替换。

**总结：**

**线上和线下都应该尽可能对数据集的质量进行检查。线下的时候，这个检查并不只是发生在数据集清洗的开始阶段，其实只要对数据集进行了修改（比如在异常样本/特征的识别和处理后增加了新的交叉特征），那么都需要再做一遍质量检查的。**