排序模型调优实践

1. 客户ML项目的上下文：

背景：计算广告DSP做IVR预估（**即从曝光到转化的预估**）的排序模型

模型选型：wide & deep模型

使用平台，框架和API：SageMaker + TF2.x + tf.keras API + tf.dataset API

训练迭代方式：T+1训练方式（客户的T用的是30天训练集+1天验证集）

数据量级：训练集400GB+，验证集10GB+

需求：模型对验证集的AUC离线指标能达到0.75+才能上线做AB test。

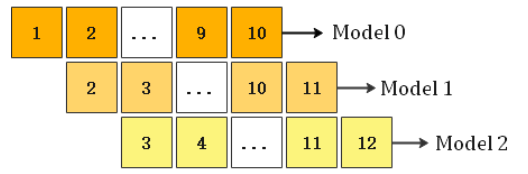
痛点：最开始的验证集的AUC只有0.6左右；**留给中国队的时间不多了**。

1. 背景知识简单介绍：
2. CTR，CVR，IVR的概念
3. 关于排序模型的选择

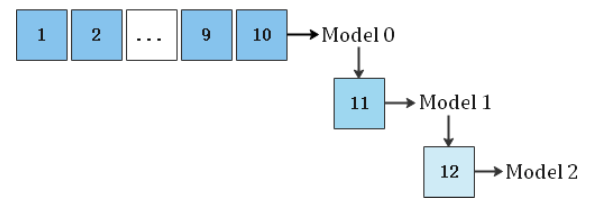
LR/GBDT，LR+GBDT，FM/FFM, DNN（wide & deep，DeepFM，DCN等等 ）

1. 计算广告排序模型的常见的训练迭代方式（参考华为的CTR预估模型的训练方式<https://www.ershicimi.com/p/30eae0af5c23231ea0ae452d832751c0> ）：

A．T+1固定时间滑窗内的全量数据来训练（**这个方式更多的客户在用**）



B．基于之前训练的模型对当天的数据进行增量训练：



C. 综合A和B的方式，即当天内做增量训练（比如小时级别的增量训练），当天结束的时候做T+1全量训练。

1. 关于排序模型的训练集，验证集的划分：

必须按照时间来划分吗？

**对于CVR和IVR，是否需要考虑反馈延迟的问题**（CTR几乎不会有这个问题）**？**

**如果没有考虑反馈延迟，如何对之前错误标注的负样本模型学到的知识进行修正**？

一个简单可行的方式是记录当天的没有获得反馈的样本，并标注为负样本；之后用最终延迟获得的反馈来修正模型，即构造一个和之前错误标记的负样本一模一样特征的正样本，并增加这个样本的sample weight来重新扔给模型来学习。

这个方法是从知名博主王喆的博客中看到的，也是天猫的做法，参考<https://zhuanlan.zhihu.com/p/75597761> 中的评论部分。（这个所谓的回刷模型操作，更多是用在online learning和增量训练的场合，对于T+1训练场景不太适用）

1. 天然的正负样本比例非常的不均衡（**1：3000这样的比例很常见**）
2. 对于业界常见的计算广告的排序公式ecpm，如果是CPC（按照点击计费）出价方式，ecpm = pctr \* CPC\_bid，如果是CPA（按照转化计费）出价方式，ecpm = pivr \* CPA\_bid。对于更复杂的排序公式，还会考虑**价格挤压因子**或者**智能出价因子**等。客户这里使用的是pivr \* CPA\_bid排序公式。
3. 过程重放：
4. **样本类别不均衡处理尝试**（对于排序任务来说，常见的是如下几种）：
5. 对负样本降采样：
6. 会浪费掉很多负样本

在客户这个项目中，我们并没有尝试这个方案。

1. 最后做ecpm排序计算的时候需要对pivr做校准（由于采样前后会改变pivr的绝对值从而改变排序结果，因此需要校准），矫正公式（参考Facebook的GBDT+LR那个paper）：

q = p/(p + (1 - p)/w)，w为负样本采样比例（比如采样比例是10%），p是排序模型对单个样本的打分概率，q是矫正后的打分概率。

上面说的是对于计算广告领域的排序任务来说，在对负样本做了采样以后需要对打分的概率进行校准（校准发生在线上推理的时候，离线训练的时候不用考虑校准）。对于个性化搜索和推荐系统来说，他们只关注打分概率的相对顺序，而采样前后的打分概率的相对顺序不会改变，所以不需要校准。

1. 对正样本过采样：

如果把正样本过采样到和负样本差不多的量级，会导致总体训练样本太大，训练时间相对变长很多。

对正样本如果进行了采样，在计算广告的排序任务中应该也需要对打分概率做校准，而个性化搜索和推荐系统中的排序任务不需要对打分概率做校准。

我们在客户项目中尝试这个方案，模型的离线效果还不错（通过简单复制正样本把正样本数量变多），就是训练时间相对长（一个epoch单机跑都要4，5个小时）。正因为这个原因，我们最终没有使用这个方案。

1. **为了不浪费负样本，每个epoch采样出和正样本1：1的数量，epoch的数量可以根据负样本数量和训练时间来权衡**。

**坑**：尝试使用tf.keras + tf.data.experimental.sample\_from\_datasets API从大量的负例采样，并与正例拼接为最终训练集，发现每个epoch后的验证集的AUC不变化。如果说模型已经充分收敛也就是loss基本不变了从而模型参数基本不变化，这个时候验证集的AUC不变化是有可能的，但是模型在刚开始的那些epoch都应该能看到不同的负样本，应该会继续学习而不会收敛，因此在开始的几个epoch验证集AUC应该是变化的。到这里，我推断可能是TF的坑，可能每个epoch都拟合了一样的负样本并且基本上在第一个epoch以后模型就基本收敛了。

**证据**：

1. 从网上看到有类似的issue：<https://stackoverflow.com/questions/59138504/how-can-i-change-dataset-at-every-epoch-in-keras>
2. 查看TF的源码（https://github.com/tensorflow/tensorflow/blob/v2.3.0/tensorflow/python/keras/engine/training.py#L824-L1146 ）：

tf.keras的fit()---> data\_adapter.DataHandler(https://github.com/tensorflow/tensorflow/blob/v2.3.0/tensorflow/python/keras/engine/data\_adapter.py )的\_\_init\_\_函数-----》select\_data\_adapter函数，这个函数对于tf.dataset会返回DatasetAdapter类，然后调用DatasetAdapter的初始化函数\_\_init\_\_，而DatasetAdapter的\_\_init\_\_函数中有如下的注释：

**# Note that the dataset instance is immutable, its fine to reuse the user**

**# provided dataset.**

self.\_dataset = x

**也就是说在tf.keras fit API的每个epoch中会一直重用传入的dataset，也就是每个epoch的dataset是一致的**，这也就印证了我们的推断是正确的。所以我们只好放弃这个方案。

1. 使用class weight或则sample weight让模型更关注正样本：

思考：Class weight一定要设置为正负样本比例吗？

最后项目上线使用的方案：

1. tf.keras的fit API有参数class weight可以方便的使用；
2. 为了每个step都尽可能有足量的正样本送入模型，shuffle操作和更大的batch size是需要的（batch size尝试过16K，64K）。
3. 正样本的class weight设置为（负样本/正样本的比例） \* 0.8
4. 使用Focal loss

Focal loss可以做困难样本学习和缓解类别样本不均衡。（时间关系，这个项目没有尝试）

1. **调试过程中很快出现了欠拟合**：

欠拟合的常见处理方法：

* + **用更多和更好的特征**

1. 根据业务语义加入更多的设备侧和广告侧的交叉特征。
2. 连续特征的特征缩放（**非常重要**）（深度模型对于连续特征的幅度变化很敏感，所以用深度模型建模的话一定要对连续特征做特征缩放处理）：

Z-score，minMax，log（**这个项目采用了这个方法**），平滑方法（比如贝叶斯平滑）等等。

项目中尝试了多种归一化，包括Z-score，log。对客户的数据集和模型来说，log表现要比Z-score要好。

注意：使用log函数时是针对的非负数的连续特征，而且需要做一下平滑，防止log0出现；平滑的值不要太大，可以考虑1e-7。

对于历史ctr这样的比率特征，在排序任务中经常会考虑对历史ctr特征做平滑，目的是为了让那些历史ctr相等但是曝光次数多且点击次数也多的item经过平滑后得到的值与那些长尾的item区别开来（比如7天内点击1次，曝光2次的item与7天内点击50次，曝光100次的item，虽然他们的点击率相同，但是他们的受欢迎度差别很大，因此这个时候直接用7天内的ctr作为特征对他们来说没有辨识度，这个时候做ctr平滑就很有意义）。由于时间原因，这个项目中并没有对历史ctr特征进行贝叶斯平滑，这个是以后一个优化的点。

* + **增加模型复杂度/容量**

1. 把全连接层层数变多或者每层的神经元数量变多。

业界一般做排序模型用到的全连接层基本都是3层。我们在项目中尝试的是把每层的神经元变多。

1. 把embedding vector size/dimension变大

注意：看到的很多文章对于排序模型中用到的usrid/itemid的embedding table的embedding向量的维度一般都是设置为8或者10（算是一个经验值，在其他项目中也见到一些客户是这样的设置的）。

除了上面提到的这些缓解欠拟合的方法，调整学习率和batch size大小，以及样本类别不均衡的处理方法都可能缓解欠拟合。因此可以看到，缓解欠拟合的组合因素有很多。建议的方式是每次调试只是改变一个因素来进行训练后效果对比（在客户的项目中，经常要同时跑多个实验来进行对比）。特征缩放肯定是要先做的，除了学习率，batch size以及样本类别不均衡处理这三个因素外，我们可以循序渐进的尝试下面的方法来缓解欠拟合：挖掘一些好的特征（不包含交叉特征，比如一些历史统计特征），接着增加模型复杂度（主要是增加层数或者神经元个数），最后逐渐增加有意义的交叉特征。

1. 欠拟合没有了之后，又发生了过拟合：

过拟合指的是模型在训练集上的效果/表现不错，但是在验证集上的表现与训练集上的差距很大。在实际生产项目中，我们更关心模型在训练集上效果不错，在验证集上的效果也不错，在这样的情况下即使过拟合我们也不关心，比如训练集上的AUC是0.95，验证集上的AUC是0.8，那这种情况是属于过拟合了，但是验证集的AUC也足够高了，所以这个情况我们能接受；如果是训练集上的AUC是0.95，验证集上的AUC是0.6，这种过拟合就是我们需要关心的了。

**欠拟合到过拟合，有时候就是一瞬间的事情**（比如特征一下子加多了就容易从欠拟合变成了过拟合）

对于使用深度模型做排序任务的场景，过拟合常见的处理方式：

1. **收集更多的数据**：

目的是让模型能更多的见到不同的数据分布，从而学习到不同的知识。比如T+1训练中的T常见的是7天的数据作为训练集（当然这个T取多少和训练集中的正样本量有多少有很大关系），但是在我们这个项目中，T取的是30天的数据，因此能获得更多的样本尤其是正样本。

1. 减少模型复杂度/容量：也就是使用小一点的神经网络，包括小一点的embedding table。

在实际的项目中，我见到过有的客户把itemid/userid embedding table的embedding向量的长度设置为几百几千的，不建议这样，太容易过拟合了，就像前面提到的，设置为8左右就是一个不错的起点。注意这里的embedding指的是input embedding，而关于output embedding以及文本embedding向量长度的选择可以参考我的另一个文章“推荐系统概览”。

1. 使用BN（本质是对神经元的激活值进行整形**，它在DL中非常有用，建议尽量用**）：
   1. **使用BN的话，batch size不能太小**，而**batch size调整一般伴随着同方向的learning rate的调整（**也就是把batch size调大的话，learning rate可以适当调大一点点**）**。
   2. 虽然BN主要是在CNN卷积层用的比较多，但是MLP层也可以用。RNN的话要用LN

项目中使用BN后的效果很明显。

1. 使用early stopping早停：

早停需要monitor训练集还是验证集的metric？

建议：monitor验证集的metric（这个项目也是这样做的）。

Early stopping并不是必须的，如果设定模型固定跑的epoch数量，之后选择一个表现最好的epoch的checkpoint也是可以的，这个情况下就不需要early stopping。

1. 正则化手段：
   1. Dropout
   2. L1/L2正则
   3. Label标签平滑（当前项目没有使用）

Dropout的比率以及L1/L2正则的超参数在调试的时候，都要小步调整，大幅调整很容易一下子就从过拟合到了欠拟合了。

1. 使用更少的特征：

在这个项目中，一下子增加了几种交叉特征后，模型从欠拟合到了过拟合。然后在去掉了几个交叉特征之后，过拟合得到缓解。因此加入新的特征要一点点加，小步走。

在使用深度模型发生过拟合的时候，首先要检查验证集的数据分布（比如每个连续特征的统计分布，每个离散特征的覆盖度，和训练集中的数据分布做一下对比）。如果训练集和验证集的数据分布相差很多，考虑如何重新构造训练集和验证集；否则建议尝试按照如下的顺序来缓解（每做完一步就训练看效果，如果验证集的效果能接受了，就先打住；否则继续下一步）：使用BN（基本上标配）---->使用更少的特征（如果特征本身就不多，可以跳过；主要关注交叉特征是否很多）--🡪收集更多的数据（如果正样本量已经足够多，可以跳过这步）--🡪使用正则化方法---🡪减少模型复杂度/容量（尤其要注意embedding table中embedding向量的长度）

1. 项目中遇到的其他问题：
2. **数据集变了，模型的离线评估AUC变化很大**。
   1. 数据集变大可能会导致容量小的模型效果变差，发生欠拟合。

建议：对于**CTR/CVR任务，训练流程跑通以后，用固定滑窗的数据集来训练调试模型；而一般固定滑窗内的数据集的量级差不多**。

* 1. **数据集的清洗和预处理每天都要保证一致性的行为**，否则出问题调试很花时间。
  2. **要尽量保证特征的线上线下一致性**（更多细节可以参考我的另一个文章“Data-centric AI之数据集的质量”）。

1. 同样的数据集和同样的模型，两个实验对比，发现对验证集的评估指标AUC有一些差别：

ML带入的随机性很多，所以做好在上下文尽量一致的情况下对比：

包括超参数的设置，训练任务的相关参数和随机种子fix（**这个非常重要**，包括python random的seed和tensorflow.random的seed都需要fix）。

1. **经常发现在分布式训练中模型的评估指标比单机训练的评估指标要差**：

这个是很常见的。

1. 使用分布式训练甚至只是单机多卡的时候，学习率可能不适合还用单机单卡训练的学习率，适当需要调整。
2. 对于horovod，一般需要把学习率变大一点，不能完全按照horovod官网建议的那样即用worker数量乘以之前单机单卡的学习率作为调整后的学习率（这个可能会得到很大的学习率，从而导致模型学习效果不好）。。
3. 对于parameter server的异步梯度更新方式，可能需要把学习率调小，为了让最慢那个stale model replica的更新对整体的影响小一些。
4. 特征的覆盖度问题：

如果某些离散特征的特征值的样本出现频率很低比如少于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"。这个方法在客户的另一个项目LightGBM模型中使用了，效果不错。对于那些长尾的itemid或者长尾的usrid，也可以考虑把这些长尾的id的样本给剔除。更多详细的关于特征覆盖度的问题，可以参考我的另一个文章“Data-centric AI之数据集的质量”。

1. **Hash trick的使用：**

细分为hash后**提升维度**和**降低维度**两个方向：

1. 如果单个离散特征在训练集中出现的唯一值个数不大（也就是说基数小），可以利用特征hash来分桶，桶的大小可以考虑几倍于训练样本中该离散特征出现的唯一值个数。

这个方法的目的就是，在尽量降低hash冲突的情况下，不用提供额外的映射表就可以在hash分桶后直接embedding（因为embedding table的索引是从0开始的）。

当前项目中使用了这个方法，对一些离散特征做了特征hash分桶（hash桶的size是离散特征的基数的3～5倍）。

1. 反过来，如果单个离散特征（比如usrid或者itemid）在训练样本中唯一值数量非常大（也就是说基数很大），那可以使用特征hash来降维（这里有个隐含的假设，高基数id类特征的样本比较稀疏，多个不同id落入同一个hash桶之后会公用一个embedding向量，这个embedding向量可以装下多个不同id的知识。在排序任务中，这个假设一般是成立的）。

Hash bucket size的选择可以用训练集中的这个id特征的基数开4次根号为一个尝试起点。

当前项目中，对一些高基数的id类特征使用了这个方法来降维。

客户的另一个项目中想使用Amazon Sagemaker 内建算法object2vec学习item的embedding，但该算法对id有300w的限制，客户非低频的itemid超过300w，这个时候可以用特征hash来降维后再使object2Vec。

1. **模型上线做AB test，发现和线上另一个模型对比，线上效果一会好一会差**：

建议：如果可能，最好给它们相似的**AB test流量**（**即流量百分比和样本分布两个维度要相似**），尽量做到公平对比。

**AB test的流量分配不要和竞价策略耦合**。（客户之前的wide & deep模型和GBDT模型做AB test，因为竞价策略和AB test耦合在一起导致线上效果的剧烈抖动；逻辑拆分后，线上效果对比变得稳定，wide & deep模型表现持续要好过GBDT）