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滑窗内的数据全量训练（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+（验证集和测试集是完全一样的数据）

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（或者叫CTCVR）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CTR/CVR/IVR预估业务中的**负样本生成问题**：

A. 全部曝光但是没有点击/转化的作为负样本（**计算广告场景这个方式用的最多**）

B. 把曝光了且位于多个点击或者转化item位置之间的那些作为负样本

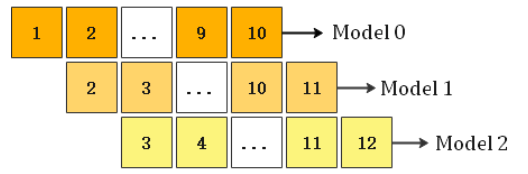
C. 对#A和#B折中，即负样本包括两部分：

一部分是#B的全部；

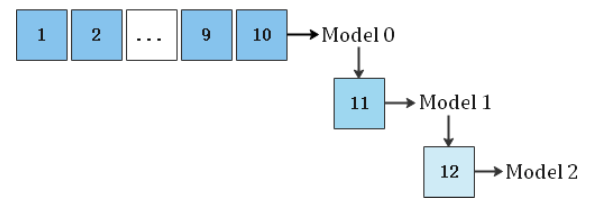
一部分是从最后一次点击/转化item位置之后的那些曝光但是可能没有被用户看到的样本中采样一些。

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，测试集的划分：

必须按照时间来划分吗？

验证集和测试集可以相同吗？**（验证集的目的就是验证不同的超参数；测试集的目的就是比较不同的模型。所以……）**

**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. 最终的**排序公式**用的业界常见的ecpm（**这个公式很重要**）：

最简单的ecpm的公式（假设是CPC出价方式）：ecpm = pctr \* CPC\_bid

这里不考虑更复杂的**价格挤压因子**，**pctr校准和智能出价因子等**。

客户这个项目用的是ecpm = ivr \* CPA\_bid

1. 过程重放：
2. **样本类别不均衡处理尝试**（在DL中常见的方式如下）：
3. 对正样本过采样：

负样本太多，这个方案会导致总体训练样本太大，训练时间相对长。

客户项目中尝试这个方案的效果还不错，就是训练时间相对长（一个epoch单机跑都要4，5个小时）。

1. 对负样本降采样**（业界经常采用的一种方案**）：
2. 会浪费掉很多负样本
3. 最后做ecpm排序计算的时候需要对ivr做校准（**目的是让正负样本比例的线上和线下的分布尽量一致**，这个思路是来自Facebook的），矫正公式：

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。

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。猜测可能是每个fit的epoch都是一样的数据，从而与我们的期望不符（提示：如果是使用tf.estimator API配合tf.data.experimental.sample\_from\_datasets来做的话，应该不会有这个问题）。

**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，这个函数对应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**，这也就解释了上面提到的那个问题。

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. 根据业务语义加入了更多的设备侧和广告侧的交叉特征。
2. 连续特征归一化（**非常重要**）：

Z-score，minMax，log（**这个项目采用了这个方法**），平滑方法等等。

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表示表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。

* + **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；因此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embedding向量的维度一般都是设置为8或者10（算是一个经验值）。

1. 欠拟合没有了之后，又发生了过拟合：

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

过拟合常见的处理方式（在DL中常用的方式）：

1. **收集更多的数据和做数据增强**

比如T+1训练中的T可以取得更大一些，获得了更多的样本。

1. 使用小一点的神经网络，包括小一点的embedding table。
2. 使用BN（本质是对激活值进行整形**，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**。
   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）。

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）
2. 使用更少的特征数量或者做特征选择

去掉了一些特征效果提升明显（数据清洗+去掉一些特征+log归一化后，离线效果提升10%）。

建议：**加入新的特征要一点点加，小步走**。

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. **数据集的清理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. **线上和线下的数据清理和预处理，特征工程都要尽量保证一致**。

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**：

这个是常见的，但是没有数学能证明！！

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训练的学习率，适当需要调整，还要考虑global batch size的影响。
2. 对于horovod，一般需要把学习率变大，都不能完全按照worker数量乘以之前单机的学习率。
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. **对于某个离散特征，在训练集中只是看到它的一部分值，但是线上预测的时候可能会有其他训练时没有看到的值出现**：

这个问题本质上是**特征值的覆盖度问题**，有几种方式可以处理：

1. 从业务语义层面把这个离散特征的多个值合并为一个值，**也就是所谓的特征向上合并**，尽量保证合并后的每个值在训练集中都有覆盖，最好是每个值都有足够的出现频率（这个方法在客户的另一个项目GBDT模型中使用了，效果不错）；
2. 除了覆盖度，还要考虑特征值出现在训练样本中的频率，如果太过低频比如出现了1，2次，可以考虑特征向上合并。
3. 对于训练集中的该特征的小于某个频次出现的所有特征值都作为other来处理。（这个在腾讯计算广告CTR预估比赛给的数据集中就是这样处理的）

比如只保留曝光最多的10k个campaign id且编码为0-9999，其余的id全部编码为10000（这个是微软的Deep cross network中提到的方法）。

**尽管other涵盖了更多的低频特征值，但是因为还有其他特征来区分不同样本，因此这种方式一般不会让模型效果变差。**

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，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离散特征出现的唯一值个数（为了尽量降低hash冲突）。

好处：不需要提供额外的映射表，因此速度快；训练和预测容易保持一致。

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3～5倍唯一值个数）。

1. 反过来，如果单个离散特征在训练样本中唯一值数量非常大，那可以使用特征hash来降维。

比如客户的另一个项目中想使用Sagemaker 内建算法object2vec学习item的embedding，但是该算法对id有300w的限制，而客户不是低频的itemid不只是300w，这个时候可以考虑用特征hash来降维后在使用object2Vec。

1. Hash一定程度上也可以缓解特征覆盖度的问题，但是建议还是优先使用上文提到的缓解覆盖度的三种方法。
2. 思考：直接embedding而不用hash不香吗？
3. **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）