客户ML case share 第2集

1. 客户ML项目的上下文：

需求：**对短视频内容做单标签多分类**（类别数量是27）；

这些类别是类似游戏，健康，健身，美女等等这样的。

使用平台和框架：SageMaker + TF

数据量级：7万+短视频（每个短视频都是几十秒）

痛点：项目时间比较紧张。

1. 过程重放：
2. 业务需求转化为ML建模的尝试：
3. 利用AWS Rekognition + 文本分类模型TextCNN

**这个思路最开始有ML SSA 郭韧提出（我觉得是一个有创意的思路，理论上也是可行的一种方案）**，步骤如下：

1. Rekognition抽取视频的字符串形式的标签：

这里Rekognition的输入是整个视频，输出的是每个帧的每个预测的标签和置信度。

需要思考的点：

Rekognition抽取的视频帧是**关键帧（即I-Frame）还是等频间隔帧**？（据说是每20ms的等频间隔帧）

**如果是等频间隔帧的话，对于一些没有什么动态变化的视频，那么多个连续间隔帧之间可能都是大量重复的标签**。重复的大量标签对下游模型的建模很可能会有影响；如果要去重的话可能需要额外的逻辑/任务来判断连续间隔帧的内容是否基本一样。

如何选取置信度阈值来排除掉Rekognition不置信的预测标签（客户项目中这里选取的置信阈值是0.5）？

**Rekognition内部训练的视频和客户的短视频是否差别很大**？

1. 把Rekognition经过步骤#1中可能的去重和阈值截断逻辑后得到的字符串化的标签，按照帧的时间顺序组成一个”文本句子”：

需要考虑的点：

**按照帧的时间顺序来组装为句子，更多是从直觉上来判断的。是否合理？**

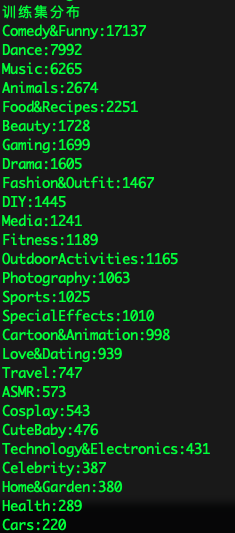
可能本身没有什么上下文依赖的标签却要送入下游的文本分类模型中建模，而使用TextCNN模型的话，一般的假设是词语之间是有上下文语义依赖的。

1. 把步骤#2中拼接后的”文本句子”送入TextCNN模型做multi-class任务。

**效果**：经过几轮训练的迭代，效果不好，客户决定尝试下面要提到的用3dCNN对短视频直接做multiclass建模。

**效果不好的可能原因**：

数据集的原因？（有可能。当前数据集的多分类是类别不均衡的，某个类别的样本数量是1万+，某个类别的样本数量是200+）如下图所示：



模型超参数调优？（有可能。因为时间原因，没有做更多的超参数调优）

Rekognition对客户的这个数据集给出的标签不准确？（有可能）

标签字符串的一些预处理不够好？（有可能，比如去重和阈值截断的逻辑）

组装的”句子”没有什么语义上下文依赖？（有可能）

1. 利用3DCNN对短视频内容做multi-class任务。

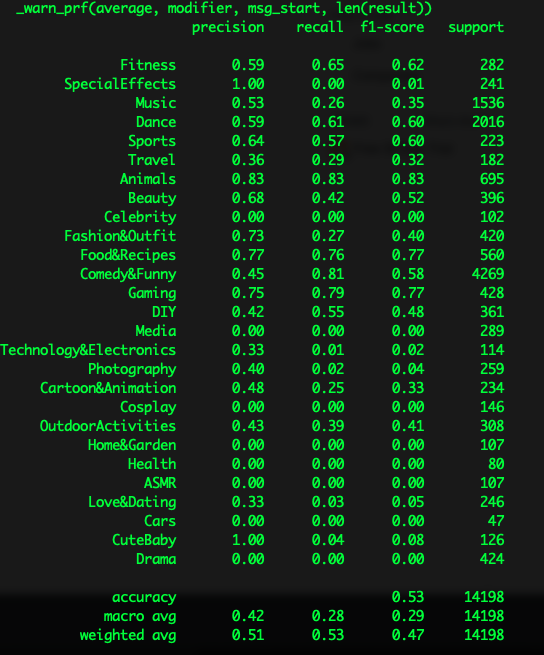
**为什么这里选择了3DCNN来建模**？

考虑到客户的数据集中的短视频的特点，不像一些动作变化频繁的运动类视频，因此没有使用很复杂的视频分类模型（比如slowfast，two-stream，I3D等等需要建模运动信息的模型）

**模型简单相对来说更容易调试，更容易在项目deadline内达到上线标准**。

1. 调试3DCNN模型
2. **选择合适的离线评估指标来看模型效果**：

对于multi-class任务，除了看整体的accuracy，macro F1-scroe，至少还需要看每个类别的精确率，召回率，F1-score等。类似下图：



1. 客户代码中实现了一个打印混淆矩阵的函数，在跑训练的时候发现每个epoch结束后内存使用都在增长，到最后OOM了。代码如下所示：

def \_print\_confusion\_matrix(X, Y, model, epoch, datatype):

y\_pred = model.predict(X)

max\_pred = np.argmax(y\_pred, axis=1)

max\_y = np.argmax(Y, axis=1)

cm = confusion\_matrix(max\_y, max\_pred)

print("{} confusion matrix of epoch {}:\n{}".format(datatype, epoch, cm))

report = classification\_report(max\_y, max\_pred, target\_names=list(label\_map.keys()))

print("{} classification report of epoch {}:\n{}".format(datatype, epoch, report))

从代码中看到在每次调用\_print\_confusion\_matrix函数的时候，会生成一些临时变量，**判断是python的垃圾回收不及时造成的**，添加如下的delete操作和手动调用gc.collect到这个函数中，尝试后解决了这个OOM问题。

del y\_pred

del max\_pred

del max\_y

del cm

del report

gc.collect()

思考：

**每个epoch做一下手动GC，开始担心会影响训练速度，实际在这个项目中对训练速度几乎没有影响**。

1. Bad case分析：
2. **为了缓解类别不均衡，这里尝试了class weight方式**，发现效果不好。

#1. 查看代码，发现超参数batch size比较小（开始设置的是64），这里判断可能的原因是由于batch size比较小，每个step可能有些小类别的样本取不到一个batch中，从而对模型学习不利。

Batch size调整到256做尝试后，发现效果有提升。

注意：**把batch size调大以后，一般都要伴随着适当调大学习率，但是学习率要一点点调大，否则对模型影响很大**。

#2. 查看代码，发现CNN层并没有使用BN做激活值整形，在#1的代码改动基础上每层的CNN都增加了BN，效果提升明显。

注意：**BN的放置位置有2种**：

第一种是在激活函数之前，第二种是激活函数之后。

第一种是原始 BN 论文提出的，另外一种是后续研究提出的，放在激活函数之后，不少研究表明将 BN 放在激活函数之后可能效果更好。（**只是可能，最好对于同一个神经网络用不同的激活函数配合不同位置的BN实际测试一下**）

**在当前客户项目中发现BN的放置位置对效果的区别不大**。

#3. 查看代码，当前把数据集分为训练集和验证集的时候，只是简单的随机切分。由于类别不均衡，这样的切分方式不能很好的保证训练集和验证集的分布大概一致，因此**使用按类别切分训练集和验证集的方式**，如下代码所示：

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X, Y, test\_size=0.2, random\_state=43, **stratify=Y**)

按类别来进行切分训练集和验证集以后，效果有一定提升。

1. 在使用class weight的情况下，经过了上面这些尝试，虽然效果有提升，但是过拟合还是比较严重。(尝试了dropout以及L1/L2等等正则化手段，效果都不好)。

#1. **从数据集下手**：

数据集如何抽帧？（这里用的是每个短视频等间隔抽帧，共抽20帧）

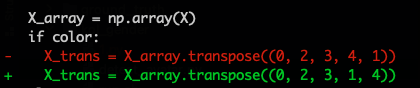
抽帧是不是太少？

抽帧后的每帧数据是否做了归一化？

把彩色帧变成灰度帧？（去掉颜色信号，原始代码的作者给出的benchmark都是灰度帧的）

通过增大抽帧频率，对每帧的像素除以255做归一化，以及变成灰度帧以后，进行训练，效果仍然没有什么改进。

#2. **客户的算法工程师发现是在对抽取的帧的shape进行处理的时候出了问题**：（**这里必须要对该算法工程师点赞！！**）



**Root cause分析**：

1. 从python opencv读取的单个视频做等频抽帧后的ndarray的shape是：

(帧 数，高，宽，通道数)

参考<https://blog.csdn.net/yuejisuo1948/article/details/80734908>

1. 所有视频组成的ndarray的shape就是

(视频数量，帧数，高，宽，通道数)

1. 使用tf.keras.layers.Conv3D()作为神经网络的第一层的话，需要设置输入参数input\_shape。由于TF默认是把通道数放到最后一个维度（data\_format的默认值就是channels\_last），也就是这里需要input\_shape为

（batch\_size，height，width，frame\_total，channels）

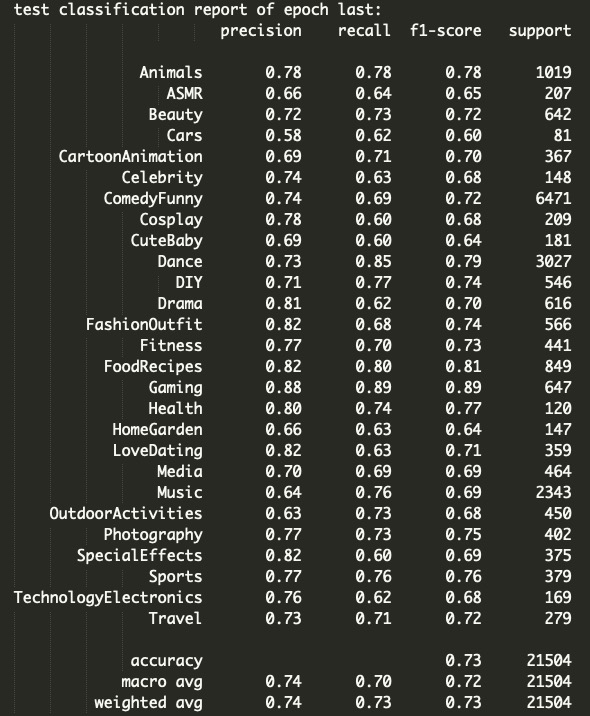
参考 <https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv3D>

1. 而原作者代码中把帧数放到在shape的最后一个维度。

最后效果：

经过shape的维度调整后，效果大幅提升，最终成功上线！！

评估指标如下所示：



loss: 0.1360 - acc\_top1: 0.8496 - acc\_top3: 0.9897 - acc\_top5: 0.9990 - acc: 0.8496 - val\_loss: 2.4356 - val\_acc\_top1: 0.7349 - val\_acc\_top3: 0.8910 - val\_acc\_top5: 0.9250 - val\_acc: 0.7349

1. 吸取的教训：
2. **参考的网上的开源代码一定要仔细阅读并读懂每一行，否则坑随时来也**！
3. **开源代码的作者的数据集和自己的数据集差别很大，而且作者给的benchmark只是记录了一些因素**。

比如这里作者使用的数据集基本上是类别均衡的，不需要向客户的数据集还需要考虑类别不均衡的处理；

比如这里作者的benchmark中只是记录中no color也就是灰度帧测试的结果，并没有给出彩色帧的情况。