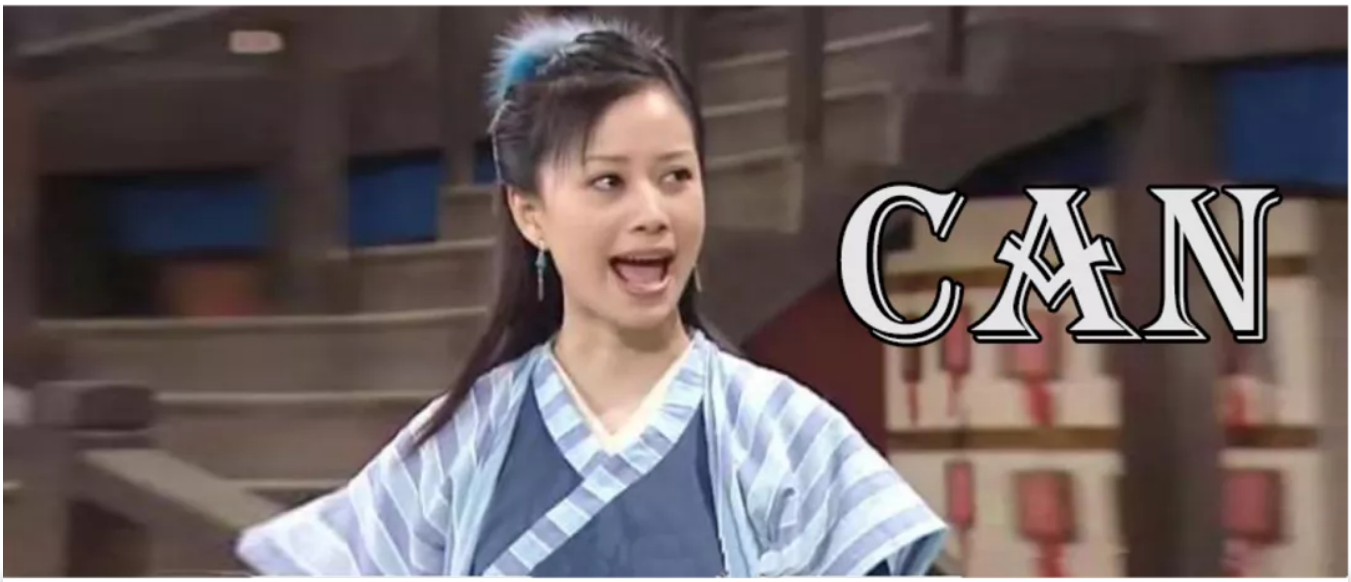
分类问题后处理技巧CAN，近乎零成本获取效果提升

原创 苏剑林 夕小瑶的卖萌屋 2021-10-26 12:05



文 | 苏剑林
编 | 智商掉了一地
单位 | 追一科技

> 思想朴素却不平凡的分类问题后处理技巧，浅显易懂的讲解，拿来吧你！

顾名思义，本文将会介绍一种用于分类问题的后处理技巧——CAN（Classification with Alternating Normalization）。经过笔者的实测，CAN确实多数情况下能提升多分类问题的效果，而且几乎没有增加预测成本，因为它仅仅是对预测结果的简单重新归一化操作。

有趣的是，其实CAN的思想是非常朴素的，朴素到每个人在生活中都应该用过同样的思想。然而，CAN的论文却没有很好地说清楚这个思想，只是纯粹形式化地介绍和实验这个方法。本文的分享中，将会尽量将算法思想介绍清楚。

论文标题：

When in Doubt: Improving Classification Performance with Alternating Normalization

论文链接：

https://arxiv.org/abs/2109.13449

思想例子

假设有一个二分类问题，模型对于输入a给出的预测结果是$p^{(a)}=[0.05,0.95]$，那么我们就可以给出预测类别为1；接下来，对于输入b，模型给出的预测结果是$p^{(b)}=[0.5,0.5]$，这时候处于最不确定的状态，我们也不知道输出哪个类别好。

但是，假如我告诉你：

1. 类别必然是0或1其中之一；
2. 两个类别的出现概率各为0.5。

在这两点先验信息之下，由于前一个样本预测结果为1，那么基于朴素的均匀思想，我们是否更倾向于将后一个样本预测为0，以得到一个满足第二点先验的预测结果？

这样的例子还有很多，比如做10道选择题，前9道你都比较有信心，第10题完全不会只能瞎蒙，然后你一看发现前9题选A、B、C的都有就是没有一个选D的，那么第10题在蒙的时候你会不会更倾向于选D？

这些简单例子的背后，有着跟CAN同样的思想，它其实就是用先验分布来校正低置信度的预测结果，使得新的预测结果的分布更接近先验分布。

2 不确定性

准确来说，CAN是针对低置信度预测结果的后处理手段，所以我们首先要有一个衡量预测结果不确定性的指标。常见的度量是“熵”[1]，对于$p=[p_1,p_2,\cdots,p_m]$，定义为：

$$H(p)=-\sum_{i=1}^{m}p_i\log p_i \qquad (1)$$

然而，虽然熵是一个常见选择，但其实它得出的结果并不总是符合我们的直观理解。比如对于$p^{(a)}=[0.5,0.25,0.25]$和$p^{(b)}=[0.5,0.5,0]$，直接套用公式得到$H(p^{(a)})>H(p^{(b)})$，但就我们的分类场景而言，显然我们会认为$p^{(b)}$比$p^{(a)}$更不确定，所以直接用熵还不够合理。

一个简单的修正是只用前top-k个概率值来算熵，不失一般性，假设$p_1,p_2,\cdots,p_k$是概率最高的k个值，那么

$$H(p)=-\sum_{i=1}^{m}p_i\log p_i \qquad (2)$$

其中$\tilde{p}_i=p_i\Big/\sum_{i=1}^{k}p_i$。为了得到一个0～1范围内的结果，我们取$H_{\text{top-}k}(p)/\log k$为最终的不确定性指标。

算法步骤

现在假设我们有N个样本需要预测类别，模型直接的预测结果是N个概率分布$p^{(1)},p^{(2)},\cdots,p^{(N)}$，假设测试样本和训练样本是同分布的，那么完美的预测结果应该有：

$$\frac{1}{N}\sum_{i=1}^{N}p^{(i)}=\tilde{p}\qquad(3)$$

其中$\tilde{p}$是类别的先验分布，我们可以直接从训练集估计。也就是说，全体预测结果应该跟先验分布是一致的，但受限于模型性能等原因，实际的预测结果可能明显偏离上式，这时候我们就可以人为修正这部分。

具体来说，我们选定一个阈值τ，将指标小于τ的预测结果视为**高置信度**的，而大于等于τ的则是**低置信度**的，不失一般性，我们假设前n个结果$p^{(1)},p^{(2)},\cdots,p^{(n)}$属于高置信度的，而剩下的$N-n$个属于低置信度的。我们认为高置信度部分是更加可靠的，所以它们不用修正，并且可以用它们来作为“标准参考系”来修正低置信度部分。

具体来说，对于$\forall j\in\{n+1,n+2,\cdots,N\}$，我们将$p^{(j)}$与高置信度的$p^{(1)},p^{(2)},\cdots,p^{(n)}$一起，执行一次 **“行间标准化”** ：

$$p^{(k)}\leftarrow p^{(k)}/\bar{p}\times\tilde{p},\quad\bar{p}=\frac{1}{n+1}\left(p^{(j)}+\sum_{i=1}^{n}p^{(i)}\right)\qquad(4)$$

这里的$k\in\{1,2,\cdots,n\}\cup\{j\}$，其中乘除法都是element-wise的。不难发现，这个标准化的目的是使得所有新的$p^{(k)}$的平均向量等于先验分布$\tilde{p}$，也就是促使式(3)的成立。然而，这样标准化之后，每个$p^{(k)}$就未必满足归一化了，所以我们还要执行一次 **行内标准化** ：

$$p^{(k)}\leftarrow\frac{p_i^{(k)}}{\sum_{i=1}^{m}p_i^{(k)}}\qquad(5)$$

理论上，这两步可以交替迭代几次（不过实验结果显示一次的效果就挺好了）。最后，我们只保留最新的$p^{(j)}$作为原来第j个样本的预测结果，其余的$p^{(k)}$均弃之不用。

注意，这个过程需要我们遍历每个低置信度结果$j\in\{n+1,n+2,\cdots,N\}$执行，也就是说是逐个样本进行修正，而不是一次性修正的，每个$p^{(j)}$都借助**原始的**高置信度结果$p^{(1)},p^{(2)},\cdots,p^{(n)}$组合来按照上述步骤迭代，虽然迭代过程中对应的$p^{(1)},p^{(2)},\cdots,p^{(n)}$都会随之更新，但那只是临时结果，最后都是弃之不用的，每次修正都是用原始的$p^{(1)},p^{(2)},\cdots,p^{(n)}$。

参考实现

这是笔者给出的参考实现代码：

```
# 预测结果，计算修正前准确率
y_pred = model.predict(
    valid_generator.fortest(), steps=len(valid_generator), verbose=True
)
y_true = np.array([d[1] for d in valid_data])
acc_original = np.mean([y_pred.argmax(1) == y_true])
print('original acc: %s' % acc_original)

# 评价每个预测结果的不确定性
k = 3
y_pred_topk = np.sort(y_pred, axis=1)[:, -k:]
y_pred_topk /= y_pred_topk.sum(axis=1, keepdims=True)
y_pred_uncertainty = -(y_pred_topk * np.log(y_pred_topk)).sum(1) / np.log(k)

# 选择阈值，划分高、低置信度两部分
threshold = 0.9
y_pred_confident = y_pred[y_pred_uncertainty < threshold]
y_pred_unconfident = y_pred[y_pred_uncertainty >= threshold]
y_true_confident = y_true[y_pred_uncertainty < threshold]
y_true_unconfident = y_true[y_pred_uncertainty >= threshold]

# 显示两部分各自的准确率
# 一般而言，高置信度集准确率会远高于低置信度的
acc_confident = (y_pred_confident.argmax(1) == y_true_confident).mean()
acc_unconfident = (y_pred_unconfident.argmax(1) == y_true_unconfident).mean()
print('confident acc: %s' % acc_confident)
print('unconfident acc: %s' % acc_unconfident)

# 从训练集统计先验分布
prior = np.zeros(num_classes)
for d in train_data:
    prior[d[1]] += 1.
```

```
prior /= prior.sum()

# 逐个修改低置信度样本，并重新评价准确率
right, alpha, iters = 0, 1, 1
for i, y in enumerate(y_pred_unconfident):
    Y = np.concatenate([y_pred_confident, y[None]], axis=0)
    for j in range(iters):
        Y = Y**alpha
        Y /= Y.sum(axis=0, keepdims=True)
        Y *= prior[None]
        Y /= Y.sum(axis=1, keepdims=True)
    y = Y[-1]
    if y.argmax() == y_true_unconfident[i]:
        right += 1

# 输出修正后的准确率
acc_final = (acc_confident * len(y_pred_confident) + right) / len(y_pred)
print('new unconfident acc: %s' % (right / (i + 1.)))
print('final acc: %s' % acc_final)
```

实验结果

那么，这样的简单后处理，究竟能带来多大的提升呢？原论文给出的实验结果是相当可观的：

▲原论文的实验结果之一

笔者也在CLUE上的两个中文文本分类任务上做了实验，显示基本也有点提升，但没那么可观（验证集结果）：

	IFLYTEK(类别数:119)	TNEWS(类别数:15)
BERT	60.06%	56.80%
BERT + CAN	60.52%	56.86%
RoBERTa	60.64%	58.06%
RoBERTa + CAN	60.95%	58.00%

大体上来说，类别数目越多，效果提升越明显，如果类别数目比较少，那么可能提升比较微弱甚至会下降（当然就算下降也是微弱的），所以这算是一个“几乎免费的午餐”了。超参数选择方面，上面给出的中文结果，只迭代了1次，k的选择为3、τ的选择为0.9，经过简单的调试，发现这基本上已经是比较优的参数组合了。

还有的读者可能想问前面说的“高置信度那部分结果更可靠”这个情况是否真的成立？至少在笔者的两个中文实验上它是明显成立的，比如IFLYTEK任务，筛选出来的高置信度集准确率为0.63+，而低置信度集的准确率只有0.22+；TNEWS任务类似，高置信度集准确率为0.58+，而低置信度集的准确率只有0.23+。

个人评价

最后再来综合地思考和评价一下CAN。

首先，一个很自然的疑问是为什么不直接将所有低置信度结果跟高置信度结果拼在一起进行修正，而是要逐个进行修正？笔者不知道原论文作者有没有对比过，但笔者确实实验过这个想法，结果是批量修正有时跟逐个修正持平，但有时也会下降。其实也可以理解，CAN本意应该是借助先验分布，结合高置信度结果来修正低置信度的，在这个过程中，如果掺入越多的低置信度结果，那么最终的偏差可能就越大，因此理论上逐个修正会比批量修正更为可靠。

说到原论文，读过CAN论文的读者，应该能发现本文介绍与CAN原论文大致有三点不同：

1. **不确定性指标的计算方法不同**。按照原论文的描述，它最终的不确定性指标计算方式应该是

$$-\frac{1}{\log m}\sum_{i=1}^{k} p_i \log p_i \qquad (6)$$

也就是说，它也是top-k个概率算熵的形式，但是它没有对这k个概率值重新归一化，并且它将其压缩到0～1之间的因子是$\log m$而不是$\log k$（因为它没有重新归一化，所以只有除$\log m$才能保证0～1之间）。经过笔者测试，原论文的这种方式计算出来的结果通常明显小于1，这不利于我们对阈值的感知和调试。

2. **对CAN的介绍方式不同**。原论文是纯粹数学化、矩阵化地陈述CAN的算法步骤，而且没有介绍算法的思想来源，这对理解CAN是相当不友好的。如果读者没有自行深入思考算法原理，是很难理解为什么这样的后处理手段就能提升分类效果的，而在彻底弄懂之后则会有一种故弄玄虚之感。

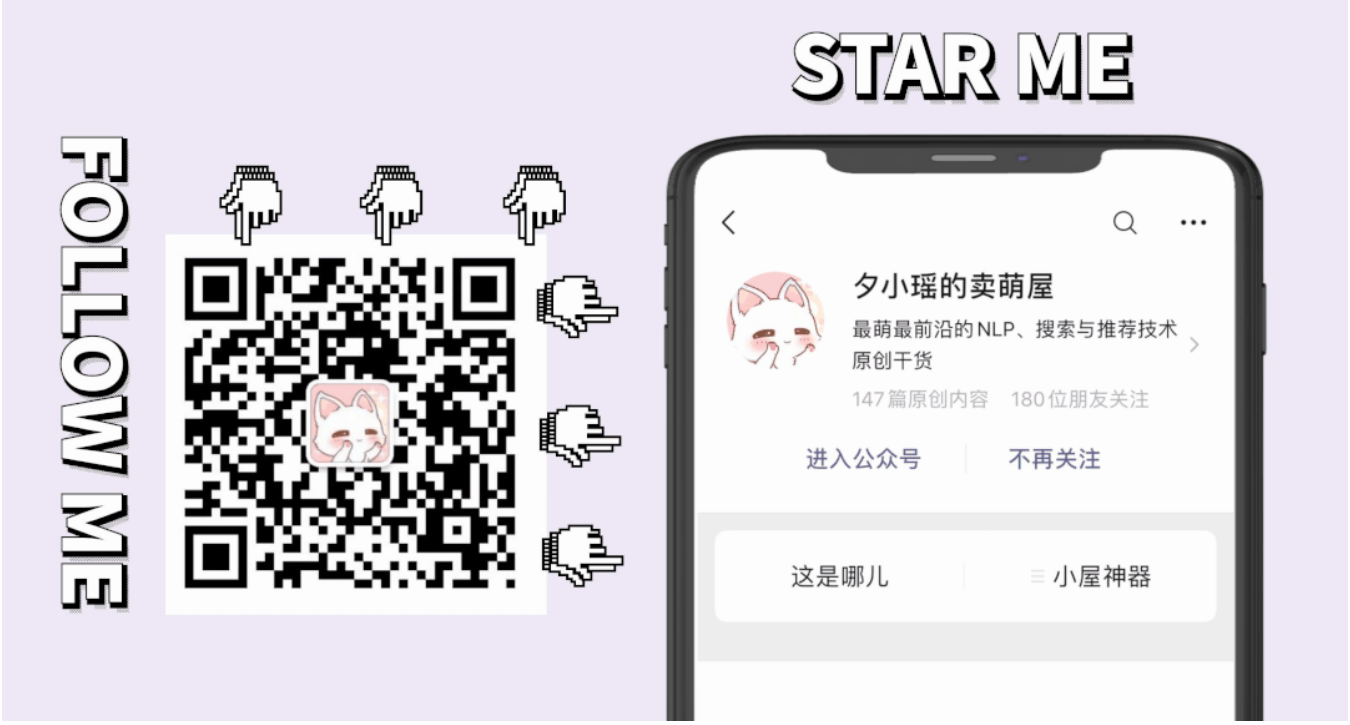
3. **CAN的算法流程略有不同**。原论文在迭代过程中还引入了参数α，使得式(4)变为

$$p^{(k)} \leftarrow [p^{(k)}]^{\alpha} / \bar{p} \times \tilde{p}, \quad \bar{p} = \frac{1}{n+1}\left([p^{(j)}]^{\alpha} + \sum_{i=1}^{n}[p^{(i)}]^{\alpha}\right) \tag{7}$$

也就是对每个结果进行α次方后再迭代。当然，原论文也没有对此进行解释，而在笔者看来，该参数纯粹是为了调参而引入的（参数多了，总能把效果调到有所提升），没有太多实际意义。而且笔者自己在实验中发现，$\alpha = 1$基本已经是最优选择了，精调α也很难获得是实质收益。

文章小结

本文介绍了一种名为CAN的简单后处理技巧，它**借助先验分布来将预测结果重新归一化**，几乎没有增加多少计算成本就能提高分类性能。经过笔者的实验，CAN确实能给分类效果带来一定提升，并且通常来说**类别数越多，效果越明显**。



参考文献

[1] 苏剑林. (Dec. 1, 2015). 《“熵”不起：从熵、最大熵原理到最大熵模型（一）》[Blog post]. Retrieved from https://kexue.fm/archives/3534