**多标签分类中的损失函数与评估指标 - 知乎**

**1 引言**

各位朋友大家好，欢迎来到 **月来客栈** 。在 **前面的一篇文章** [1]中笔者介绍了在单标签分类问题中模型损失的度量方法，即交叉熵损失函数。同时也介绍了 **多分类任务中常见的评价指标** 及其实现方法[2]。在接下来的这篇文章中，笔者将会详细介绍在多标签分类任务中两种常见的损失评估方法，以及在多标签分类场景中的模型评价指标。

**2 方法一**

将原始输出层的softmax操作替换为simoid操作，然后通过计算输出层与标签之间的sigmoid交叉熵来作为误差的衡量标准，具体计算公式如下：

loss(y,y^)=−1C∑i=1m[y(i)⋅log⁡(11+exp⁡(−y^(i)))+(1−y(i))⋅log⁡(exp⁡(−y^(i))1+exp⁡(−y^(i)))](1) loss(y,\hat{y})=-\frac{1}{C} \sum\_{i=1}^m\left[y^{(i)}\cdot\log\left(\frac{1}{1+\exp(-\hat{y}^{(i)})}\right)+\left(1-y^{(i)}\right)\cdot\log\left(\frac{\exp(-\hat{y}^{(i)})}{1+\exp(-\hat{y}^{(i)})}\right)\right]\;\;\;\;\;(1) \\

其中 C C 表示类别数量， y(i) y^{(i)} 和 y^(i) \hat{y}^{(i)} 均为一个向量，分别用来表示真实标签和未经任何激活函数处理的网络输出值。

从式 (1) (1) 可以发现，这种误差损失衡量方式其实就是在逻辑回归中用来衡量预测概率与真实标签之间误差的方法。

**2.1 numpy 实现：**

根据式 (1) (1) 的计算公式，可以通过如下Python代码来完成损失值的计算：

|  |
| --- |
| Plaintext def sigmoid(z):  return 1 / (1 + np.exp(-z))  def compute\_loss\_v1(y\_true, y\_pred):  t\_loss = y\_true \* np.log(sigmoid(y\_pred)) + \  (1 - y\_true) \* np.log(1 - sigmoid(y\_pred)) # [batch\_size,num\_class]  loss = t\_loss.mean(axis=-1) # 得到每个样本的损失值, 这里可以是  return -loss.mean() # 返回整体样本的损失均值（或其他）  if \_\_name\_\_ == '\_\_main\_\_':  y\_true = np.array([[1, 1, 0, 0], [0, 1, 0, 1]])  y\_pred = np.array([[0.2, 0.5, 0, 0], [0.1, 0.5, 0, 0.8]])  print(compute\_loss\_v1(y\_true, y\_pred)) # 0.5926 |

当然，在 TensorFlow 1.x 和 Pytorch 中也分别对这两种方法进行了实现。

**2.2 TensorFlow 实现**

在 Tensorflow 1.x 中，可以通过 tf.nn 模块下的 sigmoid\_cross\_entropy\_with\_logits 方法进行调用：

|  |
| --- |
| Plaintext def sigmoid\_cross\_entropy\_with\_logits(labels, logits):  loss = tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=labels, logits=logits)  loss = tf.reduce\_mean(loss, axis=-1)  return tf.reduce\_mean(loss)  if \_\_name\_\_ == '\_\_main\_\_':  y\_true = tf.constant([[1, 1, 0, 0], [0, 1, 0, 1]],dtype=tf.float16)  y\_pred = tf.constant([[0.2, 0.5, 0, 0], [0.1, 0.5, 0, 0.8]],dtype=tf.float16)  with tf.Session() as sess:  loss = sess.run(sigmoid\_cross\_entropy\_with\_logits(y\_true,y\_pred))  print(loss) # 0.5926 |

当然，在模型训练完成后，可以通过如下代码来得到预测的标签结果和相应的概率值：

|  |
| --- |
| Plaintext def prediction(logits, K):  y\_pred = np.argsort(-logits, axis=-1)[:,:K]  print("预测标签：",y\_pred)  p = np.vstack([logits[r,c] for r,c in enumerate(y\_pred)])  print("预测概率：",p)  prediction(y\_pred,2) ##### 预测标签： [[1 0]  [3 1]] 预测概率：  [[0.5 0.2]  [0.8 0.5]] |

**2.3 Pytorch 实现**

在 Pytorch 中，可以通过 torch.nn 模块中的 MultiLabelSoftMarginLoss 类来完成损失的计算：

|  |
| --- |
| Plaintext if \_\_name\_\_ == '\_\_main\_\_':  y\_true = torch.tensor([[1, 1, 0, 0], [0, 1, 0, 1]],dtype=torch.int16)  y\_pred = torch.tensor([[0.2, 0.5, 0, 0], [0.1, 0.5, 0, 0.8]],dtype=torch.float32)  loss = nn.MultiLabelSoftMarginLoss(reduction='mean')  print(loss(y\_pred, y\_true)) #0.5926 |

同样，在模型训练完成后也可以通过上面的 prediction 函数来完成推理预测。需要注意的是，在 TensorFlow 1.x 中 sigmoid\_cross\_entropy\_with\_logits 方法返回的是所有样本损失的均值；而在 Pytorch 中， MultiLabelSoftMarginLoss 默认返回的是所有样本损失的均值，但是可以通过指定参数 reduction 为 mean 或 sum 来指定返回的类型。

**3 方法二**

在衡量多标签分类结果损失的方法中，除了上面介绍的方法一之外还有一种常用的损失函数。这种损失函数其实就是我们在单标签分类中用到的交叉熵损失函数的拓展版，单标签可以看作是其中的一种特例情况。其具体计算公式如下所示：

loss(y,y^)=−1m∑i=1m∑j=1qyj(i)log⁡y^j(i)(2) loss(y,\hat{y})=-\frac{1}{m}\sum\_{i=1}^m\sum\_{j=1}^qy^{(i)}\_j\log{\hat{y}^{(i)}\_j}\;\;\;\;\;\;\;\;\;\;(2) \\

其中 yj(i) y^{(i)}\_j 表示第 i i 个样本第 j j 个类别的真实值， y^j(i) \hat{y}^{(i)}\_j 表示第 i i 个样本第 j j 个类别的输出经过softmax处理后的结果。

例如对于如下样本来说：

|  |
| --- |
| Plaintext y\_true = np.array([[1, 1, 0, 0], [0, 1, 0, 1.]]) y\_pred = np.array([[0.2, 0.5, 0.1, 0], [0.1, 0.5, 0, 0.8]]) |

输出值经过softmax处理后的结果为：

|  |
| --- |
| Plaintext [[0.24549354 0.33138161 0.22213174 0.20099311]  [0.18482871 0.27573204 0.16723993 0.37219932]] |

那么，根据公式 (2) (2) 可知，对于上述2个样本来说其损失值为：

loss=−12(1⋅log⁡(0.245)+1⋅log⁡(0.331)+1⋅log⁡(0.275)+1⋅log⁡(0.372))≈2.395(3) \begin{aligned} loss &= -\frac{1}{2}\left(1\cdot \log(0.245)+1\cdot \log(0.331)\\+1\cdot \log(0.275) +1\cdot \log(0.372)\right)\approx2.395 \end{aligned}\;\;\;\;\;\;\;\;\;\;(3) \\

**3.1 numpy 实现：**

根据式 (3) (3) 的计算公式，可以通过如下Python代码来完成损失值的计算：

|  |
| --- |
| Plaintext def softmax(x):  s = np.exp(x)  return s / np.sum(s, axis=-1, keepdims=True)  def compute\_loss\_v2(logits, y):  logits = softmax(logits)  print(logits)  c = -(y \* np.log(logits)).sum(axis=-1) # 计算每一个样本的在各个标签上的损失和  return np.mean(c) # 计算所有样本损失的平均值 y\_true = np.array([[1, 1, 0, 0], [0, 1, 0, 1.]]) y\_pred = np.array([[0.2, 0.5, 0.1, 0], [0.1, 0.5, 0, 0.8]]) print(compute\_loss\_v2(y\_pred, y\_true))# 2.392 |

**3.2 TensorFlow 实现**

在 Tensorflow 1.x 中，可以通过 tf.nn 模块下的 softmax\_cross\_entropy\_with\_logits\_v2 方法进行调用：

|  |
| --- |
| Plaintext def softmax\_cross\_entropy\_with\_logits(labels, logits):  loss = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(labels=labels, logits=logits)  return tf.reduce\_mean(loss) y\_true = tf.constant([[1, 1, 0, 0], [0, 1, 0, 1.]], dtype=tf.float16) y\_pred = tf.constant([[0.2, 0.5, 0.1, 0], [0.1, 0.5, 0, 0.8]], dtype=tf.float16) with tf.Session() as sess:  loss = sess.run(softmax\_cross\_entropy\_with\_logits(y\_true, y\_pred))  print(loss)# 2.395 |

**3.3 Pytorch 实现**

在 Pytorch 中，笔者目前还没找到可以调用的相应模型，但是可以通过自己来编码实现：

|  |
| --- |
| Plaintext def cross\_entropy(logits, y):  s = torch.exp(logits)  logits = s / torch.sum(s, dim=1, keepdim=True)  c = -(y \* torch.log(logits)).sum(dim=-1)  return torch.mean(c)  y\_true = torch.tensor([[1, 1, 0, 0], [0, 1, 0, 1.]]) y\_pred = torch.tensor([[0.2, 0.5, 0.1, 0], [0.1, 0.5, 0, 0.8]]) loss = cross\_entropy(y\_pred,y\_true) print(loss)# 2.392 |

需要注意的是，由于各个框架在计算时保留小数的策略不同，所以最后的结果在小数位后面会出现略微的差异。

**4 评估指标**

**4.1 不考虑部分正确的评估方法**

(1) 绝对匹配率（ **Exact Match Ratio** ）

所谓绝对匹配率指的就是，对于每一个样本来说，只有预测值与真实值完全相同的情况下才算预测正确，也就是说只要有一个类别的预测结果有差异都算没有预测正确。因此，其准确率计算公式为：

MR=1m∑i=1mI(y(i)==y^(i))(4) MR=\frac{1}{m}\sum\_{i=1}^mI(y^{(i)}==\hat{y}^{(i)})\;\;\;\;\;\;\;\;\;\;(4) \\

其中 n n 表示样本总数； I(⋅) I(\cdot) 为指示函数（indicator function），当 yi y\_i 完全等同于 y^i \hat{y}\_i 时取 1 1 ，否则为 0 0 。可以看出，MR值越大，表示分类的准确率越高。

例如现有如下真实值和预测值：

|  |
| --- |
| Plaintext  y\_true = np.array([[0, 1, 0, 1],  [0, 1, 1, 0],  [1, 0, 1, 1]])  y\_pred = np.array([[0, 1, 1, 0],  [0, 1, 1, 0],  [0, 1, 0, 1]]) |

那么其对应的MR就应该是 0.333 0.333 ，因为只有第2个样本才算预测正确。在 sklearn 中，可以直接通过 sklearn.metrics 模块中的 accuracy\_score 方法来完成计算[3]，如下所示：

|  |
| --- |
| Plaintext from sklearn.metrics import accuracy\_score print(accuracy\_score(y\_true,y\_pred)) # 0.33333333 |

（2）0-1损失

除了绝对匹配率之外，还有另外一种与之计算过程恰好相反的评估标准，即0-1损失（ **Zero-One Loss** ）。绝对准确率计算的是完全预测正确的样本占总样本数的比例，而0-1损失计算的是完全预测错误的样本占总样本的比例。因此对于上面的预测和真实结果，其0-1损失就应该为0.667。计算公式如下：

L0−1=1m∑i=1mI(y(i)≠y^(i))(5) L\_{0-1}=\frac{1}{m}\sum\_{i=1}^mI(y^{(i)}\neq\hat{y}^{(i)})\;\;\;\;\;\;\;\;\;\;(5) \\

在 sklearn 中，可以通过 sklearn.metrics 模块中的 zero\_one\_loss 方法来完成计算[3]，如下所示：

|  |
| --- |
| Plaintext from sklearn.metrics import zero\_one\_loss print(zero\_one\_loss(y\_true,y\_pred))# 0.66666 |

**4.2 考虑部分正确的评估方法**

从上面的两种评估指标可以看出，不管是绝对匹配率还是0-1损失，两者在计算结果的时候都没有考虑到部分正确的情况，而这对于模型的评估来说显然是不准确的。例如，假设正确标签为 [1,0,0,1] ，模型预测的标签为 [1,0,1,0] 。可以看到，尽管模型没有预测对全部的标签，但是预测对了一部分。因此，一种可取的做法就是将部分预测正确的结果也考虑进去[4]。为了实现这一想法，文献[5]中提出了在多标签分类场景下的准确率（Accuracy）、精确率（Precision）、召回率（Recall）和 F1 F\_1 值（ F1 F\_1 -Measure）计算方法。

（1）准确率

对于准确率来说，其计算公式为：

Accuracy=1m∑i=1m|y(i)∩y^(i)||y(i)∪y^(i)|(6) \text{Accuracy} = \frac{1}{m} \sum\_{i=1}^{m} \frac{\lvert y^{(i)} \cap \hat{y}^{(i)}\rvert}{\lvert y^{(i)} \cup \hat{y}^{(i)}\rvert}\;\;\;\;\;\;\;\;\;\;(6) \\

从公式 (6) (6) 可以看出，准确率其实计算的是所有样本的平均准确率。而对于每个样本来说，准确率就是预测正确的标签数在整个预测为正确或真实为正确标签数中的占比。例如对于某个样本来说，其真实标签为 [0, 1, 0, 1] ，预测标签为 [0, 1, 1, 0] 。那么该样本对应的准确率就应该为：

acc=11+1+1=13(7) acc = \frac{1}{1+1+1}=\frac{1}{3}\;\;\;\;\;\;\;\;\;\;(7) \\

因此，对于如下真实结果和预测结果来说：

|  |
| --- |
| Plaintext  y\_true = np.array([[0, 1, 0, 1],  [0, 1, 1, 0],  [1, 0, 1, 1]])  y\_pred = np.array([[0, 1, 1, 0],  [0, 1, 1, 0],  [0, 1, 0, 1]]) |

其准确率为：

Accuracy=13×(13+22+14)≈0.5278(8) \text{Accuracy}=\frac{1}{3}\times(\frac{1}{3}+\frac{2}{2}+\frac{1}{4})\approx0.5278\;\;\;\;\;\;\;\;\;\;(8) \\

对应的实现代码为[6]：

|  |
| --- |
| Plaintext def Accuracy(y\_true, y\_pred):  count = 0  for i in range(y\_true.shape[0]):  p = sum(np.logical\_and(y\_true[i], y\_pred[i]))  q = sum(np.logical\_or(y\_true[i], y\_pred[i]))  count += p / q  return count / y\_true.shape[0] print(Accuracy(y\_true, y\_pred)) # 0.52777 |

（2）精确率

对于精确率来说，其计算公式为：

Precision=1m∑i=1m|y(i)∩y^(i)||y^(i)|(9) \text{Precision} = \frac{1}{m} \sum\_{i=1}^{m} \frac{\lvert y^{(i)} \cap \hat{y}^{(i)}\rvert}{\lvert \hat{y}^{(i)}\rvert}\;\;\;\;\;\;\;\;\;\;(9) \\

从公式 (9) (9) 可以看出，精确率其实计算的是所有样本的平均精确率。而对于每个样本来说，精确率就是预测正确的标签数在整个预测为正确的标签数中的占比。例如对于某个样本来说，其真实标签为 [0, 1, 0, 1] ，预测标签为 [0, 1, 1, 0] 。那么该样本对应的精确率就应该为：

pre=11+1=12(10) \text{pre} = \frac{1}{1+1}=\frac{1}{2}\;\;\;\;\;\;\;\;\;\;(10) \\

因此，对于上面的真实结果和预测结果来说，其精确率为：

Precision=13×(12+22+12)≈0.6666(11) \text{Precision} = \frac{1}{3}\times(\frac{1}{2}+\frac{2}{2}+\frac{1}{2})\approx0.6666\;\;\;\;\;\;\;\;\;\;(11) \\

对应的实现代码为：

|  |
| --- |
| Plaintext def Precision(y\_true, y\_pred):  count = 0  for i in range(y\_true.shape[0]):  if sum(y\_pred[i]) == 0:  continue  count += sum(np.logical\_and(y\_true[i], y\_pred[i])) / sum(y\_pred[i])  return count / y\_true.shape[0] print(Precision(y\_true, y\_pred))# 0.6666 |

（3）召回率

对于召回率来说，其计算公式为：

Recall=1m∑i=1m|y(i)∩y^(i)||y(i)|(12) \text{Recall} = \frac{1}{m} \sum\_{i=1}^{m} \frac{\lvert y^{(i)} \cap \hat{y}^{(i)}\rvert}{\lvert y^{(i)}\rvert} \;\;\;\;\;\;\;\;\;\;(12) \\

从公式 (12) (12) 可以看出，召回率其实计算的是所有样本的平均精确率。而对于每个样本来说，召回率就是预测正确的标签数在整个正确的标签数中的占比。

因此，对于如下真实结果和预测结果来说：

|  |
| --- |
| Plaintext  y\_true = np.array([[0, 1, 0, 1],  [0, 1, 1, 0],  [1, 0, 1, 1]])  y\_pred = np.array([[0, 1, 1, 0],  [0, 1, 1, 0],  [0, 1, 0, 1]]) |

其召回率为：

Recall=13×(12+22+13)≈0.6111(13) \text{Recall}=\frac{1}{3}\times(\frac{1}{2}+\frac{2}{2}+\frac{1}{3})\approx0.6111\;\;\;\;\;\;\;\;\;\;(13) \\

对应的实现代码为：

|  |
| --- |
| Plaintext def Recall(y\_true, y\_pred):  count = 0  for i in range(y\_true.shape[0]):  if sum(y\_true[i]) == 0:  continue  count += sum(np.logical\_and(y\_true[i], y\_pred[i])) / sum(y\_true[i])  return count / y\_true.shape[0] print(Recall(y\_true, y\_pred))# 0.6111 |

（4） F1 F\_1 值

对于 F1 F\_1 值来说，其计算公式为：

F1=1m∑i=1m2|y(i)∩y^(i)||y(i)|+|y^(i)|(14) F\_{1} = \frac{1}{m} \sum\_{i=1}^{m} \frac{2 \lvert y^{(i)} \cap \hat{y}^{(i)}\rvert}{\lvert y^{(i)}\rvert + \lvert \hat{y}^{(i)}\rvert} \;\;\;\;\;\;\;\;\;\;(14) \\

从公式 (14) (14) 可以看出， F1 F\_1 计算的也是所有样本的平均精确率。因此，对于上面的真实结果和预测结果来说，其 F1 F\_1 值为：

F1=23(14+24+15)≈0.6333(15) F\_1=\frac{2}{3}(\frac{1}{4}+\frac{2}{4}+\frac{1}{5})\approx0.6333\;\;\;\;\;\;\;\;\;\;(15) \\

对应的实现代码为：

|  |
| --- |
| Plaintext def F1Measure(y\_true, y\_pred):  count = 0  for i in range(y\_true.shape[0]):  if (sum(y\_true[i]) == 0) and (sum(y\_pred[i]) == 0):  continue  p = sum(np.logical\_and(y\_true[i], y\_pred[i]))  q = sum(y\_true[i]) + sum(y\_pred[i])  count += (2 \* p) / q  return count / y\_true.shape[0] print(F1Measure(y\_true, y\_pred))# 0.6333 |

在上述4项指标中，都是值越大，对应模型的分类效果越好。同时，从公式 (6)(9)(12)(14) (6)(9)(12)(14) 可以看出，多标签场景下的各项指标尽管在计算步骤上与单标签场景有所区别，但是两者在计算各个指标时所秉承的思想却是类似的。

当然，对于后面3个指标的计算，还可以直接通过 sklearn 来完成，代码如下：

|  |
| --- |
| Plaintext from sklearn.metrics import precision\_score, recall\_score, f1\_score print(precision\_score(y\_true=y\_true, y\_pred=y\_pred, average='samples'))# 0.6666 print(recall\_score(y\_true=y\_true, y\_pred=y\_pred, average='samples'))# 0.6111 print(f1\_score(y\_true,y\_pred,average='samples'))# 0.6333 |

（5）Hamming Loss

除了前面已经介绍的6中评估方法外，下面再介绍另外一种更加直观的衡量方法Hamming Loss[3]，它的计算公式为：

Hamming Loss=1mq∑i=1m∑j=1qI(yj(i)≠y^j(i))(16) \text{Hamming Loss} = \frac{1}{m q} \sum\_{i=1}^{m}\sum\_{j=1}^{q} I\left( y^{(i)}\_{j} \neq \hat{y}^{(i)}\_{j} \right) \;\;\;\;\;\;\;\;\;\;(16) \\

其中 yj(i) y^{(i)}\_j 表示第 i i 个样本的第 j j 个标签， q q 表示一种有多少个类别。

从公式 (16) (16) 可以看出，Hamming Loss衡量的是所有样本中，预测错的标签数在整个标签标签数中的占比。所以对于Hamming Loss损失来说，其值越小表示模型的表现结果越好。因此，对于如下真实结果和预测结果来说：

|  |
| --- |
| Plaintext  y\_true = np.array([[0, 1, 0, 1],  [0, 1, 1, 0],  [1, 0, 1, 1]])  y\_pred = np.array([[0, 1, 1, 0],  [0, 1, 1, 0],  [0, 1, 0, 1]]) |

其Hamming Loss为：

Hamming Loss=13×4(2+0+3)≈0.4166(17) \text{Hamming Loss}=\frac{1}{3\times4}(2+0+3)\approx0.4166\;\;\;\;\;\;\;\;\;\;(17) \\

对应的实现代码为：

|  |
| --- |
| Plaintext def Hamming\_Loss(y\_true, y\_pred):  count = 0  for i in range(y\_true.shape[0]):  p = np.size(y\_true[i] == y\_pred[i])  q = np.count\_nonzero(y\_true[i] == y\_pred[i])  count += p - q  return count / (y\_true.shape[0] \* y\_true.shape[3]) print(Hamming\_Loss(y\_true, y\_pred))# 0.4166 |

同时也可以通过 sklearn.metrics 中的 hamming\_loss 方法来实现：

|  |
| --- |
| Plaintext from sklearn.metrics import hamming\_loss print(hamming\_loss(y\_true, y\_pred))# 0.4166 |

当然，尽管这里介绍了7种不同的评价指标，但是在多标签分类中仍旧还有其它不同的评估方法，具体可以参见文件[4]。例如还可以通过 sklearn.metric 模块中的 multilabel\_confusion\_matrix 方法来分别计算多标签中每个类别的准确率、召回率等；最后再来求每个类别各项指标的平均值。

**5 总结**

在这篇文章中，笔者首先介绍了第一种在多标签分类任务中常见的损失度量方法，其实本质上它就是逻辑回归模型的目标函数；接着笔者介绍了多种用于评估多标签分类任务结果的评价指标，包括绝对匹配率、准确率、召回率等等；最后笔者还是介绍了另外一种常见的多标签分类任务中的损失函数。

**引用**

[1] [想明白多分类，必须得谈逻辑回归](https://zhuanlan.zhihu.com/p/191625074)

[2] [多分类任务下的召回率与F值](https://link.zhihu.com/?target=https%3A//mp.weixin.qq.com/s/pB7RX0Wnr2iLN7X-iAQgHg)

[3] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[4] Sorower, Mohammad S.. “A Literature Survey on Algorithms for Multi-label Learning.” (2010).

[5] Godbole, S., & Sarawagi, S. Discriminative Methods for Multi-labeled Classification. Lecture Notes in Computer Science,(2004), 22–30.

[6] [https:// mmuratarat.github.io/20 20-01-25/multilabel\_classification\_metrics](https://link.zhihu.com/?target=https%3A//mmuratarat.github.io/2020-01-25/multilabel_classification_metrics)

发布于 2021-06-30 19:32