**sklearn中多标签分类场景下的常见的模型评估指标**

在[sklearn](https://so.csdn.net/so/search?q=sklearn&spm=1001.2101.3001.7020)中，提供了多种在多标签分类场景下的模型评估方法，本文将讲述sklearn中常见的多标签分类模型评估指标。在多标签分类中我们可以将模型评估指标分为两大类，分别为不考虑样本部分正确的模型评估方法和考虑样本部分正确的模型评估方法。

首先，我们提供真实数据与预测值结果示例，后续所有示例都基于该数据，

|  |
| --- |
| Python  import numpy as np    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]]) |

**不考虑部分正确的评估方法**

**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：

accuracy ( y , y ^ ) = 1 n samples ∑ i = 0 n samples − 1 I ( y ^ i = y i ) \texttt{accuracy}(y, \hat{y}) = \frac{1}{n\_\text{samples}} \sum\_{i=0}^{n\_\text{samples}-1} I(\hat{y}\_i = y\_i) accuracy(y,y^​)=nsamples​1​i=0∑nsamples​−1​I(y^​i​=yi​)

其中， I ( x ) I(x) I(x) 为指示函数，当 y ^ i \hat{y}\_i y^​i​完全等同于 y i y\_i yi​时，值为1，否则，值为0。

值越大，表示分类的准确率越高。

|  |
| --- |
| Python  from sklearn.metrics import accuracy\_score    print(accuracy\_score(y\_true,y\_pred)) # 0.33333333  print(accuracy\_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))) # 0.5 |

**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。

其公式为：

L 0 − 1 ( y i , y ^ i ) = 1 m ∑ i = 0 m − 1 I ( y ^ i ≠ y i ) L\_{0-1}(y\_i, \hat{y}\_i) = \frac{1}{m} \sum\_{i=0}^{m-1} I(\hat{y}\_i \not= y\_i) L0−1​(yi​,y^​i​)=m1​i=0∑m−1​I(y^​i​​=yi​)

其中， I ( x ) I(x) I(x) 为指示函数。

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

**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。Sklearn提供了在多标签分类场景下的精确率（Precision）、召回率（Recall）和F1值计算方法。

**精确率**

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分类器预测为正确的标签数中的占比。

其公式为：

P ( y s , y ^ s ) = ∣ y s ∩ y ^ s ∣ ∣ y ^ s ∣ P(y\_s, \hat{y}\_s) = \frac{\left| y\_s \cap {\hat{y}\_s} \right|}{\left| {\hat{y}\_s} \right|} P(ys​,y^​s​)=∣y^​s​∣∣ys​∩y^​s​∣​

P r e c i s i o n = 1 ∣ S ∣ ∑ s ∈ S P ( y s , y ^ s ) Precision = \frac{1}{\left|S\right|} \sum\_{s \in S} P(y\_s, \hat{y}\_s) Precision=∣S∣1​s∈S∑​P(ys​,y^​s​)

其中， y s y\_s ys​为真实值为正确的标签数据， y ^ s \hat{y}\_s y^​s​为分类器预测为正确的值。

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：

p r e c i s i o n = 1 1 + 1 = 1 2 precision = \frac{1}{1+1}=\frac{1}{2} precision=1+11​=21​

因此，对于上面的真实数据和预测结果来说，其精确率为：

P r e c i s i o n = 1 3 ∗ ( 1 2 + 2 2 + 1 2 ) ≈ 0.666 Precision = \frac{1}{3}\*(\frac{1}{2}+\frac{2}{2}+\frac{1}{2}) \approx 0.666 Precision=31​∗(21​+22​+21​)≈0.666

对应的代码实现如下：

|  |
| --- |
| Python  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 |

sklearn中的实现方法如下

|  |
| --- |
| Python  from sklearn.metrics import precision\_score  print(precision\_score(y\_true=y\_true, y\_pred=y\_pred, average='samples')) # 0.6666 |

**召回率**

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。

其公式为：

R ( y s , y ^ s ) = ∣ y s ∩ y ^ s ∣ ∣ y s ∣ R(y\_s, \hat{y}\_s) = \frac{\left| y\_s \cap \hat{y}\_s \right|}{\left| y\_s \right|} R(ys​,y^​s​)=∣ys​∣∣ys​∩y^​s​∣​

R e c a l l = 1 ∣ S ∣ ∑ s ∈ S R ( y s , y ^ s ) Recall = \frac{1}{\left|S\right|} \sum\_{s \in S} R(y\_s, \hat{y}\_s) Recall=∣S∣1​s∈S∑​R(ys​,y^​s​)

其中， y s y\_s ys​为真实值为正确的标签数据， y ^ s \hat{y}\_s y^​s​为分类器预测为正确的值。

因此，对于上面的真实数据和预测结果来说，其召回率为：

R e c a l l = 1 3 ∗ ( 1 2 + 2 2 + 1 3 ) ≈ 0.611 Recall = \frac{1}{3}\*(\frac{1}{2}+\frac{2}{2}+\frac{1}{3}) \approx 0.611 Recall=31​∗(21​+22​+31​)≈0.611

对应的代码实现如下：

|  |
| --- |
| Python  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 |

sklearn中的实现方法如下：

|  |
| --- |
| Python  from sklearn.metrics import recall\_score  print(recall\_score(y\_true=y\_true, y\_pred=y\_pred, average='samples'))# 0.6111 |

**F1值**

F 1 F\_1 F1​计算的也是所有样本的平均 F 1 F\_1 F1​值。

其公式为：

F β ( y s , y ^ s ) = ( 1 + β 2 ) P ( y s , y ^ s ) × R ( y s , y ^ s ) β 2 P ( y s , y ^ s ) + R ( y s , y ^ s ) F\_\beta(y\_s, \hat{y}\_s) = \left(1 + \beta^2\right) \frac{P(y\_s, \hat{y}\_s) \times R(y\_s, \hat{y}\_s)}{\beta^2 P(y\_s, \hat{y}\_s) + R(y\_s, \hat{y}\_s)} Fβ​(ys​,y^​s​)=(1+β2)β2P(ys​,y^​s​)+R(ys​,y^​s​)P(ys​,y^​s​)×R(ys​,y^​s​)​

F β = 1 ∣ S ∣ ∑ s ∈ S F β ( y s , y ^ s ) F\_\beta = \frac{1}{\left|S\right|} \sum\_{s \in S} F\_\beta(y\_s, \hat{y}\_s) Fβ​=∣S∣1​s∈S∑​Fβ​(ys​,y^​s​)

当 β = 1 \beta=1 β=1时，即为 F 1 F\_1 F1​值。其公式为：

F β ( y s , y ^ s ) = 1 ∣ S ∣ ∑ s ∈ S 2 ∗ P ( y s , y ^ s ) × R ( y s , y ^ s ) P ( y s , y ^ s ) + R ( y s , y ^ s ) = 1 ∣ S ∣ ∑ s ∈ S 2 ∗ ∣ y s ∩ y ^ s ∣ ∣ y ^ s ∣ + ∣ y s ∣ F\_\beta(y\_s, \hat{y}\_s) = \frac{1}{\left|S\right|} \sum\_{s \in S} \frac{2 \* P(y\_s, \hat{y}\_s) \times R(y\_s, \hat{y}\_s)}{ P(y\_s, \hat{y}\_s) + R(y\_s, \hat{y}\_s)}=\frac{1}{\left|S\right|} \sum\_{s \in S} \frac{2\* \left| y\_s \cap \hat{y}\_s \right|}{\left| \hat{y}\_s \right| + \left| y\_s \right|} Fβ​(ys​,y^​s​)=∣S∣1​s∈S∑​P(ys​,y^​s​)+R(ys​,y^​s​)2∗P(ys​,y^​s​)×R(ys​,y^​s​)​=∣S∣1​s∈S∑​∣y^​s​∣+∣ys​∣2∗∣ys​∩y^​s​∣​

因此，对于上面的真实结果和预测结果来说，其 F 1 F\_1 F1​值为

F 1 = 2 3 ∗ ( 1 4 + 1 2 + 1 5 ) ≈ 0.633 F\_1 = \frac{2}{3} \* (\frac{1}{4}+\frac{1}{2}+\frac{1}{5}) \approx 0.633 F1​=32​∗(41​+21​+51​)≈0.633

对应的代码实现如下：

|  |
| --- |
| Python  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 |

sklearn中的实现方法如下：

|  |
| --- |
| Python  from sklearn.metrics import f1\_score  print(f1\_score(y\_true,y\_pred,average='samples')) # 0.6333 |

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，从上面的公式可以看出，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。

**Hamming Score**

Hamming Score为针对多标签分类场景下另一种求取准确率的方法。Hamming Score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和真实为正确标签数中的占比。

其公式为：

Accuracy = 1 m ∑ i = 1 m ∣ y i ∩ y ^ i ∣ ∣ y i ∪ y ^ i ∣ \text{Accuracy} = \frac{1}{m} \sum\_{i=1}^{m} \frac{\left| y\_i \cap \hat{y}\_i \right|}{\left| y\_i \cup \hat{y}\_i \right|} Accuracy=m1​i=1∑m​∣yi​∪y^​i​∣∣yi​∩y^​i​∣​

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：

accuracy = 1 1 + 1 + 1 = 1 3 \text{accuracy} = \frac{1}{1+1+1} = \frac{1}{3} accuracy=1+1+11​=31​

因此，对于上面的真实数据和预测结果来说，其Hamming Score为：

Accuracy = 1 3 ∗ ( 1 3 + 2 2 + 1 4 ) ≈ 0.5278 \text{Accuracy} = \frac{1}{3}\*(\frac{1}{3}+\frac{2}{2}+\frac{1}{4}) \approx 0.5278 Accuracy=31​∗(31​+22​+41​)≈0.5278

对应的代码实现如下：

|  |
| --- |
| Python  import numpy as np    def hamming\_score(y\_true, y\_pred, normalize=True, sample\_weight=None):  '''  Compute the Hamming score (a.k.a. label-based accuracy) for the multi-label case  http://stackoverflow.com/q/32239577/395857  '''  acc\_list = []  for i in range(y\_true.shape[0]):  set\_true = set(np.where(y\_true[i])[0] )  set\_pred = set(np.where(y\_pred[i])[0] )  tmp\_a = None  if len(set\_true) == 0 and len(set\_pred) == 0:  tmp\_a = 1  else:  tmp\_a = len(set\_true.intersection(set\_pred))/float(len(set\_true.union(set\_pred)) )  acc\_list.append(tmp\_a)  return np.mean(acc\_list)      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]])      print('Hamming score: {0}'.format(hamming\_score(y\_true, y\_pred))) # 0.5277 |

**海明距离（Hamming Loss）**

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。取值在0~1之间。距离为0说明预测结果与真实结果完全相同，距离为1就说明模型与我们想要的结果完全就是背道而驰。

其公式为：

L H a m m i n g ( y , y ^ ) = 1 m ∗ n labels ∑ i = 0 m − 1 ∑ j = 0 n labels − 1 I ( y ^ j ( i ) ≠ y j ( i ) ) L\_{Hamming}(y, \hat{y}) = \frac{1}{m\*n\_\text{labels}} \sum\_{i=0}^{m - 1}\sum\_{j=0}^{n\_\text{labels} - 1} I(\hat{y}\_j^{(i)} \not= y\_j^{(i)}) LHamming​(y,y^​)=m∗nlabels​1​i=0∑m−1​j=0∑nlabels​−1​I(y^​j(i)​​=yj(i)​)

其中，m表示样本数， n lable n\_\text{lable} nlable​表示标签数。

因此，对于上面的真实结果和预测结果来说，其Hamming Loss值为

Hamming Loss = 1 3 ∗ 4 ∗ ( 2 + 0 + 3 ) ≈ 0.4166 \text{Hamming Loss} = \frac{1}{3\*4}\*(2+0+3) \approx 0.4166 Hamming Loss=3∗41​∗(2+0+3)≈0.4166

对应的代码实现如下：

|  |
| --- |
| Python  def Hamming\_Loss(y\_true, y\_pred):  count = 0  for i in range(y\_true.shape[0]):  # 单个样本的标签数  p = np.size(y\_true[i] == y\_pred[i])  # np.count\_nonzero用于统计数组中非零元素的个数  # 单个样本中预测正确的样本数  q = np.count\_nonzero(y\_true[i] == y\_pred[i])  print(f"{p}-->{q}")  count += p - q  print(f"样本数：{y\_true.shape[0]}, 标签数：{y\_true.shape[1]}") # 样本数：3, 标签数：4  return count / (y\_true.shape[0] \* y\_true.shape[1])  print(Hamming\_Loss(y\_true, y\_pred)) # 0.4166 |

sklearn中的实现方法如下：

|  |
| --- |
| Python  from sklearn.metrics import hamming\_loss    print(hamming\_loss(y\_true, y\_pred))# 0.4166  print(hamming\_loss(np.array([[0, 1], [1, 1]]), np.zeros((2, 2)))) # 0.75 |

**总结**

除了上面提供的多标签模型评估方法之外，sklean中还提供了其他的模型评估方法，如

混淆矩阵（multilabel\_confusion\_matrix）、杰卡德相似系数（jaccrd\_similarity\_score）等，这里就不一样介绍了。

**参考文档**

* [sklearn model\_evaluation](https://scikit-learn.org/stable/modules/model_evaluation.html)
* [多标签分类中的损失函数与评价指标](https://zhuanlan.zhihu.com/p/385475273)
* [Getting the accuracy for multi-label prediction in scikit-learn](https://stackoverflow.com/questions/32239577/getting-the-accuracy-for-multi-label-prediction-in-scikit-learn)
* [multi-label classification with sklearn](https://www.kaggle.com/roccoli/multi-label-classification-with-sklearn)
* [Metrics for Multilabel Classification](https://mmuratarat.github.io/2020-01-25/multilabel_classification_metrics)