**pytorch：多标签分类的损失函数和准确率计算\_明日何其多\_的博客-CSDN博客\_多标签分类损失函数**

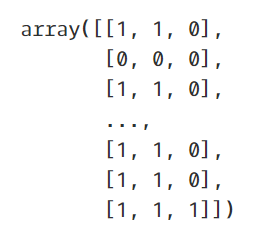
**1** [损失函数](https://so.csdn.net/so/search?q=损失函数&spm=1001.2101.3001.7020)

我们先用[sklearn](https://so.csdn.net/so/search?q=sklearn&spm=1001.2101.3001.7020)生成一个多标签分类数据集。

|  |
| --- |
| Python  from sklearn.datasets import make\_multilabel\_classification    X, y = make\_multilabel\_classification(n\_samples=1000,  n\_features=10,  n\_classes=3,  n\_labels=2,  random\_state=1)  print(X.shape, y.shape) |

Generated

看一下标签长啥样。



每一行都是0、1标签，1可能有多个，这就是多标签了。

由于仍然是二分类（标签只有0和1），所以激活函数用Sigmoid（对输出的每一个维度用Sigmoid）。这个时候损失函数就是BCELoss。

如果是普通的二分类，Sigmoid的输出是一个值。用 N N N表示样本数量， p n p\_n pn​表示预测第 n n n个样本为正例的概率， y n y\_n yn​表示第 n n n个样本的标签，则BCELoss计算公式为：

l o s s = − 1 N ∑ n = 1 N y n × l o g ( p n ) + ( 1 − y n ) × l o g ( 1 − p n ) loss=-\frac{1}{N}\sum\_{n=1}^{N}y\_n×log(p\_n)+(1-y\_n)×log(1-p\_n) loss=−N1​n=1∑N​yn​×log(pn​)+(1−yn​)×log(1−pn​)

那么对于多标签分类呢？BCELoss会计算每一个维度上的损失然后求平均。

举个例子，假如模型某个输出是[0.2，0.6，0.8]，真实值是[0，0，1]，那么该样本损失可以计算如下：

a = 0 × l n ( 0.2 ) + 1 × l n ( 1 − 0.2 ) b = 0 × l n ( 0.6 ) + 1 × l n ( 1 − 0.6 ) c = 1 × l n ( 0.8 ) + 0 × l n ( 1 − 08 ) l o s s = ( a + b + c ) / 3 a=0×ln(0.2)+1×ln(1-0.2)\\ b=0×ln(0.6)+1×ln(1-0.6)\\ c=1×ln(0.8)+0×ln(1-08)\\ loss=(a+b+c)/3 a=0×ln(0.2)+1×ln(1−0.2)b=0×ln(0.6)+1×ln(1−0.6)c=1×ln(0.8)+0×ln(1−08)loss=(a+b+c)/3

这只是单个样本的损失，最后还需要求所有样本损失的平均值。但是你就不用管了，只需要知道多标签分类用Sigmoid+BCELoss就可以完成损失计算。还有一个函数叫BCEWithLogitsLoss，是Sigmoid和BCELoss的结合。如果损失函数用这个，Sigmoid就可以不用。

**2 准确率计算**

依然是上面的例子，模型的输出是[0.2，0.6，0.8]，真实值是[0，0，1]。准确率该怎么计算呢？

|  |
| --- |
| Python  pred = torch.tensor([0.2, 0.6, 0.8])  y = torch.tensor([0, 0, 1])  accuracy = (pred.ge(0.5) == y).all().int().item()  accuracy  # output : 0 |

首先ge函数将pred中大于等于0.5的转化为True，小于0.5的转化成False，再比较pred和y（必须所有维度都相同才算分类准确），最后将逻辑值转化为整数输出即可。

训练时都是按照一个batch计算的，那就写一个循环吧。

|  |
| --- |
| Python  pred = torch.tensor([[0.2, 0.5, 0.8], [0.4, 0.7, 0.1]])  y = torch.tensor([[0, 0, 1], [0, 1, 0]])  accuracy = sum(row.all().int().item() for row in (pred.ge(0.5) == y))  accuracy  # output : 1 |

**3 完整代码**

|  |
| --- |
| Python  from sklearn.datasets import make\_multilabel\_classification  import torch  from torch.utils.data import DataLoader  from sklearn.model\_selection import train\_test\_split    def get\_dataset():  X, y = make\_multilabel\_classification(n\_samples=1000,  n\_features=10,  n\_classes=3,  n\_labels=2,  random\_state=1)  return X,y    n\_inputs, n\_outputs = X.shape[1], y.shape[1]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.33,random\_state=42)  X\_train = torch.from\_numpy(X\_train).float()  X\_test = torch.from\_numpy(X\_test).float()  y\_train = torch.from\_numpy(y\_train).float()  y\_test = torch.from\_numpy(y\_test).float()    train\_data=[(X,y) for X,y in zip(X\_train,y\_train)]  train\_loader = DataLoader(train\_data, batch\_size=64,shuffle=True)    class MLP(nn.Module):  def \_\_init\_\_(self, n\_inputs, n\_outputs, num\_hiddens):  super(MLP, self).\_\_init\_\_()  self.linear\_relu\_stack = nn.Sequential(  nn.Linear(n\_inputs, num\_hiddens),  nn.ReLU(),  nn.Linear(num\_hiddens, n\_outputs),  nn.Sigmoid())    def forward(self, x):  outputs = self.linear\_relu\_stack(x)  return outputs      num\_hiddens = 30  model = MLP(n\_inputs, n\_outputs, num\_hiddens)  print(model)    loss = nn.BCELoss()  optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)    def evaluate\_accuracy(X, y, model):  pred = model(X)  correct = sum(row.all().int().item() for row in (pred.ge(0.5) == y))  n = y.shape[0]  return correct / n    def train(train\_loader, X\_test, y\_test, model, loss, num\_epochs, batch\_size,  optimizer):  batch\_count = 0  for epoch in range(num\_epochs):  train\_l\_sum, train\_acc\_sum, n = 0.0, 0.0, 0  for X, y in train\_loader:  pred = model(X)  l = loss(pred, y)  optimizer.zero\_grad()  l.backward()  optimizer.step()  train\_l\_sum += l.item()  train\_acc\_sum += sum(row.all().int().item()  for row in (pred.ge(0.5) == y))  n += y.shape[0]  batch\_count += 1  test\_acc = evaluate\_accuracy(X\_test, y\_test, model)  print(  'epoch %d, loss %.4f, train acc %.3f, test acc %.3f'  % (epoch + 1, train\_l\_sum / batch\_count, train\_acc\_sum / n,  test\_acc))    num\_epochs, batch\_size = 20, 64  train(train\_loader, X\_test, y\_test, model, loss, num\_epochs, batch\_size,optimizer) |

