机器学习评估方法与评估指标

# 离线评估方法

离线评估的基本原理是在离线环境中，将数据集划分为“训练集”和“测试集”，用“训练集”训练模型，用“测试集”评估模型，根据数据集划分方法的不同，离线评估可以分为以下3种。

## Holdout检验

Holdout检验是基础的离线评估方法，它将原始的样本集合随机的划分为训练集和验证集两部分。通常选择7:3作为划分比例，70%的样本用于模型的训练，30%的样本用于模型的评估。

Holdout的缺点是：在验证集上计算出来的评估指标与训练集和验证集的划分有直接的关系，如果仅进行少量的Holdout检验，则得到的结论存在较大的随机性。为了消除这种随机性，“交叉检验”的思想被提出。

## 交叉检验

k-fold交叉验证：先将全部样本划分成k个大小相等的样本子集；依次遍历这k个子集，每次都先把当前子集作为验证集，其余所有子集作为训练集，进行模型的训练和评估；最后把所有k次的评估指标的平均值作为最终的评估指标。在实际经验中，k通常取10。

留一验证：每次留下一个样本作为验证集，其余所有样本作为测试集。样本总数为n，依次遍历所有n个样本，进行n次验证，再将评估指标求平均得到最终指标。在样本总数较多的情况下，留一验证发的时间开销极大。事实上，留一验证是留p验证的特例。留p验证是指每次留下p个样本作为验证集，而从n个元素中选择p个元素有种可能，因此它的时间开销远远高于留一验证，故很少在实际工程中应用。

## 自助法

不管是holdout检验还是交叉检验，都是基于划分训练集和测试集的方法进行模型评估的，然后当样本规模比较小时，将样本集进行划分会让训练集进一步减小，这可能会影响模型的训练效果。自助法可以在一定程度上解决这个问题。

自助法（Bootstrap）是基于自采样的检验方法：对于总数为n的样本集合，进行n次有放回的随机采样，得到大小为n的训练集。在n次采样过程中，有的样本会被重复采样，有的样本没有被采样过，这些没有被采样的样本作为验证集进行模型验证，这就是自助法的验证过程。

# 离线评估指标

## 准确率

准确率（Accuracy）是指分类正确样本占总样本个数的比例，即：

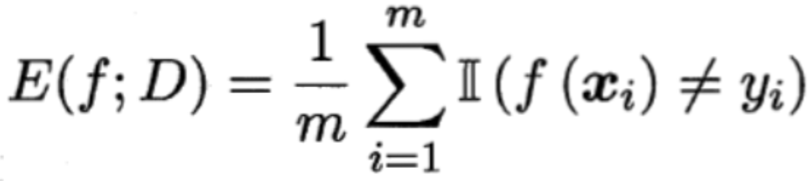
其中，为被正确分类的样本个数，为总样本个数，TP表示预测为正并且实际也为正的样本个数，TN表示预测为负并且实际也为负的样本个数，FP表示预测为正但实际为负的样本个数，FN表示预测为负但实际为正的样本个数。

准确率是分类任务中较直观的评价指标，虽然其具有较强的可解释性，但也存在缺陷：

1. 当不同类别的样本比例非常不均衡时，占比大的类别往往成为影响准确率的最主要因素。例如：如果负样本占99%，那么分类器把所有样本都预测为负样本可以获得99%的准确率。
2. 结果太笼统，实际应用中，我们可能更加关注某一类别样本的情况。比如搜索时会关心 “检索出的信息有多少是用户感兴趣的”，“用户感兴趣的信息有多少被检测出来了” 等等。

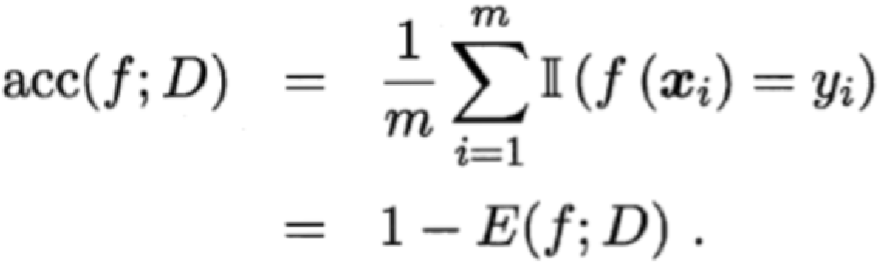
相应地还有错误率：分类错误的样本占总样本的比例。其公式化描述如下：

更一般的，分类错误率定义为：

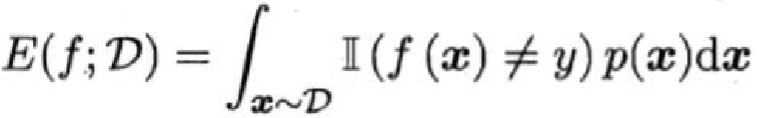


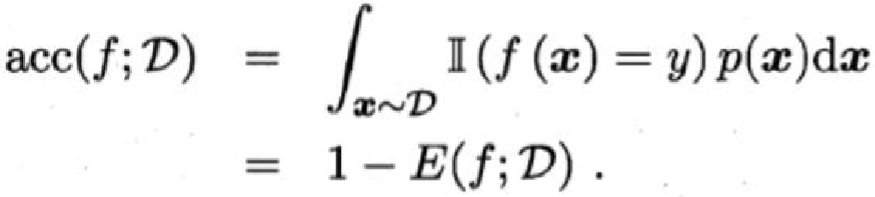
其中I() 是指示函数，当条件为真时，指示函数的函数值为1，否则为0。

基于此，准确率可以表示为：



而更一般的，对于数据分布D和概率密度函数p(⋅)，错误率与精度可分别描述为：





采用sklearn计算准确率demo如下：

|  |
| --- |
| from sklearn.metrics import accuracy\_score  y\_pred = [0, 0, 1, 1]  y\_true = [1, 0, 1, 0]  accuracy\_score(y\_true, y\_pred) # 0.5 |

## 精确率与召回率

精确率和召回率与混淆矩阵密切相关，混淆矩阵是将分类（二分类）结果通过矩阵的形式直观展现出来：

|  |  |  |
| --- | --- | --- |
| 真实情况 | 预测结果为正例 | 预测结果为反例 |
| 正例 | TP(真正例) | FN(假反例) |
| 反例 | FP(假正例) | TN(真反例) |

精确率（Precision），也称为查准率，反映的是模型判定的精确度，预测为正样本里面有多少是真实的正样本。可公式化描述如下：

召回率（Recall），也叫查全率，定义为分类正确的正样本个数占真正的正样本个数的比例，意味着真实的正样本有多少被找出来了。可公式化描述如下：

精确率和召回率是矛盾统一的两个指标：为了提高精确率，分类器需要尽量在“更有把握时”，才把样本预测为正样本，但往往会因为过于保守而漏掉很多“没有把握”的正样本，导致召回率很低。比如我们想要一个模型精确率达到 100%，那就意味着要保证每一个结果都是真正例，这就会导致有些正例被放弃；相反，要保证模型能将所有正例都预测为正例，意味着有些反例也会混进来。这背后的根本原因就在于我们的数据往往是随机、且充满噪声的，并不是非黑即白。

采用sklearn得到精确率和召回率的代码如下：

|  |
| --- |
| from sklearn.metrics import precision\_score  from sklearn.metrics import recall\_score  Precision = precision\_score(y, guesses)  Recall = recall\_score(y, guesses) |

采用sklearn得到混淆矩阵的代码如下：

|  |
| --- |
| from sklearn.metrics import confusion\_matrix  confusion\_matrix(y, guesses) |

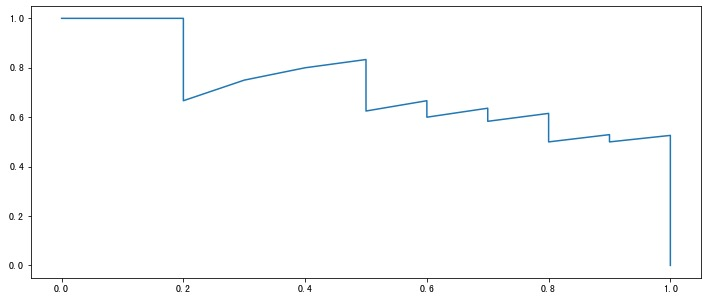
## P-R曲线

Precision-Recall曲线（精确率召回率曲线，简称P-R曲线）。P-R曲线的横轴是召回率，纵轴是精确率。P-R曲线上的一个点代表“在某一阈值下，模型将大于该阈值的结果判定为正样本，将下于该阈值的结果判定为负样本时，模型预测结果对应的召回率和精确率”。

P-R曲线绘制方法如下：

* 对模型的学习结果进行排序（一般都有一个概率值）
* 按照上面的顺序逐个把样本作为正例进行预测，每次都可以得到一个 P R 值
* 将得到的 P R 值按照 R为横坐标，P为纵坐标绘制曲线图。

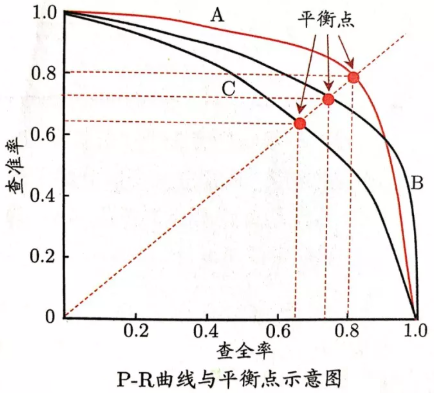
|  |
| --- |
| from typing import List, Tuple  import matplotlib.pyplot as plt  def get\_confusion\_matrix(  y\_pred: List[int],  y\_true: List[int]  ) -> Tuple[int, int, int, int]:    length = len(y\_pred)  assert length == len(y\_true)  tp, fp, fn, tn = 0, 0, 0, 0  for i in range(length):  if y\_pred[i] == y\_true[i] and y\_pred[i] == 1:  tp += 1  elif y\_pred[i] == y\_true[i] and y\_pred[i] == 0:  tn += 1  elif y\_pred[i] == 1 and y\_true[i] == 0:  fp += 1  elif y\_pred[i] == 0 and y\_true[i] == 1:  fn += 1  return (tp, fp, tn, fn)  def calc\_p(tp: int, fp: int) -> float:  return tp / (tp + fp)  def calc\_r(tp: int, fn: int) -> float:  return tp / (tp + fn)  def get\_pr\_pairs(  y\_pred\_prob: List[float],  y\_true: List[int]  ) -> Tuple[List[int], List[int]]:  ps = [1]  rs = [0]  for prob1 in y\_pred\_prob:  y\_pred\_i = []  for prob2 in y\_pred\_prob:  if prob2 < prob1:  y\_pred\_i.append(0)  else:  y\_pred\_i.append(1)  tp, fp, tn, fn = get\_confusion\_matrix(y\_pred\_i, y\_true)  p = calc\_p(tp, fp)  r = calc\_r(tp, fn)  ps.append(p)  rs.append(r)  ps.append(0)  rs.append(1)  return ps, rs  y\_pred\_prob = [0.9, 0.8, 0.7, 0.6, 0.55, 0.54, 0.53, 0.52, 0.51, 0.505,  0.4, 0.39, 0.38, 0.37, 0.36, 0.35, 0.34, 0.33, 0.3, 0.1]  y\_true = [1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0]  y\_pred = [1] \* 10 + [0] \* 10  ps, rs = get\_pr\_pairs(y\_pred\_prob, y\_true)  fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(12, 5))  ax.plot(rs, ps); |



如果有多个模型就可以绘制多条 P-R 曲线：

* 如果某个模型的曲线完全被另外一个模型 “包住”（即后者更加凹向原点），那么后者的性能一定优于前者。
* 如果多个模型的曲线发生交叉，此时不好判断哪个模型较优，一个较为合理的方法是计算曲线下面积，但这个值不太好估算。

为了获得模型优劣，需要综合P（精确率）和R（召回率），平衡点BEP（Break-Even Point）是其中一种判断方法，它是 P=R 时的取值，BEP越远离原点，说明模型效果越好。如下图中，A的BEP是0.8，B的BEP是0.73，因此可以认为模型A比模型B效果更好。



由于BEP过于简单，为了综合反映Precision和Recall的结果，可以使用F1-score（F1值）。

## F1-score

F1-score是精确率和召回率的调和平均值，其定义如下：

F1-score有更一般的形式：

* 当β > 1时，更偏向召回率
* 当β < 1时，更偏好精确率
* 当β = 1时，平衡精确率和召回率，即为 F1

F1 其实来自精确率和召回率的加权调和平均：

## 多个二分类混淆矩阵的综合考查

当有多个混淆矩阵（多次训练、多个数据集、多分类任务）时，有两种方式估算 “全局” 性能：

1. macro 方法：先计算每个 PR，取平均后，再计算 F1。这样得到的就是宏查准率（macro-P）、宏查全率（macro-R），以及相应的宏 F1（macro-F1）
2. micro 方法：先计算混淆矩阵元素的平均，再计算 PR 和 F1。这样得到的就是微查准率（micro-P），微查全率 （micro-R），微F1（micro-F1）

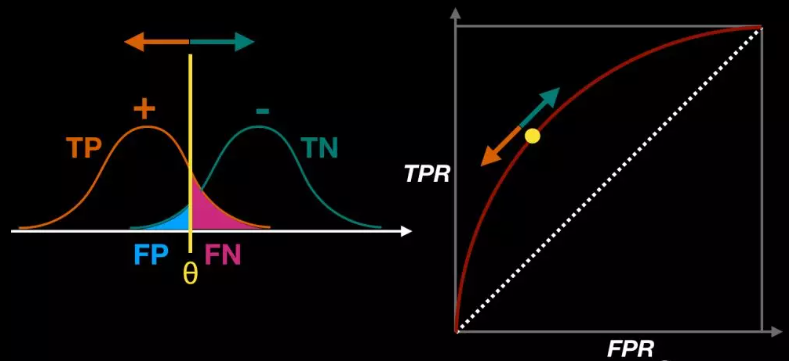
采用sklearn得到f1-score的代码如下：

|  |
| --- |
| from sklearn.metrics import f1\_score  f1\_score(y\_true=y\_test, y\_pred=y\_pred) |

## ROC曲线与AUC

ROC曲线的全称是the Receiver Operating Characteristic曲线，中文译为“受试者工作特征曲线”。ROC曲线的横坐标为False Positive Rate（FPR，假阳性率），定义为负例样本中被错误判定为正例的样本比例；纵坐标为True Positive Rate（TPR，真阳性率，也称为敏感度(Sensitivity)），定义为正例样本中被正确判定为正例的样本数比例。通常来说，ROC曲线越光滑，过拟合程度越小。FPR和TPR的计算方法如下：

在上式中，P是真实正样本的数量，N是真实负样本的数量；TP是P个正样本中被分类器预测为正样本的个数，FP是N个负样本中被分类器预测为正样本的个数。



对于一个特定的分类器和样本集合，只能计算得到一对 FPR 和 TPR，也就是曲线上的一个点，而要得到一个曲线，我们需要一系列 FPR 和 TPR 的值，这又是如何得到的呢？

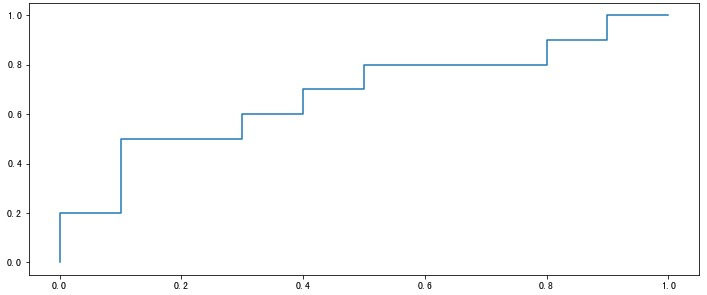
通常分类器在预测某个样本的类型时，会给出这个样本具有多大的概率属于正类，然后根据设定的某个阈值，预测其为正类还是负类。根据某个阈值我们可以计算出相应的一对 FPR 和 TPR，通过改变阈值的大小，就可以计算出一系列的FPR和TPR了。随着阈值的逐渐减小，越来越多的样本被划分为正类，但是这些正类中同样也掺杂着真正的负类，即 TPR 和 FPR 会同时增大。当阈值取最大值1时，对应坐标点为(0,0)，当阈值取最小值0时，对应坐标点为 (1,1)。

采用sklearn得到ROC的代码如下：

|  |
| --- |
| from sklearn.metrics import roc\_curve, roc\_auc\_curve  roc\_curve(y, guesses)  roc\_auc\_curve(y, guesses) |

自实现的方式得到ROC曲线的代码如下：

|  |
| --- |
| **def** **calc\_fpr**(fp: int, tn: int) -> float:  **return** fp / (fp + tn) **def** **calc\_tpr**(tp: int, fn: int) -> float:  **return** tp / (tp + fn) **def** **get\_ftpr\_pairs**(  y\_pred\_prob: List[float],   y\_true: List[int]  ) -> Tuple[List[int], List[int]]:  fprs = [0]  tprs = [0]  **for** prob1 **in** y\_pred\_prob:  y\_pred\_i = []  **for** prob2 **in** y\_pred\_prob:  **if** prob2 < prob1:  y\_pred\_i.append(0)  **else**:  y\_pred\_i.append(1)  tp, fp, tn, fn = get\_confusion\_matrix(y\_pred\_i, y\_true)  fpr = calc\_fpr(fp, tn)  tpr = calc\_tpr(tp, fn)  fprs.append(fpr)  tprs.append(tpr)  fprs.append(1)  tprs.append(1)  **return** fprs, tprs fprs, tprs = get\_ftpr\_pairs(y\_pred\_prob, y\_true) fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(12, 5)) ax.plot(fprs, tprs) |



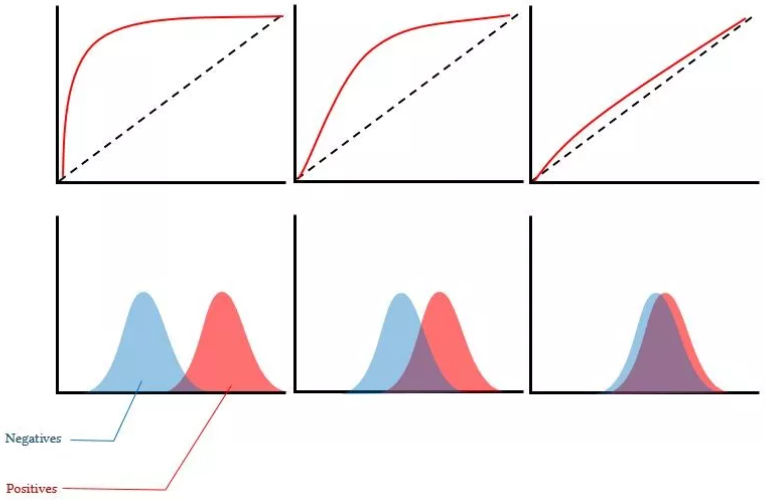
除此之外，还有一种绘制 ROC 曲线的方法：

* 假设有m+个正例，m-个负例，首先把分类阈值设为最大，即把所有样例均设为反例，此时真正例率和假正例率均为0，在坐标(0,0)处记一个点
* 然后依次将每个样本的预测值作为阈值（即将该样本作为正例），假设前一个坐标为（x, y），若当前为真正例，对应标记点为（x, y+1/m+），若当前为假正例，则对应标记点为（x+1/m-, y）
* 将所有点相连即可得到 ROC曲线

上面这种方法和下面这种方法是一样的：将纵坐标的刻度间隔设为 1/m+，横坐标的刻度间隔设为 1/m-，从（0,0）开始，每遇到一个真正例就沿着纵轴绘制一个刻度间隔的曲线，假正例就沿着横轴绘制一个刻度间隔的曲线，最终就可以得到 ROC曲线。

|  |
| --- |
| **def** **get\_ftpr\_pairs2**(  y\_pred\_prob: List[float],   y\_true: List[int]  ) -> Tuple[List[int], List[int]]:  mplus = sum(y\_true)  msub = len(y\_true) - mplus  pairs = [(0, 0)]  prev = (0, 0)  length = len(y\_pred\_prob)  **assert** length == len(y\_true)  **for** i **in** range(length):  **if** y\_true[i] == 1:  pair = (prev[0], prev[1] + 1/mplus)  **else**:  pair = (prev[0] + 1/msub, prev[1])  pairs.append(pair)  prev = pair  pairs.append((1, 1))  fprs, tprs = [], []  **for** pair **in** pairs:  fprs.append(pair[0])  tprs.append(pair[1])  **return** fprs, tprs fprs, tprs = get\_ftpr\_pairs2(y\_pred\_prob, y\_true) fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(12, 5)) ax.plot(fprs, tprs) |

ROC曲线中曲线相对于随机线的最高点，表示正例和负例分布的分离程度(一般来说分离程度越大，ROC曲线的“小山包”隆起越明显)，“小山包”的面积(ROC曲线和随机线围住的面积)就是Gini指数，如下图所示：



多个模型时，与 P-R 曲线也是类似，如果某个模型的曲线完全 “包住” 另一个，则前者性能好于后者。如果曲线相互交叉，则比较曲线下面积：AUC（Area Under ROC Curve）。

AUC 取值一般在 0.5-1 之间，处于 y=x 直线的上方（如果不是的话，把预测概率翻转成 1-p 就能获得更好的模型）。AUC 值越大，说明模型越可能把真正例排在前面，性能越好。此时，假正例率很低同时真正例率很高，意味着召回高并且误判率小。对角线对应着随机模型（各占 50%），（0，1）点对应的是理想模型，即所有正例 100% 召回且没有一个负例被判别为正例。

由于 ROC 曲线越靠近坐标点 (0,1) 分类器越优，所以从 AUC 判断分类器优劣的标准如下：

* AUC = 1：完美分类器，采用这个预测模型时，存在至少一个阈值能得出完美预测。
* 0.5 < AUC < 1：优于随机猜测。这个分类器妥善设定阈值的话，能有预测价值。
* AUC = 0.5：跟随机猜测效果一样，分类器没有预测价值。
* AUC < 0.5：比随机猜测还差，但只要总是反预测而行，就优于随机猜测。

简单来说就是 AUC 值越大，则分类器越优。

AUC 面积可以通过以下公式进行估算：

若AUC考虑的是样本预测的排序质量，与排序误差紧密相连，排序 “损失” loss 可定义为：



该式子的意思是，如果正例预测值小于负例，计1个罚分，如果相等则计 0.5 个罚分。显然，该式对应的就是 ROC 曲线上面的面积。因此有：



与 P-R 曲线相比，ROC 曲线有一个特点：当正负样本的分布发生变化时，ROC 曲线形状能基本保持不变，而 P-R 曲线的形状一般会发生比较剧烈的变化。因此，当数据不均匀时，ROC 曲线更能够反映模型好坏。而这背后的原因是：

* P-R 曲线关注的是真实的正例和预测的正例中（分别对应 Recall 和 Precision），实际是正例的比例
* ROC 曲线关注的是真实的正例和负例中（分别对应 TPR 和 FPR），被预测为正例的比例

采用sklearn计算AUC-score的代码如下：

|  |
| --- |
| import numpy as np  from sklearn.metrics import roc\_auc\_score  y\_true = np.array([0, 0, 1, 1])  y\_scores = np.array([0.1, 0.4, 0.35, 0.8])  roc\_auc\_score(y\_true, y\_scores) |

## 平均精度均值

平均精确均值（mean Average Precision，mAP）是对平均精度（Average Precision，AP）再次平均。因此需要先了解什么是平均精度

下面举例说明。

假设模型对6个样本的预测结果如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序列 | N=1 | N=2 | N=3 | N=4 | N=5 | N=6 |
| 真实标签 | 1 | 0 | 0 | 1 | 1 | 1 |

其中，1代表正样本，0代表负样本。

按顺序计算每个位置上的精确率如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序列 | N=1 | N=2 | N=3 | N=4 | N=5 | N=6 |
| 真实标签 | 1 | 0 | 0 | 1 | 1 | 1 |
| Precision | 1/1 | 1/2 | 1/3 | 2/4 | 3/5 | 4/6 |

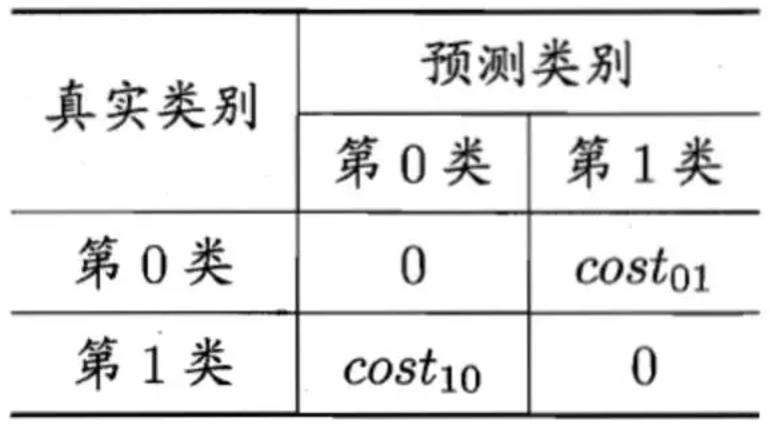
AP的计算只取正样本处的precision进行平均。则此序列的AP=(1/1 + 2/4 + 3/5 + 4/6)/4=0.6917

## 代价曲线

在某些现实任务中，不同类型的错误造成的后果不同。例如在医疗诊断中，将一名癌症患者错误地诊断成健康人士，将一名健康人士错误地诊断为癌症患者，这两种情况要承担的后果是截然不同的：后者仅是增加了进一步检查的麻烦，而前者若丧失佳治疗时机，则要付出生命的代价。

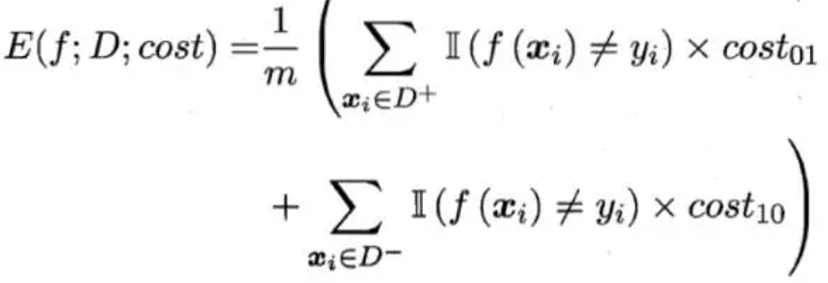
为此，我们为错误赋予非均等代价（unequal cost）来权衡不同类型错误所造成的不同损失。

以二分类任务为例，我们可以为任务设置一个代价矩阵（cost matrix），如下表所示：



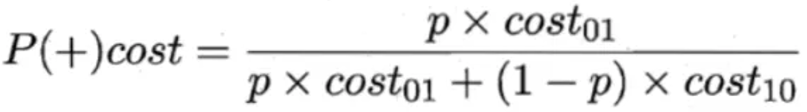
其中costij表示第i类样本错误预测为第j类的代价。一般情况下costii=0这是因为预测正确了，就没 有代价了。我们规定，若将第0类判别为第一类的所造成的损失更大，那么就有cost01>cost10， 损失程度越大则两者相差越大。但这个“相差”不是表示两者之间的差值，表示的是比值，举个例子 cost01:cost10=5:1的效果和 cost01:cost10=50:10是相同的。

前面我们考虑的性能度量都是假设在均等代价的情况下，也就是说，以减小错误次数为主，但不考虑不同的错误类型造成的后果的严重程度。而在非均等代价下，我们希望小化总体代价（total cost）。若设上表中第0类为正类，第1类为反类，令D+和D−分别代表样本集中的正类样本子集和 反类样本子集，则代价敏感（cost-sensitive）错误率为：



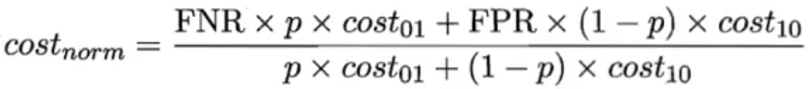
由此我们可以推出其它性能度量的代价敏感版本，或者是基于分布定义的代价敏感错误率。若 costij中的值不局限于0，1，还可以定义出多分类任务的代价敏感性能度量。

在非均等代价下，我们用代价曲线（cost curve）代替ROC曲线表现学习器的期望总体代价。代价曲线图横轴是取值为[0,1]的正例概率代价：



其中p表示样例为正例的概率。

纵轴是取值为[0,1]的归一化代价：



归一化是规范化（normalization）的特例，规范化表示将不同变化范围的值映射到某相同、固定 的范围当中，常见的固定范围是[0,1]，这个时候就是“归一化”。

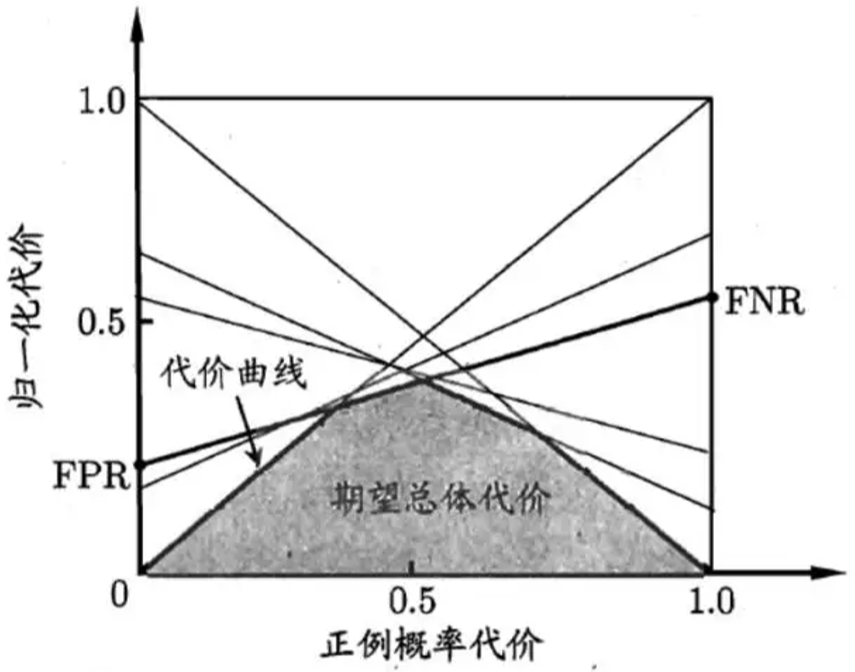
FPR即之前定义的假正例率，FNR=1-TPR 是假反例率

代价曲线的绘制方式：

（1）ROC曲线上每一点对应代价平面上的一条线段，设ROC曲线上的坐标为（FPR，TPR），计算出相应的FNR

（2）在代价平面上绘制一条从（0，FPR）到（1，FNR）的线段，线段下的面积就表示了该条件下的期望总体代价

（3）将ROC曲线上的每个点转化为代价平面上的一条线段，然后取所有线段的下界，围成的面积即为在所有条件下学习器的期望总体代价



## 马修斯相关系数

马修斯相关系数，简称MCC，与F1-score一样，是分类的综合评估指标。其公式化描述如下：

从公式中可以看出MCC和2\*2列联表的卡方检验很相近。MCC的值域为[-1,1]。

MCC的好处是：

* 正例和负例的不同划分对于最终结果没有影响

① TP = 0, FP = 0; TN = 5, FN = 95.

② TP = 95, FP = 5; TN = 0, FN = 0.

这两种条件下(正例和负例的划分不一样)得到的F1-Score差异非常大(①中为0，②中为0.97)，从这里还可以看出F1-Score在正负例样本比例差异不一致的情况下会高估模型的分类效果。

* 综合考虑了正负例样本比例不一致的情况

TP = 90, FP = 4; TN = 1, FN = 5. 这种条件下得到的分类正确率(Acc)为0.91，F1-Score为0.95，MCC得到的值为0.135. 例如风控业务中“坏”用户占整体用户的比例很小，如果看正确率或者F1-Score那就入坑了，此时MCC可能更合适。

## Balanced Classification Rate

Balanced Classification Rate简称BCR，BCR为正例样本和负例样本中各自预测正确率的均值，与F1-score一样，是分类的综合评估指标。其公式化定义为：

其中，TNR (True False Rate)，指负例样本中被正确判定为负例的样本数比例,也称为Specificity(特异度)。其公式化描述如下：

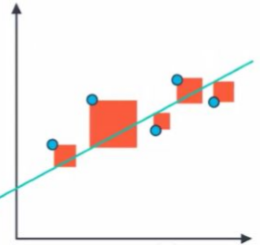
与BCR对应的是BER(Balanced Error Rate)，也称为Half Total Error Rate(HTER)。

同MCC一样，正负例的标签选择对BCR的计算没有影响，而且一定程度上也能克服正负例样本不均衡导致的评估指标虚高。

## 均方误差

均方误差(Mean Squared Error，简称MSE)误差函数把输出向量和真实向量映射到笛卡尔 坐标系的两个点上，通过计算这两个点之间的欧式距离(准确地说是欧式距离的平方)来衡 量两个向量之间差距。其公式化描述如下：

更一般的，对于数据分布D和概率密度函数p(⋅)，均方误差可描述为：



MSE 误差函数的值总是大于等于 0，当 MSE 函数达到最小值 0 时，输出等于真实标签， 此时神经网络的参数达到最优状态。

均方差误差函数广泛应用在回归问题中。在 TensorFlow 中，可以通过函数方式或层方式实现MSE误差计算。例如，使用函数方式实现MSE计算，代码如下：

|  |
| --- |
| o = tf.random.normal([2,10]) # 构造网络输出  y\_onehot = tf.constant([1,3]) # 构造真实值  y\_onehot = tf.one\_hot(y\_onehot, depth=10)  loss = keras.losses.MSE(y\_onehot, o) # 计算均方差  # 注意，MSE 函数返回的是每个样本的均方差，需要在样本维度上再次平均来获得平均样本的均方差  loss = tf.reduce\_mean(loss) # 计算 batch 均方差 |

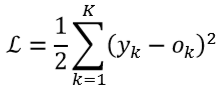
也可以通过层方式实现，对应的类为 keras.losses.MeanSquaredError()，和其他层的类一 样，调用\_\_call\_\_函数即可完成前向计算，代码如下：

|  |
| --- |
| # 创建 MSE 类  criteon = keras.losses.MeanSquaredError()  loss = criteon(y\_onehot, o) # 计算 batch 均方差 |

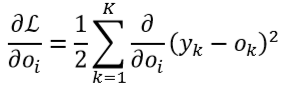
采用sklearn方式得到的均方误差的代码如下：

|  |
| --- |
| from sklearn.metrics import mean\_squared\_error  y\_true = [3, -0.5, 2, 7]  y\_pred = [2.5, 0.0, 2, 8]  mean\_squared\_error(y\_true, y\_pred) |

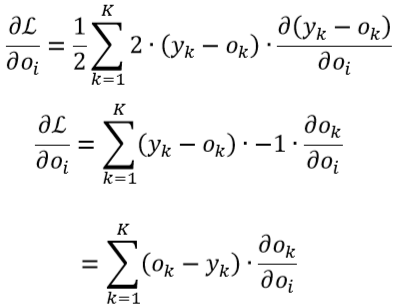
假设均方误差损失函数表达式为：



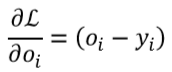
上式中1/2项用于简化计算，也可以利用1/𝐾进行平均，这些缩放运算均不会改变梯度方向。则它的偏导数∂ℒ/∂𝑜𝑖可以展开为：



利用复合函数导数法则分解为：



考虑到𝜕𝑜𝑘/𝜕𝑜𝑖，仅当𝑘 = 𝑖时才为1，其它点都为0，也就是说，偏导数𝜕ℒ/𝜕𝑜𝑖，只与第𝑖号节点相关，与其它节点无关，因此上式中的求和符号可以去掉。均方误差函数的导数可以推导为：



R²(R squared, Coefficient of determination)，中文翻译为“决定系数”或者“拟合优度”，反映的是预测值对实际值的解释程度。其取值范围是(0,1)，R²越接近1，说明拟合效果越好。其公式化描述如下：

注意：R²和相关系数的平方不是一回事(只在简单线性回归条件下成立).

## R2分数

R2分数（R2 Score）通过将我们的模型与最简单的可能模型相比得出。所谓最简单的可能模型，比如我们要拟合一个散点图，我们可以直接画一条直线穿过这些点。然后求出该直线模型下的均方误差。我们希望期望模型的均方误差比这个直线模型要小，而且是小得多，从而使得R2分数趋向于1。

采用sklearn得到R2分数的代码如下：

|  |
| --- |
| from sklearn\_metrics import r2\_score  y\_true = [1, 2, 4]  y\_pred = [1.3, 2.5, 3.7]  r2\_score(y\_true, y\_pred) |

## 均方根误差

均方根误差（Root Mean Square Error，RMSE），通常也称为，经常用于评估回归模型的好坏。其公式化定义如下：

其中，是第i个样本点的真实值，是第i个样本点的预测值，n是样本点的个数。

一般情况下，RMSE能够很好的反映回归模型的预测值与真实值的偏离程度。但如果存在一些偏离程度非常大的离群点，那么即使离群点数量非常少，也会让RMSE指标变得很差。为了解决这个问题，可以使用鲁棒性更强的平均绝对百分比误差（Mean Absolute Percent Error，MAPE）进行评估。

## 平均绝对百分比误差

平均绝对百分比误差（Mean Absolute Percent Error，MAPE）进行评估，MAPE的定义如下：

相比RMSE，MAPE相当于把每个点的误差进行了归一化，降低了离群点带来的绝对误差的影响。

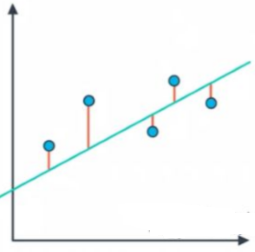
从MAPE公式中可以看出有个明显的缺陷，当实际值趋向0时就会得到无穷大值(实际 值的绝对值<1也会过度放大误差)。为了避免这个问题，MAPE一般用于实际值不会为 0的情形。为了解决这个问题，提出了MAAPE(mean arctangent absolute percentage error)。

## MAAPE

MAAPE(mean arctangent absolute percentage error)，在保持MAPE的算法思想下解决了当实际值趋向0时就会得到无穷大值的缺陷。其公式化定义为：

## 平均绝对误差

平均绝对误差（Mean Absolute Error），又被称为L1范数损失(l1-norm loss)，定义为预测值与真实值之差的绝对值，加起来求其平均。



采用sklearn得到平均绝对误差的代码如下：

|  |
| --- |
| from sklearn.metrics import mean\_absolute\_error  from sklearn.linear\_model import LinearRegression  classifier = LinearRegression()  classifier.fit(X, y)  guesses = classifier.predict(X)  error = mean\_absolute\_error(y, guesses) |

参考：

王喆，深度学习推荐系统。

龙书，TensorFlow深度学习。

<https://mp.weixin.qq.com/s/21421H7Yp-kGDXrJtGtDZA>

<https://mp.weixin.qq.com/s/oBV-WFWsCbpQcN1FssYf6A>

<https://mp.weixin.qq.com/s/Epu4g7IKOwZbxJBEK8CMIg>

<https://mp.weixin.qq.com/s/_38dpogJ4r3JArY9rdNDOw>

<https://mp.weixin.qq.com/s/R7XUko-dpvfu6WNBTv7YRw>

<https://mp.weixin.qq.com/s/0WPrEeN5rKLRxFME0dneSg>

<https://mp.weixin.qq.com/s/pj97tGpLdcDpOHJ9rYBmTA>

<https://mp.weixin.qq.com/s/vGuNd0guGzSFMpj81r3Z2A>

<https://mp.weixin.qq.com/s/93Qs0-NqLZkP_GKO373QHw>

<https://mp.weixin.qq.com/s/5kWdmi8LgdDTjJ40lqz9_A>