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

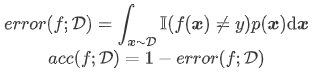
准确率（Accuracy）是指分类正确样本占总样本个数的比例，即：

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。

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：



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（Precision），也称为查准率，定义为分类正确的正样本个数占分类器判定为正样本的个数的比例。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。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。

精准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：

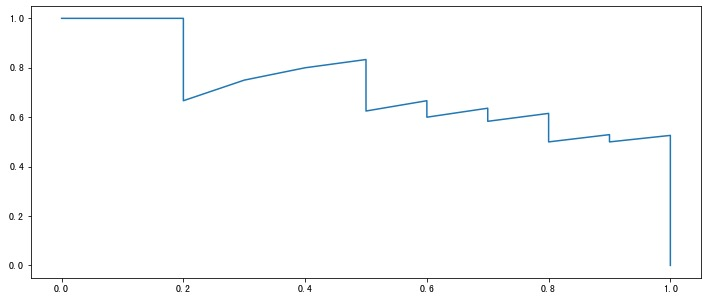
|  |  |  |
| --- | --- | --- |
| 真实情况 | 预测结果为正例 | 预测结果为反例 |
| 正例 | TP(真正例) | FN(假反例) |
| 反例 | FP(假正例) | TN(真反例) |

根据混淆矩阵可以得到精准率（P）和召回率（R）的计算公式：

得到P和R后就可以画出更加直观的P-R 图（P-R 曲线），横坐标为召回率，纵坐标是精准率。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和R，平衡点BEP（Break-Even Point），它是 P=R 时的取值，BPE 越远离原点，说明模型效果越好。由于BPE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，更偏好召回
* 当β < 1时，更偏好精准
* 当 β = 1 时，平衡精准和召回，即为 F1

F1 其实来自精准和召回的加权调和平均：

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
2. micro 方法：先计算混淆矩阵元素的平均，再计算 PR 和 F1

## 均方差

均方差(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：

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 |

## 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，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。

## 对数损失函数

对数损失函数（LogLoss）是经常在离线评估中使用的指标，在一个二分类问题中，LogLoss的定义如下：

其中，为输入实例的真实类别，为预测输入实例是正样本的概率，N是样本总数。

## 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”。

补充P-R曲线的绘制方法。

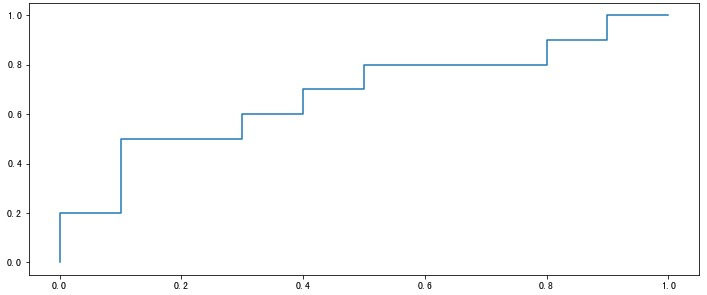
## ROC曲线

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）；纵坐标为True Positive Rate（TPR，真阳性率）。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。

补充ROC曲线的绘制方法。

|  |
| --- |
| **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，对模型输出的预测概率按从高到低排序
* 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); |

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。

AUC 面积可以通过以下公式进行估算：

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

## 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。

# 交叉熵损失函数

熵(Entropy)源于信息学，用于度量信息的不确定度。熵越大，代表信息的不确定性越大，信息也就越大（一种类比的理解就是，对于十分稳定的物体来说，一直不发生变化是确定的，发生变化所蕴含的信息量就更大）。某个分布P(i)的熵定义为：

实际上，𝐻(𝑃)也可以使用其他底数的log函数计算。举个例子，对于 4 分类问题，如果某个样本的真实标签是第4类，那么标签的One-hot编码为[0,0,0,1]，即这张图片的分类是唯一确定的，它属于第4类的概率𝑃(𝑦为4|𝒙) = 1，不确定性为0，它的熵可以简单的计算为：

也就是说，对于确定的分布，熵为0，不确定性最低。如果它预测的概率分布是[0.1,0.1,0.1,0.7]，它的熵可以计算为：

这种情况比前面确定性类别的例子的确定性要稍微大点。

考虑随机分类器，它每个类别的预测概率是均等的：[0.25,0.25,0.25,0.25]，同样的方法，可以计算它的熵约为2，这种情况的不确定性略大于上面一种情况。

由于𝑃(𝑖) ∈ [0,1], log2 𝑃(𝑖) ≤ 0，因此熵𝐻(𝑃)总是大于等于0。当熵取得最小值0时， 不确定性为0。

交叉熵(Cross Entropy)的定义如下：

𝐻(𝑝||𝑞) ≜ −∑𝑝(𝑖)log2 𝑞(𝑖)

# KS

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

# 评分卡

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

参考：

王喆，深度学习推荐系统。

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