机器学习模型评估指标总结！

原创 太子长琴 Datawhale 昨天

Datawhale干货

作者：太子长琴，Datawhale优秀学习者

本文对机器学习模型评估指标进行了完整总结。机器学习的数据集一般被划分为训练集和测试集，训练集用于训练模型，测试集则用于评估模型。针对不同的机器学习问题（分类、排序、回归、序列预测等），评估指标决定了我们如何衡量模型的好坏。

本文目录

一、Accuracy

准确率是最简单的评价指标，公式如下：

$$\frac{N_{correct}}{N_{total}}$$

但是存在明显的缺陷：

- 当样本分布不均匀时，指标的结果由占比大的类别决定。比如正样本占 99%，只要分类器将所有样本都预测为正样本就能获得 99% 的准确率。
- 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。

$$error(f;\mathcal{D}) = \int_{\boldsymbol{x}\sim\mathcal{D}} \mathbb{I}(f(\boldsymbol{x}) \neq y)p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$
$$acc(f;\mathcal{D}) = 1 - error(f;\mathcal{D})$$

```
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
```

二、Precision Recall 和 F1

精准率（Precision）也叫查准率，衡量的是所有预测为正例的结果中，预测正确的（为真正例）比例。

召回率（Recall）也叫查全率，衡量的是实际的正例有多少被模型预测为正例。

在排序问题中，一般以 TopN 的结果作为正例，然后计算前 N 个位置上的精准率 Precision@N 和召回率 Recall@N。

精确率和召回率是一对相互矛盾的指标，一般来说高精准往往低召回，相反亦然。其实这个是比较直观的，比如我们想要一个模型准确率达到 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	预测结果正例	预测结果反例
正例	TP(真正例)	FN(假反例)
反例	FP(假正例)	TN(真反例)

然后，很容易就得到精准率（P）和召回率（R）的计算公式：

$$P = \frac{TP}{TP+FP}$$
$$R = \frac{TP}{TP+FN}$$

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。

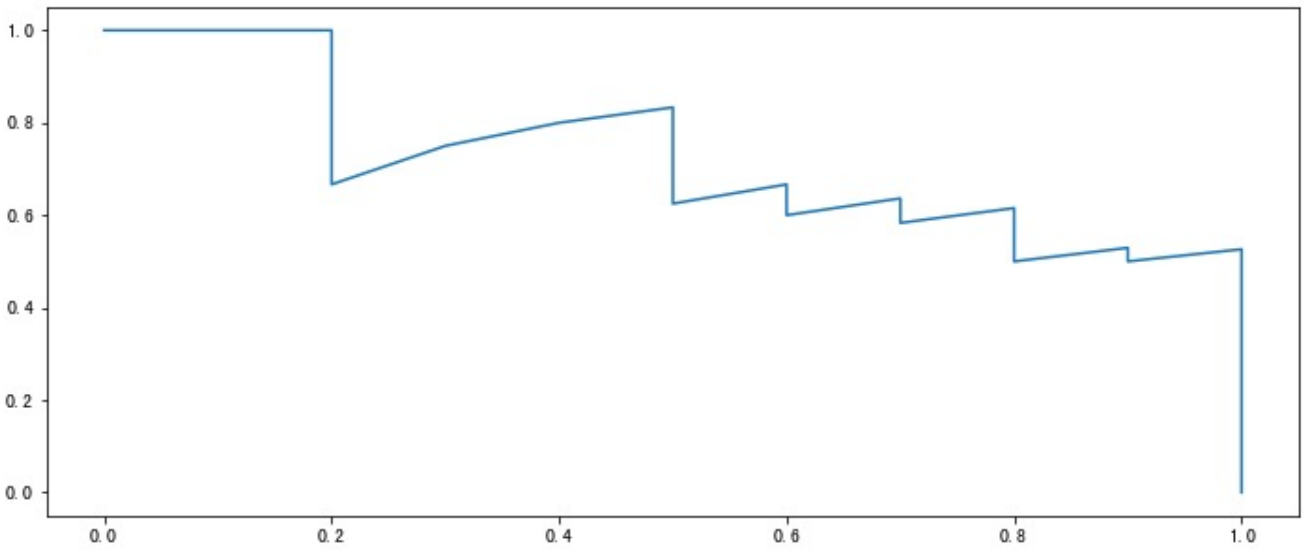
```
from typing import List, Tuple
import matplotlib.pyplot as plt

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] == 1and y_true[i] == 0:
            fp += 1
        elif y_pred[i] == 0and 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）就是这样一个度量，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，实际中常用 F1 值衡量：

$$F1 = \frac{2PR}{P+R}$$

F1 有更一般的形式：

$$F_\beta = \frac{\left(1+\beta^2\right) \times P \times R}{\left(\beta^2 \times P\right) + R}$$

- 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：

$$HarmonicMean(a_1, a_2, \ldots, a_n) = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots + \frac{1}{a_n}} F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = F_\beta\beta^2$$

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = F_\beta\beta^2 = \frac{1-\alpha}{\alpha}$$

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：

- macro 方法：先计算每个 PR，取平均后，再计算 F1
- micro 方法：先计算混淆矩阵元素的平均，再计算 PR 和 F1

三、RMSE

均方根误差 RMSE（Root Mearn Square Error）主要用在回归模型，也就是俗称的 R 方。计算公式为：

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

但是如果有非常严重的离群点时，那些点会影响 RMSE 的结果，针对这个问题：

- 如果离群点为噪声，则去除这些点

- 如果离群点为正常样本，可以重新建模

- 换一个评估指标，比如平均绝对百分比误差 MAPE（Mean Absolute Percent Error），MAPE 对每个误差进行了归一化，一定程度上降低了离群点的影响。

$$MAPE = \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times \frac{100}{n}$$

四、ROC 和 AUC

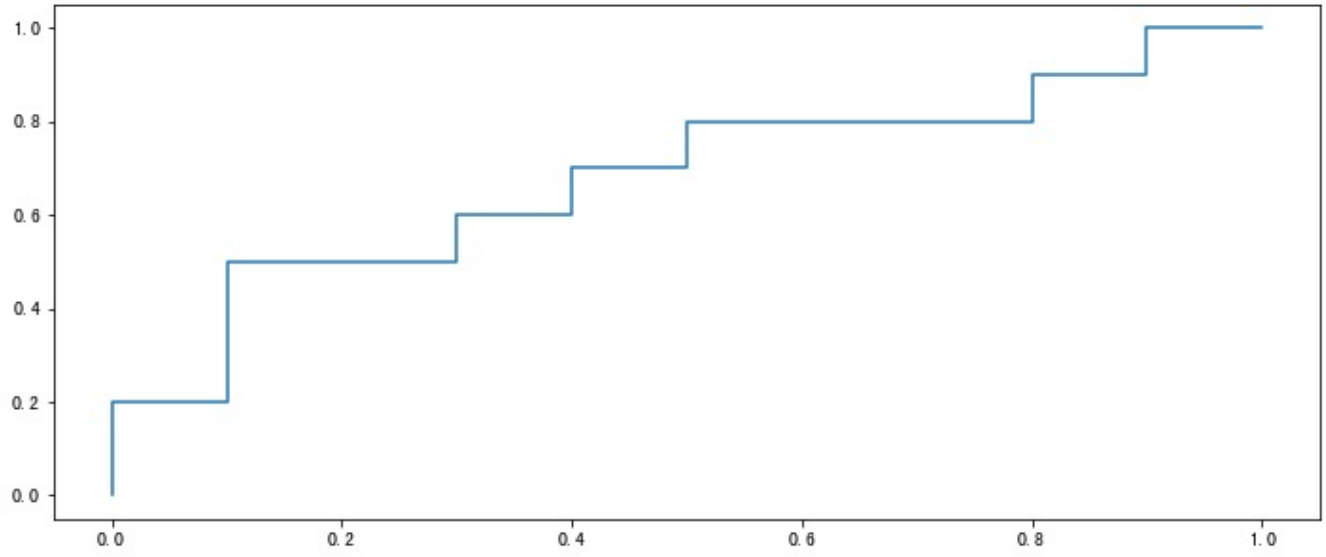
受试者工作特征 **ROC**（Receiver Operating Characteristic）曲线是另一个重要的二分类指标。它的横坐标是 “假正例率” **FPR**（False Positive Rate），纵坐标是 “真正例率” **TPR**（True Positive Rate），计算公式如下：

$$FPR = \frac{FP}{FP + TN} TPR = \frac{TP}{TP + FN}$$

绘制方法和上面的 P-R 曲线类似，不再赘述。

```
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);
```



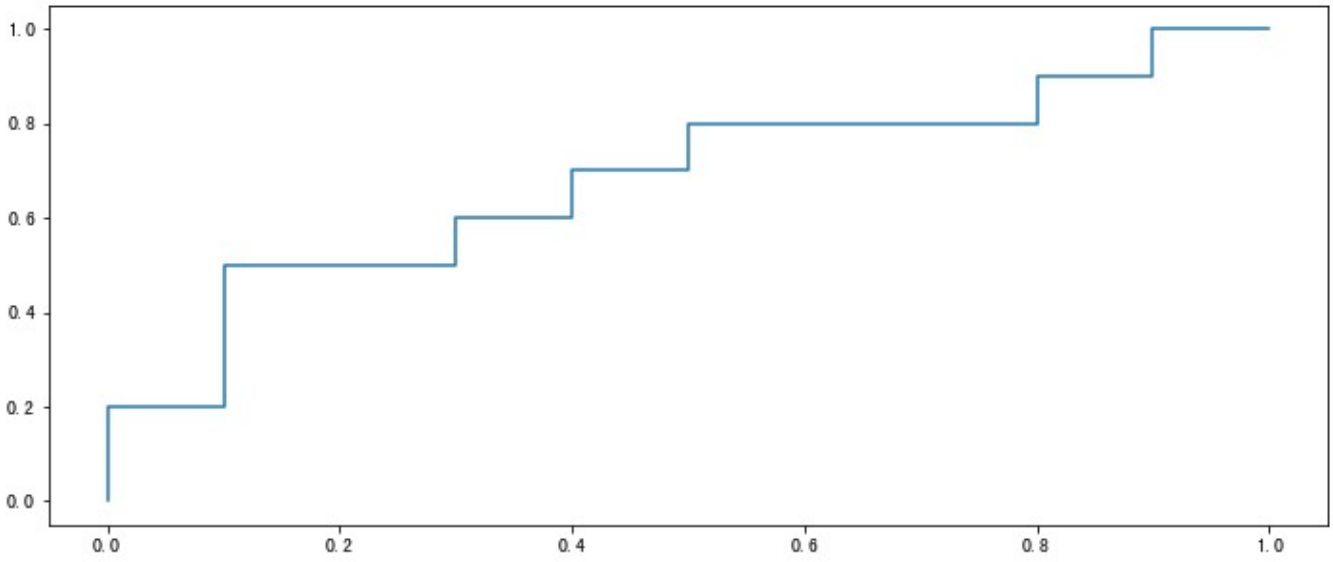
除此之外，还有一种绘制 ROC 曲线的方法：

- 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

该方法和这种做法是一样的：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);
```



该方法和上面第一种方法得到的曲线完全一致。

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：

$$AUC = \frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1} - x_i)\cdot(y_i + y_{i+1})$$

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：

$$\ell_{rank} = \frac{1}{m^+m^-}\sum_{x^+\in D^+}\sum_{x^-\in D^-}\left(\mathbb{I}(f(x^+) < f(x^-)) + \frac{1}{2}\mathbb{I}(f(x^+) = f(x^-))\right)$$

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：

$$AUC = 1 - \ell_{rank}$$

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

五、KS

作为一个工程师，看到 KS 我们的第一反应应该是：既然已经有了 PR、ROC 等评价指标，为什么还需要 KS？它解决了前面指标解决不了的什么问题？它究竟有什么特点？

KS Test（Kolmogorov-Smirnov）是由两位苏联数学家 A.N. Kolmogorov 和 N.V. Smirnov 提出的，用于比较样本与参考概率分布或比较两个样本的非参数检验。

我们以两样本为例，假设 m 个 sample 来自分布 F(x)，n 个来自 G(x)，定义 KS 统计量（KS 距离）为：

$$D_{m,n} = \sup_x |F_m(x) - G_n(x)|$$

其中 F(x) 和 G(x) 都是经验累积分布函数 ECDF（empirical distribution function），定义如下：

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[-\infty,x]}(X_i) if \quad X_i \le x, \quad I(X_i) = 1, \quad else \quad 0$$

sup 表示上确界，也是最小上界。

原始假设 H0：两组 sample 来自统一分布，在大样本上，在置信水平 α 下如果满足下面的条件则拒绝零假设（认为两组样本来自不同分布）：

$$D_{m,n} > c(\alpha)\sqrt{\frac{m+n}{m \cdot n}} s.t. \quad c(\alpha) = \sqrt{-\ln(\frac{\alpha}{2}) \cdot \frac{1}{2}}$$

代入后得到：

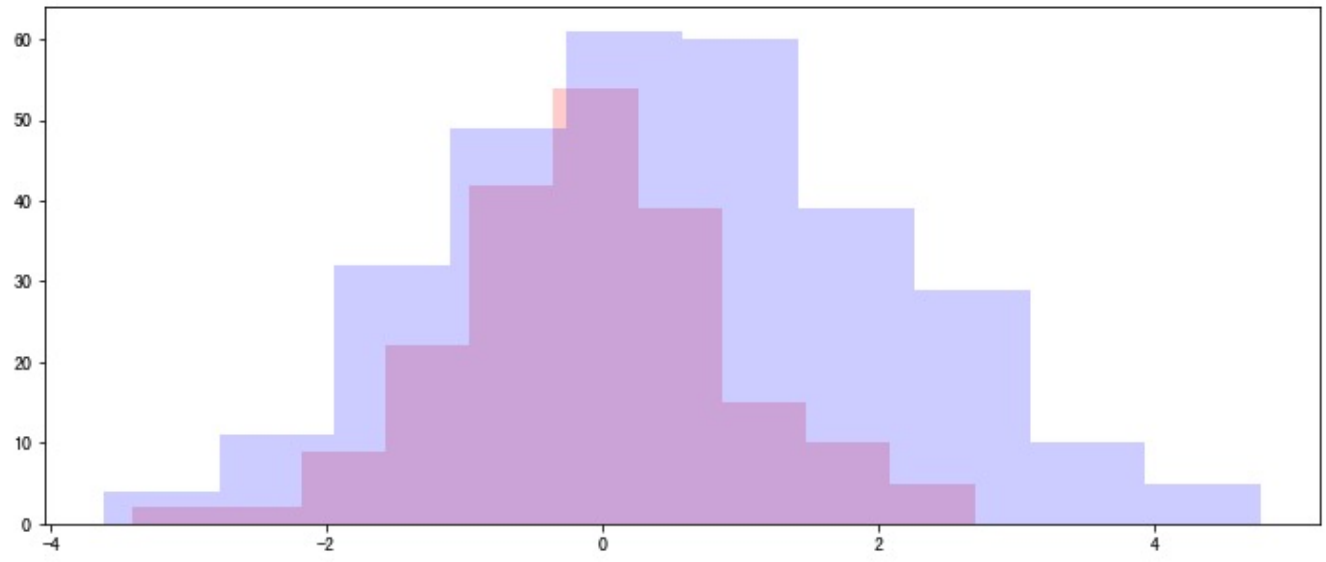
$$D_{m,n} > \frac{1}{\sqrt{m}}\sqrt{-\ln(\frac{\alpha}{2}) \cdot \frac{1+\frac{m}{n}}{2}}$$

常用的值如下：

ALPHA	0.10	0.05	0.01	0.005
c(α)	1.224	1.358	1.628	1.731

```
from scipy import stats
rvs1 = stats.norm.rvs(size=200, loc=0., scale=1)
rvs2 = stats.norm.rvs(size=300, loc=0.5, scale=1.5)
stats.ks_2samp(rvs1, rvs2)
# 在置信度 0.05 水平下：1.358 * np.sqrt(500/60000) = 0.124
# Ks_2sampResult(statistic=0.265, pvalue=7.126401335710852e-08)
# 0.265 > 0.124 所以拒绝原假设，即认为两组样本来自不同分布
# 事实上，即便是 0.005 的置信水平下依然要拒绝原假设
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(12, 5))
ax.hist(rvs1, density=False, histtype='stepfilled', alpha=0.2, color='red');
ax.hist(rvs2, density=False, histtype='stepfilled', alpha=0.2, color='blue');
```

其中 statistic 就是 ks 统计量。



那这又和评价指标有啥关联呢？

我们考虑这么一种情况，假设数据集的 Label 并不是离散的（如二分类的 0-1），而是可能满足一定分布，也就是说标签有很多灰色地带。其实这在实际生活中倒是更加常见，以金融风控为例，不少特征都是基于某个时间点做划分的，比如逾期还款 x 天，这个 x 是非常灵活的，而且也很难说 x-1 天的就一定比 x+1 天的信用好。这就意味着给定特征下，我们的标签最好能够有一定 “弹性”。

那么，怎么去体现这个 “弹性” 呢？因为 KS 正好是衡量两个 “分布” 的 “距离”，我们可以构造一个函数：

$$ks = |TPR - FPR|$$

然后我们可以画出 KS 曲线，可以证明，KS 和 ROC 等价，且满足如下公式：

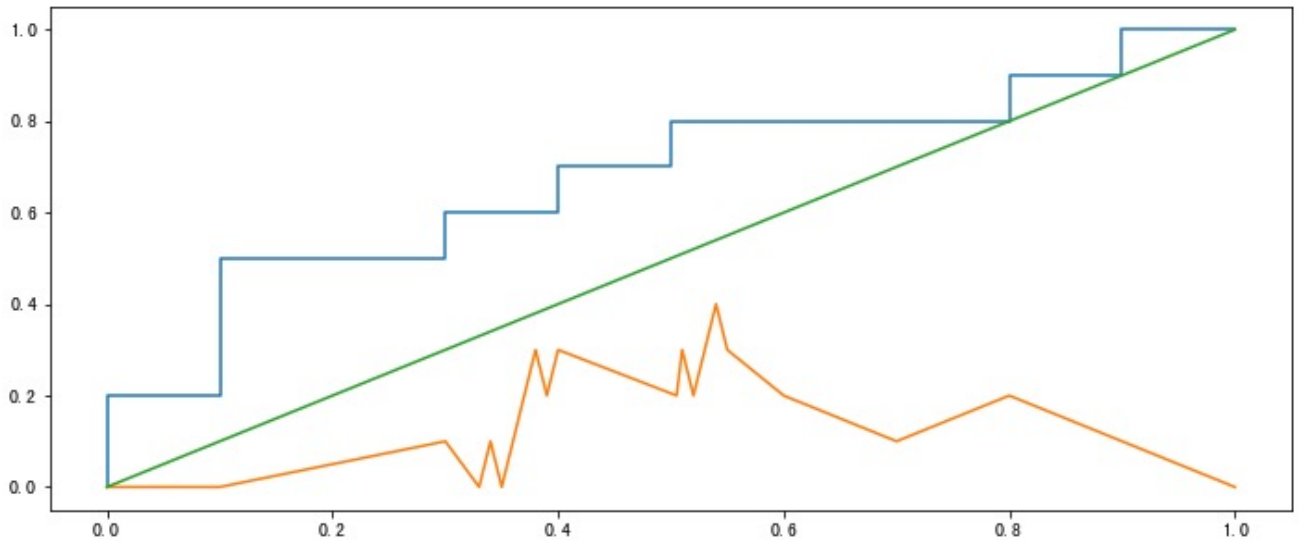
$$AUC_{ROC} = 0.5 + AUC_{KS}$$

KS 的最大值就用来评估模型的区分度。而所谓的区分度正可以看作是正负例的差异，具体而言，如果正负例对于标签没有区分度，说明两个样本重叠较大；区分度越大，说明两个概率分布相隔越远。回到 KS 上：

- 如果 KS 的最大值很小，说明 TPR 和 FPR 接近同一分布，也就意味着真实的正例和负例被预测为正例的比例相似，说明模型很差。
- 如果 KS 的最大值很大，说明 TPR 和 FPR 区别很大，意味着真实的正例被预测为正例和真实的负例被预测为正例相差很大，说明模型效果较好（能够区分真实正例和真实负例）。

事实上，KS 的确常用在金融风控中，用来评估模型的区分度，区分度越大说明模型的风险排序能力越强。但值太大也有问题（可能过拟合），一般超过 0.75 就认为过高，而低于 0.2 则过

低。关于这个我们可以看图说明：



我们假设曲线光滑，那么 AUC_KS ≈ 1/2 × max_KS，根据前面的公式：

$$AUC_{ROC} \approx \frac{1}{2} + \frac{\max_{KS}}{2}$$

由于上面提到的金融风控中 Label 的弹性，当 KS 过高时，ROC 的 AUC 就会很高，说明结果并没有这种弹性（模糊性、连续性），此时模型有过拟合风险。

既然 KS 可以，那我们自然就要问了，t 检验行不行？因为 t 检验也是检验两组样本是否来自同一个分布的统计量啊。答案是：不行。因为我们实际上是使用了它的定义（距离），而 t-test 的定义并没有体现出这一点。

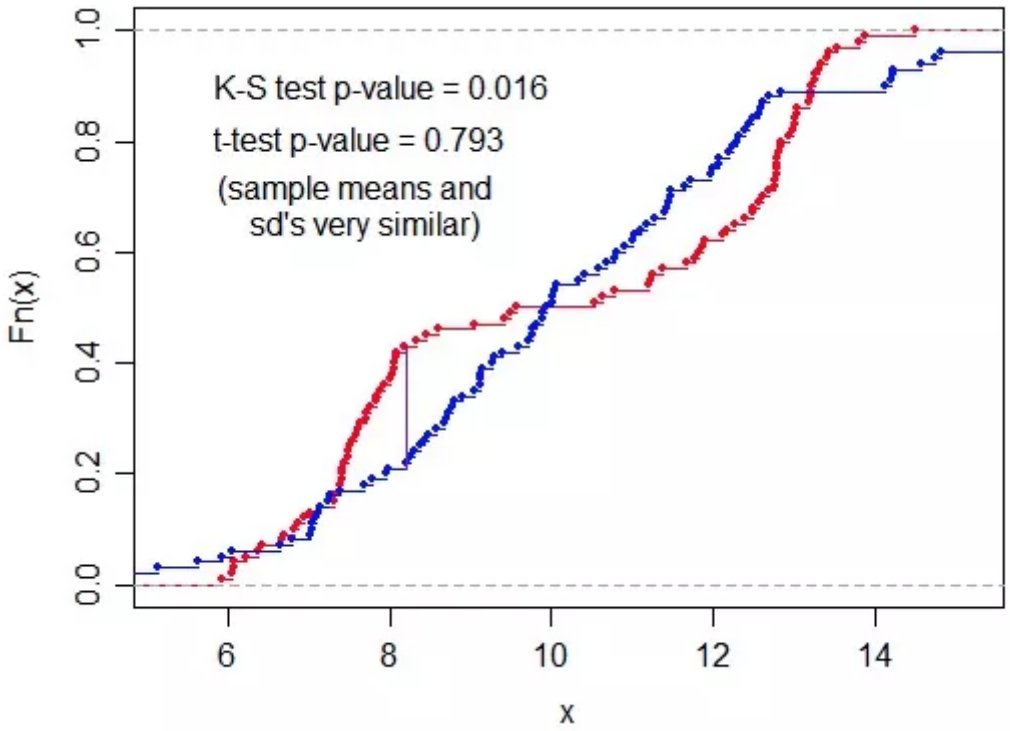
独立双样本 t 检验，方差不相等：

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_{\bar{\Delta}}}$$
$$s.t. \quad s_{\bar{\Delta}} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

独立双样本 t 检验，样本数相同，方差相似：

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_p\sqrt{\frac{2}{n}}}$$
$$s.t. \quad s_p = \sqrt{\frac{s_{X_1}^2 + s_{X_2}^2}{2}}$$

这里的图也可以说明这一点：



其他距离其实也没有太多意义，因为 FPR 和 TPR 的 x 是一样的，不同的也就是 y 值。

六、评分卡

评分卡模型是一个线性回归模型：

$$Y = \sum_{i=0}^{n} \theta_i x_i + b$$

特征覆盖率高，保持稳定，特征变量有明显的可解释性。样本为 0 时可以根据专家历史经验设定权重；样本为几百时，可根据单特征区分能力如 KS/IV 值等进行权重设定。

6.1 非线性处理

有两种方式：WOE 处理和分桶。

证据权重 WOE（Weight of Evidence）是一种自变量编码方案，定义为：

$$WOE_i = \ln\left(\frac{B_i/B_T}{G_i/G_T}\right)$$

其中，Bi 表示第 i 个分组里 bad label 的数量，Bt 为总的 bad label 数量；G 表示 good label。WOE 越大，bad label 比例越高，此时的 WOE 值可以作为该分组的特征值。

分桶是指对有一定跳变的连续值特征进行分桶，将弱线性特征转化为强线性特征。

6.2 交叉特征处理

主要采取对客户分群的方式，对细分群体进行单独建模（本质上是一种交叉特征的体现）。

“整理不易，**点赞三连↓**