

关于集成学习, sklearn中是ensemble模块, 即sklearn.ensemble. 里面包含了AdaBoost, 随机森林等常用的集成方法, 本文用的是AdaBoostClassifier.

## [sklearn.ensemble](#) : Ensemble Methods ¶

The `sklearn.ensemble` module includes ensemble-based methods for classification, regression and anomaly detection.

**User guide:** See the [Ensemble methods](#) section for further details.

<code>ensemble.AdaBoostClassifier</code> ([...])	An AdaBoost classifier.
<code>ensemble.AdaBoostRegressor</code> ([base_estimator, ...])	An AdaBoost regressor.
<code>ensemble.BaggingClassifier</code> ([base_estimator, ...])	A Bagging classifier.
<code>ensemble.BaggingRegressor</code> ([base_estimator, ...])	A Bagging regressor.
<code>ensemble.ExtraTreesClassifier</code> ([...])	An extra-trees classifier.
<code>ensemble.ExtraTreesRegressor</code> ([n_estimators, ...])	An extra-trees regressor.
<code>ensemble.GradientBoostingClassifier</code> ([loss, ...])	Gradient Boosting for classification.
<code>ensemble.GradientBoostingRegressor</code> ([loss, ...])	Gradient Boosting for regression.
<code>ensemble.IsolationForest</code> ([n_estimators, ...])	Isolation Forest Algorithm
<code>ensemble.RandomForestClassifier</code> ([...])	A random forest classifier.
<code>ensemble.RandomForestRegressor</code> ([...])	A random forest regressor.
<code>ensemble.RandomTreesEmbedding</code> ([...])	An ensemble of totally random trees.
<code>ensemble.VotingClassifier</code> (estimators[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
<code>ensemble.VotingRegressor</code> (estimators[, ...])	Prediction voting regressor for unfitted estimators.
<code>ensemble.HistGradientBoostingRegressor</code> ([...])	Histogram-based Gradient Boosting Regression Tree.
<code>ensemble.HistGradientBoostingClassifier</code> ([...])	Histogram-based Gradient Boosting Classification Tree.

# — ensemble.AdaBoostClassifier

## sklearn.ensemble.AdaBoostClassifier

```
class sklearn.ensemble. AdaBoostClassifier (base_estimator=None, n_estimators=50, learning_rate=1.0,  
algorithm='SAMME.R', random_state=None) [source]
```

An AdaBoost classifier.

An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

This class implements the algorithm known as AdaBoost-SAMME [2].

Read more in the [User Guide](#).

**Parameters:** **base\_estimator** : *object, optional (default=None)*

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier(max_depth=1)`

**n\_estimators** : *integer, optional (default=50)*

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

**learning\_rate** : *float, optional (default=1.)*

Learning rate shrinks the contribution of each classifier by `learning_rate`. There is a trade-off between `learning_rate` and `n_estimators`.

**algorithm** : *{'SAMME', 'SAMME.R'}, optional (default='SAMME.R')*

If 'SAMME.R' then use the SAMME.R real boosting algorithm. `base_estimator` must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

**random\_state** : *int, RandomState instance or None, optional (default=None)*

If int, `random_state` is the seed used by the random number generator; If `RandomState` instance, `random_state` is the random number generator; If `None`, the random number generator is the `RandomState` instance used by `np.random`.

### 参数说明

AdaBoostClassifier中主要的参数如下:

- **base\_estimator: 基分类器, 可选参数, 默认为决策树 (DecisionTreeClassifier)**. 理论上可以选择任何一个分类或者回归学习器, 不过需要支持样本权重. 我们常用的一般是CART决策树或者神经网络MLP. 默认是决策树, 即AdaBoostClassifier默认使用CART分类树DecisionTreeClassifier, 而AdaBoostRegressor默认使用CART回归树DecisionTreeRegressor. 另外有一个要注意的点是, 如果我们选择的AdaBoostClassifier算法是SAMME.R, 则我们的弱分类学习器还需要支持概率预测, 也就是在scikit-learn中弱分类学习器对应的预测方法除了predict还需要有predict\_proba.
- **algorithm: 算法, 也就是模型提升准则. 可选参数, 默认为SAMME.R**. scikit-learn实现了两种Adaboost分类算法, SAMME和SAMME.R. 两者的主要区别是弱学习器权重的度量, SAMME使用对样本集分类效果作为弱学习器权重, 而SAMME.R使用了对样本集分类的预测概率大小来作为弱学习器权重. 由于SAMME.R使用了概率度量的连续值, 迭代一般比SAMME快, 因此AdaBoostClassifier的默认算法algorithm的值也是SAMME.R. 我们一般使用默认的SAMME.R就够了, 但是要注意的是使用了SAMME.R, 则弱分类学习器参数base\_estimator必须限制使用支持概率预测的分类器. SAMME算法则没有这个限制.
- **n\_estimators: 基分类器提升 (循环) 次数, 整数型, 可选参数, 默认为50**. 弱学习器的最大迭代次数, 或者说最大的弱学习器的个数. 一般来说n\_estimators太小, 容易欠拟合, n\_estimators太大, 又容易过拟合, 一般选择一个适中的数值. 默认是50. 在实际调参的过程中, 我们常常将n\_estimators和下面介绍的参数learning\_rate一起考虑.

- **learning\_rate:** 学习率, 表示梯度收敛速度, 浮点型, 可选参数, 默认为1.0. 每个弱学习器的权重缩减系数, 取值范围为0到1, 对于同样的训练集拟合效果, 较小的 $\nu$ 意味着我们需要更多的弱学习器的迭代次数. 通常我们用步长和迭代最大次数一起来决定算法的拟合效果. 所以这两个参数  $\nu$ 和learning\_rate要一起调参. 一般来说, 可以从一个小一点的 $\nu$ 开始调参, 默认是1.
- **random\_state:** 随机种子设置, 整数型, 可选参数, 默认为None. 如果RandomState的实例, random\_state是随机数生成器; 如果None, 则随机数生成器是由np.random使用的RandomState实例.

### 主要方法:

同时, AdaBoostClassifier中还有一些方法, 如下:

#### Methods

<code>decision_function</code> (self, X)	Compute the decision function of X .
<code>fit</code> (self, X, y[, sample_weight])	Build a boosted classifier from the training set (X, y).
<code>get_params</code> (self[, deep])	Get parameters for this estimator.
<code>predict</code> (self, X)	Predict classes for X.
<code>predict_log_proba</code> (self, X)	Predict class log-probabilities for X.
<code>predict_proba</code> (self, X)	Predict class probabilities for X.
<code>score</code> (self, X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
<code>set_params</code> (self, **params)	Set the parameters of this estimator.
<code>staged_decision_function</code> (self, X)	Compute decision function of X for each boosting iteration.
<code>staged_predict</code> (self, X)	Return staged predictions for X.
<code>staged_predict_proba</code> (self, X)	Predict class probabilities for X.
<code>staged_score</code> (self, X, y[, sample_weight])	Return staged scores for X, y.

`decision_function(X)`: 返回决策函数值 (比如svm中的决策距离)

`fit(X,Y)`: 在数据集 (X,Y) 上训练模型.

`get_parms()`: 获取模型参数

`predict(X)`: 预测数据集X的结果.

`predict_log_proba(X)`: 预测数据集X的对数概率.

`predict_proba(X)`: 预测数据集X的概率值.

`score(X,Y)`: 输出数据集 (X,Y) 在模型上的准确率.

`staged_decision_function(X)`: 返回每个基分类器的决策函数值

`staged_predict(X)`: 返回每个基分类器的预测数据集X的结果.

`staged_predict_proba(X)`: 返回每个基分类器的预测数据集X的概率结果.

`staged_score(X, Y)`: 返回每个基分类器的预测准确率.

## 二 利用Sklearn构建AdaBoost分类器

机器学习实战中的"在一个难数据集上应用 AdaBoost ", 马疝病数据集上应用AdaBoost分类器.

具体代码如下:

```
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
```

```

from sklearn.tree import DecisionTreeClassifier

def loadDataset(fileName):
    #特征数目
    numFeat = len((open(fileName).readline().split('\t')))
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = []
        curLine = line.strip().split('\t')
        for i in range(numFeat - 1):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat, labelMat

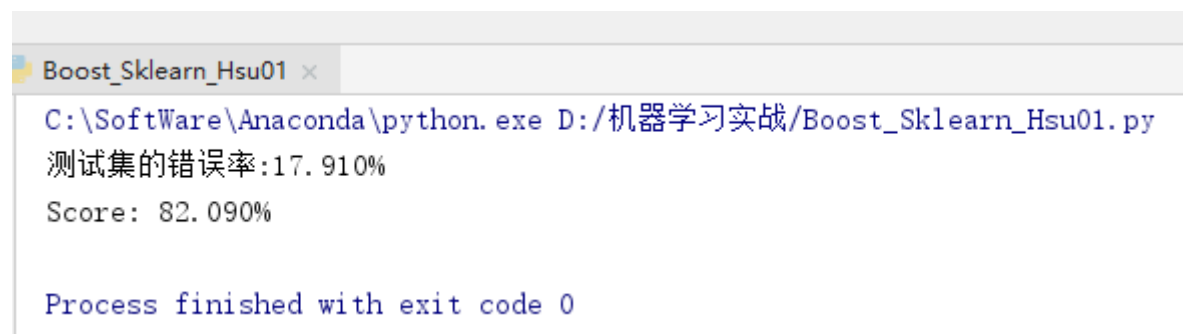
if __name__ == '__main__':
    dataArr, classLabels = loadDataset('horseColicTraining2.txt')
    testArr, testLabelArr = loadDataset('horseColicTest2.txt')
    bdt =
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2), algorithm=
'SAMME', n_estimators= 10)
    bdt.fit(dataArr, classLabels)
    predictions = bdt.predict(testArr)
    errArr = np.mat(np.ones((len(testArr), 1)))
    print('测试集的错误率:%.3f%%' % float(errArr[predictions !=
testLabelArr].sum() / len(testArr)*100))

    scores = bdt.score(testArr, testLabelArr)
    print("Score: %.3f%%" % (scores*100))

```

上面的我用了两种方式来看准确率的情况, 第一个是自己构建的错误率情况, 第二个是用里面的score方法, 测出准确率情况.

结果如下:



```

Boost_Sklearn_Hsu01 x
C:\Software\Anaconda\python.exe D:/机器学习实战/Boost_Sklearn_Hsu01.py
测试集的错误率:17.910%
Score: 82.090%

Process finished with exit code 0

```

## 三 性能度量

对学习器泛化性能的评估, 在有了可行的实验估计方法后, 还需要有衡量模型泛化能力的评价标准, 这就是性能度量.

在对比不同模型的能力时,使用**不同的性能度量**往往会导致不同的评判结果;这意味着模型的**"好坏"是相对的**,什么样的模型是好,不仅取决于**算法和数据**,还决定于**任务需求**.

给定样例集  $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ , 其中  $y_i$  是示例  $\mathbf{x}_i$  的真实标记. 要评估学习器  $f$  的性能, 就要把学习器预测结果  $f(\mathbf{x})$  与真实标记  $y$  进行比较.

回归任务中常用的性能度量是**"均方误差"** (mean squared error)

$$E(f; D) = \frac{1}{m} \sum_{i=1}^m (f(\mathbf{x}_i) - y_i)^2 \quad (2.2)$$

更一般的, 对于数据分布  $\mathcal{D}$  和概率密度函数  $p(\cdot)$ , 均方误差可描述为

$$E(f; \mathcal{D}) = \int_{\mathbf{x} \sim \mathcal{D}} (f(\mathbf{x}) - y)^2 p(\mathbf{x}) d\mathbf{x} \quad (2.3)$$

## (一) 错误率与精度

**错误率**是分类错误的样本数占样本总数的比例, **精度**则是分类正确的样本数占样本总数的比例.

对于样例集  $D$ , 分类错误率定义为

$$E(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(\mathbf{x}_i) \neq y_i) \quad (2.4)$$

精度则为

$$\begin{aligned} \text{acc}(f; D) &= \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(\mathbf{x}_i) = y_i) \\ &= 1 - E(f; D) \end{aligned} \quad (2.5)$$

更为一般的, 对于数据分布  $\mathcal{D}$  和概率密度函数  $p(\cdot)$ , 错误率与精度可以定义为

$$E(f; \mathcal{D}) = \int_{\mathbf{x} \sim \mathcal{D}} \mathbb{I}(f(\mathbf{x}) \neq y) p(\mathbf{x}) d\mathbf{x} \quad (2.6)$$

$$\begin{aligned} \text{acc}(f; \mathcal{D}) &= \int_{\mathbf{x} \sim \mathcal{D}} \mathbb{I}(f(\mathbf{x}) = y) p(\mathbf{x}) d\mathbf{x} \\ &= 1 - E(f; \mathcal{D}) \end{aligned} \quad (2.7)$$

## (二) 查准率、查全率与 $F1$

错误率衡量了有多少比例的瓜被判别错误, 但**"挑出的西瓜中有多少比例是好瓜"**—查准率 或者 **"所有好瓜中有多少比例被挑了出来"**—查全率, 这两个指标同样非常重要.

### 1 查准率和查全率的定义

对于二分类问题, 可将样例根据其**真实类别**与**学习器预测类别**的组合划分为**真正例 (true positive)**、**假正例 (false positive)**、**真反例 (true negative)**、**假反例 (false negative)** 四种情形, 令  $TP$ 、 $FP$ 、 $TN$ 、 $FN$  分别表示其对应的样例数, 则显然有  $TP + FP + TN + FN = \text{样例总数}$ . 分类结果的**"混淆矩阵"** (confusion matrix) 如表 2.1 所示:

表 2.1 分类结果混淆矩阵

真实情况	预测结果	
	正例	反例
正例	$TP$ (真正例)	$FN$ (假反例)
反例	$FP$ (假正例)	$TN$ (真反例)

查准率  $P$  和查全率  $R$  分别定义为

$$P = \frac{TP}{TP + FP} \quad (2.8)$$

$$R = \frac{TP}{TP + FN} \quad (2.9)$$

查准率和查全率互为矛盾体:

查准率和查全率是一对矛盾的度量. 一般来说, 查准率高时, 查全率往往偏低; 而查全率高时, 查准率往往偏低.

## 2 $P - R$ 曲线

根据学习器的预测结果对样例进行排序, 排在前面的是学习器认为"最可能"是正例的样本, 排在最后的则是学习器认为"最不可能"是正例的样本.

按此顺序逐个把样本作为正例进行预测, 则每次可以计算出当前的查全率、查准率. 以查准率为纵轴、查全率为横轴作图, 就得到了查准率-查全率曲线, 简称" $P - R$  曲线". 如下图

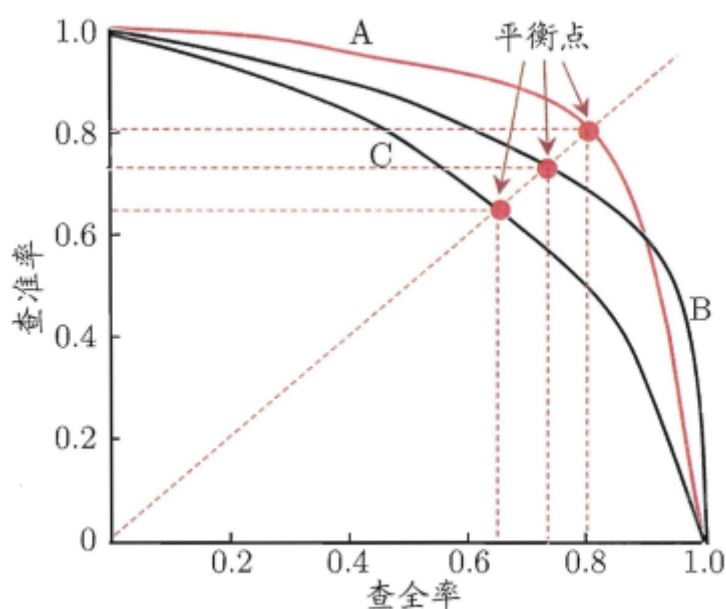


图 2.3  $P-R$  曲线与平衡点示意图

$P - R$  图直观地显示出学习器在样本总体上的查全率、查准率.

- **1 包住:** 在进行比较时, 若一个学习器的  $P - R$  曲线被另一个学习器的曲线完全"包住", 则可断言后者的性能优于前者, 如图 2.3 中学习器 A 的性能优于学习器 C;

- **2 交叉**: 如果两个学习器的  $P-R$  曲线发生了**交叉**, 例如图 2.3 中的 学习器  $A$  与  $B$ , 则难以一般性地断言两者孰优孰劣, 只能在具体的查准率或查全率条件下进行比较. 这时一个比较合理的判据是比较  $P-R$  曲线节面积的大小. 但这个值不太容易估算. 下面的**平衡点**就时一个综合考虑查准率、查全率的性能度量指标.

### 3 平衡点

"**平衡点**" (Break-Event Point, 简称  $BEP$ ) 就是这样一个度量, 它是"**查准率=查全率**"时的取值, 例如图 2.3 中学习器  $C$  的  $BEP$  是 0.65, 而基于  $BEP$  的比较, 可认为学习器  $A$  **优于**  $B$ .

### 4 $F1$ 度量

但  $BEP$  还是过于简化了些, 更常用的是  $F1$  度量,  $F1$  的定义如下:

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{\text{样例总数} + TP + TN} \quad (2.10)$$

### 5 $F1$ 的一般形式 $F_\beta$

在一些实际应用中, 对**查准率**和**查全率**的重视程度有所不同. 因此引入  $F1$  度量的一般形式— $F_\beta$ , 能让我们表达出对查准率/查全率的不同偏好, 具体定义为:

$$F_\beta = \frac{(1 + \beta^2) \times P \times R}{(\beta^2 \times P) + R} \quad (2.11)$$

其中  $\beta > 0$  度量了**查全率**对**查准率**的**相对重要性**.  $\beta = 1$  时退化为标准的  $F1$ ;  $\beta > 1$  时, **查全率**有更大影响;  $\beta < 1$  时, **查准率**有更大影响.

## 6 全局性能度量

如何在  $n$  个二分类混淆矩阵上**综合考察查准率和查全率**

**宏查准率和宏查全率**

先在各混淆矩阵上**分别计算出查准率和查全率**, 记为  $(P_1, R_1), (P_2, R_2), \dots, (P_n, R_n)$ , 再**计算平均值**, 这样就得到"**宏查准率**" ( $macro-P$ )、"**宏查全率**" ( $macro-R$ ), 以及相应的"**宏  $F1$** " ( $macro-F1$ ):

$$macro-P = \frac{1}{n} \sum_{i=1}^n P_i \quad (2.12)$$

$$macro-R = \frac{1}{n} \sum_{i=1}^n R_i \quad (2.13)$$

$$macro-F1 = \frac{2 \times macro-P \times macro-R}{macro-P + macro-R} \quad (2.14)$$

**微查准率和微查全率**

先将各混淆矩阵的对应元素进行**平均**, 得到  $TP, FP, TN, FN$  的平均值, 分别记为  $\overline{TP}, \overline{FP}, \overline{TN}, \overline{FN}$ , 再基于这些平均值计算出"**微查准率**" ( $micro-P$ )、"**微查全率**" ( $micro-R$ ) 和"**微  $F1$** " ( $micro-F1$ )

$$micro-P = \frac{\overline{TP}}{\overline{TP} + \overline{FP}} \quad (2.15)$$



$$\text{micro}-R = \frac{\overline{TP}}{\overline{TP} + \overline{FN}} \quad (2.16)$$

$$\text{micro}-F1 = \frac{2 \times \text{micro}-P \times \text{micro}-R}{\text{micro}-P + \text{micro}-R} \quad (2.17)$$

小结:

先定义查准率  $P$  和查全率  $R$ , 接着引入  $P-R$  曲线, 如果完全包住, 这包住的为更优学习器, 其他情况这计算面积; 但  $P-R$  曲线中计算面积不易求得, 接着引入比较"平衡点"(BEP), 即查准率=查全率的点; 但平衡点也过于简单, 于是引入  $F1$ ; 但  $F1$  中, 查准率与查全率的地位相同的, 为了在实际使用中对查准率或查全率有所侧重, 引入  $F1$  的一般形式  $F_\beta$ ; 最后, 对于多个混淆矩阵, 如何计算全局的  $F1$ , 分为宏和微.

即 查准率  $P$  和查全率  $R \rightarrow P-R$  曲线  $\rightarrow$  "平衡点"(BEP)  $\rightarrow F1 \rightarrow F_\beta$

### (三) ROC 与 AUC

#### 1 ROC 曲线

很多学习器是为测试样本产生一个实值或概率预测, 然后将这个预测值与一个分类阈值 (threshold) 进行比较, 若大于阈值则分为正类, 否则为反类. 这个实值或概率预测结果的好坏, 直接决定了学习器的泛化能力.

根据这个实值或概率预测结果, 我们可将测试样本进行排序, "最可能"是正例的排在最前面, "最不可能"是正例的排在最后面. 这样, 分类过程就相当于在这个排序中以某个"截断点" (cut point) 将样本分为两部分, 前一部分判作正例, 后一部分则判作反例.

可根据任务需求来采用不同的截断点, 若我们更重视"查准率", 则可选择排序中靠前的位置进行截断; 若更重视"查全率", 则可选择靠后的位置进行截断. 因此, 排序本身的质量好坏, 体现了综合考虑学习器在不同任务下的"期望泛化性能"的好坏, 或者说"一般情况下"泛化性能的好坏.

ROC 全称是"受试者工作特征" (Receiver Operating Characteristic) 曲线. 与  $P-R$  曲线相似, 我们根据学习器的预测结果对样例进行排序, 按此顺序逐个把样本作为正例进行预测, 每次计算出两个重要量的值, 分别以它们为横、纵坐标作图, 就得到了"ROC 曲线". 与  $P-R$  曲线使用查准率、查全率为纵、横轴不同, ROC 曲线的纵轴是"真正例率" (True Positive Rate, 简称  $TPR$ ), 横轴是"假正例率" (False PositiveRate, 简称  $FPR$ ), 基于表 2.1 中的符号, 两者分别定义为:

$$TPR = \frac{TP}{TP + FN} \quad (2.18)$$

$$FPR = \frac{FP}{TN + FP} \quad (2.19)$$

显示 ROC 曲线的图称为"ROC 图". 如图 2.4(a)



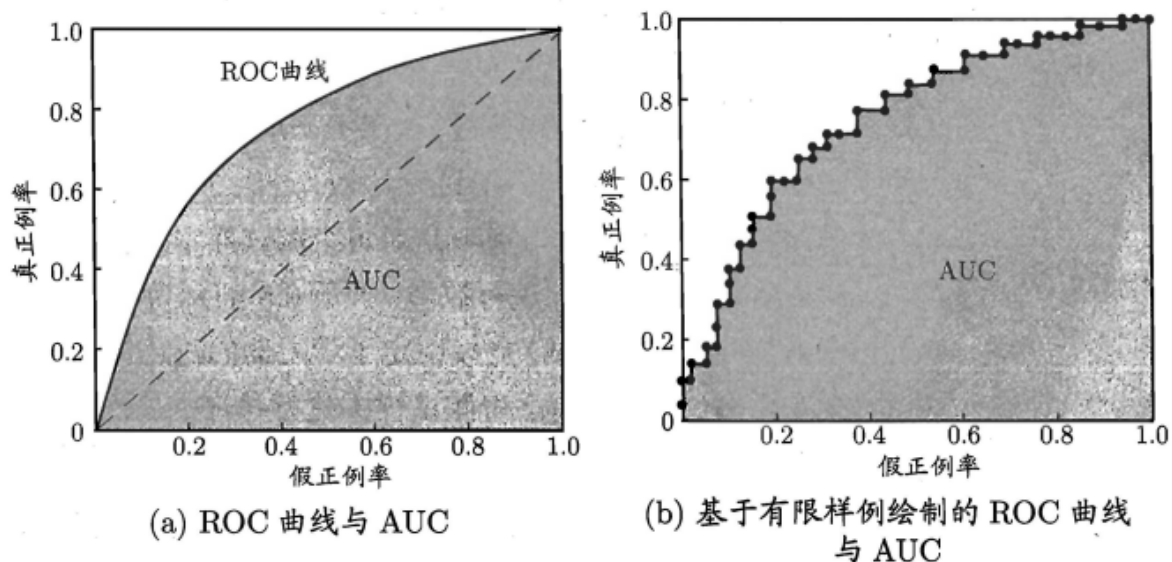


图 2.4 ROC 曲线与 AUC 示意图

现实任务中通常是利用有限个测试样例来绘制 ROC 图, 此时仅能获得有限个 (真正例率, 假正例率) 坐标对, 无法产生图 2.4(a) 中的光滑 ROC 曲线, 只能绘制出如图 2.4(b) 所示的近似 ROC 曲线。

## 2 AUC 曲线

进行学习器的比较时, 与  $P-R$  图相似, 若一个学习器的 ROC 曲线被另一个学习器的曲线完全“包住”, 则可断言后者的性能优于前者; 若两个学习器的 ROC 曲线发生交叉, 则难以一般性地断言两者孰优孰劣。此时如果一定要进行比较, 则较为合理的判据是比较 ROC 曲线下的面积, 即 AUC (Area Under ROC Curve), 如图 2.4 所示

**AUC 的计算方法:**

AUC 可通过对 ROC 曲线下各部分的面积求和而得。假定 ROC 曲线是由坐标为  $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$  的点按序连接而形成 ( $x_1 = 0, x_m = 1$ ), 参见图 2.4(b), 则 AUC 可估算为:

$$AUC = \frac{1}{2} \sum_{i=1}^{m-1} (x_{i+1} - x_i) \cdot (y_i + y_{i+1}) \quad (2.20)$$

**排序损失:**

AUC 考虑的是样本预测的排序质量, 因此它与排序误差有紧密联系。给定  $m^+$  个正例和  $m^-$  个反例, 令  $D^+$  和  $D^-$  分别表示正、反例集合, 则排序“损失” (loss) 定义为:

$$\ell_{\text{rank}} = \frac{1}{m^+ m^-} \sum_{x^+ \in D^+} \sum_{x^- \in D^-} \left( \mathbb{I}(f(x^+) < f(x^-)) + \frac{1}{2} \mathbb{I}(f(x^+) = f(x^-)) \right) \quad (2.21)$$

即考虑每一对正、反例, 若正例的预测值小于反例, 则记一个“罚分”, 若相等, 则记 0.5 个“罚分”。

$\ell_{\text{rank}}$  对应的是 ROC 曲线之上的面积:

$$AUC = 1 - \ell_{\text{rank}} \quad (2.22)$$

**注1:** 公式 (2.22) 不是很明白, 暂放。

## (四) 代价敏感错误率与代价曲线

现实任务中, 会有这样的情况: 不同类型的错误所造成的后果不同.

为权衡不同类型错误所造成的不同损失, 可为错误赋予"非均等代价" (unequal cost).

以二分类任务为例, 我们可根据任务的领域知识设定一个"代价矩阵" (cost matrix), 如表 2.2 所示, 其中  $\text{cost}_{ij}$  表示将第  $i$  类样本预测为第  $j$  类样本的代价. 一般来说,  $\text{cost}_{ii} = 0$ ; 若将第 0 类判别为第 1 类所造成的损失更大, 则  $\text{cost}_{01} > \text{cost}_{10}$ ; 损失程度相差越大,  $\text{cost}_{01}$  与  $\text{cost}_{10}$  值的差别越大.

表 2.2 二分类代价矩阵

真实类别	预测类别	
	第 0 类	第 1 类
第 0 类	0	$\text{cost}_{01}$
第 1 类	$\text{cost}_{10}$	0

前面介绍的性能度量, 都隐式地假设了均等代价. 在非均等代价下, 我们所希望的不再是简单地最小化错误次数, 而是希望最小化"总体代价" (total cost).

若将表 2.2 中的第 0 类作为正类、第 1 类作为反类, 令  $D^+$  与  $D^-$  分别代表样例集  $D$  的正例子集和反例子集, 则"代价敏感" (cost-sensitive) 错误率为

$$E(f; D; \text{cost}) = \frac{1}{m} \left( \sum_{x_i \in D^+} \mathbb{I}(f(x_i) \neq y_i) \times \text{cost}_{01} + \sum_{x_i \in D^-} \mathbb{I}(f(x_i) \neq y_i) \times \text{cost}_{10} \right) \quad (2.23)$$

代价曲线:

在非均等代价下,  $ROC$  曲线不能直接反映出学习器的期望总体代价, 而"代价曲线" (cost curve) 则可达到该目的. 代价曲线图的横轴是取值为  $[0, 1]$  的正例概率代价

$$P(+)\text{cost} = \frac{p \times \text{cost}_{01}}{p \times \text{cost}_{01} + (1 - p) \times \text{cost}_{10}} \quad (2.24)$$

其中  $p$  是样例为正例的概率

纵轴是取值为  $[0, 1]$  的归一化代价

$$\text{cost}_{\text{norm}} = \frac{\text{FNR} \times p \times \text{cost}_{01} + \text{FPR} \times (1 - p) \times \text{cost}_{10}}{p \times \text{cost}_{01} + (1 - p) \times \text{cost}_{10}} \quad (2.25)$$

其中  $FPR$  是式 (2.19) 定义的假正例率,  $FNR = 1 - TPR$  是假反例率

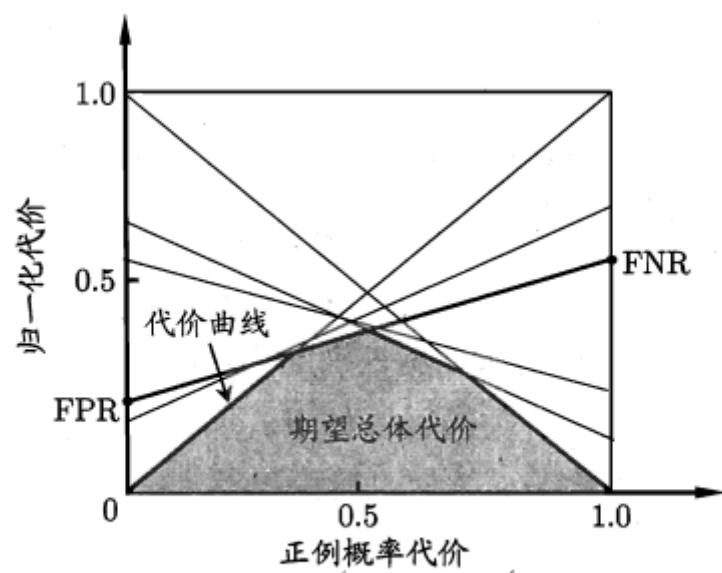


图 2.5 代价曲线与期望总体代价