第2章 模型评估与选择

中国矿业大学 计算机科学与技术学院

2.1 经验误差与过拟合

■**错误率(error rate)**：m个样本中有a个样本分错，则错误率为a/m。

$$err = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq f(\boldsymbol{x}_i))$$

■**精度(accuracy)**、又称**准确率**： 1-a/m，即“精度 =1-错误率”。

$$acc = \frac{1}{N}\sum_{i=1}^{N} I(y_i = f(\boldsymbol{x}_i))$$

$$err + acc = 1$$

2.1 经验误差与过拟合

- **误差(error)**：学习器实际预测输出与样本真实输出间的差异。
- **经验误差(empirical error)**：在训练集上的误差，又称“训练误差”（training error）
- **泛化误差(generalization error)**：在“未来”样本上的误差。

□泛化误差越小越好

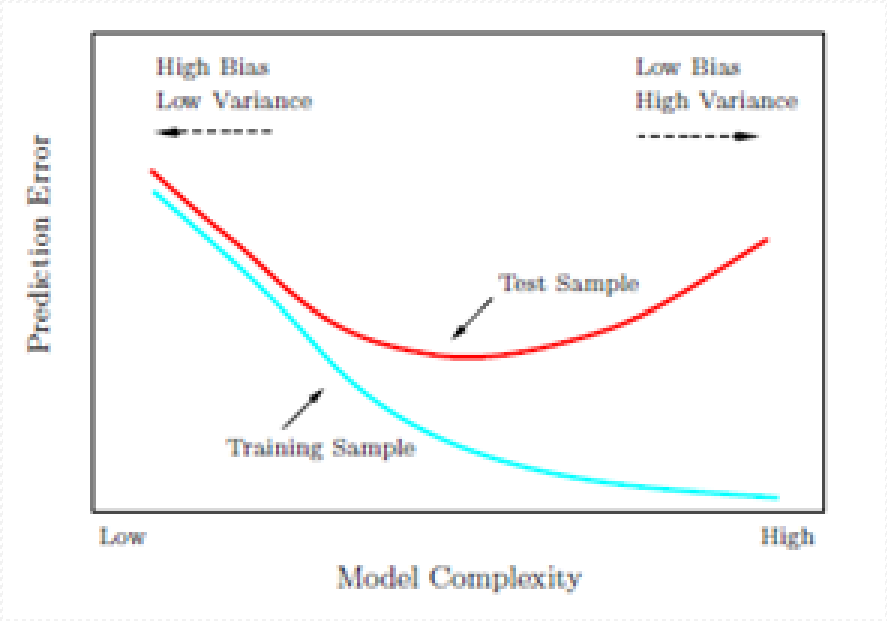
□经验误差是否越小越好?

NO! 因为会出现“过拟合”(overfitting)

过拟合 VS 欠拟合

过拟合(overfitting)：当学习器把训练样本学得“太好”了的时候，很可能已经把训练样本自身的一些特点当作了所有潜在样本都会具有的一般性质，这样就会导致泛化性能下降。

欠拟合(underfitting)：对训练样本的一般性质尚未学好。



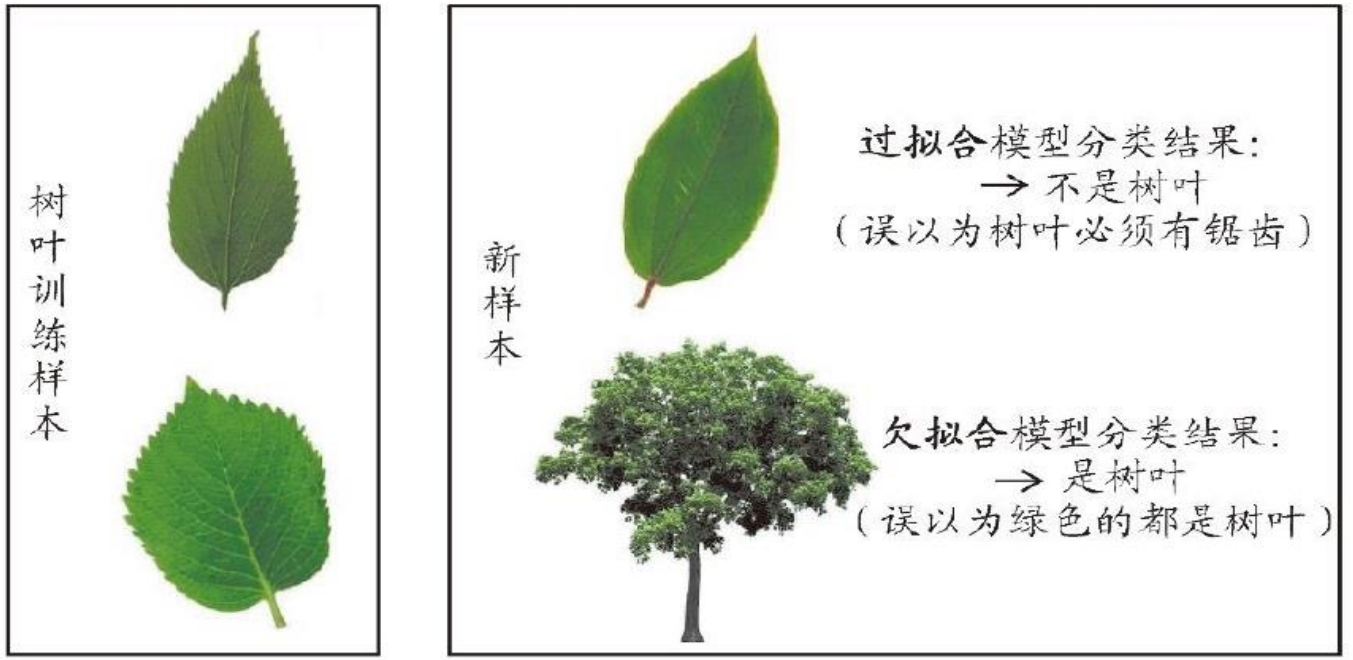
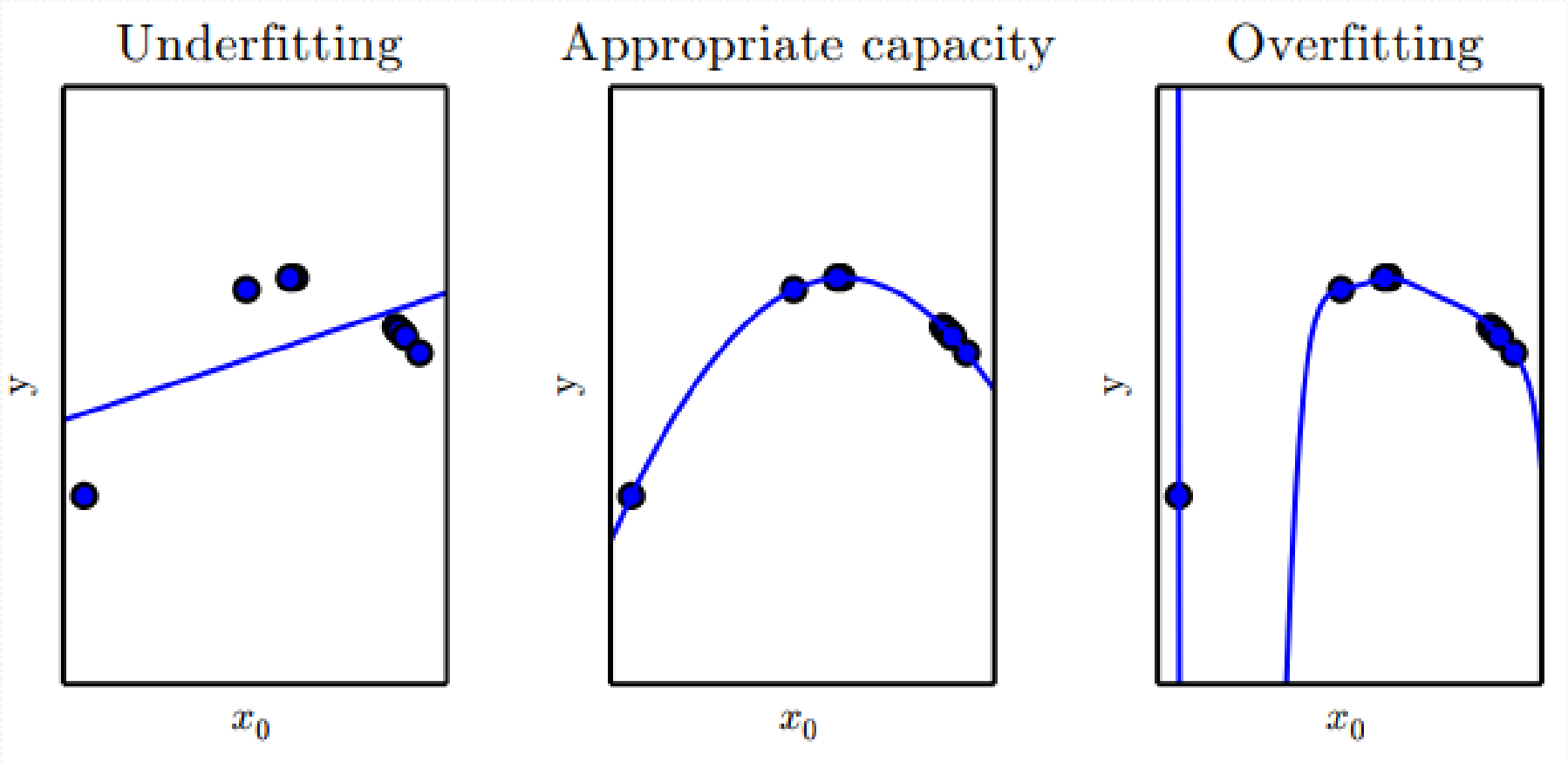


图 2.1 过拟合、欠拟合的直观类比

过拟合 VS 欠拟合



$\hat{y} = b + \omega x$

$\hat{y} = b + \omega_1 x + \omega x^2$

$\hat{y} = b + \sum_{i=1}^{9} \omega_i x^i$

模型选择(model selection)

三个关键问题:

- 如何获得测试结果? **评估方法**
- 如何评估性能优劣? **性能度量**
- 如何判断实质差别? **比较检验**

2.2 评估方法

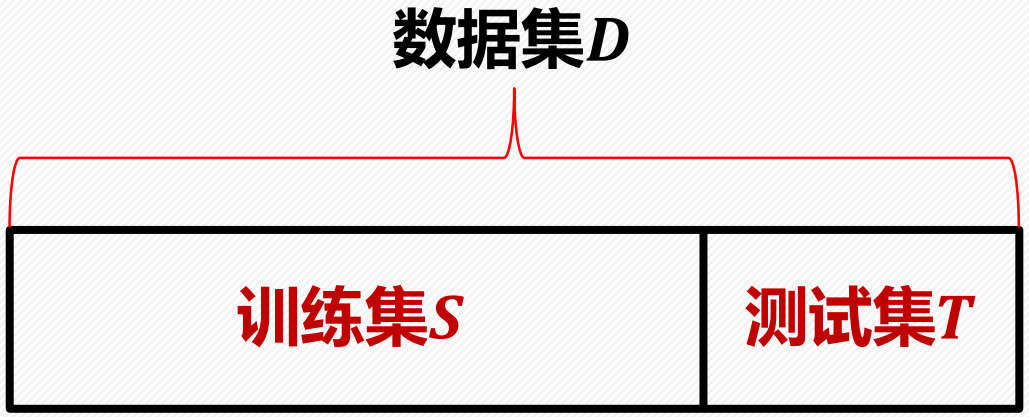
关键：怎么获得“测试集”(test set)？

测试集应该与训练集“互斥”

常见方法：

- 留出法 (hold-out)
- 交叉验证法 (cross validation)
- 自助法 (bootstrap)

2.2.1 留出法(hold-out)



$$D = S \cup T, S \cap T = \emptyset$$

以二分类任务为例：假定D中有1000个样本，将其划分为S包含700个样本，T包含300个样本。用S进行训练后，如果模型有90个样本分类错误。
错误率=(90/300)=30%
精度=1-30%=70%

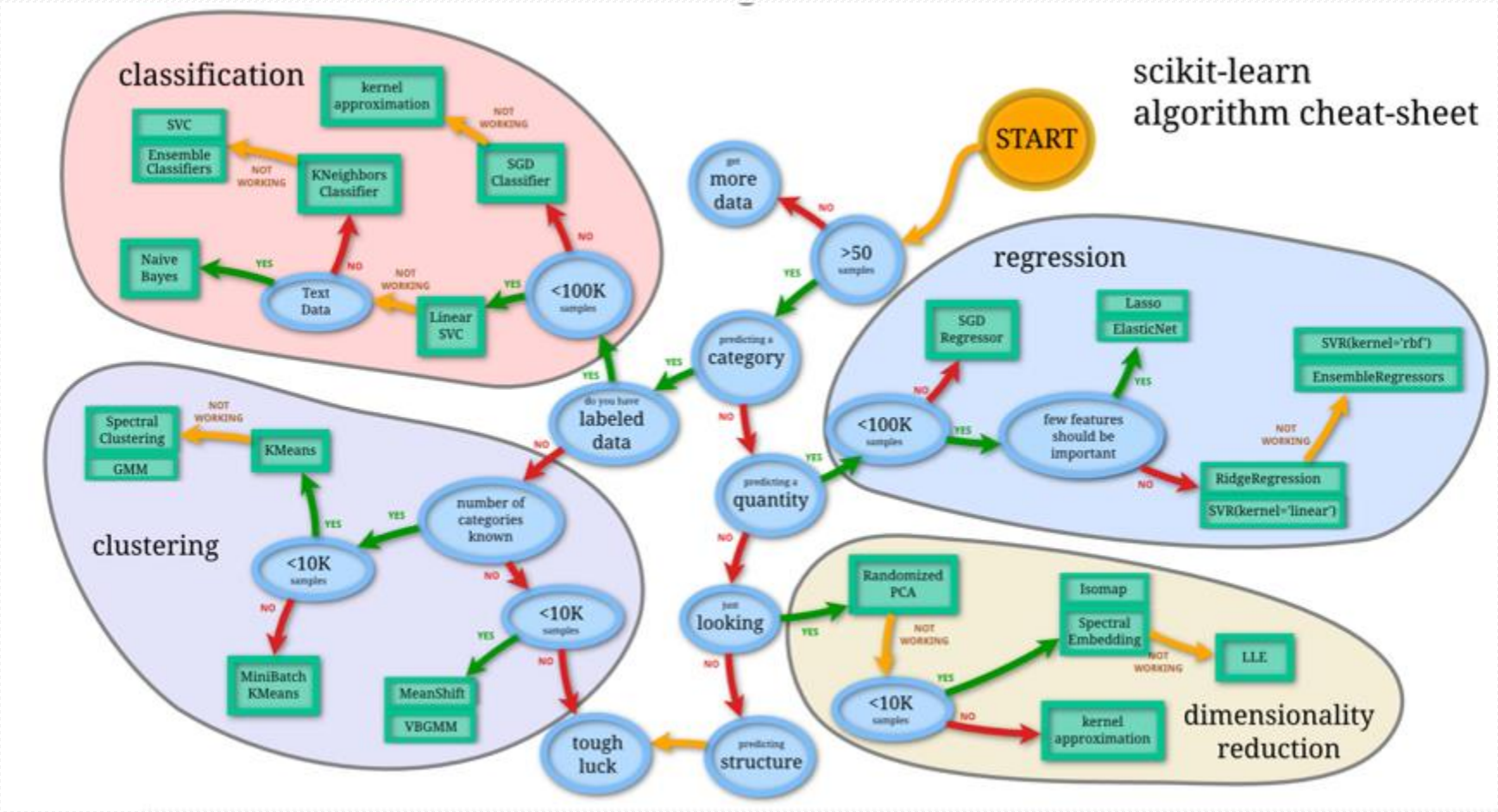
注意：

- 保持数据分布一致性，避免因训练/测试**数据分布的差异**而产生偏差
- **多次随机划分、重复进行实验评估**后取平均值作为留出法的评估结果
- **测试集不能太大、也不能太小** (例如：1/5~1/3)

Scikit-learn简介

https://scikit-learn.org/stable/user_guide.html

- **Scikit-learn(sklearn) is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities.**



sklearn的train_test_split

- **train_test_split函数**：将样本集随机划分为训练集和测试集，并返回划分好的训练集数据、测试集数据、训练集标签、测试集标签。
- **格式**：X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=None)
- **形式参数**
  - X：待划分的样本（集合）特征数据。
  - y：待划分的样本（集合）标签。
  - test_size：实数表示样本占比，整数则是样本数量。
  - random_state：随机数种子。
- **返回值**
  - X_train：划分出的训练集征数据。
  - X_test：划分出的测试集征数据。
  - y_train：划分出的训练集标签。
  - y_test：划分出的测试集标签。

例2.1，留出法。

```
from sklearn.model_selection import train_test_split

X, y = np.arange(10).reshape((5, 2)), range(5)
print("X:", X)
print("y:", list(y))

X_train, X_test, y_train, y_test=train_test_split(X, y,
test_size=0.4, random_state=10)
print("X_train:", X_train)
print("X_test:", X_test)
print("y_train:", y_train)
print("y_test:", y_test)
```

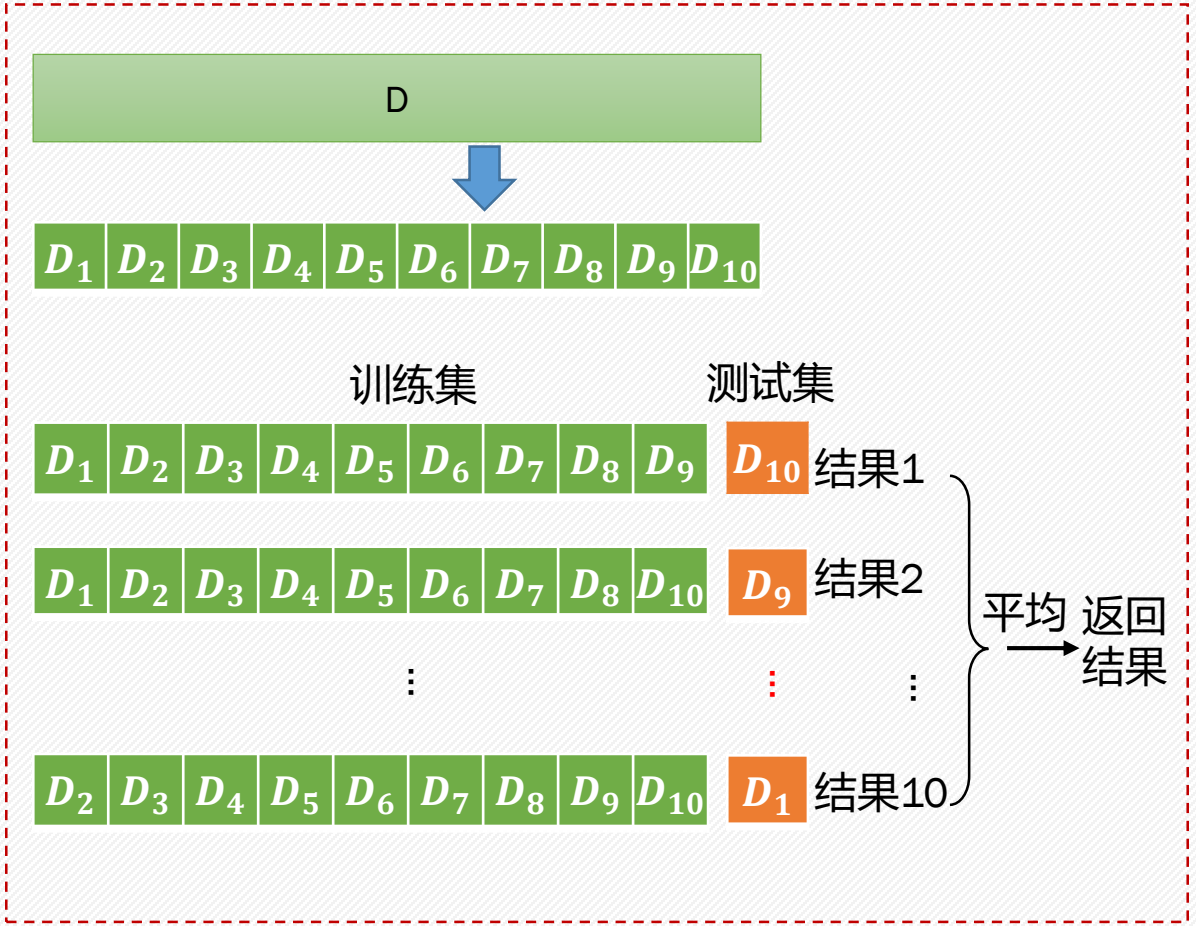
```
X: [[0 1]
    [2 3]
    [4 5]
    [6 7]
    [8 9]]
y: [0, 1, 2, 3, 4]
X_train: [[0 1]
          [8 9]
          [2 3]]
X_test: [[4 5]
         [6 7]]
y_train: [0, 4, 1]
y_test: [2, 3]
```

2.2.2 交叉验证法

■“**交叉验证法**”(**cross validation**)：先将数据集D划分为k个大小相似的互斥子集，即$D = D_1 \cup D_2 \cup \cdots D_k, D_i \cap D_j = \emptyset (i \neq j)$。每个子集$D_i$都尽可能保持数据分布的一致性，即从$D$中通过分层采样得到. 然后，每次用$k-1$个子集的并集作为训练集，余下的那个子集作为测试集；这样就可获得k组训练/测试集，从而可进行k次训练和测试，最终返回的是这k个测试结果的均值。

■k最常用的取值是**10**，此时称为**10**折交叉验证。

2.2.2 交叉验证法


D
D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_{10}
训练集
测试集
D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_{10} 结果1
D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_{10} D_9 结果2
D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_{10} D_1 结果10
平均
返回结果

例2.2，k折交叉验证

sklearn.model_selection.cross_validate(***estimator***, *X*, *y=None*, **, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', return_train_score=False, return_estimator=False, error_score=nan*)

```
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_validate
diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
lasso = linear_model.Lasso()
cv_results = cross_validate(lasso, X, y, cv=3)
```

```
{'fit_time': array([0.0009985 , 0.00099897, 0.00199795]),
 'score_time': array([0.0010004 , 0.00100064, 0.00100183]),
 'test_score': array([0.33150734, 0.08022311, 0.03531764])}
```

sklearn.model_selection.cross_val_score(*estimator*, *X*, *y=None*, **, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan*)

例2.3，k折交叉验证

StratifiedKFold

■**KFold(n_splits=5, shuffle=False, random_state=None)**

■**参数**

- **n_splits：** 要划分的折数。
- **shuffle：** 表示是否打乱划分，默认False，即不打乱。
- **random_state：** 随机数种子。

```
from sklearn.model_selection import KFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1, 2, 2])
kf = KFold(n_splits=3)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [2 3 4 5] TEST: [0 1]
TRAIN: [0 1 4 5] TEST: [2 3]
TRAIN: [0 1 2 3] TEST: [4 5]
```

例2.4，重复p次k折交差验证

RepeatedStratifiedKFold

■**RepeatedKFold(n_splits=5, n_repeats=2, random_state=None)**

■**参数**

- **n_splits：** 几折，划分为几等分。
- **n_repeats：** 重复的次数，即p次K折的p。
- **random_state ：** 随机数种子

```
from sklearn.model_selection import RepeatedKFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 0, 1, 1, 1])
rkf = RepeatedKFold(n_splits=3, n_repeats=2, random_state=18)
for train_index, test_index in rkf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
```

```
TRAIN: [0 1 2 3] TEST: [4 5]
TRAIN: [2 3 4 5] TEST: [0 1]
TRAIN: [0 1 4 5] TEST: [2 3]
TRAIN: [0 1 2 4] TEST: [3 5]
TRAIN: [1 2 3 5] TEST: [0 4]
TRAIN: [0 3 4 5] TEST: [1 2]
```

例2.5， 分层k折交叉验证

```
from sklearn.model_selection import KFold, StratifiedKFold

X=np.array([[1,2,3,4],  [11,12,13,14],  [21,22,23,24],  [31,32,33,34],
[41,42,43,44],  [51,52,53,54],  [61,62,63,64], [71,72,73,74]])
y=np.array([1,1,0,0,1,1,0,0])

floder = KFold(n_splits=4, shuffle=False) #k-fold
sfolder = StratifiedKFold(n_splits=4,shuffle=False) #stratified k-fold

for train_idx, test_idx in floder.split(X, y):
    print('Train: %s | test: %s' % (train_idx, test_idx))
print()

for train_idx, test_idx in sfolder.split(X, y):
    print('Train: %s | test: %s' % (train_idx, test_idx))
```

```
Train: [2 3 4 5 6 7] | test: [0 1]
Train: [0 1 4 5 6 7] | test: [2 3]
Train: [0 1 2 3 6 7] | test: [4 5]
Train: [0 1 2 3 4 5] | test: [6 7]

Train: [1 3 4 5 6 7] | test: [0 2]
Train: [0 2 4 5 6 7] | test: [1 3]
Train: [0 1 2 3 5 7] | test: [4 6]
Train: [0 1 2 3 4 6] | test: [5 7]
```

交叉验证法的特例“留一法”

■若$k = m$，则得到交叉验证法的特例“**留一法**” **(leave-one-out, LOO)**

■**优点**

- 不受随机样本划分方式的影响，因为m个样本只有唯一方式划分为m个子集——每个子集包含一个样本；
- 使用的训练集与初始数据集相比只少了一个样本，使得被实际评估的模型与期望评估的用D训练出的模型很相似。

■**缺陷**

- 在数据集较大时，训练m个模型的计算开销往往难以忍受(例如数据集包含一百万个样本，则需训练一百万个模型)，而这还是在未考虑算法调参的情况下。
- 估计结果也未必永远比其他评估方法准确。

例2.6 留一法

```
from sklearn.model_selection import LeaveOneOut
X = np.array([[4, 5], [6, 7], [8, 9]])
y = np.array([1, 2, 3])
loo = LeaveOneOut()

for train_index, test_index in loo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    print(X_train, X_test, y_train, y_test)
```

```
TRAIN: [1 2] TEST: [0]
[[6 7]
 [8 9]] [[4 5]] [2 3] [1]
TRAIN: [0 2] TEST: [1]
[[4 5]
 [8 9]] [[6 7]] [1 3] [2]
TRAIN: [0 1] TEST: [2]
[[4 5]
 [6 7]] [[8 9]] [1 2] [3]
```

Note: LeaveOneOut() is equivalent to KFold(n_splits=n) where n is the number of samples.

2.2.3 自助法

- **自助采样 (bootstrap sampling)**：又称**有放回采样/可重复采样**。给定包含m个样本的数据集D，对它采样产生数据集D'：每次随机从D中挑选一个样本，将其拷贝放入D'，然后再将该样本放回初始数据集D中，使得该样本在下次采样时仍有可能被采到；该过程重复执行m次，得到包含m个样本的数据集D'。
- **采样结果**：D中有一部分样本会在D'中多次出现，而另一部分样本不出现。

$$\lim_{m\to\infty}(1-\frac{1}{m})^m=\frac{1}{e}\approx 0.368$$

约有 36.8% 的样本不出现

- **划分**：将D'用作训练集，$D\backslash D'$用作测试集。
- **优点**：在数据集较小、难以有效划分训练/测试集时很有用。
- **缺点**：改变了初始数据集的分布，会引入估计偏差。

2.2.4 调参与最终模型

算法的参数：一般由人工设定，亦称“超参数”。

模型的参数：一般由学习（自动）确定。

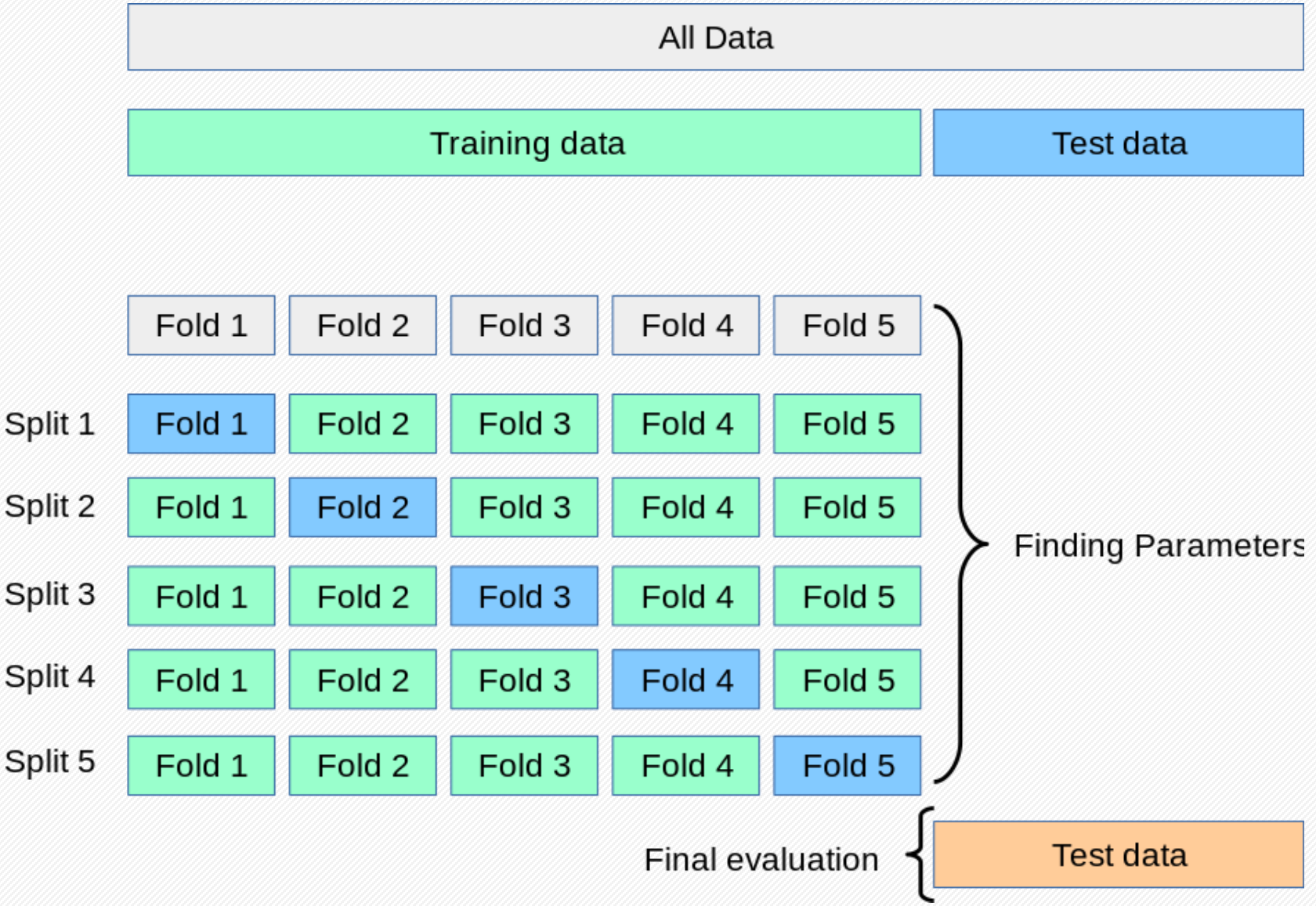
调参过程相似：先产生若干模型，然后基于某种评估方法进行选择

参数调得好不好对性能往往对最终性能有关键影响

区别：**训练集** vs. **测试集** vs. **验证集** (validation set)

算法参数选定后，要用“**训练集+验证集**”重新训练最终模型

2.2.4 调参与最终模型


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

2.3性能度量

性能度量(performance measure)：衡量模型泛化能力的评价标准。

使用不同的性能度量往往会导致不同的评判结果

要评估学习器f的性能，就要把学习器的预测结果$f(\boldsymbol{x}_i)$与真实标记y_i进行比较。

什么样的模型是“好”的，不仅取决于算法和数据，还取决于**任务需求**

- 回归(regression) 任务常用**均方误差**（mean squared error, MSE）

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

$$E(f;D)=\int_{\boldsymbol{x}\in D}(f(\boldsymbol{x})-y)^2 p(\boldsymbol{x})d\boldsymbol{x}$$

例2.7 均方误差（mean squared error）

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(f(\boldsymbol{x}_i) - y_i)^2$$

```
import numpy as np
from sklearn import metrics

y_true = np.array([1.0, 5.0, 4.0, 3.0, 2.0, 5.0, -3.0])
y_pred = np.array([1.0, 4.5, 3.5, 5.0, 8.0, 4.5, 1.0])

print(metrics.mean_squared_error(y_true, y_pred))
```

```
8.107142857142858
```

扩充

（1）均方根误差（Root Mean Squared Error, RMSE）

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f(\boldsymbol{x}_i) - y_i)^2}$$

```
import numpy as np
from sklearn import metrics
y_true = np.array([1.0, 5.0, 4.0, 3.0, 2.0, 5.0, -3.0])
y_pred = np.array([1.0, 4.5, 3.5, 5.0, 8.0, 4.5, 1.0])
print(np.sqrt(metrics.mean_squared_error(y_true, y_pred)))
```

```
2.847304489713536
```

（2）平均绝对误差（Mean Absolute Error, MAE）

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|f(\boldsymbol{x}_i) - y_i|$$

```
import numpy as np
from sklearn import metrics
y_true = np.array([1.0, 5.0, 4.0, 3.0, 2.0, 5.0, -3.0])
y_pred = np.array([1.0, 4.5, 3.5, 5.0, 8.0, 4.5, 1.0])
print(metrics.mean_absolute_error(y_true, y_pred))
```

```
1.9285714285714286
```

2.3.1 错误率和精度

- **错误率：** $E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i)\neq y_i)$

$$E(f;D)=\int_{\boldsymbol{x}\in D}\mathbb{I}(f(\boldsymbol{x})\neq y_i)p(\boldsymbol{x})d\boldsymbol{x}$$

- **精度（准确率）：**

$$\mathrm{acc}(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i)=y_i)=1-E(f;D)$$

$$\mathrm{acc}(f;D)=\int_{x\in D}\mathbb{I}(f(\boldsymbol{x})=y_i)p(\boldsymbol{x})d\boldsymbol{x}=1-E(f;D)$$

例2.9 精度（准确率）

```
from sklearn.metrics import accuracy_score

y_pred = [0, 2, 1, 3, 4]
y_true =  [0, 1, 2, 3, 4]
print(accuracy_score(y_true, y_pred))
print(accuracy_score(y_true, y_pred, normalize=False))
```

```
0.6
3
```

2.3.2 查准率、查全率和F1

表2.1 二分类结果混淆矩阵

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）

误报

漏报

- 查准率(precision)：$P = \frac{TP}{TP + FP}$ 宁愿漏掉，不可错过
- 查全率(recall)：$R = \frac{TP}{TP + FN}$ 宁愿错过，不可漏掉
- 准确率(accuracy)：$\mathrm{ACC} = \frac{TP + TN}{TP + FP + TN + FN}$

例2.10，混淆矩阵（confusion matrix）

```
from sklearn.metrics import confusion_matrix

y_true  = np.array([1, 0, 2, 0, 1, 0, 2, 0, 0, 2])
y_pred = np.array([1, 0, 1, 0, 0, 0, 2, 0, 2, 1])
print(confusion_matrix(y_true, y_pred, labels=[0,1,2]))
```

```
[[4 0 1]
 [1 1 0]
 [0 2 1]]
```

	类0	类1	类2
类0	4	0	1
类1	1	1	0
类2	0	2	1

P-R图, 平衡点BEP

根据预测结果按正例可能性大小对样例排序，并逐个把样本作为正例进行预测。

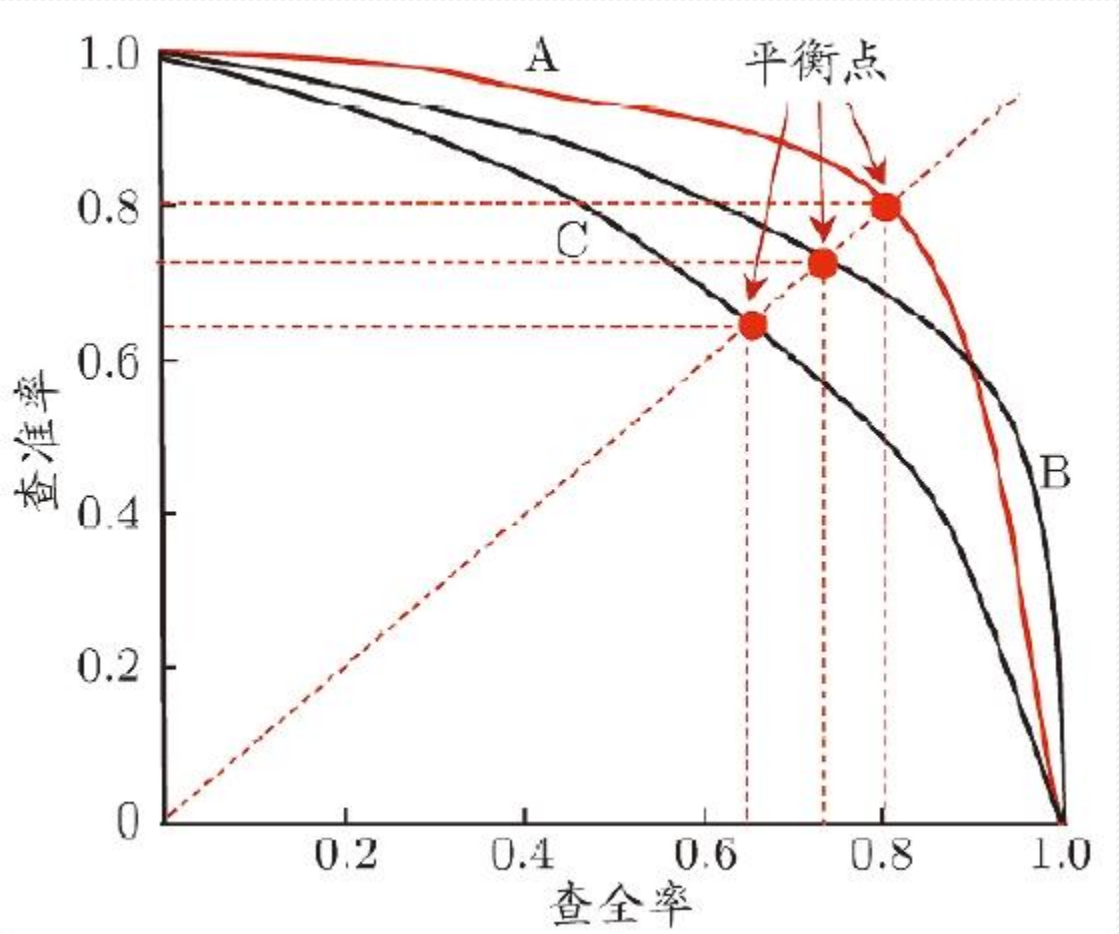


图2.3 P-R曲线与平衡点示意图

PR图：

- 学习器 A 优于 学习器 C
- 学习器 B 优于 学习器 C
- 学习器 A **??** 学习器 B

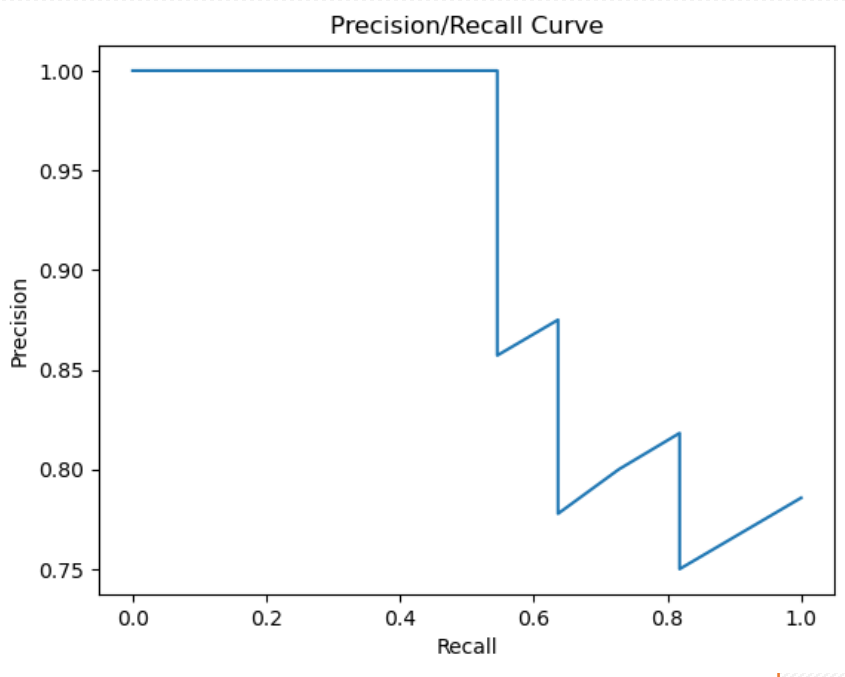
BEP(break even point, BEP)：

- 学习器 A 优于 学习器 B
- 学习器 A 优于 学习器 C
- 学习器 B 优于 学习器 C

例2.11 绘制P-R图

```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve


plt.figure()
plt.title('P-R Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
#y_true为样本实际的类别，y_scores为样本为正例的概率
y_true = np.array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0])
y_pred = np.array([0.9, 0.75, 0.86, 0.47, 0.55, 0.56, 0.74, 0.62, 0.5, 0.86, 0.8, 0.47,
0.44, 0.67, 0.43, 0.4, 0.52, 0.4, 0.35, 0.1])
precision, recall, thresholds = precision_recall_curve(y_true, y_pred)
plt.plot(recall, precision)
plt.show()
```



F1值

F1值：比 **BEP** 更常用。

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{\text{样例总数} + TP - TN}$$

$$\frac{1}{F1} = \frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)$$

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

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

若对查准率/查全率有不同偏好：

- **ß=1**时退化为标准的 $F1$
- **ß>1**时查全率有更大影响（更重要）
- **ß<1**时查准率有更大影响（更重要）

宏xx vs 微xx

得到多个混淆矩阵(例如多次训练/测试的结果，多分类的两两混淆矩阵)

■ 宏(macro-)查准率、查全率、F1

先在各混淆矩阵上分贝计算出查准率和查全率，记作P_i和R_i，再计算出平均值，得

$$\text{macro\_}P = \frac{1}{n}\sum_{i=1}^{n} P_i$$

$$\text{macro\_}R = \frac{1}{n}\sum_{i=1}^{n} R_i$$

$$macro\_F1 = \frac{2 \times macro\_P \times macro\_R}{macro\_P + macro\_R}$$

■ 微(micro-)查准率、查全率、F1

先将各混淆矩阵的对应元素平均，得到TP、FP、TN、FN的平均值，分别记为 $\overline{TP}$、$\overline{FP}$、$\overline{TN}$、$\overline{TN}$

$$micro\_P = \frac{\overline{TP}}{\overline{TP} + \overline{FP}}$$

$$micro\_R = \frac{\overline{TP}}{\overline{TP} + \overline{FN}}$$

$$micro\_F1 = \frac{2 \times \text{micro\_}P \times \text{micro\_}R}{\text{micro\_}P + \text{micro\_}R}$$

例2.12 计算查准率

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

$$\text{macro\_}P = \frac{1}{C}\sum_{i=1}^{C} P_i$$

$$micro\_P = \frac{\overline{TP}}{\overline{TP} + \overline{FP}}$$

```
from sklearn.metrics import precision_score
y_true =  [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
print(precision_score(y_true, y_pred, average='macro'))
print(precision_score(y_true, y_pred, average='micro'))
print(precision_score(y_true, y_pred, average='weighted'))
print(precision_score(y_true, y_pred, average=None))
```

	类0	类1	类2
类0	2	0	0
类1	1	0	1
类2	0	2	0

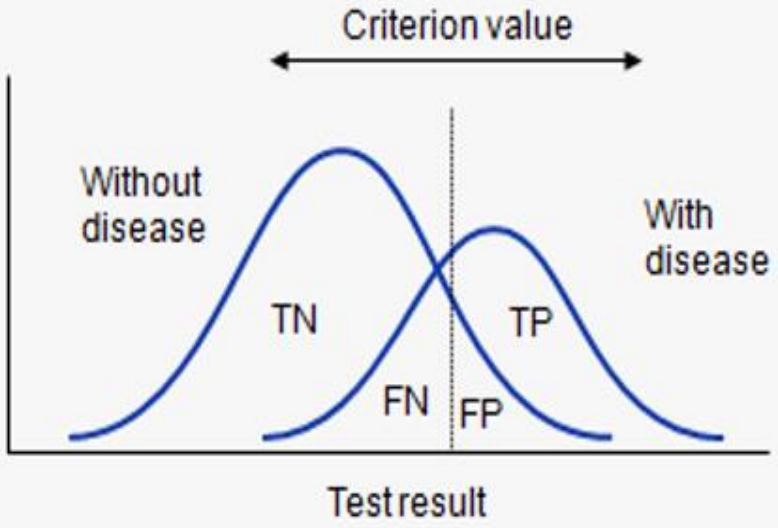
```
0.2222222222222222
0.3333333333333333
0.2222222222222222
[0.66666667 0.         0.        ]
```

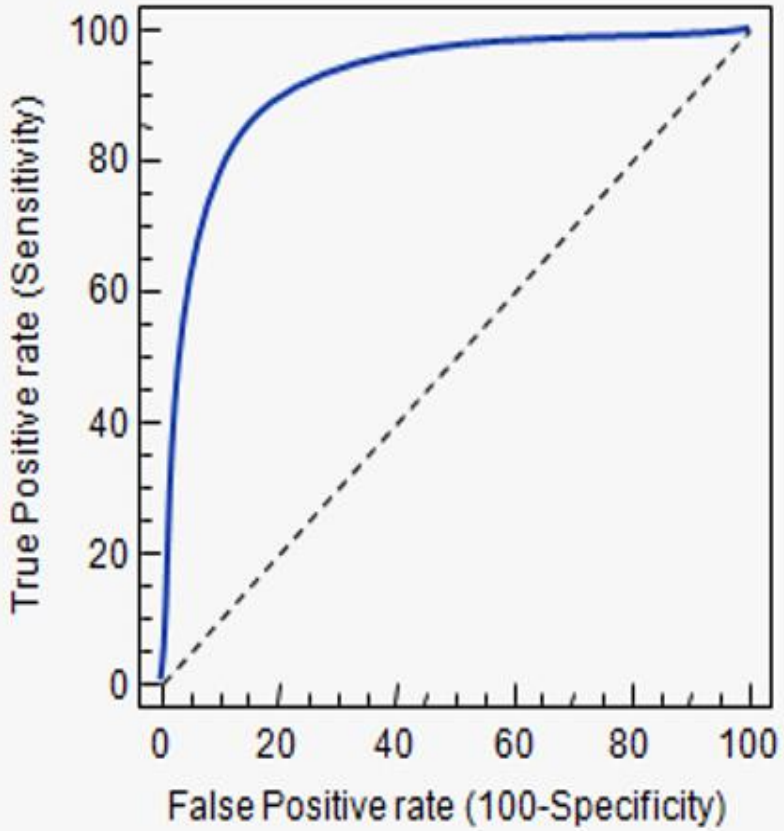
macro: (2/3+0/2+0/1)/3=0.222
micro: (2+0+0)/3/((2+0+0)/3+(1+2+1)/3)=0.333
weighted: (2/3×2/6+0/2×2/6+0/1×2/6)
None: 2/3, 0/2, 0/1

2.3 ROC和AUC

ROC (Receiver Operating Characteristic) Curve [Spackman, IWML'89]

AUC: Area Under the ROC Curve





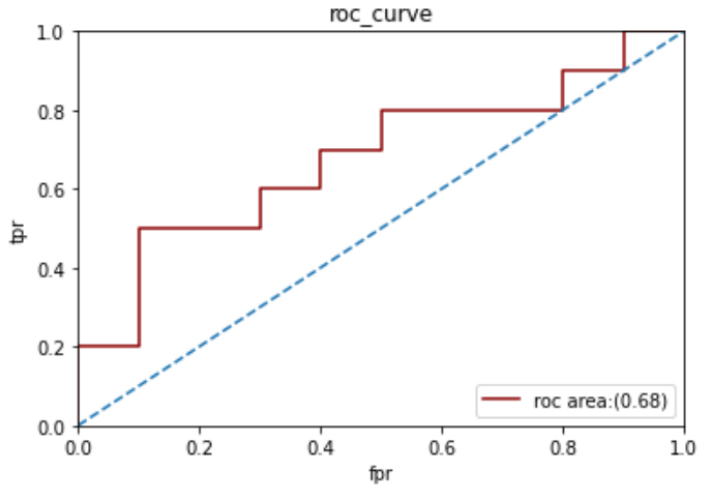
$$tpr = \frac{TP}{TP + FN} = \frac{TP}{m^+}$$

$$fpr = \frac{FP}{FP + TN} = \frac{FP}{m^-}$$

例2.13, ROC曲线

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_pred = np.array([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_test = np.array([1,1,0,1,1,1,0,0,1,0,1,0,1,0,0,0,1,0,1,0])
fpr, tpr, thre = roc_curve(y_test, y_pred)
auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color = 'darkred', label = 'roc area:(%0.2f)'%auc)
plt.plot([0, 1], [0, 1], linestyle = '--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('roc_curve')
plt.legend(loc = 'lower right')
```



2.3.4 代价敏感错误率与代价曲线

犯不同的错误往往会造成不同的损失

此时需考虑“非均等代价” (unequal cost)

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	$cost_{01}$
第1类	$cost_{10}$	1

■ 代价敏感(cost-sensitive)错误率

$$E(f;D;const)=\frac{1}{m}\left(\sum_{\boldsymbol{x}_i\in D^+}\mathbb{I}(f(\boldsymbol{x}_i)\neq y_i\times cost_{01})+\sum_{\boldsymbol{x}_i\in D^-}\mathbb{I}(f(\boldsymbol{x}_i)\neq y_i\times cost_{10})\right)$$

2.3.4 代价敏感错误率与代价曲线

■**假反例率**：FNR = 1 − TPR

■ROC 由线上每一点对应了代价平面上的一条线段，设ROC曲线上点的坐标为(TPR，FPR)，则可相应计算出FNR，然后在代价平面上绘制一条从 (0，FPR)到(1，FNR)的线段，线段下的面积即表示了该条件下的期望总体代价；如此将ROC曲线土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。

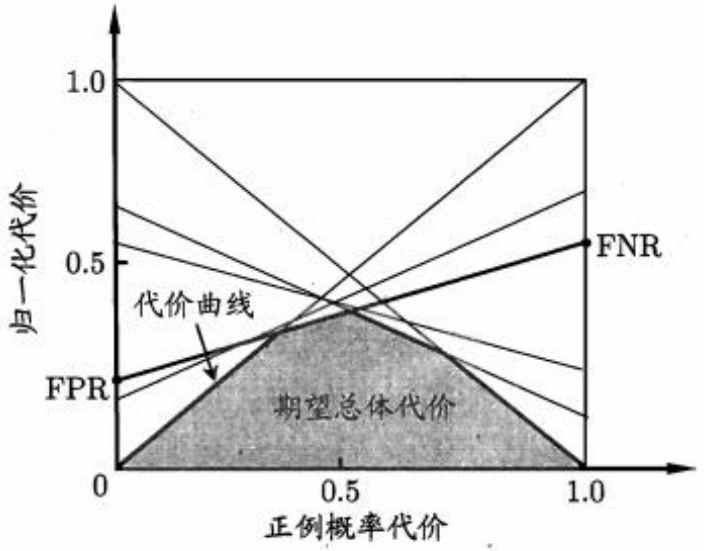
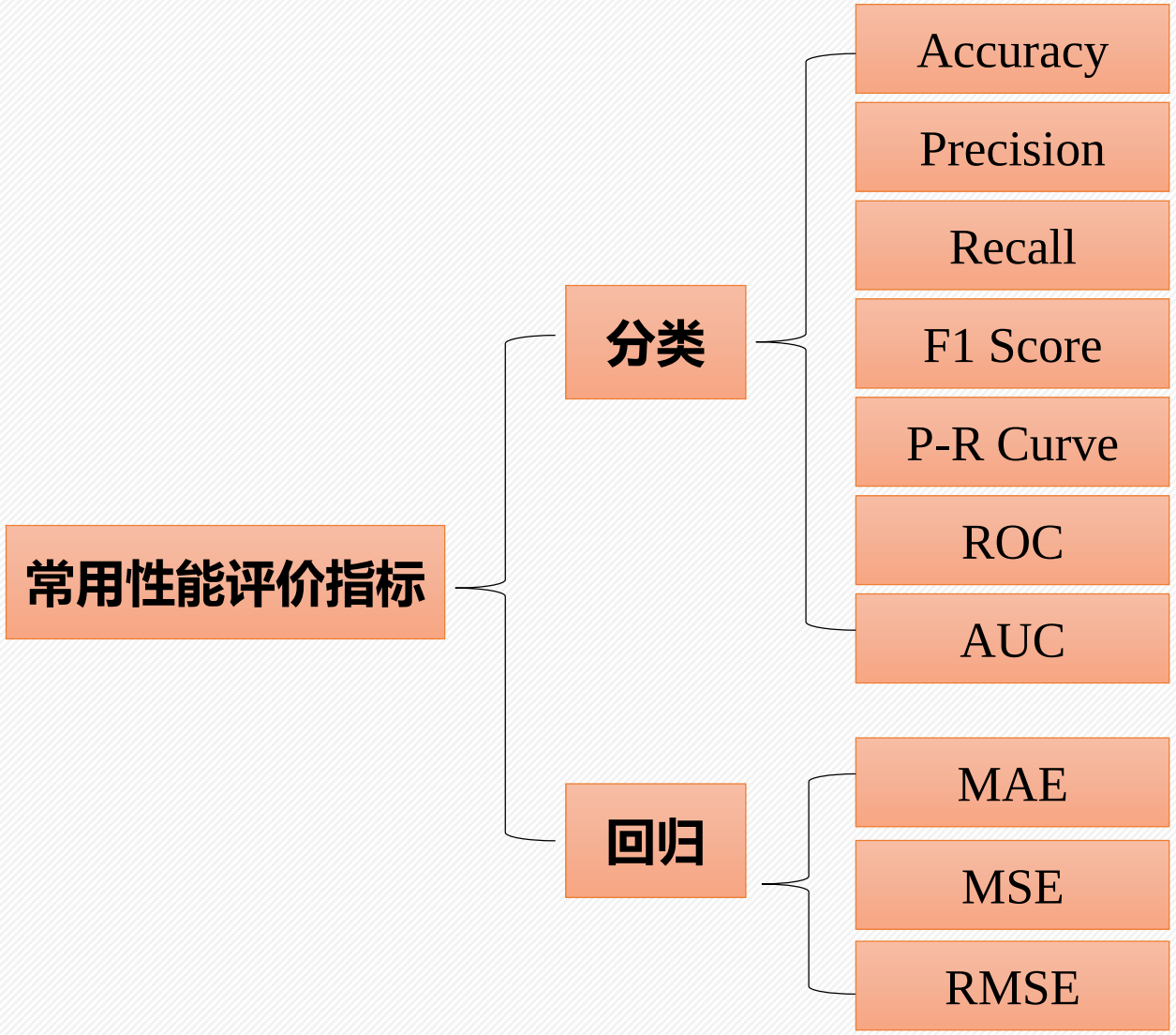


图 2.5 代价曲线与期望总体代价

常用的性能评价指标


常用性能评价指标
分类
Accuracy
Precision
Recall
F1 Score
P-R Curve
ROC
AUC
回归
MAE
MSE
RMSE

2.4 比较检验

在某种度量下取得评估结果后，是否可以直接比较以评判优劣？

不可以！因为：

- 希望比较泛化性能，但测试性能不等于泛化性能
- 测试性能随着测试集的变化而变化
- 很多机器学习算法本身有一定的随机性

统计假设检验（**hypothesis test**）：为学习器的性能比较提供了重要依据！基于假设检验结果可以推断出，若在测试集上观察到学习器**A**比**B**好，则**A**的泛化性能是否在统计意义上优于**B**，以及这个结论的把握有多大。

2.4.1 假设检验

■**泛化错误率**ϵ：学习器在一个样本上犯错的概率。

■**测试错误率**$\hat{\epsilon}$：m个测试样本中恰好有$m \times \hat{\epsilon}$个被误分。

■假定测试样本是独立采样而来，泛化错误率为ϵ的学习器将其中m'个样本误分，其余样本分类正确的概率是$\binom{m}{m'}\epsilon^{m'}(1-\epsilon)^{m-m'}$，由此估算出恰有$m \times \hat{\epsilon}$个样本误分类的概率，即在$m$个样本测试集上，泛化错误率为$\epsilon$的学习器被测得错误率为$\hat{\epsilon}$的概率：

$$P(\hat{\epsilon};\epsilon)=\binom{m}{m\times\hat{\epsilon}}\epsilon^{m\times\hat{\epsilon}}(1-\epsilon)^{m-m\times\hat{\epsilon}}$$

■给定测试错误率，则令$\frac{\partial P(\hat{\epsilon};\epsilon)}{\partial\epsilon}=0$，则有$P(\hat{\epsilon};\epsilon)$在$\epsilon=\hat{\epsilon}$时最大。

$\hat{\epsilon}$无偏估计证明

$$E(\hat{\epsilon}) = \sum_{i=0}^{m} \hat{\epsilon}\, P(\hat{\epsilon}, \epsilon)$$

$$= \sum_{i=0}^{m} \hat{\epsilon} \binom{m}{\hat{\epsilon} * m} \epsilon^{\hat{\epsilon}*m} (1-\epsilon)^{m-\hat{\epsilon}*m}$$

$$= \sum_{i=0}^{m} \frac{i}{m} \binom{m}{i} \epsilon^{i} (1-\epsilon)^{m-i}$$

$$= \sum_{i=1}^{m} \binom{m-1}{i-1} \epsilon \epsilon^{i-1} (1-\epsilon)^{m-i}$$

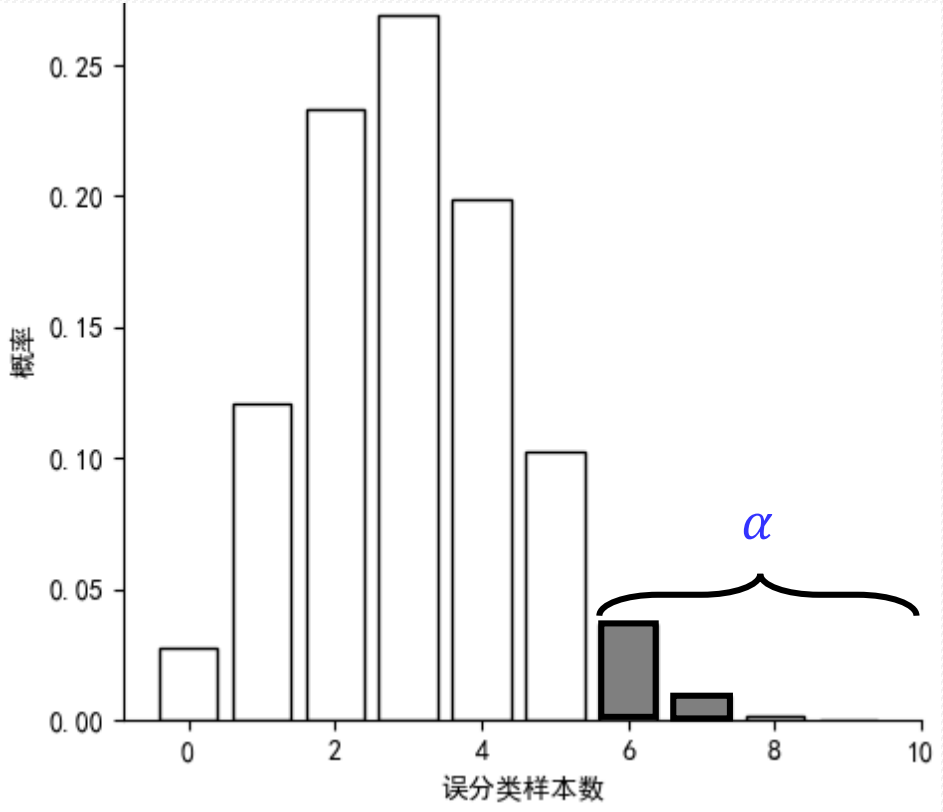
$$= \epsilon \sum_{i=1}^{m} \binom{m-1}{i-1} \epsilon^{i-1} (1-\epsilon)^{(m-1)-(i-1)}$$

$$= \epsilon(\epsilon + (1-\epsilon))^{m-1}$$

$$= \epsilon$$

∴ $\hat{\epsilon}$是ϵ的无偏估计。

二项检验（binomial test）



$$\bar{\epsilon} = \min \epsilon \quad \text{s.t.} \sum_{i=\epsilon_0 \times m+1}^{m} \binom{m}{i} \epsilon^i (1-\epsilon)^{m-i} < \alpha$$

t检验

■假定得到了k个测试错误率，$\hat{\epsilon}_1$，$\hat{\epsilon}_2$，…，$\hat{\epsilon}_k$，则：

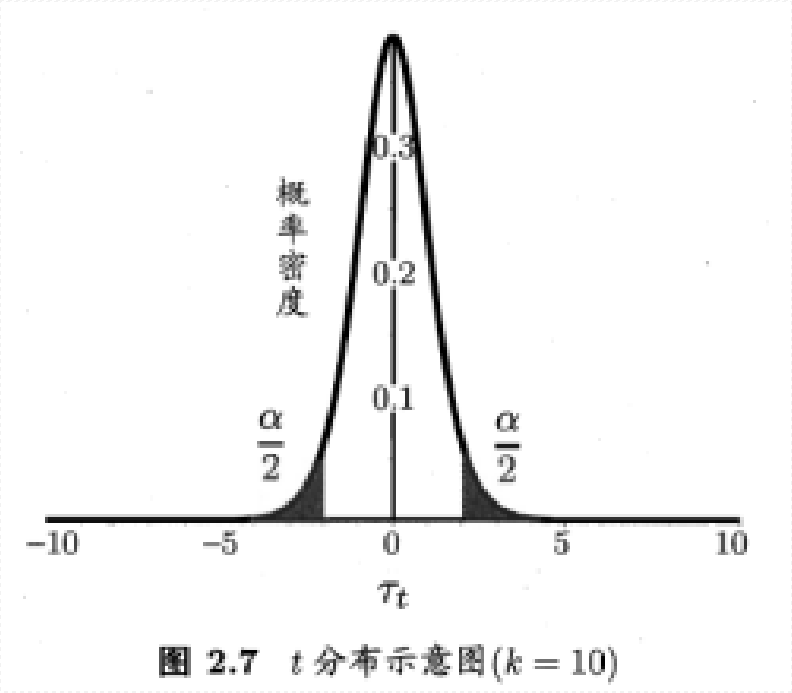


图 2.7 t 分布示意图($k = 10$)

$$\mu = \frac{1}{k}\sum_{i=1}^{k}\hat{\epsilon}_i$$

$$\sigma^2 = \frac{1}{k-1}\sum_{i=1}^{k}(\hat{\epsilon}_i - \mu)^2$$

$$T_t = \frac{\sqrt{k}(\mu - \epsilon_0)}{\sigma}$$

变量T_t服从自由度为$k-1$的t分布

表2.3 双边t检验的常用临界值

α	k				
	2	5	10	20	30
0.05	12.706	2.776	2.262	2.093	2.045
0.10	6.314	2.132	1.833	1.729	1.699

2.4.2 交叉验证t检验

■两个学习器A、B，使用k折交叉验证得到的测试错误率分别为ϵ_1^A，ϵ_2^A，…，ϵ_k^A和ϵ_1^B，ϵ_2^B，…，ϵ_k^B，其中ϵ_i^A和ϵ_i^B是在相同的第i折训练/测试集上得到的结果，可用k折交叉验证“**成对t检验**”来进行比较检验。

■k折交叉验证产生k对测试错误率，对每对结果求差$\Delta_i = \epsilon_i^A - \epsilon_i^B$；若性能相同则差值均值为0。用$\Delta_1$，$\Delta_2$，…，$\Delta_k$来对两个学习器性能相同做t检验，计算差值的均值$\mu$和方差$\sigma^2$。

$$T_t = \left| \frac{\sqrt{k}\mu}{\sigma} \right|$$

■若$T_t < t_{\frac{\alpha}{2},k-1}$，则认为两个学习器的性能没有显著差别；否则，可认为两个学习器性能有显著差别，错误平均率小的那个学习器性能较优。

5*2交叉验证

- 学习器A、B第i次2折交叉验证产生两对测试错误率，对它们分别求差，得到第1折上的差值Δ_i^1和第2折上的差值Δ_i^2。
- 为缓解测试错误率的非独立性，仅计算第一次2折交叉验证的结果平均值$\mu = 0.5(\Delta_1^1 + \Delta_1^2)$。
- 但对每次结果都计算出方差$\sigma_i^2 = (\Delta_i^1 - \frac{\Delta_i^1+\Delta_i^2}{2})^2 + (\Delta_i^2 - \frac{\Delta_i^1+\Delta_i^2}{2})^2$
- 变量$T_t = \frac{\mu}{\sqrt{\frac{1}{5}\sum_{i=1}^{5}\sigma_i^2}}$服从自由度为5的$t$分布，其双边检验的临界值$t_{\alpha/2,5}$。
- 当α=0.05时为2.5706；α=0.1时为2.0150。

2.4.3 McNemar检验

表2.4 两学习器分类差别列联表

算法B	算法A	
	正确	错误
正确	e_{00}	e_{01}
错误	e_{10}	e_{11}

假设两学习器性能相同，则应有$e_{01} = e_{10}$，因此变量$|e_{01} - e_{10}|$应服从正态分布。

McNemar检验

$$T_{\chi^2} = \frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}}$$

服从自由度为1的χ^2(卡方)分布，即标准正态分布变量的平方。给定显著度α，当以上变量值小于临界值时，认为两学习器性能没有显著差别；否则性能有显著差别。当α=0.05时为3.8415；α=0.1是为2.7055.

2.4.4 Friedman和Nemenyi检验

表2.5 算法比较序值表

数据集	算法 A	算法 B	算法 C
D_1	1	2	3
D_2	1	2.5	2.5
D_3	1	2	3
D_4	1	2	3
平均序值	1	2.125	2.875

均值$(k+1)/2$和**方差**$(k^2-1)/12$

■ Friedman检验

$$\mathcal{T}_{\chi^2} = \frac{k-1}{k} \cdot \frac{12N}{k^2-1} \sum_{i=1}^{k} (r_i - \frac{k+1}{2})^2$$

$$= \frac{12N}{k(k+1)} \left(\sum_{i=1}^{k} r_i^2 - \frac{k(k+1)^2}{4} \right)$$

$$\mathcal{T}_F = \frac{(N-1)\,\mathcal{T}_{\chi^2}}{N(k-1) - \mathcal{T}_{\chi^2}}$$

■ **Nemenyi检验**

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

表2.6 F检验的常用临界值

$\alpha = 0.05$									
数据集	算法个数 k								
个数 N	2	3	4	5	6	7	8	9	10
4	10.128	5.143	3.863	3.259	2.901	2.661	2.488	2.355	2.250
5	7.709	4.459	3.490	3.007	2.711	2.508	2.359	2.244	2.153
8	5.591	3.739	3.072	2.714	2.485	2.324	2.203	2.109	2.032
10	5.117	3.555	2.960	2.634	2.422	2.272	2.159	2.070	1.998
15	4.600	3.340	2.827	2.537	2.346	2.209	2.104	2.022	1.955
20	4.381	3.245	2.766	2.492	2.310	2.179	2.079	2.000	1.935

$\alpha = 0.1$									
数据集	算法个数 k								
个数 N	2	3	4	5	6	7	8	9	10
4	5.538	3.463	2.813	2.480	2.273	2.130	2.023	1.940	1.874
5	4.545	3.113	2.606	2.333	2.158	2.035	1.943	1.870	1.811
8	3.589	2.726	2.365	2.157	2.019	1.919	1.843	1.782	1.733
10	3.360	2.624	2.299	2.108	1.980	1.886	1.814	1.757	1.710
15	3.102	2.503	2.219	2.048	1.931	1.845	1.779	1.726	1.682
20	2.990	2.448	2.182	2.020	1.909	1.826	1.762	1.711	1.668

表2.7 Nemeyi检验中常用的q_α值

α	算法个数k								
	2	3	4	5	6	7	8	9	10
0.05	1.960	2.344	2.569	2.728	2.850	2.949	3.031	3.102	3.164
0.10	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.885	2.920

2.4.4 Friedman和Nemenyi检验

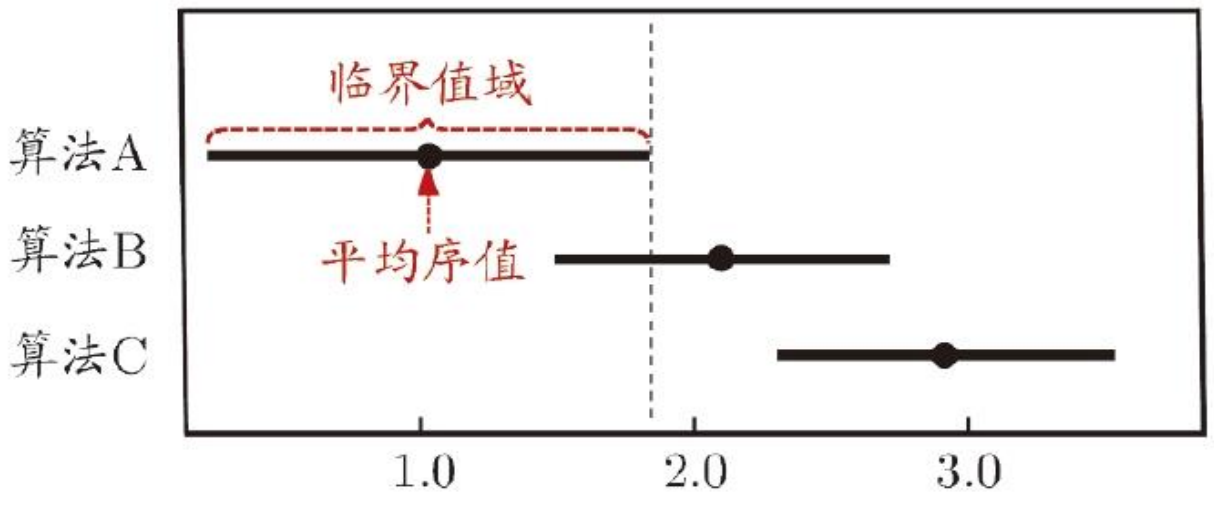


图 2.8 Friedman 检验图

数据集	算法 A	算法 B	算法 C
D_1	1	2	3
D_2	1	2.5	2.5
D_3	1	2	3
D_4	1	2	3
平均序值	1	2.125	2.875

纵轴显示各个算法，横轴为平均序值，每个算法圆点为其平均序值，线段为临界阈值的大小。

若两个算法有交叠 (A 和 B)，则说明没有显著差别。
否则有显著差别 (A 和 C)，算法 A 显著优于算法 C。

2.5 偏差与方差

对于测试样本$\boldsymbol{x}$，令y_D为$\boldsymbol{x}$在数据集中的标记（观测标记），y为$\boldsymbol{x}$的真实标记，$f(\boldsymbol{x}, D)$为训练集D上学得模型f在$\boldsymbol{x}$上的预测输出。

期望预测：$\bar{f}(\boldsymbol{x}) = E_D[f(\mathbf{x}; D)]$

方差：$var(x) = E_D[f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})]^2$

噪声：$\varepsilon^2 = E_D[(y_D - y)]^2$

偏差：$bias^2(x) = E_D(\bar{f}(\boldsymbol{x}) - y)^2$

期望泛化误差：

$$E(f; D) = E_D[(f(\boldsymbol{x}; D) - y_D)^2]$$

$$= E_D[(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x}) + \bar{f}(\boldsymbol{x}) - y_D)^2$$

$$= E_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + E_D\left[\left(\bar{f}(\boldsymbol{x}) - y_D\right)^2\right] + E_D[2(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x}))(\bar{f}(\boldsymbol{x}) - y_D)]$$

$$= E_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + E_D\left[\left(\bar{f}(\boldsymbol{x}) - y_D\right)^2\right]$$

$$= E_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + E_D\left[\left(\bar{f}(\boldsymbol{x}) - y + y - y_D\right)^2\right]$$

$$= E_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + E_D\left[\left(\bar{f}(\boldsymbol{x}) - y\right)^2\right] + E_D[(y - y_D)^2] + E_D[2(\bar{f}(\boldsymbol{x}) - y)(y - y_D)]$$

$$= E_D\left[\left(f(\boldsymbol{x}; D) - \bar{f}(\boldsymbol{x})\right)^2\right] + E_D\left[\left(\bar{f}(\boldsymbol{x}) - y\right)^2\right] + E_D[(y - y_D)^2]$$

偏差-方差分解(bias-variance decomposition)

对回归任务，期望泛化误差可通过“偏差-方差分解”拆解为：

$$E(f;D) = bias^2(\boldsymbol{x}) + var(\boldsymbol{x}) + \varepsilon^2$$

期望输出与真实结果的偏移程度

$$bias^2(\boldsymbol{x}) = E_D\left[\left(\bar{f}(\boldsymbol{x}) - y\right)^2\right]$$

同尺寸训练集变动，所导致的性能变化

$$var(\boldsymbol{x}) = E_D\left[\left(f(\boldsymbol{x};D) - \bar{f}(\boldsymbol{x})\right)^2\right]$$

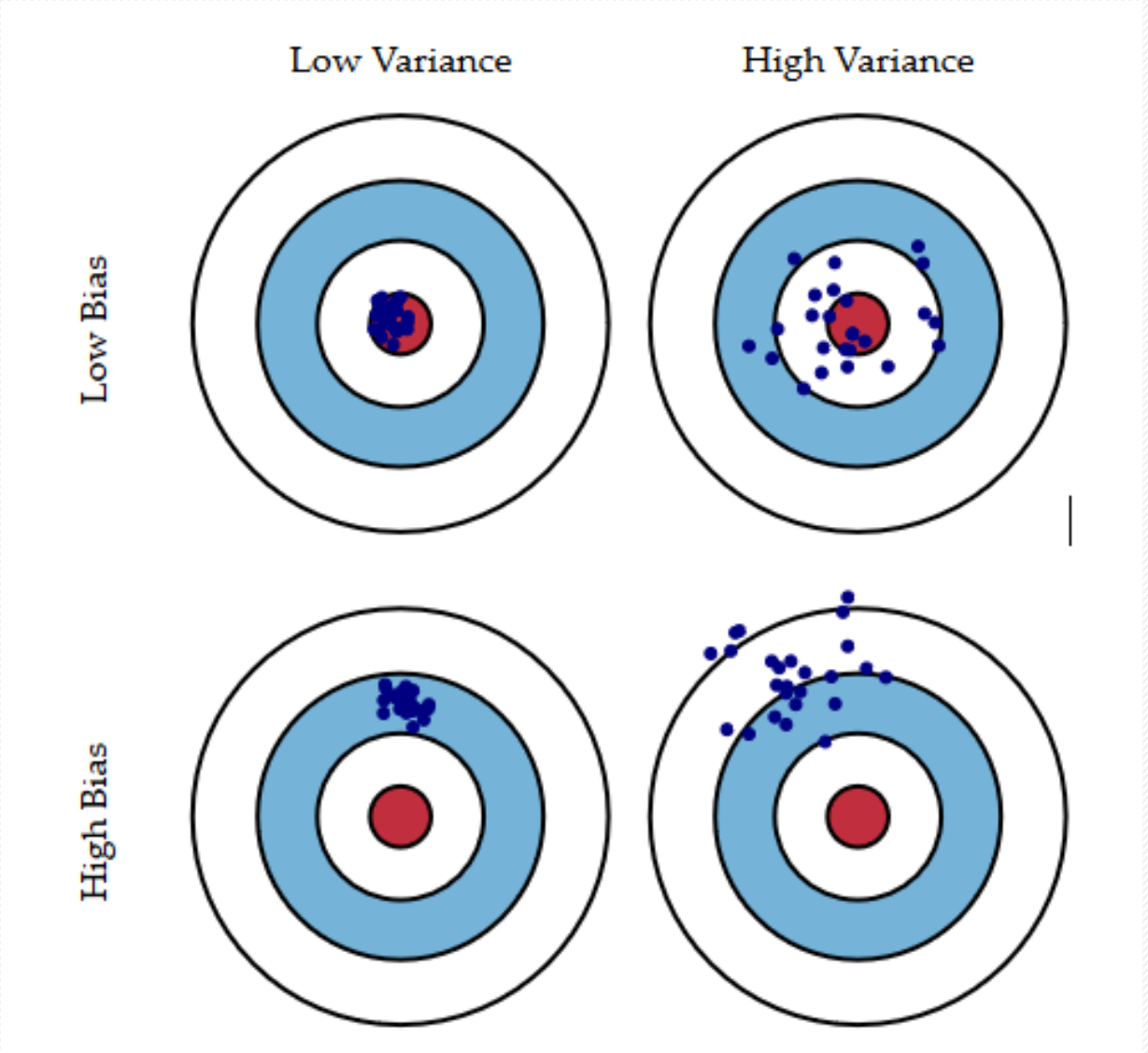
当前任务上任何学习算法所能达到的期望泛化误差下界

$$\varepsilon^2 = E_D[(y - y_D)^2]$$

训练样本的标记与真实标记有区别

泛化性能由学习算法的能力、数据的充分性以及学习任务本身的难度共同决定。

2.5 偏差与方差



偏差-方差窘境(bias-variance dilemma)

一般而言，偏差与方差存在冲突：

- 训练不足时，学习器拟合能力不强，**偏差主导**
- 随着训练程度加深，学习器拟合能力逐渐增强，**方差逐渐主导**
- 训练充足后，学习器的拟合能力很强，**方差主导**

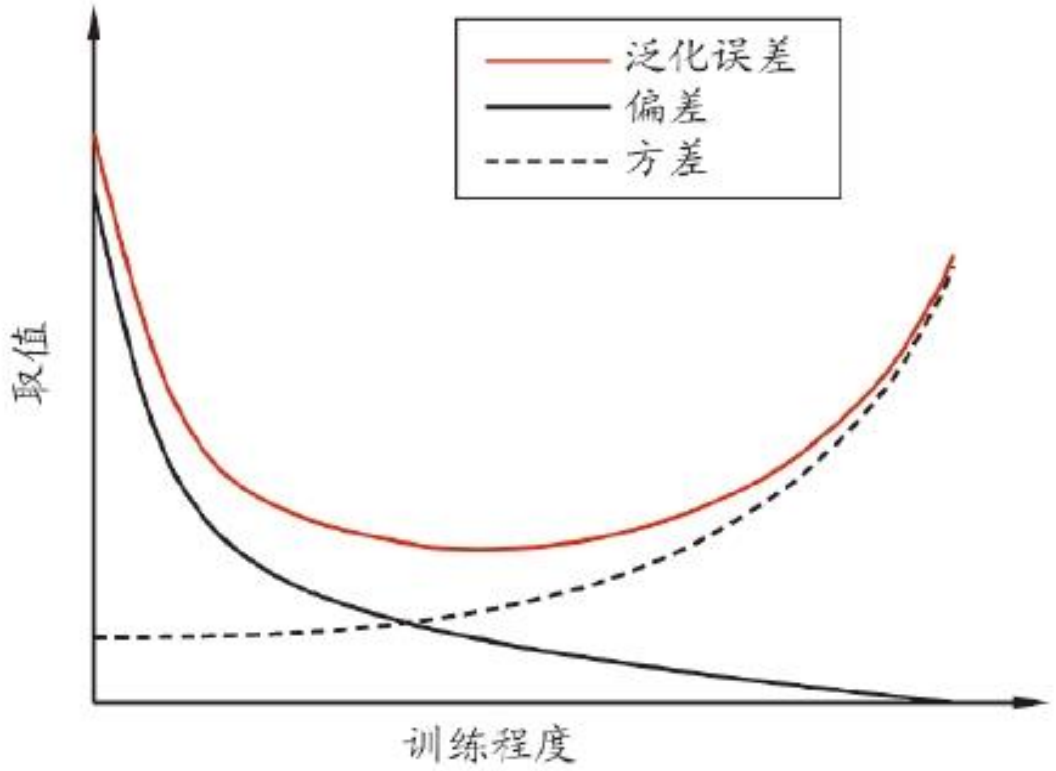


图 2.9 泛化误差与偏差、方差的关系示意图

The end