第3章

模型评估与选择

CHAPTER 3

模型评估与选择在机器学习中至关重要，因为它可以帮助确定哪个模型在特定任务上表现最佳。通过使用指标如准确率、精确率、召回率和 F_1 分数，可以量化模型的性能，避免过拟合或欠拟合。有效的评估方法（如交叉验证）还能提高模型的泛化能力，从而确保在新数据上的表现也同样优异。

3.1 评估方法

人们常说："物以类聚，人以群分"，判别一个人是一个什么样品质特征的人，常常可以从他/她身边的朋友入手，所谓观其友而识其人。例如，要判别图 3-1 中圆点属于哪一类数据，一个简单的方法是从它的邻居下手。但一次性看多少个邻居呢？

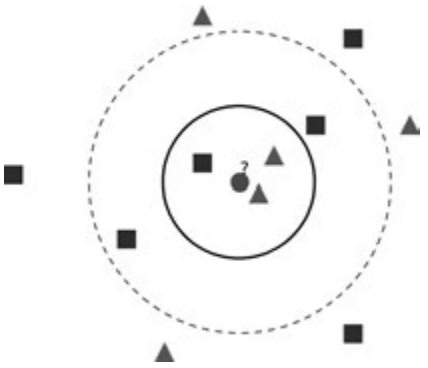

图 3-1 *K* 近邻算法示意图

如果 $K=3$，距离圆点最近的 3 个邻居是 2 个三角形和 1 个正方形，少数从属于多数，基于统计的方法，判定这个待分类点属于三角形一类。

如果 $K=5$，距离圆点最近的 5 个邻居是 2 个三角形和 3 个正方形，还是少数从属于多数，基于统计的方法，判定这个待分类点属于正方形一类。

可见，K 值的选择会对算法的结果产生重大影响，较小的 K 值意味着只有与输入实例较近的训练实例才会对预测结果起作用；如果 K 值较大，这时与输入实例较远的训练实例也会对预测起作用，使预测发生错误。

在实际应用中，如何确定 K 的取值呢？回顾评估学习效果的方法，一般的做法是，先学习再测试。例如，现在要检验某位同学线性代数课程学得怎么样，可以先给这位同学一些练习题，让他训练，然后再给出一些类似的题目对他进行测试，从测试结果来判断他学习的效果。

对于机器学习算法，也采用这样的策略，先使用一部分数据对模型进行训练，然后利用训练好的模型对新样本进行预测，通过模型在新样本上的表现来判断模型的优劣。机器学习算法对新样本的适应能力被称为算法的泛化能力(Generalization Ability)。泛化能力强的模型错误率低、精度高，能更好地适用于未知的样本。在机器学习任务中，通常希望找到能准确预测未知样本的标签，即泛化能力强的模型。

模型评估的目标是筛选出泛化能力强的模型来完成机器学习任务，实际的机器学习任务往往需要进行大量的实验，经过反复调参、使用多种模型算法，甚至是多模型融合策略来构建最终的机器学习模型，并观察模型算法在什么样的参数下能够最好地完成任务。常见的评估方法有留出法、交叉验证法以及自助采样法。

3.1.1 留出法

对于一个机器学习问题，通常有数据集 D(用于训练模型)，但还需要进行模型评估，因此不能把整个 D 用于训练，因为使用训练过的数据再去评估必然无效。留出法是机器学习中最常见的评估方法之一，它会从训练数据中留出测试样本集，这部分数据不用于训练，而用于模型评估，其完整的数学定义如下：把 D 划分为两部分——训练集 S 和测试集 T，其中，$S \cup T = D$，$S \cap T = \varnothing$。留出法的代码实现如下。

```
from sklearn.datasets import load_iris
```

```
X,y = load_iris(return_X_y = True)
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(
                                        X,
                                        y,
                                        test_size = 0.2,
                                        random_state = 10)
```

上述代码将数据集拆分成了训练集和测试集,其中,测试集的大小为原始数据集的 0.2 倍。

对于 K 近邻分类,可以利用留出法和准确率来确定 K 的取值。以下是一个示例。

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(
                                        X,
                                        y,
                                        test_size = 0.2,
                                        random_state = 10)
from sklearn.neighbors import KNeighborsClassifier
ks = [3,5,7,9,11,13,15,17,19,21,23,25]
accuracies = []
for k in ks:
    knn = KNeighborsClassifier(n_neighbors = k,weights = 'uniform')
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = sum(y_pred == y_test)/len(y_test) * 100
    accuracies.append(accuracy)
import matplotlib.pyplot as plt
plt.rcParams['font.sans - serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(dpi = 200)
plt.plot(ks,accuracies,'bo - ')
plt.xlabel("近邻数 K")
plt.ylabel("准确率( % )")
plt.title('使用留出法选择近邻数')
```

从图 3-2 可以看到,不同的 K 值会得到不同的准确率,在这个例子中,K 值取 9 时可以得到较高的准确率。

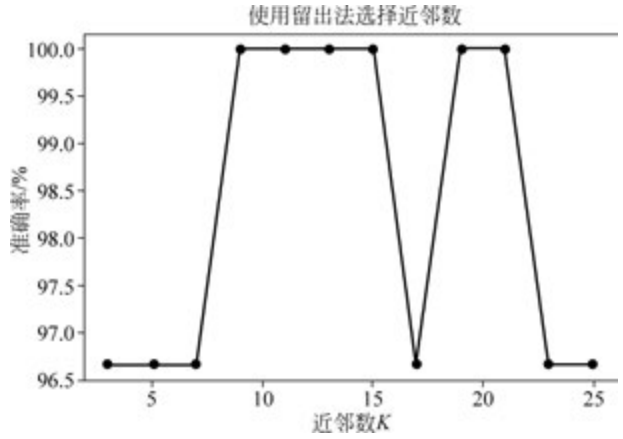

图 3-2　留出法选择近邻数

对于使用留出法进行数据划分，需要注意：①随机划分不一定能保证有效性，因为如果 T 中正好只取到某一种特殊类型数据，会带来额外的误差，此时处理方法要视具体情况而定，如当数据明显地分为有限类时，可以采用分层抽样方式选择测试数据，保证数据分布比例的平衡；②单次划分不一定能得到合适的测试集，一般多次重复划分、训练、求测试误差的步骤，取误差的平均值；③划分的验证集，太大或者太小都不合适，通常做法是选择 1/5～1/3 的数据当作验证集用于评估，留出法非常适用于大型数据集。

3.1.2 交叉验证法

交叉验证(Cross Validation)是在机器学习建立模型和验证模型参数时常用的办法。交叉验证，顾名思义，就是重复地使用数据，把得到的样本数据进行切分，组合为不同的训练集和测试集，用训练集来训练模型，用测试集来评估模型预测的好坏。在此基础上，可以得到多组不同的训练集和测试集，某次训练集中的某样本在下次可能成为测试集中的样本，即“交叉”。5 折交叉验证的示意图如图 3-3 所示。

测试集	训练集	训练集	训练集	训练集
训练集	**测试集**	训练集	训练集	训练集
训练集	训练集	**测试集**	训练集	训练集
训练集	训练集	训练集	**测试集**	训练集
训练集	训练集	训练集	训练集	**测试集**

图 3-3 5 折交叉验证示意图

当样本数量很少时，可以使用一种特殊的交叉验证方法：留一交叉验证。实现 10 折交叉验证的代码如下。

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)
from sklearn.model_selection import KFold
kf = KFold(n_splits = 10)
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]
```

对于分类问题，KFold 在进行数据集的划分时，可能抽取的样本与总体之间有较大的差距，不具有代表性，如某一个小的测试集中只有某一种类别。针对这个问题可以采用分层采样。分层采样具有以下优点：样本对总体的代表性较好，提高抽样精度，除估计总体指标外，还可估计各层指标，不同的层还可以采取不同的抽样方法。对于分类问题，实现分层采样交叉验证的代码如下。

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits = 10)
for train_index, test_index in skf.split(X, y):
    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]
```

使用 10 折交叉验证和准确率进行近邻算法 K 值选取的示例如下。

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)
from sklearn.model_selection import StratifiedKFold
n_splits = 10
skf = StratifiedKFold(n_splits = n_splits)
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
ks = [3,5,7,9,11,13,15,17,19,21,23,25]
accuracies = np.zeros((len(ks),n_splits))
for i,k in enumerate(ks):
    knn = KNeighborsClassifier(n_neighbors = k,weights = 'uniform')
    temp = []
    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        accuracy = sum(y_pred == y_test)/len(y_test) * 100
        temp.append(accuracy)
    accuracies[i,:] = np.array(temp)
import matplotlib.pyplot as plt
plt.rcParams['font.sans - serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(dpi = 200)
plt.plot(ks,accuracies.mean(axis = 1),'bo - ')
plt.xlabel("近邻数 K")
plt.ylabel("准确率(%)")
plt.title('使用 10 折交叉验证选择近邻数')
```

从图 3-4 可以看到，不同的 K 值会得到不同的准确率，在这个例子中，K 值取 13 时可以得到较高的准确率。

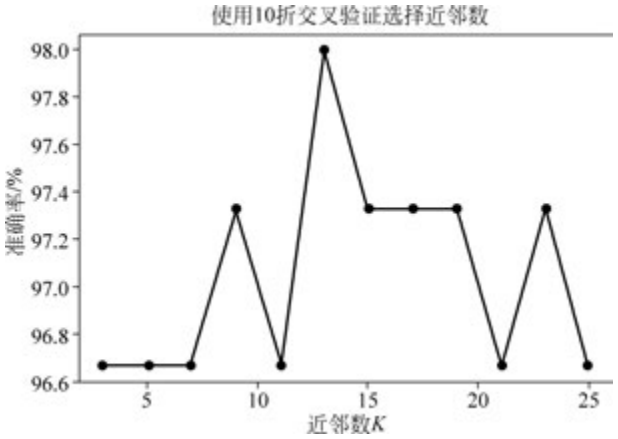

图 3-4　交叉验证法选择近邻数

以上代码的一个简化版本如下。

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)
from sklearn.neighbors import KNeighborsClassifier
ks = [3,5,7,9,11,13,15,17,19,21,23,25]
accuracies = []
```

```
from sklearn.model_selection import cross_val_score
for k in ks:
    knn = KNeighborsClassifier(n_neighbors = k,weights = 'uniform')
    accuracy = cross_val_score(knn,X,y,cv = 10,scoring = 'accuracy').mean() * 100
    accuracies.append(accuracy)
import matplotlib.pyplot as plt
plt.rcParams['font.sans - serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(dpi = 200)
plt.plot(ks,accuracies,'bo - ')
plt.xlabel("近邻数 K")
plt.ylabel("准确率(%)")
plt.title('使用 10 折交叉验证选择近邻数')
```

当数据不是很充足时,交叉验证可以评估训练好的模型在新数据上的表现,可以通过交叉验证来选择模型及其对应的参数。交叉验证可以在一定程度上减少过拟合,还可以从有限的数据中获取尽可能多的有效信息。

3.1.3 自助采样法

自助采样(Bootstrap)是一种有放回的随机抽样方法,用于从给定的数据集中生成具有相同大小的新数据集。假设原始数据集包含 N 个样本,需要得到一个包含 M 个样本的采样,那么自助采样就是从中随机选择一个样本并将其放回原始数据集,然后再进行下一次的随机选择,重复这个过程共 M 次,最终得到一个具有 M 个样本的新数据集。由于每次选择都是独立的,因此一些样本可能被选择多次,而另一些样本可能根本没有被选择。当样本数很大时,由极限运算 $\lim\limits_{\substack{N\to\infty\\M\to N}}\left(1-\frac{1}{N}\right)^M$ 可知,约有 36.8%的样本未出现在采样数据集中,约有 63.2%的样本出现在采样集中。可以利用这些从未被采样到的数据来对模型的性能进行评估,这样的评估结果叫作包外估计。利用自助采样法并进行 K 近邻分类算法 K 值选取的示例代码如下。

```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y = True)
from sklearn.utils import resample
from sklearn.neighbors import KNeighborsClassifier
ks = [3,5,7,9,11,13,15,17,19,21,23,25]
accuracies = []
temp = range(len(X))
for k in ks:
    train_index = resample(list(temp), n_samples = len(X),random_state = 170)
    test_index = list(set(temp) - set(train_index))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    knn = KNeighborsClassifier(n_neighbors = k,weights = 'uniform')
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = sum(y_pred == y_test)/len(y_test) * 100
    accuracies.append(accuracy)
import matplotlib.pyplot as plt
plt.rcParams['font.sans - serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(dpi = 200)
```

```
plt.plot(ks,accuracies,'bo-')
plt.xlabel("近邻数 K")
plt.ylabel("准确率(%)")
plt.title('使用自助采样-包外估计选择近邻数')
```

从图 3-5 可以看到，不同的 K 值会得到不同的准确率，在这个例子中，K 值取 13、15、17 时会得到较高的准确率。

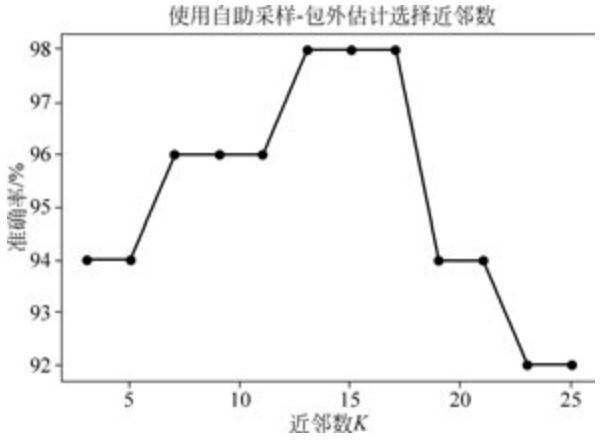

图 3-5　自助采样-包外估计选择近邻数

3.2　性能度量

对于一个机器学习模型，若其在训练数据集上表现很好，但在测试集上表现很差，称为过拟合；若其在训练集和测试集上表现都很差，则称为欠拟合。自然地，通常希望找到的是在测试集上表现好的模型，即泛化能力强的模型。

对机器学习算法泛化能力的评估，不仅需要评估方法，还需要有衡量算法优劣的评价标准，如分类算法的准确率，这就是性能度量(Performance Measure)指标。性能度量是衡量模型泛化能力的评价标准，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，对于不同的任务，有不同的性能度量准则。

3.2.1　回归的性能度量

在回归问题中，有多种性能度量指标用于评估模型的预测性能。以下是一些常见的回归性能度量指标。

均方误差(Mean Squared Error，MSE)：MSE 衡量了预测值与真实值之间的平方误差的平均值，值越小表示模型性能越好，其定义如下。

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

其中，n 是样本数量，y_i 是真实值，$\hat{y}_i$ 是模型的预测值。

均方根误差(Root Mean Squared Error，RMSE)：RMSE 是 MSE 的平方根，与 MSE 类似，但在量纲上与原始目标变量相同，因此更容易解释，其定义如下。

$$\mathrm{RMSE}=\sqrt{\mathrm{MSE}}$$

平均绝对误差(Mean Absolute Error,MAE):MAE 是预测值与真实值之间绝对误差的平均值,不易受异常值的影响,对异常值不敏感,值越小表示模型性能越好,其定义如下。

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i|$$

平均绝对百分比误差(Mean Absolute Percentage Error,MAPE):MAPE 表示预测误差相对于真实值的绝对百分比误差的平均值,越接近 0 越好,其定义如下。

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

注意,当真实值为零时,MAPE 的计算不能实现。

可决系数 R^2(拟合优度、决定系数):R^2 度量了模型解释方差的比例,取值范围为 0～1,越接近 1 表示模型越好,其定义如下。

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

其中,$\bar{y}$ 是真实值的均值。

可解释方差分数(Explained Variance Score,EVS):如果 $\hat{y}$ 是标签估计值,y 是与之相一致的真实的标签输出,则可解释方差的计算公式如下。

$$\mathrm{EVS}=1-\frac{\mathrm{var}(y-\hat{y})}{\mathrm{var}(y)}$$

EVS 最好的可能取值是 1,值越低模型性能越不好。

在实际问题中,选择哪个指标取决于具体问题的特点和对误差的敏感程度。实现以上指标的代码如下。

```
import numpy as np
y_true = np.random.rand(100,1) * 10
y_pred = y_true + np.random.rand(100,1) * 0.1
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_true,y_pred)
RMSE = mean_squared_error(y_true,y_pred) ** 0.5
from sklearn.metrics import mean_absolute_error
MAE = mean_absolute_error(y_true,y_pred)
MAPE = (abs((y_true - y_pred)/y_true) * 100).mean()
from sklearn.metrics import r2_score
R2 = r2_score(y_true,y_pred)
from sklearn.metrics import explained_variance_score
EVS = explained_variance_score(y_true,y_pred)
```

以上代码的简洁形式如下。

```
import numpy as np
from sklearn.metrics import (mean_squared_error,
                             mean_absolute_error,
                             r2_score,
                             explained_variance_score)
```

```
y_true = np.random.rand(100,1) * 10
y_pred = y_true + np.random.rand(100,1) * 0.1
MSE = mean_squared_error(y_true,y_pred)
RMSE = mean_squared_error(y_true,y_pred) ** 0.5
MAE = mean_absolute_error(y_true,y_pred)
MAPE = (abs((y_true - y_pred)/y_true) * 100).mean()
R2 = r2_score(y_true,y_pred)
EVS = explained_variance_score(y_true,y_pred)
```

对于 K 近邻算法，一个使用回归指标进行 K 值选取的示例如下。

```
from sklearn.datasets import load_diabetes
df = load_diabetes()
X = df.data
y = df.target
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
ks = list(range(1,50,2))
EVS,R2 = [],[]
for k in ks:
    knn = KNeighborsRegressor(n_neighbors = k,weights = 'uniform')
    R2.append(cross_val_score(knn, X,y,cv = 10,scoring = 'r2').mean())
    EVS.append(cross_val_score(knn, X,y,cv = 10,scoring = 'explained_variance').mean())
import matplotlib.pyplot as plt
plt.rcParams['font.sans - serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(dpi = 200)
plt.plot(ks,R2,'* - ',label = 'R^2')
plt.plot(ks,EVS,'o -- ',label = 'EVS')
plt.legend(loc = 'best')
plt.xlabel("近邻数 K")
plt.ylabel("指标取值")
plt.title('使用 R^2 和 EVS 选择近邻数')
```

从图 3-6 可知，随着 K 值的增加，R^2 和 EVS 的取值变大，但当 $K=15$ 以后，R^2 和 EVS 的取值没有太大变化，因此可以考虑 K 取 15。

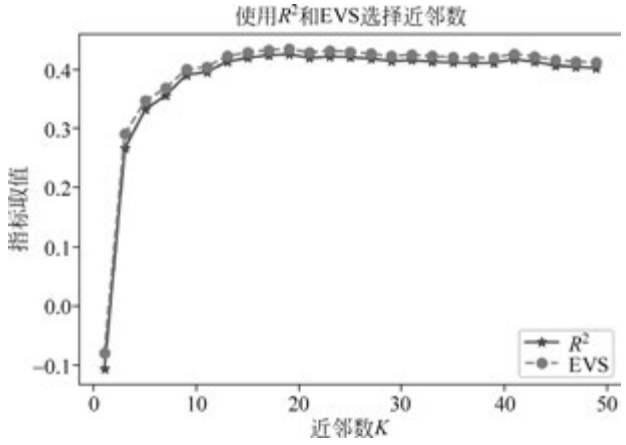

图 3-6　使用 R^2 和 EVS 选择近邻数

3.2.2 分类的性能度量

机器学习分类算法的评价指标有混淆矩阵(Confuse Matrix)、准确率(Accuracy)、错误率(Error Rate)、精确率(Precision)和召回率(Recall)、P-R 曲线、F_1 分数、真正例率(True Positive Rate,TPR)、假正例率(False Positive Rate,FPR)、ROC 曲线(Receiver Operating Characteristic Curve)、AUC(Area Under the Curve)。接下来对以上这些指标进行一一解释。

混淆矩阵可以使人们更好地了解机器学习分类算法的性能。对于鸢尾花数据集,假设 $K=5$ 时近邻算法的表现如表 3-1 所示。

表 3-1 鸢尾花混淆矩阵示例

真 实 值	预 测 值		
	山鸢尾	变色鸢尾	维吉尼亚鸢尾
山鸢尾	48	1	1
变色鸢尾	3	46	1
维吉尼亚鸢尾	0	1	49

以上表格可以用一个矩阵 $\boldsymbol{C}$ 来表示:

$$\boldsymbol{C}=\begin{bmatrix}48 & 1 & 1\\ 3 & 46 & 1\\ 0 & 1 & 49\end{bmatrix}$$

其中,元素 c_{ij} 表示真实值为第 i 类被预测为第 j 类的数量,矩阵 $\boldsymbol{C}$ 被称为该算法在鸢尾花数据集上的混淆矩阵。

准确率是预测正确的样本占所有样本的比例,其公式如下。

$$\text{Accuracy}=\frac{\sum_{i=1}^{n}c_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}}=\frac{\text{tr}(\boldsymbol{C})}{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}}$$

错误率指的是预测错误的样本占所有样本的比例,其公式如下。

$$\text{Error Rate}=1-\text{Accuracy}=\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}-\sum_{i=1}^{n}c_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}}=\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}-\text{tr}(\boldsymbol{C})}{\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}}$$

当数据类别不均衡时,特别是数据存在极端偏差的情况,准确率或错误率这种评价指标就无法客观评价机器学习算法的优劣。例如,在信用卡欺诈检测任务中,有 1 000 000 个测试样本,其中 999 000 条交易是正常交易,只有 1000 条记录存在欺诈。如果模型对任意一个测试样本都预测是正常交易,那么模型的准确率为 99%,从数值上看是非常好的,但事实上,这样的算法没有任何的预测能力。

在一个机器学习任务中,有时候人们更关心某一个类别的样本,如信用卡欺诈检测任务中,更加关心的是那些可能存在欺诈的交易。我们将这类对象叫作正例,除此之外的对象叫作反例。

精确率(Precision)又称查准率,它是针对预测结果而言的,它的含义是在所有被预测为正例样本中实际为正例样本的概率,即在预测为正样本的结果中,有多少把握可以预测正确,其计算公式如下。

$$\text{Precision}=\frac{\text{TP}}{\text{TP}+\text{FP}}$$

其中,TP 为真正例(True Positive),即一个正例被正确预测为正例; FP 为假正例(False Positive),即一个反例被错误预测为正例。

进一步,TN 表示真反例(True Negative),即一个反例被正确预测为反例; FN 表示假反例(False Negative),即一个正例被错误预测为反例。

例如,在鸢尾花分类任务中,如果更关心山鸢尾,则山鸢尾被视为正例,变色鸢尾和维吉尼亚鸢尾则被视为反例,则由混淆矩阵可知:

$$\text{TP}=48,\quad \text{FP}=3,\quad \text{TN}=46+1+1+49=97,\quad \text{FN}=2$$

于是可知: $\text{Precision}=\frac{48}{48+3}\approx 0.9412$。

召回率(Recall)又称为查全率,它是针对原样本而言的,它的含义是在实际为正例的样本中被预测为正例样本的概率,即有多少把握将正例样本预测出来,其计算公式如下。

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}}$$

接上面的例子,可知: $\text{Recall}=\frac{48}{48+2}=0.9600$。

在不同的应用场景下,关注点不同。例如,在预测股票的时候,更关心精确率,即预测升的那些股票里,真的升了有多少,因为那些预测升的股票都是投钱的。而在预测病患的场景下,更关注召回率,即真的患病的那些人里预测错了的情况应该越少越好。

通常使用精确率和召回率这两个指标来评价分类模型的效果,但是一般来说,精确率和召回率是一对此消彼长的度量,即精确率高了,召回率就下降。例如,在推荐系统中,想让推送的内容尽可能是用户全感兴趣的,那只能推送把握高的内容,这样就可能漏掉一些用户感兴趣的内容,召回率就低了; 如果想让用户感兴趣的内容都被推送,那可以将所有内容都推送,宁可错杀一千,不可放过一个,这样精确率就很低了,用户会感觉推送内容太多。

根据机器学习算法的预测结果(一般为一个实值或概率)对测试样本进行排序,将最可能是“正例”的样本排在前面,最不可能是“正例”的排在后面,按此顺序逐个把样本作为“正例”进行预测,依次计算出当前的精确率和召回率,将其作为坐标点(精确率作为纵坐标,召回率作为横坐标)绘制在坐标系中,最后将这些所有的点用平滑的曲线连接起来得到的曲线叫作 P-R 曲线,如图 3-7 所示。

当 P-R 曲线越靠近右上方时,表明模型性能越好。在对不同模型进行比较时,一般有以下两种情况: 若一个模型的 P-R 曲线被另一个模型的 P-R 曲线完全包住,则说明后者的性能优于前者,如图 3-7 中 C 代表的模型要优于 A、B 代表的模型。

若模型的 P-R 曲线发生了交叉,则可以通过查看“平衡点”(Break Even Point,BEP)来判断模型的性能,如图 3-7 所示黑色点,即当 $P=R$ 时的取值,平衡点的取值越高,性能更优。也可以根据 P-R 曲线、X 轴、Y 轴所围成图形的面积来判断模型的性能,谁曲线下的面积大,谁的性能更优,曲线下的面积用 AP 来表示。

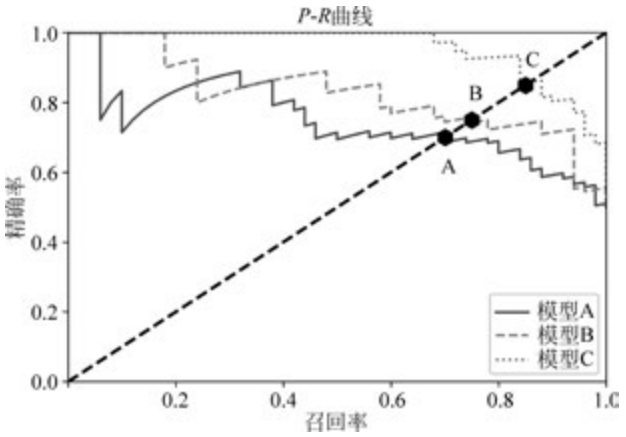

图 3-7 *P-R* 曲线与平衡点示意图

为了兼顾精确率和召回率，最常见的方法就是 F_1 度量，又称 F_1 分数。F_1 分数是精确率和召回率的调和平均，计算公式如下。

$$F_1=\frac{2}{\frac{1}{P}+\frac{1}{R}}=\frac{2PR}{P+R}$$

由均值不等式可知 $F_1\leqslant 1$ 且

$$\frac{\partial F_1}{\partial P}=\frac{2R^2}{(P+R)^2}\geqslant 0,\quad \frac{\partial F_1}{\partial R}=\frac{2P^2}{(P+R)^2}\geqslant 0$$

所以 F_1 的取值范围为 0～1，1 代表模型性能最好，0 代表模型性能最差，当 F_1 取值较大时，模型的性能较好。

在某些应用中，对查准率和查全率的重视程度不一样，此时使用精确率和召回率的加权调和平均来度量模型的性能。

$$F_\beta=\frac{1+\beta^2}{\frac{1}{P}+\beta^2\frac{1}{R}}=\frac{(1+\beta^2)PR}{R+\beta^2P},\quad \beta>0$$

当 $\beta=1$ 时，退化为 F_1 分数；当 $\beta>1$ 时，召回率有更大影响；当 $\beta<1$ 时，精确率有更大影响。

如果得到了多个混淆矩阵（如多次训练或者在多个同分布数据集上的训练结果），对这些混淆矩阵，有两种处理方式，第一种是在各个混淆矩阵上计算出每次的精确率 P_i 和召回率 R_i，然后计算 P_i 和 R_i 的平均值即可得到宏精确率、宏召回率和宏 F_1 分数。

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

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

$$\text{macro } F_1=\frac{2\times\text{macro } P\times\text{macro } R}{\text{macro } P+\text{macro } R}$$

还可以先将各个混淆矩阵进行平均，从而得到 TP、FP、TN、FN 的平均值 $\overline{\text{TP}}$、$\overline{\text{FP}}$、$\overline{\text{TN}}$、

$\overline{\mathrm{FN}}$，再基于这些平均值计算出微精确率、微召回率以及微 F_1 分数。

$$\text{micro } P = \frac{\overline{\mathrm{TP}}}{\overline{\mathrm{TP}} + \overline{\mathrm{FP}}}$$

$$\text{micro } R = \frac{\overline{\mathrm{TP}}}{\overline{\mathrm{TP}} + \overline{\mathrm{FN}}}$$

$$\text{micro } F_1 = \frac{2 \times \text{micro } P \times \text{micro } R}{\text{micro } P + \text{micro } R}$$

受试者工作特征曲线(ROC 曲线)，又称接收者操作特性曲线，最初用于评价雷达性能，后面被引入机器学习领域，用来评判分类、检测结果的好坏。类似于 P-R 曲线，ROC 曲线是根据一系列不同的阈值(按照从大到小排序)，以真正例率为纵坐标、假正例率为横坐标绘制的曲线。

真正例率(True Positive Rate，TPR)被定义为：在所有真实的正样本中，被模型正确地判断为正例的比例，其计算公式如下。

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

对比可知，真正例率就是召回率。

假正例率(False Positive Rate，FPR)被定义为：在所有真实的负样本中，被模型错误判断为正例的比例，计算公式如下。

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

显示 ROC 曲线的图称为 ROC 图，图 3-8 给出了 ROC 曲线的一个示意图。

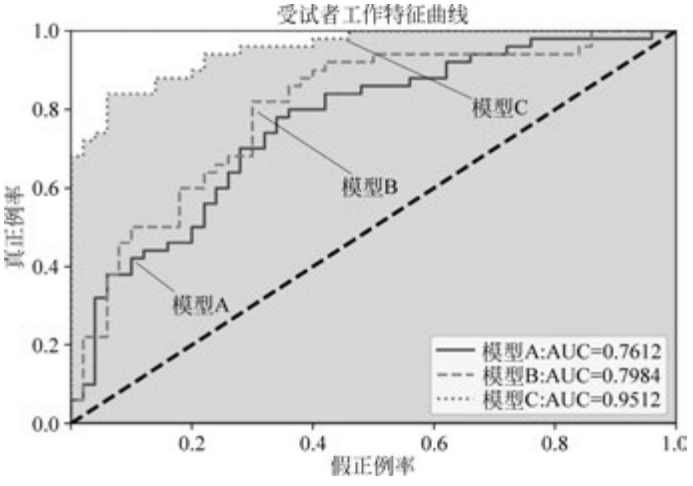

图 3-8　ROC 曲线与 AUC 示意图

一个模型的 ROC 曲线越靠近左上角，模型的准确度越高，模型越理想。若一个模型的 ROC 曲线被另一个模型的 ROC 曲线完全包住(从左上角往下看)，则说明后者的性能优于前者，如图 3-8 中 C 代表的模型要优于 A、B 代表的模型。

ROC 曲线与 X 轴、$X=1$ 所围成的面积称为曲线下的面积(Area Under Curve，AUC)。AUC 给出的是分类器的平均性能值，每一条 ROC 曲线对应一个 AUC 值，通常 AUC 的取

值为 0.5～1，模型的 AUC 值越大，模型越理想。

以 K 近邻算法和鸢尾花数据集为例，训练一个 $K=5$ 的近邻分类算法，然后以此计算以上性能指标，具体代码如下。

```
from sklearn.datasets import load_iris
X,y_true = load_iris(return_X_y = True)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X, y_true)
import numpy as np
np.random.seed(10)
X_new = X + 1e - 1 * np.random.rand(150,4)
y_pred = knn.predict(X_new)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
```

变量 cm 表示混淆矩阵，具体如表 3-2 所示。

表 3-2 K 近邻算法在测试数据上的混淆矩阵

真 实 值	预 测 值		
	山鸢尾	变色鸢尾	维吉尼亚鸢尾
山鸢尾	50	0	0
变色鸢尾	0	47	3
维吉尼亚鸢尾	0	2	48

进一步，可以将混淆矩阵以图形的形式显示，具体代码如下(需结合前面的代码才能运行)。

```
from sklearn.metrics import ConfusionMatrixDisplay
display_labels = ['山鸢尾','变色鸢尾','维吉尼亚鸢尾']
disp = ConfusionMatrixDisplay(confusion_matrix = cm,
                              display_labels = display_labels)
import matplotlib.pyplot as plt
plt.rcParams['font.sans - serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
fig,ax = plt.subplots(dpi = 200)
disp.plot(ax = ax,cmap = 'Purples')
ax.xaxis.set_ticks_position('top')
ax.set_xlabel('预测标签')
ax.set_ylabel('真实标签')
```

混淆矩阵可视化的结果如图 3-9 所示。

计算准确率的代码如下。

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)
```

计算精确率和召回率的代码如下。

```
from sklearn.metrics import (precision_score,
                             recall_score)
precision = precision_score(y_true, y_pred,average = None)
recall = recall_score(y_true, y_pred,average = None)
```

绘制三个类别的 P-R 曲线的代码如下。

```
from sklearn.metrics import (precision_recall_curve,
```

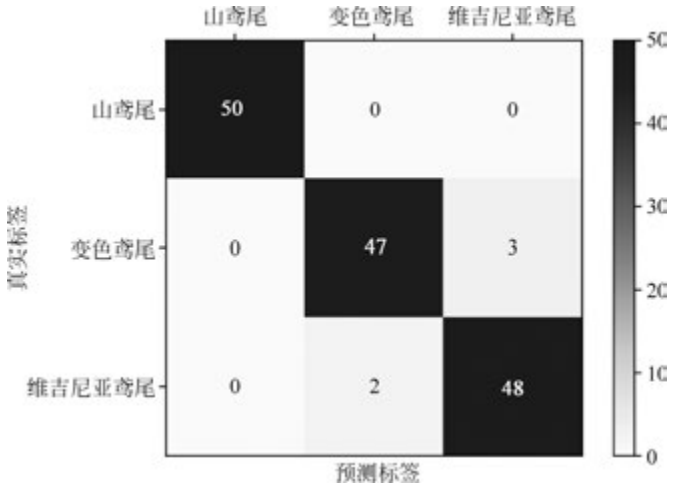

图 3-9　鸢尾花数据集混淆矩阵示意图

```
                                    PrecisionRecallDisplay)
y_pred_proba = knn.predict_proba(X_new)
fig,ax = plt.subplots(dpi = 200)
for pos_label in range(3):
    precision, recall, thresholds = \
        precision_recall_curve(y_true,
                               y_pred_proba[:,pos_label],
                               pos_label = pos_label)
    disp = PrecisionRecallDisplay(precision,recall)
    disp.plot(ax = ax,label = display_labels[pos_label])
ax.plot([0,1],[0,1],linewidth = 2,linestyle = '--',,c = 'k')
ax.set_xlim(0,1)
ax.set_ylim(0,1)
ax.legend(loc = 'best')
ax.set_xlabel('召回率')
ax.set_ylabel('精确率')
ax.set_title('P - R 曲线')
```

P-*R* 曲线可视化的结果如图 3-10 所示。

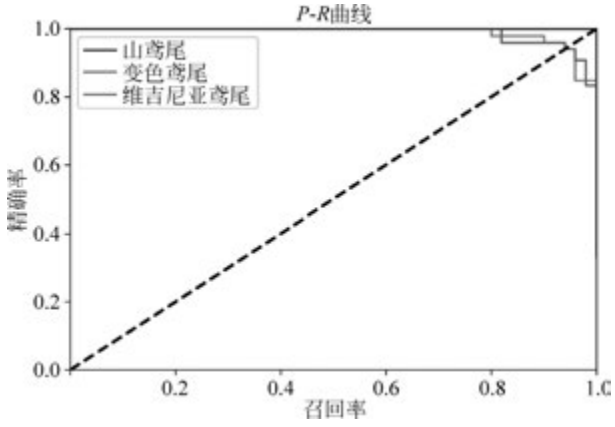

图 3-10　鸢尾花数据集 *P*-*R* 曲线示意图

计算 P-R 曲线下方面积的代码如下。

```
def ap_score(precision,recall):
    ap_score = 0
    recall = sorted(recall)
    precision = sorted(precision,reverse = True)
    for i in range(len(recall) - 1):
        ap_score += (precision[i + 1] + precision[i]) * (recall[i + 1] - recall[i]) * 1/2
    return ap_score
from sklearn.metrics import precision_recall_curve
ap_scores = []
for pos_label in range(3):
    precision, recall, thresholds = \
        precision_recall_curve(y_true,
                               y_pred_proba[:,pos_label],
                               pos_label = pos_label)
    ap_scores.append(ap_score(precision,recall))
```

使用分类报告查看各个类别的精确率、召回率：

```
from sklearn.metrics import classification_report
print(classification_report(y_true,y_pred))
```

计算 F_1 分数和加权 F_1 分数的代码如下。

```
from sklearn.metrics import f1_score,fbeta_score
f1 = f1_score(y_true,y_pred,average = None)
fbeta = fbeta_score(y_true,y_pred,beta = 0.5,average = None)
```

绘制三个类别的 ROC 曲线，计算曲线下面积的代码如下。

```
from sklearn.metrics import roc_curve,auc
plt.figure(dpi = 200)
for pos_label in range(3):
    fpr,tpr,thresholds = roc_curve(y_true,
                                   y_pred_proba[:,pos_label],
                                   pos_label = pos_label)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr,tpr,label = f'{display_labels[pos_label]}:AUC = {roc_auc}')
    temp = np.zeros(len(fpr))
    plt.plot(fpr,temp)
plt.plot([0, 1], [0, 1], 'k -- ', lw = 2)
plt.fill_between(fpr,temp,tpr,facecolor = 'blue',alpha = 0.05)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('假正例率')
plt.ylabel('真正例率')
plt.title('受试者工作特征曲线')
plt.legend(loc = "lower right")
```

ROC 曲线可视化的结果如图 3-11 所示。

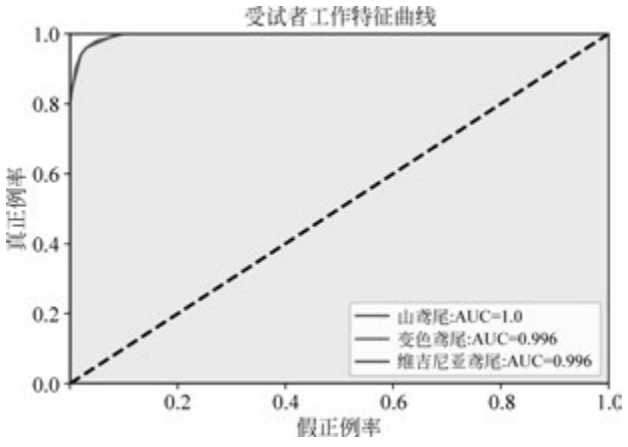

图 3-11　鸢尾花数据集 ROC 曲线与 AUC 示意图

3.2.3　聚类的性能度量

聚类算法属于非监督学习，它并不像分类算法那样可以使用训练集或测试集中的数据来计算准确率、召回率等。那么如何评估聚类算法的好坏呢？好的聚类算法，一般要求类簇的簇内(Intra-cluster)相似度高但是簇间(Inter-cluster)相似度低。一般来说，聚类的性能度量指标分为两类：内部评估评价指标和外部评估指标。

(1) **内部评估方法**：内部评估指标主要基于数据集的集合结构信息从紧致性、分离性、连通性和重叠度等方面对聚类划分进行评价，即基于聚类数据自身进行评估。常用的内部评价指标包括轮廓系数、Calinski-Harabasz 指数和 Davies-Bouldin 指数等。

(2) **外部评估方法**：外部有效指标是指当数据集的外部信息可用时，通过比较聚类划分与外部准则的匹配度，可以评价不同聚类算法的性能，即通过将聚类结果与已经存在的“真实”分类进行对比。但这类方法有一些问题，如果已经知道真实标签了，就不需要聚类了。但是可以利用已知信息的一个数据集去估计聚类算法的泛化性能。常用的外部评价指标包括兰德指数(调整后的)、基于互信息得分、同质性、完整性、V-measure 和 Fowlkes-Mallows 得分等。

接下来，简单介绍每个指标的优缺点及其实现代码，感兴趣的读者可以查阅相关资料了解各指标具体的数学原理。

轮廓系数：最早由 Peter J. Rousseeuw 在 1987 年提出，它结合内聚度和分离度两种因素。轮廓系数的取值范围为－1～1，轮廓系数接近 1 表示样本聚类合理，簇内距离较小且簇间距离较大；轮廓系数接近 0 表示样本聚类重叠；轮廓系数接近－1 表示样本被错误地分配到了相邻簇。凸簇的轮廓系数通常比其他类型的簇更高，如使用 DBSCAN 聚类算法获得的基于密度的簇。

Calinski-Harabasz 指数：Calinski-Harabasz 指数是 Calinski-Harabasz 距离的特殊形式，其本质是簇间距离与簇内距离的比值，且整体计算过程与方差计算方式类似，所以又将其称为方差比准则。Calinski-Harabasz 指数的分数越大，说明聚类效果越好(类别内部协方差越小越好，类别之间协方差越大越好)。凸簇的 Calinski-Harabasz 指数通常比其他类

型的簇更高，如使用 DBSCAN 聚类算法获得的基于密度的簇。

Davies-Bouldin 指数：Davies-Bouldin 指数表示簇之间的平均“相似度”，其中相似度是将簇之间的距离与簇本身的大小进行比较。如果簇与簇之间的相似度越高，也就说明簇与簇之间的距离越小，那么此时聚类结果就越差，此时 Davies-Bouldin 指数越大，所以 Davies-Bouldin 指数越小聚类效果越好，Davies-Bouldin 指数接近于零表示理想的聚类。

接下来，运用轮廓系数、Calinski-Harabasz 指数以及 Davies-Bouldin 指数作为评价聚类性能的标准来确定聚类的最佳类别数 K。首先构建 4 个不同标准差的二维样本数据并可视化这些不同类别的数据，然后运行 K-Means 聚类算法进行聚类并计算聚类结果的内部评价指数，最后可视化类别数和评价分数，具体代码如下。

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import (silhouette_score,
                                  calinski_harabasz_score,
                                 davies_bouldin_score)
plt.rcParams['font.sans-serif'] = ['KaiTi']
plt.rcParams['axes.unicode_minus'] = False
#定义 4 个簇类中心
centers = [[0,0],[1, 1],[1.9, 2],[3, 3]]
#定义每个簇类的标准差
stds = [0.1,0.2,0.3,0.4]
#产生 4 个簇类的 2000 个样本数据
X, labels_true = make_blobs(n_samples = 2000,
                                centers = centers,
                                cluster_std = stds,
                                random_state = 170)
plt.figure(dpi = 200)
plt.scatter(X[:,0],X[:,1],marker = '.',c = labels_true)
columns = ['silhouette_score','calinski_harabasz_score','davies_bouldin_score']
scores = {columns[0]: [],columns[1]:[],columns[2]:[]}
ks = [2,3,4,5,6]
for k in ks:
    k_Means = KMeans(init = 'k-Means++', n_clusters = k, n_init = 10,random_state = 10)
    y_pred = k_Means.fit_predict(X)
    scores['silhouette_score'].append(silhouette_score(X,y_pred))
    scores['calinski_harabasz_score'].append(calinski_harabasz_score(X,y_pred))
    scores['davies_bouldin_score'].append(davies_bouldin_score(X,y_pred))
fig,axs = plt.subplots(3,1,sharex = True,dpi = 200)
ys = ['轮廓系数','CH 指数','DB 指数']
for i in range(3):
    axs[i].plot(ks, scores[columns[i]],'o-')
    axs[i].set_ylabel(ys[i])
axs[2].set_xlabel('类别')
plt.tight_layout()
```

生成数据的散点图如图 3-12 所示。

类别数及其对应的评价指标分数如图 3-13 所示。

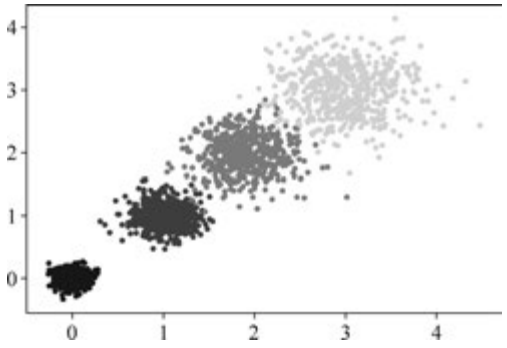

图 3-12　示例数据散点图

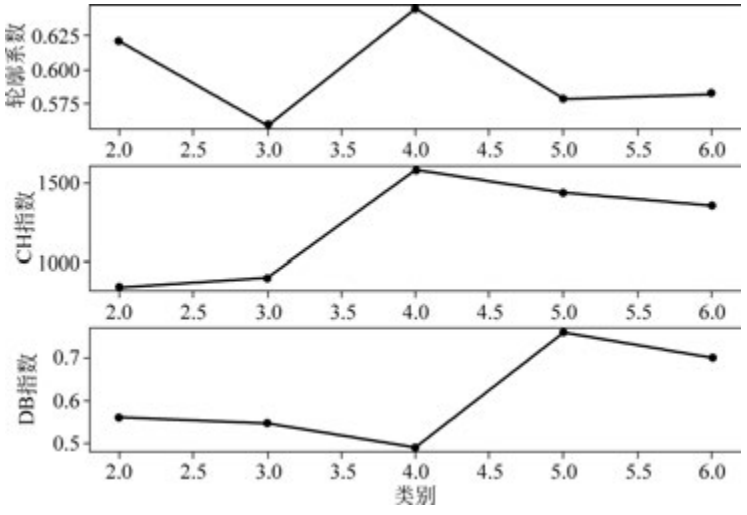

图 3-13　类别-评价指标分数图

由图 3-13 可知，类别数 $K=4$ 时，轮廓系数、Calinski-Harabasz 指数最高，同时 Davies-Bouldin 指数最低，因此选择聚类个数为 4。

兰德指数（调整后的兰德指数）：兰德指数衡量的是聚类算法将数据点分配到聚类中的准确程度。调整后的兰德指数取值范围为－1～1，1 表示两个聚类完全相同，0 表示聚类结果与随机结果相当，值越接近－1 表示聚类结果与真实类别完全相反。

基于互信息得分：基于互信息的分数衡量的是聚类结果与真实标签之间的相似性。这种度量方案有两个不同的归一化版本：Normalized Mutual Information（NMI）和 Adjusted Mutual Information（AMI）。基于互信息的分数取值范围为 0～1，其中，值越接近 1 表示聚类结果越准确，值越接近 0 表示聚类结果与随机结果相当，值越小表示聚类结果与真实类别之间的差异越大。基于互信息的分数是一种相对指标，它的取值受到真实类别数量的影响。当真实类别数量很大时，基于互信息的分数可能会受到偏差。

同质性、完整性和 V-measure：同质性和完整性是两个用于评估聚类结果的指标，它们可以用于衡量聚类算法对数据的划分是否“同质”（即簇内只包含同一类别的样本点）以及是否“完整”（即同一类别的样本点是否被划分到同一个簇中）。两者取值均为 0～1，取值越大越好。V-measure 是同质性和完整性的加权调和平均，一种综合考虑同质性和完整性的评

估指标，可以用于衡量聚类算法对数据的划分质量。V-measure 的取值范围也是 0～1，值越大表示聚类结果越好。

Fowlkes-Mallows 得分：Fowlkes-Mallows 得分是精确率和召回率的几何平均值：

$$\mathrm{FMI}=\sqrt{\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}\times\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}}$$

FMI 取值范围为 0～1，取值越大越好。

接下来，使用以上外部评价指标来比较不同聚类算法的性能。首先生成三分类的数据集，然后分别使用 K-Means、高斯混合模型和谱聚类算法进行聚类并计算各种聚类算法的外部评价指标，最后将所得结果保存为 Excel 文件，具体代码如下。

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import  KMeans,SpectralClustering
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
from sklearn.metrics import (adjusted_rand_score,
                                adjusted_mutual_info_score,
                                v_measure_score,
                                fowlkes_mallows_score
                                )
centers = [[1, 1], [-1, -1], [1, -1]]
n_clusters = len(centers)
n_samples = 3000
n_features = 2
X, labels_true = make_blobs(n_samples = n_samples,
                                n_features = n_features,
                                centers = centers,
                                cluster_std = [0.3,0.2,0.1],
                                random_state = 170)
np.random.seed(170)
X = X + np.random.rand(n_samples, n_features)
plt.figure(dpi = 200)
plt.scatter(X[:,0],X[:,1],marker = '.',c = labels_true)
columns = ['adjusted_rand_score','adjusted_mutual_info_score',
          'v_measure_score','fowlkes_mallows_score']
scores = {columns[0]: [],columns[1]:[],columns[2]:[],columns[3]:[]}
k_Means = KMeans(init = 'k-Means++', n_clusters = n_clusters, n_init = 10)
labels_pred = k_Means.fit(X).labels_
scores[columns[0]].append(adjusted_rand_score(labels_true,labels_pred))
scores[columns[1]].append(adjusted_mutual_info_score(labels_true,labels_pred))
scores[columns[2]].append(v_measure_score(labels_true,labels_pred))
scores[columns[3]].append(fowlkes_mallows_score(labels_true,labels_pred))
gauss = GaussianMixture(n_components = n_clusters)
labels_pred = gauss.fit(X).predict(X)
scores[columns[0]].append(adjusted_rand_score(labels_true,labels_pred))
scores[columns[1]].append(adjusted_mutual_info_score(labels_true,labels_pred))
scores[columns[2]].append(v_measure_score(labels_true,labels_pred))
scores[columns[3]].append(fowlkes_mallows_score(labels_true,labels_pred))
spe = SpectralClustering(n_clusters = n_clusters)
labels_pred = spe.fit(X).labels_
```

```
scores[columns[0]].append(adjusted_rand_score(labels_true,labels_pred))
scores[columns[1]].append(adjusted_mutual_info_score(labels_true,labels_pred))
scores[columns[2]].append(v_measure_score(labels_true,labels_pred))
scores[columns[3]].append(fowlkes_mallows_score(labels_true,labels_pred))
scores = pd.DataFrame(scores,index = ['KMeans','GaussianMixture','SpectralClustering'])
scores.to_excel('scores.xlsx')
```

具体结果如表 3-3 所示。

表 3-3　聚类评价结果

算　　法	指　　标			
	adjusted_rand _score	adjusted_mutual _info_score	v_measure _score	fowlkes_mallows _score
K 均值	0.9930	0.9856	0.9856	0.9953
高斯混合模型	0.9970	0.9932	0.9932	0.9980
谱聚类	0.9920	0.9841	0.9841	0.9947

从上述表格可以看到，这三个算法在这个数据集上各种评价指标取值相当，其中最好的是高斯混合模型。

3.3　比较检验

比较检验在统计学和研究方法学中具有重要的意义，其主要目的是评估不同组或条件之间的差异或相似性。比较检验是科学研究中不可或缺的工具，有助于研究者从统计学的角度评估和解释不同对象之间的关系，为决策提供可靠的统计证据。

在后续的章节中，读者会更多地了解机器学习算法。当读者对多个模型进行多次试验后，可以得到性能度量指标的多组数据，通过对这些数据进行比较检验会得出更为可靠的结果。下面以地表水水质评价为例，详细说明比较检验的一般过程。

地表水环境问题日益严峻，制约着我国实体经济的稳步发展，准确的地表水水质评价是解决地表水环境问题的前提条件，不仅可以实现对水资源的有效管理，更有利于水体污染综合防治方案的合理制定，因此对全国地表水环境的保护与规划具有重要的现实意义。

将水质等级作为目标变量，使用化学需氧量、总磷、高锰酸钾指数、溶氧量、氟化物、氨氮、五日生化需氧量 7 个属性作为特征变量，作为模型的输入，建立极端随机树、随机森林、AdaBoost、CatBoost、LightGBM、XGBoost、神经网络 7 个模型，模型的 10 折交叉验证准确率如表 3-4 所示。

表 3-4　各模型 10 折交叉验证准确率

序号	极端随机树准确率	随机森林准确率	AdaBoost准确率	CatBoost准确率	LightGBM准确率	XGBoost准确率	神经网络准确率
1	96.27%	96.02%	90.02%	95.54%	97.81%	96.27%	83.04%
2	97.73%	97.48%	94.89%	97.32%	97.48%	97.73%	86.61%
3	95.53%	96.10%	93.50%	96.59%	97.64%	96.67%	84.40%
4	96.26%	96.34%	93.74%	96.02%	96.18%	96.59%	85.54%
5	96.10%	96.10%	92.69%	96.18%	97.16%	95.69%	83.43%
6	97.56%	97.89%	95.53%	97.81%	97.64%	97.73%	83.83%

续表

序号	极端随机树准确率	随机森林准确率	AdaBoost准确率	CatBoost准确率	LightGBM准确率	XGBoost准确率	神经网络准确率
7	96.91%	97.24%	94.64%	96.67%	97.08%	97.64%	82.45%
8	98.70%	97.64%	96.02%	97.73%	97.40%	97.97%	85.13%
9	98.46%	97.73%	94.80%	97.32%	97.32%	97.89%	86.84%
10	98.13%	97.48%	94.96%	97.32%	97.81%	98.13%	83.59%

以准确率作为评估标准，可以发现神经网络和AdaBoost算法的表现一般，平均准确率明显低于其他5个模型，那剩下的5个模型呢？它们的平均准确率是否存在显著差异呢？

方差分析(ANOVA)是一种对模型的预测能力是否具有显著性差异的检验方法，使用方差分析的前提条件是被检验的每一个随机变量都要服从正态分布并且满足方差齐性，因此需要对5个模型的准确率进行正态性检验和方差齐性检验。利用各模型十折交叉验证的准确率绘制正态性检验的“分位数-分位数”(Quantile-Quantile，QQ)图与核密度估计直方图，结果如图3-14～图3-18所示。

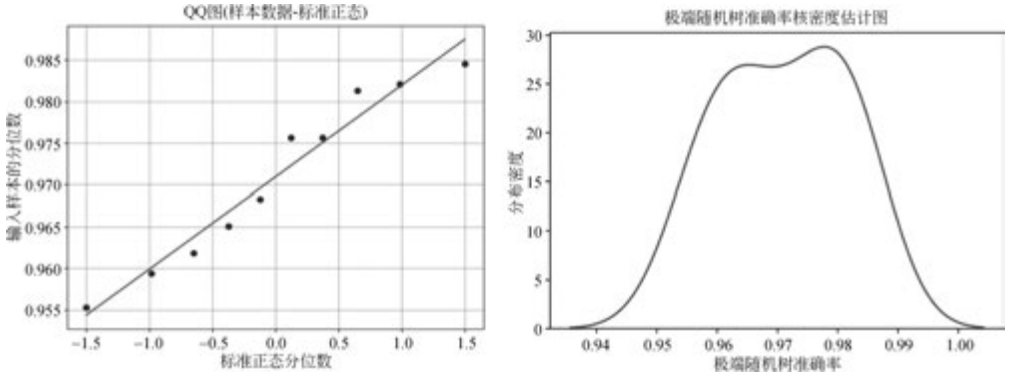

图3-14 极端随机树QQ图与核密度估计图

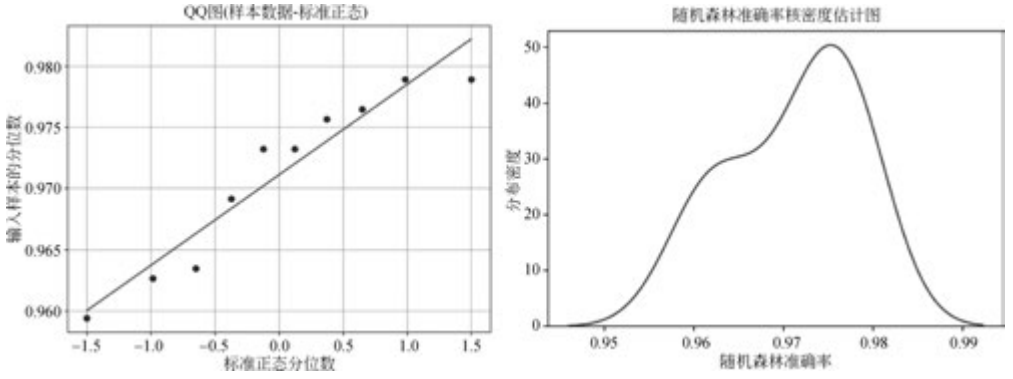

图3-15 随机森林QQ图与核密度估计图

由图3-14～图3-17可知，样本点与直线重合度较好，对应4个模型准确率基本服从正态分布，所对应的核密度估计图反映出随机森林与正态分布有差距，而如图3-18所示的

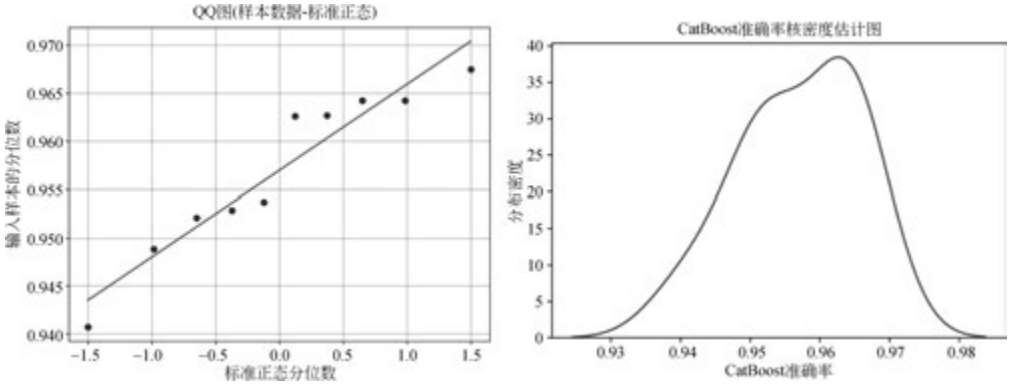

图 3-16　CatBoost QQ 图与核密度估计图

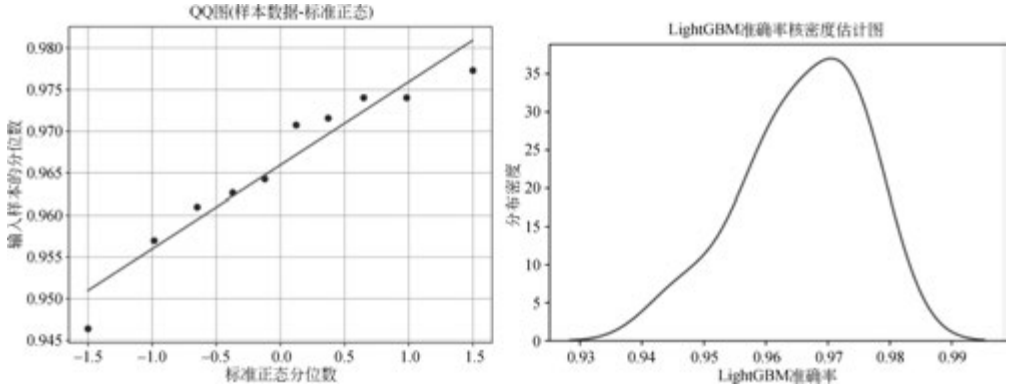

图 3-17　LightGBM QQ 图与核密度估计图

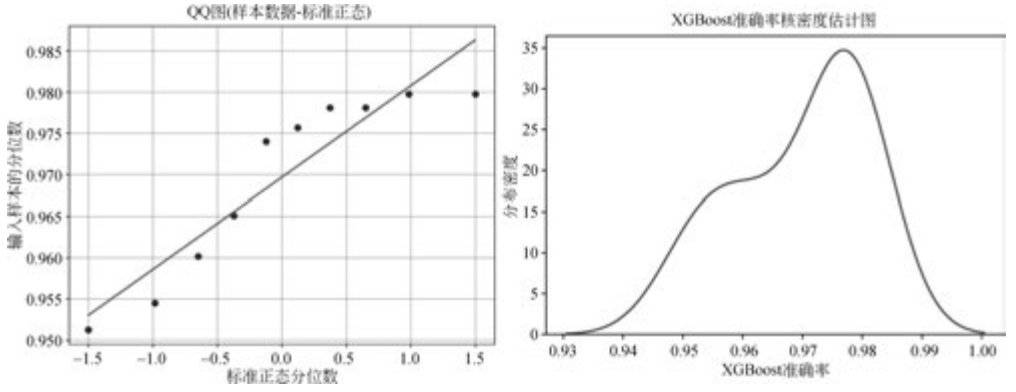

图 3-18　XGBoost QQ 图与核密度估计图

XGBoost 样本点与直线之间有所偏移，所对应的核密度估计图同样与正态分布有差距。因此认为 5 个模型中至少 XGBoost 的准确率不服从正态分布，不能使用方差分析。

进一步采用 W 检验验证上述可视化得出的结论，在显著性水平为 0.05 的情况下，W 检验的原假设是：模型准确率的分布与正态分布无显著差异。备择假设是：模型准确率的分布与正态分布有显著差异。检验结果如表 3-5 所示。

表 3-5 5 个模型 W 检验结果

模型	极端随机树	随机森林	CatBoost	LightGBM	XGBoost
P 值	0.4992	0.2089	0.3043	0.4162	0.0455

由表 3-5 知，XGBoost 的 P 值小于 0.05，因此拒绝原假设，即 XGBoost 总体不服从正态分布，因此进一步验证了不能使用方差分析对以上模型的预测能力是否相当做假设检验，故使用非参数假设检验方法。

Kruskal-Wallis(K-W)检验是一种用于比较三个或更多个不相关组之间差异的非参数统计检验方法。它的目的是确定组之间是否存在显著差异，而不需要满足方差分析的正态性和方差齐性假设。在显著性水平为 $\alpha=0.05$ 的情况下，原假设为：5 个模型十折交叉验证的准确率均值相同。备择假设为：5 个模型十折交叉验证的准确率均值不同。模型间的准确率均值相同就代表其预测能力相当。K-W 检验结果如表 3-6 所示。

表 3-6 K-W 检验结果

模型	模　型				
	极端随机树	随机森林	CatBoost	LightGBM	XGBoost
极端随机树	—	0.5966	0.3845	0.5449	1
随机森林	0.5966	—	0.5960	0.1123	0.4267
CatBoost	0.3845	0.5960	—	0.0586	0.1981
LightGBM	0.5449	0.1123	0.0586	—	0.2567
XGBoost	1	0.4267	0.1981	0.2567	—

由表 3-6 知，P 值均大于 0.05，故接受原假设，即极端随机树、随机森林、CatBoost、LightGBM、XGBoost 在显著性水平为 0.05 的条件下各模型间准确率均值无显著差异。

进一步利用 Mann-Whitney U 检验(U 检验)对上述结果进行验证。U 检验用于检验样本两两之间总体的均值是否相等，即在显著性水平为 $\alpha=0.05$ 的情况下，原假设为：所选两个模型十折交叉验证的准确率均值相同。备择假设为：所选两个模型十折交叉验证的准确率均值不同，检验结果如表 3-7 所示。

表 3-7 U 检验结果

模型	模　型				
	极端随机树	随机森林	CatBoost	LightGBM	XGBoost
极端随机树	—	0.5701	0.3642	0.5201	1
随机森林	0.5706	—	0.5700	0.1040	0.4050
CatBoost	0.3642	0.5700	—	0.0537	0.1852
LightGBM	0.5201	0.1040	0.0537	—	0.2411
XGBoost	1	0.4050	0.1852	0.2411	—

由表 3-7 可知，所有模型样本两两配对检验的 P 值均大于 0.05，故接受原假设，即在显著性水平为 0.05 的条件下各模型间准确率均值两两比较均无显著差异。

综上所述，两种非参数假设检验结果都通过，5 个单一模型的预测能力相当。

3.4　偏差与方差

在机器学习中，偏差(Bias)和方差(Variance)是两个关键的概念，有助于人们理解模型的性能和泛化能力。

偏差：偏差衡量了模型的预测值与实际值之间的差异。它表示模型对问题的简化程度，或者说模型对于真实关系的拟合能力。高偏差的模型往往会对训练数据和新数据都表现得较为简单，忽略了数据的复杂性。高偏差的模型容易出现欠拟合，即在训练和测试数据上都表现不佳。

方差：方差衡量了模型在不同训练数据集上预测值的变化程度。它表示模型对训练数据的敏感程度，或者说模型的波动性。高方差模型对训练数据表现得很好，但在新数据上的表现可能较差，容易受到噪声的影响，高方差模型容易出现过拟合，即在训练数据上表现良好，但在新数据上表现不佳。

一个机器学习算法偏差和方差的组合，可能情况如图 3-19 所示。在实际应用中，我们追求偏差和方差的平衡，以获得良好的泛化性能。高偏差低方差模型：倾向于欠拟合，对数据拟合不足，忽略了数据的复杂性。低偏差高方差模型：倾向于过拟合，对训练数据过于敏感，无法很好地泛化到新数据。

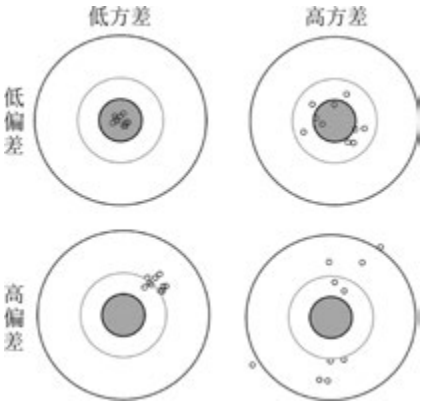

图 3-19　偏差-方差示意图

对于回归问题，将多个预测能力相当的模型进行组合可以降低最终模型的方差。假设存在 n 个预测能力相当的模型，其各自的预测值是 $Y_1, Y_2, \cdots, Y_n$，组合预测的结果为

$$Y=\frac{Y_1+Y_2+\cdots+Y_n}{n}$$

则 Y 的方差为

$$\begin{aligned}D(Y)&=D\left(\frac{Y_1+Y_2+\cdots+Y_n}{n}\right)=\frac{1}{n^2}D(Y_1+Y_2+\cdots+Y_n)\\&=\frac{1}{n^2}\left(\sum_{i=1}^{n}D(Y_i)+\sum_{1\leqslant i,j\leqslant n,i\neq j}^{n}\operatorname{cov}(Y_i,Y_j)\right)\\&=\frac{1}{n^2}\left(n\sigma^2+\sum_{1\leqslant i,j\leqslant n,i\neq j}^{n}\rho_{ij}\sigma^2\right)=\frac{1}{n^2}\left(n+\sum_{1\leqslant i,j\leqslant n,i\neq j}^{n}\rho_{ij}\right)\sigma^2\end{aligned}$$

因为 $\rho_{ij}\leqslant 1$，所以 $D(Y)\leqslant\sigma^2$。

习题

本书提供在线测试习题，读者扫描下方二维码可以获取本章习题。

在线测试