第 5 章 机器学习的基本算法

机器学习(Machine Learning,ML)是一门多领域交叉学科,涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科,专门研究计算机如何模拟或实现人类的学习行为,以获取新的知识或技能,重新组织已有的知识结构使之不断改善自身的性能。

本章围绕最基本的聚类和分类算法,阐述 K-Means、K 近邻、决策树、支持向量机和回归模型的基本原理、应用技术和实战方法。

学习目标:

(1) 能够分析机器学习的聚类和分类基本算法。

(2) 通过编程实现,形成聚类和分类基本算法的应用能力。

5.1 K-Means 算法原理及应用

聚类是一个将数据集中在某些方面相似的数据成员进行分类组织的过程,聚类就是一种发现这种内在结构的技术,聚类技术经常被称为无监督学习。在人工智能算法中,聚类算法一般都是作为其他算法分析的基础,对数据进行聚类可以从整体上分析数据的一些特性。

5.1.1 K-Means 算法描述

K-Means 可视化微视频

K-Means 算法是一种迭代求解的聚类分析算法,是一种最简单最实用的聚类算法。

假定要对 N 个样本观测做聚类,要求聚为 K 类,则 K-Means 算法的步骤描述如下。

(1) 给出 K 个初始聚类中心。

(2) 按照距离初始中心点最小的原则,把每一个数据对象重新分配到 K 个聚类中心处,形成 K 个簇。

(3) 重新计算每一个簇的聚类中心。

(4) 重复第(2)、(3)步,直到收敛(聚类中心不再发生变化,或误差平方和局部最小,或达到指定的迭代次数),聚类过程结束。

下面以二维平面中的点 $X_i=(x_{i1},x_{i2})$,$i=1,2,\cdots,n$ 为例,展示 $K=2$ 时的迭代过程,如图 5.1 所示。

5.1.2 K-Means 算法的参数设计

从 K-Means 算法可见,存在以下参数设计要点:初始中心点如何确定、K 如何选择、距离如何计算、各类中心点如何计算等。下面重点阐述。

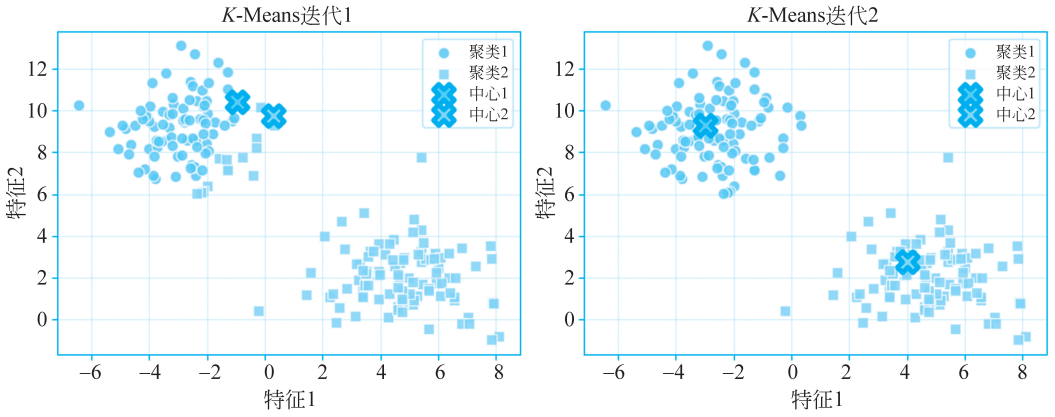

图 5.1 K-Means 的迭代过程示例

1. K-Means 最近邻的度量

在 K-Means 算法中，需要把数据集分到距离聚类中心最近的那个簇中，这样就需要最近邻的度量策略。K-Means 算法中最常用的度量公式：在欧氏空间中采用的是欧氏距离，在处理文档时采用的是余弦相似度函数，有时候也采用曼哈顿距离作为度量，不同的情况下使用的度量公式是不一样的。

K-Means 算法要解决的问题是把数据分成不同的簇，目标是使得同一个簇的差异很小，不同簇之间的数据差异最大化。一般采用误差平方和 SSE(Sum of the Squared Errors)作为衡量的目标函数：

$$\mathrm{SSE}=\sum_{i=1}^{K}\sum_{x\in C_i}(C_i-x)^2 \tag{5-1}$$

式中，C 表示聚类中心的均值；x 是属于这个簇的数据点。可以证明，当聚类中心为簇中的均值时，才能使 SSE 最小。

2. 初始中心的选择

例如，先随便选个点作为第 1 个初始中心 C_1，接下来计算所有样本点与 C_1 的距离，距离最大的被选为下一个中心 C_2，直到选完 K 个中心。这个算法叫作 K-Means++，可以理解为 K-Means 的改进版，它能有效地解决初始中心的选取问题，但无法解决离群点问题。

最好的解决办法是多设置几个不同的初始点，从中选最优，也就是具有最小 SSE 值的那组作为最终聚类。

3. K 的选择

确定聚类数 K 的方法主要有手肘法和轮廓系数法两种。

1) 手肘法

手肘法的核心指标是 SSE，其核心思想是：随着聚类数 K 的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和 SSE 自然会逐渐变小。并且，当 K 小于真实聚类数时，由于 K 的增大会大幅增加每个簇的聚合程度，故 SSE 的下降幅度会很大，而当 K 达到真实聚类数时，再增加 K 所得到的聚合程度会迅速变小，所以 SSE 的下降幅度会骤减，然后随着 K 值的继续增大而趋于平缓，也就是说，SSE 和 K 的关系图是一个

手肘的形状，而这个肘部对应的 K 值就是数据的真实聚类数，作为最佳选择。

下面给出一个 Python 实现代码。

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

df_features = pd.read_csv(r'C:\test.csv',encoding='gbk')    #读入数据
SSE = []                                                    #存放每次结果的误差平方和
for k in range(1,9):
    estimator = KMeans(n_clusters=k)                        #构造聚类器
    estimator.fit(df_features[['R','F','M']])
    SSE.append(estimator.inertia_)
X = range(1,9)
plt.xlabel('k')
plt.ylabel('SSE')
plt.plot(X,SSE,'o-')
plt.show()
```

据此画出 K 与 SSE 的关系，如图 5.2 所示。

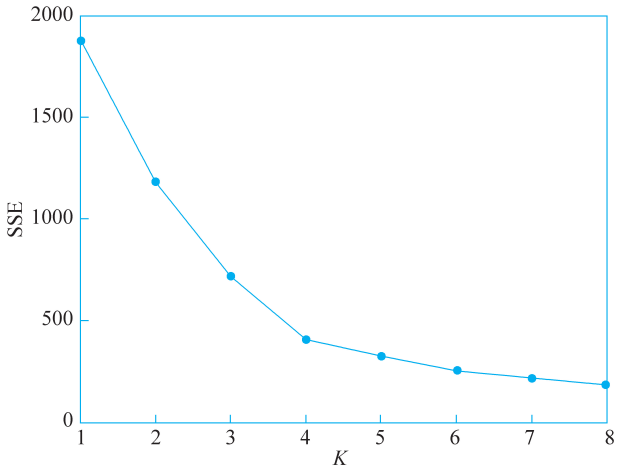

图 5.2　手肘法求解 *K*-Means 算法的最佳 *K*

可见，肘部对应的 K 值为 4，故对于这个数据集的聚类而言，最佳聚类数应该选 4。

2）轮廓系数法

该方法的核心指标是轮廓系数 S（Silhouette Coefficient），某个样本点 X_i 的轮廓系数定义如下。

$$S=\frac{b-a}{\max(a,b)} \tag{5-2}$$

其中，a 是 X_i 与同簇的其他样本的平均距离，称为凝聚度；b 是 X_i 与最近簇中所有样本的平均距离，称为分离度。最近簇的定义是：

$$C_j=\arg\min_{C_k}\frac{1}{n}\sum_{p\in C_k}|p-X_i|^2 \tag{5-3}$$

其中，p 是某个簇 C_k 中的样本。事实上，就是用 X_i 到某个簇所有样本的平均距离作为衡量该点到该簇的距离后，选择离 X_i 最近的一个簇作为最近簇。

求出所有样本的轮廓系数后再求平均值就得到了平均轮廓系数。平均轮廓系数的取值范围为[−1,1]，且簇内样本的距离越近，簇间样本距离越远，平均轮廓系数越大，聚类效果越好。因此，平均轮廓系数最大的 K 便是最佳聚类数。

比较而言，目前多采用手肘法。

5.1.3 K-Means 算法的应用示例

【例 5-1】 sklearn 库中包含 Iris 鸢尾花数据集，共有 150 个实例，属性有萼片长度、萼片宽度、花瓣长度、花瓣宽度(Sepal Length，Sepal Width，Petal Length，Petal Width)，均匀分布在三个亚种上：山鸢尾、杂色鸢尾、弗吉尼亚鸢尾(Iris Setosa，Iris Versicolor，Iris Virginica)。

鸢尾花照片如图 5.3 所示。

下面基于 Iris 数据集，说明 K-Means 算法的 Python 实现。

(1) 数据分布描述，如图 5.4 所示。

图 5.3 鸢尾花照片示例

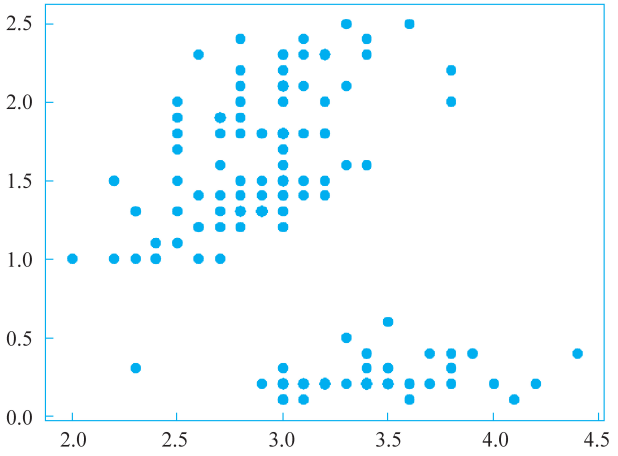

图 5.4 Iris 数据分布图

(2) K-Means 算法实现。

```
k = 3                                                    #k 为聚类的类别数
n = 150                                                  #n 为样本总个数
t = 4                                                    #t 为数据集的特征数
def k_means():
    #随机选 k 个初始聚类中心，聚类中心为每一类的均值向量
    m = np.zeros((k,d))
    for i in range(k):
            m[i] = data[np.random.randint(0,n)]
    #k_means 聚类
    m_new = m.copy()
    t = 0
    while (1):
```

```
        #更新聚类中心
        m[0] = m_new[0]
        m[1] = m_new[1]
        m[2] = m_new[2]
        w1 = np.zeros((1,d))
        w2 = np.zeros((1,d))
        w3 = np.zeros((1,d))

        for i in range(n):
            distance = np.zeros(3)
            sample = data[i]
            for j in range(k):                    #将每一个样本与聚类中心比较
                distance[j] = np.linalg.norm(sample - m[j])
            category = distance.argmin()
            if category==0:
                w1 = np.row_stack((w1,sample))
            if category==1:
                w2 = np.row_stack((w2,sample))
            if category==2:
                w3 = np.row_stack((w3,sample))
        #新的聚类中心
        w1 = np.delete(w1,0,axis=0)
        w2 = np.delete(w2,0,axis=0)
        w3 = np.delete(w3,0,axis=0)
        m_new[0] = np.mean(w1,axis=0)
        m_new[1] = np.mean(w2,axis=0)
        m_new[2] = np.mean(w3,axis=0)

        #聚类中心不再改变时,聚类停止
        if (m[0]==m_new[0]).all() and (m[1]==m_new[1]).all() and (m[2]==m_new
[2]).all():
            break
        print(t)
        t+=1

        #画出每一次迭代的聚类效果图
        w = np.vstack((w1,w2))
        w = np.vstack((w,w3))
        label1 = np.zeros((len(w1),1))
        label2 = np.ones((len(w2),1))
        label3 = np.zeros((len(w3),1))
        for i in range(len(w3)):
            label3[i,0] = 2
        label = np.vstack((label1,label2))
        label = np.vstack((label,label3))
        label = np.ravel(label)
        test_PCA(w,label)
        plot_PCA(w,label)
    return w1,w2,w3
```

聚类结果如图 5.5 所示。

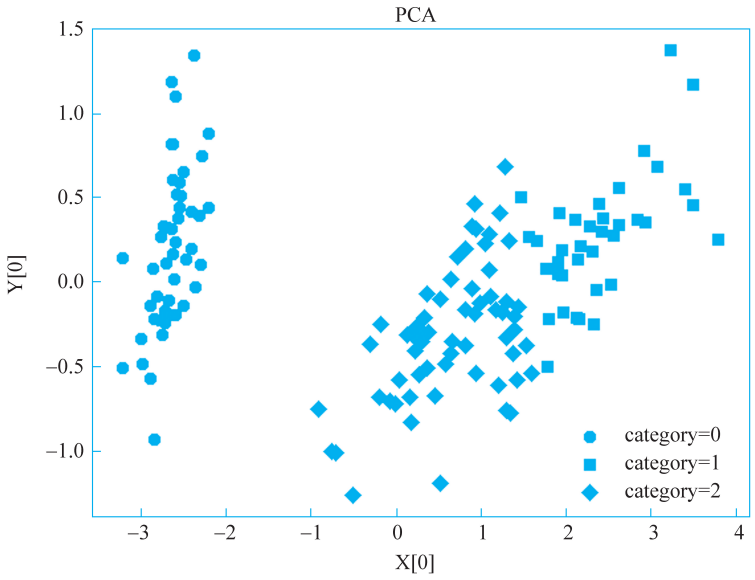

图 5.5　采用 *K*-Means 的 Iris 数据聚类结果

5.2　*K* 近邻算法

5.2.1　KNN 算法概述

K 近邻(*K*-Nearest Neighbors,KNN)算法是一种分类算法,也是最简单易懂的机器学习算法。1968 年由 Cover 和 Hart 提出,应用场景有字符识别、文本分类、图像识别等领域。KNN 的工作原理:存在一个样本数据集合,也称为训练样本集,并且样本集中每个数据都存在标签,即我们知道样本集中每一个数据与所属分类对应的关系。输入没有标签的数据后,将新数据中的每个特征与样本集中数据对应的特征进行比较,提取出样本集中特征最相似数据(最近邻)的分类标签。一般来说,只选择样本数据集中前 *K* 个最相似的数据,这就是 *K* 近邻算法中 *K* 的出处,通常 *K* 是不大于 20 的整数。最后选择 *K* 个最相似数据中出现次数最多的分类作为新数据的分类。

KNN 算法的一般流程如下。

(1) 计算测试数据与各个训练数据之间的距离。

(2) 按照距离的递增关系进行排序。

(3) 选取距离最小的 *K* 个点。

(4) 确定前 *K* 个点所在类别的出现频率。

(5) 返回前 *K* 个点中出现频率最高的类别作为测试数据的预测分类。

使用 KNN 做回归和分类的主要区别在于最后做预测时的决策方式不同。KNN 做分类预测时,一般是选择多数表决法,即训练集里和预测的样本特征最近的 *K* 个样本,预测为里面有最多类别数的类别。而 KNN 做回归时,一般是选择平均法,即最近的 *K* 个样本的样本输出的平均值作为回归预测值。

为了便于理解，先看图 5.6。图中有正方形和三角形两种类型的样本数据，中间那个圆形是待分类数据。

思考：图中圆形样本属于正方形和三角形的哪个类型？

由图可见，如果 $K=3$，那么离圆形点最近的有两个三角形和一个正方形，这三个点进行投票，于是待分类的圆形就属于三角形样本。而如果 $K=5$，那么离圆形点最近的有两个三角形和三个正方形，这 5 个点进行投票，于是待分类点就属于正方形。

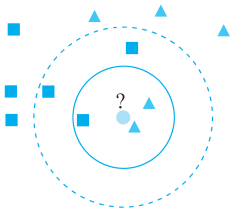

图 5.6　KNN 的理解

简单来说，KNN 可以看成：有那么一堆已经知道分类的数据，如图 5.7 中的 ω_1、ω_2 和 ω_3 三个类别，然后当一个新数据 X_u 进入的时候，就开始跟所有类别中的每个点求距离，然后选择距离最近的 K 个点，看看这几个点属于什么类型。按照少数服从多数的原则，给新数据归类。

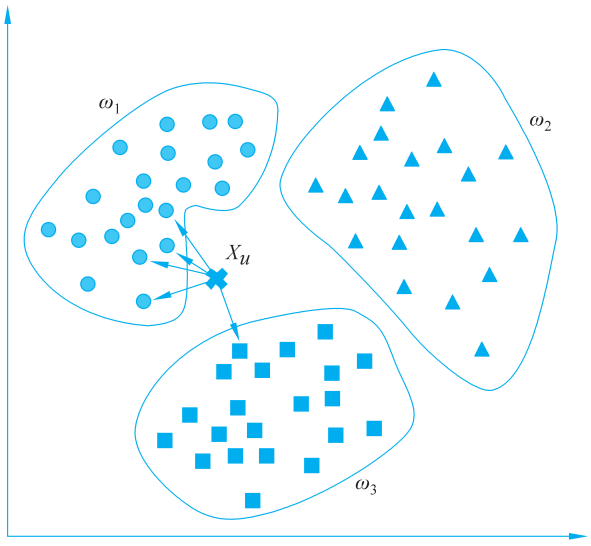

图 5.7　具有三个类别的距离计算示意图

5.2.2　KNN 的应用方法示例

KNN 的应用
微视频

【例 5-2】　先设计一个电影分类数据集（镜头数量纯属虚构），如表 5.1 所示。

表 5.1　电影分类数据集示例

序号	电 影 名 称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
1	瞧这一家子	38	4	4	喜剧片
2	疯狂的石头	62	11	8	喜剧片
3	望子成龙	56	7	9	喜剧片
4	天下无贼	45	4	21	喜剧片
5	太极张三丰	7	6	57	动作片
6	精武英雄	2	3	65	动作片

续表

序号	电 影 名 称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
7	师弟出马	25	4	45	动作片
8	飞鹰计划	6	7	63	动作片
9	牛郎织女	0	36	7	爱情片
10	天仙配	1	32	12	爱情片
11	山楂树	5	37	3	爱情片
12	新步步惊心	8	41	13	爱情片
13	美人鱼	25	4	16	?

数据集中序号 1～12 为已知的电影分类，分为喜剧片、动作片、爱情片三个种类，使用的特征值分别为搞笑镜头、拥抱镜头、打斗镜头的数量。那么来了一部新电影《美人鱼》，它应该属于上述三个电影分类中的哪个类型？下面采用 KNN 进行设计。

(1) 使用 Python 的字典 dict 构造数据集。

```
movie_data = {"瞧这一家子": [38, 4, 4, "喜剧片"],
              "疯狂的石头": [62, 11, 8, "喜剧片"],
              "望子成龙": [56, 7, 9, "喜剧片"],
              "天下无贼": [45, 4, 21, "喜剧片"],
              "太极张三丰": [7, 6, 57, "动作片"],
              "精武英雄": [2, 3, 65, "动作片"],
              "师弟出马": [25, 4, 45, "动作片"],
              "飞鹰计划": [6,7, 63, "动作片"],
              "牛郎织女": [0, 36, 7, "爱情片"],
              "天仙配": [1, 32, 12, "爱情片"],
              "山楂树": [5, 37, 3, "爱情片"],
              "新步步惊心": [8, 41, 13, "爱情片"]}
```

(2) 计算新样本与数据集中所有数据的距离。

新样本是："美人鱼"：[25，4，16，"？片"]。采用欧氏距离进行计算。如 x 为"美人鱼"：[25，4，16，"？片"]，y 为"师弟出马"：[25，4，45，"动作片"]，则两者之间的距离为

$$d=\sqrt{(25-25)^2+(4-4)^2+(16-45)^2}=29.0$$

下面为求与数据集中所有数据的距离代码。

```
x = [25, 4, 16]
KNN = []
for key, v in movie_data.items():
    d = math.sqrt((x[0] - v[0]) ** 2 + (x[1] - v[1]) ** 2 + (x[2] - v[2]) ** 2)
    KNN.append([key, round(d, 2)])
print(KNN)
```

输出结果：

```
[['瞧这一家子', 17.69], ['疯狂的石头', 38.5], ['望子成龙', 31.92], ['天下无贼', 20.62], ['太极张三丰', 44.82], ['精武英雄', 54.14], ['师弟出马', 29.0], ['飞鹰计划', 50.78], ['牛郎织女', 41.59], ['天仙配', 37.09], ['山楂树', 40.72], ['新步步惊心', 40.83]]
```

（3）按照距离大小进行递增排序。

```
KNN.sort(key=lambda dis: dis[1])
```

输出结果：

```
[['瞧这一家子', 17.69], ['天下无贼', 20.62], ['师弟出马', 29.0], ['望子成龙', 31.92], ['天仙配', 37.09], ['疯狂的石头', 38.5], ['山楂树', 40.72], ['新步步惊心', 40.83], ['牛郎织女', 41.59], ['太极张三丰', 44.82], ['飞鹰计划', 50.78], ['精武英雄', 54.14]]
```

（4）选取距离最小的 K 个样本。

选取 $K=5$，有：

```
KNN=KNN[:5]
```

输出为

```
[['瞧这一家子', 17.69], ['天下无贼', 20.62], ['师弟出马', 29.0], ['望子成龙', 31.92], ['天仙配', 37.09]]
```

（5）确定前 K 个样本所在类别出现的频率，并输出出现频率最高的类别。

```
labels = {"喜剧片":0,"动作片":0,"爱情片":0}
for s in KNN:
    label = movie_data[s[0]]
    labels[label[3]] += 1
labels =sorted(labels.items(), key=lambda l: l[1], reverse=True)
print(labels, labels[0][0], sep='\n')
```

输出结果：

```
[('喜剧片', 3), ('动作片', 1), ('爱情片', 1)]
喜剧片
```

因此，KNN 具有以下几个特点。

（1）KNN 属于惰性学习。

这是与急切学习相对应的，因为 KNN 没有显式的学习过程。也就是说，没有训练阶段。从上例可以看出，数据集事先已有了分类和特征值，待收到新样本后直接进行处理。

（2）KNN 的计算复杂度较高。

新样本需要与数据集中每个数据进行距离计算，计算复杂度和数据集中的数据数目 n 成正比。也就是说，KNN 的时间复杂度为 $O(n)$，因此 KNN 一般适用于样本数较少的数据集。

(3) K 取不同值时,分类结果可能会有显著不同。

上例中,如果 K 取值为 $K=1$,那么分类就是动作片,而不是喜剧片。一般 K 的取值不超过 20,上限是 n 的开方。

5.2.3 手写数字识别 KNN 示例分析

【例 5-3】 下面通过一个手写数字识别实例,计算不同 K 值下 KNN 算法的效果。

sklearn 手写数字数据集是 scikit-learn 库中的一个内置数据集,它包含 0～9 的手写数字图像。这些图像通常用于训练各种机器学习算法,特别是分类算法,以识别手写数字。

该数据集由 1797 个 8×8px 的灰度图像组成,每个图像代表一个手写数字,示例如图 5.8 所示。数据集被划分为一个训练集(1500 个样本)和一个测试集(297 个样本)。每个图像被展平为一个 64 维的向量,因此数据集的形状为(n_samples, 64)。数据集还包含每个图像对应的标签(0～9 的整数),标签的数组形状为(n_samples,)。

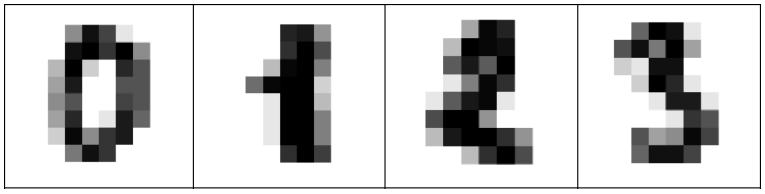

图 5.8 4 个手写数字示例

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors, datasets
from sklearn.model_selection import train_test_split
```

```
def load_classification_data():
    #使用 scikit-learn 自带的手写识别数据集 Digit Dataset
    digits=datasets.load_digits()
    X_train=digits.data
    y_train=digits.target
    #进行分层采样拆分,测试集大小占 1/4
    return train_test_split(X_train, y_train,test_size=0.25,random_state=0,
stratify=y_train)
```

```
#KNN 分类 KNeighborsClassifier 模型
def test_KNeighborsClassifier(*data):
    X_train,X_test,y_train,y_test=data
    clf=neighbors.KNeighborsClassifier()
    clf.fit(X_train,y_train)
    print("Training Score:%f"%clf.score(X_train,y_train))
    print("Testing Score:%f"%clf.score(X_test,y_test))
    #获取分类模型的结果
X_train,X_test,y_train,y_test=load_classification_data() # 调用 test
_KNeighborsClassifier
test_KNeighborsClassifier(X_train,X_test,y_train,y_test)
```

输出结果: