第3章 数据预处理

学习目标

- 了解数据预处理的基本概念
- 掌握并实现常用的数据清洗方法
- 掌握并实现常用的数据降维方法
- 了解敏感数据的识别方法
- 了解敏感信息的去除算法
- 根据案例学习结构化数据和文本数据的预处理

在实际业务处理中，人们获得的数据往往是不完整、不一致的“脏”数据，例如存在缺失值、重复值、不一致等问题，直接使用原数据训练模型可能会影响模型的性能。因此，在训练模型前，往往会对数据进行预处理。数据预处理一方面可以提高数据的质量，另一方面可以使数据中蕴含的信息更容易被挖掘。本章从数据预处理的概述开始，深入浅出地讲解常用的多种数据预处理方法，例如数据清洗、数据降维等。其中，数据清洗主要处理数据中的缺失值、异常点、噪声等；而数据降维指的是降低数据的维度，主要用来减少冗余特征，提升模型训练速度与模型精度。数据降维算法多种多样，本章将介绍几种常用的降维算法，包括主成分分析 [1]、多维缩放降维 [2]、等度量映射降维 [3] 以及局部线性嵌入 [4]。另外，本章将通过实际案例展示如何利用现有的数据预处理方法分别处理结构化数据和文本数据。最后，本章还将介绍不同数据类型的敏感数据识别方法，以及 3 种常用的敏感信息去除算法。

3.1 数据预处理概述

在工程实践中，使用的数据往往不像在科学研究中使用的数据那么干净，普遍存在缺失值、重复值、不一致等问题，这就需要进行数据预处理。数据预处理没有标准的流程，通常因为任务的不同、数据集属性的不同而有所变化。本章将根据数据类型与任务的不同介绍多种数据预处理方法，包括以下部分：

- 数据清洗。数据清洗是数据挖掘工程实践中非常重要的一环。现实世界中的数据往往存在缺失值、重复值、不一致等问题，需要先经过数据清洗才能使用。
- 数据降维。数据降维的主要作用包括减少冗余的特征、提升模型训练速度与模型精度；降维后的特征维度低，便于可视化 (通常为二维或者三维)。
- 结构化数据预处理技术。结构化数据即数据库数据，存储在数据表中，表中的数值特

征可能存在范围不一致的问题，需要进行归一化或者标准化操作。

- 文本数据预处理技术。文本数据是一种非结构化数据，预处理过程包括过滤特殊字符、过滤停止词、词形还原等。
- 隐私保护与数据脱敏。许多数据挖掘任务中的数据包含用户隐私信息，针对不同的隐私敏感性与数据可用性要求，可以使用不同的数据脱敏方法对敏感信息进行去除。

3.2 数据清洗

本节主要介绍数据清洗的一些方法。在这之前，我们先思考一个问题——为什么要进行数据清洗？理想中的数据应该是完美的，数据质量非常高，不存在异常点、缺失值，也没有噪声数据。但实际上，现实中的数据可能充斥着大量的噪声，可能包含许多缺失值，或者存在由于人工输入错误而导致的异常点，这些情况的出现使得我们难以挖掘出有效的信息。因此，在数据挖掘中需要进行数据清洗，数据清洗不仅可以提高数据的质量，还可以使数据更适合挖掘。

3.2.1 缺失值处理

无论是项目数据还是竞赛数据，都会遇到数据缺失的情况。有些做法比较简单粗暴，即直接删除缺失的数据或者用 0 值或其他特殊值进行填充。那么，到底如何处理数据缺失的情况呢？其实，根据不同的数据，应该采取不同的策略。

首先要进行数据分析，查看数据缺失的分布情况。这里采用的数据集来源于 Kaggle 竞赛平台的电信用户流失数据，该数据集共有 13259 条样本、79 维特征。接下来针对其中一维特征进行分析。

```
import pandas as pd
data = pd.read_csv('sampletelecomfinal.csv') # 读取数据
print(data.shape)
print(features['mou_Mean].isnull().sum())
```

输出结果如下：

```
(13259, 79)
mou_Mean 26
```

可以看到，13259 条样本中，'mou_Mean'列只有 26 条数据缺失，占总体数据约 0.2%，此时数据缺失比较少。因此，删除缺失数据对整体的数据分布几乎没有影响。如果数据缺失比较多，而且该列数据又比较重要，直接删除缺失数据势必会影响模型的性能。因此，当数据缺失比较多时，通常采用以下方法进行处理：

- 均值或中位数填充法。如果数据的分布是比较均匀的，那么可以采用均值进行填充；相反，如果数据分布比较倾斜，那么可以采用中位数进行填充。该填充方法的处理方式比较简单，也不会减少样本。

- 随机插补法。从数据集中随机选择一个不含缺失值的样本替换缺失的样本。
- 热平台插补法 [5]。从不含缺失值的样本集中检索出与包含缺失值的样本相似的样本，然后利用该样本的观测值插补缺失值。该方法比较简单，准确率也比较高。但是当变量数量较多时，难以检索出与包含缺失值的样本相似的样本，这时通常可以按照一些变量将数据进行分层，然后在层中使用均值进行插补。
- 多重插补法 [6]。通过对缺失的数据构造多个插补值可以生成多个完整的数据集，然后对每个完整数据集分别使用相同的方法进行处理，最终分析这些数据集并总结分析结果，以得到对缺失变量的估计。
- 建模法。根据数据集中的其他数据列训练模型，对缺失数据进行预测。

3.2.2 异常点检测

异常点就是通常说的“离群点”“孤立点”。异常点的出现可能是由于人工输入错误而导致的。在处理异常点之前，需要先进行异常点检测。下面介绍几种检测异常点的常用方法。

1. 统计分析

获取数据之后，可以先简单地对数据做一个统计分析，例如统计属性的最大/最小值。最大/最小值可以用来判断该属性的取值是否在合理范围内，如某个用户的身高为 −30cm，这显然是不合理的。

```
import pandas as pd
data = pd.read_csv('sampletelecomfinal.csv') # 读取数据
data[['mou_Mean', 'totmrc_Mean', 'rev_Range', 'mou_Range']].describe()
    #生成描述性统计数据
```

输出结果如下：

```
        mou_Mean      totmrc_Mean     rev_Range      mou_Range
count 13233.000000  13233.000000  13233.000000  13233.000000
mean    529.348409     46.960394     44.696770    382.439696
std     546.503814     24.145539    139.109904    569.885497
min       0.000000     -6.167500      0.000000      0.000000
25%     160.250000     30.000000      1.980000    114.000000
50%     365.250000     44.990000     15.990000    246.000000
75%     719.250000     59.990000     57.360000    486.000000
max   12206.750000    399.990000  13740.540000  43050.000000
```

输出结果统计数据集的集中趋势，分散和行列的分布情况 (不包括 NaN 值)。默认情况下仅返回数值类型列的统计结果，结果指标包括数量 (count)、平均值 (mean)、方差 (std)、最小值 (min)、第 25 百分位、中位数 (第 50 百分位)、第 75 百分位和最大值 (max)。

2. 3α 原则

如果数据的分布遵循正态分布，那么在 3α 原则下，那些偏离平均值 3 倍标准差的值就是异常值。因为在数据遵循正态分布的情况下，偏离平均值 3 倍标准差的值出现的概率为 $P(|x-u|>3\alpha)\leqslant 0.003$，属于极小概率事件，因此出现在该范围内的值可以视为异常点。在数据不遵循正态分布的情况下，也可以根据偏离平均值多少倍的标准差判断异常点。

3. 基于模型检测

基于模型检测的方法是指创建一个数据模型以拟合数据对象，而那些不能与模型完全拟合的对象就是异常点。如果模型是若干簇组成的集合，那么不属于任何簇的数据对象就是异常点。使用回归模型时，那些与预测值相对较远的对象就是异常点。

3.2.3 异常点处理

前面介绍了几种异常点检测方法，下面介绍处理异常点的方法。

（1）删除异常点。对于非常明显且数量较少的异常点，可以直接删除。

（2）不处理。如果算法对异常点不敏感，则不需要处理；如果算法对异常点敏感，则最好不要使用该算法，例如 K 均值聚类算法 (K-means Clustering Algorithm)、K 最近邻 (K-Nearest Neighbors，KNN) 算法等。

（3）平均值替代。使用该属性列的平均值代替异常点，这种做法不会减少样本，处理方式也比较简单。

（4）视为缺失值。参考处理缺失值的方法处理异常点。

3.2.4 重复数据处理

获取的数据中可能存在一些重复数据，这时就需要进行过滤。首先创建数据集：

```
# 创建数据集
import pandas as pd
data = pd.DataFrame({'name': ['Ming', 'Hong', 'Li', 'Feng', 'Hong', 'Ming'], 'score'
    : [90,85,95,85,85,90]})
print(data)
```

输出结果如下：

```
index name   score
0     Ming   90
1     Hong   85
2     Li     95
3     Feng   85
4     Hong   85
5     Ming   90
```

接下来查看数据是否包含重复行：

```
# 调用DataFrame的duplicated函数返回一个布尔型Series，表示该行是否重复
print(data.duplicated())
```

输出结果如下：

```
0    False
1    False
2    False
3    False
4    True
5    True
dtype: bool
```

可以看到第 5 条和第 6 条数据是重复的，需要进行过滤：

```
# 调用DataFrame的drop_duplicates函数返回去除重复数据的DataFrame
data.drop_duplicates()
print(data)
```

输出结果如下：

```
index   name   score
0       Ming   90
1       Hong   85
2       Li     95
3       Feng   85
```

3.2.5 噪声处理

噪声是指机器无法解析的无意义数据，它可能由于数据收集错误、数据录入错误等而产生。噪声主要包括错误值或偏离期望的孤立点值，噪声的存在给数据的挖掘与分析造成了一定的干扰，因此需要清洗噪声。下面介绍几种处理噪声的方法。

（1）分箱法。分箱法就是将数据按照一定的规则放入不同的箱中，然后对每个箱中的数据进行分析，并根据分析结果对箱中的数据进行处理。这里所说的分箱规则是根据数据的条数进行分箱，这样可以确保每个箱中的数据量是一样的；也可以定义区间进行分箱，将数据放入对应区间的箱中。箱中数据的处理方法根据取值的不同可以划分为以下 3 种。

- 根据箱的均值平滑：计算每个箱的平均值，然后用该平均值替换箱中的数据。
- 根据箱的中位数平滑：计算每个箱的中位数，然后用该中位数替换箱中的数据。
- 根据箱的边界平滑：计算每个箱的最大和最小边界，然后对于箱中的每个数据用最近的边界进行替换。

（2）回归法。回归法就是利用函数建模变量与变量之间的关系，然后根据这个函数预测其中一个变量。回归法分为两种：单线性回归和多线性回归。单线性回归就是利用一条

直线拟合两个变量，从而使用已知的其中一个变量预测另外一个变量。多线性回归是单线性回归的扩展，它利用一个多维面拟合数据。

3.3 数据降维

3.3.1 数据降维概述

在机器学习领域中，数据降维是指通过一个映射函数将原始的高维数据映射到低维空间中。数据降维的核心在于学习一个映射函数 $f: x \to y$，其中，x 是原始数据的向量表示，y 是原始数据经过映射函数后的向量表示，通常向量 y 的维度小于向量 x 的维度。数据降维的作用以及使用降维后的数据进行数据挖掘的优势在于：

（1）原始高维数据可能包含大量冗余信息和噪声信息，这些信息的存在给数据的挖掘与分析造成了误差，降低了模型的性能。数据降维可以减少冗余信息和噪声信息，从而提升模型的性能。

（2）当数据的特征维度较高时，特征间往往存在多重共线性，即特征属性之间存在相互关联关系。数据降维能够一定程度上消除多重共线性的影响，提升机器学习模型的鲁棒性与泛化性。

（3）数据降维可以减缓高维特征中稀疏性的问题，帮助机器学习模型更好地找到有意义的数据特征。

（4）数据降维可以减少机器学习模型在训练高维特征数据时的训练时长，提高训练速度。

在很多算法中，降维算法都是数据预处理的一部分，下面介绍主成分分析（Principal Component Analysis，PCA）、多维缩放（Multiple Dimensional Scaling，MDS）降维、等度量映射（Isometric Mapping，Isomap）降维以及局部线性嵌入（Locally Linear Embedding，LLE）等多种主流的数据降维算法及其代码实现。

3.3.2 主成分分析降维

主成分分析是最常用的一种数据降维方式，它通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量，转换后的这组变量称为主成分。算法步骤如下：

输入：样本集 $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$；样本数 N；原始特征维度 n；低维空间维数 n'。

输出：样本集在低维空间中的矩阵 $\boldsymbol{Z} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_N)$；映射矩阵 $\boldsymbol{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{n'})$；样本均值向量$\boldsymbol{\mu}$。

算法步骤：

计算样本均值向量 $\boldsymbol{\mu}$：

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{j=1}^{N}\boldsymbol{x}_j \tag{3.1}$$

对所有样本进行中心化操作：

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \boldsymbol{\mu} \tag{3.2}$$

计算样本的协方差矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$；
对协方差矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 做特征值分解；
对求得的特征值排序，取最大的 n' 个特征值对应的特征向量 $\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{n'}$，
构造投影矩阵 $\boldsymbol{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{n'})$；
通过映射矩阵 $\boldsymbol{W}$ 将所有样本映射到低维空间，得到样本集在低维空间中的矩阵

$$\boldsymbol{Z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_N\}$$

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}^{\mathrm{T}} \tag{3.3}$$

在处理新样本时，只需要将新样本通过样本均值向量$\boldsymbol{\mu}$ 进行中心化，再与映射矩阵 $\boldsymbol{W}$ 相乘，即可得到降维后的样本数据。代码实现如下：

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import NullFormatter
from sklearn import datasets
from sklearn import decomposition

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False    #用来正常显示负号

iris = datasets.load_iris() # 加载Iris数据集
X = iris.data
y = iris.target
y = np.choose(y, [1, 2, 0]).astype(np.float)

fig = plt.figure(1, figsize=(15, 8))        #初始化输出画布
plt.clf()
ax = fig.add_subplot(121, projection='3d')  #设置画布中的第一个子图
plt.cla()

#执行PCA降维
pca = decomposition.PCA(n_components=3)
X_t = pca.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              X_t[y == label, 2],
              label=name,
              color=color,
              edgecolor='k')
```

```
plt.legend(prop={'size': 15})
ax = fig.add_subplot(122) # 设置画布中的第二个子图

#执行PCA降维
pca = decomposition.PCA(n_components=3)
X_t = pca.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              label=name,
              color=color,
              edgecolor='k')
plt.show() # 输出画布
```

PCA 运行结果如图 3.1 所示。

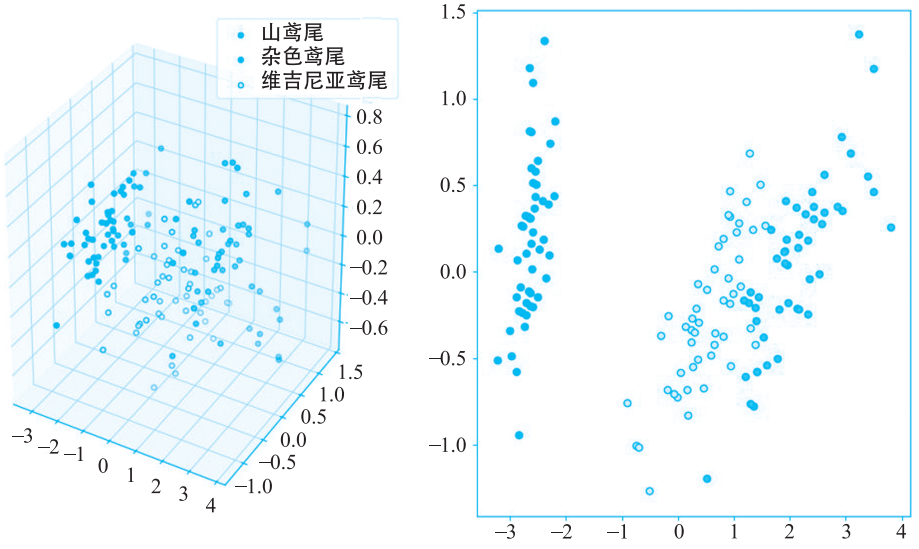

图 3.1 PCA 运行结果

3.3.3 多维缩放降维

多维缩放降维要求原始空间中的样本之间的距离在低维空间中得到保持。首先计算距离矩阵 $\boldsymbol{D} \in \mathbb{R}^{N\times N}$，其第 i 行第 j 列的元素 $d_{i,j}$ 表示样本 x_i 与 x_j 的距离。算法步骤如下。

- 输入：样本集 $X=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$；距离矩阵 $D\in\mathbb{R}^{N\times N}$；低维空间数 n'。
- 输出：样本集在低维空间中的矩阵 $\boldsymbol{Z}$。

- 算法步骤：
- 根据下列式子计算 $d_{i,\cdot}^2, d_{j,\cdot}^2, d_{\cdot,\cdot}^2$。

$$d_{i,\cdot}^2 = \frac{1}{N}\sum_{j=1}^{N} d_{ij}^2 \tag{3.4}$$

$$d_{j,\cdot}^2 = \frac{1}{N}\sum_{i=1}^{N} d_{ij}^2 \tag{3.5}$$

$$d_{\cdot,\cdot}^2 = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}^2 \tag{3.6}$$

- 计算得到矩阵 $\boldsymbol{B}$，其中第 i 行第 j 列的值 $b_{i,j}$ 为

$$b_{i,j} = \frac{d_{i,\cdot}^2 + d_{j,\cdot}^2 - d_{\cdot,\cdot}^2 + d_{ij}^2}{2} \tag{3.7}$$

- 根据式 (3.8)，对矩阵 $\boldsymbol{B}$ 进行特征值分解。

$$\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}} \tag{3.8}$$

- 对求得的特征值排序，取 $\boldsymbol{\Lambda}$ 为 n' 个最大特征值构成的对角矩阵, 和对应的特征向量 $\widetilde{\boldsymbol{V}}$，则

$$\boldsymbol{Z} = \widetilde{\boldsymbol{\Lambda}}^{1/2}\widetilde{\boldsymbol{V}}^{\mathrm{T}} \in \mathbb{R}^{n' \times N} \tag{3.9}$$

代码实现如下：

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import NullFormatter
from sklearn import datasets
from sklearn import manifold

plt.rcParams['font.sans-serif'] = ['SimHei']      #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False        #用来正常显示负号

iris = datasets.load_iris() # 加载Iris数据集
X = iris.data
y = iris.target
y = np.choose(y, [1, 2, 0]).astype(np.float)

fig = plt.figure(1, figsize=(15, 8))        #初始化输出画布
plt.clf()
ax = fig.add_subplot(121, projection='3d') #设置画布中的第一个子图
```

```
plt.cla()

#执行MDS降维
mds = manifold.MDS(max_iter=100, n_components=3, n_init=1)
X_t = mds.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              X_t[y == label, 2],
              label=name,
              color=color,
              edgecolor='k')

plt.legend(prop={'size': 15})
ax = fig.add_subplot(122) # 设置画布中的第二个子图

#执行MDS降维
mds = manifold.MDS(max_iter=100, n_components=3, n_init=1)
X_t = mds.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              label=name,
              color=color,
              edgecolor='k')
plt.show() # 输出画布
```

MDS 算法运行结果如图 3.2 所示。

3.3.4 等度量映射降维

等度量映射降维是最早的流形方法之一，Isomap 可以看作多维缩放分析或核主成分分析的拓展。Isomap 寻求一种低维嵌入，以保持所有样本点之间的测地距离 (geodesic distances)。测地距离相比于两点间的直线距离更能反映样本在高维空间的分布。Isomap 利用流形在局部上与欧几里得空间同胚的性质，使用每个点与其相邻的点构建近邻连接图，通过计算近邻连接图上两点间的最短路径而近似得到流形上的测地距离。算法步骤如下：

- 输入：样本集 $X = \{x_1, x_2, \cdots, x_N\}$；低维空间维数 n'；近邻参数 k。
- 输出：样本集在低维空间中的矩阵 $\boldsymbol{Z}$。
- 算法步骤:

- 对样本集中的每个样本点 x_i，计算它的 k 近邻；同时将 x_i 与它的 k 近邻的距离设置为欧几里得距离，与其他点的距离设置为无穷大。
- 通过最短路径算法计算任意两个样本点之间的距离，获得距离矩阵 $\boldsymbol{D} \in \mathbb{R}^{N\times N}$，其中，第 i 行第 j 列表示样本 x_i 与样本 x_j 之间的近似测地距离。
- 将样本集 X 与距离矩阵 $\boldsymbol{D}$ 作为输入，调用多维缩放算法获得样本集在低维空间中的矩阵 $\boldsymbol{Z}$。

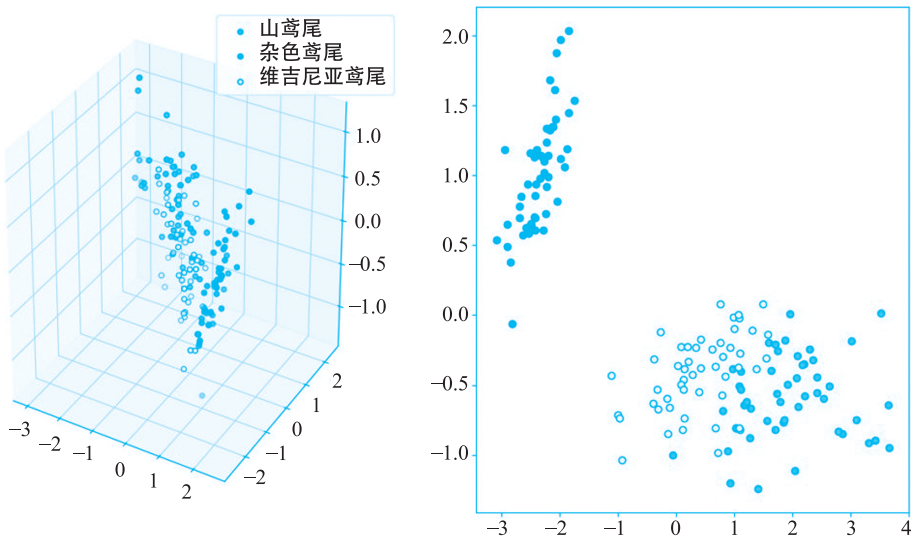

图 3.2　MDS 算法运行结果

相比于主成分分析降维方法，MDS 算法和 Isomap 算法在处理新样本时需要计算所有样本间的距离以得到距离矩阵，再重新计算所有样本的降维矩阵。这样的做法运算量太大，不适合用在对延迟要求较高的场景中。可行的解决方法是：将原样本集看作数据的特征 X，将降维后的样本集看作数据的标签 Y，通过训练一个回归模型 $Y = f(X)$，当处理新样本时，将新样本输入回归模型即可得到一个预测的近似降维结果。代码实现如下：

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import NullFormatter
from sklearn import datasets
from sklearn import manifold

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False   #用来正常显示负号

iris = datasets.load_iris() # 加载Iris数据集
X = iris.data
y = iris.target
y = np.choose(y, [1, 2, 0]).astype(np.float)
```

```
fig = plt.figure(1, figsize=(15, 8))        #初始化输出画布
plt.clf()
ax = fig.add_subplot(121, projection='3d') #设置画布中的第一个子图
plt.cla()

#执行Isomap降维
isomap = manifold.Isomap(n_neighbors=10, n_components=3)
X_t = isomap.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              X_t[y == label, 2],
              label=name,
              color=color,
              edgecolor='k')

plt.legend(prop={'size': 15})
ax = fig.add_subplot(122) # 设置画布中的第二个子图

#执行Isomap降维
isomap = manifold.Isomap(n_neighbors=10, n_components=3)
X_t = isomap.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              label=name,
              color=color,
              edgecolor='k')
plt.show() # 输出画布
```

Isomap 运行结果如图 3.3 所示。

3.3.5 局部线性嵌入降维

局部线性嵌入降维维护的是样本集中每个样本点与其邻域内其他样本点的线性关系，它可以看作一系列局部主成分分析，通过全局比较找出最佳非线性嵌入。算法步骤如下。

- 输入：样本集 $B=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$；低维空间维数 n'；近邻参数 k。
- 输出：样本集在低维空间中的矩阵 $\boldsymbol{Z}$。

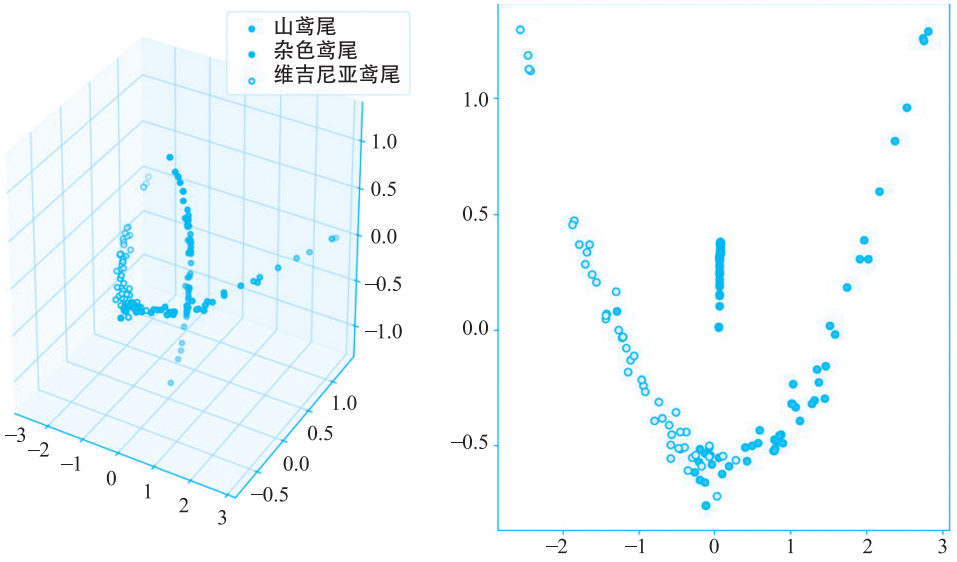

图 3.3　Isomap 运行结果

算法步骤：

对于样本集中的每个点 $\boldsymbol{x}_i, i = 1, 2, \cdots, N$ 执行下列操作。

对于样本 $\boldsymbol{x}_i$ 的 k 个近邻集合 Q_i，根据 Q_i 中的每个样本 x_j 计算：

$$w_{i,j} = \frac{\sum\limits_{k \in Q_i} C_{j,k}^{-1}}{\sum\limits_{l,s \in Q_i} C_{l,s}^{-1}}$$

$$C_{j,k} = (\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}}(\boldsymbol{x}_i - \boldsymbol{x}_k)$$

对于样本集中其他所有不在集合 Q_i 的样本 $x_j, w_{i,j} = 0$。

构建矩阵 $\boldsymbol{W}$，其中，第 i 行第 j 列的值为 $w_{i,j}$。

计算 $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{W})^{\mathrm{T}}(\boldsymbol{I} - \boldsymbol{W})$。

对 $\boldsymbol{M}$ 进行特征值分解，取其最小的 n' 个特征值对应的特征向量，即得到样本集在低维空间中的矩阵 $\boldsymbol{Z}$。

代码实现如下：

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import NullFormatter
from sklearn import datasets
from sklearn import manifold

plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False   #用来正常显示负号
```

```
iris = datasets.load_iris() # 加载Iris数据集
X = iris.data
y = iris.target
y = np.choose(y, [1, 2, 0]).astype(np.float)

fig = plt.figure(1, figsize=(15, 8))       #初始化输出画布
plt.clf()
ax = fig.add_subplot(121, projection='3d') #设置画布中的第一个子图
plt.cla()

#执行LLE降维
lle = manifold.LocallyLinearEmbedding(n_components=3, n_neighbors=10)
X_t = lle.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              X_t[y == label, 2],
              label=name,
              color=color,
              edgecolor='k')

plt.legend(prop={'size': 15})
ax = fig.add_subplot(122) # 设置画布中的第二个子图

#执行LLE降维
lle = manifold.LocallyLinearEmbedding(n_components=2, n_neighbors=10)
X_t = lle.fit_transform(X)
# 将降维后的数据点映射到画布上
for name, label, color in [('山鸢尾', 0, 'red'), ('杂色鸢尾', 1, 'blue'), ('维吉尼亚
    鸢尾', 2, 'yellow')]:
   ax.scatter(X_t[y == label, 0],
              X_t[y == label, 1],
              label=name,
              color=color,
              edgecolor='k')
plt.show() # 输出画布
```

LLE 运行结果如图 3.4 所示。

3.3.6 降维效果比较

本节将对上文提到的 4 个降维算法进行降维效果对比。具体对比方案如下：

- 采用 Iris 分类数据集作为测试数据集，原始特征维度为 4，类别标签为 Setosa、

Versicolor、Virginica，分别对应 3 种鸢尾花类型。

- 分别用 4 种降维方法对其进行降维，且降维维度一致 ($n'=2$)。
- 对降维后的数据采用同一分类算法分别训练 4 次，并测试分类性能。

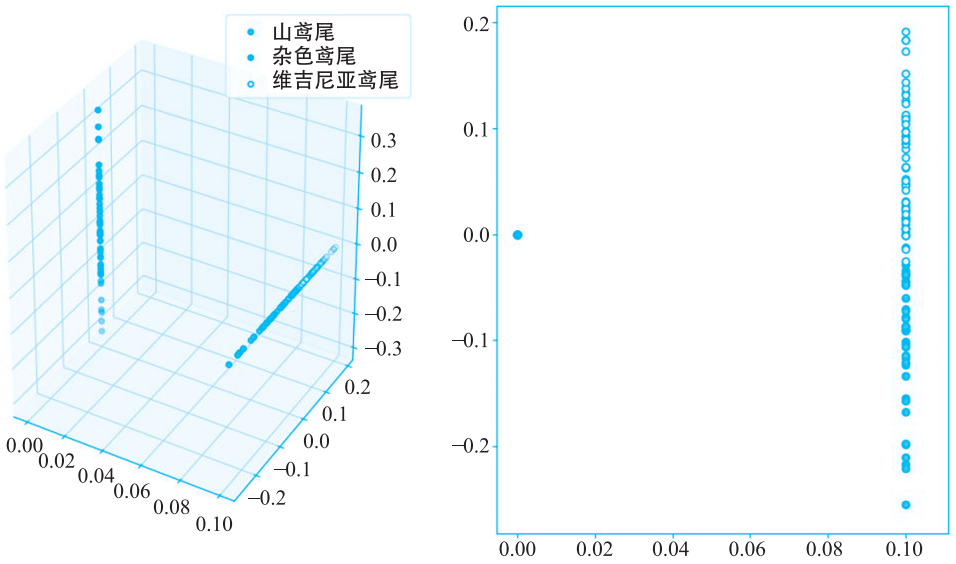

图 3.4　LLE 运行结果

代码实现如下：

```
import numpy as np
from sklearn import datasets
from sklearn import manifold
from sklearn import decomposition
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
np.random.seed(50)
plt.rcParams['font.sans-serif'] = ['SimHei']  #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False     #用来正常显示负号

# 将数据点映射到画布上
def draw(ax, X_t, y):
    for name, label, color in [('山鸢尾',0,'red'),('杂色鸢尾',1,'blue'), ('维吉尼亚鸢
        尾', 2, 'yellow')]:
        ax.scatter(X_t[y == label, 0],
                   X_t[y == label, 1],
                   label=name,
                   color=color,
                   edgecolor='k')
```

```
fig = plt.figure(1, figsize=(15, 8)) #初始化画布
iris = datasets.load_Iris()          #加载Iris数据集
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
    state=20) # 按照八二比例分割出训练集和测试集

classifier = RandomForestClassifier(n_estimators=100) # 设置评测用的分类器

# 测试原始数据的分类效果
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print('原始测试集分类准确率为{:.2f}$\%$'.format(100*accuracy_score(y_true=y_test, y_
    pred=y_pred)))

# 设置各个降维算法的配置
params = {'主成分分析': {'param': {'n_components': 2}, 'model': decomposition.PCA},
          '多维缩放':{'param':{'max_iter':100,'n_components': 2,'n_init': 1}, 'model
              ': manifold.MDS},
          '等度量映射':{'param':{'n_neighbors':10,'n_components': 2}, 'model':
              manifold.Isomap},
          '局部线性嵌入': {'param': {'n_neighbors': 10, 'n_components': 2}, 'model':
               manifold.LocallyLinearEmbedding}}

for index, model_name in enumerate(params.keys()):
    param = params[model_name]['param']
    model = params[model_name]['model'](**param)
    X_t = model.fit_transform(X) #执行数据降维
    X_train_t,X_test_t,y_train,y_test=train_test_split(X_t, y, test_size=0.2, random_
        state=20) #对降维后的数据进行分割，得到训练集和测试集
    ax = fig.add_subplot(241+index) # 设置该降维算法对应的第一个子图
    plt.title('{}-训练集'.format(model_name)) # 设置子图标题
    draw(ax, X_train_t, y_train) # 刻画训练集的数据点
    plt.legend()
    ax = fig.add_subplot(245+index) # 设置该降维算法对应的第二个子图
    plt.title('{}-测试集'.format(model_name)) # 设置子图标题
    draw(ax, X_test_t, y_test) # 刻画测试集的数据点
    classifier = RandomForestClassifier(n_estimators=100)
    classifier.fit(X_train_t, y_train) # 利用降维后的数据训练分类器
    y_pred = classifier.predict(X_test_t) # 测试分类器效果
    print(''使用降维模型 '{}' 后，测试集分类准确率为'{:.2f}$\%$'.
        format (model_name, 100*accuracy_score (y_true=y_test, y_pred=y_pred))) #输出
            分类结果
```

输出结果如下：

```
原始测试集分类准确率为 90.00%
使用降维模型 PCA 后，测试集分类准确率为 96.67%
使用降维模型 MDS 后，测试集分类准确率为 96.67%
使用降维模型 Isomap 后，测试集分类准确率为 90.00%
使用降维模型 LLE 后，测试集分类准确率为 86.67%
```

从上述实战中可以看出，不同的降维方法会取得不同的降维效果（图 3.5），并且在分类任务测试中可以看到，在 Iris 数据集上，PCA 和 MDS 降维后的数据能取得了更高的分类准确率。针对不同的数据集选取不同的降维方法，可以达到提高模型性能的效果。

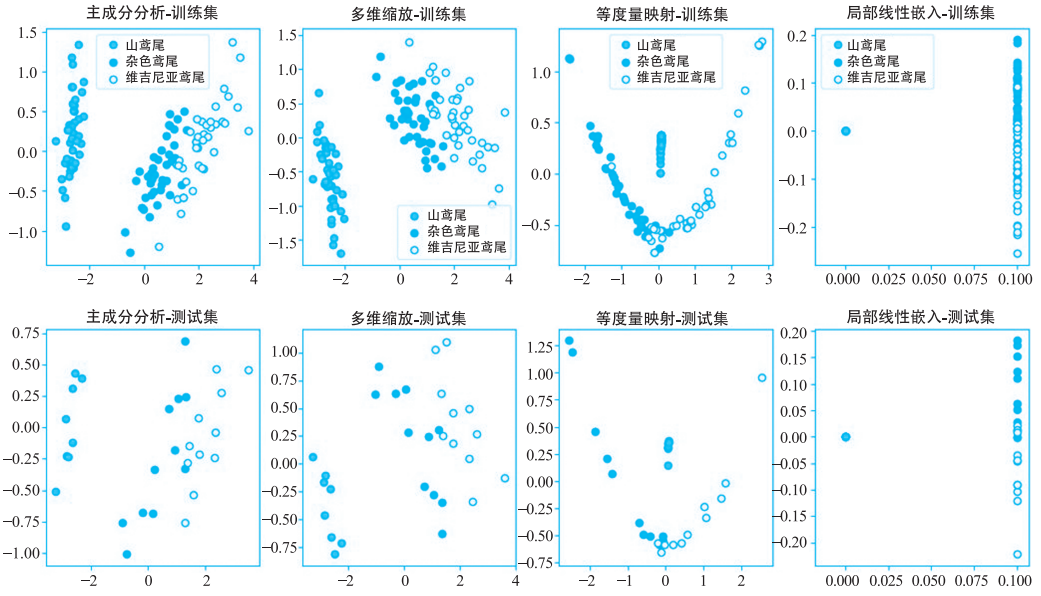

图 3.5　不同降维方法的数据分布比较

3.4　结构化数据预处理技术

结构化数据通常是由二维数据表的结构呈现的，其每行可以看作单个数据点或观察值，每列则作为不同属性的字段。对于结构化数据，其数据类型大致可以分为数值属性和类别属性，数值属性在输入模型之前需要进行标准化、归一化等一系列操作；类别属性需要进行编码后才可被模型处理。下面介绍常用的结构化数据预处理技术。

3.4.1　数据清洗

由于原始数据不同属性的数据类型不同，其数值也相差较大，因此需要先对数据进行清洗才能被模型使用。目前有很多开源的数据分析处理工具，这里使用 pandas 进行数据预处理。

1. 编码

对于类别属性，其数据类型无法被模型直接接收，在使用前需要进行编码操作。对于字符串类型，可根据内容将其编码为不同的类别表示；对于时间类型，可从中提取出年、月、日等数值或将其转换为时间戳作为模型输入。sklearn 工具包提供了编码方法，可直接将类别映射到连续整数中。

```
# 对属性进行编码
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
encoder.fit(['a', 'b', 'c', 'd']) # 训练LabelEncoder
label = encoder.transform(['a', 'b', 'a', 'c', 'b', 'd'])
    #使用训练好的LabelEncoder对原数据进行编码
print(label)
```

输出结果如下：

```
[0, 1, 0, 2, 1, 3]
```

从输出结果可以看到，LabelEncoder 可以将类别分配一个从 0 到类别数 −1 之间的编码，如类别 'a'转换成了数字 0。这样处理过后，使得后续的数据分析变得更容易，且方便模型训练。

2. 标准化

数据标准化是指将数值属性的数据按照一定的比例进行缩放，使其值落在特定的区间内。在对不同类型的数据进行操作的过程中，往往需要将数值转换为相同的范围及量纲，以便后续进行比较或线性计算。标准化的方法有很多，最典型的是 Z-score 标准化方法，该方法令原始数据减去其均值再除以标准差，使其服从标准正态分布。具体代码如下：

```
#Z-score标准化
import pandas as pd
data = pd.DataFrame([1, 2, 3, 2, 1])
mean = data.mean()        #计算均值
std = data.std()          #计算方差
data = (data - mean) / std # 根据均值和方差标准化原数据
print(data)
```

输出结果如下：

```
[-0.956183, 0.239046, 1.434274, 0.239046, -0.956183]
```

3. 归一化

归一化是标准化的一种典型方法，它通过缩放将数据映射到 [0,1] 区间，以提升模型的收敛速度和精度。其中，常见的方法是离差标准化 (min-max 标准化)，它通过使用数据中的最大值减去最小值得到标准化公式中的分母，数据的当前值减去最小值得到公式中的分子，从而得到数据处理后的结果。具体代码如下：

```
#Z-score归一化
import pandas as pd
data = pd.DataFrame([1, 2, 3, 2, 1])
max = data.max() # 得到数据中的最大值
min = data.min() # 得到数据中的最小值
data = (data - min) / (max - min) # 根据最大值和最小值归一化原数据
print(data)
```

输出结果如下：

```
[0.0, 0.5, 1.0, 0.5, 0.0]
```

4. apply 函数

数据清洗过程中，可能需要对数据按行或者按列进行操作，例如对行或列的数据进行运算。下面介绍对列数据进行平方以及列之间进行求和的例子，具体代码如下：

```
# 按列apply
import pandas as pd
data = pd.DataFrame([[1, 1, 1], [2, 2, 2], [3, 3, 3]],
      columns=['c1', 'c2', 'c3']) # 数据初始化并指定列名
print(data)
```

输出结果如下：

```
   c1  c2  c3
0   1   1   1
1   2   2   2
2   3   3   3
```

apply 函数接收列名作为参数。本例对 c1 列进行平方操作，并使用 lambda 匿名函数实现平方运算。

```
data['c1'] = data['c1'].apply(lambda x: x * x)
print(data)
```

输出结果如下：

```
   c1  c2  c3
0  1   1   1
1  4   2   2
2  9   3   3
```

将 c2 列和 c3 列逐元素求和，并将结果保存为 c4 列。

```
data['c4'] = data.apply(lambda x: x['c2'] + x['c3'], axis=1)
    # 使用lambda匿名函数实现列之间逐元素求和
print(data)
```

输出结果如下：

```
   c1  c2  c3  c4
0  1   1   1   2
1  4   2   2   4
2  9   3   3   6
```

3.4.2 分组与聚合

除了对整个数据集进行操作外，还可以选择不同的属性对数据进行分组，在每个组中对数据进行一系列操作。最后，可将结果进行聚合，将每组产生的结果进行合并。具体代码如下：

```
# 分组与聚合
import pandas as pd
data = pd.DataFrame([[1, 2, 1],
                     [2, 1, 2],
                     [2, 3, 1],
                     [1, 3, 2]], columns=['c1', 'c2', 'c3'])    # 数据初始化并指定列名
print(data)
```

输出结果如下：

```
   c1  c2  c3
0  1   2   1
1  2   1   2
2  2   3   1
3  1   3   2
```

将数据按照 c1 列进行组合，可以发现整个数据会按照 c1 列取值为 1 和 2 划分为两组：