数据分析与可视化

3.1 Python 数据分析与可视化入门

Python 因其简洁的语法和强大的库支持，已成为大数据分析领域的热门选择。Python 不仅在数据处理和分析中提供了丰富的库支持，还在机器学习和深度学习领域提供了强大的算法和工具。从数据处理和分析到机器学习和深度学习，Python 用途广泛。

3.1.1 Python 用于数据分析的背景

在过去的几十年中，Python 已经从一个简单的脚本语言发展成为数据科学和大数据分析的核心工具。这一转变得益于 Python 的灵活语法、庞大的社区支持、丰富的库生态系统，以及开源和跨平台的特性。Python 在全球范围内迅速普及，成为数据分析、人工智能研究、Web 开发、科学计算和数据可视化等领域的首选语言。

3.1.2 数据处理和分析工具

Python 提供了丰富的数据处理和分析工具，如 Pandas、NumPy 和 Scikit-learn。这些工具使得数据处理和分析变得高效和简单。

Pandas：Pandas 是一个专为数据处理和分析而生的开源库。它提供了两种核心数据结构：Series（单维）和 DataFrame（双维），让操作结构化数据变得轻而易举。Pandas 的主要功能包括高效管理缺失数据、数据汇总与筛选、简便的数据集合并与连接，以及从 CSV、Excel、SQL、JSON 等格式导入和导出数据。

使用场景：适用于需要快速读取、写入和处理大型表格数据，进行数据清洗和转换的场合。例如，读取 Excel 表格，提取数据并进行清洗，或者将 JSON 数据转换为 CSV 文件等。

以下是调用 Pandas 库进行简单数据处理的代码。

```
    import pandas as pd
  #创建一个 DataFrame
  data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
          'Age': [25, 30, 35, 40, 45],
          'City': ['New York', 'Paris', 'London', 'Berlin', 'Tokyo']}
  df = pd.DataFrame(data)
  #数据清洗: 过滤数据
  #例如,过滤出年龄大于 30 的数据
  filtered_data = df[df['Age'] > 30]
  print(filtered_data)
python pandas_example.py
output:
      Name  Age     City
2  Charlie    35    London
3    David   40    Berlin
4      Eve   45    Tokyo
```

代码示例中,使用 Pandas 创建了一个 DataFrame 对象,包含三列数据: Name、Age 和 City。然后使用了过滤数据的方法,过滤出年龄大于 30 的数据。最后,将过滤后的数据打印出来。

NumPy: NumPy 是 Python 中进行数值计算的基础库,能够处理大型的多维数组和矩阵,并提供了丰富的数学函数来对它们进行操作。NumPy 是 Pandas 等更高级库的基石,是处理数字或大数据集时的首选库。

使用场景: 适用于需要进行数值计算、矩阵运算和科学计算的场合,如机器学习、图像处理、信号处理、科学模拟等。

以下调用 NumPy 库进行简单数据处理的代码。

```
  import numpy as np
  #创建一个 NumPy 数组
  arr = np.array([[1, 2, 3], [4, 5, 6]])
  #执行逐元素操作
  arr_squared = arr ** 2
  print(arr_squared)
```

代码示例中,使用 NumPy 创建了一个二维数组,然后使用了逐元素操作来计算每个元素的平方。

```
'''bash
python numpy_example.py
    output:
    [[ 1  4  9]
    [16 25 36]]
'''
```

3.1.3 数据可视化

Python 拥有多个强大的数据可视化库,如 Matplotlib、Seaborn 和 Plotly。

Matplotlib: Matplotlib 是一个功能强大的可视化库,支持在 Python 中绘制多种静态、

动态和交互式图表。它是绘制条形图、折线图、散点图和直方图等图形的首选工具。

使用场景：适用于需要快速创建基本图表、自定义图形和调整图形样式的场合。

以下是调用 Matplotlib 库进行简单数据处理的代码。

```
import matplotlib.pyplot as plt
#示例数据
names = ['A', 'B', 'C', 'D']
values = [10, 20, 30, 40]
#创建柱状图
plt.bar(names, values)
plt.title('Bar Chart Example')
plt.xlabel('Names')
plt.ylabel('Values')
plt.show()
```

代码示例中，使用了 Matplotlib 库中的 bar 函数来创建一个柱状图，如图 3.1 所示。首先，定义了一个名为 names 的列表，包含 4 个字符串：A、B、C 和 D。然后，定义了一个名为 values 的列表，包含 4 个整数：10、20、30 和 40。接下来，使用 bar 函数将这两个列表传递给它，并使用 title 函数设置图表的标题，使用 xlabel 函数和 ylabel 函数来设置 x 轴和 y 轴的标签。最后，使用 show 函数来显示图表。

```
python matplotlib_example.py

output:
```

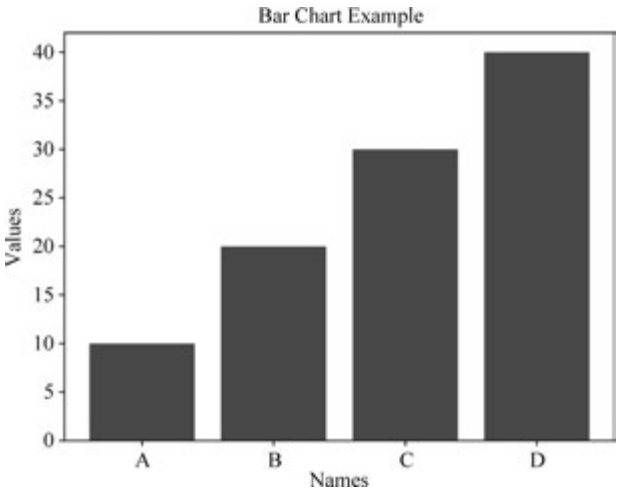

图 3.1　柱状图

Seaborn：Seaborn 是基于 Matplotlib 构建的，提供了一个用户友好的高级接口，用于绘制既美观又富有信息的统计图表。

使用场景：适用于需要快速创建美观统计图表，如绘制成对图、小提琴图、箱形图等的场合。

以下是调用 Seaborn 库进行简单数据处理的代码。

```
  import seaborn as sns
  #加载示例数据集
  data = sns.load_dataset('tips')
  #创建小提琴图
  sns.violinplot(x = 'day', y = 'total_bill', data = data, palette = 'Set3')
  plt.show()
python seaborn_example.py

output:
```

在上面的代码示例中,使用了 Seaborn 库中的 violinplot 函数来创建一个小提琴图,如图 3.2 所示。这个函数需要传递三个参数:x 轴、y 轴和数据集。x 轴是 day,y 轴是 total_bill,数据集是 tips。在这个示例中,使用了 palette='Set3'来设置颜色,最后,使用 show 函数来显示图表。

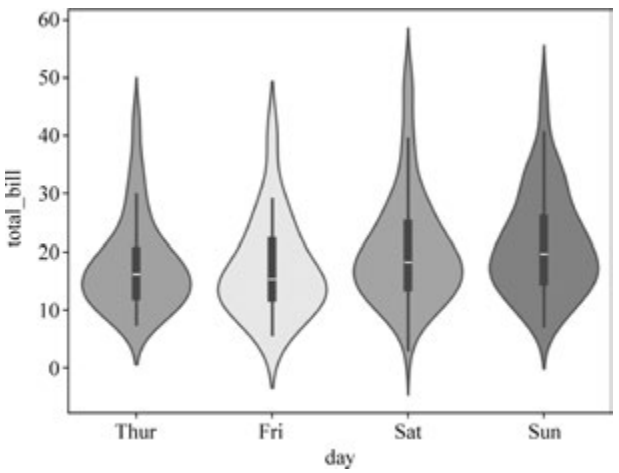

图 3.2 小提琴图

Plotly:Plotly 支持生成交互式图表,非常适合用于 Web 应用和仪表盘的开发。

使用场景:适用于需要在 Web 应用、仪表盘、数据故事和数据科学报告中创建交互式图表的场合。

以下是调用 Plotly 库进行简单数据处理的代码。

```
  import plotly.express as px
  #加载示例数据集
  data = px.data.gapminder().query("year == 2007")
  #创建交互式 bubble 图
  fig = px.scatter(data, x = "gdpPercap", y = "lifeExp", color = "continent",
                   size = "pop", hover_name = "country", size_max = 70)
  fig.show()
'''bash
  python plotly_example.py

  output:
```

代码示例中,使用 Plotly 库中的 scatter 函数来创建一个交互式 bubble 图。这个函数

需要传递 4 个参数：x 轴、y 轴、颜色和大小。x 轴是 gdpPercap，y 轴是 lifeExp，颜色是 continent，大小是 pop。在这个示例中，使用了 hover_name 参数来设置悬停框的名称为 country，使用了 size_max 参数来设置 bubble 的最大大小为 70。最后，使用 show 函数来显示图表，如图 3.3 所示。

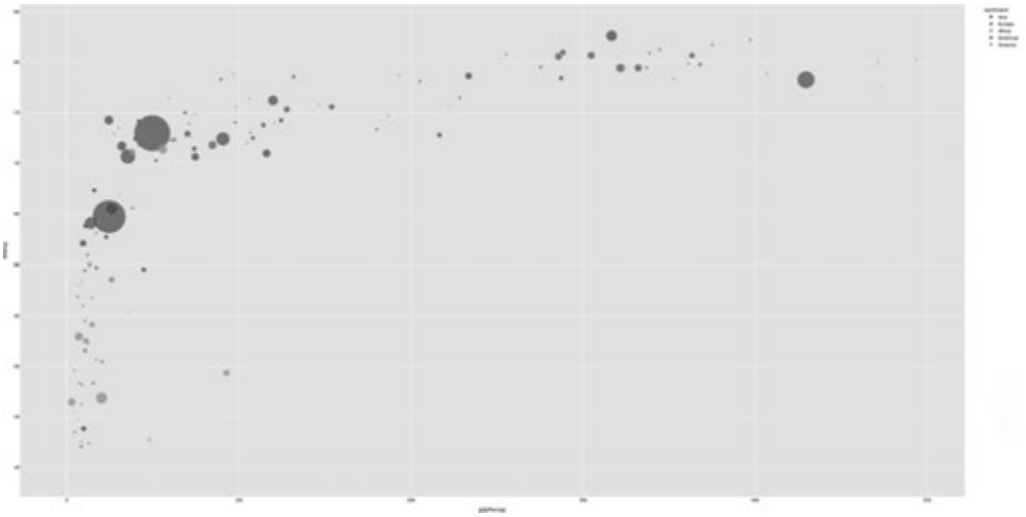

图 3.3　交互式 bubble 图

3.1.4　机器学习和深度学习

Scikit-learn：Scikit-learn 是一个流行的机器学习库，提供了丰富的算法和工具，适用于分类、回归、聚类等任务。

使用场景：适用于需要快速构建和评估机器学习模型的场合。

例如，鸢尾花数据集是一个经典的机器学习数据集，是 Iris 鸢尾花的 4 个特征（花萼长度、花萼宽度、花瓣长度、花瓣宽度）和 3 种类别（山鸢尾、杂色鸢尾、变色鸢尾）的数据集。这个案例解决了鸢尾花的分类问题，即根据鸢尾花的 4 个特征来预测鸢尾花的类别。

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
#加载鸢尾花数据集
iris = load_iris()
X = iris.data
y = iris.target
#划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
#创建高斯朴素贝叶斯模型
model = GaussianNB()
model.fit(X_train, y_train)
#预测
y_pred = model.predict(X_test)
#评估模型
accuracy = model.score(X_test, y_test)
print(f"模型准确率: {accuracy:.2f}")
```

代码示例中，使用 Scikit-learn 库中的 GaussianNB 类来创建一个高斯朴素贝叶斯模型。然后，使用 fit 方法来训练模型，并使用了 predict 方法来进行预测。最后，使用 score 方法来评估模型的准确率。

```
python scikit_learn_example.py

output:

模型准确率: 1.00
```

PyTorch 和 TensorFlow：PyTorch 和 TensorFlow 是两大主流的深度学习框架，支持构建和训练复杂的神经网络模型。

使用场景：适用于需要构建深度神经网络和进行大规模数据训练的场合，例如，图像识别、自然语言处理、语音识别等。

```
''' python
  import torch
  import torch.nn as nn
  #创建一个简单的神经网络模型
  class Net(nn.Module):
      def init(self):
          super(Net, self).init()
          self.fc1 = nn.Linear(1, 10)
          self.fc2 = nn.Linear(10, 1)
     def forward(self, x):
       x = torch.relu(self.fc1(x))
       x = self.fc2(x)
       return x

#创建模型和损失函数
  model = Net()
  criterion = nn.MSELoss()
  optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
#训练模型
  for epoch in range(10):
      optimizer.zero_grad()
      output = model(X_train)
      loss = criterion(output, y_train)
      loss.backward()
      optimizer.step()
      print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
#预测
  y_pred = model(X_test)
  #评估模型
  mse = criterion(y_pred, y_test)
  print(f"均方误差 (MSE): {mse.item():.4f}")
  '''
python pytorch_example.py

output:
```

```
Epoch 1, Loss: 4.8879
Epoch 2, Loss: 4.7918
Epoch 3, Loss: 4.6969
Epoch 4, Loss: 4.6030
Epoch 5, Loss: 4.5103
Epoch 6, Loss: 4.4188
Epoch 7, Loss: 4.3285
Epoch 8, Loss: 4.2393
Epoch 9, Loss: 4.1513
Epoch 10, Loss: 4.0646
均方误差 (MSE): 4.3409
```

3.1.5　Python 在数据分析和可视化中的优势

Python 在大数据分析中的优势是众所周知的。它提供了简洁的语法、丰富的库支持、开源免费的特性等多种优势，使得数据分析变得更加高效和灵活。以下是 Python 在大数据分析中的主要优势，以及如何利用这些优势进行实际工作中的数据分析。

1. 简洁的语法

Python 的语法简洁易读，使得数据分析工作更加高效。Python 代码通常比其他语言更简洁，减少了编写和调试代码的时间，提高了开发效率。

代码案例：使用 Pandas 进行数据清洗。

```
import pandas as pd
#示例数据
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Score': [90, 80, 70, 60, 50]}
df = pd.DataFrame(data)
#数据清洗：过滤出分数大于 80 的数据
filtered_df = df[df['Score'] > 80]
print(filtered_df)
python pandas_filter_example.py
output:
    Name   Score
0  Alice     90
```

2. 丰富的库支持

Python 拥有超过 125 000 个第三方库，特别是数据分析和可视化库，使得数据处理和分析变得更加简单和灵活。以下是一些常用的数据分析库。

代码案例：使用 Matplotlib 和 Seaborn 进行数据可视化，如图 3.4 所示。

```
import matplotlib.pyplot as plt
import seaborn as sns
#示例数据
tips = sns.load_dataset("tips")
#使用 Seaborn 绘制散点图
sns.scatterplot(x = "total_bill", y = "tip", hue = "sex", data = tips)
plt.show()
python pandas_filter_example.py
output:
```

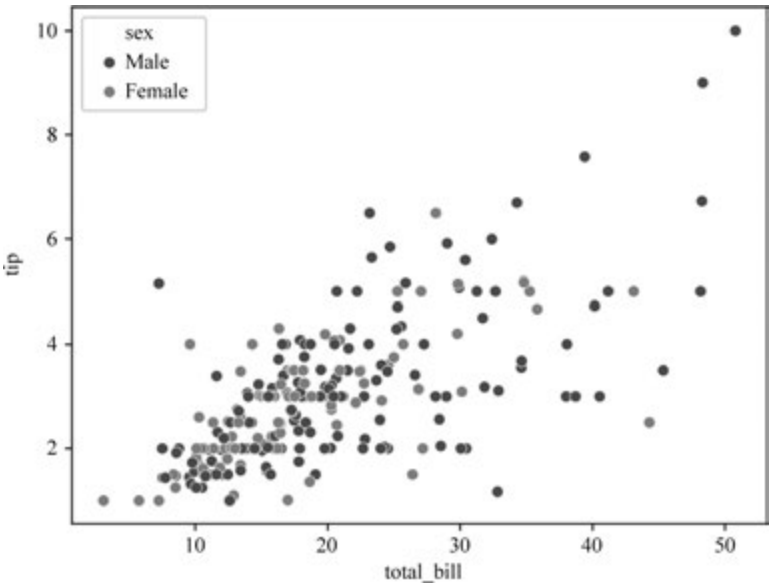

图 3.4 散点图

3.2 桑基图

3.2.1 介绍

在数据分析领域,许多复杂数据结构可以通过“流动关系”这一概念进行简化理解。

桑基图(Sankey Diagram)是一种精妙的可视化工具,由 19 世纪爱尔兰工程师 Matthew Henry Phineas Riall Sankey 首创,最初用于展示蒸汽机的能量流动效率。如今,它已成为数据可视化领域的瑰宝,专门用于展示复杂的流量关系。

桑基图的独特之处在于,帮助分析师立即识别出主要数据流向和关键连接点,从而快速掌握复杂系统中的重要数据传递路径,为决策提供直观支持。桑基图广泛应用于能源分析、物流管理、预算分配、用户流量分析等多个领域。

3.2.2 示例数据

如表 3.1 所示是某些国家体育运动流行度指数。

表 3.1 运动流行度指数

Country	Sport	Popularity
US	Cricket	10
US	Football	40
US	Basketball	50
India	Cricket	85
India	Football	10
India	Basketball	5

续表

Country	Sport	Popularity
England	Cricket	10
England	Football	85
England	Basketball	5
Australia	Cricket	50
Australia	Football	45
Australia	Basketball	5

其中，Country 国家和 Sports 体育运动是实体。它们之间流动的信息就是流行度值。这是一个非常典型的实体之间流动关系的数据分析场景，数据分析师可以使用多种方法，如柱状图来表示这些数据的分配流动关系，如图 3.5 所示。

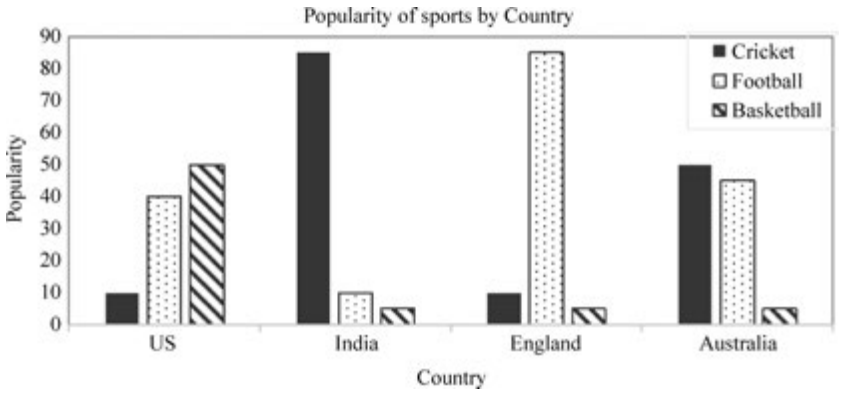

图 3.5　柱状图

然而，桑基图是一种表示流数据集的更酷且优雅的替代方案，如图 3.6 所示。桑基图是一种图形可视化工具，用于表示流动数据的路径和量。它以流线图的形式展示实体之间的流动关系，并使用不同宽度的箭头来表示流量的大小，使得数据分析师可以快速看到实体之间的流动关系，并且可以快速地判断出哪些流动关系是最重要的。

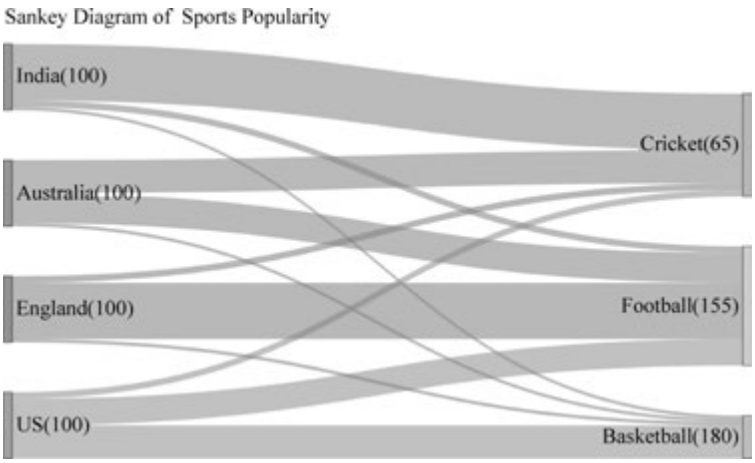

图 3.6　桑基图

如图 3.6 所示，弧线表示两个实体之间的数据流动关系，弧线的宽度与流行度值成正比。这极大地简化了数据分析过程。从上面的桑基图可以很快判断出来：哪些实体之间的流动关系是最重要的(或流量最大的)。

3.2.3 绘制柱状图和桑基图

以下是绘制柱状图和桑基图的完整代码。

1. 绘制柱状图代码

```
import matplotlib.pyplot as plt
import pandas as pd

#文件名: grouped_bar.py
#定义数据字典,包含国家、体育项目和受欢迎程度
#数据表示每个国家在不同体育项目上的受欢迎程度

data = {
    'Country': ['US', 'US', 'US', 'India', 'India', 'India', 'England', 'England', 'England',
'Australia', 'Australia', 'Australia'],
    'Sport': ['Cricket', 'Football', 'Basketball', 'Cricket', 'Football', 'Basketball', 'Cricket',
'Football', 'Basketball', 'Cricket', 'Football', 'Basketball'],
    'Popularity': [10, 40, 50, 85, 10, 5, 10, 85, 5, 50, 45, 5]
}

#创建 DataFrame
#将数据字典转换为 DataFrame 格式,便于处理和分析
df = pd.DataFrame(data)

#绘制分组条形图
fig, ax = plt.subplots(figsize=(10, 6))

#使用 pivot_table 将数据重塑为适合绘制的格式
#通过 pivot_table 方法,将受欢迎程度数据按照国家和体育项目进行整理
pivot_df = df.pivot_table(values='Popularity', index='Country', columns='Sport')

#绘制条形图
#使用 plot 方法绘制分组条形图,展示每个国家在不同体育项目上的受欢迎程度
pivot_df.plot(kind='bar', ax=ax)

#设置图表标题和标签
#设置条形图的标题和轴标签
ax.set_title('Popularity of Sports by Country')
ax.set_ylabel('Popularity')
ax.set_xlabel('Country')

#显示图表
#设置 x 轴标签的旋转角度,并调整布局以适应图表显示
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

2. 绘制桑基图代码

```
import plotly.graph_objects as go
import pandas as pd

#文件名: sankey.py
#定义数据字典,包含国家、体育项目和受欢迎程度

data = {
    'Country': ['US', 'US', 'US', 'India', 'India', 'India', 'England', 'England', 'England',
'Australia', 'Australia', 'Australia'],
    'Sport': ['Cricket', 'Football', 'Basketball', 'Cricket', 'Football', 'Basketball', 'Cricket',
'Football', 'Basketball', 'Cricket', 'Football', 'Basketball'],
    'Popularity': [10, 40, 50, 85, 10, 5, 10, 85, 5, 50, 45, 5]
}

#创建 DataFrame
#将数据字典转换为 DataFrame 格式,便于处理和分析
df = pd.DataFrame(data)

#计算每个国家和体育项目的总体数值
#使用 groupby 方法按国家和体育项目分组,并计算受欢迎程度的总和
country_values = df.groupby('Country')['Popularity'].sum().tolist()
sport_values = df.groupby('Sport')['Popularity'].sum().tolist()

#准备桑基图数据
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=60,                                    #节点之间的填充
        thickness=10,                              #节点的厚度
        line=dict(color="black", width=0.5), #节点边框的颜色和宽度
        #标签包含国家或体育项目名称以及其总体数值
        label=[f"{country} ({value})" for country, value in zip(df['Country'].unique(),
country_values)] +
              [f"{sport} ({value})" for sport, value in zip(df['Sport'].unique(), sport_
values)],
        #节点的颜色
        color=["rgba(102, 194, 165, 0.8)", "rgba(252, 141, 98, 0.8)", "rgba(141, 160, 203,
0.8)", "rgba(231, 138, 195, 0.8)", "rgba(166, 216, 84, 0.8)", "rgba(255, 217, 47, 0.8)",
"rgba(229, 196, 148, 0.8)"],
        hovertemplate="%{label}<extra></extra>" #悬停时显示的模板
    ),
    link=dict(
        #定义链接的来源和目标节点
        source=[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3],
        target=[4, 5, 6, 4, 5, 6, 4, 5, 6, 4, 5, 6],
        value=df["Popularity"],                    #链接的值,即受欢迎程度
        #链接的颜色
        color=["rgba(102, 194, 165, 0.5)", "rgba(102, 194, 165, 0.5)", "rgba(102, 194,
165, 0.5)",
```

```
                "rgba(252, 141, 98, 0.5)", "rgba(252, 141, 98, 0.5)", "rgba(252, 141, 98, 0.5)",
                "rgba(141, 160, 203, 0.5)", "rgba(141, 160, 203, 0.5)", "rgba(141, 160, 203, 0.5)",
                "rgba(231, 138, 195, 0.5)", "rgba(231, 138, 195, 0.5)", "rgba(231, 138, 195, 0.5)"],
        hovertemplate = " %{source.label} → %{target.label}: %{value}<extra></extra>"
    #悬停时显示的模板
        )
    )])

    #更新图表布局
    #设置图表的标题、字体大小、尺寸和边距
    fig.update_layout(
        title_text = "Sankey Diagram of Sports Popularity",
        font_size = 20,
        height = 600,
        width = 1000,
        margin = dict(l = 50, r = 50, t = 50, b = 50)
    )
    #显示图表
    fig.show()
```

3.2.4 Plotly.graph_objects 模块

桑基图的绘制，用到 Plotly 库的 graph_objects 模块。

Plotly.graph_objects 是 Plotly 库的主要模块，提供了一个统一的 API 来创建和自定义图表。

Plotly.graph_objects 将图表分解为多个部分，包括 node、link、marker、axis、layout 等，每个部分都可以单独地进行配置和自定义。

Plotly.graph_objects 提供了一个名为 go 的别名，可以使用 go.Figure 来创建图表，使用 go.Sankey 来创建桑基图等。

Plotly.graph_objects 还提供了一个名为 make_subplots 的函数，可以用来快速创建子图表。

3.3 随机森林算法实现乳腺癌良恶性分类

3.3.1 引言

本节学习如何使用随机森林算法实现一个简单的乳腺癌良恶性分类系统。如图 3.7 所示是低核级乳腺导管原位癌的病理图片。尽管这个系统结构相对简单，但它在乳腺癌诊断数据集上仍能达到超过 95%的准确率，充分展示了机器学习在医疗健康领域的应用潜力。

本案例适合机器学习初学者，下面将详细解释代码的每一部分，帮助读者理解随机森林算法的基本原理和实现方法。图 3.8 展示了乳腺癌诊断数据分析流程。

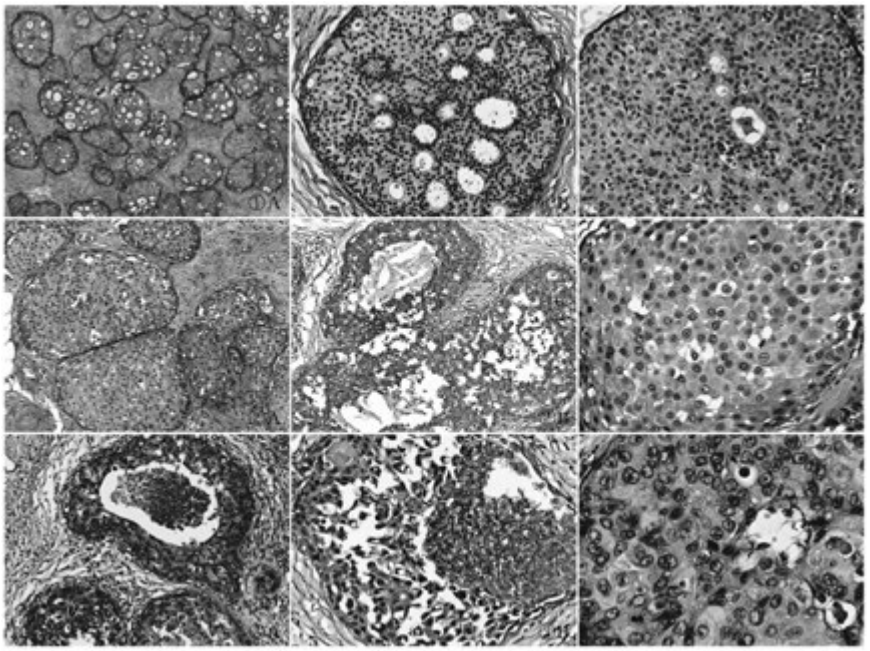

图 3.7　低核级乳腺导管原位癌

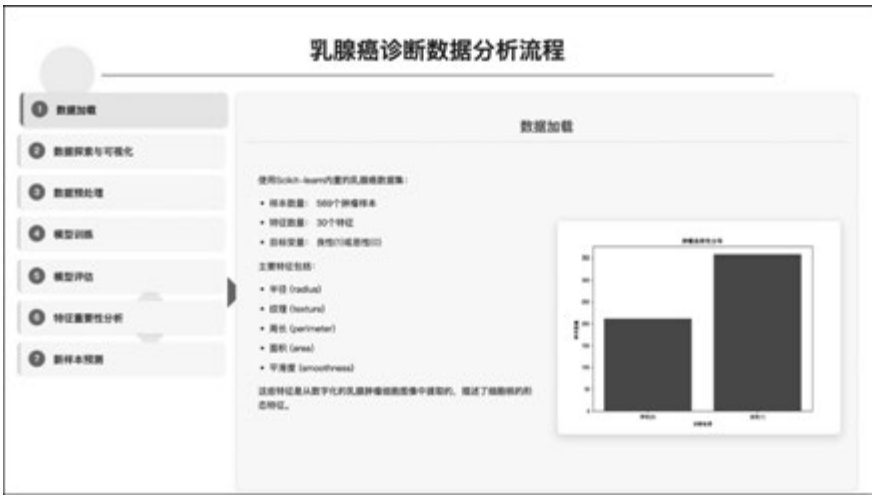

图 3.8　乳腺癌诊断数据分析流程

3.3.2　随机森林算法基础

1. 随机森林的定义

随机森林(Random Forest)是一种集成学习方法,它通过构建多个决策树并将它们的预测结果进行组合来提高整体预测性能。随机森林克服了单个决策树容易过拟合的缺点,具有较好的泛化能力。

2. 随机森林的核心原理

随机森林的工作原理基于两个关键概念：Bootstrap 抽样与决策树的构建。

这种随机性使得森林中的每棵树都有所不同，从而提高了模型的多样性和鲁棒性。

3. 随机森林的优势

通过集成多棵决策树降低方差，抗噪声与过拟合能力强，泛化性能出色。不需要复杂特征工程，可自动计算特征重要性，适配各类特征类型。支持并行训练，计算效率高，内存占用低。利用袋外数据验证模型，无须额外划分数据集，且比神经网络等黑箱模型更具可解释性。适用于分类、回归、特征筛选等多场景，是兼顾精度与易用性的强大集成学习模型。

3.3.3 乳腺癌数据集

1. 数据集来源

这里使用的是 Scikit-learn 内置的乳腺癌诊断数据集（Breast Cancer Wisconsin Diagnostic Dataset）。该数据集最初由威斯康星大学的 William H. Wolberg、W. Nick Street 和 Olvi L. Mangasarian 创建，是乳腺癌研究和机器学习领域的经典数据集。

这个数据集是通过对乳腺肿瘤细针抽吸（FNA，一种穿刺技术）的数字化图像进行特征提取得到的。每个样本代表一个患者的乳腺肿瘤细胞核的测量结果。在这个数据集中，恶性肿瘤（malignant）被标记为 0，良性肿瘤（benign）被标记为 1。

2. 数据集特点

该数据集包含 569 个样本，每个样本有 30 个特征，具体特点如下。

（1）肿块尺寸相关：如半径、周长、面积的均值、标准差、最坏值（最大值）。

（2）纹理特征：灰度值的均匀性、对比度等统计量。

（3）形状特征：边缘不规则性、凹点数量等几何指标。

（4）紧凑性：面积与周长的比例相关特征。

（5）平滑度：半径变化的曲率统计量。

3. 特征说明

数据集中的 30 个特征是从细胞核图像中提取的 10 个基本特征的均值、标准误差和“最差”（最大）值计算得来的。图 3.9 展示了电子显微镜下乳腺癌细胞图片。

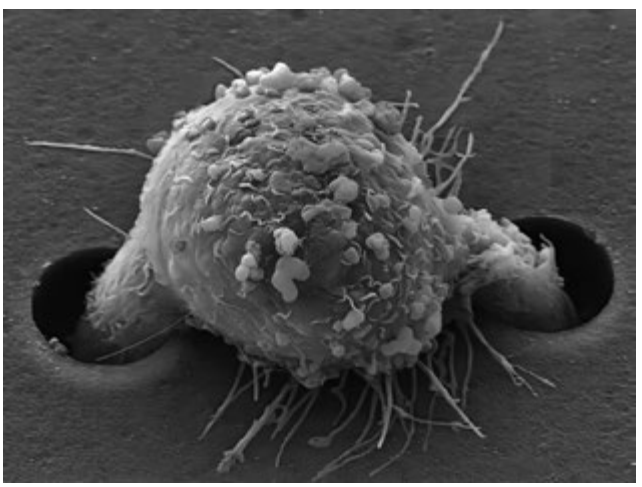

图 3.9 电子显微镜下乳腺癌细胞

4. 数据示例

以下是数据集中前三个样本的特征值示例(为了简洁,只显示部分数值)。

```
样本 1(恶性):
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01 … ]

样本 2(恶性):
[2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02 … ]

样本 3(恶性):
[1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01 … ]
```

这些数值分别对应前面提到的 30 个特征。例如,第一个样本的平均半径为 17.99,平均纹理为 10.38,平均周长为 122.8,平均面积为 1001,等等。

在代码中,这些特征被加载到一个 Pandas DataFrame 中,使其更易于处理和可视化。

5. 特征可视化示例

下面是一些重要特征在良性和恶性样本中的分布对比。图 3.10 展示了重要特征在良恶性肿瘤中的分布对比箱线图。

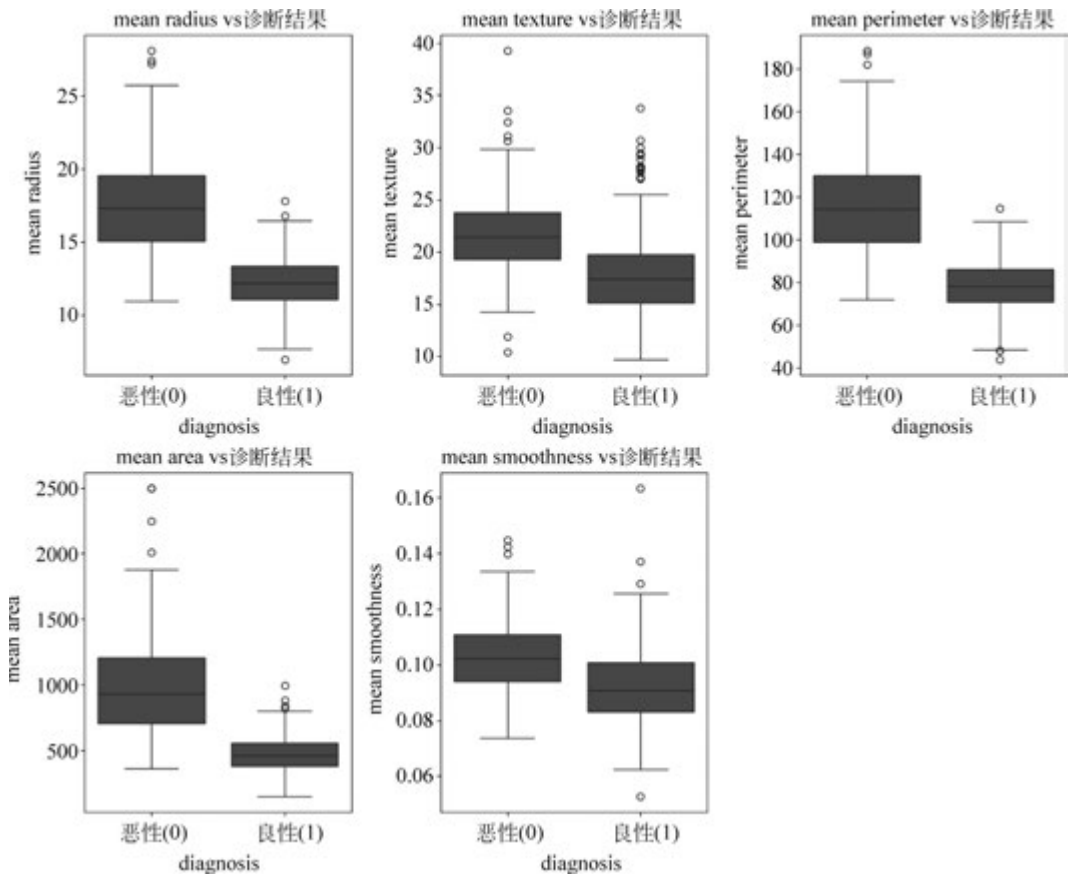

图 3.10　重要特征在良恶性肿瘤中的分布对比箱线图

这些特征之间的差异使得机器学习算法能够有效地区分良性和恶性肿瘤。

3.3.4 代码实现详解

下面逐步解析整个代码实现过程。

1. 导入必要的库

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn. metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report

#设置 matplotlib 支持中文显示
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS', 'SimHei', 'Microsoft YaHei']
#设置中文字体
plt.rcParams['axes.unicode_minus'] = False     #解决保存图像负号'-'显示为方块的问题
mpl.rcParams['font.family'] = ['sans-serif']  #解决负号显示问题
```

2. 加载数据

```
def load_data():
    """加载乳腺癌数据集"""
    print("正在加载乳腺癌数据集...")

    #从 sklearn 内置数据集加载数据
    data = load_breast_cancer()
    X = data.data
    y = data.target                              #0 表示恶性,1 表示良性
    feature_names = data.feature_names
    target_names = data.target_names

    print(f"数据集加载完成!")
    print(f"样本数量: {X.shape[0]}")
    print(f"特征数量: {X.shape[1]}")
    print(f"目标类别: {target_names}")

    #转换为 DataFrame 以便于分析
    df = pd.DataFrame(X, columns=feature_names)
    df['diagnosis'] = y
    df['diagnosis'] = df['diagnosis'].map({0: '恶性(0)', 1: '良性(1)'})

    return df, X, y, feature_names
```

3. 数据探索与可视化

```
def explore_data(df):
#这个函数执行探索性数据分析,帮助我们理解数据的分布和特征
    """探索性数据分析"""
    print("\n 数据探索与可视化...")

    #检查目标变量分布
    diagnosis_counts = df['diagnosis'].value_counts()
    print("\n 目标变量分布:")
    print(diagnosis_counts)

    #可视化目标变量分布
    plt.figure(figsize=(8, 6))
    sns.countplot(x='diagnosis', data=df)
    plt.title('肿瘤良恶性分布')
    plt.xlabel('诊断结果')
    plt.ylabel('样本数量')
    plt.savefig('diagnosis_distribution.png')
    plt.close()

    #选择几个重要特征进行可视化
    plt.figure(figsize=(12, 10))
    features_to_plot = ['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean
smoothness']
    for i, feature in enumerate(features_to_plot):
        plt.subplot(2, 3, i+1)
        sns.boxplot(x='diagnosis', y=feature, data=df)
        plt.title(f'{feature} vs 诊断结果')
    plt.tight_layout()
    plt.savefig('features_boxplot.png')
    plt.close()

    print("数据探索完成,图表已保存。")
```

4. 数据预处理

```
def preprocess_data(df, X, y):
    """数据预处理"""
    print("\n 数据预处理...")

    #注意: 在 load_data 函数中,将原始数值标签映射为文本标签,这里需要将其转回数值便于模
    #型训练
    y_numeric = df['diagnosis'].map({'恶性(0)': 0, '良性(1)': 1})

    #划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    print(f"训练集样本数: {X_train.shape[0]}")
    print(f"测试集样本数: {X_test.shape[0]}")
```

```
    #特征标准化
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print("数据预处理完成!")

    return X_train_scaled, X_test_scaled, y_train, y_test, scaler
```

1）数据集划分

使用 train_test_split 函数将数据集分为训练集(80%)和测试集(20%)。

2）特征标准化

使用 StandardScaler 将特征转换为均值为 0、标准差为 1 的标准正态分布。

5. 模型训练

```
def train_model(X_train, y_train):
#这个函数负责创建和训练随机森林模型
    """训练随机森林模型"""
    print("\n训练随机森林模型...")

    #创建随机森林分类器
    model = RandomForestClassifier(
        n_estimators=100,                    #决策树数量
        max_depth=10,                        #决策树最大深度
        random_state=42                      #随机种子,确保结果可重现
    )

    #训练模型
    model.fit(X_train, y_train)

    print("模型训练完成!")

    return model
```

1）模型创建

使用 RandomForestClassifier 创建随机森林分类器。

2）模型训练

使用 fit 方法在训练数据上训练模型。

6. 模型评估

```
def evaluate_model(model, X_test, y_test, feature_names):
    """评估模型性能"""
    print("\n评估模型性能...")

    #在测试集上进行预测
    y_pred = model.predict(X_test)

    #计算准确率
```

```
    accuracy = accuracy_score(y_test, y_pred)
    print(f"准确率: {accuracy:.4f}")

    #生成分类报告
    print("\n 分类报告:")
    report = classification_report(y_test, y_pred, target_names=['恶性(0)', '良性(1)'])
    print(report)

    #生成混淆矩阵
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['恶性(0)', '良性(1)'],
                yticklabels=['恶性(0)', '良性(1)'])
    plt.title('混淆矩阵', fontsize=15)
    plt.xlabel('预测标签', fontsize=12)
    plt.ylabel('真实标签', fontsize=12)
    plt.savefig('confusion_matrix.png')
    plt.close()

    #计算其他评估指标
    precision = precision_score(y_test, y_pred, pos_label=1)     #良性(1)为正类
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)

    print(f"精确率: {precision:.4f}")
    print(f"召回率: {recall:.4f}")
    print(f"F1 分数: {f1:.4f}")

    #绘制特征重要性图
    plt.figure(figsize=(12, 8))
    feature_importance = model.feature_importances_
    indices = np.argsort(feature_importance)[::-1]

    plt.title('特征重要性排名')
    plt.bar(range(X_test.shape[1]), feature_importance[indices], align='center')
    plt.xticks(range(X_test.shape[1]), [feature_names[i] for i in indices], rotation=90)
    plt.tight_layout()
    plt.savefig('feature_importance.png')
    plt.close()

    print("模型评估完成,结果已保存。")

    return accuracy, precision, recall, f1
```

模型评估是确定模型性能的关键步骤。

1）性能指标计算

准确率(Accuracy)：正确预测的样本比例。

2）可视化评估结果

混淆矩阵：直观展示预测结果与真实标签的对比。

7. 新样本预测

```
def predict_sample(model, scaler, feature_names):
#这个函数展示了如何使用训练好的模型对新样本进行预测
    """使用模型预测新样本"""
    print("\n使用模型预测新样本...")

    #创建一个模拟的新样本(这里使用数据集中的一个恶性样本特征值)
    #这些特征包括模型识别为最重要的特征,如 worst area、worst radius 等
    #在实际应用中,这些值应该来自用户输入或医学检测
    sample = np.array([
        17.99, 10.38, 122.8, 1001.0, 0.1184,          #这些值模拟恶性肿瘤的特征
        0.2776, 0.3001, 0.1471, 0.2419, 0.07871,
        1.095, 0.9053, 8.589, 153.4, 0.006399,
        0.04904, 0.05373, 0.01587, 0.03003, 0.006193,
        25.38, 17.33, 184.6, 2019.0, 0.1622, #这里的 worst area 和 worst radius 等是重要
                                             #特征
        0.6656, 0.7119, 0.2654, 0.4601, 0.1189
    ]).reshape(1, -1)

    #标准化新样本
    sample_scaled = scaler.transform(sample)

    #预测
    prediction = model.predict(sample_scaled)
    probability = model.predict_proba(sample_scaled)

    #输出预测结果
    result = {
        'prediction': '良性(1)' if prediction[0] == 1 else '恶性(0)',
        'probability': probability[0][prediction[0]]
    }

    print(f"预测结果: {result['prediction']}")
    print(f"预测概率: {result['probability']:.4f}")

    #输出特征重要性
    feature_importance = model.feature_importances_
    top_features = sorted(zip(feature_names, feature_importance), key = lambda x: x[1],
reverse = True)[:5]

    print("\n对预测影响最大的 5 个特征:")
    for feature, importance in top_features:
        print(f"{feature}: {importance:.4f}")

    return result
```

1) 准备新样本

在实际应用中,这些值应来自医学检测或用户输入。

2) 数据预处理

使用之前训练的标准化器对新样本进行标准化。

3）预测

model.predict：获取类别预测（良性或恶性）。

4）结果分析

输出预测结果和概率。

8. 主函数

```
def main():
    """主函数"""
    try:
        #加载数据
        df, X, y, feature_names = load_data()

        #探索数据
        explore_data(df)

        #预处理数据
        X_train, X_test, y_train, y_test, scaler = preprocess_data(df, X, y)

        #训练模型
        model = train_model(X_train, y_train)

        #评估模型
        accuracy, precision, recall, f1 = evaluate_model(model, X_test, y_test, feature_names)

        #预测新样本
        prediction = predict_sample(model, scaler, feature_names)

        print("\n乳腺癌诊断系统(简化版)运行完成!")

    except Exception as e:
        print(f"程序运行出错: {str(e)}")

if __name__ == "__main__":
    main()
```

主函数是整个程序的入口，它协调了整个流程。

1）流程协调

加载数据。

2）程序入口点

if __name__ == "__main__"：确保代码只在直接运行该文件时执行，而不是在被导入为模块时执行。

3.3.5　如何执行代码

执行这个代码的命令行语句非常简单。假设已经在项目目录下（包含 breast_cancer_diagnosis_simple.py 文件的目录），只需要在终端中输入：

```
python breast_cancer_diagnosis_simple.py
```

这个命令会执行 breast_cancer_diagnosis_simple.py 文件，程序会自动调用 main()函数，从而执行整个流程：加载数据、探索数据、预处理数据、训练模型、评估模型并预测新样本。

3.3.6 运行结果

1. 数据集信息

该数据集为乳腺癌诊断相关数据集，包含用于判断乳腺肿瘤为良性或恶性的相关特征数据，后续将用于加载、探索、预处理、训练模型、评估模型及预测新样本等流程，其中目标变量为 diagnosis，用于标识肿瘤是良性还是恶性。

2. 目标变量分布

```
diagnosis
良性(1)    357
恶性(0)    212
```

这表明数据集中良性样本(357 个)多于恶性样本(212 个)，存在一定的类别不平衡，如图 3.11 所示。

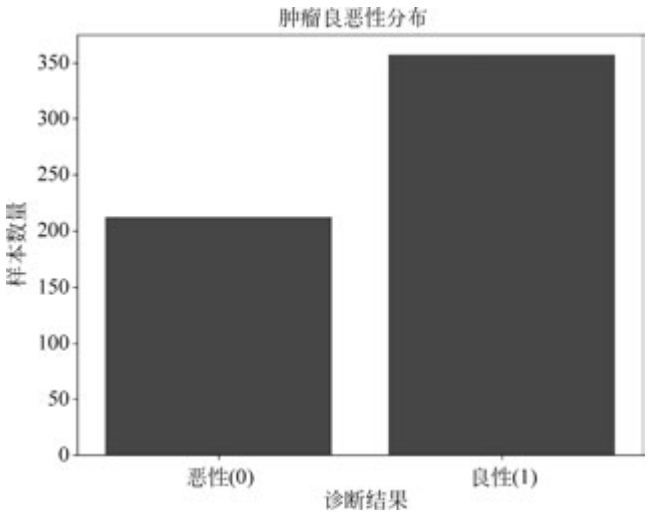

图 3.11 肿瘤良恶性分布

3. 模型性能

在测试集上，随机森林模型通常能达到以下性能。较高的准确率、精确率、召回率和 F1 分数，对良性和恶性肿瘤的区分能力较强，能够有效减少误诊和漏诊情况。

这表明模型在乳腺癌诊断任务上表现出色，能够准确区分良性和恶性肿瘤。如图 3.12 所示为模型预测结果混淆矩阵，展示了正确和错误预测的数量。

4. 特征重要性

随机森林模型的一个优势是可以评估特征重要性。通常，以下特征对预测结果影响较大。

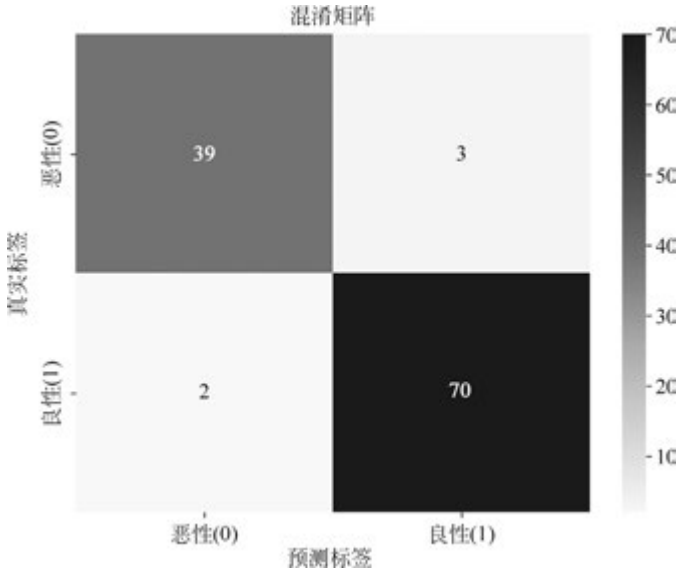

图 3.12　模型预测结果混淆矩阵

细胞核的平均半径、平均纹理、平均周长、平均面积、平均光滑度、平均紧凑度、平均凹度、平均凹点数量、平均对称性、平均分形维数，以及这些特征的标准差和最大值等。

如图 3.13 所示，特征重要性排名显示了各个特征对模型预测的影响程度。

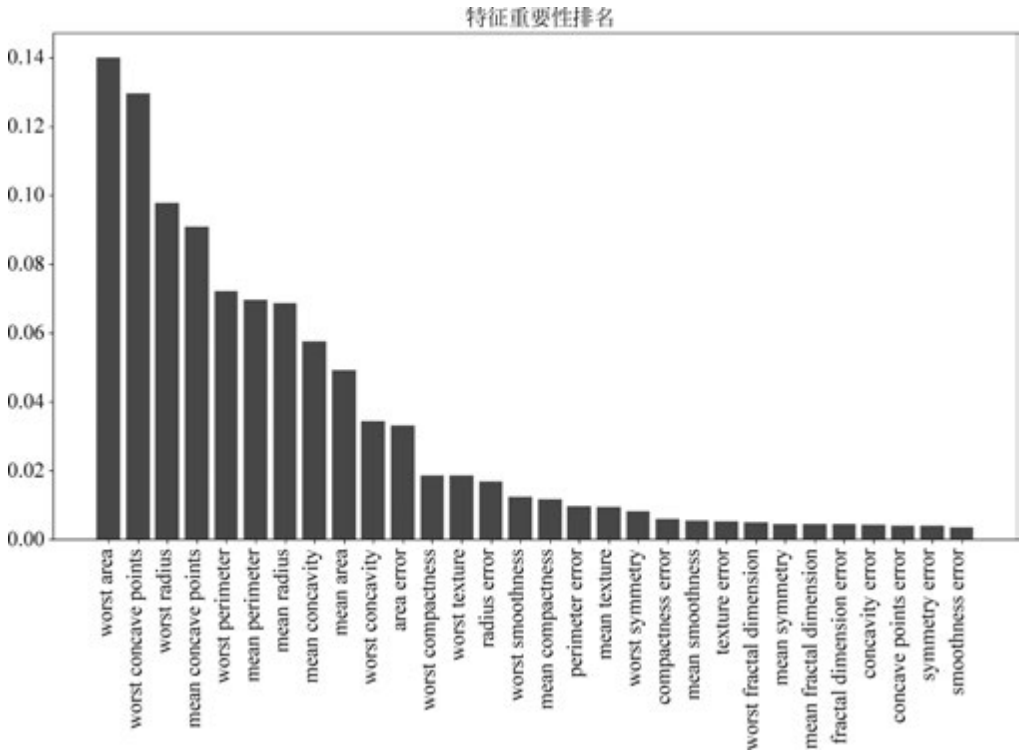

图 3.13　特征重要性排名

这些特征主要与细胞核的大小、形状和纹理有关，这与医学研究中发现的恶性肿瘤特征相符。

3.3.7 总结

在本案例中，学习了如何使用随机森林算法实现一个简单的乳腺癌良恶性分类系统。尽管结构简单，但这个系统在乳腺癌诊断数据集上表现出色，展示了机器学习在医疗健康领域的应用潜力。

通过这个项目，读者可掌握机器学习项目的完整流程：数据加载、探索性分析、预处理、模型训练、评估和预测。这些技能可以应用于各种机器学习任务，特别是医疗诊断相关的分类问题。

随机森林作为一种强大的集成学习方法，在许多分类任务中表现出色。它的高准确率、抗过拟合能力和特征重要性评估功能使其成为医疗诊断等领域的理想选择。