第5章

监督学习


观看视频

本章将介绍监督学习的基础知识，包括其定义、流程及关键因素。这是构建机器学习模型的基石，其核心在于理解数据特征和目标之间的关系。通过详细阐述输入（特征）和输出（标签）的概念，本章将深入讲解如何构建和训练模型以及进行预测。此外，本章还提供了监督学习在分类、回归等多个领域应用的实例，助力读者掌握监督学习方法。

本章的学习目标：

- 掌握监督学习的概念、流程，以及评估指标；
- 掌握线性回归、逻辑回归、决策树、支持向量机、K 近邻算法。

5.1 监督学习简介

5.1.1 监督学习的概念

监督学习是机器学习的一种主要方法，其基本思想是利用带有标签的训练数据来训练模型，使其能够预测未标记数据的标签。在监督学习中，训练数据通常由一组输入和对应的输出标签组成，每个样本都包括输入、输出和模型。

（1）输入：输入通常被称为特征（feature），它是一个包含一组数值或特征向量的向量，用于描述样本的属性和特征。

（2）输出：输出通常被称为标签（label），它是一个数值或分类结果，用于表示样本的类别或属性。

（3）模型：在监督学习中，这些输入和输出之间的映射关系被称为模型。通过在训练数据上训练模型，可以使模型能够对未标记数据进行预测。

监督学习的应用非常广泛，举例如下。

（1）分类：监督学习可以用于对数据进行分类和识别，例如图像分类、人脸识别、语音识别、文本分类、医学诊断等。

（2）预测：监督学习可以用于预测未来的趋势和结果，例如股票价格预测、天气预报、交通流量预测等。

（3）推荐系统：监督学习可以用于构建推荐系统，例如电影推荐、购物推荐等。

（4）自然语言处理：监督学习可以用于自然语言处理任务，例如情感分析、文本生成、机器翻译等。

总的来说，监督学习可以帮助人们从数据中提取有用的信息和知识，为决策和问题解决提供支持和指导。

5.1.2 监督学习的流程

监督学习是一种机器学习方法，通过从带有标签的训练数据中学习模式和关系，以进行预测和分类。以下从实践的角度，介绍 Scikit-learn 演示监督学习的基本流程。

1. 模块导入

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

2. 数据准备

收集和整理用于训练和测试的数据集。数据集包括输入特征和对应的标签(目标变量)。

```
#准备特征和标签
X = ...
y = ...
#划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3. 数据预处理

对输入特征进行选择和预处理，包括特征提取、特征变换、特征缩放等。这可以帮助提高模型的性能和泛化能力。

```
#进行特征缩放
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

4. 模型选择

选择适合问题的机器学习模型，如决策树、支持向量机、神经网络等，并将训练数据输入模型进行训练。

```
model = LinearRegression()
```

5. 模型训练

训练过程是通过调整模型参数来最小化损失函数(或最大化模型的似然函数)的过程。使用 Scikit-learn 一般会自动调整模型参数。

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

6. 模型评估

使用测试数据集评估训练好的模型的性能。常用的评估指标包括准确率、精确率、召回率、F1 分数等。通过评估结果可以了解模型的预测能力和泛化能力。

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

7. 参数调优

根据评估结果，对模型进行参数调优或改进，以进一步提高模型性能，可以通过交叉验证、

网格搜索等技术来进行。

得到模型以后便可应用了，使用训练好的模型对新的未标记数据进行预测和分类。模型通过输入特征进行推断，并生成相应的预测结果。预测结果可以用于实际应用中的决策制定、问题解决等。监督学习是一个迭代的过程，持续监控模型的性能和表现，并根据需要对模型进行更新和改进，这可能涉及重新收集和整理数据、重新训练模型、调整参数等。

以上是监督学习的一般流程，具体的实施步骤和技术选择会根据具体问题和数据的特点而有所不同。

5.1.3 监督学习的分类

监督学习可以用于多种任务，监督学习可以分为两大类：回归(Regression)和分类(Classification)。这两种类型的任务涉及了对不同类型的目标变量进行预测。

1. 回归

任务描述：在回归问题中，目标变量是连续的实数值。任务的目标是通过学习输入变量与目标变量之间的关系，建立一个预测模型，使得对于新的输入，可以预测其对应的实数值。

如线性回归在生活中的一个例子是用于设计弹簧称重计。弹簧称重计是一种常见的称重装置，通过测量弹簧的形变来确定物体的重量(见图 5.1)，因此，可以使用线性回归来建立弹簧形变 x 与物体重量 m 之间的关系。

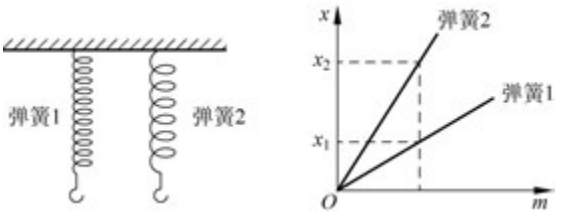

图 5.1 弹簧弹性系数测量

假设已经进行了实验，收集了不同重量下弹簧的形变数据。现在想要使用线性回归来建立一个模型，通过测量弹簧的形变就能估计物体的重量。

以下是一个简单的 Python 代码示例，使用 Scikit-learn 进行线性回归的实现。

【例 5.1】 线性回归(见图 5.2)。

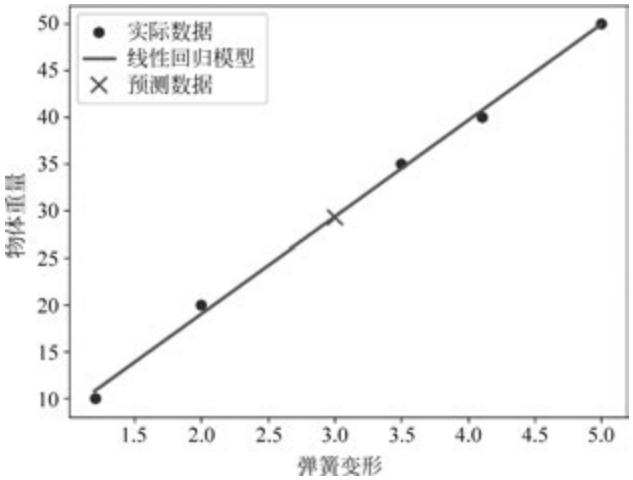

图 5.2 线性回归示例的可视化结果

设计弹簧秤(也称为弹簧称重计)的原理是基于弹簧的弹性特性。当一个物体放在弹簧上时,弹簧会发生形变,其形变的程度与物体的重量成正比。线性回归模型可以建立弹簧形变与物体重量之间的简单、直观的关系,使得弹簧秤在测量物体重量时能更可靠。

```
#引入必要的库
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

#设置字体为 SimHei(黑体)
plt.rcParams['font.sans-serif'] = ['SimHei']

#假设有弹簧形变和相应的物体重量数据
spring_deformation = np.array([1.2, 2.0, 3.5, 4.1, 5.0])
object_weight = np.array([10, 20, 35, 40, 50])

#将数据整理成二维数组的形式
X = spring_deformation.reshape(-1, 1)

#创建线性回归模型
model = LinearRegression()

#拟合模型
model.fit(X,object_weight)

#获取弹性系数和截距
slope = model.coef_[0]
intercept = model.intercept_

#打印弹性系数和截距
print(f"弹性系数(斜率):{slope}")
print(f"截距:{intercept}")

#预测新的弹簧形变对应的物体重量
new_deformation = np.array([3.0]).reshape(-1, 1)
predicted_weight = model.predict(new_deformation)

#打印预测结果
print(f"当弹簧变形为{new_deformation[0][0]}时,预测物体重量为{predicted_weight[0]}")

#可视化结果
plt.scatter(spring_deformation,object_weight, color = 'blue', label = '实际数据')
plt.plot(spring_deformation, model.predict(X), color = 'red', linewidth = 2, label = '线性回归模
型')
plt.scatter(new_deformation, predicted_weight, color = 'green', marker = 'x', s = 100, label = '预测
数据')
plt.xlabel('弹簧形变')
plt.ylabel('物体重量')
plt.legend()
plt.show( )
```

运行结果:

```
弹性系数(斜率):10.311324697033017
截距:-1.5837860426243324
当弹簧形变为 3.0 时,预测物体重量为 29.35018804847472
```

2. 分类

任务描述：在分类问题中，目标变量是离散的类别标签。任务的目标是通过学习输入变量与目标类别之间的关系建立一个分类模型，使得对于新的输入，可以将其分配到预定义的类别中。

【例 5.2】 本例提供一个逻辑回归二分类问题，预测学生是否通过了一门考试。在这个场景中，使用逻辑回归来预测学生是否能通过了考试，基于他们的学习时间，来预测学生是否能通过考试(实际问题比本例复杂得多，这里假设这种方式是合理的)。已知数据如表 5.1 所示。

表 5.1 数据表

学习时间/小时	通过考试(1)/未通过考试(0)
2	0
3	0
4	0
5	1
6	1
7	1

以下是一个使用 Scikit-learn 演示学生是否能通过考试的逻辑回归二分类问题的简单代码，运行结果如图 5.3 所示。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

#学习时间(小时)
study_hours = np.array([2, 3, 4, 5, 6, 7])

#是否能通过考试(1 表示通过，0 表示未通过)
pass_exam = np.array([0, 0, 0, 1, 1, 1])

#将数据整理成二维数组的形式
X = study_hours.reshape(-1, 1)

#创建逻辑回归模型
model = LogisticRegression()

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

#拟合模型
model.fit(X_train, y_train)

#预测测试集
y_pred = model.predict(X_test)

#计算准确度和混淆矩阵
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

#打印结果
print(f"准确度:{accuracy}")
print(f"混淆矩阵:\n{conf_matrix}")

#可视化决策边界
```

```
plt.scatter(X, pass_exam, color = 'blue', marker = 'o', label = '实际数据')
plt.plot(X, model.predict_proba(X)[:, 1], color = 'red', linewidth = 2, label = '逻辑回归模型')
plt.xlabel('学习时间/小时')
plt.ylabel('是否通过考试')
plt.legend()
plt.show()
```

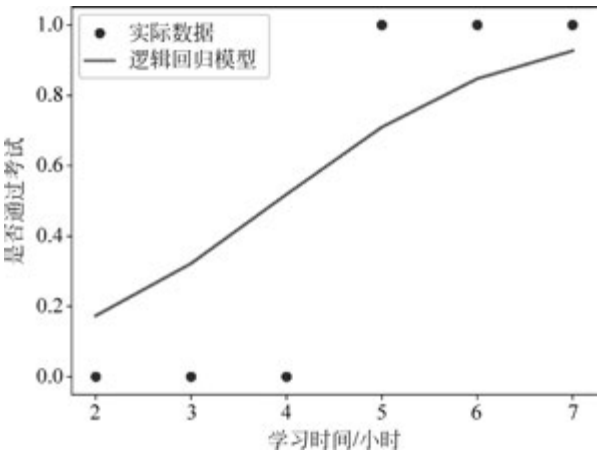

图 5.3　逻辑回归解决分类问题

本例包括以下几个步骤。

(1) 建立模型：使用逻辑回归模型，将学习时间作为特征，考试是否通过作为目标变量。

(2) 模型训练：通过训练模型，找到学习时间和通过考试的关系，使得模型能够根据学习时间来预测学生是否能通过考试。

(3) 预测：当有一个新学生学习了一定时间，可以使用训练好的模型来预测该学生是否能通过考试。

解释：模型的输出是一个介于 0 和 1 之间的概率值，可以解释为学生通过考试的概率。例如，如果模型输出是 0.8，表示学生有 80%的概率通过考试。

线性回归与逻辑回归这两种类型的任务都是监督学习的一部分，因为模型的训练需要使用带有标签的数据集，其中包含了输入变量和对应的目标变量。在训练期间，模型学习输入和输出之间的关系，以便在未见过的数据上进行准确的预测。回归和分类问题在许多领域都有广泛的应用，涵盖了从经济学到医学等多种不同的应用场景。

5.1.4　回归模型评估

1. 回归模型的常用算法

监督学习回归和分类都有多种常用的算法，以下是一些常用的回归算法。

(1) 线性回归(Linear Regression)：通过拟合一个线性模型来预测目标变量。

(2) 岭回归(Ridge Regression)和 Lasso 回归(Lasso Regression)：是线性回归的正则化版本，用于处理多重共线性或进行特征选择。

(3) 决策树回归(Decision Tree Regression)：使用树结构来建模，根据输入特征逐步做出预测。

(4) 随机森林回归(Random Forest Regression)：是决策树的集成算法，通过组合多个树来提高预测性能。

(5) 梯度提升回归(Gradient Boosting Regression):通过迭代训练弱模型并将它们组合来提升性能。

2. 回归模型评估

在回归问题中,用于评估模型性能的一些常见指标包括均方误差(Mean Squared Error,MSE)、平均绝对误差(Mean Absolute Error,MAE)、R^2(R-squared),以及一些其他的指标。通常 y_{true} 是实际值,y_{pred} 是预测值。以下是回归模型评估的一些指标及其计算公式。

1) 均方误差

(1) 意义:衡量模型预测值与实际值之间的平方差的平均值,均方误差越小表示模型对数据的拟合越好。

(2) 计算公式:

$$\text{MSE}=\frac{1}{n}\sum(y_{\text{true}}-y_{\text{pred}})^2 \tag{5-1}$$

其中,n 是样本数,y_{true} 是实际值,y_{pred} 是预测值。

(3) Scikit-learn 代码计算均方误差的函数:mean_squared_error(y_true, y_pred)。

2) 平均绝对误差

(1) 意义:衡量模型预测值与实际值之间的绝对差的平均值,平均绝对误差越小表示模型对数据的拟合越好。

(2) 计算公式:

$$\text{MAE}=\frac{1}{n}\sum|y_{\text{true}}-y_{\text{pred}}| \tag{5-2}$$

其中,n 是样本数,y_{true} 是实际值,y_{pred} 是预测值。

(3) Scikit-learn 代码计算平均绝对误差的函数:mean_absolute_error(y_true, y_pred)。

3) R^2

(1) 意义:衡量模型对目标变量方差的解释程度。R^2 越接近 1,表示模型对变量的解释越好;R^2 越接近 0,表示模型的解释能力较弱。

(2) 计算公式:

$$R^2=1-\frac{\sum_{i=1}^{n}(y_{\text{true}_i}-y_{\text{pred}_i})^2}{\sum_{i=1}^{n}(y_{\text{true}_i}-\bar{y})^2} \tag{5-3}$$

其中,y_{true} 是实际值,y_{pred} 是预测值。$\bar{y}$ 是观测值的平均值。

(3) Scikit-learn 代码计算 R^2 的函数:r2_score(y_true, y_pred)。

4) 解释方差得分

(1) 意义:衡量模型对目标变量方差的解释比例。解释方差得分越接近 1,表示模型对方差的解释越好。

(2) 计算公式:

$$\text{Explained Variance}=1-\frac{\text{var}(y_{\text{true}}-y_{\text{pred}})}{\text{var}(y_{\text{true}})} \tag{5-4}$$

其中,y_{true} 是实际值,y_{pred} 是预测值。

(3) Scikit-learn 代码计算解释方差得分的函数:explained_variance_score(y_true, y_pred)。

以下是一个简单的代码示例，演示如何使用这些指标。

【例 5.3】 回归模型量化评估(见图 5.4)。

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_
variance_score
from sklearn.linear_model import LinearRegression

#生成一些示例数据
np.random.seed(42)
X_regression = np.random.rand(100, 1)
y_true_regression = 2 * X_regression.squeeze() + 1 + 0.1 * np.random.randn(100)
y_pred_regression = 2 * X_regression.squeeze() + 1 + 0.1 * np.random.randn(100)

#训练线性回归模型
model = LinearRegression()
model.fit(X_regression, y_true_regression)
y_pred_regression_model = model.predict(X_regression)

#计算回归模型评估指标
mse = mean_squared_error(y_true_regression, y_pred_regression_model)
mae = mean_absolute_error(y_true_regression, y_pred_regression_model)
r2 = r2_score(y_true_regression, y_pred_regression_model)
explained_variance = explained_variance_score(y_true_regression, y_pred_regression_model)

#打印指标值
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)
print("Explained Variance:", explained_variance)

#绘制散点图
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.scatter(X_regression, y_true_regression, label='Actual', alpha=0.8)
plt.scatter(X_regression, y_pred_regression_model, label='Predicted', alpha=0.8)
plt.title('Scatter Plot of Actual vs Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()

#绘制残差图
residuals = y_true_regression - y_pred_regression_model
plt.subplot(1, 2, 2)
sns.residplot(X_regression.squeeze(), residuals, lowess=True)
plt.title('Residual Plot')
plt.xlabel('X')
plt.ylabel('Residuals')
plt.tight_layout()
plt.show()
```

运行结果：

```
Mean Squared Error (MSE): 0.008065845639670534
Mean Absolute Error (MAE): 0.07010426719637759
R-squared (R2): 0.9764567983510799
Explained Variance: 0.9764567983510799
```

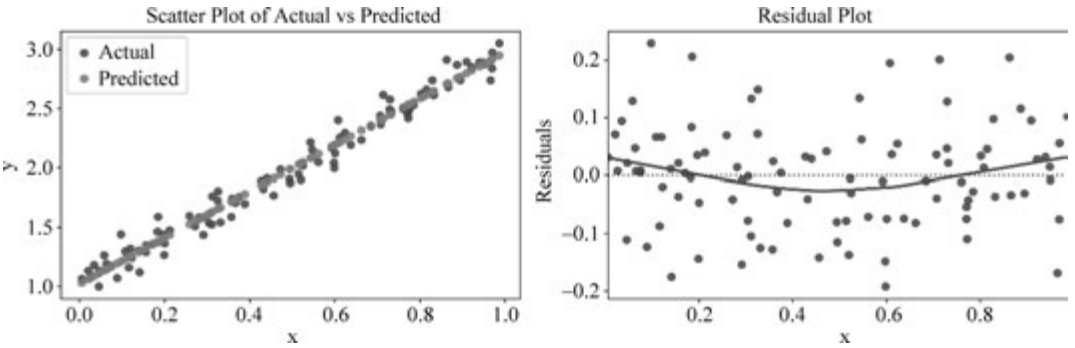

图 5.4 回归模型量化评估图示

这些指标提供了对回归模型性能的不同视角，我们可以根据具体的任务需求选择适当的指标进行评估。

5.1.5 分类模型评估

1. 分类模型算法

常用的分类算法如下。

(1) 逻辑回归(Logistic Regression)：用于二分类问题，基于概率模型进行分类。

(2) 支持向量机(Support Vector Machine，SVM)：通过在特征空间中找到一个超平面来实现分类。

(3) 决策树分类(Decision Tree Classification)：使用树结构进行分类，通过一系列条件判断来决定样本属于哪个类别。

(4) 随机森林分类(Random Forest Classification)：是决策树的集成算法，通过多棵树的投票来进行分类。

(5) K 近邻算法(K-Nearest Neighbor，KNN)：基于离一个样本点最近的 K 个邻居来进行分类。

(6) 朴素贝叶斯分类(Naive Bayes Classification)：基于贝叶斯定理，假设特征之间是相互独立的。

(7) 梯度提升分类(Gradient Boosting Classification)：类似于梯度提升回归，通过迭代训练弱分类器并组合它们来提高性能。

(8) 神经网络(Neural Network)：深度学习中的一种算法，适用于处理大规模和复杂的数据。

以上列举的算法只是分类算法的一部分，每种算法都有其适用的场景和优缺点。在选择算法时，需要考虑数据的性质、问题的复杂度以及对解释性和性能的需求。

2. 分类模型评估

Scikit-learn 的 metrics 模块提供了一些常用的分类性能指标。

1) 混淆矩阵

混淆矩阵(Confusion Matrix)是一个用于衡量分类模型性能的表格(见表 5.2)，它显示了模型在测试集上的预测结果与真实标签的对应关系。其中行表示实际类别，列表示预测类别。对于二元分类问题，混淆矩阵的 4 个主要元素如下。

(1) True Positive(TP)：模型正确地预测为正类别的数量。

(2) False Negative(FN)：模型将实际为正类别错误地预测为负类别的数量。

(3) False Positive(FP)：模型将实际为负类别错误地预测为正类别的数量。

(4) True Negative(TN)：模型正确地预测为负类别的数量。

表 5.2 混淆矩阵

	预测为正类	预测为负类
实际为正类	True Positive(TP)	False Negative(FN)
实际为负类	False Positive(FP)	True Negative(TN)

这些元素有助于计算许多性能指标，如准确率(Accuracy)、精确率(Precision)、召回率(Recall)等，从而更全面地评估分类模型的性能。

Scikit-learn 代码计算混淆矩阵的函数：confusion_matrix(y_true,y_pred)。

2) 准确率

准确率(Accuracy)是指模型正确预测的样本数量占总样本数量的比例。计算公式：

$$\text{Accuracy}=\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{5-5}$$

Scikit-learn 代码计算准确率的函数：accuracy_score(y_true, y_pred)。

3) 精确率

精确率(Precision)是指模型预测为正类的样本中，实际为正类的比例。计算公式：

$$\text{Precision}=\frac{\text{TP}}{\text{TP}+\text{FP}} \tag{5-6}$$

Scikit-learn 代码计算精确率的函数：precision_score(y_true, y_pred)。

4) 召回率

召回率(Recall)是指实际为正类的样本中，模型成功预测为正类的比例。

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}} \tag{5-7}$$

Scikit-learn 代码计算召回率的函数：recall_score(y_true, y_pred)。

5) F1 分数

F1 分数(F1 Score)是精确率和召回率的调和平均。计算公式：

$$\text{F1}=\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} \tag{5-8}$$

Scikit-learn 代码计算 F1 分数的函数：f1_score(y_true, y_pred)。

【例 5.4】 监督学习的分类模型量化评估。

```
from sklearn import metrics
#定义真实标签和预测标签
y_true = [0, 1, 1, 0, 1, 0]
y_pred = [0, 1, 0, 0, 1, 1]

rules = {
    '混淆矩阵':metrics.confusion_matrix(y_true, y_pred),
    '准确率':metrics.accuracy_score(y_true, y_pred),
    '召回率':metrics.recall_score(y_true, y_pred),
    '精确率':metrics.precision_score(y_true, y_pred),
    'F1 分数':metrics.f1_score(y_true, y_pred),
    'AUC':metrics.roc_auc_score(y_true, y_pred),
    'R^2':metrics.r2_score(y_true, y_pred)
}
print(rules))
```

运行结果：

```
{'AUC': 0.6666666666666667,
'F1 分数': 0.6666666666666666,
'R^2': -0.33333333333333326,
'准确率': 0.6666666666666666,
'召回率': 0.6666666666666666,
'混淆矩阵': array([[2, 1],
       [1, 2]], dtype=int64),
'精确率': 0.6666666666666666}
```

6) AUC-ROC

AUC(Area Under Curve)是一种用于评估二元分类模型性能的指标，通常用于衡量模型在不同阈值下的分类能力。ROC(Receiver Operating Characteristic)曲线是一个以假正例率(False Positive Rate，FPR)为横轴，真正例率(True Positive Rate，TPR，又称为召回率)为纵轴的曲线。

(1) FPR 的计算公式：

$$\mathrm{FPR}=\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} \tag{5-9}$$

其中，FP 是假正例的数量，TN 是真负例的数量。FPR 是 ROC 曲线的横轴，表示在实际为负例的样本中，被错误地预测为正例的比例。

(2) TPR 的计算公式：

$$\mathrm{FPR}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{5-10}$$

其中，TP 是真正例的数量，FN 是假负例的数量。TPR 是 ROC 曲线的纵轴，表示在实际为正例的样本中，被正确地预测为正例的比例，也叫召回率。

Scikit-learn 代码计算 ROC 的函数：roc_curve(y_true, y_scores)。该函数返回 ROC 曲线的 FPR、TPR 以及阈值。这些值可以用于可视化 ROC 曲线，评估模型在不同阈值下的性能。

【例 5.5】 下面是一个 ROC 曲线可视化的例子，包括数值和可视化，其中 thresholds 是阈值数组，表示计算 FPR 和 TPR 时使用的不同阈值。运行结果如图 5.5 所示。

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import numpy as np
#生成一些示例数据
np.random.seed(42)
y_true = np.random.randint(2, size=100)
y_scores = np.random.rand(100)
#计算 ROC 曲线
fpr, tpr, thresholds = roc_curve(y_true, y_scores)

#打印数值
print("FPR:", fpr)
print("TPR:", tpr)
print("Thresholds:", thresholds)

#可视化 ROC 曲线
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color = 'darkorange', lw = 2, label = 'ROC curve')
plt.plot([0, 1], [0, 1], color = 'navy', lw = 2, linestyle = '--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc = 'lower right')
plt.show()
```

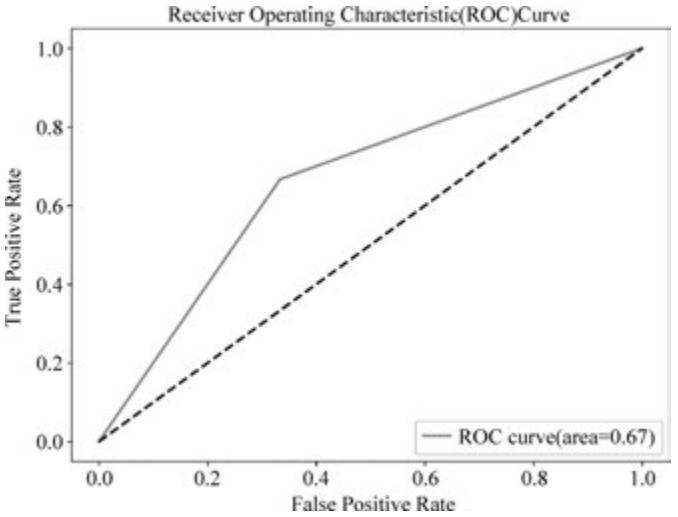

图 5.5 ROC 曲线

在本例的代码中，fpr、tpr 和 thresholds 包含了计算得到的数值。ROC 曲线通过 plt.plot()函数进行绘制，其中横轴代表 fpr，纵轴代表 tpr。

7）多分类问题的指标

对于多分类问题，通常会考虑使用其他评估指标，如混淆矩阵、准确率、精确率、召回率、F1 分数等，或者综合考虑这些指标的多分类版本。对于多分类问题，一些性能指标提供了不同的平均计算方式，其中包括 Micro(微平均)、Macro(宏平均)和 Weighted(加权平均)。这些平均方式是为了综合考虑多类别问题中各类别的贡献。下面是这三种平均方式的解释。

(1) Micro 计算的是总体的真正例、假正例和假负例的数量，然后基于这些总数计算指标。在 Micro 中，每个类别的权重相同。

计算方式：将所有类别的真正例数和假正例数加总，然后计算精确率。适用于各类别的重要性相近，或者类别不平衡的情况。公式：

$$\text{Precision (Micro)} = \text{TP\_total}/(\text{TP\_total} + \text{FP\_total}) \tag{5-11}$$

代码示例：

```
precision_micro = precision_score(y_true, y_pred, average = 'micro')
```

(2) Macro 计算每个类别的指标，然后对这些指标取算术平均值。在 Macro 平均中，每个类别都被同等对待，不考虑类别不平衡的情况。

计算方式：对每个类别单独计算精确率，然后取平均值。适用于各类别的重要性相近的情况。公式：

$$\text{Precision (Macro)} = (\text{Precision\_class1} + \text{Precision\_class2} + \cdots + \text{Precision\_class}N)/N \tag{5-12}$$

代码示例：

```
precision_macro = precision_score(y_true, y_pred, average = 'macro')
```

(3) Weighted 计算每个类别的指标,但是在计算平均值时,会考虑每个类别的样本数量作为权重。这样,样本数量较多的类别对平均值的贡献更大。

计算方式:对每个类别的精确率进行加权平均,权重为各类别的样本数占总样本数的比例。适用于类别不平衡的情况。公式:

$$\text{Precision (Weighted)} = (\text{Precision\_class1} \times \text{w\_class1} + \text{Precision\_class2} \times \text{w\_class2} + \cdots + \text{Precision\_class}N \times \text{w\_class}N)/(\text{w\_class1} + \text{w\_class2} + \cdots + \text{w\_class}N) \tag{5-13}$$

代码示例:

```
precision_weighted = precision_score(y_true, y_pred, average = 'weighted')
```

这些平均方式通常用于多分类问题的评估指标,例如准确率、精确率、召回率、F1 分数等。可以通过设置 average 参数来选择使用哪种平均方式。

在实际应用中,选择哪种平均方式取决于对各类别的关注程度以及样本分布的不同。如果关心每个样本的权重,可以选择 Micro;如果关心每个类别的性能平均,可以选择 Macro;如果关心每个类别的样本量,可以选择 Weighted。

【例 5.6】 多分类模型评估。

```
from sklearn.metrics import precision_score, recall_score, f1_score
y_true = [0, 1, 2, 2, 1, 3, 2,4]
y_pred = [0, 1, 2, 2, 0, 1, 2,2]
#y_true 和 y_pred 是真实标签和预测标签
precision_micro = precision_score(y_true, y_pred, average = 'micro')
precision_macro = precision_score(y_true, y_pred, average = 'macro')
precision_weighted = precision_score(y_true, y_pred, average = 'weighted')

print("Precision (Micro):", precision_micro)
print("Precision (Macro):", precision_macro)
print("Precision (Weighted):", precision_weighted)
```

代码运行结果:

```
Precision (Micro): 0.625
Precision (Macro): 0.35
Precision (Weighted): 0.46875
```

在这个例子中,average 参数用于指定计算平均值的方式。此方式可以应用于 recall_score 和 f1_score。

Scikit-learn 提供了 classification_report()函数,用于生成分类模型的详细性能报告。这个报告包括了准确率、精确率、召回率、F1 分数等多个指标,以及每个类别的详细信息。以下是 classification_report()函数的使用方法。

```
from sklearn.metrics import classification_report

#假设 y_true 是真实标签,y_pred 是模型预测的标签
report = classification_report(y_true, y_pred)

#打印报告
print(report)
```

classification_report()函数提供了以下 6 种具体指标。

(1) 准确率(Accuracy)：所有样本中正确分类的比例。

(2) 精确率(Precision)：预测为正类别的样本中实际为正类别的比例。

(3) 召回率(Recall)：实际为正类别的样本中被正确预测为正类别的比例。

(4) F1 分数(F1-Score)：精确率和召回率的调和平均数，用于综合考虑两者。

(5) 每个类别的精确率、召回率和 F1 分数：对于每个类别，都会计算精确率、召回率和 F1 分数。

(6) 支持度(Support)：每个类别的实际样本数量。

具体来说，classification_report()函数的输出格式如下：

```
                precision  recall  f1-score  support

     class 0      0.81      0.88      0.85       100
     class 1      0.72      0.60      0.65        50

    accuracy                           0.79       150
   macro avg      0.77      0.74      0.75       150
weighted avg      0.78      0.79      0.78       150
```

在这个示例中，class 0 和 class 1 是两个类别的名称。输出中包含了这两个类别的精确率、召回率和 F1 分数，以及整体的准确率、宏平均和加权平均。

5.2 线性回归

线性回归是一种用于建立和预测变量之间线性关系的统计模型。它是监督学习中的一种方法，用于预测一个连续的目标变量(因变量)与一个或多个自变量之间的关系。

线性回归模型可以用于解决多种问题，例如预测房屋价格、销售量、股票价格等。它假设目标变量和自变量之间存在线性关系，并且基于这个假设进行预测和推断。此外，线性回归还可以通过分析回归系数的大小和显著性来评估自变量对目标变量的影响程度。

5.2.1 算法原理

在线性回归中，假设输入特征有 n 个，那么线性回归的目标就是找到一组权重 $\boldsymbol{w}$ 和一个偏置 b，使得预测结果 $y=\boldsymbol{wx}+b$ 与实际结果 y' 最接近。这个距离可以用损失函数来度量，通常使用的是均方误差(MSE)：

$$\mathrm{MSE}=\frac{1}{n}\sum(y-y')^2 \tag{5-14}$$

其中，n 是样本数，y 是预测结果，y' 是实际结果。为了最小化损失函数，需要通过梯度下降等优化方法来调整权重 $\boldsymbol{w}$ 和偏置 b。

线性回归的主要任务是通过拟合训练数据集，估计回归系数的值，从而构建一个线性模型。常用的方法是最小二乘法，即通过最小化目标变量实际观测值与模型预测值之间的差异来确定最佳的回归系数。

5.2.2 算法应用

通过 sklearn.linear_model.LinearRegression 类可以实现一个简单的线性回归模型，以下

是一个简单的代码示例。

【例 5.7】 线性回归代码示例。

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

#创建一个线性回归模型
model = LinearRegression()

#训练模型
x = np.array([[1, 2], [2, 4], [3, 6], [4, 8]])
y = np.array([2, 4, 6, 8])
model.fit(x, y)

#预测新数据
x_new = np.array([[5, 10], [6, 12]])
y_true = np.array([10, 12]) #实际的标签值

y_pred = model.predict(x_new)

#输出预测结果
print("预测结果:", y_pred)

#计算回归模型评估指标
mse = mean_squared_error(y_true, y_pred)
mae = mean_absolute_error(y_true, y_pred)
r2 = r2_score(y_true, y_pred)

#输出模型评估指标
print("均方误差 (MSE):", mse)
print("平均绝对误差 (MAE):", mae)
print("R-squared (R2):", r2)

#输出模型参数
print("模型系数 (coefficients):", model.coef_)
print("模型截距 (intercept):", model.intercept_)
```

运行结果：

```
预测结果: [10. 12.]
均方误差 (MSE): 1.5777218104420236e-30
平均绝对误差 (MAE): 8.881784197001252e-16
R-squared (R2): 1.0
模型系数(coefficients): [0.4 0.8]
模型截距 (intercept): 0.0
```

在这个示例中，首先创建了一个 LinearRegression 模型，然后使用 fit()方法来训练模型。在训练完成后，可以使用 predict()方法来预测新数据。使用 mean_squared_error()、mean_absolute_error()和 r2_score()这三个函数来计算均方误差、平均绝对误差和 R-squared(R2)值。这些指标可以用来评估模型对新数据的预测性能。

注意：线性回归的前提是目标变量和自变量之间存在线性关系，并且数据符合一些基本的假设，如线性性、独立性、常态性和同方差性等。对于非线性关系或违反假设的数据，线性回归可能不适用，需要考虑其他回归模型或进行数据转换和处理。

5.3 逻辑回归

逻辑回归是一种常用的统计学习方法，用于解决二分类问题（将样本分为两个类别）。虽然名字中带有“回归”，但实际上逻辑回归是一种分类算法，而不是回归算法。与线性回归不同的是，逻辑回归的输出是一个概率值，表示输入特征属于正类的概率。

逻辑回归的目标是通过建立一个逻辑函数（也称为 sigmoid 函数）来估计输入特征与目标变量之间的概率关系。它使用了线性回归模型的形式，但通过逻辑函数对结果进行转换，将预测结果限定在 0 到 1 之间。

5.3.1 算法原理

在逻辑回归中，输入特征通过一个线性组合，然后通过一个 sigmoid 函数进行映射，得到输出概率值。sigmoid 函数（见图 5.6）的表达式为：

$$\text{sigmoid}(z)=\frac{1}{1+e^{-z}} \tag{5-15}$$

其中，z 是输入特征的线性组合，即

$$z=\boldsymbol{w}_1\boldsymbol{x}_1+\boldsymbol{w}_2\boldsymbol{x}_2+\cdots+\boldsymbol{w}_n\boldsymbol{x}_n+b \tag{5-16}$$

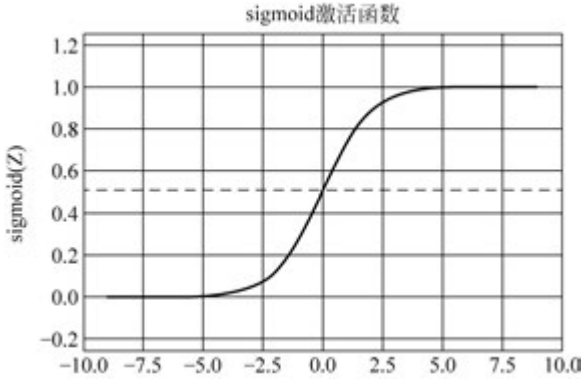

图 5.6 sigmoid 函数（常作为激活函数使用）

可以通过最大化似然函数来调整模型参数，使得模型能够对输入数据进行准确的分类。逻辑回归的优化算法通常使用的是梯度下降等。

5.3.2 算法应用

逻辑回归是通过 LogisticRegression 类来实现的，以下是一个简单的代码示例。

【例 5.8】 逻辑回归代码示例。

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_
matrix

import numpy as np

#创建一个逻辑回归模型
model = LogisticRegression()

#训练模型
```

```
x = np.array([[1, 2], [2, 4], [3, 6], [4, 8]])
y = np.array([0, 0, 1, 1])
model.fit(x, y)

#预测新数据
x_new = np.array([[5, 10], [6, 12]])
y_true = np.array([1, 1]) #实际的标签值

y_pred = model.predict(x_new)

#输出预测结果
print("预测结果:", y_pred)

#输出模型参数
print("模型系数 (coefficients):", model.coef_)
print("模型截距 (intercept):", model.intercept_)

#计算分类模型评估指标
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
conf_matrix = confusion_matrix(y_true, y_pred)

#输出分类模型评估指标
print("准确率 (Accuracy):", accuracy)
print("精确率 (Precision):", precision)
print("召回率 (Recall):", recall)
print("F1 分数 (F1 Score):", f1)
print("混淆矩阵 (Confusion Matrix):\n", conf_matrix)
```

运行结果：

```
预测结果: [1 1]
模型系数(coefficients): [[0.40450393 0.80900786]]
模型截距 (intercept): [ - 5.0563055]
准确率 (Accuracy): 1.0
精确率 (Precision): 1.0
召回率 (Recall): 1.0
F1 分数 (F1 Score): 1.0
混淆矩阵 (Confusion Matrix):
[[2]]
```

在这个示例中，上述代码涉及了使用逻辑回归模型进行二分类任务的训练、预测和评估。以下是对代码中各部分的解释。

(1) 创建逻辑回归模型：

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

导入了 LogisticRegression 类，并创建了一个逻辑回归模型的实例。

(2) 训练模型：

```
x = np.array([[1, 2], [2, 4], [3, 6], [4, 8]])
y = np.array([0, 0, 1, 1])
model.fit(x, y)
```

使用示例数据 x 和对应的二元标签 y 对逻辑回归模型进行训练。

(3) 预测新数据：

```
x_new = np.array([[5, 10], [6, 12]])
y_true = np.array([1, 1])  #实际的标签值
y_pred = model.predict(x_new)
print("预测结果:", y_pred)
```

使用训练好的模型对新数据 x_new 进行预测，并输出预测结果。

(4) 输出模型参数：

```
print("模型系数 (coefficients):", model.coef_)
print("模型截距 (intercept):", model.intercept_)
```

输出训练得到的逻辑回归模型的系数和截距。

(5) 计算分类模型评估指标：

```
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
conf_matrix = confusion_matrix(y_true, y_pred)
```

使用 accuracy_score()、precision_score()、recall_score()、f1_score()和 confusion_matrix()函数计算分类模型的评估指标，包括准确率、精确率、召回率、F1 分数以及混淆矩阵。

(6) 输出分类模型评估指标：

```
print("准确率 (Accuracy):", accuracy)
print("精确率 (Precision):", precision)
print("召回率 (Recall):", recall)
print("F1 分数 (F1 Score):", f1)
print("混淆矩阵 (Confusion Matrix):\n", conf_matrix)
```

输出计算得到的分类模型评估指标的数值。

例 5.8 的代码演示了一个完整的逻辑回归模型的使用流程，包括训练、预测和评估。在实际应用中，评估指标的选择取决于具体问题和任务需求。

5.4 贝叶斯分类

贝叶斯[①](见图 5.7)分类基于贝叶斯定理，贝叶斯定理是一种关于条件概率的数学公式，描述了在给定先验知识的情况下，如何通过新的证据来更新对事件的概率估计。贝叶斯定理在统计学、机器学习和人工智能等领域中广泛应用，尤其在推断和决策问题上有着重要作用。

贝叶斯分类根据已知的先验概率和条件概率，计算样本属于每个类别的后验概率，通过估计每个类别的概率分布以及给定输入样本的条件下每个类别的后验概率来进行分类，并将后验概率最大的类别作为分类结果。

5.4.1 算法原理

贝叶斯分类是基于贝叶斯定理的一种概率统计算法，其核心思想是通过已知的信息来推

① 指的是托马斯·贝叶斯(Thomas Bayes，1702—1761 年)是一位英国数学家和神父，他的名字常常与贝叶斯定理相关联，这个定理在概率统计领域中起着重要的作用。

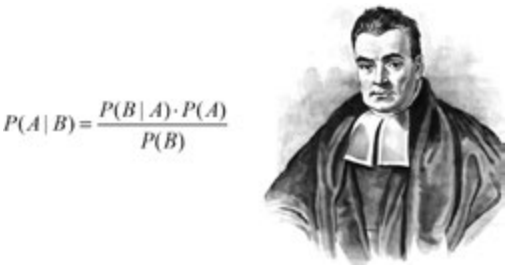

图 5.7 托马斯·贝叶斯(Thomas Bayes)

断未知的信息。该定理描述了在已知先验信息的情况下,如何更新这些信息以得到后验概率。对于事件 A 和 B,贝叶斯定理可以表示为:

$$P(\mathrm{A}\mid\mathrm{B})=\frac{P(\mathrm{B}\mid\mathrm{A})\times P(\mathrm{A})}{P(\mathrm{B})} \tag{5-17}$$

其中,$P(\mathrm{A}|\mathrm{B})$是在已知 B 的情况下 A 发生的概率,$P(\mathrm{B}|\mathrm{A})$是在已知 A 的情况下 B 发生的概率,$P(\mathrm{A})$是 A 发生的先验概率,$P(\mathrm{B})$是 B 发生的概率。

贝叶斯分类的应用场景包括文本分类、垃圾邮件过滤、情感分析、信用评级等领域。例如,在文本分类中,可以根据每个单词在不同文本类别中的出现概率计算后验概率,从而将文本分类到具有最高后验概率的类别中。

在实际应用中,贝叶斯分类通常使用朴素贝叶斯分类器。它假设特征之间是独立的,从而简化了计算过程,减少了计算量。通过使用先验概率和条件概率,朴素贝叶斯分类器可以对新的数据进行分类。Scikit-learn 提供了多个常用的贝叶斯分类模型。

(1) GaussianNB:高斯朴素贝叶斯,适用于特征为连续变量的分类任务。

(2) MultinomialNB:多项式朴素贝叶斯,适用于特征为计数变量的分类任务。

(3) BernoulliNB:伯努利朴素贝叶斯,适用于特征为二元变量的分类任务。

(4) ComplementNB:补充朴素贝叶斯,是一种对 MultinomialNB 的改进。

这些模型都是基于贝叶斯定理实现的分类器,可以在许多应用中使用,如文本分类、垃圾邮件过滤、情感分析等。

5.4.2 算法应用

以下是一个使用 Scikit-learn 实现朴素贝叶斯分类器的代码示例,用于对鸢尾花数据集进行分类。

【例 5.9】 朴素贝叶斯分类器的代码示例。

```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#加载数据集
iris = load_iris()
X = iris.data
y = iris.target

#划分数据集为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#建立高斯朴素贝叶斯分类器
```

```
clf = GaussianNB()

#训练模型
clf.fit(X_train, y_train)

#预测测试集
y_pred = clf.predict(X_test)

#计算准确率
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

这段代码演示了使用高斯朴素贝叶斯分类器进行分类任务的基本步骤。以下是代码的解释。

(1) 导入所需的库和模块：

```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

(2) 加载鸢尾花数据集：

```
iris = load_iris()
X = iris.data
y = iris.target
```

通过 load_iris()函数加载鸢尾花数据集，将特征保存在 X 中，将目标变量保存在 y 中。

(3) 划分数据集为训练集和测试集：

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

使用 train_test_split()函数将数据集划分为训练集和测试集。test_size=0.2 表示将20%的数据用于测试，random_state=42 是为了确保每次运行代码时得到的划分结果一致。

(4) 建立高斯朴素贝叶斯分类器：

```
clf = GaussianNB()
```

创建高斯朴素贝叶斯分类器对象。

(5) 训练模型：

```
clf.fit(X_train, y_train)
```

使用训练集数据 X_train 和对应的目标值 y_train 来训练朴素贝叶斯分类器。

(6) 预测测试集：

```
y_pred = clf.predict(X_test)
```

使用训练好的模型对测试集数据 X_test 进行预测，得到预测结果 y_pred。

(7) 计算准确率：

```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

使用 accuracy_score()函数计算模型在测试集上的准确率，并打印结果。

例 5.9 的代码展示了一个简单的机器学习流程，包括数据加载、数据划分、模型训练、预测

和性能评估。在这个例子中,使用的是高斯朴素贝叶斯分类器来解决鸢尾花分类问题。

5.5 决策树

决策树是一种基于树状结构的监督学习算法,用于建立分类或回归模型。在分类问题中,决策树通过对特征的分裂,将数据分成不同的类别;在回归问题中,决策树通过对特征的分裂,建立一个预测模型。

决策树的优点在于它们易于理解和解释,可以可视化,对于具有高维度特征的数据集也很有效,而且也可以处理缺失数据。决策树算法在数据挖掘、机器学习、模式识别等领域得到了广泛应用。

5.5.1 算法原理

决策树算法的核心是选择最佳的特征进行分裂。在每次分裂时,算法都会计算每个特征的信息增益或信息增益比,以确定哪个特征最能有效地将数据集分成不同的类别或预测结果。在分类问题中,通常使用信息熵或基尼不纯度作为衡量分裂效果的指标;在回归问题中,通常使用平方误差或平均绝对误差等指标来衡量分裂效果。决策树的构建过程是递归的,直到达到某个停止条件。停止条件通常为:

- 所有数据属于同一类别或拥有相同的预测结果;
- 已经到达了预定义的树的深度;
- 数据集中的样本数小于某个预定义值。

在决策树的构建过程中,可以使用剪枝技术对树进行优化。剪枝技术分为预剪枝和后剪枝两种方法。预剪枝是在构建树的过程中,通过设置一些限制条件来避免过拟合;后剪枝是在构建好树之后,通过移除一些子树来降低复杂度。剪枝技术可以有效地防止过拟合问题,提高模型的泛化能力。

总的来说,决策树算法是一种简单但强大的分类和回归算法,具有易于理解和解释、可视化、处理缺失数据等优点。

5.5.2 算法应用

在 Scikit-learn 中,可以使用 DecisionTreeClassifier 和 DecisionTreeRegressor 分别进行分类和回归任务的建模。使用这些类可以轻松地创建和训练决策树模型,也可以使用诸如可视化决策树等方法来解释和理解模型。为了方便观察,可以使用 graphviz 库将决策树绘制出来,首先需要安装 graphviz 库。安装命令为:

```
pip install graphviz
```

下面是一个简单的 Scikit-learn 决策树的代码示例,用于演示如何使用决策树分类器对鸢尾花数据集进行分类。

【例 5.10】 决策树的代码示例。

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import graphviz

#加载数据集
```

```
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)

#定义决策树分类器
clf = DecisionTreeClassifier()

#训练分类器
clf.fit(X_train, y_train)

#使用分类器进行预测
y_pred = clf.predict(X_test)

#计算分类器的准确率
accuracy = clf.score(X_test, y_test)
print("Accuracy:", accuracy)

#将决策树导出为 Graphviz 格式
dot_data = export_graphviz(clf,out_file = None, feature_names = iris.feature_names, class_names
    = iris.target_names, filled = True, rounded = True, special_characters = True)

#使用 Graphviz 库绘制决策树
graph = graphviz.Source(dot_data)
graph.render("iris_decision_tree", format = "png") #保存为 PNG 图片文件
graph.view("iris_decision_tree")                   #打开可视化窗口
```

程序运行结果：

```
Accuracy: 1.0
```

在这个例子中，首先加载了鸢尾花数据集，然后使用 train_test_split()函数将数据集划分为训练集和测试集。接着，定义了一个决策树分类器并训练它，最后使用训练好的模型对测试集进行预测，并计算分类器的准确率。

例 5.10 只是一个简单的例子，决策树算法可以用于更复杂的分类和回归问题。通过掌握该算法，我们可以轻松地使用决策树算法来解决各种问题。

5.6 支持向量机

支持向量机(SVM)是一种基于统计学习理论的分类和回归算法。SVM 可以有效地处理高维空间中的数据，并且在处理小样本数据时表现出色。SVM 的核心思想是通过构建一个超平面，将不同类别的样本分开。

5.6.1 算法原理

SVM 的算法原理如图 5.8 所示。对于给定的训练数据，SVM 找到一个最优的超平面，使得不同类别的数据离超平面最近的点到该超平面的距离(即间隔)最大化。这个超平面可以表示为：

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0 \tag{5-18}$$

其中 $\boldsymbol{w}$ 是超平面的法向量，b 是偏移量；SVM 的目标是最大化边界(即支持向量到超平面的

距离),并使得分类错误的样本点的误差最小化。

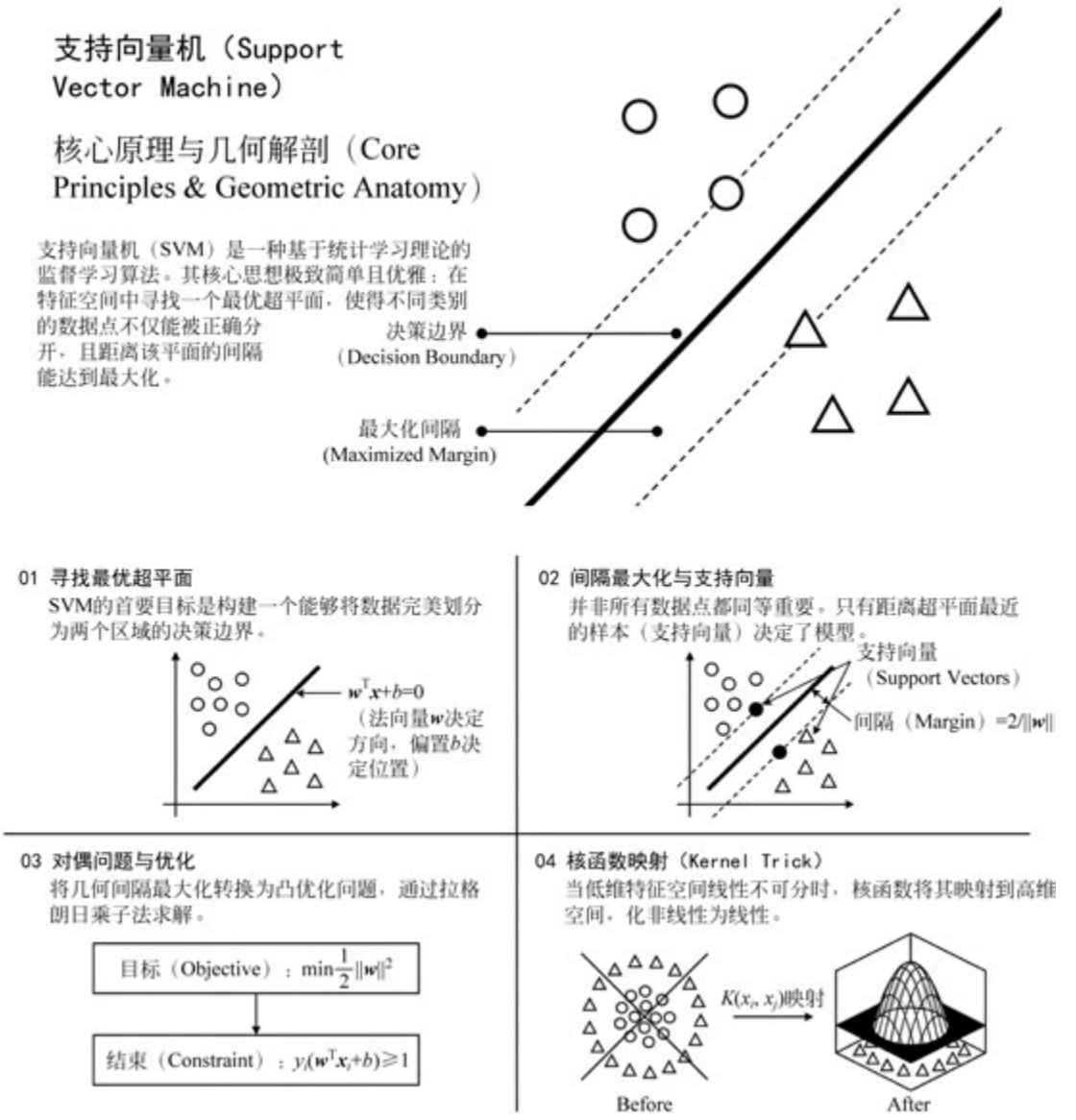

图 5.8 SVM 原理图示

为了实现这个目标,需要在最大化边界的同时,将每个样本点的间隔与边界距离之和最小化;在实际应用中,往往需要使用核函数将数据从低维空间映射到高维空间,以便于更好地分离不同类别的数据。具体算法步骤如下。

1. 数据准备

SVM 使用的数据是一个包含 N 个样本的训练集,每个样本有 M 个特征。对于分类问题,每个样本还有一个对应的类别标签。

2. 寻找最优的超平面

SVM 的目标是找到一个最优的超平面,将不同类别的样本尽可能地分开。在二分类问题中,超平面是一个$(M-1)$维的决策边界,可以将两个类别的样本分为两个不同的区域。

SVM 将样本映射到高维特征空间,并在该空间中寻找一个最优的超平面。常用的映射方

法包括线性映射、多项式映射和径向基函数(Radial Basis Function,RBF)映射。

3. 最大间隔分类器

SVM 通过最大化间隔来构建一个最优的分类器,间隔是指超平面与最靠近它的样本点之间的距离。

4. 支持向量

在 SVM 中,离超平面最近的一些样本点被称为支持向量。这些样本点对于定义超平面起到关键作用,它们决定了超平面的位置和形状。支持向量用于构建决策边界,并且在预测阶段起到关键作用。

5. 核函数

SVM 可以通过核函数来处理非线性可分的问题。核函数可以将低维特征空间中的样本映射到高维特征空间,使得原本线性不可分的样本在高维空间中线性可分。

常用的核函数有线性核、多项式核和 RBF 核等,可以根据问题的性质和数据的特点选择合适的核函数。

6. 对偶问题与优化求解

SVM 的原始问题是一个凸优化问题,可以通过拉格朗日乘子法将其转换为对偶问题。对偶问题可以通过求解对偶优化问题来得到最优的超平面参数。

7. 预测

在预测阶段,对于新的样本点,SVM 通过计算其与超平面的位置关系,将其分为不同的类别。

SVM 算法具有较好的泛化性能和鲁棒性,并且在处理小样本和高维数据时表现良好。此外,SVM 还具有以下特点。

(1) 只依赖于支持向量,而不依赖于整个数据集,因此对于大规模数据集的存储和计算要求较低。

(2) 由于最大化间隔的特性,SVM 对于噪声和异常点的鲁棒性较高。

(3) SVM 对于非线性问题具有较强的拟合能力,通过核函数可以将非线性问题映射到高维特征空间。

(4) 需要注意的是,SVM 算法中的一些重要参数包括正则化参数 C、核函数类型和参数等,这些参数的选择会影响模型的性能。在应用 SVM 算法时,需要根据具体问题和数据集进行合适的参数调优。

总结起来,SVM 通过构建最优的超平面将不同类别的样本分开,具有较好的泛化能力和鲁棒性。它是一种常用的分类和回归算法,在许多实际应用中取得了良好的效果。

5.6.2 算法应用

Scikit-learn 提供了 SVM 分类器和回归器的工具。下面是一个使用 SVM 分类器对鸢尾花数据集进行分类的代码示例。

【例 5.11】 使用 SVM 分类器的代码示例。

```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

#加载数据集
```

```
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)

#定义 SVM 分类器
clf = SVC(kernel = 'linear')

#训练分类器
clf.fit(X_train, y_train)

#使用分类器进行预测
y_pred = clf.predict(X_test)

#计算分类器的准确率
accuracy = clf.score(X_test, y_test)
print("Accuracy:", accuracy)
```

在这个例子中，首先加载了鸢尾花数据集，然后使用 train_test_split()函数将数据集划分为训练集和测试集。接着，定义一个 SVM 分类器并训练它，最后使用训练好的模型对测试集进行预测，并计算分类器的准确率。

需要注意的是，SVM 分类器和回归器有很多参数可以调整，如核函数的类型、正则化参数 C 的大小等。在实际应用中，需要根据具体问题选择合适的参数。

5.7 *K* 近邻算法

K 近邻算法(K-Nearest Neighbor，KNN)是一种常用的基于实例的监督学习算法。KNN 的基本思想是根据距离度量来对未知样本进行分类，即将未知样本分类为距离其最近的 K 个已知样本中出现最多的类别。KNN 算法非常简单，但在实际应用中具有广泛的用途。

5.7.1 算法原理

KNN 算法的原理如下。

1. 数据准备

KNN 算法使用的数据是一个包含 N 个样本的训练集，每个样本有 M 个特征。对于分类问题，每个样本还有一个对应的类别标签。

2. 计算距离

对于一个未知样本，KNN 算法首先计算它与训练集中每个样本的距离。常用的距离度量方法包括欧几里得距离、曼哈顿距离、闵可夫斯基距离等。

3. 选择最近的 *K* 个邻居

从训练集中选择距离最近的 K 个样本作为该未知样本的邻居。这些邻居被称为最近邻(Nearest Neighbor)。

4. 进行分类或回归

对于分类问题，KNN 算法采用投票的方式，统计 K 个邻居中各类别的出现次数，将出现次数最多的类别作为该未知样本的预测类别。

对于回归问题，KNN 算法采用取平均值的方式，计算 K 个邻居的目标值的平均值作为该

未知样本的预测值。

5. 输出预测结果

将预测的类别或回归值作为最终的预测结果。

KNN 算法的核心思想是基于邻居的投票或平均值来进行分类或回归的。它的优势在于简单易理解、无须训练过程和模型参数的选择。同时，KNN 算法对于异常值和噪声数据相对鲁棒。

需要注意的是，KNN 算法中的一个重要参数是 K 值，即选择的最近邻居的数量。K 值的选择会影响算法的性能，较小的 K 值会增加模型的复杂度和噪声敏感性，较大的 K 值会增加模型的偏差和欠拟合风险。因此，在应用 KNN 算法时，需要根据具体问题和数据集选择合适的 K 值。

5.7.2 算法应用

Scikit-learn 提供了 KNN 分类器和回归器的工具。下面是一个使用 KNN 分类器对鸢尾花数据集进行分类的代码示例。

【例 5.12】 KNN 的代码示例。

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

#加载数据集
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)

#定义 KNN 分类器
clf = KNeighborsClassifier(n_neighbors = 3)

#训练分类器
clf.fit(X_train, y_train)

#使用分类器进行预测
y_pred = clf.predict(X_test)

#计算分类器的准确率
accuracy = clf.score(X_test, y_test)
print("Accuracy:", accuracy)
```

在这个例子中，首先加载了鸢尾花数据集，然后使用 train_test_split()函数将数据集划分为训练集和测试集。接着，定义了一个 KNN 分类器并训练它，最后使用训练好的分类器对测试集进行预测，并计算了分类器的准确率。需要注意的是，KNN 算法的一个重要参数是 K，即近邻的数量。在实际应用中，需要根据具体问题选择合适的 K 值。

5.8 综合案例

使用监督学习常用的算法得到婚嫁模型，预测样本数据是否选择婚嫁。

【例 5.13】 KNN 的代码示例。

(1) 读取样本数据(见图 5.9):

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
#读取数据
data = pd.read_csv('11.csv')
```

(2) 将分类变量转换为数字(见图 5.10):

```
le = LabelEncoder()
data['颜值'] = le.fit_transform(data['颜值'])
data['性格'] = le.fit_transform(data['性格'])
data['上进心'] = le.fit_transform(data['上进心'])
data['嫁否'] = le.fit_transform(data['嫁否'])
```

	颜值	性格	身高	上进心	嫁否
0	帅	不好	165	不上进	不嫁
1	不帅	好	171	上进	不嫁
2	帅	好	172	上进	嫁
3	不帅	爆好	185	上进	嫁
4	帅	不好	170	上进	不嫁
5	帅	不好	172	上进	不嫁
6	帅	好	182	不上进	嫁
7	不帅	好	175	上进	嫁
8	帅	爆好	176	上进	嫁
9	不帅	不好	186	上进	嫁
10	帅	好	172	不上进	不嫁
11	帅	好	171	不上进	不嫁

图 5.9 样本数据

	颜值	性格	身高	上进心	嫁否
0	1	0	165	1	0
1	0	1	171	0	0
2	1	1	172	0	1
3	0	2	185	0	1
4	1	0	170	0	0
5	1	0	172	0	0
6	1	1	182	1	1
7	0	1	175	0	1
8	1	2	176	0	1
9	0	0	186	0	1
10	1	1	172	1	0
11	1	1	171	1	0

图 5.10 转换后的样本数据

(3) 分离特征和标签:

```
X = data.iloc[:, 0:-1]
y = data.iloc[:, -1]
#分割数据为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
#导入必要的模块
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
```

(4) 逻辑回归:

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print('Logistic Regression Accuracy:', acc)
print('Logistic Regression F1-score:', f1)
```

（5）决策树：

```
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print('Decision Tree Accuracy:', acc)
print('Decision Tree F1 - score:', f1)
```

（6）贝叶斯：

```
svm = GaussianNB()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print('SVM Accuracy:', acc)
print('SVM F1 - score:', f1)
from sklearn.model_selection import cross_val_score
clf = GaussianNB()
scores = cross_val_score(clf, X, y, cv = 5)
scores
```

这里小结一下，常用的监督学习算法有线性回归、逻辑回归、决策树、支持向量机和 K 近邻算法等，适用于不同场景和数据类型。

- 线性回归：适用于连续数值预测问题，如房价预测、股票价格预测等。对于特征数量很大的数据集，可以使用正则化来防止过拟合。
- 逻辑回归：适用于二分类问题和多分类问题，如信用卡欺诈检测、癌症诊断等。对于特征数量很大的数据集，可以使用 L1 或 L2 正则化来防止过拟合。
- 贝叶斯分类：在处理小数据、文本分类、多分类、噪声较大和实时分类等方面表现良好。但是，该算法也有一些局限性，例如对于特征之间存在复杂依赖关系的问题，该算法可能表现不佳。
- 决策树：适用于分类和回归问题，尤其是非线性关系的问题。决策树也可以用于特征选择，可以通过分析树的结构来确定哪些特征最重要。
- 随机森林：适用于分类和回归问题，可以处理高维数据和大型数据集。随机森林可以减少过拟合的风险，还可以用于特征选择。
- 支持向量机：适用于二分类问题和多分类问题，可以处理高维数据集和非线性关系的问题。该算法可以使用不同的内核函数来适应不同的数据分布。
- K 近邻算法：适用于分类和回归问题，可以处理小型数据集和多分类问题。该算法的训练时间很短，但在预测时需要计算距离，因此对于大型数据集的预测速度较慢。

总之，选择适合的监督学习算法取决于数据的性质、特征数量、样本数量等因素。在实际应用中，可以通过交叉验证等方法来比较不同算法的性能，并选择最适合的算法。

5.9 本章小结

通过本章的学习，读者了解到监督学习是机器学习中的一项核心技术，它对数据进行分析和模式识别以做出预测或判决。监督学习流程的关键步骤包括数据预处理、模型选择、模型训练与评估，以及参数调优；通过探讨线性回归、逻辑回归、贝叶斯分类、决策树和支持向量机等

经典算法的原理和应用，展现了如何融合各种监督学习技术来解决实际问题。通过本章的学习，读者能够对监督学习的复杂性有深入的理解，并能够应用其原理解决具体问题。

5.10 习题

5.10.1 选择题

1. 监督学习是一种机器学习范式，其特点是(　　)。
   A. 无须标签数据　　B. 仅使用无监督数据
   C. 使用标签数据进行训练和预测　　D. 不需要模型评估指标
2. 监督学习的流程步骤包括(　　)。
   A. 特征提取、模型评估、数据归一化
   B. 数据清洗、模型训练、特征选择
   C. 数据收集、数据预处理、模型训练
   D. 特征选择、数据归一化、模型评估
3. 监督学习按照问题类型可以分为(　　)。
   A. 分类、聚类、回归　　B. 分类、聚类、降维
   C. 回归、聚类、降维　　D. 分类、回归、聚类
4. 在监督学习中，用于评估模型性能的指标有(　　)。
   A. 准确率、召回率、F1 分数
   B. 方差、均方误差、平均绝对误差
   C. 轮廓系数、DB 指数、互信息
   D. 离散系数、调整兰德指数、归一化互信息
5. 线性回归用于解决(　　)。
   A. 分类问题　　B. 聚类问题　　C. 回归问题　　D. 降维问题
6. 逻辑回归用于解决(　　)。
   A. 分类问题　　B. 聚类问题　　C. 回归问题　　D. 降维问题
7. 贝叶斯分类方法基于(　　)。
   A. 概率论　　B. 线性代数　　C. 图论　　D. 微积分
8. 决策树用于解决(　　)。
   A. 分类问题　　B. 聚类问题　　C. 回归问题　　D. 降维问题
9. 随机森林是一种集成学习算法，它的基本单元是(　　)。
   A. 决策树　　B. 支持向量机　　C. K 近邻算法　　D. 线性回归模型
10. 支持向量机适用于(　　)。
    A. 分类问题　　B. 聚类问题　　C. 回归问题　　D. 降维问题
11. K 近邻算法的基本思想是(　　)。
    A. 基于数据样本之间的距离进行分类
    B. 基于数据样本之间的相似度进行分类
    C. 使用决策树进行分类
    D. 使用贝叶斯分类进行分类
12. 监督学习算法中的“监督”一词指的是(　　)。
    A. 算法是由监督者训练的　　B. 算法使用标签数据进行训练

C. 算法需要监督者进行指导　　D. 算法需要进行监控和调整

13. 监督学习的应用领域包括(　　)。

A. 图像分类、自然语言处理、异常检测

B. 数据聚类、模式识别、特征提取

C. 机器翻译、数据压缩、数据可视化

D. 网络安全、人脸识别、数据挖掘

14. 监督学习中,模型评估指标"准确率"是指(　　)。

A. 预测结果中正确分类的样本数与总样本数之比

B. 预测结果中真正例与真正例和假负例之和之比

C. 预测结果中真正例与真正例和假正例之和之比

D. 预测结果中真负例与真负例和假正例之和之比

15. 线性回归的算法原理是(　　)。

A. 基于数据样本之间的距离进行分类

B. 基于数据样本之间的相似度进行分类

C. 拟合一个线性方程来预测连续型数值数据

D. 使用决策树进行分类

16. 逻辑回归的算法原理是(　　)。

A. 拟合一个线性方程来预测连续型数值数据

B. 基于数据样本之间的距离进行分类

C. 基于数据样本之间的相似度进行分类

D. 使用决策树进行分类

17. 贝叶斯分类算法是一种概率模型,它基于(　　)。

A. 线性代数　　B. 图论　　C. 概率论　　D. 微积分

18. 决策树的算法原理是(　　)。

A. 拟合一个线性方程来预测连续型数值数据

B. 基于数据样本之间的距离进行分类

C. 基于数据样本之间的相似度进行分类

D. 使用树状结构进行分类

19. 随机森林算法是一种集成学习算法,它的基本单元是(　　)。

A. 决策树　　B. 支持向量机　　C. K 近邻算法　　D. 线性回归模型

20. 支持向量机适用于(　　)。

A. 分类问题　　B. 聚类问题　　C. 回归问题　　D. 降维问题

5.10.2 编程题

1. 使用的 LinearRegression 模型,利用给定的特征矩阵和标签数据进行线性回归模型的训练,并输出模型的系数和截距。

2. 使用的 LogisticRegression 模型,利用给定的特征矩阵和标签数据进行逻辑回归模型的训练,并输出模型的系数和截距。

3. 使用的 MultinomialNB 模型,利用给定的特征矩阵和标签数据进行贝叶斯分类模型的训练,并输出模型的分类准确率。

4. 使用的 DecisionTreeClassifier 模型,利用给定的特征矩阵和标签数据进行决策树分类

模型的训练，并输出模型的准确率。

5. 使用的 RandomForestClassifier 模型，利用给定的特征矩阵和标签数据进行随机森林分类模型的训练，并输出模型的准确率。

6. 使用的 SVC 模型，利用给定的特征矩阵和标签数据进行支持向量机分类模型的训练，并输出模型的准确率。

7. 使用的 KNeighborsClassifier 模型，利用给定的特征矩阵和标签数据进行 K 近邻分类模型的训练，并输出模型的准确率。

8. 创建一个包含两个特征的 DataFrame，并使用 sklearn. model_selection 的 train_test_split()函数将数据集划分为训练集和测试集。

9. 创建一个包含标签数据的 Series，并使用 sklearn. preprocessing 的 LabelEncoder 将标签数据进行编码。

10. 使用 sklearn. metrics 的 accuracy_score()函数计算模型预测结果的准确率。