机器学习

一、机器学习概述

知识点 1 机器学习的定义

定义 1：一个计算机程序，利用经验 E 来学习任务 T，性能表现为 P，如果针对任务 T 的性能 P 能随着经验 E 不断增长，则称为机器学习。

定义 2：通过不断地喂养数据，让算法变得越来越聪明，就叫做机器学习。

机器学习是 AI 的子集，是一门通过计算算法构建统计模型，使系统能够从数据中自主识别规律并优化性能的学科。

机器学习的发展历程是从基于规则与统计学习的早期探索，演进到以大数据和深度神经网络为代表的现代人工智能的飞跃过程。

知识点 2 机器学习的分类

知识点 3 机器学习的应用场景

●医疗行业

应用案例：

（1）疾病诊断与预测：通过分析患者的病历、影像数据（如 X 光片、CT 扫描）和基因数据，机器学习可以帮助医生诊断疾病（如癌症、心脏病）或预测疾病风险。IBM Watson Health 使用机器学习帮助医生制定癌症治疗方案。

（2）药物开发：机器学习可以加速新药的研发过程，通过分析化学分子结构和生物数据，预测哪些药物可能有效。DeepMind 的 AlphaFold 使用机器学习预测蛋白质结构，帮助研发新药。

●交通行业

应用案例：

（1）短期的交通预测和流量控制，在历史交通信息的基础上预测给定时间内的机动车数量，将准确的实时交通预测提供给道路使用者和相关交通部门。谷歌地图使用机器学习预测实时交通状况。

（2）自动驾驶。分析路面上的不同对象并选择车辆控制方式。

●金融行业

应用案例：

金融领域中，使用机器学习算法完成风险评估与信用评分。银行使用机器学习分析客户的收入、消费习惯和信用记录，评估贷款风险并决定是否批准贷款。诸如蚂蚁金服使用机器学习为用户提供信用评分（芝麻信用）。

●零售与电商行业

应用案例：

（1）个性化推荐：电商平台（如淘宝、亚马逊）使用机器学习分析用户的浏览和购买记录，推荐用户可能感兴趣的商品。Netflix 使用推荐系统为用户推荐电影和电视剧。

（2）库存管理与需求预测：零售商使用机器学习预测未来商品的需求量，优化库存管理，避免缺货或积压。沃尔玛使用机器学习优化供应链管理。

●农业

应用案例：

（1）精准农业：通过分析土壤、天气和作物生长数据，机器学习可以帮助农民优化灌溉、施肥和收割。John Deere 使用机器学习优化农业机械的操作。

（2）病虫害预测：机器学习可以分析农田的传感器数据和卫星图像，预测病虫害的发生，帮助农民提前采取措施。IBM 的 Watson Decision Platform 为农民提供病虫害预测服务。

知识点 4 机器学习的数据集

数据集来源

数据集通常由以下几个部分组成

特征（Features）：也叫输入数据，是模型用来学习的“线索”。比如，预测房价时，**房子的面积、位置、房间数量就是特征**。**一句话记住它：特征是那些你知道的、能看到的信息。**

标签（Labels）：**也叫目标值，是模型需要预测的结果**。比如，房价本身就是一个标签。**一句话记住它：标签是你想知道但还不知道的答案。**

样本（Samples）：每一行数据就是一个样本，代表一个具体的例子。比如，一套房子的信息就是一个样本。**一句话记住它：样本就是一个具体的例子或一条完整的记录。**

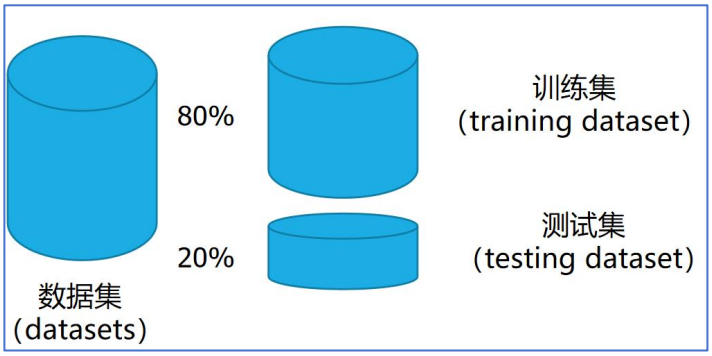
	房子面积	房子位置	房子楼层	房子朝向	目标值
数据1	80	9	3	0	80
数据2	100	9	5	1	120
数据3	80	10	3	0	100

根据数据的用途，数据集可以分为以下几种：

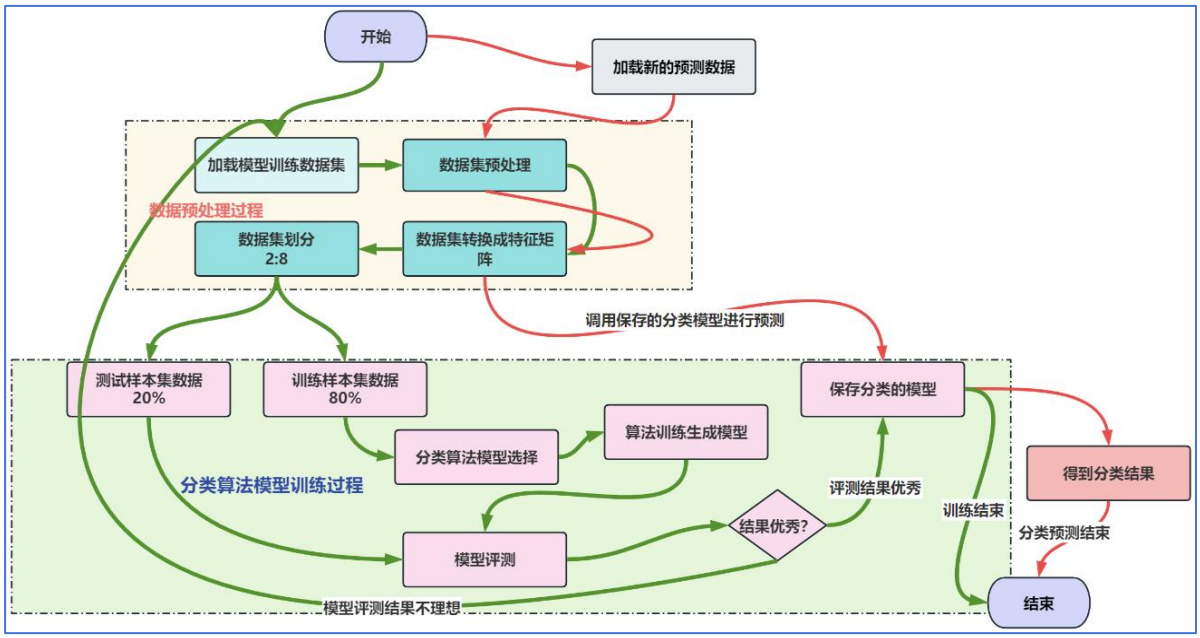
训练集（Training Set）：用来训练模型，让模型学会从特征中预测标签。

验证集（Validation Set）：用来调整模型的参数，防止模型过拟合（即模型只记住了训练数据，但不会处理新数据）。

测试集（Test Set）：用来评估模型的最终性能，测试模型是否能够很好地处理新数据



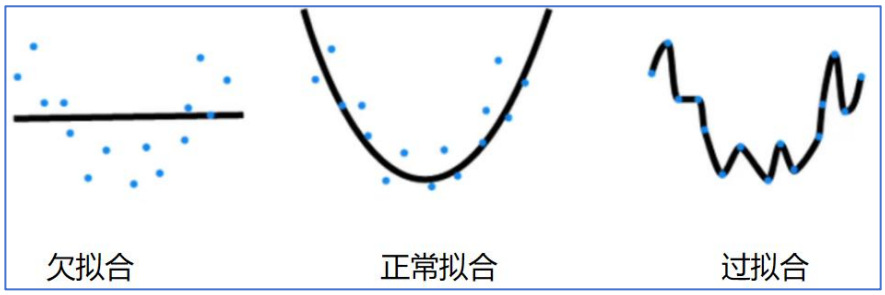
知识点 5 机器学习的流程



知识点 6 过拟合和欠拟合

欠拟合：模型太简单，没有学到数据中的规律，导致在训练集和测试集上表现都很差。原因通常是：模型复杂度过低；特征量过少。

过拟合：模型太复杂，把训练集中的噪声和细节都记住了，导致在训练集上表现很好，但在测试集上表现很差。原因通常是：训练数据集样本单一，样本不足；训练数据中噪声干扰过大；模型过于复杂。



不同拟合状态在数据集中的表现

	欠拟合	正常拟合	过拟合
训练集 (training dataset)	低 30%	高 90%	很高 98%
测试集 (testing dataset)	低 26%	高 88%	较低 56%

二、 分类算法

知识点 1 五大经典算法对比总览

算法	最佳适用场景	数据特征要求	优点	局限性
逻辑回归	概率预测、二分类问题	线性可分、特征独立	可解释性强、输出概率	无法处理复杂非线性关系
K近邻	模式识别、推荐系统	特征尺度一致、数据密度均匀	简单直观、无需训练	计算成本高、维度灾难
决策树	规则提取、分类回归	混合数据类型、缺失值容忍	可解释性极强、非线性	容易过拟合、不稳定
朴素贝叶斯	文本分类、垃圾过滤	特征条件独立	计算高效、小样本表现好	独立性假设太强
支持向量机	小样本、高维数据	特征数值化、尺度标准化	边界清晰、泛化能力强	大规模数据训练慢

分类算法选择的时机

相关知识链接

知识点 2 数据的特征处理

特征处理就像给数据"化妆"和"整理"，让机器学习算法能更好地理解和学习数据中的规律。就像我们要把食材清洗、切块后才能烹饪一样，原始数据也需要经过处理才能被算法有效使用。

相关知识链接

知识点 3 分类算法的评估

在机器学习中，仅仅训练一个模型是不够的。我们需要知道这个模型在现实世界中的表现如何。评估指标就像考试的"评分标准"，帮助我们判断模型的好坏，比较不同模型的性能，并指导我们改进模型。

不同的评估指标从不同角度衡量模型性能，有些关注整体准确率，有些关注特定类别的表现，有些则关注模型的预测概率质量。

相关知识链接

知识点 4 逻辑回归算法

逻辑回归是一个“侦探模型”，它通过分析数据特征，预测某个事件发生的概率（比如用户是否贷款违约、邮件是否是垃圾邮件）。

核心思想

输入：特征数据（如年龄、收入、负债）

输出：概率值（0~1 之间，例如 0.8 表示有 80%的可能性属于正类）

关键公式：

$$P(y=1)=\frac{1}{1+e^{-(w_1x_1+w_2x_2+...+b)}}$$

比喻：就像用“加权打分”判断是否通过考试，分数超过某个阈值就判定为“通过”。它的原理是通过一个特殊的函数（逻辑函数），把输入的数据转化为一个概率值，这个概率值表示数据属于某一类的可能性。如果概率值大于 0.5，我们就认为数据属于这一类；如果小于 0.5，就属于另一类。

优缺点

优点：计算速度快、结果可解释性强

缺点：只能处理线性可分问题（复杂的非线性问题需要其他模型）

逻辑回归参数设定

逻辑回归是机器学习中最常用的分类算法之一，但它的参数名称常常让人困惑。

相关知识链接

实战案例：用户信用检测

任务目标

基于用户年龄、收入、负债等特征，预测其信用是否良好。

数据集

字段：年龄、月收入、负债比率、历史逾期次数、信用标签（0=差，1=好）样本量 3000 条数据。

模型训练实现代码

```
import pandas as pd
```

```python
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
roc_auc_score
import joblib
from sklearn.model_selection import GridSearchCV
import warnings
# ================ 模型参数优化 ================
def model_optimization(X_train, y_train, model):
    param_grid = {
        'classifier__C': [0.001, 0.01, 0.1, 1, 10],
        'classifier__penalty': ['l1', 'l2'],
        'classifier__solver': ['liblinear', 'saga','lbfgs'],
        'classifier__max_iter': [100,  1000]
    }
    # 忽略收敛警告
    with warnings.catch_warnings():
        print("正在参数寻优中......")
        warnings.filterwarnings("ignore", category=ConvergenceWarning)
        grid_search = GridSearchCV(model, param_grid, cv=5)
        grid_search.fit(X_train, y_train)
        print("最优参数:", grid_search.best_params_)
    return grid_search


def feature_engineering():
    # ================ 特征工程 ================
    # 定义特征类型
    numeric_features = ["年龄", "收入", "负债比率"]
    categorical_features = ["工作类型", "信用历史"]
    # 构建预处理管道
    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), numeric_features),
```

```
            ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features)
        ])
    return preprocessor
# ================= 逻辑回归模型训练 =================
def train_logistic_regression(X, y):
    # ================= 训练数据划分 =================
    X_train, X_test, y_train, y_test = (
        train_test_split(X, y, test_size=0.2,  random_state=42))
    preprocessor = feature_engineering()
    pipline = Pipeline(
        steps=[("preprocessor", preprocessor),
               ("classifier", LogisticRegression())]
    )
    # ======= 进行模型的参数寻优 根据寻优结果重新定义 =================
    grid_search = model_optimization(X_train, y_train, pipline)
    # 训练模型
    best_model = grid_search.best_estimator_
    best_model.fit(X_train, y_train)
    # ================= 模型训练模型 =================
    y_pred = best_model.predict(X_test)
    y_proba = best_model.predict_proba(X_test)[:, 1]
    # ================ 训练模型的评估  =================
    print("测试集准确率:",
          round(accuracy_score(y_test, y_pred),3)
          )
    print("\n分类报告:")
    print(classification_report(y_test, y_pred,target_names=['优','差']))
    print("\nAUC-ROC:", round(roc_auc_score(y_test, y_proba),3))
    # 保存训练好的模型
    joblib.dump(best_model, 'credit_prediction_model.joblib')


if __name__ == '__main__':
    # 读取训练数据
    train_data = pd.read_csv('train_data_set/customer_credit_data_train.csv')
    # 定义特征和目标变量
    X = train_data.drop('信用标签', axis=1)
    y = train_data['信用标签']
```

```
    # 进行逻辑回归模型训练
    train_logistic_regression(X, y)
    print("模型训练完成！")
```

模型预测实现代码

```
import joblib
import pandas as pd
def predict_credit(test_data):
    # 加载模型
    model = joblib.load("credit_prediction_model.joblib")
    # 使用模型进行预测
    prediction = model.predict(test_data)
    return prediction

if __name__ == '__main__':
    # 读取测试数据
    test_data = pd.read_csv('train_data_set/customer_credit_data_test.csv')
    prediction = predict_credit(test_data)
    test_data["预测结果"] = prediction
    print("测试数据：", prediction)
    test_data.to_csv("pred_data_set/customer_credit_data_test_result.csv",
                     index=False)
```

知识点 5 决策树算法

想象你要决定周末是否去打球，你会考虑：

天气晴吗？→ 是→去球场

不是晴天→有没有带伞？→有→去室内场

没带伞→在家看比赛

这就是决策树！计算机会用数学方法（信息增益）自动找到最佳判断顺序，就像帮你做选择题的智能助手。

核心思想

决策树就是用计算机模拟这个过程：

核心逻辑：从“根节点”开始（如“今天要不要出门”），根据数据特征（如“天气”“温度”“是否堵车”）**逐层提问**，每个问题对应一个“分支”，**最终在“叶子节点”得出结论（如“出门”或“不出门”）**。

如何选择 “提问顺序”？

优先问“最能减少不确定性”的问题。例如：“天气好不好”比“今天星期几”更能决定是否出门，因为天气对出门的影响更大。

计算机用“数学工具”量化这种影响，**比如“信息熵”（衡量数据的混乱程度）和“信息增益”（提问后混乱程度减少了多少）**。

决策树参数设置说明

1. criterion（分裂标准）

它决定了决策树如何选择"最佳分裂点"【数据具体哪一列当树的节点】。就像你考试时，老师要决定是按"分数≥60"还是"完成作业次数≥5"来划分学生是否及格，criterion 就是帮树找到最好的划分标准。

两种常见选项：

"gini"（基尼系数）：计算"分错类"的概率，越小越好（更纯），**基尼系数越小越好**

"entropy"（信息增益）：计算分裂前后的信息混乱程度，减少得越多越好**【头脑风暴】**

如何选择：

大多数情况下两者效果相似，默认用"gini"（计算更快）

如果数据有大量类别不平衡（比如 99%是"是"，1%是"否"），可以试试 entropy

---

2. max_depth（树的最大深度）

控制树的"层数"。就像你问问题，最多只能问 5 个问题就必须做出决定（比如：天气？→湿度？→风速？→气压？→温度？→停止）。

如何设置：

小数据集（<1000 条）：3-5 层足够（防止过拟合）

大数据集：可以尝试 10-15 层，但要用交叉验证检查

关键技巧：

从 max_depth=3 开始画树，观察是否需要更深

---

3. min_samples_split（节点最小分裂样本数）

一个节点至少要有多少个样本，才允许继续分裂。比如设为 10，意味着如果某个节点只有 9 个样本，就不再问问题，直接给出答案。

如何设置：

默认值=2（每个叶子最少 2 个样本才分裂）

防止过拟合：如果数据噪声多，设为 10-50

大数据集：可以按比例设，比如 min_samples_split=0.01（样本的 1%）

---

4. min_samples_leaf（叶子节点最小样本数）

每个"最终答案"至少需要多少个样本支持。比如设为 5，意味着所有分类结果（带伞/不带伞）必须有≥5 个样本支持，避免出现"只有 1 个人就下结论"的情况。

如何设置：

分类问题：常用 1-5（小数据集用 3-5 防过拟合）

回归问题：设为 5-20（需要更稳定的平均值）

类别不平衡时：增大此值（比如 10-20）让少数类更可靠

不同数据集的设置策略

情况 1：小型干净数据集（比如 500 条天气数据）

```
DecisionTreeClassifier(
    criterion="gini",  # 默认
    max_depth=3,       # 浅树防过拟合
    min_samples_split=10,  # 节点至少 10 样本才分裂
    min_samples_leaf=3     # 每个结论至少 3 个样本)
```

情况 2：大型噪声数据集（比如 10 万条电商数据）

```
DecisionTreeClassifier(
    criterion="entropy",  # 噪声多时信息增益更稳定
    max_depth=10,         # 数据量大可以深一些
    min_samples_split=100,  # 避免无意义分裂
    min_samples_leaf=50     # 确保结论可靠性)
```

情况 3：类别极度不平衡（比如欺诈检测，99%正常/1%欺诈）

```
DecisionTreeClassifier(
    criterion="entropy",    # 处理不平衡更好
    max_depth=5,            # 限制深度
    min_samples_leaf=100,   # 确保欺诈样本足够
    class_weight="balanced" # 更重要！告诉树关注少数类)
```

实战案例：Iris 数据集决策树分类

任务目标

输入 Iris 的花萼长度 (sepal length)、花萼宽度 (sepal width)、花瓣长度 (petal length)和花瓣宽度 (petal width)几个值，实现对鸢尾花进行多分类。

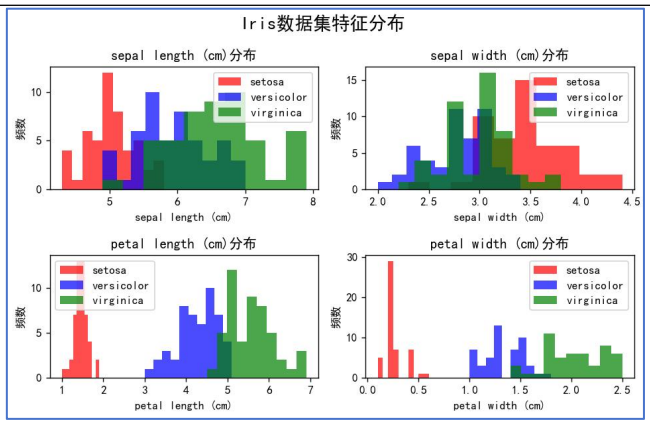
数据集

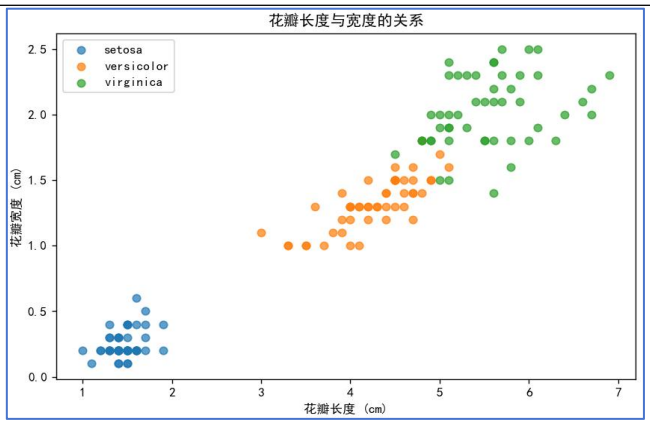
Iris 数据集是机器学习中最经典的数据集之一，包含 150 个样本，每个样本有 4 个特征：花萼长度 (sepal length)、花萼宽度 (sepal width)、花瓣长度 (petal length)、花瓣宽度 (petal width)。这些样本分为 3 类鸢尾花：Setosa (山鸢尾)、Versicolor (变色鸢尾)和 Virginica (维吉尼亚鸢尾)。

Iris 数据探索

```
# 导入必要的库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
# 设置中文字体和图形样式
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
# 1. 加载数据
print("\n1. 加载数据...")
iris = load_iris()
X = iris.data  # 获取数据
y = iris.target # 获取数据的标签
feature_names = iris.feature_names # 获取数据特征的名称
target_names = iris.target_names #获取数据特征的名称

# 创建 DataFrame 以便更好地查看数据
df = pd.DataFrame(X, columns=feature_names)
df['species'] = y
df['species_name'] = [target_names[i] for i in y]

print(f"数据集形状: {X.shape}")
print(f"特征名称: {feature_names}")
print(f"目标类别: {target_names}")
print("\n 前 5 行数据:")
print(df.head())

# 2. 数据探索
print("\n2. 数据探索...")
print("\n 数据集基本信息:")
print(df.describe())

print("\n 各类别样本数量:")
print(df['species_name'].value_counts())

# 可视化数据分布
fig, axes = plt.subplots(2, 2, figsize=(8, 5))
fig.suptitle('Iris 数据集特征分布', fontsize=16)
# 特征定义 花萼长度 花萼宽度  花瓣长度  花瓣宽度
features = ['sepal length (cm)', 'sepal width (cm)',
            'petal length (cm)', 'petal width (cm)']
colors = ['red', 'blue', 'green']

for i, feature in enumerate(features):
    row, col = i // 2, i % 2
    for species in range(3):
        species_data = df[df['species'] == species][feature]
        axes[row, col].hist(species_data, alpha=0.7,
                            label=target_names[species],
                            color=colors[species])
    axes[row, col].set_title(f'{feature}分布')
    axes[row, col].set_xlabel(feature)
    axes[row, col].set_ylabel('频数')
```

```
    axes[row, col].legend()

plt.tight_layout()
plt.show()

# 特征关系散点图
plt.figure(figsize=(8, 5))
for i, species in enumerate(target_names):
    plt.scatter(df[df['species_name'] == species]['petal length (cm)'],
                df[df['species_name'] == species]['petal width (cm)'],
                label=species, alpha=0.7)
plt.xlabel('花瓣长度 (cm)')
plt.ylabel('花瓣宽度 (cm)')
plt.title('花瓣长度与宽度的关系')
plt.legend()
plt.show()
```





Iris 数据探索的结论

分类边界清晰度，花瓣特征提供了几乎完美的线性可分性

- Setosa 与其他两类完全分离，没有重叠区域
- Versicolor 和 Virginica 之间有少量重叠，但大部分样本可清晰区分
- 仅凭花瓣长度和宽度两个特征就能达到很高的分类准确率

基于决策树算法的 Iris 数据的多分类

```
# 导入必要的库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
# 设置中文字体和图形样式
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
# 1. 加载数据
print("\n1. 加载数据...")
iris = load_iris()
X = iris.data  # 获取数据
```

```
y = iris.target # 获取数据的标签
feature_names = iris.feature_names # 获取数据特征的名称
target_names = iris.target_names #获取数据特征的名称
# 2. 数据预处理
print("\n2. 数据集划分与标准化处理...")
# 分割数据集
(X_train, X_test,
 y_train, y_test) = train_test_split(X, y,
                                     test_size=0.2,
                                     random_state=42)
# 标准化特征（决策树通常不需要，但为了演示标准化过程）
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"训练集大小: {X_train.shape}")
print(f"测试集大小: {X_test.shape}")

# 3. 训练决策树模型
print("\n3. 训练决策树模型...")
# 创建决策树分类器
dt_classifier = DecisionTreeClassifier(random_state=42)

# 定义参数网格
param_grid = {
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# 使用网格搜索寻找最佳参数
print("正在进行参数调优...")
grid_search = GridSearchCV(
    dt_classifier,
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1)
grid_search.fit(X_train, y_train)

# 获取最佳模型
best_dt = grid_search.best_estimator_

print(f"最佳参数: {grid_search.best_params_}")
print(f"最佳交叉验证分数: {grid_search.best_score_:.4f}")

# 4. 模型评估
print("\n4. 模型评估...")
```

```python
# 在训练集和测试集上进行预测
y_train_pred = best_dt.predict(X_train)
y_test_pred = best_dt.predict(X_test)

# 计算准确率
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"训练集准确率: {train_accuracy:.4f}")
print(f"测试集准确率: {test_accuracy:.4f}")

# 分类报告
print("\n分类报告:")
print(classification_report(y_test, y_test_pred, target_names=target_names))

# 5. 可视化决策树
print("\n5. 可视化决策树...")
plt.figure(figsize=(10, 8))
plot_tree(best_dt,
          feature_names=feature_names,
          class_names=target_names,
          filled=True,
          rounded=True,
          fontsize=12)
plt.title('决策树结构')
plt.show()

# 打印文本形式的决策树规则
print("\n决策树规则:")
tree_rules = export_text(best_dt, feature_names=feature_names)
print(tree_rules)

# 6. 特征重要性分析
print("\n6. 特征重要性分析...")
feature_importance = best_dt.feature_importances_
feature_importance_df = pd.DataFrame({
    '特征': feature_names,
    '重要性': feature_importance
}).sort_values('重要性', ascending=False)

print("特征重要性排序:")
print(feature_importance_df)

# 7 使用模型进行预测
print("\n7. 使用模型进行预测...")
# 创建一些示例数据
# 特征定义 花萼长度sepal length 花萼宽度sepal width 花瓣长度petal length 花瓣宽度petal width
example_flowers = [
```

```
    [5.1, 3.5, 1.4, 0.2],  # 应该预测为setosa
    [6.0, 2.7, 5.1, 1.6],  # 应该预测为versicolor
    [7.2, 3.0, 5.8, 2.1]  # 应该预测为virginica
]

for i, flower in enumerate(example_flowers):
    prediction = best_dt.predict([flower])[0]
    probability = best_dt.predict_proba([flower])[0]

    print(f"\n示例花 {i + 1}: {flower}")
    print(f"预测类别: {target_names[prediction]}")
    print("类别概率:")
    for j, prob in enumerate(probability):
        print(f"  {target_names[j]}: {prob:.4f}")

# 8. 模型复杂度与性能关系
print("\n9. 分析不同树深度对性能的影响...")
max_depths = range(1, 11)
train_scores = []
test_scores = []

for depth in max_depths:
    dt_temp = DecisionTreeClassifier(max_depth=depth, random_state=42)
    dt_temp.fit(X_train, y_train)
    train_scores.append(accuracy_score(y_train, dt_temp.predict(X_train)))
    test_scores.append(accuracy_score(y_test, dt_temp.predict(X_test)))

plt.figure(figsize=(8, 6))
plt.plot(max_depths, train_scores, 'o-', label='训练集准确率')
plt.plot(max_depths, test_scores, 'o-', label='测试集准确率')
plt.xlabel('决策树深度')
plt.ylabel('准确率')
plt.title('决策树深度与性能关系')
plt.legend()
plt.grid(True)
plt.show()

print("\n" + "=" * 60)
print("模型训练和评估完成!")
print(f"最终测试集准确率: {test_accuracy:.4f}")
print("=" * 60)
```

知识点 6 朴素贝叶斯算法

核心思想

像预言家一样，通过历史事件的概率预测新事件，可将它当做一个“经验判断小能手”。比如通过过去 100 天中"阴天→下雨"的概率，预测明天是否带伞。

朴素贝叶斯是一种基于贝叶斯定理的简单概率分类器，它假设特征之间相互独立（即

"朴素"的假设）。

贝叶斯定理

贝叶斯定理描述了在**已知先验知识的情况下**，如何根据新的证据更新事件的概率：

$$P(A|B) = P(B|A) * P(A) / P(B)$$

其中：

- P(A|B)：在 B 发生的情况下 A 发生的概率（后验概率）
- P(B|A)：在 A 发生的情况下 B 发生的概率（似然）
- P(A)：A 发生的概率（先验概率）
- P(B)：B 发生的概率（证据）

在分类中的应用

在分类问题中，我们想找到最可能的类别 C，给定特征 $X_1, X_2, ..., X_n$：

$$P(C|X_1, X_2, ..., X_n) \propto P(C) * P(X_1|C) * P(X_2|C) * ... * P(X_n|C)$$

我们选择使这个概率最大的类别作为预测结果。

示例：垃圾邮件分类

假设我们有一个包含"免费"和"赢取"两个词的邮件，想判断它是否是垃圾邮件：

- P(垃圾邮件) = 0.3（先验概率）
- P("免费"|垃圾邮件) = 0.6
- P("赢取"|垃圾邮件) = 0.4
- P("免费"|非垃圾邮件) = 0.1
- P("赢取"|非垃圾邮件) = 0.05

对于包含这两个词的邮件：

- P(垃圾邮件|"免费","赢取") $\propto$ 0.3 × 0.6 × 0.4 = 0.072
- P(非垃圾邮件|"免费","赢取") $\propto$ 0.7 × 0.1 × 0.05 = 0.0035

由于 0.072 > 0.0035，我们将其分类为垃圾邮件。

朴素贝叶斯分类器在以下场景中表现优异：

- 文本分类：垃圾邮件检测、情感分析、新闻分类
- 多类别分类：文档分类、产品分类
- 实时预测：需要快速响应的应用
- 高维数据：特征数量多的数据集
- 推荐系统：基于内容的推荐

优点	局限性
简单、快速、高效 对小规模数据表现良好 能够处理多分类问题 对缺失数据不敏感 适合增量学习（可以逐步更新模型）	特征独立性假设在现实中很少成立 对输入数据的分布有假设（如高斯朴素贝叶斯假设数据服从正态分布） 概率估计可能不准确（零频率问题） 性能可能不如更复杂的模型

朴素贝叶斯变体

变体	适用数据类型	特点	典型应用
高斯朴素贝叶斯	连续数据	假设特征服从高斯分布	数值型分类问题
多项式朴素贝叶斯	离散计数数据	使用特征的频率计数	文本分类、文档分类
伯努利朴素贝叶斯	二元/布尔数据	特征值为 0 或 1	文档分类（词出现与否）

实战案例：基于朴素贝叶斯算法的诈骗短信识别

任务目标

该任务的核心是**利用机器学习技术从海量短信数据中学习诈骗短信的语言特征模式**，构建一个高效、准确的自动化识别系统，**能够准确区分诈骗与正常短信的 AI 模型**。

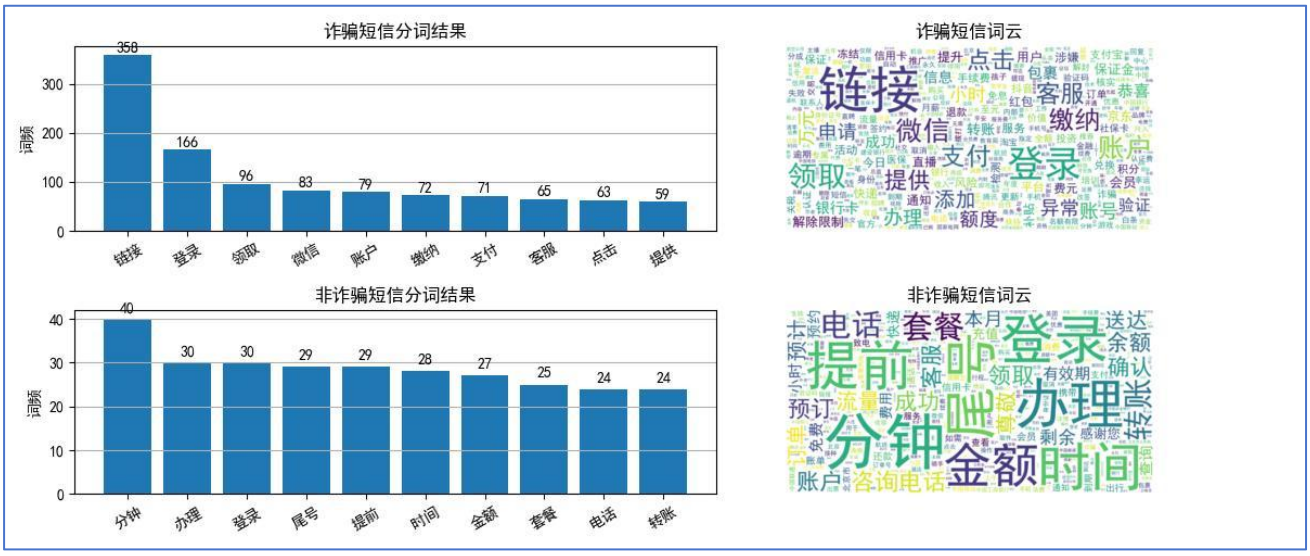
数据集

id，content 和 target 三个字段。其中 content 为短信的内容信息，target 有两个值即“诈骗”和“非诈骗”。

数据探索关键代码

```
# 导入必要的库
import re
from collections import Counte
import jieba
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from wordcloud import WordCloud
warnings.filterwarnings('ignore')
# 设置中文字体和图形样式
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
#加载用户字典
jieba.load_userdict("data_set/user_dict.txt")
#加载停用词表
with open("data_set/stop_words.txt","r",encoding="utf-8") as fr:
    stop_words = fr.read().split("\n")
scam_words =[]  # 保存诈骗关键词
non_scam_words =[]  # 保存非诈骗关键词
def words_extract(text,scam=True):
    # 使用正则表达式剔除文本中的非中文字符 [^\u4e00-\u9fa5] 固定代码格式
    text = re.sub(r"[^\u4e00-\u9fa5]", "", text)
    # 使用 jieba 进行分词并过滤停用词
    fliter_words = [wd for wd in jieba.lcut(text)
                    if wd not in stop_words and len(wd) > 1]
    #  保存分词结果
    if scam:
        scam_words.extend(fliter_words)
    else:
        non_scam_words.extend(fliter_words)
```

```
# end scam_words_extract(text)

if __name__ == '__main__':
    # 读取数据
    df = pd.read_csv('data_set/Scam_text_message.csv')
    # 处理诈骗的数据
    df[df["target"] == "诈骗"]["content"].apply(lambda text: words_extract(text))
    print(scam_words)
    scam_words_count = Counter(scam_words)
    print(scam_words_count)
    # 提取 10 个高频涉及到诈骗的词
    scam_top_10_words = scam_words_count.most_common(10)
    # 将结果转换成字典
    scam_top_10_words = dict(scam_top_10_words)

    # 非诈骗数据信息的处理
    df[df["target"] == "非诈骗"]["content"].apply(lambda text:
words_extract(text,scam=False))
    print(non_scam_words)
    non_scam_words_count = Counter(non_scam_words)
    print(non_scam_words_count)
    # 提取 10 个高频涉及到非诈骗的词
    non_scam_top_10_words = non_scam_words_count.most_common(10)
    # 将结果转换成字典
    non_scam_top_10_words = dict(non_scam_top_10_words)

    # 调用可视化的函数进行分析结果的展示

plot_result(scam_top_10_words,scam_words_count,non_scam_top_10_words,non_scam_words
_count)
```



模型训练关键代码

```
import re
import jieba
```

```
import joblib
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
#加载停用词表
with open("data_set/stop_words.txt", "r", encoding="utf-8") as fr:
    stop_words = fr.read().split("\n")
    stop_words.append("\n")
# 用户词典
jieba.load_userdict("data_set/user_dict.txt")
def chinese_text_cut(text):
    text = re.sub("[^\u4e00-\u9fa5]", "", text)
    words = jieba.lcut(text)
    words = [wd for wd in words if wd not in stop_words and len(wd)>1]
    return words
# end chinese_text_cut(text)


def train_nb_model(data, tag):
    print("装载文本特征处理的函数...")
    # 进行向量化 文本的特征处理
    tfidf = TfidfVectorizer(
        tokenizer=chinese_text_cut,
        stop_words=stop_words,
        token_pattern=None
    )
    print("进行训练文本数据的划分...")
    # 划分训练集和测试集
    (X_train,
     X_test,
     y_train,
     y_test) = train_test_split(
        data, tag,
        test_size=0.2,
        random_state=1
    )
    print(f"训练集: {len(X_train)} 条")
    print(f"测试集: {len(X_test)} 条")
    # 构建 Pipeline
    print("构建 Pipeline 流水线管道......")
    pipeline = Pipeline([
        ('tfidf', tfidf), # 先进行文本的处理
        ('nb', MultinomialNB(alpha=0.1)) # 之后调用模型进行分类处理
    ])
    # 训练模型
    print("开始训练模型...")
```

```
    pipeline.fit(X_train, y_train)
    # 评估模型
    print("评估训练的模型...")
    y_pred = pipeline.predict(X_test)
    train_accuracy = pipeline.score(X_train, y_train)
    test_accuracy = pipeline.score(X_test, y_test)
    print(f"训练集准确率: {train_accuracy:.4f}")
    print(f"测试集准确率: {test_accuracy:.4f}")
    print("分类报告\n",
          classification_report(y_test, y_pred)
    )
    # 保存模型和词袋
    print("保存模型......")
    joblib.dump(tfidf, "train_model/tfidf.pkl")
    print(f"TF-IDF 向量化器已保存到: train_model 目录下")
    joblib.dump(pipeline, 'train_model/sms_classifier.pkl')
    print("模型已保存在 train_mode 目录下")
# end train_nb_model


if __name__ == '__main__':
    df = pd.read_csv("data_set/Scam_text_message.csv")
    tag = df["target"]
    data = df["content"]
    train_nb_model(data, tag)
```

分类模型预测

```
import re
import jieba
import joblib
#加载停用词表
with open("data_set/stop_words.txt", "r", encoding="utf-8") as fr:
    stop_words = fr.read().split("\n")
    stop_words.append("\n")
# 用户词典
jieba.load_userdict("data_set/user_dict.txt")


def chinese_text_cut(text):
    text = re.sub("[^\u4e00-\u9fa5]", "", text)
    words = jieba.lcut(text)
    words = [wd for wd in words if wd not in stop_words and len(wd)>1]
    return words
# end chinese_text_cut(text)

def scam_msg_predict(text):
    try:
        # 加载模型
```

```
        model = joblib.load('train_model/sms_classifier.pkl')
        # 进行预测
        prediction = model.predict([text])[0]
        proba = model.predict_proba([text])[0]
        # 返回结果
        result = {
            'text': text,
            'prediction': prediction,
            'probability': {
                '诈骗概率': round(proba[0] * 100, 2),
                '非诈骗概率': round(proba[1] * 100, 2)
            }
        }
        return result
    except Exception as e:
        return {'error': str(e)}


if __name__ == '__main__':
    test_messages = [
        "航旅纵横：您预订的 9C8888 航班值机已开放，可手机选座",
        "【共享屏幕】您的银行 APP 操作被录屏！转账 3 万元删除记录：http://screen-hack.net",
        "恭喜您获得大奖，点击链接领取",
        "您的快递已到达小区快递柜"
    ]

    for msg in test_messages:
        result = scam_msg_predict(msg)
        print(f"文本: {result['text']}")
        print(f"预测结果: {result['prediction']}")
        print(f"概率: 诈骗 {result['probability']['诈骗概率']}%, "
              f"非诈骗 {result['probability']['非诈骗概率']}%")
        print("-" * 50)
```

三、情感数据分析与标注

知识点 1 情感分析原理

情感分析（Sentiment Analysis）是自然语言处理（NLP）的一个重要分支，旨在识别和提取文本中的主观信息，判断作者的情感倾向。

知识点 2 情感分析的应用场景

- 产品评论分析
- 社交媒体监控
- 客户反馈处理
- 市场趋势分析
- 舆情监测

知识点 3 SnowNLP 库介绍

SnowNLP的主要功能

功能	描述
中文分词	将中文文本分割成有意义的词汇单元
词性标注	识别文本中每个词的词性（名词、动词等）
情感分析	分析文本的情感倾向，返回0-1之间的情感值
文本分类	对文本进行分类（需要训练自定义分类器）
关键词提取	从文本中提取关键词
文本摘要	生成文本的摘要

提示：SnowNLP 的情感分析模型是基于商品评论数据训练的，因此在分析商品评论时效果最佳。

知识点 4 SnowNLP 基本表用法

```
from snownlp import SnowNLP
text1 = "这个电影太精彩了，演员表演出色，剧情扣人心弦！"
s1 = SnowNLP(text1) # 创建SnowNLP对象
print(f"文本：{text1}")
print(f"情感分数：{s1.sentiments}") # 接近1表示积极
text2 = "产品质量很差，使用不到一周就坏了，非常失望。"
s2 = SnowNLP(text2)
print(f"文本：{text2}")
print(f"情感分数：{s2.sentiments}") # 接近0表示消极
```

情感分数解释

- **0.0-0.3：** 消极情感
- **0.3-0.7：** 中性情感
- **0.7-1.0：** 积极情感

> **示例：** 情感分数为0.85表示文本表达的是强烈的积极情感。

实战案例：基于 SnowNLP 的自动情感标注

任务目标

构建一个基于 SnowNLP 的自动化情感分析系统，对网购评论数据进行情感标注和分类，实现从原始文本到结构化情感标签的完整处理流程。

实现代码

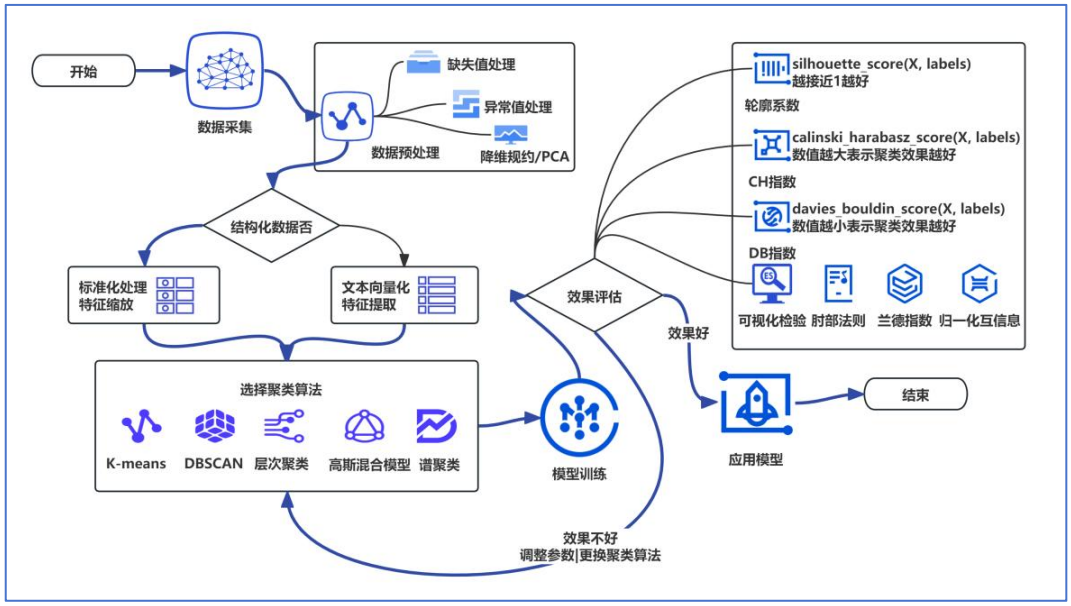
```
import time
import pandas as pd
from snownlp import SnowNLP
import re
```

```
def analyze_sentiment(text):
    clean_text = re.sub("[^\u4e00-\u9fa5]", "", text)
    s = SnowNLP(clean_text)
    _score = s.sentiments
    if _score <= 0.3:
        return '消极情感'
    elif _score <= 0.7:
        return '中性情感'
    else:
        return '积极情感'
# end analyze_sentiment

if __name__ == '__main__':
    print("加载数据")
    df = pd.read_csv("data_set/comment_data.csv")
    print("情感数据标注中.....")
    df.loc[:,"sentiment_category"] = df["comment"].apply(lambda
text:analyze_sentiment(text))
    time.sleep(2)
    print("数据文件标注完成,正在保存中......")
    df.to_csv("data_set/comment_data_tag.csv"
              ,encoding="utf-8"
              ,header=True
              ,index=False)
```

四、聚类分析

聚类分析是一种无监督学习方法，旨在将数据集中的对象分组，使得同一组（即簇）内的对象相似度较高，而不同组之间的对象相似度较低。与监督学习不同的是，非监督学习技术不使用带标签的数据，算法需要自己去发现数据中的结构。常见的聚类算法包括 K-means、层次聚类、DBSCAN 等。

知识点 1 聚类算法的处理流程

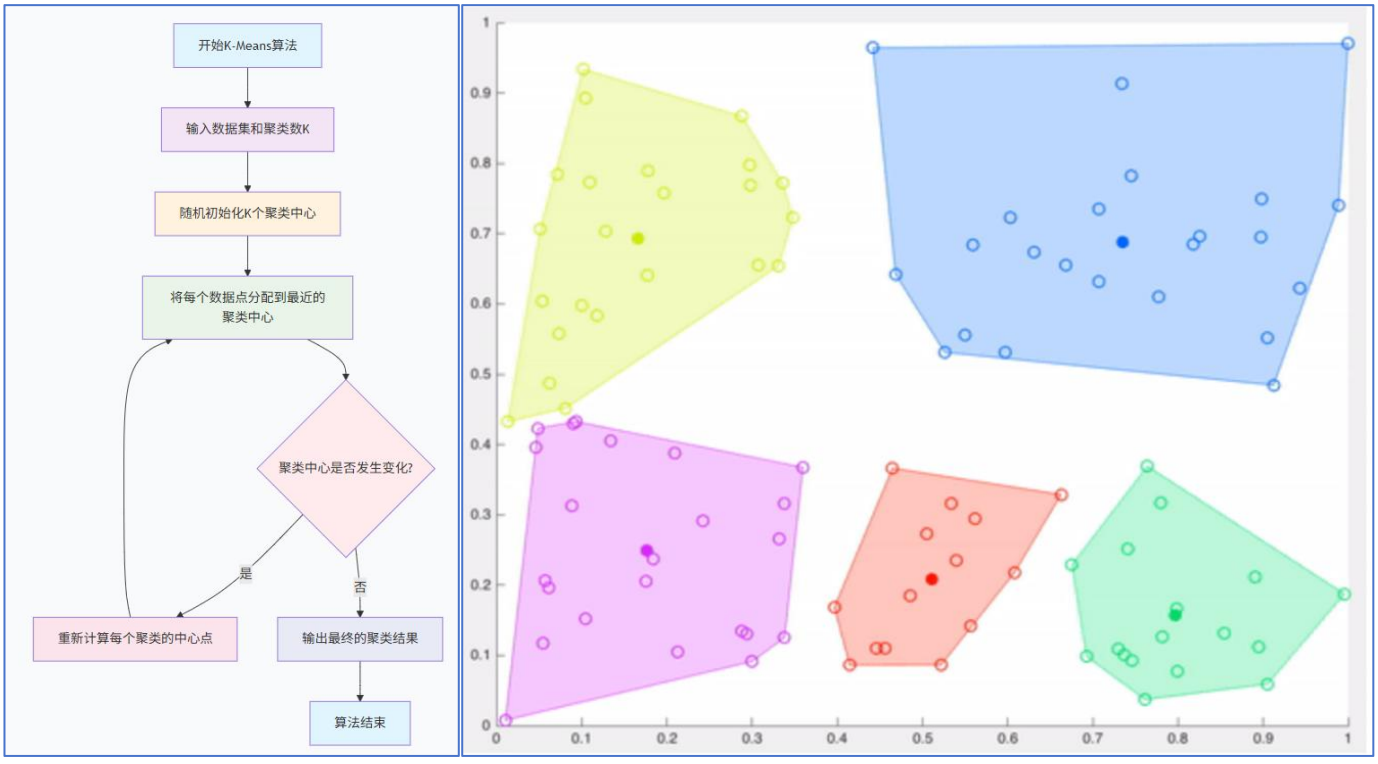


知识点 2 聚类算法的评测指标

指标名称	指标含义	使用方法	参数含义	适用算法
轮廓系数 (Silhouette)	衡量样本与自身簇的紧密度和与其他簇的分离度，取值[-1,1]，越接近 1 越好	silhouette_score(X,labels)	X:原始数据数 labels:聚类结果标签数组	通用
CH 指数	簇间离散度与簇内离散度的比值，值越大越好	calinski_harabasz_score(X,labels)		
DB 指数	衡量各簇之间的平均相似度，值越小越好	davies_bouldin_score(X,labels)		
同质性评分 (Homogeneity)	需要真实标签，衡量同类样本的聚集程度，取值[0,1]，1 表示完全一致	homogeneity_score(true_labels,pred_labels)	true_labels：数据真实标签 pred_labels：数据预测标签	有标签时通用
调整兰德指数 (ARI)	需要真实标签，衡量聚类结果与真实标签的相似度，取值[-1,1]，1 表示完全一致	adjusted_rand_score(true_labels,pred_labels)		
归一化互信息 (NMI)	衡量聚类结果与真实标签的信息共享程度（需要真实标签），取值[0,1]，1 表示完全一致	normalized_mutual_info_score(true_labels, labels)		

知识点 3 划分聚类

K-means 算法是一种常用的聚类算法，其基本思想是使用迭代细化方法，基于用户定义的集群数量（由变量 K 表示）和数据集来产生其最终聚类。例如，如果将 K 设置为 3，则数据集将分组为 3 个群集，如果将 K 设置为 4，则将数据分组为 4 个群集，依此类推。



知识点 4 密度聚类

核心思想

故事讲解：DBSCAN 的冒险之旅：小精灵的寻友记

密度王国的奇妙冒险，在数据王国里，住着一群活泼的小精灵（数据点）。他们有的喜欢热闹，成群结队（核心点）；有的性格孤僻，独自生活（噪声点）；还有的虽然朋友不多，但总爱跟着热闹的群体（边界点）。

一天，王国里来了一位名叫 DBSCAN 的探险家，他决定帮小精灵们找到自己的朋友圈（聚类）。DBSCAN 有两个神奇的魔法道具：

魔法望远镜（ε-邻域）：能看到每个小精灵周围一定距离内的其他精灵

最小朋友数（minPts）：判断一个精灵是否喜欢热闹的标准

DBSCAN 的探险规则很简单：

- 随机选择一个还没被访问的小精灵，用魔法望远镜看看他周围有多少朋友
- 如果朋友数≥minPts，说明这里有个新群体！标记为核心点，并开始"滚雪球"式地寻找所有可达的朋友
- 如果朋友数不够但位于某个核心点的望远镜范围内，就标记为边界点
- 如果既不是核心点也不在任何核心点的范围内，就是孤独的噪声点
- 就这样，DBSCAN 走遍了整个王国，把所有小精灵都分到了合适的朋友圈，或者标记为喜欢独处的个体。

DBSCAN 适用场景：

- **处理任意形状的簇。**适用场景：当数据中的簇呈现非球形、不规则形状（如长条形、环形、嵌套结构等）时，DBSCAN 能准确识别，而 K-means 等基于距离的算法会失效。典型应用：
  - 地理数据（如地震震中分布、城市热点区域）
  - 生物信息学（基因表达模式、蛋白质结构分析）
  - 图像分割（识别不规则物体边界）
- **数据包含噪声或异常值。**适用场景：数据中存在大量噪声点或异常值，需要自动过滤。典型应用：
  - 欺诈检测（标记异常交易）
  - 传感器数据分析（剔除故障传感器读数）
  - 客户分群（排除低价值或异常用户）
- **无需预先指定簇数量。**数据中的簇数量未知，且希望算法自动发现，典型应用：
  - 社交网络分析（发现潜在社区）
  - 天文数据分析（识别未知星团）
  - 市场调研（探索用户自然分群）
- **空间或时空数据分析。**处理具有空间或时空属性的数据，关注局部密度特征，典型应用。
  - 城市规划（识别人口密度相近的居民区）

- 气象学（分析相似密度的气象模式）
- 物流优化（仓库货物分布聚类）
- 交通管理（识别拥堵路段）
- 环境监测（污染源扩散分析）
- 流行病学（疾病爆发区域追踪）

何时不适用 DBSCAN？

- 高维数据：维度灾难导致距离计算失效。
- 密度差异过大：同一数据集中同时存在极密集和极稀疏的簇。
- 计算资源受限：大规模数据时时间复杂度较高。

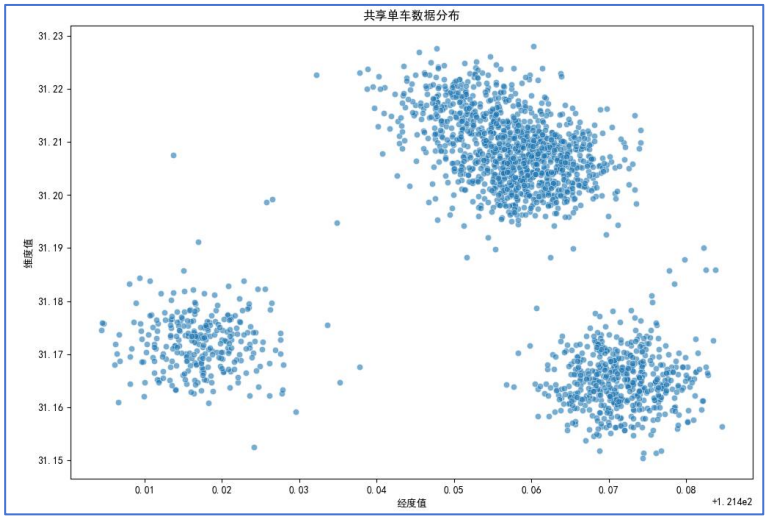
参数设置指南

场景类型	推荐算法	关键参数建议	注意事项
用户分群（已知分组）	K-means	K=业务预期分组数	需标准化分类与数值特征
异常检测	DBSCAN	eps=0.3-0.5,min_samples=5	优先使用标准化后的数据
地理热点分析	DBSCAN	eps=实际距离单位,min_samples=10	转换坐标系（如 UTM）
图像压缩	K-means	K=16/256（颜色数）	使用 Lab 颜色空间提升感知均匀性
时序事件聚类	DBSCAN	eps=时间窗口（如 1 小时）,min_samples=3	结合时间空间双维度

实战案例：共享单车数据聚类分析

任务目标

对上海的共享单车数据进行聚类分析，已知共享单车数据中有经度和纬度两个属性，分别使用 KMeans 和 DBSCAN 聚类算法进行聚类分析。最后可视化聚类的结果。原始数据分布如下所示。



DBSCAN 聚类实现代码

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

```
from sklearn.preprocessing import StandardScaler
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['Simhei']
# 解决负号显示为方块的问题
plt.rcParams['axes.unicode_minus'] = False
# 示例：使用共享单车数据

def my_dbscan(data, eps_value, k):
    # 使用 k-距离图确定的参数
    optimal_eps = eps_value  # 从前面计算得到
    min_samples = k + 1  # k 是最近邻数量
    # 应用 DBSCAN
    db = DBSCAN(eps=optimal_eps, min_samples=min_samples )
    db.fit(data)
    # 获取聚类结果
    labels = db.labels_
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise = list(labels).count(-1)
    print(f"发现聚类数: {n_clusters}")
    print(f"噪声点数: {n_noise}")
    # 可视化结果
    plt.figure(figsize=(12, 8))
    plt.scatter(data[:, 1],
                data[:, 0],
                c=labels,
                cmap='viridis',
                edgecolor='k',
                alpha=0.5)
    plt.title(f'DBSCAN 聚类结果 (eps={optimal_eps:.3f}km, min_samples={min_samples})')
    plt.xlabel('经度')
    plt.ylabel('纬度')
    plt.colorbar(label='聚类标签')
    plt.savefig("密度聚类分析的共享单车数据.png")
    plt.show()

if __name__ == '__main__':
    bike_df = pd.read_csv('../shared_bike_data.csv')
    data = bike_df[["纬度", "经度"]].values
    # 进行数据的标准化处理
    data = StandardScaler().fit_transform(data)
    my_dbscan(data,eps_value=0.21568,k=5)
```

K-Means 聚类实现代码

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

```
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['Simhei']
# 解决负号显示为方块的问题
plt.rcParams['axes.unicode_minus'] = False

def get_k_value(data):
    # 使用肘部法则确定最佳 K 值
    # 创建一个字典，用于存储每个 k 值对应的轮廓系数得分
    k_sl_score = {}
    # 遍历每个 k 值
    for k in range(2, 10):
        # 创建 KMeans 模型
        kmeans = KMeans(n_clusters=k, random_state=42)
        # 训练模型
        kmeans.fit(data)
        # 计算轮廓系数
        sl_score = silhouette_score(data, kmeans.labels_)
        # 将轮廓系数和惯性值添加到列表中
        k_sl_score[k] = sl_score
    # end for
    # 找到字典 k_sl_score 中值最大对应的 key 值
    best_k = max(k_sl_score, key=k_sl_score.get)
    print('最佳 K 值:', best_k)
    return best_k,k_sl_score

def my_KMeans(k, data):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data)
    return kmeans.labels_, kmeans.cluster_centers_

def plot_cluster_result(data,centers,pred_target,best_k,k_sl_score):
    ''' 可视化聚类结果'''
    plt.figure(figsize=(12, 5))
    # 绘制 1 行 3 列中的第 1 个子图
    # ======== 绘制原始数据分布 ========
    plt.subplot(1, 3, 1)
    plt.scatter(data[:, 0],data[:, 1],
                edgecolor='k',s=30)
    plt.xlabel('经度(标准化)')
    plt.ylabel('经度(标准化)')
    plt.title('原始数据分布图')
    plt.subplot(1, 3, 2)
    # ======== 绘制最佳 K 值的确定 ========
    plt.plot(  # 绘制轮廓系数得分曲线
        k_sl_score.keys(),
        k_sl_score.values(),
        'bx-')
    plt.scatter(best_k,
                k_sl_score[best_k],
```

```
                color='red', s=40,
                )
    plt.xlabel('聚类促中心点个数')
    plt.ylabel('轮廓系数得分')
    plt.title('使用肘部法则确定最佳 K 值')
    plt.grid(True)
    # ======== 绘制最终的聚类结果 ========
    plt.subplot(1, 3, 3)
    for i in range(best_k):
        plt.scatter(data[pred_target == i, 0],
                    data[pred_target == i, 1],
                    edgecolor='k',
                    label=f'中心点 {i}')
    # 绘制中心点
    plt.scatter(centers[:, 0],centers[:, 1],
                s=300,c='red',
                marker='*',label='聚类中心点')
    plt.title(f'K-means 聚类 (K={best_k})')
    plt.xlabel('经度(标准化)')
    plt.ylabel('维度(标准化)')
    plt.legend( )
    plt.tight_layout()
    plt.savefig('K-means 聚类分析共享单车数据.png')
    plt.show()

if __name__ == '__main__':
    bike_df = pd.read_csv('../shared_bike_data.csv')
    data = bike_df[["纬度", "经度"]]

    # 进行数据的标准化处理
    data = StandardScaler().fit_transform(data)
    best_k, k_sl_score = get_k_value(data)
    # 使用 KMeans 模型对数据进行聚类
    pred_target,centers= my_KMeans(best_k, data)
    plot_cluster_result(data, centers, pred_target, best_k, k_sl_score)
```

聚类的结果