一、实验内容

1. **问题描述**：分类算法是解决分类问题的方法，是数据挖掘、机器学习和模式识别中一个重要的研究领域。分类算法通过对已知类别训练集的分析，从中发现分类规则，以此预测新数据的类别。分类算法的应用非常广泛，银行中风险评估、客户类别分类、文本检索和搜索引擎分类、安全领域中的入侵检测以及软件项目中的应用等等。

2. **内容提要**：针对教师指定的两类公用数据集（纯数值型例如UCI Iris，混杂型数据例如UCI Bank Marketing），学生对给定的数据进行分类。本次实验主要内容包括数据处理、算法实现和评价方法。鼓励与其他方法尤其是业界领先算法进行比较分析，鼓励创新设计求解方法。

- 纯数值型数据集，UCI Iris，150个样本，4维。（参考iris.data文件）
  http://archive.ics.uci.edu/ml/datasets/Iris
- 混杂型数据集，UCI Bank Marketing，4119个样本，20维。（参考bank-additional.csv文件）
  http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

二、实验设备

1. 实验设备：台式机/笔记本等不限
2. 平台：Visual C++ / Python等不限

三、实验步骤

sepal_length_cm、sepal_width_cm、petal_length_cm、petal_width_cm、class字段代表的含义分别是花萼长度、花萼宽度、花瓣长度、花瓣宽度、尾鸢花的类别。

1. 读取数据，并做预处理
2. 至少实现一种分类算法，选择评价方法并分析原因
3. 选择适当可视化方法显示结果

4. *扩展选做题：可以尝试多种分类算法并比较结果

四、分析说明（包括结果图表分析说明，主要核心代码及解释）

对于问题一，使用KNN、支持向量机、决策树、随机森林等算法进行分类并比较不同。

1. KNN算法

原理：基于样本的相似性分类，对于给定的测试点，计算它与训练集中每个样本的 距离，选取最近的K个样本，根据这些样本的类别进行分类。

核心步骤：

（1）计算距离：常用欧几里得距离，也可以使用曼哈顿距离等。

（2）参数选择：K的大小对结果影响较大。通过交叉验证选取合适的K值。

（3）特征归一化：因为距离计算对特征的尺度敏感，需要对数据进行归一化或标准化。

实验过程：

（1）数据预处理

A. 加载数据

从 UCI 数据库中读取 Iris 数据集，并命名特征列为 sepal_length、sepal_width、petal_length、petal_width 和分类标签 class。

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)
```

B. 新增特征

通过计算萼片面积 (sepal_area = sepal_length * sepal_width) 和花瓣面积 (petal_area = petal_length * petal_width) 生成新的特征，用于分类。将四维降为二维，维数较少，不采用主成分分析。

```
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']
```

（2）设置分类任务的输入和输出

特征矩阵：X仅包含新增的面积特征（sepal_area 和 petal_area）。

标签：y 为数据集中的class列。

```
# 3. 构建新特征矩阵和标签
X = data[['sepal_area', 'petal_area']].values  # 只保留面积特征
y = data['class'].values  # 分类标签
```

（3）参数调优

A. 搜索范围

k_values：KNN中最近邻的数量，从1到20。
split_ratios：测试集占比，取值范围为 0.2、0.3、0.4、0.5

```
# 定义参数搜索范围
k_values = range(1, 21)  # 测试 k 值从 1 到 20
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比
```

B. 模型训练与验证（交叉验证）

遍历所有split_ratios和k_values组合。在每种组合下，使用 train_test_split 划分数据集，并对特征进行标准化（StandardScaler）。标准化之后使用 KNN 分类器进行模型训练和预测，计算测试集的分类准确率。将不同参数组合的准确率存储在accuracy_matrix中并找到最佳的 k 和 split_ratio，以及对应的最高准确率。

```
# 搜索最优的 k 值和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for k in k_values:
        # 使用KNN分类
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)  # 训练模型
        y_pred = knn.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_k = k
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果
```

（4）热力图可视化参数组合效果

A.将 accuracy_matrix 转为 NumPy 数组。

```
# 转换为 NumPy 数组以便绘制热力图
accuracy_matrix = np.array(accuracy_matrix)
```

B.使用 seaborn.heatmap 绘制热力图，　　展示不同 k 值和测试集占比对分类准确率的影响。

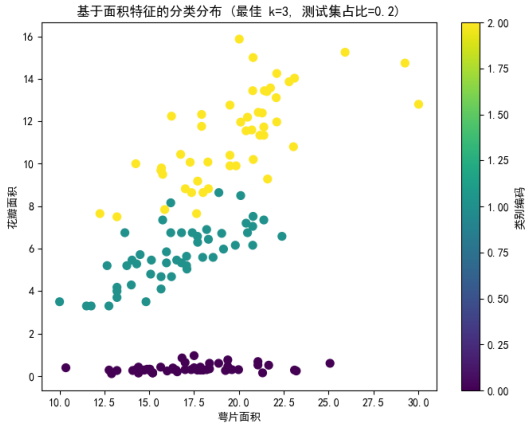
```
# 绘制热力图
plt.figure(figsize=(12, 8))
sns.heatmap(accuracy_matrix, annot=True, cmap="YlGnBu", xticklabels=k_values, yticklabels=split_ratios, cbar=True)
plt.xlabel("k 值")
plt.ylabel("测试集占比")
plt.title("K 值与测试集占比对准确率的影响")
plt.show()
```

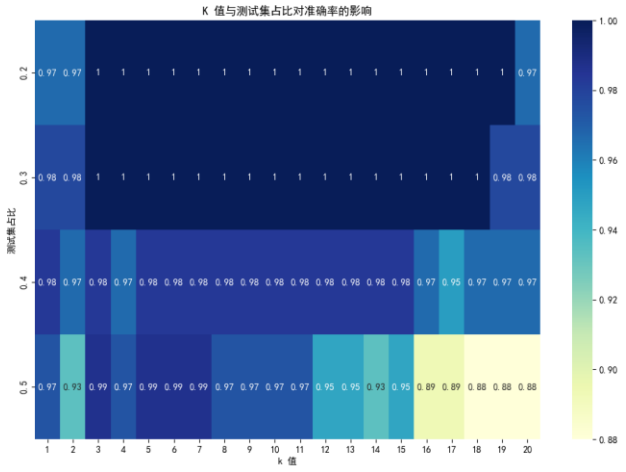
（5）最优结果可视化

使用找到的最佳 k 和测试集占比重新训练 KNN 模型。绘制散点图，展示数据在新增特征空间（萼片面积和花瓣面积）的分类分布。

```
# 可视化最优结果
# 使用最佳参数重新训练模型并可视化
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=best_split, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# 可视化分类分布
plt.figure(figsize=(8, 6))
plt.scatter(data['sepal_area'], data['petal_area'], c=pd.Categorical(data['class']).codes, cmap='viridis', s=50)
plt.colorbar(label='类别编码')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.title(f'基于面积特征的分类分布 (最佳 k={best_k}, 测试集占比={best_split})')
plt.show()
```

（6）运行结果截图分析





分析：

（1）小测试集比例时，无论K值为何，模型往往能达到很高准确率（多数接近1）。

（2）当测试集比例增大后，K值变得重要。一些较小或较大的K值可能导致准确率下降，而中间某些合适的K值可保持较高的准确率。

整体而言，KNN对K值的选择较为敏感，在测试集占比较大时，选对K值才能在相对更大样本的独立测试中保持高准确率。

2. 支持向量机算法

原理：寻找能够最大化分类边界（支持向量）的超平面。

核心步骤：

（1）选择核函数：根据数据特性选择合适的核函数，如线性核适合线性分布数据，RBF 核适合复杂分布。

（2）参数调整：通过网格搜索或交叉验证调整超参数

（3）数据标准化：SVM对特征尺度敏感，需要进行标准化处理。

实验过程：

与KNN算法的实验过程类似，不过在参数调优方面有所不同，故只对这方面进行解释。

（1）数据预处理

（2）设置分类任务的输入和输出

（3）参数调优

A. 定义搜索范围

kernels: SVM 的 4 种核函数类型（linear、poly、rbf、sigmoid）。

split_ratios: 测试集占比（0.2、0.3、0.4、0.5）

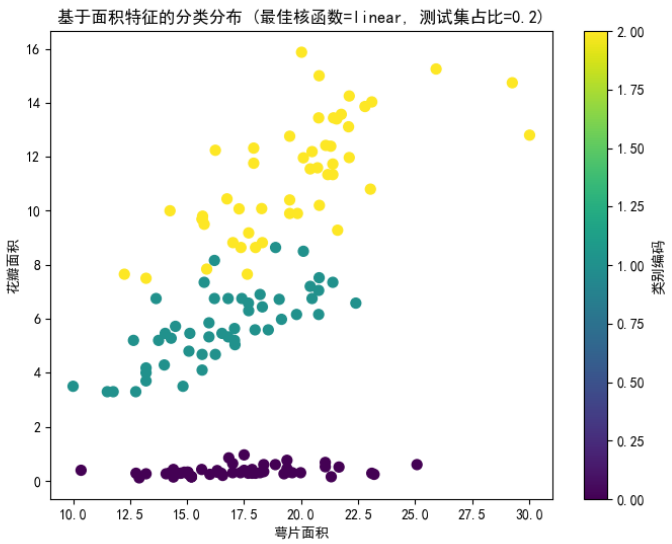
```
# 定义参数搜索范围
kernels = ['linear', 'poly', 'rbf', 'sigmoid']  # SVM 的核函数类型
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比
```

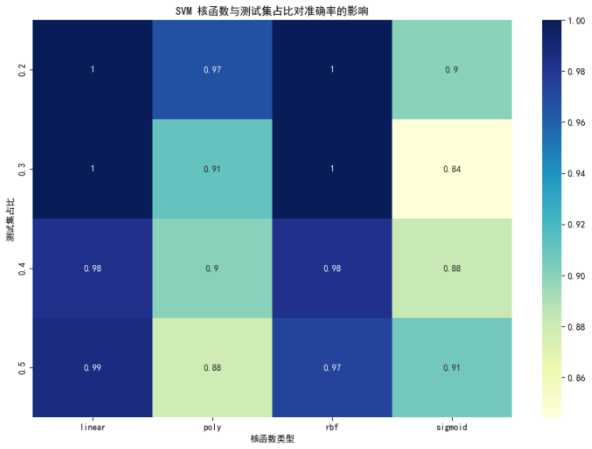
B. 搜索流程

外层循环：遍历不同的测试集占比，划分训练集和测试集，并对特征标准化。
内层循环：遍历不同核函数类型。使用SVM训练模型，预测测试集结果，计算分类准确率并记录。更新当前最高准确率、最佳核函数和测试集占比。
将每种测试集占比对应的准确率存入矩阵accuracy_matrix

```python
# 搜索最优的核函数和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for kernel in kernels:
        # 使用支持向量机分类
        svc = SVC(kernel=kernel, random_state=42)
        svc.fit(X_train, y_train)  # 训练模型
        y_pred = svc.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_kernel = kernel
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果
```

（4）热力图可视化参数组合效果
（5）最优结果可视化
（6）运行结果截图分析





分析：

（1）不同核函数（linear、poly、rbf、sigmoid）在小测试集比例下同样能够获得极高的准确率（接近1）。

（2）随着测试集比例增大，不同核函数的表现出现分化：

linear与rbf核函数表现相对稳定，即使测试集比例加大，准确率仍能保持在较高水平（0.9以上）。

poly与sigmoid核函数在测试集比例较大时准确率显著下降，说明这些核函数在该数据集下的泛化表现不如linear和rbf稳定或强健。

3. 决策树算法

原理：

基于树状结构进行分类或回归：通过特征划分点（如信息增益、基尼系数）将数据递归分裂成子集，最终在叶节点形成决策。

核心步骤：

（1）特征选择：根据分裂标准（如信息增益或基尼指数）选择最优特征。

（2）停止条件：可以设置最大深度、最小样本数或纯度阈值。

（3）剪枝处理：为了防止过拟合，可以对决策树进行剪枝（如预剪枝或后剪枝）。

实验过程：

与KNN算法的实验过程类似，不过在参数调优方面有所不同，故只对这方面进行解 释。

（1）数据预处理

（2）设置分类任务的输入和输出

（3）参数调优

A. 定义搜索范围

max_depth_values: 决策树的最大深度，取值范围为1到 20。

split_ratios: 测试集占比，取值为 0.2、0.3、0.4、0.5。

```
# 定义参数搜索范围
max_depth_values = range(1, 21)  # 测试决策树最大深度从 1 到 20
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比
```

B. 搜索过程

外层循环：遍历测试集占比。使用 train_test_split 将数据集划分为训练集和测试集。对特征进行标准化。

内层循环：遍历决策树的最大深度。使用DecisionTreeClassifier训练模型，并在测试集上进行预测。计算分类准确率，并记录每种参数组合的结果。

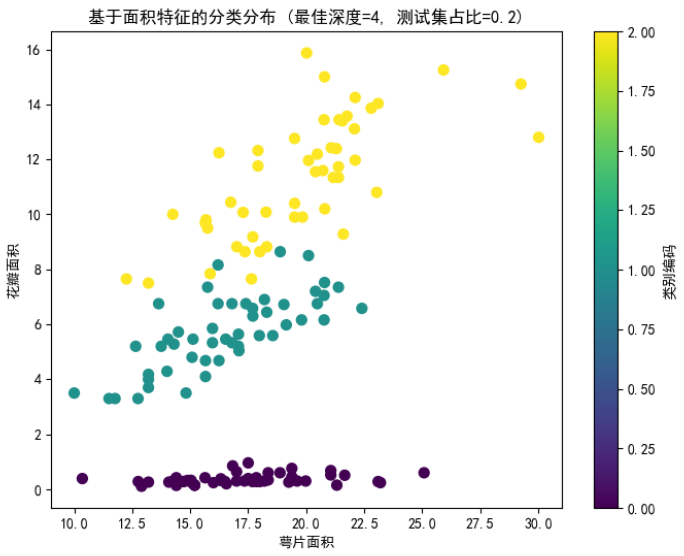
将每种测试集占比的结果存储到 accuracy_matrix，以便后续绘制热力图。

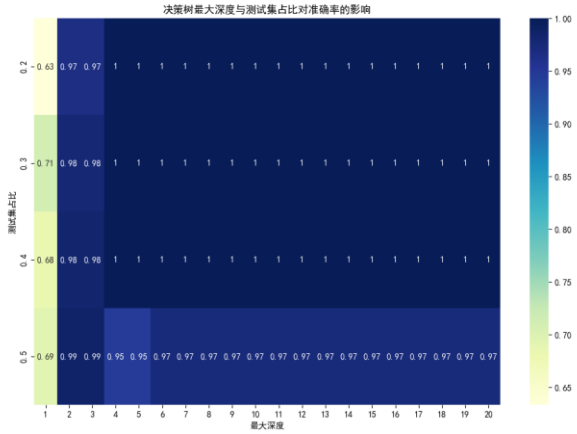
```python
# 搜索最优的最大深度和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for depth in max_depth_values:
        # 使用决策树分类
        dt = DecisionTreeClassifier(max_depth=depth, random_state=42)
        dt.fit(X_train, y_train)  # 训练模型
        y_pred = dt.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_depth = depth
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果
```

（4）热力图可视化参数组合效果

（5）最优结果可视化

（6）运行结果截图分析





分析：

（1）在较低的测试集比例时（0.2、0.3）和较大的最大深度设置下，决策树能轻松达到接近1.0的准确率。这说明数据集可能较为简单，或者决策树有过拟合的倾向。

（2）随着最大深度变化，最终差异不大，尤其在小测试比例时表现近乎完美。然而在测试比例增大（如0.5）后，准确率虽有下降，但仍保持在0.95上下，说明在该数据集上决策树即便简单调参也能有相当强的表现。

4. 随机森林算法

原理：

由多个决策树组成的集成学习方法：通过对训练数据的随机采样（袋外法）和特征随机选择，构建多棵决策树，最终通过分类结合所有树的输出结果。

核心步骤：

（1）随机性：通过数据随机采样和特征随机选择，降低单一决策树的方差，增强泛化能力。

（2）参数选择：关键参数包括决策树的数量（n_estimators）、每棵树的最大深度（max_depth）等。

（3）集成策略：对多个弱分类器（决策树）结果进行整合

实验过程：

与KNN算法的实验过程类似，不过在参数调优方面有所不同，故只对这方面进行解 释。

（1）数据预处理

（2）设置分类任务的输入和输出

（3）参数调优

A. 定义搜索范围

树的数量：选择不同的值（10, 20, 40, 50, 100, 150, 200），影响模型复杂度和准确率。

测试集占比：选择不同的测试集比例（20%，30%，40%，50%）来影响训练和测试的分配。

```
# 定义参数搜索范围
n_estimators_values = [10, 20, 40, 50, 100,150, 200]  # 随机森林树的数量
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比
```

B. 搜索过程

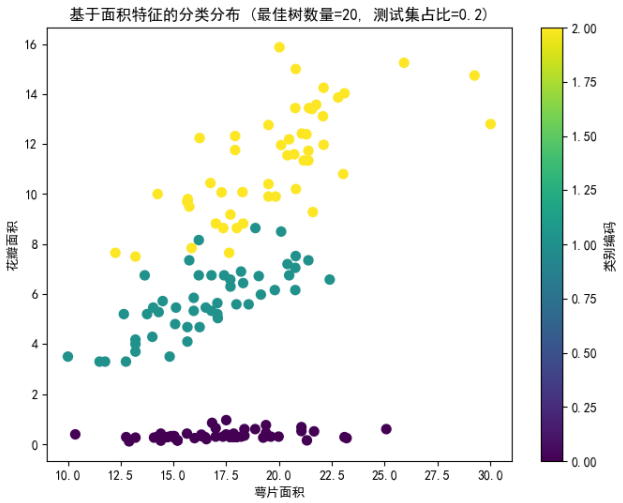
使用 train_test_split 将数据集划分为训练集和测试集，根据不同的测试集占比进行循环。对每个占比，再通过循环调整树的数量，分别训练不同的随机森林模型。计算每种参数组合的模型准确率，并记录结果。寻找最优参数组合，即获得最高准确率的树的数量和测试集占比。

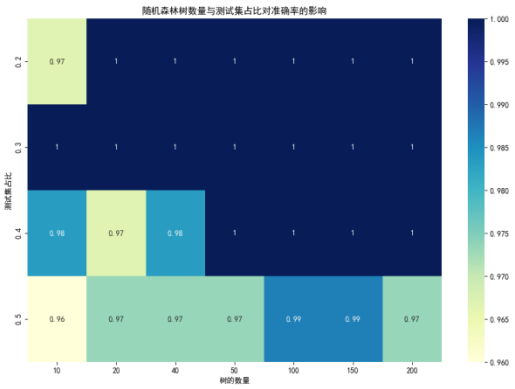
```python
# 搜索最优的树数量和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for n_estimators in n_estimators_values:
        # 使用随机森林分类
        rf = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
        rf.fit(X_train, y_train)  # 训练模型
        y_pred = rf.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_n_estimators = n_estimators
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果
```

（4）热力图可视化参数组合效果

（5）最优结果可视化

（6）运行结果截图分析





分析：

（1）随机森林在较小测试比例下也轻松达到1.0的准确率。在较大的测试比例下，只要树的数量足够多（如100棵树以上），准确率依旧能维持在0.97~0.99左右，表现极其稳定。

（2）相对决策树而言，随机森林通过集成能够更有效地降低方差，从而在更大比例测试集下仍保持很高的泛化性能。

5. 比较分析

（1）所有模型在较小的测试集比例下获得超高准确率，可能并不代表模型真实的泛化能力，应在相对较大比例的测试集（如0.3或0.4以上）下考察模型表现更为合理。

（2）决策树与KNN尽管简单，但在本数据集上也能表现突出，尤其当参数（最大深度或K值）优化到合理范围时。

（3）随机森林通过增加树的数量，即便在较大的测试集比例下也能保持接近1的准确率，表现非常强劲，说明其对数据集有良好的拟合与泛化能力。

（4）SVM的核函数选择在此数据集下非常关键，linear与rbf在泛化表现上相对更稳定优异，而poly与sigmoid在测试集比例增大后精度显著下降。

对于问题二，使用KNN、支持向量机、决策树、随机森林等算法进行分类。
原理与核心步骤不再赘述，只给出实验过程。
1.KNN算法：
实验过程：
（1）数据加载与清洗

A. 加载数据：从 bank-additional.csv 文件中读取数据。

B. 清理列名：移除多余的空格和双引号，确保列名清晰整洁。

```
# 1. 数据加载
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)
```

（2）数据预处理

A. 处理分类变量：提取所有分类变量（除目标列 y），并对这些变量使用独热编码 (one-hot encoding)。目标变量y被转化为数值形式（0和1），使用LabelEncoder。

B. 分割特征和目标：将数据分为特征矩阵 X 和目标向量y。

C. 标准化：对特征矩阵进行标准化，确保不同量纲的特征不会对模型产生不均衡的影响。

```
# 2. 数据预处理
# 提取分类变量并删除目标列
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# 将目标变量进行编码
label_encoder = LabelEncoder()
data['y'] = label_encoder.fit_transform(data['y'])

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

（3）数据集划分

划分训练集和测试集

```
# 3. 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

（4）参数调优

A.设置参数网格：

选择 KNN 模型的三个主要超参数：

a)n_neighbors：邻居数量。

b)weights：权重分配方式（等权或基于距离）。

c)p：距离度量方式（曼哈顿距离或欧几里得距离）

B.网格搜索：

利用 GridSearchCV 对不同参数组合进行交叉验证，寻找最佳超参数。

使用 cv=5 表示五折交叉验证。

```
# 4. 参数调优和模型训练
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=1)  # 禁用并行

grid_search.fit(X_train, y_train)
```

（5）结果评估

评估模型的准确率，并生成分类报告，提供精确率、召回率和 F1 分数。

```python
# 5. 最优参数与结果评估
best_params = grid_search.best_params_
print("最佳参数:", best_params)

# 使用最佳参数预测
y_pred = grid_search.best_estimator_.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"最佳KNN模型准确率: {accuracy:.2f}")
print("分类报告:\n", classification_report(y_test, y_pred, target_names=['否', '是']))
```

（6）可视化分析

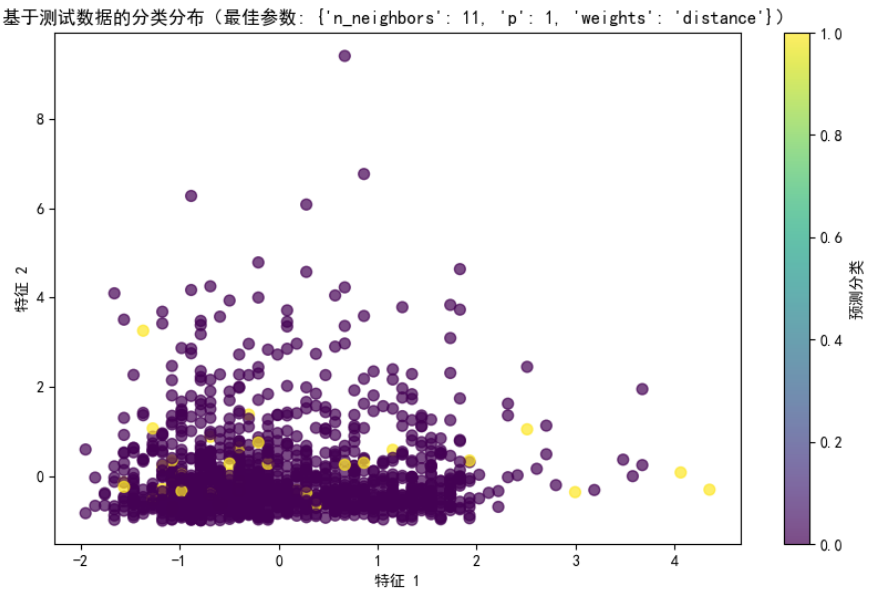
A. 提取交叉验证结果，将不同超参数组合的平均测试得分以热力图展示，直观呈 现参数对模型性能的影响。

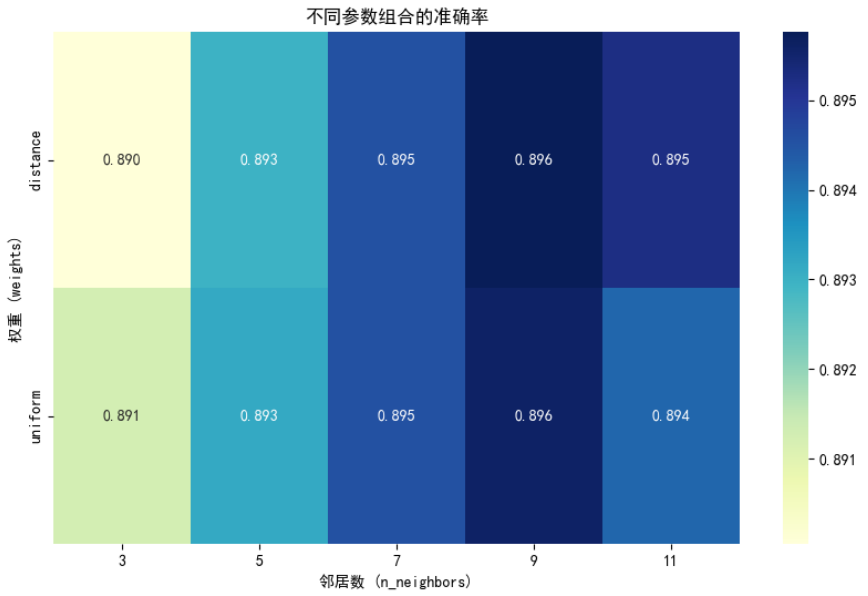
```python
# 6. 可视化参数组合效果
results = pd.DataFrame(grid_search.cv_results_)
heatmap_data = results.pivot_table(index='param_weights', columns='param_n_neighbors', values='mean_test_score')

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, fmt='.3f', cmap='YlGnBu')
plt.title('不同参数组合的准确率')
plt.xlabel('邻居数 (n_neighbors)')
plt.ylabel('权重 (weights)')
plt.show()
```

B.利用测试集的两个特征，绘制分类散点图，展示分类效果及类别分布。

```python
# 7. 最优参数的散点图
# 使用测试集中的两个特征（假设特征0和特征1）进行散点图展示
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
plt.colorbar(label='预测分类')
plt.title(f'基于测试数据的分类分布（最佳参数: {best_params}）')
plt.xlabel('特征 1')
plt.ylabel('特征 2')
plt.show()
```

（7）运行结果截图分析






```
最佳参数: {'n_neighbors': 11, 'p': 1, 'weights': 'distance'}
最佳KNN模型准确率: 0.90
分类报告:
              precision    recall  f1-score   support

          否       0.91      0.99      0.95      1105
          是       0.61      0.18      0.27       131

    accuracy                           0.90      1236
   macro avg       0.76      0.58      0.61      1236
weighted avg       0.88      0.90      0.88      1236
```

分析:

（1）随着邻居数n_neighbors的增加，模型准确率有小幅上升，但在达到某个临界点（如9）后准确率下降。

（2）weights（权重）选择对结果的影响不大，distance权重略优于uniform。

（3）KNN对于邻居数参数敏感，需要仔细调节以找到合适的值。

2.向量机算法:

与KNN算法的实验过程类似，不过在参数调优方面有所不同，故只对这方面进行解释。

实验过程:

（1）数据加载与清洗

（2）数据预处理

（3）数据集划分

（4）参数调优

A.设置参数网格:

选择 SVM 模型的三个主要超参数:

a)C: 惩罚参数，控制模型复杂度与正则化的权衡。

b)gamma: 核函数参数，影响支持向量对决策边界的影响范围。

c)kernel: 核函数类型，支持 rbf（高斯核）和 linear。

B.网格搜索:

利用 GridSearchCV 对不同参数组合进行交叉验证，寻找最佳超参数。

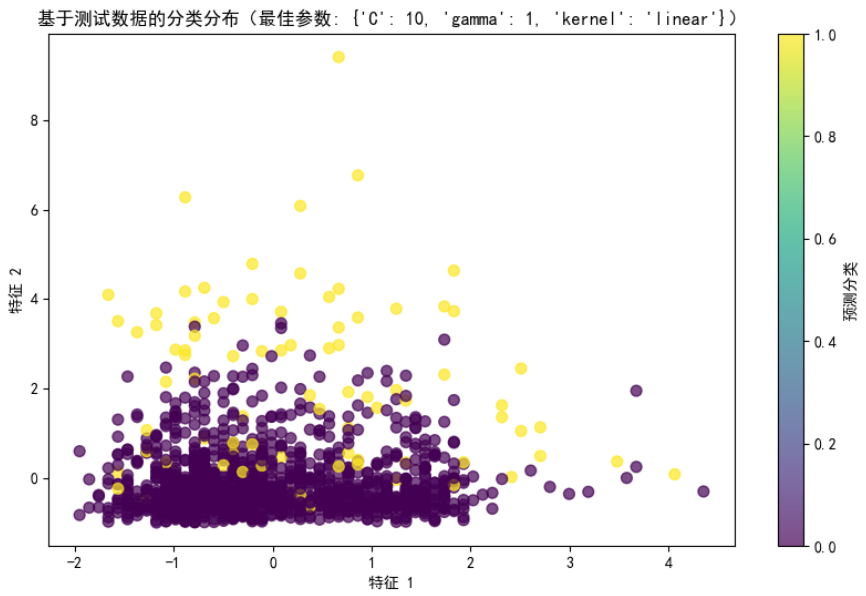
使用 cv=5 表示五折交叉验证。

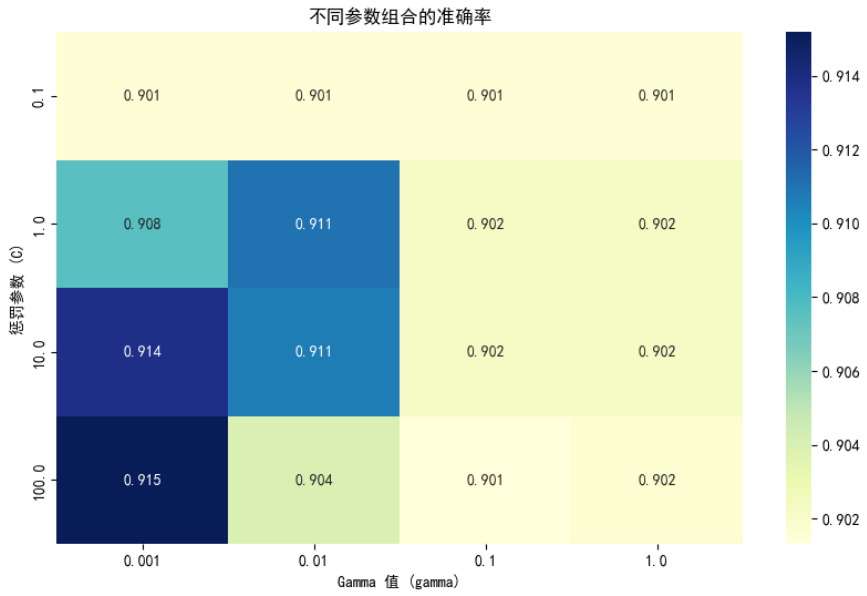
```
# 4. 参数调优和模型训练
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear']
}

svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=1)

grid_search.fit(X_train, y_train)
```

（5）结果评估

（6）可视化分析

（7）运行结果截图分析






```
最佳参数: {'C': 10, 'gamma': 1, 'kernel': 'linear'}
最佳支持向量机模型准确率: 0.90
分类报告:
              precision    recall  f1-score   support

          否       0.93      0.96      0.95      1105
          是       0.56      0.38      0.45       131

    accuracy                           0.90      1236
   macro avg       0.74      0.67      0.70      1236
weighted avg       0.89      0.90      0.89      1236
```

分析:

（1）SVM的准确率在不同参数组合中总体表现稳定，变化幅度较小。

（2）随着惩罚参数C的增大，模型的准确率有小幅提升，但Gamma值对结果的影响较小。

（3）SVM对参数的敏感性较低，适合用在准确率稳定性要求较高的场景。

3.决策树算法:

与KNN算法的实验过程类似，不过在参数调优方面有所不同，故只对这方面进行解释。

实验过程:

（1）数据加载与清洗

（2）数据预处理

（3）数据集划分

（4）参数调优

A.设置参数网格:

为决策树的超参数（如 criterion, max_depth, min_samples_split, min_samples_leaf）设定多个候选值。

B.网格搜索:

利用 GridSearchCV 对不同参数组合进行交叉验证，寻找最佳超参数。

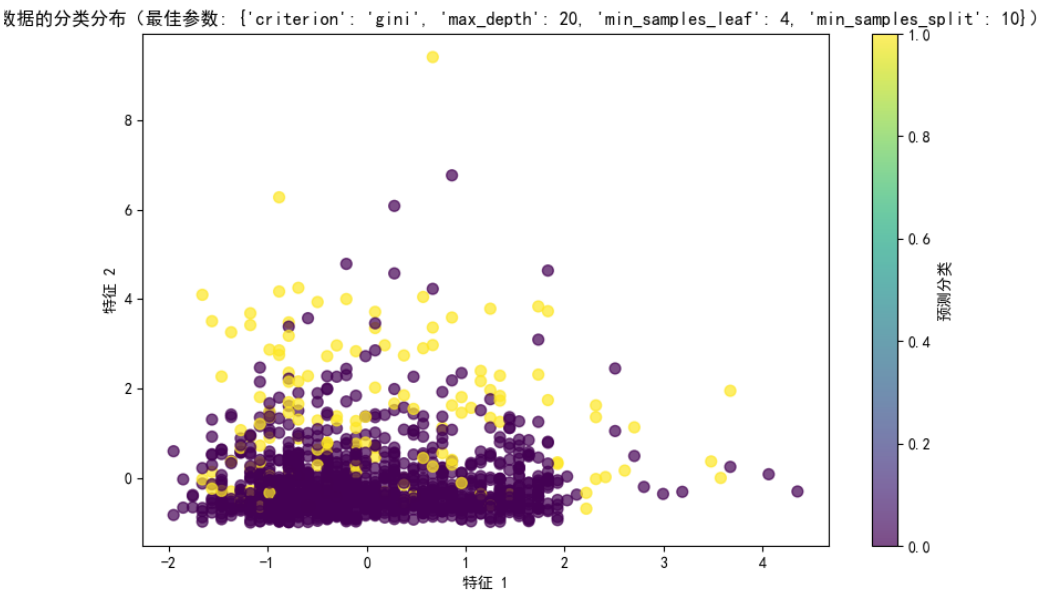
使用 cv=5 表示五折交叉验证。

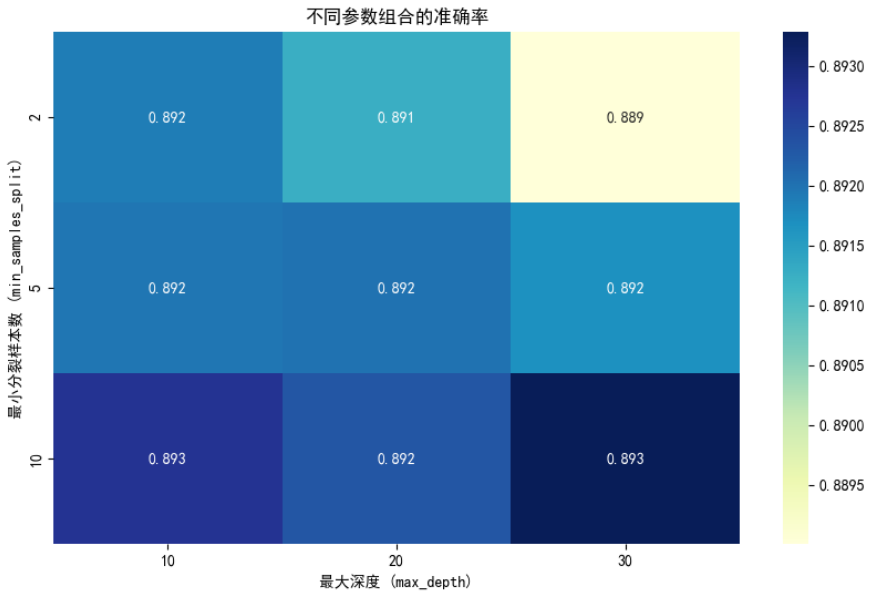
```
# 4. 参数调优和模型训练
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt = DecisionTreeClassifier()
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy', n_jobs=1)

grid_search.fit(X_train, y_train)
```

（5）结果评估

（6）可视化分析

（7）运行结果截图分析






```
最佳参数: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10}
最佳决策树模型准确率: 0.88
分类报告:
              precision    recall  f1-score   support

          否       0.94      0.93      0.94      1105
          是       0.46      0.47      0.46       131

    accuracy                           0.88      1236
   macro avg       0.70      0.70      0.70      1236
weighted avg       0.89      0.88      0.89      1236
```

分析：

（1）最大深度max_depth的增加对性能提升作用不显著。

（2）减少min_samples_split（最小分裂样本数）能在一定程度上提高准确率。

（3）决策树的单模型效果有限，但其简单性和快速性使其更适合快速建立基线模型。

4.随机森林算法：

与KNN算法的实验过程类似，不过在参数调优方面有所不同，故只对这方面进行解释。

实验过程：

（1）数据加载与清洗

（2）数据预处理

（3）数据集划分

（4）参数调优

A.设置参数网格：

随机森林的主要超参数包括：

a)n_estimators：树的数量。

b)max_depth：树的最大深度。

c)min_samples_split：节点分裂所需的最小样本数。

d)min_samples_leaf：叶子节点的最小样本数。

B.网格搜索:

利用 GridSearchCV 对不同参数组合进行交叉验证，寻找最佳超参数。

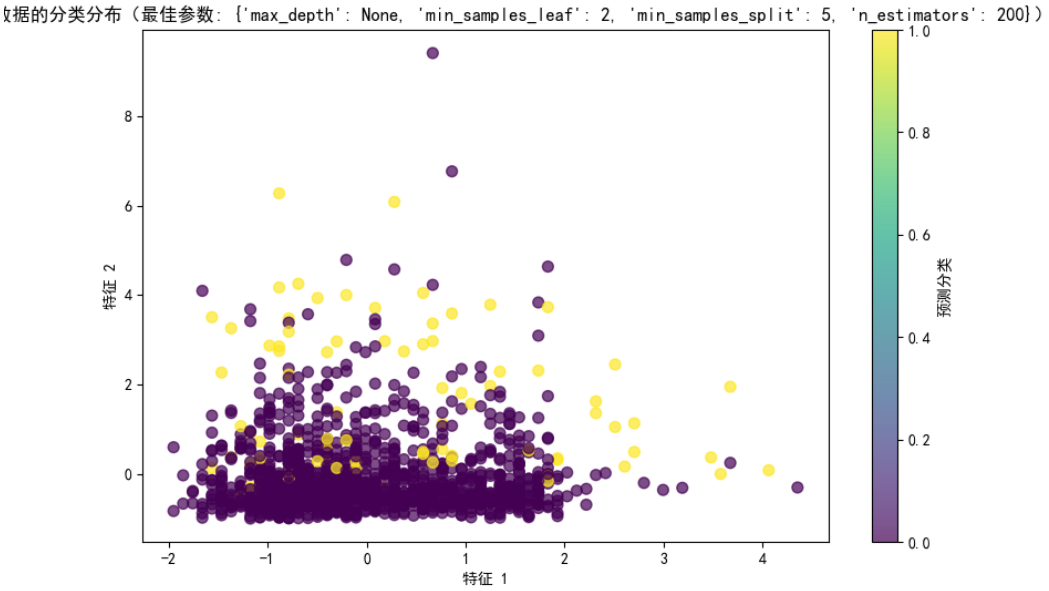
使用 cv=5 表示五折交叉验证。

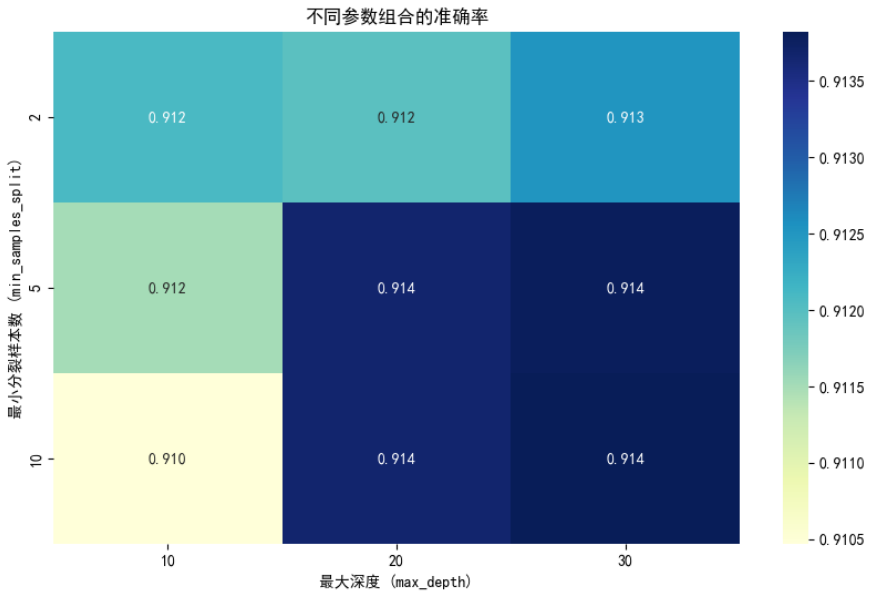
```
# 4. 参数调优和模型训练
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=1)

grid_search.fit(X_train, y_train)
```

（5）结果评估

（6）可视化分析

（7）运行结果截图分析






```
最佳参数: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 200}
最佳随机森林模型准确率: 0.90
分类报告:
              precision    recall  f1-score   support

          否       0.93      0.97      0.95      1105
          是       0.57      0.37      0.45       131

    accuracy                           0.90      1236
   macro avg       0.75      0.67      0.70      1236
weighted avg       0.89      0.90      0.89      1236
```

分析:

（1）随着max_depth（最大深度）的增加，模型的准确率也有小幅提升。

（2）较小的min_samples_split（最小分裂样本数）表现出稍高的准确率。

（3）模型对参数的变化不敏感，准确率差距很小，表现较为平稳。

（4）随机森林的鲁棒性较强，能够处理较大的参数范围。

5.比较分析

（1）模型表现稳定性:

随机森林和SVM的表现更稳定，对参数变化不敏感。

KNN和决策树的表现对参数调整更敏感，需更细致的调优。

（2）最高准确率:

随机森林和SVM在多种参数组合下的准确率表现最佳，分别达到了0.914和0.915。

（3）适用场景:

SVM适合于对准确率稳定性要求较高的任务。

随机森林适合处理高维数据和参数范围大的问题。

KNN适用于小数据集的快速预测。

决策树适用于快速构建基线模型。

五、总结心得

通过本次实验，我了解了KNN、SVM、决策树和随机森林等分类算法的原理和应用，重点分析了参数调优对模型性能的影响。实验表明，随机森林和SVM在准确率和稳定性上表现最佳，适用于高维数据和对精度要求高的场景；决策树和KNN虽简单，但在快速建模中效果突出。通过特征工程、热力图可视化和多算法对比，我深刻体会到数据预处理、参数调优和模型选择的重要性。我不仅巩固了分类算法的理论知识，还提升了数据处理、算法实现、参数调优和模型评估的综合能力。

附录（所有代码）

Question1_KNN.py

```python
# 导入必要的库
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns  # 用于绘制热力图

# 配置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)

# 打印数据预览
print("数据集预览：")
print(data.head())

# 2. 特征工程：计算面积
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']

# 打印新特征
print("\n新增特征（萼片面积和花瓣面积）：")
print(data[['sepal_area', 'petal_area']].head())

# 3. 构建新特征矩阵和标签
X = data[['sepal_area', 'petal_area']].values  # 只保留面积特征
```

```
y = data['class'].values  # 分类标签

# 定义参数搜索范围
k_values = range(1, 21)  # 测试 k 值从 1 到 20
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比

# 存储最佳参数和最高准确率
best_k = None
best_split = None
highest_accuracy = 0

# 用于存储不同参数组合的准确率
accuracy_matrix = []

# 搜索最优的 k 值和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for k in k_values:
        # 使用KNN分类
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)  # 训练模型
        y_pred = knn.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_k = k
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果

# 转换为 NumPy 数组以便绘制热力图
```

```
accuracy_matrix = np.array(accuracy_matrix)

# 输出最佳参数组合
print("\n最优结果：")
print(f"最佳 k 值：{best_k}")
print(f"最佳测试集占比：{best_split}")
print(f"最高准确率：{highest_accuracy:.2f}")

# 绘制热力图
plt.figure(figsize=(12, 8))
sns.heatmap(accuracy_matrix, annot=True, cmap="YlGnBu", xticklabels=k_values,
yticklabels=split_ratios, cbar=True)
plt.xlabel("k 值")
plt.ylabel("测试集占比")
plt.title("K 值与测试集占比对准确率的影响")
plt.show()

# 可视化最优结果
# 使用最佳参数重新训练模型并可视化
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=best_split, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# 可视化分类分布
plt.figure(figsize=(8, 6))
plt.scatter(data['sepal_area'], data['petal_area'], c=pd.Categorical(data['class']).codes, cmap='viridis',
s=50)
plt.colorbar(label='类别编码')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.title(f'基于面积特征的分类分布（最佳 k={best_k}，测试集占比={best_split}）')
plt.show()
```

Question1_SWM.py

```
# 导入必要的库
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

```python
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns  # 用于绘制热力图

# 配置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)

# 打印数据预览
print("数据集预览：")
print(data.head())

# 2. 特征工程：计算面积
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']

# 打印新特征
print("\n新增特征（萼片面积和花瓣面积）：")
print(data[['sepal_area', 'petal_area']].head())

# 3. 构建新特征矩阵和标签
X = data[['sepal_area', 'petal_area']].values  # 只保留面积特征
y = data['class'].values  # 分类标签

# 定义参数搜索范围
kernels = ['linear', 'poly', 'rbf', 'sigmoid']  # SVM 的核函数类型
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比

# 存储最佳参数和最高准确率
best_kernel = None
best_split = None
highest_accuracy = 0

# 用于存储不同参数组合的准确率
accuracy_matrix = []

# 搜索最优的核函数和测试集占比
```

```
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for kernel in kernels:
        # 使用支持向量机分类
        svc = SVC(kernel=kernel, random_state=42)
        svc.fit(X_train, y_train)  # 训练模型
        y_pred = svc.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_kernel = kernel
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果

# 转换为 NumPy 数组以便绘制热力图
accuracy_matrix = np.array(accuracy_matrix)

# 输出最佳参数组合
print("\n最优结果：")
print(f"最佳核函数：{best_kernel}")
print(f"最佳测试集占比：{best_split}")
print(f"最高准确率：{highest_accuracy:.2f}")

# 绘制热力图
plt.figure(figsize=(12, 8))
sns.heatmap(accuracy_matrix, annot=True, cmap="YlGnBu", xticklabels=kernels,
yticklabels=split_ratios, cbar=True)
plt.xlabel("核函数类型")
plt.ylabel("测试集占比")
plt.title("SVM 核函数与测试集占比对准确率的影响")
```

```
plt.show()

# 可视化最优结果
# 使用最佳参数重新训练模型并可视化
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=best_split, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svc = SVC(kernel=best_kernel, random_state=42)
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)

# 可视化分类分布
plt.figure(figsize=(8, 6))
plt.scatter(data['sepal_area'], data['petal_area'], c=pd.Categorical(data['class']).codes, cmap='viridis',
s=50)
plt.colorbar(label='类别编码')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.title(f'基于面积特征的分类分布（最佳核函数={best_kernel}，测试集占比={best_split}）')
plt.show()
```

Question1_Desicion_Tree.py

```
# 导入必要的库
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns  # 用于绘制热力图

# 配置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)

# 打印数据预览
```

```
print("数据集预览：")
print(data.head())

# 2. 特征工程：计算面积
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']

# 打印新特征
print("\n新增特征（萼片面积和花瓣面积）：")
print(data[['sepal_area', 'petal_area']].head())

# 3. 构建新特征矩阵和标签
X = data[['sepal_area', 'petal_area']].values  # 只保留面积特征
y = data['class'].values  # 分类标签

# 定义参数搜索范围
max_depth_values = range(1, 21)  # 测试决策树最大深度从 1 到 20
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比

# 存储最佳参数和最高准确率
best_depth = None
best_split = None
highest_accuracy = 0

# 用于存储不同参数组合的准确率
accuracy_matrix = []

# 搜索最优的最大深度和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for depth in max_depth_values:
        # 使用决策树分类
        dt = DecisionTreeClassifier(max_depth=depth, random_state=42)
        dt.fit(X_train, y_train)  # 训练模型
        y_pred = dt.predict(X_test)  # 测试集预测
```

```
        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
          accuracy_row.append(accuracy)

          # 更新最佳参数
        if accuracy > highest_accuracy:
                highest_accuracy = accuracy
                best_depth = depth
                best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果

# 转换为 NumPy 数组以便绘制热力图
accuracy_matrix = np.array(accuracy_matrix)

# 输出最佳参数组合
print("\n最优结果：")
print(f"最佳最大深度：{best_depth}")
print(f"最佳测试集占比：{best_split}")
print(f"最高准确率：{highest_accuracy:.2f}")

# 绘制热力图
plt.figure(figsize=(12, 8))
sns.heatmap(accuracy_matrix, annot=True, cmap="YlGnBu", xticklabels=max_depth_values,
yticklabels=split_ratios, cbar=True)
plt.xlabel("最大深度")
plt.ylabel("测试集占比")
plt.title("决策树最大深度与测试集占比对准确率的影响")
plt.show()

# 可视化最优结果
# 使用最佳参数重新训练模型并可视化
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=best_split, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

dt = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)

# 可视化分类分布
plt.figure(figsize=(8, 6))
```

```
plt.scatter(data['sepal_area'], data['petal_area'], c=pd.Categorical(data['class']).codes, cmap='viridis',
s=50)
plt.colorbar(label='类别编码')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.title(f'基于面积特征的分类分布（最佳深度={best_depth}，测试集占比={best_split}）')
plt.show()
```

Question1_Random_Forest.py

```
# 导入必要的库
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns  # 用于绘制热力图

# 配置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文字体
plt.rcParams['axes.unicode_minus'] = False    # 解决负号显示问题

# 1. 加载数据
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(url, header=None, names=columns)

# 打印数据预览
print("数据集预览：")
print(data.head())

# 2. 特征工程：计算面积
# 计算萼片面积和花瓣面积
data['sepal_area'] = data['sepal_length'] * data['sepal_width']
data['petal_area'] = data['petal_length'] * data['petal_width']

# 打印新特征
print("\n新增特征（萼片面积和花瓣面积）：")
print(data[['sepal_area', 'petal_area']].head())

# 3. 构建新特征矩阵和标签
X = data[['sepal_area', 'petal_area']].values  # 只保留面积特征
y = data['class'].values  # 分类标签
```

```
# 定义参数搜索范围
n_estimators_values = [10, 20, 40, 50, 100,150, 200]  # 随机森林树的数量
split_ratios = [0.2, 0.3, 0.4, 0.5]  # 测试集占比

# 存储最佳参数和最高准确率
best_n_estimators = None
best_split = None
highest_accuracy = 0

# 用于存储不同参数组合的准确率
accuracy_matrix = []

# 搜索最优的树数量和测试集占比
for split_ratio in split_ratios:
    accuracy_row = []  # 每种测试集占比对应的一行准确率
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split_ratio, random_state=42)

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

    for n_estimators in n_estimators_values:
        # 使用随机森林分类
        rf = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
        rf.fit(X_train, y_train)  # 训练模型
        y_pred = rf.predict(X_test)  # 测试集预测

        # 计算准确率
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_row.append(accuracy)

        # 更新最佳参数
        if accuracy > highest_accuracy:
            highest_accuracy = accuracy
            best_n_estimators = n_estimators
            best_split = split_ratio

    accuracy_matrix.append(accuracy_row)  # 添加每种测试集占比的结果

# 转换为 NumPy 数组以便绘制热力图
accuracy_matrix = np.array(accuracy_matrix)
```

```
# 输出最佳参数组合
print("\n最优结果：")
print(f"最佳树数量：{best_n_estimators}")
print(f"最佳测试集占比：{best_split}")
print(f"最高准确率：{highest_accuracy:.2f}")

# 绘制热力图
plt.figure(figsize=(12, 8))
sns.heatmap(accuracy_matrix, annot=True, cmap="YlGnBu", xticklabels=n_estimators_values,
yticklabels=split_ratios, cbar=True)
plt.xlabel("树的数量")
plt.ylabel("测试集占比")
plt.title("随机森林树数量与测试集占比对准确率的影响")
plt.show()

# 可视化最优结果
# 使用最佳参数重新训练模型并可视化
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=best_split, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

rf = RandomForestClassifier(n_estimators=best_n_estimators, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# 可视化分类分布
plt.figure(figsize=(8, 6))
plt.scatter(data['sepal_area'], data['petal_area'], c=pd.Categorical(data['class']).codes, cmap='viridis',
s=50)
plt.colorbar(label='类别编码')
plt.xlabel('萼片面积')
plt.ylabel('花瓣面积')
plt.title(f'基于面积特征的分类分布 (最佳树数量={best_n_estimators}, 测试集占比={best_split})')
plt.show()
```

Question2_KNN.py

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# 设置 Matplotlib 中文字体
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 1. 数据加载
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)

# 2. 数据预处理
# 提取分类变量并删除目标列
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# 将目标变量进行编码
label_encoder = LabelEncoder()
data['y'] = label_encoder.fit_transform(data['y'])

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# 检查数据格式
print("X_train 数据类型:", type(X_train))
print("y_train 数据类型:", type(y_train))
print("X_train 是否有空值:", pd.isnull(X_train).any())
print("y_train 是否有空值:", pd.isnull(y_train).any())

# 4. 参数调优和模型训练
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
```

```
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=1)  # 禁用并行

grid_search.fit(X_train, y_train)

# 5. 最优参数与结果评估
best_params = grid_search.best_params_
print("最佳参数:", best_params)

# 使用最佳参数预测
y_pred = grid_search.best_estimator_.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"最佳KNN模型准确率: {accuracy:.2f}")
print("分类报告:\n", classification_report(y_test, y_pred, target_names=['否', '是']))

# 6. 可视化参数组合效果
results = pd.DataFrame(grid_search.cv_results_)
heatmap_data = results.pivot_table(index='param_weights', columns='param_n_neighbors',
values='mean_test_score')

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, fmt='.3f', cmap='YlGnBu')
plt.title('不同参数组合的准确率')
plt.xlabel('邻居数 (n_neighbors)')
plt.ylabel('权重 (weights)')
plt.show()

# 7. 最优参数的散点图
# 使用测试集中的两个特征（假设特征0和特征1）进行散点图展示
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
plt.colorbar(label='预测分类')
plt.title(f'基于测试数据的分类分布（最佳参数: {best_params}）')
plt.xlabel('特征 1')
plt.ylabel('特征 2')
plt.show()
```

Question2_SVM.py

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 设置 Matplotlib 中文字体
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 1. 数据加载
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)

# 2. 数据预处理
# 提取分类变量并删除目标列
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# 将目标变量进行编码
label_encoder = LabelEncoder()
data['y'] = label_encoder.fit_transform(data['y'])

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# 检查数据格式
print("X_train 数据类型:", type(X_train))
print("y_train 数据类型:", type(y_train))
print("X_train 是否有空值:", pd.isnull(X_train).any())
```

```
print("y_train 是否有空值:", pd.isnull(y_train).any())

# 4. 参数调优和模型训练
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear']
}

svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=1)

grid_search.fit(X_train, y_train)

# 5. 最优参数与结果评估
best_params = grid_search.best_params_
print("最佳参数:", best_params)

# 使用最佳参数预测
y_pred = grid_search.best_estimator_.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"最佳支持向量机模型准确率: {accuracy:.2f}")
print("分类报告:\n", classification_report(y_test, y_pred, target_names=['否', '是']))

# 6. 可视化参数组合效果
results = pd.DataFrame(grid_search.cv_results_)
heatmap_data = results.pivot_table(index='param_C', columns='param_gamma',
values='mean_test_score')

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, fmt='.3f', cmap='YlGnBu')
plt.title('不同参数组合的准确率')
plt.xlabel('Gamma 值 (gamma)')
plt.ylabel('惩罚参数 (C)')
plt.show()

# 7. 最优参数的散点图
# 使用测试集中的两个特征（假设特征0和特征1）进行散点图展示
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
plt.colorbar(label='预测分类')
plt.title(f'基于测试数据的分类分布（最佳参数: {best_params}）')
plt.xlabel('特征 1')
plt.ylabel('特征 2')
```

```
plt.show()

# 7. 使用测试集中的两个特征绘制分类分布散点图
# 使用 PCA 将特征降维到二维
#from sklearn.decomposition import PCA
#pca = PCA(n_components=2)
#X_test_pca = pca.fit_transform(X_test)

#plt.figure(figsize=(10, 6))
#plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
#plt.colorbar(label='预测分类')
#plt.title(f'基于 PCA 降维的分类分布（最佳参数: {best_params}）')
#plt.xlabel('主成分 1')
#plt.ylabel('主成分 2')
#plt.show()
```

Question2_Desicion_Tree.py

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 设置 Matplotlib 中文字体
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 1. 数据加载
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)

# 2. 数据预处理
# 提取分类变量并删除目标列
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)
```

```
# 将目标变量进行编码
label_encoder = LabelEncoder()
data['y'] = label_encoder.fit_transform(data['y'])

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# 检查数据格式
print("X_train 数据类型:", type(X_train))
print("y_train 数据类型:", type(y_train))
print("X_train 是否有空值:", pd.isnull(X_train).any())
print("y_train 是否有空值:", pd.isnull(y_train).any())

# 4. 参数调优和模型训练
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt = DecisionTreeClassifier()
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy', n_jobs=1)

grid_search.fit(X_train, y_train)

# 5. 最优参数与结果评估
best_params = grid_search.best_params_
print("最佳参数:", best_params)

# 使用最佳参数预测
y_pred = grid_search.best_estimator_.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"最佳决策树模型准确率: {accuracy:.2f}")
print("分类报告:\n", classification_report(y_test, y_pred, target_names=['否', '是']))
```

```
# 6. 可视化参数组合效果
results = pd.DataFrame(grid_search.cv_results_)
heatmap_data = results.pivot_table(index='param_min_samples_split', columns='param_max_depth',
values='mean_test_score')

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, fmt='.3f', cmap='YlGnBu')
plt.title('不同参数组合的准确率')
plt.xlabel('最大深度 (max_depth)')
plt.ylabel('最小分裂样本数 (min_samples_split)')
plt.show()

# 7. 最优参数的散点图
# 使用测试集中的两个特征（假设特征0和特征1）进行散点图展示
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
plt.colorbar(label='预测分类')
plt.title(f'基于测试数据的分类分布（最佳参数: {best_params}）')
plt.xlabel('特征 1')
plt.ylabel('特征 2')
plt.show()
```

Question2_Random_Froest.py

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 设置 Matplotlib 中文字体
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体
matplotlib.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 1. 数据加载
file_path = "bank-additional.csv"
data = pd.read_csv(file_path, sep=';')

# 清理列名
data.columns = data.columns.str.strip().str.replace('"', '')
print("清理后的列名:", data.columns)
```

```
# 2. 数据预处理
# 提取分类变量并删除目标列
categorical_columns = data.select_dtypes(include=['object']).columns.drop('y')  # 确保 y 是 object 类型
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)

# 将目标变量进行编码
label_encoder = LabelEncoder()
data['y'] = label_encoder.fit_transform(data['y'])

# 分割特征和目标
X = data.drop('y', axis=1)
y = data['y']

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# 检查数据格式
print("X_train 数据类型:", type(X_train))
print("y_train 数据类型:", type(y_train))
print("X_train 是否有空值:", pd.isnull(X_train).any())
print("y_train 是否有空值:", pd.isnull(y_train).any())

# 4. 参数调优和模型训练
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=1)

grid_search.fit(X_train, y_train)

# 5. 最优参数与结果评估
best_params = grid_search.best_params_
print("最佳参数:", best_params)
```

```
# 使用最佳参数预测
y_pred = grid_search.best_estimator_.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"最佳随机森林模型准确率: {accuracy:.2f}")
print("分类报告:\n", classification_report(y_test, y_pred, target_names=['否', '是']))

# 6. 可视化参数组合效果
results = pd.DataFrame(grid_search.cv_results_)
heatmap_data = results.pivot_table(index='param_min_samples_split', columns='param_max_depth',
values='mean_test_score')

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, fmt='.3f', cmap='YlGnBu')
plt.title('不同参数组合的准确率')
plt.xlabel('最大深度 (max_depth)')
plt.ylabel('最小分裂样本数 (min_samples_split)')
plt.show()

# 7. 最优参数的散点图
# 使用测试集中的两个特征（假设特征0和特征1）进行散点图展示
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
plt.colorbar(label='预测分类')
plt.title(f'基于测试数据的分类分布（最佳参数: {best_params}）')
plt.xlabel('特征 1')
plt.ylabel('特征 2')
plt.show()

# 7. 使用测试集中的两个特征绘制分类分布散点图
# 使用 PCA 将特征降维到二维
#from sklearn.decomposition import PCA
#pca = PCA(n_components=2)
#X_test_pca = pca.fit_transform(X_test)

#plt.figure(figsize=(10, 6))
#plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred, cmap='viridis', s=50, alpha=0.7)
#plt.colorbar(label='预测分类')
#plt.title(f'基于 PCA 降维的分类分布（最佳参数: {best_params}）')
#plt.xlabel('主成分 1')
#plt.ylabel('主成分 2')
#plt.show()
```