一. (30 points) 随机森林的原理

集成学习是一种通用技术，通过将多个不同模型的预测结果结合起来，以平均值或多数投票的方式生成单一预测，从而有效应对过拟合问题。

1. 考虑一组互不相关的随机变量 $\{Y_i\}_{i=1}^n$，其均值为 μ，方差为 σ^2。请计算这些随机变量平均值的期望和方差，给出计算过程。提示：在集成方法的背景下，这些 Y_i 类似于分类器 i 所作的预测。(5 points)

解:

设这组随机变量的平均值为 $\overline{Y}$，即：

$$\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

$$\mathbb{E}[\overline{Y}] = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[Y_i]$$

由于每个 Y_i 的期望均为 μ，因此：

$$\mathbb{E}[\overline{Y}] = \frac{1}{n}\sum_{i=1}^{n}\mu = \frac{n\mu}{n} = \mu$$

对于常数 a 和随机变量 X，有 $\mathrm{Var}(aX) = a^2\mathrm{Var}(X)$。

$$\mathrm{Var}(\overline{Y}) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \frac{1}{n^2}\mathrm{Var}\left(\sum_{i=1}^{n} Y_i\right)$$

由于 Y_i 互不相关，且方差的性质表明：

$$\mathrm{Var}\left(\sum_{i=1}^{n} Y_i\right) = \sum_{i=1}^{n}\mathrm{Var}(Y_i)$$

每个 Y_i 的方差均为 σ^2，因此：

$$\mathrm{Var}\left(\sum_{i=1}^{n} Y_i\right) = \sum_{i=1}^{n}\sigma^2 = n\sigma^2$$

$$\mathrm{Var}(\overline{Y}) = \frac{1}{n^2}\cdot n\sigma^2 = \frac{\sigma^2}{n}$$

这组随机变量平均值的期望为 μ，方差为 $\frac{\sigma^2}{n}$。

由此我们可以得到，随着集成中模型数量 n 的增加，平均预测的方差会减小，从而提高预测的稳定性和准确性。

2. 在第 1 小问中，我们看到对于不相关的分类器，取平均可以有效减少方差。尽管现实中的预测不可能完全不相关，但降低决策树之间的相关性通常能减少最终方差。现在，重新考虑一组具有相关性的随机变量 $\{Z_i\}_{i=1}^n$，其均值为 μ，方差为 σ^2，每个 $Z_i \in \mathbb{R}$ 为标量。假设对任意 $i \neq j$，$\mathrm{Corr}(Z_i, Z_j) = \rho$。提示：如果你不记得相关性与协方差之间的关系，请回顾你的概率论等课程内容。

- 请计算随机变量 Z_i 平均值的方差，以 σ、ρ 和 n 为变量表示，给出计算过程。(5 points)
- 当 n 非常大时，会发生什么？这对于取平均的潜在有效性说明了什么？(……如果 ρ 很大（$|\rho| \approx 1$）会怎样？……如果 ρ 非常小（$|\rho| \approx 0$）又会怎样？……如果 ρ 处于中等水平（$|\rho| \approx 0.5$）呢?）无需严格推导——基于你得出的方差公式，用定性分析进行讨论即可。(6 points)

解:

(1) 设这组随机变量的平均值为：$\overline{Z} = \frac{1}{n}\sum_{i=1}^n Z_i$

$\overline{Z}$ 的方差：$\mathrm{Var}(\overline{Z}) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^n Z_i\right)$

对于常数 a 和随机变量 X，有 $\mathrm{Var}(aX) = a^2\mathrm{Var}(X)$，因此：

$$\mathrm{Var}(\overline{Z}) = \frac{1}{n^2}\mathrm{Var}\left(\sum_{i=1}^n Z_i\right)$$

$\sum_{i=1}^n Z_i$ 的方差：

$$\mathrm{Var}\left(\sum_{i=1}^n Z_i\right) = \sum_{i=1}^n \mathrm{Var}(Z_i) + \sum_{i\neq j} \mathrm{Cov}(Z_i, Z_j)$$

其中，$\mathrm{Var}(Z_i) = \sigma^2$，而协方差 $\mathrm{Cov}(Z_i, Z_j)$ 可以通过相关系数 ρ 表示为：

$$\mathrm{Cov}(Z_i, Z_j) = \rho\sigma^2$$

$$\mathrm{Var}\left(\sum_{i=1}^n Z_i\right) = n\sigma^2 + n(n-1)\rho\sigma^2$$

$$\mathrm{Var}(\overline{Z}) = \frac{1}{n^2}\left(n\sigma^2 + n(n-1)\rho\sigma^2\right) = \frac{\sigma^2}{n}\left(1 + (n-1)\rho\right)$$

因此，$\overline{Z}$ 的方差为：

$$\mathrm{Var}(\overline{Z}) = \frac{\sigma^2}{n}\left(1 + (n-1)\rho\right)$$

(2) 当 n 趋于无穷大时，$\mathrm{Var}(\overline{Z})$ 的行为取决于 ρ 的值：

- 如果 $\rho \approx 1$：此时，$\mathrm{Var}(\overline{Z}) \approx \frac{\sigma^2}{n} \cdot n = \sigma^2$，即平均值的方差趋近于单个随机变量的方差，表明高度相关的随机变量的平均值并不能有效降低方差。
- 如果 $\rho \approx 0$：此时，$\mathrm{Var}(\overline{Z}) \approx \frac{\sigma^2}{n}$，即平均值的方差随着 n 的增加而减小，表明不相关的随机变量的平均值可以有效降低方差。
- 如果 $\rho \approx 0.5$：此时，$\mathrm{Var}(\overline{Z}) \approx \frac{\sigma^2}{n} \cdot (1 + 0.5n)$，当 n 增加时，方差趋近于 $0.5\sigma^2$，即平均值的方差仍然会减小，但减小的程度取决于相关性的大小。

综上所述，集成学习中，降低模型之间的相关性有助于减少平均预测的方差，从而提高预测的稳定性和准确性。

3. Bagging 是一种通过随机化从同一数据集生成多个不同学习器的方法。给定一个大小为 n 的训练集，Bagging 会通过有放回抽样生成 T 个随机子样本集，每个子样本集大小为 n'。在每个子样本集中，一些数据点可能会被多次选中，而另一些可能完全未被选中。当 $n' = n$ 时，大约 63% 的数据点会被选中，而剩下的 37% 被称为袋外样本点（Out-of-Bag，OOB）。

- 为什么是 63%？提示：当 n 很大时，某个样本点未被选中的概率是多少？请只考虑在任意一个子样本集中（而不是所有 T 个子样本集）未被选中的概率。（7 points）
- 关于决策树的数量 T。集成中的决策树数量通常需要在运行时间和降低方差之间进行权衡（典型值范围从几十到几千棵树）。而子样本大小 n' 对运行时间的影响较小，因此选择 n' 时主要考虑如何获得最优预测结果。虽然常见实践是设置 $n' = n$，但这并非总是最佳选择。你会如何建议我们选择超参数 n'？（7 points）

解:

（1）对于原始数据集中的任意一个数据点，在一次抽样中未被选中的概率为：$1 - \frac{1}{n}$

由于是有放回抽样，进行 n 次抽样后，该数据点始终未被选中的概率为：$\left(1 - \frac{1}{n}\right)^n$

当 n 很大时，$\left(1 - \frac{1}{n}\right)^n$ 近似于 e^{-1}，即 0.3679

因此，任意一个数据点未被选中的概率约为 36.79%，被选中的概率约为 63.21%。

（2）

- 训练数据较少的情况下，可以设置 $n' = n$，以最大化每个分类器的训练数据利用率，从而降低偏差。
- 训练数据较多的情况下，可以适当减小 n'，增加分类器之间的多样性，从而降低方差。
- 使用交叉验证测试不同的 n' 值，根据验证集的性能选择最优的子样本大小。

二. (20 points) 随机森林的实现

在本题中，你将实现决策树和随机森林，用于在以下两个数据集上进行分类：1）垃圾邮件数据集，2）泰坦尼克号数据集（用于预测这场著名灾难的幸存者）。数据已随作业提供。

为方便起见，我们提供了初始代码，其中包括预处理步骤和部分功能的实现。你可以自由选择使用或不使用这些代码来完成实现。

```
"""
注意：
1. 这个框架提供了基本的结构，您需要完成所有标记为 'pass' 的函数。
2. 记得处理数值稳定性问题，例如在计算对数时避免除以零。
3. 在报告中详细讨论您的观察结果和任何有趣的发现。
"""
from collections import Counter

import numpy as np
from numpy import genfromtxt
import scipy.io
from scipy import stats
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import cross_val_score
import pandas as pd
from pydot import graph_from_dot_data
import io

import random
random.seed(246810)
np.random.seed(246810)

eps = 1e-5  # a small number

class BaggedTrees(BaseEstimator, ClassifierMixin):

    def __init__(self, params=None, n=200):
        if params is None:
            params = {}
        self.params = params
        self.n = n
        self.decision_trees = [
            DecisionTreeClassifier(random_state=i, self.params)
            for i in range(self.n)
        ]

    def fit(self, X, y):
        # TODO
        pass

    def predict(self, X):
        # TODO
        pass
```

```
45
46
47 class RandomForest(BaggedTrees):
48
49     def __init__(self, params=None, n=200, m=1):
50         if params is None:
51             params = {}
52         params['max_features'] = m
53         self.m = m
54         super().__init__(params=params, n=n)
55
56
57 class BoostedRandomForest(RandomForest):
58     # OPTIONAL
59     def fit(self, X, y):
60         # TODO
61         pass
62
63     def predict(self, X):
64         # TODO
65         pass
66
67
68 def preprocess(data, fill_mode=True, min_freq=10, onehot_cols=[]):
69     # Temporarily assign -1 to missing data
70     data[data == b''] = '-1'
71
72     # Hash the columns (used for handling strings)
73     onehot_encoding = []
74     onehot_features = []
75     for col in onehot_cols:
76         counter = Counter(data[:, col])
77         for term in counter.most_common():
78             if term[0] == b'-1':
79                 continue
80             if term[-1] <= min_freq:
81                 break
82             onehot_features.append(term[0])
83             onehot_encoding.append((data[:, col] == term[0]).astype(float))
84         data[:, col] = '0'
85     onehot_encoding = np.array(onehot_encoding).T
86     data = np.hstack(
87         [np.array(data, dtype=float),
88          np.array(onehot_encoding)])
89
90     # Replace missing data with the mode value. We use the mode instead of
91     # the mean or median because this makes more sense for categorical
92     # features such as gender or cabin type, which are not ordered.
93     if fill_mode:
94         # TODO
95         pass
96
```

```
    return data, onehot_features


def evaluate(clf):
    print("Cross validation", cross_val_score(clf, X, y))
    if hasattr(clf, "decision_trees"):
        counter = Counter([t.tree_.feature[0] for t in clf.decision_trees])
        first_splits = [
            (features[term[0]], term[1]) for term in counter.most_common()
        ]
        print("First splits", first_splits)


if __name__ == "__main__":
    dataset = "titanic"
    # dataset = "spam"
    params = {
        "max_depth": 5,
        # "random_state": 6,
        "min_samples_leaf": 10,
    }
    N = 100

    if dataset == "titanic":
        # Load titanic data
        path_train = 'datasets/titanic/titanic_training.csv'
        data = genfromtxt(path_train, delimiter=',', dtype=None)
        path_test = 'datasets/titanic/titanic_testing_data.csv'
        test_data = genfromtxt(path_test, delimiter=',', dtype=None)
        y = data[1:, 0]  # label = survived
        class_names = ["Died", "Survived"]

        labeled_idx = np.where(y != b'')[0]
        y = np.array(y[labeled_idx], dtype=float).astype(int)
        print("\n\nPart (b): preprocessing the titanic dataset")
        X, onehot_features = preprocess(data[1:, 1:], onehot_cols=[1, 5, 7, 8])
        X = X[labeled_idx, :]
        Z, _ = preprocess(test_data[1:, :], onehot_cols=[1, 5, 7, 8])
        assert X.shape[1] == Z.shape[1]
        features = list(data[0, 1:]) + onehot_features

    elif dataset == "spam":
        features = [
            "pain", "private", "bank", "money", "drug", "spam", "prescription",
            "creative", "height", "featured", "differ", "width", "other",
            "energy", "business", "message", "volumes", "revision", "path",
            "meter", "memo", "planning", "pleased", "record", "out",
            "semicolon", "dollar", "sharp", "exclamation", "parenthesis",
            "square_bracket", "ampersand"
        ]
        assert len(features) == 32

```

```
        # Load spam data
        path_train = 'datasets/spam_data/spam_data.mat'
        data = scipy.io.loadmat(path_train)
        X = data['training_data']
        y = np.squeeze(data['training_labels'])
        Z = data['test_data']
        class_names = ["Ham", "Spam"]

    else:
        raise NotImplementedError("Dataset %s not handled" % dataset)

    print("Features", features)
    print("Train/test size", X.shape, Z.shape)

    # Decision Tree
    print("\n\nDecision Tree")
    dt = DecisionTreeClassifier(max_depth=3)
    dt.fit(X, y)

    # Visualize Decision Tree
    print("\n\nTree Structure")
    # Print using repr
    print(dt.__repr__())
    # Save tree to pdf
    graph_from_dot_data(dt.to_graphviz())[0].write_pdf("%s-basic-tree.pdf" % dataset)

    # Random Forest
    print("\n\nRandom Forest")
    rf = RandomForest(params, n=N, m=np.int_(np.sqrt(X.shape[1])))
    rf.fit(X, y)
    evaluate(rf)

    # Generate Test Predictions
    print("\n\nGenerate Test Predictions")
    pred = rf.predict(Z)
```

Listing 1: 随机森林模型接口

1. 请参考以上模板实现随机森林算法。你也可以选择不参考模板，自行实现，但是不允许使用任何现成的随机森林实现。不过你可以使用库中提供的单棵决策树实现（我们在模板代码中使用了 sklearn.tree .DecisionTreeClassifier）。如果使用模板代码，你主要需要实现随机森林继承的超类，即 bagged trees 的实现，它会基于不同的数据样本创建并训练多棵决策树。完成后，请在模板中补充上缺失的部分。(5 points)

```
from collections import Counter
from sklearn.model_selection import train_test_split
import numpy as np
import scipy.io
from scipy import stats
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import cross_val_score
```

```python
import pandas as pd
from pydot import graph_from_dot_data
import io
import random

random.seed(246810)
np.random.seed(246810)

eps = 1e-5  # a small number

class BaggedTrees(BaseEstimator, ClassifierMixin):
    def __init__(self, params=None, n=200):
        if params is None:
            params = {}
        self.params = params
        self.n = n
        self.decision_trees = [
            DecisionTreeClassifier(self.params, random_state=i)
            for i in range(self.n)
        ]

    def fit(self, X, y):
        self.trees_ = []
        n_samples = X.shape[0]
        for tree in self.decision_trees:
            indices = np.random.choice(n_samples, n_samples, replace=True)
            X_bootstrap = X[indices]
            y_bootstrap = y[indices]
            tree.fit(X_bootstrap, y_bootstrap)
            self.trees_.append(tree)
        return self

    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees_])
        y_pred, _ = stats.mode(predictions, axis=0)
        return y_pred.flatten()

class RandomForest(BaggedTrees):
    def __init__(self, params=None, n=200, m=1):
        if params is None:
            params = {}
        params['max_features'] = m
        self.m = m
        super().__init__(params=params, n=n)

    def fit(self, X, y):
        self.samples = []
        self.feature_subsets = []  # 保存每棵树使用的特征索引
        for tree in self.decision_trees:
            indices = np.random.choice(len(X), len(X), replace=True)
            X_sample = X[indices]
            y_sample = y[indices]
```

```
            features = np.random.choice(X.shape[1], self.m, replace=False)
            X_sample = X_sample[:, features]
            self.feature_subsets.append(features)
            model = tree.fit(X_sample, y_sample)
            self.samples.append((X_sample, y_sample))
        return self

    def predict(self, X):
        predictions = np.array([tree.predict(X[:, features]) for tree, features in zip(self.
    decision_trees, self.feature_subsets)])
        y_pred, _ = stats.mode(predictions, axis=0)
        return y_pred.flatten()

class BoostedRandomForest(RandomForest):
    # OPTIONAL
    def fit(self, X, y):
        # TODO
        pass

    def predict(self, X):
        # TODO
        pass

def preprocess(data, fill_mode=True, min_freq=10, onehot_cols=[]):
    # Temporarily assign -1 to missing data
    data[data == b''] = '-1'

    # Hash the columns (used for handling strings)
    onehot_encoding = []
    onehot_features = []
    for col in onehot_cols:
        counter = Counter(data[:, col])
        for term in counter.most_common():
            if term[0] == b'-1':
                continue
            if term[-1] <= min_freq:
                break
            onehot_features.append(f"col{col}_{term[0]}")
            onehot_encoding.append((data[:, col] == term[0]).astype(float))
        data[:, col] = '0'
    onehot_encoding = np.array(onehot_encoding).T
    data = np.hstack(
        [np.array(data, dtype=float),
         np.array(onehot_encoding)])

    # Replace missing data with the mode value. We use the mode instead of
    # the mean or median because this makes more sense for categorical
    # features such as gender or cabin type, which are not ordered.
    if fill_mode:
        for col in range(data.shape[1]):
            mode_value, _ = stats.mode(data[:, col])
            data[:, col] = np.where(data[:, col] == '-1', mode_value, data[:, col])
```

```
    return data, onehot_features

def evaluate(clf):
    print("Cross validation", cross_val_score(clf, X, y))
    if hasattr(clf, "decision_trees"):
        counter = Counter([t.tree_.feature[0] for t in clf.decision_trees])
        first_splits = [
            (features[term[0]], term[1]) for term in counter.most_common()
        ]
        print("First splits", first_splits)

def print_decision_path(tree, X, feature_names, class_names):
    node_indicator = tree.decision_path(X)
    leave_id = tree.apply(X)
    feature = tree.tree_.feature
    threshold = tree.tree_.threshold

    sample_id = 0
    node_index = node_indicator.indices[node_indicator.indptr[sample_id]:node_indicator.indptr[
    sample_id + 1]]

    decision_path = []
    for node_id in node_index:
        if leave_id[sample_id] == node_id:
            continue
        if feature[node_id] == -2:
            continue
        if X[sample_id, feature[node_id]] <= threshold[node_id]:
            threshold_sign = "<="
        else:
            threshold_sign = ">"
        decision_path.append(f"'{feature_names[feature[node_id]]}' {threshold_sign} {threshold[
    node_id]}")
    for i, decision in enumerate(decision_path):
        print(f"{i+1}. {decision}")
    predicted_class = tree.predict(X)[0]
    print(f"因此，该样本被分类为 '{class_names[int(predicted_class)]}'。")

def get_random_forest_root_stats(rf, feature_names):
    root_feature_indices = [tree.tree_.feature[0] for tree in rf.decision_trees]

    feature_counts = Counter(root_feature_indices)

    feature_counts_named = {}
    for feature_index, count in feature_counts.items():
        feature_name = feature_names[feature_index]
        feature_counts_named[feature_name] = count

    sorted_feature_counts = sorted(feature_counts_named.items(), key=lambda x: x[1], reverse=True
    )

    print("\n随机森林根节点常见分裂特征（按使用次数排序）:")
```

```
    for feature_name, count in sorted_feature_counts:
        print(f" - 特征 '{feature_name}' 被 {count} 棵树用作根节点分裂")

if __name__ == "__main__":
    #dataset = "titanic"
    dataset = "spam"
    params = {
        "max_depth": 5,
        # "random_state": 6,
        "min_samples_leaf": 10,
    }
    N = 100

    if dataset == "titanic":
        train_path = 'datasets/titanic/train.csv'
        test_path = 'datasets/titanic/test.csv'
        gender_path = 'datasets/titanic/gender_submission.csv'
        class_names = ["Died", "Survived"]
        train_titantic = pd.read_csv(train_path)
        test_titantic = pd.read_csv(test_path)
        gender = pd.read_csv(gender_path)

        y_train = train_titantic['Survived'].to_numpy()
        y_test = gender['Survived'].to_numpy()

        train_data = train_titantic.drop(columns=['PassengerId','Survived', 'Name']).copy()
        test_data = test_titantic.drop(columns=['PassengerId', 'Name']).copy()

        train_data = train_data.fillna('-1').astype(str).to_numpy()
        test_data = test_data.fillna('-1').astype(str).to_numpy()
        onehot_cols = [1,5,7,8]
        X_train, onehot_features = preprocess(train_data, onehot_cols=onehot_cols)
        X_test, _ = preprocess(test_data, onehot_cols=onehot_cols)

        assert X_train.shape[1] == X_test.shape[1]

        features = list(train_titantic.columns[2:])
        features.remove('Name')
        features=features+ onehot_features


    elif dataset == "spam":
        features = [
            "pain", "private", "bank", "money", "drug", "spam", "prescription",
            "creative", "height", "featured", "differ", "width", "other",
            "energy", "business", "message", "volumes", "revision", "path",
            "meter", "memo", "planning", "pleased", "record", "out",
            "semicolon", "dollar", "sharp", "exclamation", "parenthesis",
            "square_bracket", "ampersand"
        ]
        assert len(features) == 32

```

```
        # Load spam data
        path_train = 'datasets/spam_data/spam_data.mat'
        data = scipy.io.loadmat(path_train)
        print(data.keys())
        X = data['training_data']
        y = np.squeeze(data['training_labels'])
        Z = data['test_data']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        class_names = ["Ham", "Spam"]

    else:
        raise NotImplementedError("Dataset %s not handled" % dataset)

    print("Features", features)
    print("Train/test size", X_train.shape, X_test.shape)

    # Decision Tree
    print("\n\nDecision Tree")
    dt = DecisionTreeClassifier(max_depth=3)
    dt.fit(X_train, y_train)

    # Visualize Decision Tree
    print("\n\nTree Structure")
    # Print using repr
    print(dt.__repr__())
    # Save tree to pdf
    dot_data = io.StringIO()
    export_graphviz(dt, out_file=dot_data, feature_names=features, class_names=class_names)
    graph = graph_from_dot_data(dot_data.getvalue())[0]
    graph.write_pdf("%s-basic-tree.pdf" % dataset)

    # Random Forest
    print("\n\nRandom Forest")
    m1=max(np.int_(np.ceil(np.sqrt(X_train.shape[1]))),np.int_(np.ceil(X_train.shape[1]/2)))
    rf = RandomForest(params, n=N, m=m1)
    rf.fit(X_train, y_train)
    evaluate(rf)

    # Generate Test Predictions
    print("\n\nGenerate Test Predictions")
    if dataset == "spam":
        # 选择垃圾邮件和正常邮件的样本
        spam_sample = X_train[y_train == 1][0:1]  # 第一封垃圾邮件
        ham_sample = X_train[y_train == 0][0:1]   # 第一封正常邮件

        print("\n垃圾邮件的决策树分裂路径:")
        print_decision_path(dt, spam_sample, features, class_names)

        print("\n正常邮件的决策树分裂路径:")
        print_decision_path(dt, ham_sample, features, class_names)

    elif dataset == "titanic":
```

```
        # 选择幸存者和未幸存者的样本
        survived_sample = X_train[y_train == 1][0:1]
        not_survived_sample = X_train[y_train == 0][0:1]

        print("\n幸存者的决策树分裂路径:")
        print_decision_path(dt, survived_sample, features, class_names)

        print("\n未幸存者的决策树分裂路径:")
        print_decision_path(dt, not_survived_sample, features, class_names)

    # 获取随机森林中根节点使用的特征的统计信息
    get_random_forest_root_stats(rf, features)


    # 计算决策树的训练和测试准确率
    print("\n\n决策树准确率")
    train_pred_dt = dt.predict(X_train)  # 使用训练数据进行预测
    test_pred_dt = dt.predict(X_test)   # 使用测试数据进行预测
    train_acc_dt = accuracy_score(y_train, train_pred_dt)
    test_acc_dt = accuracy_score(y_test, test_pred_dt)
    print(f"决策树训练准确率: {train_acc_dt:.4f}")
    print(f"决策树测试准确率: {test_acc_dt:.4f}")

    # 计算随机森林的训练和测试准确率
    print("\n\n随机森林准确率")
    train_pred_rf = rf.predict(X_train)  # 使用训练数据进行预测
    test_pred_rf = rf.predict(X_test)   # 使用测试数据进行预测
    train_acc_rf = accuracy_score(y_train, train_pred_rf)
    test_acc_rf = accuracy_score(y_test, test_pred_rf)
    print(f"随机森林训练准确率: {train_acc_rf:.4f}")
    print(f"随机森林测试准确率: {test_acc_rf:.4f}")
```

Listing 2: 随机森林完整实现

2. 不需要长篇大论，每个问题用 1-2 句话回答即可：(5 points)

- 你是如何处理分类特征和缺失值的?
- 你是如何实现随机森林的?
- 你是否采用了特殊的方法加速训练?（回答“没有”也是可以的。）
- 还有什么特别棒的功能你实现了吗?（回答“没有”也是可以的。）

解:

1. 我们对指定的分类特征进行了独热编码（one-hot encoding），将类别转换为二进制特征。对于缺失值，首先用占位符'-1' 进行替换，然后在预处理过程中将其替换为每列的众数。
2. 我们通过训练多棵决策树来实现随机森林。对于每棵树，使用自助法（bootstrap）从原始数据中有放回地抽样生成训练集，并随机选择 m 个特征用于训练。预测时，通过所有树的多数投票来确定最终分类结果。

3. 没有

4. 我们实现了函数来获取单个数据点在决策树中的分裂路径，以及统计随机森林中最常见的根节点分裂特征，以帮助分析模型的决策过程。

3. 对于这两个数据集，请分别训练一棵决策树和一个随机森林，并报告它们的训练准确率和测试准确率。你需要报告 8 个数字（2 个数据集 × 2 个分类器 × 训练/测试）。(5 points)

解:

```
Optimal number of clusters: 4
Optimal number of clusters (revised): 4
Cluster Inertia (Sum of Squared Distances): 205.22514747675916
Customer Cluster Analysis with Genre Distribution:
                      Age  Annual Income (k$)  Spending Score (1-100)  Count
Cluster Genre
0       Female  32.545455           85.272727               80.590909     22
        Male    33.277778           87.111111               82.666667     18
1       Female  26.000000           39.529412               59.500000     34
        Male    24.608696           40.695652               61.478261     23
2       Female  51.405405           47.594595               40.567568     37
        Male    57.392857           47.857143               39.178571     28
3       Female  40.263158           87.105263               24.947368     19
        Male    38.473684           85.894737               14.210526     19
```

图 1: titanic

```
spam决策树准确率
决策树训练准确率: 0.7955
决策树测试准确率: 0.7971

spam随机森林准确率
随机森林训练准确率: 0.8204
随机森林测试准确率: 0.8106
```

图 2: spam

4. 决策树和随机森林的决策分析。(5 points)

- 对于决策树，选择来自每个类别（垃圾邮件和正常邮件）的一条数据点，列出决策树为对其分类所做的分裂（即，在哪个特征上以及该特征的哪个值上进行分裂）。以下是一个示例：

  1. (‘hot”) ≥ 2
  2. (‘thanks”) < 1
  3. (‘nigeria”) ≥ 3
  4. 因此这封邮件是垃圾邮件。
  1. (‘budget”) ≥ 2
  2. (‘spreadsheet”) ≥ 1
  3. 因此这封邮件是正常邮件。

- 对于随机森林，找出并列出树根节点处最常见的分裂。例如：

  1. (‘viagra”) ≥ 3 (20 trees)
  2. (‘thanks”) < 4 (15 trees)
  3. (‘nigeria”) ≥ 1 (5 trees)

解:

```
垃圾邮件的决策树分裂路径:
1. 'exclamation' > 0.5
2. 'ampersand' <= 0.5
3. 'meter' <= 0.5
因此，该样本被分类为 'Spam'。

正常邮件的决策树分裂路径:
1. 'exclamation' <= 0.5
2. 'meter' <= 0.5
3. 'parenthesis' > 0.5
因此，该样本被分类为 'Ham'。
```

图 3: 决策树为单个数据点分类时的分裂路径

```
随机森林根节点常见分裂特征（按使用次数排序）:
 - 特征 'prescription' 被 11 棵树用作根节点分裂
 - 特征 'spam' 被 9 棵树用作根节点分裂
 - 特征 'creative' 被 8 棵树用作根节点分裂
 - 特征 'energy' 被 7 棵树用作根节点分裂
 - 特征 'private' 被 7 棵树用作根节点分裂
 - 特征 'bank' 被 7 棵树用作根节点分裂
 - 特征 'message' 被 7 棵树用作根节点分裂
 - 特征 'featured' 被 6 棵树用作根节点分裂
 - 特征 'differ' 被 6 棵树用作根节点分裂
 - 特征 'pain' 被 6 棵树用作根节点分裂
 - 特征 'drug' 被 5 棵树用作根节点分裂
 - 特征 'money' 被 5 棵树用作根节点分裂
 - 特征 'width' 被 5 棵树用作根节点分裂
 - 特征 'business' 被 5 棵树用作根节点分裂
 - 特征 'height' 被 3 棵树用作根节点分裂
 - 特征 'other' 被 3 棵树用作根节点分裂
```

图 4: 随机森林根节点最常见分裂

三. (20 points) 聚类理论

聚类是一种无监督学习任务，其核心是根据数据样本之间的相似性（通常由距离度量定义）将数据分成若干个簇。常用聚类算法如 k-均值和 DBSCAN 都需要特定的距离度量和超参数设置。以下问题围绕距离度量、目标函数、以及超参数设置展开：

1. 在聚类算法中，距离度量是衡量样本间相似性的基础，选择合适的距离度量对聚类效果有显著影响（5 points）。

(a) 给定样本 $x = (x_1, x_2, \ldots, x_d)$ 和 $y = (y_1, y_2, \ldots, y_d)$，分别写出以下三种常见距离度量的数学公式：

- 欧几里得距离
- 曼哈顿距离
- 余弦相似度（将其转换为距离形式）

(b) 在以下场景中，分析哪种距离更适合使用，并简要说明原因：

- 场景 1：高维稀疏特征向量（如文本数据的 TF-IDF 表示）
- 场景 2：二维几何分布数据（如图像中的空间点分布）

解:

(1) 三种常见距离度量的数学公式：

欧几里得距离：

$$d_{\text{Euclidean}}(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

曼哈顿距离：

$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$

余弦相似度转换为距离形式：

$$d_{\text{cosine}}(x, y) = 1 - \text{cosine\_similarity}(x, y) = 1 - \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \sqrt{\sum_{i=1}^{d} y_i^2}}$$

(2)

在高维稀疏特征向量中，**余弦距离**更适合，因为高维空间中，欧几里得距离可能失去分辨力，样本之间的距离可能趋于一致，从而难以有效区分样本。TF-IDF 等特征往往是稀疏向量，向量的模长差异较大，而余弦距离主要关注向量之间的方向相似性，与特征的稀疏性无关，因此效果更优。

对于二维几何分布数据，**欧几里得距离**更适合，因为二维几何数据直观地表示点在空间中的位置，欧几里得距离能够直接反映两点之间的实际空间距离。曼哈顿距离虽然也可以用，但在规则网格或道路网络等特定场景中更常见，而非一般几何分布数据。余弦距离对数据方向敏感，但在低维几何分布数据中意义较小，因此不适用。

2. k-均值聚类的目标函数与迭代过程, k-均值聚类的目标是最小化以下目标函数（10 points）:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

其中，C_i 表示第 i 个聚类簇，μ_i 为第 i 个簇的中心。

(a) 推导在分配样本点到最近的簇中心时，为什么目标函数 J 会减少。

(b) 推导为什么更新簇中心为簇内样本点的均值时，目标函数 J 会减少。

(c) k-均值的超参数 k 对结果有何影响?

- 如果 k 设置过大或过小，分别可能会导致什么问题?
- 提出一种确定 k 的方法，并解释其原理。

解:

(1) k-均值聚类的目标是最小化以下目标函数:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

在 k-均值算法的分配步骤中，对于每个样本点 x，将其分配到最近的簇中心对应的簇 C_i，即:

$$i = \arg\min_j \|x - \mu_j\|^2$$

因为目标函数 J 是所有簇内点到簇中心的平方距离总和，重新分配样本点到最近簇中心后，每个点的距离非增，目标函数总和 J 也会减小。

(2) 在更新步骤中，将每个簇的中心 μ_i 更新为该簇内所有样本点的均值，即:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

目标函数的每一项 $\sum_{x \in C_i} \|x - \mu_i\|^2$ 表示簇 C_i 的样本点到簇中心的平方距离总和。

设 $f(\mu_i) = \sum_{x \in C_i} \|x - \mu_i\|^2$, 这是关于 μ_i 的凸函数。求导得:

$$\frac{\partial f(\mu_i)}{\partial \mu_i} = -2 \sum_{x \in C_i} (x - \mu_i)$$

令导数为 0, 得:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

即簇中心更新为簇内样本点的均值时，使得 $f(\mu_i)$ 最小。

因此，更新簇中心为簇内样本点均值后，簇内样本点到簇中心的平方距离总和减小，即目标函数 J 减少。

3. 密度聚类（如 DBSCAN）依赖以下两个超参数（5 points）：

- ε（邻域半径）：定义一个点的邻域范围。
- MinPts（核心点的最小邻域点数）：定义核心点的密度阈值。

(a) 核心点、边界点和噪声点的定义是什么？它们在聚类中的作用分别是什么？

(b) 如果 ε 和 MinPts 设置不当，可能会出现哪些问题？

- ε 过大或过小
- MinPts 过大或过小

(c) 为什么 DBSCAN 不需要预先指定聚类簇的数量 k？这对实际应用有什么优势？

解:

(1)

- 核心点：若一个点的 ε 邻域内包含至少 MinPts 个点，则该点为核心点。核心点位于高密度区域的内部，是聚类的核心组成部分。
- 边界点：若一个点的 ε 邻域内的点数小于 MinPts，但该点位于某个核心点的 ε 邻域内，则该点为边界点。边界点位于簇的边缘，连接核心点和噪声点。
- 噪声点：既不是核心点，也不在任何核心点的 ε 邻域内的点被视为噪声点。噪声点不属于任何簇，通常被视为离群点。

在聚类过程中，核心点是簇的中心，边界点扩展簇的边界，而噪声点则被排除在簇之外。

(2)

- ε 过大或过小：
  - ε 过大：会导致更多的点被包含在同一个 ε 邻域内，可能将不同的簇合并为一个簇，降低聚类的分辨率。
  - ε 过小：会导致大多数点的 ε 邻域内点数不足 MinPts，使得许多核心点无法形成，导致大部分点被标记为噪声，无法有效识别簇。
- MinPts 过大或过小：
  - MinPts 过大：需要更多的点才能形成核心点，可能导致只有密度非常高的区域才能形成簇，其他区域的点被标记为噪声。
  - MinPts 过小：容易将密度较低的区域也识别为簇，可能导致噪声点被错误地包含在簇中，降低聚类的准确性。

(3)

DBSCAN 通过密度来定义簇，而不是预先指定簇的数量。它根据数据点的分布，自适应地发现密度足够高的区域，并将这些区域定义为簇。因此，无需预先指定簇的数量 k。

在实际应用中的优势如下：

- 无需人工设定簇的数量，减少了人为干预，适用于对数据分布未知的情况。
- 能够识别任意形状的簇，而不仅限于球形簇，适用于复杂数据分布。
- 能够有效识别并排除噪声点，提高聚类的鲁棒性。

四. (30 points) 聚类实战

使用商场客户数据集 (Mall Customer Dataset) 完成客户分群分析。该数据集包含客户的年龄 (Age)、年收入 (Annual Income) 和消费积分 (Spending Score) 等特征。你需要通过实现和优化聚类算法来完成客户画像分析。数据随作业提供。

为方便起见，每小题我们提供了部分初始代码，其中包括预处理步骤和部分功能的实现。你可以自由选择使用或不使用这些代码来完成实现。

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# 加载数据
def load_mall_data():
    df = pd.read_csv("Mall_Customers.csv")
    X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
    # 数据标准化
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, X, df
```

Listing 3: 数据加载

1. 在不借助外部实现的情况下，手动实现 KMeans 方法 (4 points)，在数据集上进行聚类，可视化聚类结果 (3 points)，并解决下列问题 (3 points)：

- 如何使用肘部法则确定合适的 k 值，绘图说明
- 简单分析每个客户群的特征
- 计算和分析簇内平方和 (inertia)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D

# 数据加载部分代码
def load_mall_data():
    df = pd.read_csv(r"E:\大三上\机器学习\作业\作业4\4\Mall_Customers.csv")
    X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
    # 数据标准化
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, X, df


# KMeans算法实现
class KMeans:
    def __init__(self, n_clusters=3, max_iters=100):
        self.n_clusters = n_clusters
```

```
        self.max_iters = max_iters
        self.centroids = None
        self.labels = None
        self.inertia_ = None  # 簇内平方和

    def fit(self, X):
        np.random.seed(42)
        n_samples, n_features = X.shape
        # 随机初始化聚类中心
        random_idx = np.random.choice(n_samples, self.n_clusters, replace=False)
        self.centroids = X[random_idx]

        for _ in range(self.max_iters):
            # 计算每个样本到聚类中心的距离
            distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
            self.labels = np.argmin(distances, axis=1)

            # 计算新的聚类中心
            new_centroids = np.array([X[self.labels == k].mean(axis=0) for k in range(self.
    n_clusters)])

            # 检查是否收敛
            if np.all(self.centroids == new_centroids):
                break
            self.centroids = new_centroids

        # 计算簇内平方和
        self.inertia_ = np.sum([np.sum((X[self.labels == k] - self.centroids[k]) ** 2) for k in
    range(self.n_clusters)])
        return self

    def predict(self, X):
        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
        return np.argmin(distances, axis=1)

# 绘制聚类结果的散点图
def plot_clusters(X, labels, centroids=None):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # 绘制数据点
    for k in range(np.max(labels) + 1):
        cluster_points = X[labels == k]
        ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2], label=f"
    Cluster {k+1}")

    # 绘制聚类中心
    if centroids is not None:
        ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], color='black', marker='x',
    s=150, label='Centroids')

    ax.set_title("3D Cluster Visualization")
```

```
    ax.set_xlabel("Age")
    ax.set_ylabel("Annual Income (k$)")
    ax.set_zlabel("Spending Score (1-100)")
    ax.legend()
    plt.show()

# 使用肘部法则确定最佳k值
def elbow_method(X, max_k=10):
    inertias = []
    for k in range(1, max_k + 1):
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        inertias.append(kmeans.inertia_)
        print(f"k = {k}, Inertia = {kmeans.inertia_:.2f}")

    # 可视化肘部法则
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, max_k + 1), inertias, marker='o')
    plt.title("Elbow Method")
    plt.xlabel("Number of Clusters (k)")
    plt.ylabel("Inertia")
    plt.show()

    # 找到最佳 k 值
    optimal_k = find_elbow_point(inertias)
    print(f"Optimal number of clusters: {optimal_k}")
    return inertias, optimal_k

# 使用几何方法（线段垂距）寻找肘部点
def find_elbow_point(inertias):
    x = np.arange(1, len(inertias) + 1)
    y = np.array(inertias)

    # 直线两端点
    line_start = np.array([x[0], y[0]])
    line_end = np.array([x[-1], y[-1]])

    # 计算垂直距离
    distances = np.abs((line_end[1] - line_start[1]) * x -
                       (line_end[0] - line_start[0]) * y +
                       line_end[0] * line_start[1] -
                       line_end[1] * line_start[0]) / np.sqrt(
        (line_end[1] - line_start[1])**2 + (line_end[0] - line_start[0])**2
    )
    optimal_k = np.argmax(distances) + 1
    return optimal_k

if __name__ == "__main__":
    X_scaled, X, df = load_mall_data()

    # 使用肘部法则
    inertias, optimal_k = elbow_method(X_scaled)
```

```
121
122     # 根据肘部法则选择的最佳 k 值
123     print(f"Optimal number of clusters (revised): {optimal_k}")
124
125     # 进行聚类
126     kmeans = KMeans(n_clusters=optimal_k)
127     kmeans.fit(X_scaled)
128
129     # 输出聚类结果与簇内平方和
130     print("Cluster Inertia (Sum of Squared Distances):", kmeans.inertia_)
131
132     # 可视化聚类结果
133     plot_clusters(X_scaled, kmeans.labels, kmeans.centroids)
134
135     # 分析聚类结果
136     df['Cluster'] = kmeans.labels
137     grouped = df.groupby(['Cluster', 'Genre']).agg({
138         'Age': 'mean',
139         'Annual Income (k$)': 'mean',
140         'Spending Score (1-100)': 'mean',
141         'Genre': 'count'
142     }).rename(columns={'Genre': 'Count'})
143     print("Customer Cluster Analysis with Genre Distribution:")
144     print(grouped)
145
146     # 可视化按簇划分的分布
147     genre_distribution = df.groupby(['Cluster', 'Genre']).size().unstack()
148     genre_distribution.plot(kind='bar', stacked=True, figsize=(10, 6))
149     plt.title("Genre Distribution Across Clusters")
150     plt.xlabel("Cluster")
151     plt.ylabel("Count")
152     plt.legend(title="Genre")
153     plt.show()
```

Listing 4: Kmeans 部分实现

解:

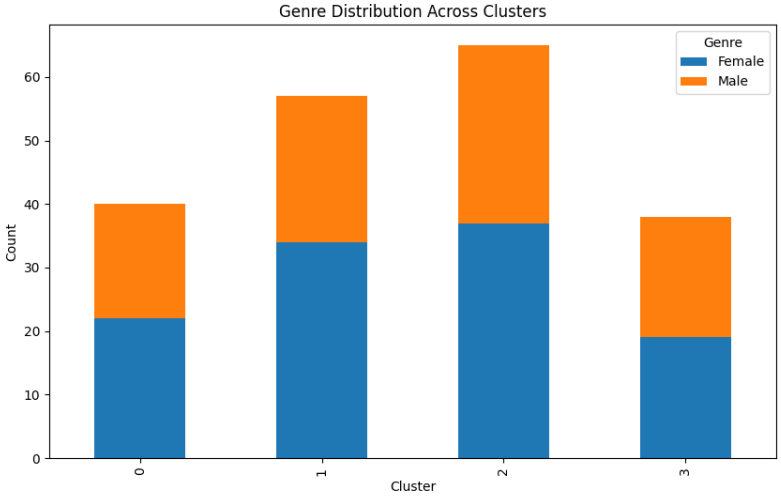

图 7: 性别聚类分析

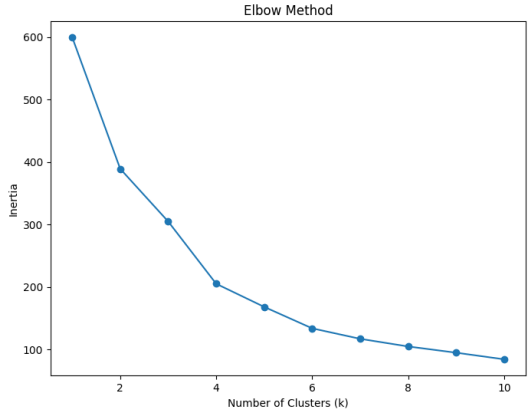

图 5: 肘部法则绘图

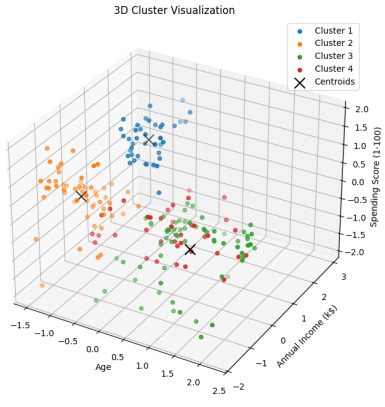

图 6: 聚类分析绘图

```
Optimal number of clusters: 4
Optimal number of clusters (revised): 4
Cluster Inertia (Sum of Squared Distances): 205.22514747675916
Customer Cluster Analysis with Genre Distribution:
                      Age  Annual Income (k$)  Spending Score (1-100)  Count
Cluster Genre
0       Female  32.545455           85.272727               80.590909     22
        Male    33.277778           87.111111               82.666667     18
1       Female  26.000000           39.529412               59.500000     34
        Male    24.608696           40.695652               61.478261     23
2       Female  51.405405           47.594595               40.567568     37
        Male    57.392857           47.857143               39.178571     28
3       Female  40.263158           87.105263               24.947368     19
        Male    38.473684           85.894737               14.210526     19
```

图 8: 输出结果

结合肘部法则绘图和代码输出结果来看，在 $k = 4$ 时达到了“肘点”。此时，簇内平方和为 205.23。

根据代码输出结果，每个客户群的特征如下：

- 簇 0：
  - 性别分布：女性略多于男性（22:18）。
  - 平均年龄：客户群较年轻，平均年龄约为 32-33 岁。
  - 年收入：高收入群体，平均年收入约为 85-87 千美元。
  - 消费评分：消费评分很高，平均值为 80-83。
  - 特征总结：高收入、高消费的年轻客户群。
- 簇 1：
  - 性别分布：女性占优势（34:23）。
  - 平均年龄：客户群年龄偏小，平均年龄约为 24-26 岁。
  - 年收入：中低收入群体，平均年收入约为 39-41 千美元。
  - 消费评分：消费评分中等偏高，平均值为 59-61。
  - 特征总结：年轻的中低收入客户，具备一定消费潜力，但可能受收入限制。
- 簇 2：
  - 性别分布：男性稍多于女性（28:37）。
  - 平均年龄：年龄较大，女性约 51 岁，男性约 57 岁。
  - 年收入：中等收入群体，年收入约为 47-48 千美元。
  - 消费评分：消费评分偏低，平均值为 39-41。

  - 特征总结：中等收入的中老年客户群，消费意愿较低。

- 簇 3：
  - 性别分布：男女比例相等（19:19）。
  - 平均年龄：中年客户，女性约 40 岁，男性约 38 岁。
  - 年收入：高收入群体，年收入约为 85-87 千美元。
  - 消费评分：消费评分很低，女性约为 25，男性约为 14。
  - 特征总结：高收入但低消费的中年客户群。

2. 基于问题一的实现，我们发现随机初始化可能导致结果不稳定。请实现和分析以下改进：

- 实现 K-means++ 初始化方法 (4 分)
- 实现聚类评估指标 (3 分)：轮廓系数 (Silhouette Score)、聚类稳定性评估
- 对比分析 (3 分)
    1. 比较随机初始化和 K-means++ 的结果差异，可以通过可视化聚类图进行对比
    2. 比较两种方法的稳定性
    3. 分析初始化对收敛速度的影响

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D

# 数据加载部分代码
def load_mall_data():
    df = pd.read_csv(r"E:\大三上\机器学习\作业\作业4\4\Mall_Customers.csv")
    X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
    # 数据标准化
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, X, df

class KMeans:
    def __init__(self, n_clusters=3, max_iters=100):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.centroids = None
        self.labels = None
        self.inertia_ = None  # 簇内平方和

    def fit(self, X):
        np.random.seed(None)
        n_samples, n_features = X.shape
        # 随机初始化聚类中心
        random_idx = np.random.choice(n_samples, self.n_clusters, replace=False)
        self.centroids = X[random_idx]

        for _ in range(self.max_iters):
            # 计算每个样本到聚类中心的距离
            distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
            self.labels = np.argmin(distances, axis=1)

            # 计算新的聚类中心
            new_centroids = np.array([X[self.labels == k].mean(axis=0) for k in range(self.
    n_clusters)])

```

```
            # 检查是否收敛
            if np.all(self.centroids == new_centroids):
                break
            self.centroids = new_centroids

        # 计算簇内平方和
        self.inertia_ = np.sum([np.sum((X[self.labels == k] - self.centroids[k]) ** 2) for k in range(self.n_clusters)])
        return self

    def predict(self, X):
        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
        return np.argmin(distances, axis=1)

# KMeans++算法实现
class KMeansPlusPlus:
    def __init__(self, n_clusters=3, max_iters=100):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.centroids = None
        self.labels = None
        self.inertia_ = None  # 簇内平方和

    def _kmeans_plus_plus_init(self, X):
        n_samples, _ = X.shape
        centroids = [X[np.random.randint(0, n_samples)]]

        for _ in range(1, self.n_clusters):
            distances = np.min([np.linalg.norm(X - c, axis=1) for c in centroids], axis=0)
            probabilities = distances**2 / np.sum(distances**2)
            cumulative_probs = np.cumsum(probabilities)
            r = np.random.rand()
            new_centroid_idx = np.searchsorted(cumulative_probs, r)
            centroids.append(X[new_centroid_idx])

        return np.array(centroids)

    def fit(self, X):
        n_samples, n_features = X.shape
        self.centroids = self._kmeans_plus_plus_init(X)

        for _ in range(self.max_iters):
            distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
            self.labels = np.argmin(distances, axis=1)
            new_centroids = np.array([X[self.labels == k].mean(axis=0) for k in range(self.n_clusters)])
            if np.all(self.centroids == new_centroids):
                break

            self.centroids = new_centroids

        self.inertia_ = np.sum([np.sum((X[self.labels == k] - self.centroids[k])**2) for k in
```

```
    range(self.n_clusters)])

        return self

# 计算轮廓系数
def compute_silhouette_score(X, labels):
    return silhouette_score(X, labels)

# 评估聚类结果的稳定性
def compute_cluster_stability(X, k, n_runs=10):
    labels_list = []
    for _ in range(n_runs):
        kmeans = KMeansPlusPlus(n_clusters=k)
        kmeans.fit(X)
        labels_list.append(kmeans.labels)

    # 计算一致性
    stability = np.mean([np.sum(labels_list[i] == labels_list[j]) / len(X)
                         for i in range(n_runs) for j in range(i + 1, n_runs)])

    return stability

# 绘制聚类结果的散点图
def plot_clusters(X, labels, centroids=None):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # 绘制数据点
    for k in range(np.max(labels) + 1):
        cluster_points = X[labels == k]
        ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2], label=f"
    Cluster {k+1}")

    # 绘制聚类中心
    if centroids is not None:
        ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], color='black', marker='x',
    s=150, label='Centroids')

    ax.set_title("3D Cluster Visualization")
    ax.set_xlabel("Age")
    ax.set_ylabel("Annual Income (k$)")
    ax.set_zlabel("Spending Score (1-100)")
    ax.legend()
    plt.show()

# 使用肘部法则确定最佳k值
def elbow_method(X, max_k=10):
    inertias = []
    for k in range(1, max_k + 1):
        kmeans = KMeansPlusPlus(n_clusters=k)
        kmeans.fit(X)
        inertias.append(kmeans.inertia_)
```

```
        print(f"k = {k}, Inertia = {kmeans.inertia_:.2f}")

    # 可视化肘部法则
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, max_k + 1), inertias, marker='o')
    plt.title("Elbow Method")
    plt.xlabel("Number of Clusters (k)")
    plt.ylabel("Inertia")
    plt.show()

    # 找到最佳 k 值
    optimal_k = find_elbow_point(inertias)
    print(f"Optimal number of clusters: {optimal_k}")
    return inertias, optimal_k

# 使用几何方法（线段垂距）寻找肘部点
def find_elbow_point(inertias):
    x = np.arange(1, len(inertias) + 1)
    y = np.array(inertias)

    # 直线两端点
    line_start = np.array([x[0], y[0]])
    line_end = np.array([x[-1], y[-1]])

    # 计算垂直距离
    distances = np.abs((line_end[1] - line_start[1]) * x -
                       (line_end[0] - line_start[0]) * y +
                       line_end[0] * line_start[1] -
                       line_end[1] * line_start[0]) / np.sqrt(
        (line_end[1] - line_start[1])**2 + (line_end[0] - line_start[0])**2
    )
    optimal_k = np.argmax(distances) + 1
    return optimal_k


if __name__ == "__main__":
    X_scaled, X, df = load_mall_data()
    # 使用肘部法则
    inertias, optimal_k = elbow_method(X_scaled)

    # 根据肘部法则选择的最佳 k 值
    print(f"Optimal number of clusters (revised): {optimal_k}")

    kmeans_pp = KMeansPlusPlus(n_clusters=optimal_k)
    kmeans_pp.fit(X_scaled)

    silhouette_pp = compute_silhouette_score(X_scaled, kmeans_pp.labels)
    stability_pp = compute_cluster_stability(X_scaled, optimal_k)

    print(f"K-means++ Silhouette Score: {silhouette_pp:.4f}")
    print(f"K-means++ Stability Score: {stability_pp:.4f}")

```

```
191
192     # 可视化K-means++结果
193     df['Cluster'] = kmeans_pp.labels
194     plot_clusters(X_scaled, kmeans_pp.labels, kmeans_pp.centroids)
```

Listing 5: Kmeans++ 部分实现

解:

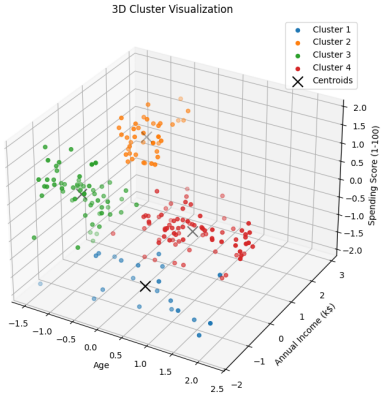

图 9: K-means++ 聚类分析

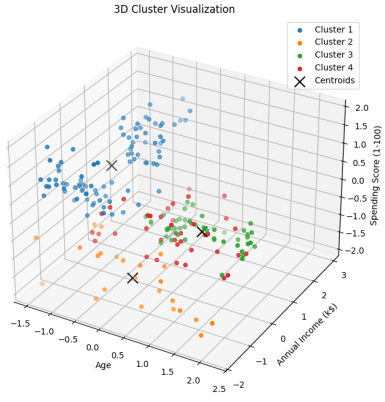

图 10: K-means 聚类分析

```
k = 1, Inertia = 600.00
k = 2, Inertia = 389.39
k = 3, Inertia = 297.03
k = 4, Inertia = 205.23
k = 5, Inertia = 168.25
k = 6, Inertia = 134.35
k = 7, Inertia = 124.12
k = 8, Inertia = 109.37
k = 9, Inertia = 99.30
k = 10, Inertia = 99.72
Optimal number of clusters: 4
Optimal number of clusters (revised): 4
K-means++ Silhouette Score: 0.4040
K-means++ Stability Score: 0.2807
```

图 11: K-means++ 输出结果

```
k = 1, Inertia = 600.00
k = 2, Inertia = 389.39
k = 3, Inertia = 297.03
k = 4, Inertia = 205.23
k = 5, Inertia = 168.25
k = 6, Inertia = 133.87
k = 7, Inertia = 132.94
k = 8, Inertia = 106.64
k = 9, Inertia = 102.08
k = 10, Inertia = 98.47
Optimal number of clusters: 4
Optimal number of clusters (revised): 4
K-means Silhouette Score: 0.3515
K-means Stability Score: 0.2244
```

图 12: K-means 输出结果

K-means++ 的轮廓系数为 0.4048，高于 K-means 的 0.3515；K-means++ 的稳定性得分为 0.2807，优于 K-means 的 0.2244。

因为 K-means++ 使用数据分布感知的初始化方式，使得初始聚类中心更接近实际聚类中心，从而提高了同簇内样本的紧密度，并增加了簇与簇之间的分离度，与此同时，K-means++ 减少了运行之间的随机性，每次运行的结果更加接近。

在聚类图中，采用 K-means++ 初始化方法的簇中心较为均匀，样本点分布比 K-means 的更加紧凑且分离度良好。

3. 在实际业务中，不同特征的重要性可能不同，且某些客户群可能需要大小相近。请实现带权重和大小约束的改进版本：

- 实现带约束的聚类算法，需要支持特征加权和簇大小约束 (4 points)
- 在以下两个场景下重新进行实验：收入特征权重加倍，限制每个客户群至少包含 20% 的客户，绘制聚类结果 (6 points)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import pandas as pd

# 数据加载部分代码
def load_mall_data():
    df = pd.read_csv(r"E:\大三上\机器学习\作业\作业4\4\Mall_Customers.csv")
    X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
    # 数据标准化
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, X, df

# KMeans++算法实现
class KMeansPlusPlus:
    def __init__(self, n_clusters=3, max_iters=100):
        self.n_clusters = n_clusters
        self.max_iters = max_iters
        self.centroids = None
        self.labels = None
        self.inertia_ = None  # 簇内平方和

    def _kmeans_plus_plus_init(self, X):
        n_samples, _ = X.shape
        centroids = [X[np.random.randint(0, n_samples)]]

        for _ in range(1, self.n_clusters):
            distances = np.min([np.linalg.norm(X - c, axis=1) for c in centroids], axis=0)
            probabilities = distances**2 / np.sum(distances**2)
            cumulative_probs = np.cumsum(probabilities)
            r = np.random.rand()
            new_centroid_idx = np.searchsorted(cumulative_probs, r)
            centroids.append(X[new_centroid_idx])

        return np.array(centroids)

    def fit(self, X):
```

```
        n_samples, n_features = X.shape
        self.centroids = self._kmeans_plus_plus_init(X)

        for _ in range(self.max_iters):
            distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
            self.labels = np.argmin(distances, axis=1)
            new_centroids = np.array([X[self.labels == k].mean(axis=0) for k in range(self.
    n_clusters)])
            if np.all(self.centroids == new_centroids):
                break

            self.centroids = new_centroids

        self.inertia_ = np.sum([np.sum((X[self.labels == k] - self.centroids[k])**2) for k in
    range(self.n_clusters)])

        return self

class ConstrainedKMeans(KMeansPlusPlus):
    def __init__(self, n_clusters=3, max_iters=100, weights=None, size_constraints=None):
        super().__init__(n_clusters, max_iters)
        self.weights = weights
        self.size_constraints = size_constraints

    def _weighted_distance(self, X, centroids):
        weighted_X = X * self.weights
        weighted_centroids = centroids * self.weights
        return np.linalg.norm(weighted_X[:, np.newaxis] - weighted_centroids, axis=2)

    def _reassign_clusters(self, X, labels, distances):
        cluster_sizes = np.bincount(labels, minlength=self.n_clusters)
        for cluster in range(self.n_clusters):
            while cluster_sizes[cluster] < self.size_constraints[0]:  # 小于最小限制
                # 从其他簇中挑选样本重新分配
                for donor_cluster in range(self.n_clusters):
                    if cluster_sizes[donor_cluster] > self.size_constraints[0]:
                        # 找到距离当前簇最近的样本并重新分配
                        donor_idx = np.where(labels == donor_cluster)[0]
                        distances_to_current = distances[donor_idx, cluster]
                        closest_idx = donor_idx[np.argmin(distances_to_current)]
                        labels[closest_idx] = cluster
                        cluster_sizes = np.bincount(labels, minlength=self.n_clusters)
                        break
                else:
                    raise ValueError("无法满足最小簇大小限制，可能是由于数据分布问题")
        return labels

    def fit(self, X):
        n_samples, n_features = X.shape
        if self.weights is None:
            self.weights = np.ones(n_features)
        self.centroids = self._kmeans_plus_plus_init(X)
```

```
90
91          for _ in range(self.max_iters):
92              distances = self._weighted_distance(X, self.centroids)
93              labels = np.argmin(distances, axis=1)
94              if self.size_constraints is not None:
95                  labels = self._reassign_clusters(X, labels, distances)
96              new_centroids = np.array([X[labels == k].mean(axis=0) for k in range(self.n_clusters)
    ])
97              if np.all(self.centroids == new_centroids):
98                  break
99
100             self.centroids = new_centroids
101
102         self.labels = labels
103         self.inertia_ = np.sum([np.sum((X[labels == k] - self.centroids[k])**2) for k in range(
    self.n_clusters)])
104         return self
105
106 # 绘制聚类结果的散点图
107 def plot_clusters(X, labels, centroids=None):
108     fig = plt.figure(figsize=(10, 8))
109     ax = fig.add_subplot(111, projection='3d')
110
111     # 绘制数据点
112     for k in range(np.max(labels) + 1):
113         cluster_points = X[labels == k]
114         ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2], label=f"
    Cluster {k+1}")
115
116     # 绘制聚类中心
117     if centroids is not None:
118         ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], color='black', marker='x',
    s=150, label='Centroids')
119
120     ax.set_title("3D Cluster Visualization")
121     ax.set_xlabel("Age")
122     ax.set_ylabel("Annual Income (k$)")
123     ax.set_zlabel("Spending Score (1-100)")
124     ax.legend()
125     plt.show()
126
127 # 使用肘部法则确定最佳k值
128 def elbow_method(X, max_k=10):
129     inertias = []
130     for k in range(1, max_k + 1):
131         kmeans = KMeansPlusPlus(n_clusters=k)
132         kmeans.fit(X)
133         inertias.append(kmeans.inertia_)
134         print(f"k = {k}, Inertia = {kmeans.inertia_:.2f}")
135
136     # 找到最佳 k 值
137     optimal_k = find_elbow_point(inertias)
```

```
    print(f"Optimal number of clusters: {optimal_k}")
    return inertias, optimal_k

# 使用几何方法（线段垂距）寻找肘部点
def find_elbow_point(inertias):
    x = np.arange(1, len(inertias) + 1)
    y = np.array(inertias)

    # 直线两端点
    line_start = np.array([x[0], y[0]])
    line_end = np.array([x[-1], y[-1]])

    # 计算垂直距离
    distances = np.abs((line_end[1] - line_start[1]) * x -
                       (line_end[0] - line_start[0]) * y +
                       line_end[0] * line_start[1] -
                       line_end[1] * line_start[0]) / np.sqrt(
        (line_end[1] - line_start[1])**2 + (line_end[0] - line_start[0])**2
    )
    optimal_k = np.argmax(distances) + 1
    return optimal_k


if __name__ == "__main__":
    X_scaled, X, df = load_mall_data()

    # 使用肘部法则
    inertias, optimal_k = elbow_method(X_scaled)

    # 根据肘部法则选择的最佳 k 值
    print(f"Optimal number of clusters (revised): {optimal_k}")

    # 场景1: 收入特征权重加倍
    weights = np.array([1, 2, 1])  # Age:1, Income:2, Spending:1
    constrained_kmeans_1 = ConstrainedKMeans(n_clusters=optimal_k, weights=weights)
    constrained_kmeans_1.fit(X_scaled)

    print("Scenario 1: Income Feature Weighted Double")
    plot_clusters(X_scaled, constrained_kmeans_1.labels, constrained_kmeans_1.centroids)

    # 分析聚类结果
    df['Cluster'] = constrained_kmeans_1.labels
    grouped = df.groupby(['Cluster', 'Genre']).agg({
        'Age': 'mean',
        'Annual Income (k$)': 'mean',
        'Spending Score (1-100)': 'mean',
        'Genre': 'count'
    }).rename(columns={'Genre': 'Count'})
    print("Customer Cluster Analysis with Genre Distribution:")
    print(grouped)

    # 场景2: 每个客户群至少包含20%的客户
```

```
    min_size = int(len(X) * 0.2)
    max_size = len(X)  # 无上限
    constrained_kmeans_2 = ConstrainedKMeans(n_clusters=optimal_k, size_constraints=(min_size,
    max_size))
    constrained_kmeans_2.fit(X_scaled)

    print("Scenario 2: Minimum 20% of Customers per Cluster")
    plot_clusters(X_scaled, constrained_kmeans_2.labels, constrained_kmeans_2.centroids)
    # 分析聚类结果
    df['Cluster'] = constrained_kmeans_2.labels
    grouped = df.groupby(['Cluster', 'Genre']).agg({
        'Age': 'mean',
        'Annual Income (k$)': 'mean',
        'Spending Score (1-100)': 'mean',
        'Genre': 'count'
    }).rename(columns={'Genre': 'Count'})
    print("Customer Cluster Analysis with Genre Distribution:")
    print(grouped)
```

Listing 6: 带约束的聚类算法部分实现

解:

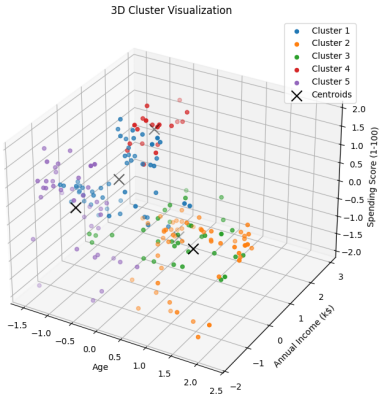

图 13: 场景一聚类分析

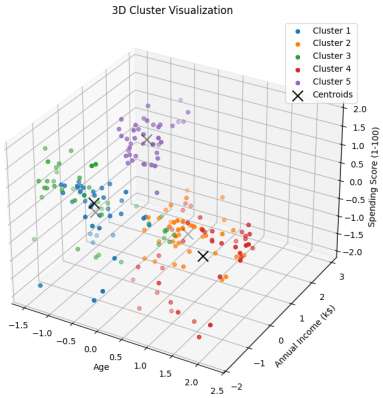

图 14: 场景二聚类分析

```
Optimal number of clusters: 5
Optimal number of clusters (revised): 5
Scenario 1: Income Feature Weighted Double
Customer Cluster Analysis with Genre Distribution:
                      Age  Annual Income (k$)  Spending Score (1-100)  Count
Cluster Genre
0       Female  29.200000           72.828571               64.857143     35
        Male    30.956522           74.000000               71.782609     23
1       Female  26.950000           31.500000               69.550000     20
        Male    23.785714           30.785714               69.357143     14
2       Female  41.538462           26.538462               20.692308     13
        Male    52.000000           25.875000               17.250000      8
3       Female  43.000000           95.611111               31.277778     18
        Male    37.047619           91.666667               21.904762     21
4       Female  53.538462           53.500000               49.153846     26
        Male    57.454545           55.045455               47.681818     22
Scenario 2: Minimum 20% of Customers per Cluster
Customer Cluster Analysis with Genre Distribution:
                      Age  Annual Income (k$)  Spending Score (1-100)  Count
Cluster Genre
0       Female  24.666667           37.571429               70.666667     21
        Male    22.368421           41.473684               63.315789     19
1       Female  56.789474           56.105263               49.736842     19
        Male    61.047619           51.142857               41.904762     21
2       Female  38.000000           84.523810               26.904762     21
        Male    38.473684           85.894737               14.210526     19
3       Female  39.862069           38.965517               34.620690     29
        Male    42.363636           37.636364               38.909091     11
4       Female  32.545455           85.272727               80.590909     22
        Male    33.277778           87.111111               82.666667     18
```

图 15: 输出结果

场景 1：收入特征权重加倍

簇 0：年轻，高收入，高消费

簇 1：年轻，低收入，高消费

簇 2：年长，低收入，低消费

簇 3：中年，高收入，低消费

簇 4：年长，中等收入，中等消费

收入特征的主导性明显，各簇的年收入区分度较高。

场景 2：每个客户群至少包含 20% 的客户

簇 0：年轻，低收入，高消费

簇 1：年长，中等收入，中等消费

簇 2：中年，高收入，低消费

簇 3：中年，中等收入，低消费

簇 4：年轻，高收入，高消费

每个簇都包含 40 个客户。