电子科技大学计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 数据结构与算法

电子科技大学教务处制表

电 子 科 技 大 学

实 验 报 告

学生姓名：陶浩轩　　学 号：**2023080902011**　　指导教师：陈端兵

一、 实验室名称：

学知三组团 10 栋 139

二、 实验项目名称：

基于决策树的分类与随机森林

三、 实验原理：

决策树是一种常用的分类算法，其原理是利用信息增益或信息增益比等指标选择最优特征进行划分，将数据集逐步划分为不同的子集，直到每个子集都属于同一类别或无法继续划分。ID3 算法是决策树的一种实现方式，它使用信息增益作为特征选择的依据。随机森林则是由多个决策树构成的集成学习模型，通过随机选择特征和样本数据构建多棵决策树，并对它们的预测结果进行投票表决，从而提高分类的准确率和泛化能力。

四、 实验目的：

理解决策树和随机森林的基本原理和实现方式，掌握决策树和随机森林在分类中的应用。

五、 实验内容：

使用 ID3 算法构建决策树与随机森林，对鸢尾花数据集进行分类，并评估其准确率。

六、 实验器材（设备、元器件）：

Window10，Python 3.12.6，NumPy 2.1.3

七、实验步骤：

1. 构建 DecisionTree 类，设置最大深度 max_depth，特征标签 feature_labels，左右子树 left 和 right，分割特征索引 split_idx，分割阈值 thresh，到达叶节点的数据 data，叶节点的预测结果 pred，以及随机特征子集的大小 m

```
def __init__(self, max_depth = 3, feature_labels = None, m = None):
    self.max_depth = max_depth       # max depth of the tree
    self.features = feature_labels   # names of features
    self.left = None
    self.right = None
    self.split_idx = None    # decide which feature to split
    self.thresh = None       # for splitting
    self.data = None         # feature values reach the leaf
    self.pred = None         # result
    self.m = m    # size of the randomly selected feature subset
```

2. 使用公式计算信息熵，基于给定的特征和阈值计算信息增益

```
@staticmethod
def calculate_entropy(y):
    probabilities = []

    for class_label in np.unique(y):
        count = len(y[np.where(y == class_label)])
        probabilities.append(float(count / len(y)))

    H_S = -1 * sum([pi * np.log2(pi) for pi in probabilities])
    return H_S
```

```
@staticmethod
def information_gain(X, y, thresh):
    H_S = DecisionTree.calculate_entropy(y)
    # get left and right subset labels after splitting
    S_l_y = y[np.where(X < thresh)]
    S_r_y = y[np.where(X >= thresh)]

    H_S_l = DecisionTree.calculate_entropy(S_l_y)
    H_S_r = DecisionTree.calculate_entropy(S_r_y)

    # H_S after splitting
    H_after = (len(S_l_y) * H_S_l + len(S_r_y) * H_S_r) / len(y)

    return H_S - H_after
```

3. 编写 split_test 方法，根据给定的特征索引和阈值进行数据分割；split 方法调用前者分割数据集，并返回分割后的特征和标签

```python
def split_test(self, X, idx, thresh):
    # idxx is a ndarray containing rows idx
    idx0 = np.where(X[:, idx] < thresh)[0]
    idx1 = np.where(X[:, idx] >= thresh)[0]
    # X0, X1 are matrixs
    X0, X1 = X[idx0, :], X[idx1, :]
    return X0, idx0, X1, idx1


def split(self, X, y, idx, thresh):
    X0, idx0, X1, idx1 = self.split_test(X, idx, thresh)
    y0, y1 = y[idx0], y[idx1]
    return X0, y0, X1, y1
```

4. 实现 predict 方法，递归地遍历树，根据特征值和阈值决定向左还是向右子树进行预测，直到达到叶节点。用于对数据的分类进线预测

```python
def predict(self, X, verbose = False):
    if self.max_depth == 0:
        # return prediction
        return self.pred * np.ones(X.shape[0])
    else:
        if (verbose and X.shape[0] != 0):
            print(
                "feature", self.features[self.split_idx],
                "value", X[0, self.split_idx],
                ">/<", self.thresh
            )

        X0, idx0, X1, idx1 = self.split_test(X, self.split_idx, self.thresh)

        yhat = np.zeros(X.shape[0])
        yhat[idx0] = self.left.predict(X0, verbose=verbose)
        yhat[idx1] = self.right.predict(X1, verbose=verbose)

        return yhat
```

5. 完成训练决策树模型的方法 fit，递归地选择最优特征和阈值进行分割，直到达到最大深度或数据无法进一步分割。如果设置了随机特征子集的大小 m，则随机选择特征进行分割

```python
def fit(self, X, y):
    if self.max_depth == 0:
        # reach max depth
        self.data = X
        self.labels = y
        # find the most frequent in y
        self.pred = stats.mode(y).mode[0]
        return self

    gains = []
    original_X = X

    if self.m:
        # if m is set, randomly chose m features
        attribute_bag = np.random.choice(list(range(len(self.features))), size = self.m, replace = False)
        X = original_X[:, attribute_bag]
    else:
        attribute_bag = None
        X = original_X

    thresh = np.array([
        # generate threshold of 10 linear intervals for each feature
        np.linspace(np.min(X[:, i]) + eps, np.max(X[:, i] - eps), num=10)
        # X.shape[1]: number of columns
        for i in range(X.shape[1])
    ])

    for i in range(X.shape[1]):
        gains.append([self.information_gain(X[:, i], y, t) for t in thresh[i, :]])

    gains = np.nan_to_num(np.array(gains))

    # find split_idx and thresh_idx with max inform_gain
    self.split_idx, thresh_idx = np.unravel_index(np.argmax(gains), gains.shape)
    self.thresh = thresh[self.split_idx, thresh_idx]

    if self.m:
        self.split_idx = attribute_bag[self.split_idx]

    X0, y0, X1, y1 = self.split(original_X, y, self.split_idx, self.thresh)

    if X0.size > 0 and X1.size > 0:
        self.left = DecisionTree(
            max_depth = self.max_depth - 1,
            feature_labels = self.features,
            m = self.m
        )
        self.left.fit(X0, y0)
        self.right = DecisionTree(
            max_depth = self.max_depth - 1,
            feature_labels = self.features,
            m = self.m
        )
        self.right.fit(X1, y1)
    else:
        self.max_depth = 0
        self.data = original_X
        self.labels = y
        self.pred = stats.mode(y).mode[0]

    return self
```

6. 在实现决策树后，编写随机森林类 RandomForest。该类初始化方法设置了森林中树的数量 n，样本大小 sample_size，并为每棵树创建一个 DecisionTree 实例

```
def __init__(self, max_depth=3, n=25, features=None, sample_size=None):
    self.n = n
    self.sample_size = sample_size
    self.decision_trees = [
        DecisionTree(max_depth=max_depth, feature_labels=features)
        for i in range(n)
    ]
```

7. 实现训练随机森林模型的方法 fit，具体是对数据集有放回的随机抽样，并用抽样的数据集训练森林中的树

```
def fit(self, X, y):
    assert self.sample_size < len(y), "Sample size must be less than test size"

    connect = np.concatenate((X, y.reshape(-1,1)), axis=1)

    for tree in self.decision_trees:
        samples = np.random.choice(list(range(len(connect))), size=self.sample_size, replace=True)

        train_data = connect[samples, :]
        train_X = train_data[:, :-1]
        train_y = train_data[:, -1:]

        tree.fit(train_X, train_y)
```

8. predict 方法，用森林中的每一棵树得出的预测结果，“投票”得出最终的预测

```
def predict(self, X):
    predictions = []
    for tree in self.decision_trees:
        predictions.append(tree.predict(X))

    total_pred = np.vstack(predictions);

    mode_predictions = []
    for i in range(total_pred.shape[1]):
        mode_prediction = stats.mode(total_pred[:, i])[0]
        mode_predictions.append(mode_prediction)

    return np.array(mode_predictions)
```

9. 最后编写主函数，加载鸢尾花数据集，并划分训练集与测试机。随后创建随机森林实例，训练模型。最后使用测试集进行预测和评估

```python
def main():
    iris = load_iris()
    X = iris.data
    y = iris.target

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    rf = RandomForest(
        max_depth=3,
        n=25,
        features=iris.feature_names,
        sample_size=100
    )

    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)

    # print(y_pred)
    # print(y_test)

    accuracy = np.mean(y_pred == y_test)
    print(f"Model accuracy on test set: {accuracy:.2f}")


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

七、 实验数据及结果分析：

本次实验采用了鸢尾花数据集，包含了 150 个样本，都属于鸢尾属下的 3 个亚属，分别是山鸢尾、变色鸢尾和维吉尼亚鸢尾。每个样本都包含 4 项特征，即花萼和花瓣的长度和宽度，它们可用于样本的定量分析

在 main 函数中，我将这 150 个样本中的 80% 用于训练随机森林，20% 用于测试。并使用 np.mean() 来求得预测的准确率，得到了理想的结果

```
[21:40] Shell master ≡ ~1
F:\Projects\Data Structrue & Algorithm\UESTC-DataStructure\proj2
❯ & D:/Python312/python.exe "f:/Projects/Data Structrue & Algorithm/UESTC-DataStructure/proj2/DecisionTree.py"
Model accuracy on test set: 1.00
```

接下来，我又用相同的数据集，添加了对决策树准确性的测试。

```python
def testDecisionTree(X_train, X_test, y_train, y_test, feature_names):
    dt = DecisionTree(max_depth=7, feature_labels=feature_names)
    dt.fit(X_train, y_train)
    y_pred = dt.predict(X_test)
    accuracy = np.mean(y_pred == y_test)
    print(f"DecisionTree Model accuracy on test set: {accuracy:.2f}")
```

运行，得到决策树和随机森林预测的准确性。可以看到随机森林可以提供更准确的预测

```
[23:22] Shell master ≡ ~2
F:\Projects\Data Structrue & Algorithm\UESTC-DataStructure\proj2
❯ & D:/Python312/python.exe "f:/Projects/Data Structrue & Algorithm/UESTC-DataStructure/proj2/DecisionTree.py"
DecisionTree Model accuracy on test set: 0.97
RandomForest Model accuracy on test set: 1.00
```

八、 实验结论：

通过本次实验，我们可以得出：决策树和随机森林是解决分类问题的两个可靠的方法。其中随机森林可以提供比决策树更加准确的预测。

九、 总结及心得体会：

1. Python 在数据科学与机器学习方面提供了许多非常强大的库，比如 NumPy，SciPy，Sklearn 等等。在相关的工作中，比起其它语言，可以在保持程序高效运行的同时，提供更加便捷与快速的开发
2. 由于 Python 不像 C，Java 那样对数据类型进行限制，在写 Python 代码的时候应当更加注重方法与成员变量规范的编写，并在关键代码处使用 assert 实现 fast fail 方便 debug

十一、对本实验过程及方法、手段的改进建议：

1. 使用可视化工具。如 Matplotlib 和 Pandas，我们可以更加直观的观察决策树的构成，将“黑盒”变为“玻璃盒”
2. 可以尝试由一个节点，根据多个阈值范围分叉出多个节点。比较与二叉树的实现的准确性
3. 使用更多的数据集以及不同的划分方式，测试不同参数下的两个模型的准确性

报告评分：

指导教师签字：

附录：源代码

```python
import numpy as np
from scipy import stats
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import graphviz

eps = 1e-5      # a small number

class DecisionTree:
    def __init__(self, max_depth = 3, feature_labels = None, m = None):
        self.max_depth = max_depth      # max depth of the tree
        self.features = feature_labels   # names of features
        self.left = None
        self.right = None
        self.split_idx = None   # decide which feature to split
        self.thresh = None      # for splitting
        self.data = None        # feature values reach the leaf
        self.pred = None        # result
        self.m = m    # size of the randomly selected feature subset


    @staticmethod
    def calculate_entropy(y):
        probabilities = []

        for class_label in np.unique(y):
            count = len(y[np.where(y == class_label)])
            probabilities.append(float(count / len(y)))

        H_S = -1 * sum([pi * np.log2(pi) for pi in probabilities])
        return H_S


    @staticmethod
    def information_gain(X, y, thresh):
        H_S = DecisionTree.calculate_entropy(y)
        # get left and right subset labels after splitting
        S_l_y = y[np.where(X < thresh)]
        S_r_y = y[np.where(X >= thresh)]

        H_S_l = DecisionTree.calculate_entropy(S_l_y)
        H_S_r = DecisionTree.calculate_entropy(S_r_y)

        # H_S after splitting
        H_after = (len(S_l_y) * H_S_l + len(S_r_y) * H_S_r) / len(y)

        return H_S - H_after
```

```python
    def split_test(self, X, idx, thresh):
        # idxx is a ndarray containing rows idx
        idx0 = np.where(X[:, idx] < thresh)[0]
        idx1 = np.where(X[:, idx] >= thresh)[0]
        # X0, X1 are matrixs
        X0, X1 = X[idx0, :], X[idx1, :]
        return X0, idx0, X1, idx1


    def split(self, X, y, idx, thresh):
        X0, idx0, X1, idx1 = self.split_test(X, idx, thresh)
        y0, y1 = y[idx0], y[idx1]
        return X0, y0, X1, y1


    def fit(self, X, y):
        if self.max_depth == 0:
            # reach max depth
            self.data = X
            self.labels = y
            # find the most frequent in y
            self.pred = stats.mode(y).mode
            return self

        gains = []
        original_X = X

        if self.m:
            # if m is set, randomly chose m features
            attribute_bag = np.random.choice(list(range(len(self.features))), size =
self.m, replace = False)
            X = original_X[:, attribute_bag]
        else:
            attribute_bag = None
            X = original_X

        thresh = np.array([
            # generate threshold of 10 linear intervals for each feature
            np.linspace(np.min(X[:, i]) + eps, np.max(X[:, i] - eps), num=10)
            for i in range(X.shape[1])
        ])

        for i in range(X.shape[1]):
            gains.append([self.information_gain(X[:, i], y, t) for t in thresh[i, :]])

        gains = np.nan_to_num(np.array(gains))

        # find split_idx and thresh_idx with max inform_gain
        self.split_idx, thresh_idx = np.unravel_index(np.argmax(gains), gains.shape)
        self.thresh = thresh[self.split_idx, thresh_idx]
```

```
        if self.m:
            self.split_idx = attribute_bag[self.split_idx]

        X0, y0, X1, y1 = self.split(original_X, y, self.split_idx, self.thresh)

        if X0.size > 0 and X1.size > 0:
            self.left = DecisionTree(
                max_depth = self.max_depth - 1,
                feature_labels = self.features,
                m = self.m
            )
            self.left.fit(X0, y0)
            self.right = DecisionTree(
                max_depth = self.max_depth - 1,
                feature_labels = self.features,
                m = self.m
            )
            self.right.fit(X1, y1)
        else:
            self.max_depth = 0
            self.data = original_X
            self.labels = y
            self.pred = stats.mode(y).mode

        return self


    def predict(self, X, verbose = False):
        if self.max_depth == 0:
            # return prediction
            return self.pred * np.ones(X.shape[0])
        else:
            if (verbose and X.shape[0] != 0):
                print(
                    "feature", self.features[self.split_idx],
                    "value", X[0, self.split_idx],
                    ">/<", self.thresh
                )

            X0, idx0, X1, idx1 = self.split_test(X, self.split_idx, self.thresh)

            yhat = np.zeros(X.shape[0])
            yhat[idx0] = self.left.predict(X0, verbose=verbose)
            yhat[idx1] = self.right.predict(X1, verbose=verbose)

            return yhat



class RandomForest:
    def __init__(self, max_depth=3, n=25, features=None, sample_size=None):
        self.n = n
        self.sample_size = sample_size
```

```
        self.decision_trees = [
            DecisionTree(max_depth=max_depth, feature_labels=features)
            for i in range(n)
        ]


    def fit(self, X, y):
        assert self.sample_size < len(y), "Sample size must be less than test size"

        connect = np.concatenate((X, y.reshape(-1,1)), axis=1)

        for tree in self.decision_trees:
            samples = np.random.choice(list(range(len(connect))),
size=self.sample_size, replace=True)

            train_data = connect[samples, :]
            train_X = train_data[:, :-1]
            train_y = train_data[:, -1:]

            tree.fit(train_X, train_y)


    def predict(self, X):
        predictions = []
        for tree in self.decision_trees:
            predictions.append(tree.predict(X))

        total_pred = np.vstack(predictions);

        mode_predictions = []
        for i in range(total_pred.shape[1]):
            mode_prediction = stats.mode(total_pred[:, i])[0]
            mode_predictions.append(mode_prediction)

        return np.array(mode_predictions)



def testRandomForest(X_train, X_test, y_train, y_test, feature_names):
    rf = RandomForest(
        max_depth=3,
        n=25,
        features=feature_names,
        sample_size=100
    )

    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)

    # print(y_pred)
    # print(y_test)

    accuracy = np.mean(y_pred == y_test)
```

```python
    print(f"RandomForest Model accuracy on test set: {accuracy:.2f}")


def testDecisionTree(X_train, X_test, y_train, y_test, feature_names):
    dt = DecisionTree(
        max_depth=7,
        feature_labels=feature_names,
    )

    dt.fit(X_train, y_train)
    y_pred = dt.predict(X_test)

    # print(y_pred)
    # print(y_test)

    accuracy = np.mean(y_pred == y_test)
    print(f"DecisionTree Model accuracy on test set: {accuracy:.2f}")


def main():
    iris = load_iris()
    X = iris.data
    y = iris.target

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    testDecisionTree(X_train, X_test, y_train, y_test, iris.feature_names)
    testRandomForest(X_train, X_test, y_train, y_test, iris.feature_names)


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