《机器学习基础》实验报告

实验题目	**决策树算法实践**		
实验时间	2024/05/19	**实验地点**	DS3401
实验成绩		**实验性质**	**☐验证性　☐设计性　☐综合性**

教师评价：

☐算法/实验过程正确；　☐源程序/实验内容提交　☐程序结构/实验步骤合理；

☐实验结果正确；　☐语法、语义正确；　☐报告规范；

其他：

评价教师签名：

一、实验目的

掌握决策树回归算法原理。

二、实验项目内容

1. 理解并**描述**决策树分类、回归算法原理。

2. **编程**实践，将决策树分类、回归算法分别应用于合适的数据集(如鸢尾花、波士顿房价预测、UCI 数据集、Kaggle 数据集)，要求算法至少用于两个数据集(分类 2 个，回归 2 个)。

三、实验过程或算法（源程序）

1.实验原理

（1）决策树分类算法原理

决策树分类算法是一种基于树形结构进行分类的无监督学习算法，它通过对特征的逐步判断和分割，从而对数据进行分类。其核心思想是通过构建一棵树状结构，使得每个内部节点代表一个属性测试，每个分支代表一个属性值，叶节点代表一种分类结果。在分类过程中，从根节点开始，根据样本的特征值依次向下遍历直至叶节点，最终得到样本的分类结果。

其主要步骤为：

1）选择最佳的划分属性：通过计算各个属性的信息增益或者基尼指数等指标，选择最佳的划分属性作为当前节点的划分标准

2）划分数据集：根据选择的划分属性，将数据集划分成多个子集，每个子集对应于该属性的一个取值

3）递归构建决策树：对每个子集递归地构建决策树，直至满足停止条件（如达到最大深度、样本数量小于阈值等）

4）剪枝处理：对构建好的决策树进行剪枝操作，以防止过拟合

（2）决策树回归算法原理

决策树回归算法是一种基于树形结构进行回归分析的无监督学习算法，它通过对特征的逐步判断和分割，从而对数据进行回归预测。其核心思想与决策树分类算法类似，不同之处在于叶节点代表的是一个数值结果，而非分类结果。

其主要步骤与决策树分类算法类似，包括选择最佳的划分属性、划分数据集、递归构建决策树以及剪枝处理等。

与分类问题采用的准确率指标不同的是，在回归问题中，决策树的划分标准通常采用均方误差等指标，以最小化预测值与真实值之间的差异。在预测过程中，根据样本的特征值依次向下遍历决策树，最终得到样本的回归预测结果。

2.实验代码

（1）mydecision.py

```
from collections import Counter
from math import log2
import numpy as np

class Node:
        def __init__(self, feature_index=None, threshold=None,
value=None, left=None, right=None, label=None):
            self.feature_index = feature_index  # 划分特征索引
          self.threshold = threshold        # 划分阈值（对于连续特征）
            self.value = value                  # 结点值
            self.left = left                    # 左子树
            self.right = right                  # 右子树
            self.label = label                  # 叶子节点标签


class DecisionClassifier:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.root = None

    def fit(self, X, y):
        # 计算熵值
        def calculate_entropy(labels):
            c = Counter(labels)
           ent = sum(-(c[i] / len(labels)) * log2(c[i] / len(labels))
for i in c)
            return ent

        #将数据按照 threshold 分割
        def split_data(X, y, feature_index, threshold):
            left_indices = [i for i in range(len(X)) if
X[i][feature_index] <= threshold]
            right_indices = [i for i in range(len(X)) if
X[i][feature_index] > threshold]
            return (X[left_indices], y[left_indices]),
(X[right_indices], y[right_indices])

        def best_split(X, y):
            best_gain = 0
            best_feature_index = None
            best_threshold = None
            original_entropy = calculate_entropy(y)

            for feature_index in range(X.shape[1]):
                thresholds = sorted(set(X[:, feature_index]))
                for threshold in thresholds:
                   left_X, left_y = split_data(X, y, feature_index,
threshold)[0]
                  right_X, right_y = split_data(X, y, feature_index,
threshold)[1]

                    if len(left_y) == 0 or len(right_y) == 0:
                        continue

                    weighted_ent = (len(left_y) / len(y)) *
calculate_entropy(left_y) +  (len(right_y) / len(y)) *
calculate_entropy(right_y)
                    gain = original_entropy - weighted_ent

                    if gain > best_gain:
                        best_gain = gain
                        best_feature_index = feature_index
                        best_threshold = threshold

            return best_gain, best_feature_index, best_threshold

        def build_tree(X, y, depth=0):
           if len(y) == 0 or (self.max_depth is not None and depth >=
self.max_depth):
               return Node(label=Counter(y).most_common(1)[0][0])

            if len(set(y)) == 1:
                return Node(label=y[0])

            gain, feature_index, threshold = best_split(X, y)
            if gain <= 0:
               return Node(label=Counter(y).most_common(1)[0][0])

            left_X, left_y = split_data(X, y, feature_index,
threshold)[0]
            right_X, right_y = split_data(X, y, feature_index,
threshold)[1]

            root = Node(feature_index=feature_index,
threshold=threshold)
            root.left = build_tree(left_X, left_y, depth + 1)
            root.right = build_tree(right_X, right_y, depth + 1)

            return root

        self.root = build_tree(X, y)

    def predict(self, X):
        def traverse_tree(node, x):
            if node.label is not None:
                return node.label

            if x[node.feature_index] <= node.threshold:
                return traverse_tree(node.left, x)
            else:
                return traverse_tree(node.right, x)
        predictions = []
        for x in X:
            predictions.append(traverse_tree(self.root, x))
        return np.array(predictions)
```

```
class DecisionRegressor:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.root = None

    def fit(self, X, y):
        def _find_best_split(X, y, depth):
            best_mse = float('inf')
            best_feature_index = None
            best_threshold = None

            for feature_index in range(X.shape[1]):
                thresholds = sorted(set(X[:, feature_index]))
                #thresholds = np.unique(X[:, feature_index])
                for threshold in thresholds:
                    # 根据阈值分割数据
                    left_indices = [i for i in range(len(X)) if
X[i][feature_index] <= threshold]
                    right_indices = [i for i in range(len(X)) if
X[i][feature_index] > threshold]
                  #left_indices = X[:, feature_index] <= threshold
                    #right_indices = ~left_indices
                    left_y = y[left_indices]
                    right_y = y[right_indices]
```

```
                    if len(left_y) == 0 or len(right_y) == 0:
                        continue
```

```
                    # 计算MSE
                   mse_left = np.mean((left_y - np.mean(left_y))**2)
                 mse_right = np.mean((right_y - np.mean(right_y))**2)
                   mse = len(left_y) / len(y) * mse_left + len(right_y)
/ len(y) * mse_right

                    # 如果MSE 更小，更新最佳分割
                    if mse < best_mse:
                        best_mse = mse
                        best_feature_index = feature_index
                        best_threshold = threshold

            return best_feature_index, best_threshold, best_mse

        def build_tree(X, y, depth=0):
            # 终止条件：达到最大深度或所有样本值相同
            if depth == self.max_depth or len(np.unique(y)) == 1:
                return Node(value=np.mean(y))

            # 寻找最佳分割
         feature_index, threshold, _ = _find_best_split(X, y, depth)

            # 如果没有合适的分割，返回叶子节点
            if feature_index is None:
                return Node(value=np.mean(y))

            # 分割数据集
            left_indices = [i for i in range(len(X)) if 
X[i][feature_index] <= threshold]
            right_indices = [i for i in range(len(X)) if 
X[i][feature_index] > threshold]
            #left_indices = X[:, feature_index] <= threshold
            #right_indices = ~left_indices
            left_X = X[left_indices]
            right_X = X[right_indices]
            left_y = y[left_indices]
            right_y = y[right_indices]

            # 递归构建左右子树
            left_child = build_tree(left_X, left_y, depth + 1)
            right_child = build_tree(right_X, right_y, depth + 1)

            # 返回节点
            return Node(feature_index=feature_index, 
threshold=threshold, left=left_child, right=right_child)

        self.root = build_tree(X, y)

    def predict(self, X):
        def traverse_tree(node, x):
            if node.value is not None:
                return node.value

            if x[node.feature_index] <= node.threshold:
                return traverse_tree(node.left, x)
            else:
                return traverse_tree(node.right, x)

        return np.array([traverse_tree(self.root, x) for x in X])
```
（2）Expriment3.py（主程序）

```
from sklearn.datasets import load_iris
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
import mydecision
import pandas as pd
import numpy as np

 #分类任务
```

```
def taskfenlei(X_train, X_test, y_train, y_test):
    clf = mydecision.DecisionClassifier()
    clf.fit(X_train, y_train)  #训练
    y_pred = clf.predict(X_test)  #测试集
    accuracy = accuracy_score(y_test, y_pred)
    print(f'这是决策树分类,精准度 Accuracy:\n {accuracy}')
```

```
#回归任务
def taskhuigui(X_train, X_test, y_train, y_test):
    reg = mydecision.DecisionRegressor()
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)
    '''
    valid_indices = ~np.isnan(y_test) & ~np.isnan(y_pred)
    y_true_valid = y_test[valid_indices]
    y_pred_valid = y_pred[valid_indices]
    rmse = np.sqrt(np.mean((y_true_valid - y_pred_valid) ** 2))
    '''
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    print(f'这是决策树回归，均方根误差 RMSE:\n {rmse}')
```

```
def classify():
    # 加载鸢尾花数据集
    iris = load_iris()
    X = iris.data
    y = iris.target
    # 划分数据集
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5, random_state=42)
    print('数据集：鸢尾花')
    taskfenlei(X_train, X_test, y_train, y_test)

    # 加载乳腺癌数据集
    breast_cancer = load_breast_cancer()
    X = breast_cancer.data
    y = breast_cancer.target
    y = y.astype(int)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    print('数据集：乳腺癌')
    taskfenlei(X_train, X_test, y_train, y_test)
```

```
def regressor():
    # 加州房价数据集：数据处理
    datafile = 'D:\zjw\demo\machine learning\housing.data'
    data = np.fromfile(datafile, sep=' ')
    feature_names = [ 'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
'AGE','DIS',
                    'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV' ]
    feature_num = len(feature_names)
   data = data.reshape([data.shape[0] // feature_num, feature_num])
    ratio = 0.8 #此处的含义就是将80%的数据用于训练
    offset = int(data.shape[0] * ratio)
```

```python
    maximums, minimums= data.max(axis=0),data.min(axis=0)
    # 对数据进行归一化处理
    for i in range(feature_num):
        data[:, i] = (data[:, i] - minimums[i]) / (maximums[i] - 
minimums[i])
    test_data = data[offset:]
    x = data[:, :-1]
    y = data[:, -1:]
    X_train, X_test, y_train, y_test = train_test_split(x, y, 
test_size=0.2, random_state=42)
    print('数据集：加州房价数据集')
    taskhuigui(X_train, X_test, y_train, y_test)
```

```python
    file_path = 'D:\zjw\demo\machine learning\mpg.csv'
    mpg_data = pd.read_csv(file_path)
    # 选择需要的列（最后一列是名字不需要）
    selected_columns = ['mpg', 'cylinders', 'displacement', 
'horsepower', 'weight', 'acceleration', 'model_year', 'origin']
    mpg_selected_data = mpg_data[selected_columns]
    # 独热编码转换三分类数据
    mpg_one_hot = pd.get_dummies(mpg_selected_data, 
columns=['origin'])
    # 显示处理后的数据
    x_init = mpg_one_hot.drop('mpg',axis=1)
    y_init = mpg_one_hot['mpg']
    x=x_init.values
    y=y_init.values
    X_train, X_test, y_train, y_test = train_test_split(x, y, 
test_size=0.2, random_state=42)
    print('数据集：mpg，燃油价格数据集')
    taskhuigui(X_train, X_test, y_train, y_test)
```

```python
if __name__ == '__main__':
    classify()
    regressor()
```

四、实验结果及分析

1.实验结果

```
数据集：鸢尾花
这是决策树分类,精准度 Accuracy:
 0.8933333333333333
数据集：乳腺癌
这是决策树分类,精准度 Accuracy:
 0.9298245614035088
数据集：加州房价数据集
这是决策树回归，均方根误差 RMSE:
 0.09350176730407676
数据集：mpg，燃油价格数据集
这是决策树回归，均方根误差 RMSE:
 3.4754496112014053
```

2.实验结果分析

（1）决策树分类结果分析：

对于鸢尾花数据集和乳腺癌数据集，决策树分类模型的精确度均大于89%。这意味着模型能够准确地对鸢尾花进行分类，但仍有一些分类错误。对于，决策树分类模型的精确度为约 92.98%。这表明模型在对乳腺癌进行分类时具有相当高的准确性。

综合来看，这些分类结果显示决策树在不同数据集上的分类效果较好，尤其是在乳腺癌数据集上表现出色。

（2）决策树回归结果分析：

对于加州房价数据集，决策树回归模型的均方根误差（RMSE）为0.0935，小于 0.1，说明决策树回归模型在加州房价数据集上十分适用，具有较高的预测准确性。

对于 mpg 燃油价格数据集，决策树回归模型的均方根误差为约3.4754。数值相对较大，说明决策树回归不适用于燃油数据集，可能需要进一步的特征工程或模型调优。

总体而言，决策树模型在分类和回归任务上都取得了不错的成绩，说明了本实验的成功性。