# 试 验 报 告

## 实验名称： 智能机器人大作业

## 课程名称： 智能机器人原理

院系： 人工智能学院

班级： 2021 级本科 1 班

姓名： 刘元昊

学号： 21311223

日期： 2024 年 7 月 10 日

**中山大学**

1. 问题理解

**题目：**设计一个场景，包括起点，终点，障碍物，采用2种及以上的规划方法实现规划过程，采用2种及以上的横向控制方法实现路径跟踪过程，要求实现模拟移动机器人运动的过程，机器人本体的运动控制模型不限。

**要求：**

1. 植入代码语言不限，要求可运行，有readme，制作代码运行讲解视频文件；
2. 简单介绍不同方法的原理，优缺点；
3. 试验分析验证、在规划路径质量，规划效率2个方面对比不同规划方法；
4. 在跟踪误差方面对比不同的跟踪控制方法；
5. 要求有效果图、数据分析图、数据分析表；
6. 可以去网上寻找开源代码，但不允许同学之间抄袭，否则被确认后，2个人都是0分处理。

**问题理解：**

1. 实验环境

本次实验是在标准的Python编程环境中完成，具体依赖以下软件包和相关版本：

Python==3.11

panda==2.1.3

matplotlib==3.8.2

numpy==1.26.

1. 算法设计

1. 数据清洗

在开始模型构建之前，首先进行了数据的预处理，以确保数据质量和可用性。使用`pandas`库从文本文件中读取数据，并指定列名，以便能够针对特定的特征进行操作。

加载数据，删除缺失值，删除不相关的列: 根据数据分析的需求，删除了`'native-country'`和`'fnlwgt'`这两列，因为这些特征对于预测目标变量`'income'`不具有直接的相关性。

def dataclean(data\_path):

    columns = [

        "age", "workclass", "fnlwgt", "education", "education-num", "marital-status",

        "occupation", "relationship", "race", "sex", "capital-gain", "capital-loss",

        "hours-per-week", "native-country", "income"

    ]

    # 读取具有适当列名的数据集

    data = pd.read\_csv(data\_path, names=columns, sep=',\s\*', engine='python')

    # 删除由'?'表示的缺失值的行

    cleaned\_data = data.replace('?', pd.NA).dropna()

    # 删除'native-country'列

    cleaned\_data = cleaned\_data.drop(columns=['native-country'])

    cleaned\_data = cleaned\_data.drop(columns=['fnlwgt'])

    return cleaned\_data

2. 熵和信息增益计算

为了构建决策树，首先需要计算数据集的熵和各个特征的信息增益：

- 熵计算: 熵用于衡量数据集中随机变量的不确定性。计算公式涉及所有类别的概率。

- 信息增益: 信息增益用于评估使用特定特征进行数据分割后不确定性减少的量。通过计算分割前后的熵差值得到。

def entropy(target\_col):

    elements, counts = np.unique(target\_col, return\_counts=True)

    entropy = -np.sum([(count / len(target\_col)) \* np.log2(count / len(target\_col)) for count in counts])

return entropy

3. 构建决策树

决策树是通过递归选择最高信息增益的特征进行构建的。同时，考虑了剪枝策略如最大深度、最小样本分割和最小样本叶节点，以避免过拟合：

def build\_decision\_tree(data, features, target\_name="income", depth=0, max\_depth=10, min\_samples\_split=2, min\_samples\_leaf=1):

    """

    递归构建决策树，添加了剪枝参数。

    """

    # 检查样本数是否少于最小分割样本数，如果是，则停止分割，返回最常见的结果

    global cnt

    cnt = cnt + 1

    if(cnt % 100 == 0):

        print(cnt,features)

    if len(data) < min\_samples\_split or len(features) == 0 or depth == max\_depth:

        most\_common = np.unique(data[target\_name])[np.argmax(np.unique(data[target\_name], return\_counts=True)[1])]

        return DecisionTreeNode(leaf=True, prediction=most\_common)

    # 从特征中选择最高信息增益的特征

    igains = {feature: information\_gain(data, feature, target\_name) for feature in features if len(np.unique(data[feature])) > 1}

    if not igains:

        most\_common = np.unique(data[target\_name])[np.argmax(np.unique(data[target\_name], return\_counts=True)[1])]

        return DecisionTreeNode(leaf=True, prediction=most\_common)

    best\_feature = max(igains, key=igains.get)

    tree\_node = DecisionTreeNode(feature\_name=best\_feature)

    depth += 1  # 增加树的深度

    # 对最佳特征的每个独特值递归构建子树

    for value in np.unique(data[best\_feature]):

        sub\_data = data[data[best\_feature] == value]

        if len(sub\_data) < min\_samples\_leaf:

            prediction = np.unique(data[target\_name])[np.argmax(np.unique(data[target\_name], return\_counts=True)[1])]

            tree\_node.children[value] = DecisionTreeNode(leaf=True, prediction=prediction)

        else:

            subtree = build\_decision\_tree(sub\_data, [f for f in features if f != best\_feature], target\_name, depth, max\_depth, min\_samples\_split, min\_samples\_leaf)

            tree\_node.children[value] = subtree

            subtree.value = value

    return tree\_node

4. 设计多组参数，批量化实验：如下代码所示。

    for max\_depth in max\_depths:

        for min\_samples\_split in min\_samples\_splits:

            for min\_samples\_leaf in min\_samples\_leafs:

                now = time.time()

                decision\_tree = build\_decision\_tree(adult\_data, features, max\_depth=max\_depth, min\_samples\_split=min\_samples\_split, min\_samples\_leaf=min\_samples\_leaf)

                with open('log.txt', 'a') as f:

                    f.write(f"Decision Tree with max\_depth={max\_depth}, min\_samples\_split={min\_samples\_split}, min\_samples\_leaf={min\_samples\_leaf} is built in {time.time()-now} seconds\n")

                evaluate(decision\_tree) # evaluate

1. 试验分析

实验结果如下所示。在选定的3\*4\*4=48种参数组合种，“Decision Tree with max\_depth=5, min\_samples\_split=20, min\_samples\_leaf=5”的表现最佳，用时仅69.99 秒，并得到了83.00% 的准确率。

1. 结论

实验结果如下所示。

1. 实验总结

本次实验中，构建了一个决策树模型来预测个人年收入是否超过50,000美元，使用的是Adult数据集。实验流程包括数据清洗，计算熵和信息增益，构建和优化决策树。通过手动编程，我们成功地对数据进行了预处理，删除了缺失值和不相关的列，确保了数据的质量。在构建决策树的过程中，我们特别注重选择信息增益最高的特征进行分割，以此构造树模型，并采用了剪枝策略来防止过拟合，如设定最大深度、最小样本分割数和最小样本叶节点数。

通过多组参数的批量实验，发现当最大深度为5，最小样本分割数为20，最小样本叶节点数为5时，模型的表现最佳，准确率达到了83.00%。这一参数组合不仅保证了模型的预测准确性，同时也显著减少了计算时间和过拟合的风险。此外，我们还记录了不同参数组合下的实验结果，并对比分析了它们在准确率和运行时间上的表现，以此优化决策树模型的整体性能。

总的来说，这次实验不仅加深了我对决策树理论的理解，还提高了我在实际数据集上应用机器学习模型的能力。