题目 1. 自选数据集（例如 lris 数据集）实现决策树算法，并用 Micro-F1 和 Macro-F1 分数进行验证集评估，语言和工具库不限。• 提交 pdf 格式报告以及可运行代码压缩包，报告内容包括但不限于：

- 数据的分析与处理（1）；
- 决策树的设计原理和核心代码（2）；
- 验证集评估结果（Micro-F1 和 Macro-F1 截图）（1）；
- 使用开源库对决策树的可视化结果（1）。

解答.

1. 数据分析与处理

   使用 iris = load_iris() 并查看 iris 的内容 (数据集的获取借助 sklearn 包提供的函数)：

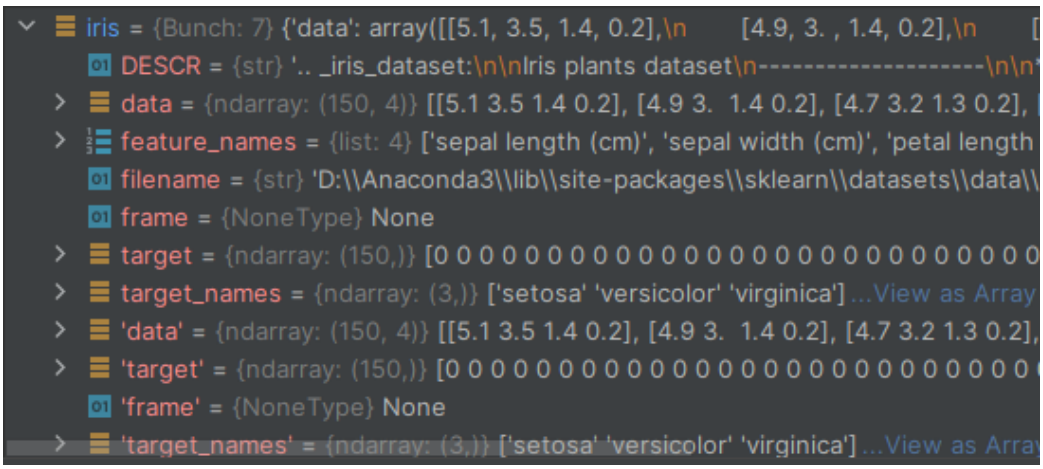

图 1: iris 数据集

关键的属性为 data，其中存储了每个样本的各个属性的值，以及 tar-

get，其中存储了每个样本所属的类别。

然后使用 init_data_set 函数对数据集进行预处理，将 ndarray 的数据转换为 list 类型，并将标签附加到每一个样本后面。

```
"""
初始化数据集,返回一个列表,值为一个样本的属性与其标签,比如一个样本为[1.5, 2.5, 3.5, 4.5, 2]
# 则前四个元素为对应的属性,最后一个元素2为这个样本所属的类别
"""


1 usage
def init_data_set():
    iris = load_iris()  # 导入数据集iris
    iris_feature = iris.data.tolist()  # 样本属性
    iris_target = iris.target.tolist()  # 样本类别
    for i in range(len(iris_feature)):
        iris_feature[i].append(iris_target[i])
    return iris_feature
```

图 2: 处理数据集

然后将数据集划分为训练集和测试集

```
"""
划分数据集 将total_data_set 中比例为 split_rate的部分划分为训练集 剩下的作为数据集
"""


1 usage
def create_train_and_test_set(total_data_set, split_rate=0.75):
    # 0的是测试集，1的是训练集
    length = len(total_data_set)
    train_num = int(length * split_rate)
    test_num = length - train_num
    random_list = [1] * train_num
    random_list.extend([0] * test_num)
    random.shuffle(random_list)
    test_set = []
    train_set = []
    for i in range(length):
        if random_list[i] == 0:
            test_set.append(total_data_set[i])
        else:
            train_set.append(total_data_set[i])
    return test_set, train_set
```

图 3: 划分训练集与测试集

2. 决策树设计原理与核心代码构造决策树的算法借鉴了 ID3 的思想，遍历所有属性，按照每一个所有可能得取值对训练集进行预划分，例如遍历到了第三个属性（花瓣长度）且一个取值为 3.0，则将训练集分为花瓣长度小于 3.0 和花瓣长度大于 3.0 两部分，可以预见，最终构造出来的决策树应该是一个二叉树。然后计算信息增益，选择信息增益最大的属性的取值 value 构造当前决策树的根节点。

```
"""
选择最好的特征值进行分类
返回最好的特征值以及最佳的信息增益
例如返回[2,3.0] 与 0.8
则代表最好的分类特征为第2个属性(下标从0开始),并应该以3.0为划分标准将其划分为两部分,其所能达到的信息增益为0.8
"""


2 usages
def choose_best_split(data_set):
    base_Ent = calculate_Entropy(data_set)
    best_increase = 0.0
    best_feature = [-1, -1]
    for i in range(4):
        features = [j[i] for j in data_set]
        unique = set(features)
        for feature in unique:
            less_Set, more_Set = split_Set(data_set, i, feature)
            tmp = len(less_Set) / float(len(data_set))
            new_Ent = tmp * calculate_Entropy(less_Set) + (1 - tmp) * calculate_Entropy(more_Set)
            increase = base_Ent - new_Ent
            if increase > best_increase:
                best_increase = increase
                best_feature = [i, feature]
    return best_feature, best_increase
```

图 4: 选择如何划分

构造树的函数采用了递归的思想，当当前的数据集全是同一类别的时候，构造叶子节点并返回，否则先计算最佳划分的属性以及划分的值，进行划分，然后递归地构造左子树和右子树。返回的决策树使用字典类型来保存。

```
"""
使用递归的方式以字典的形式构造并返回决策树
叶子结点只有一个键为class,值为其应属于的类别
否则在node键中存储分类的特征  并在left和right键中存储左子树和右子树构成的字典
"""


3 usages
def create_tree(data_set):
    myTree = {}
    label = [i[-1] for i in data_set]
    label_set = set(label)
    if len(label_set) == 1:
        myTree['class'] = label[0]
        return myTree
    best_feature, best_increase = choose_best_split(data_set)
    myTree['node'] = best_feature
    less_Set, more_Set = split_Set(data_set, best_feature[0], best_feature[1])
    myTree['left'] = create_tree(less_Set)
    myTree['right'] = create_tree(more_Set)
    return myTree
```

图 5: 构造决策树

3. 验证集评估结果

   调用 sklearn 提供的 f1_score 函数对决策树在测试集上的分类结果进行评分：

```
micro-F1分数为:0.9666666666666667
macro-F1分数为:0.9682539682539683
```

图 6: micro 和 macro 评估结果

   经过多次测试评分分数在 0.92-0.98 之间

4. 决策树可视化

使用 matplotlib 对决策树进行可视化绘图：

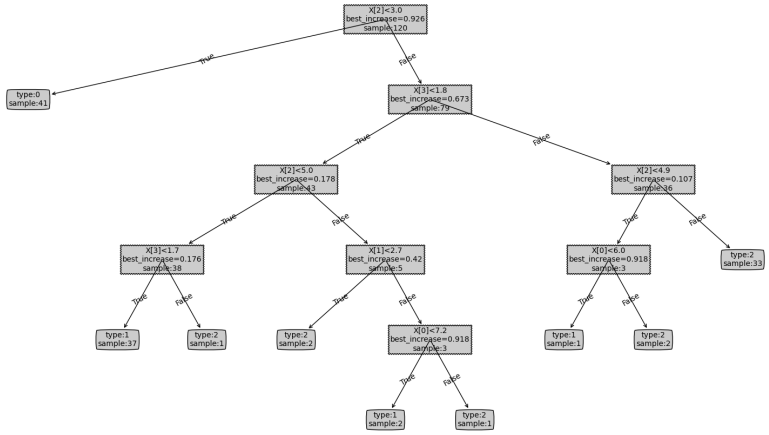

图 7: 决策树可视化