Dtree 小作业

211870287 丁旭

2023 年 11 月 14 日

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）。
- 实现决策树改进方案（加分项）（1`）

解答.

数据分析与处理

使用 iris = load_iris() 并查看 iris 的内容，并将 ndarray 的数据转换为 list 类型，并将标签附加到每一个样本后面。

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

```
iris_feature:  [[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5.0, 3.6, 1.4, 0.2], [5.4, 3.9, 1.7, 0.4], [4.6, 3.4, 1.4, 0.3], [5.0, 3.4, 1.5, 0.2], [4.4, 2
iris_category:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

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。

将数据集划分为训练集和测试集

```python
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 for _ in range(train_num)] + [0 for _ in range(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

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，则将训练集分为花瓣长度小于3和花瓣长度大于3两部分，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。

```
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
```

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。

```
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)

    # 进行预剪枝
    if best_increase == 0.0:
        # myTree['class'] = max(label, key=label.count)
        myTree['class'] = majorityCnt(label)
        return myTree

    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
```

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：

```python
if __name__ == '__main__':
    data_set = init_data_set()
    # print(data_set)
    test_set, train_set = create_train_and_test_set(data_set, 0.8)
    res = create_tree(train_set)
    y_true = [example[-1] for example in test_set]
    y_pred = [predict(res, example) for example in test_set]
    # # 用Micro-F1和Macro-F1分数进行验证集评估
    print('micro-F1分数为:', f1_score(y_true, y_pred, labels=[0, 1, 2], average='micro'))
    print('macro-F1分数为:', f1_score(y_true, y_pred, labels=[0, 1, 2], average='macro'))
```

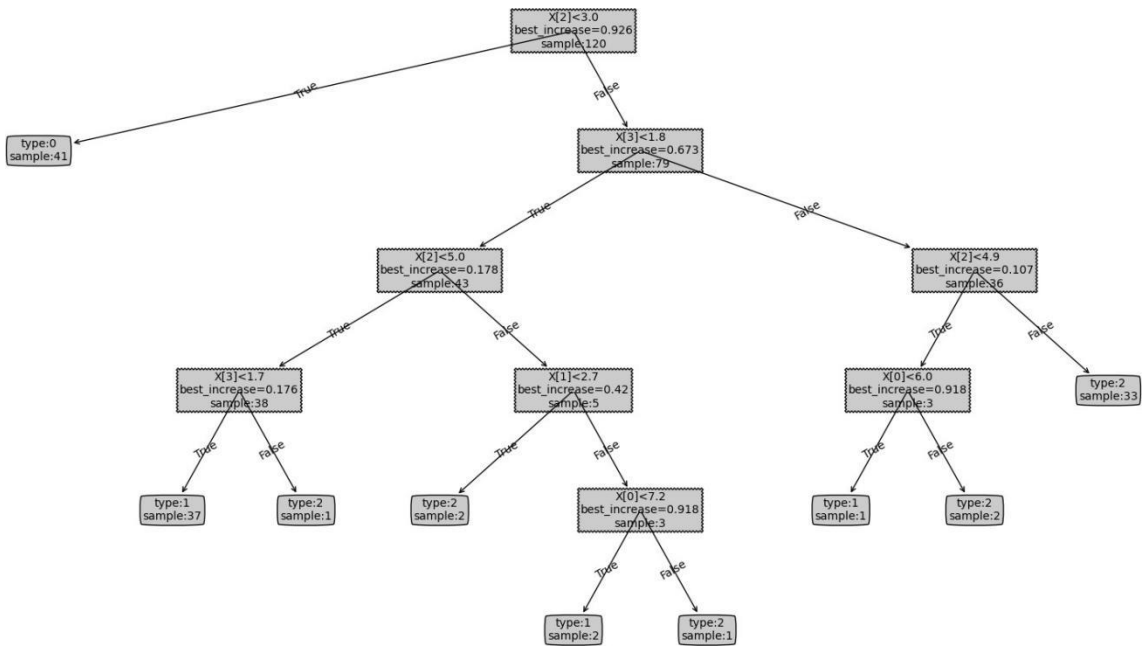
```
micro-F1分数为: 0.9666666666666667
macro-F1分数为: 0.9721739130434782

Process finished with exit code 0
```

图 6: micro 和 macro 评估结果

决策树可视化

使用 matplotlib 对决策树进行可视化绘图：



决策树可视化

决策树改进方案

预剪枝

```python
def prePruning(dataSet, pruneSet, labels):
    classList = dataSet[:, -1]
    if len(set(classList)) == 1:
        return classList[0]
    if len(dataSet[0]) == 1:
        return mayorClass(classList)
    # 获取最好特征
    bestFeat = chooseBestFeature(dataSet, labels)  # √
    bestFeatLabel = labels[bestFeat]  # 获取特征名称
    sublabels = np.delete(labels, bestFeat)  # 从特征名列表删除
    # 计算初始正确率
    baseRightNums = cntAccNums(dataSet, pruneSet)
    # 得到最好划分属性取值
    featureList = dataSet[:, bestFeat]  # 最佳的特征列也选出来了
    features = set(featureList)  # 取出特征下的属性集合
    # 计算尝试划分节点时的正确率
    splitRightNums = 0.0
    for value in features:  # 遍历每一个属性
        # 每个属性取值得到的子集
        subDataSet = splitDataSetByValue(dataSet, bestFeat, value)
        if len(subDataSet) != 0:
            # 把用来剪枝的子集也按照相应属性值划分下去
            subPruneSet = splitDataSetByValue(pruneSet, bestFeat, value)
            # if value == "凹陷":
            #     print(subPruneSet)
            splitRightNums += cntAccNums(subDataSet, subPruneSet)
    if baseRightNums < splitRightNums:  # 如果不划分的正确点数少于尝试划分的点数，则继续划分。
        myTree = {bestFeatLabel: {}}
    else:
        return mayorClass(dataSet[:, -1])  # 否则，返回不划分时投票得到的类
    # 以下代码和不预剪枝的代码大致相同，一点不同在于每次测试集也要参与划分。
    for value in features:
        subDataSet = splitDataSetByValue(dataSet, bestFeat, value)
        subPruneSet = splitDataSetByValue(pruneSet, bestFeat, value)
        if len(subDataSet) != 0:
            myTree[bestFeatLabel][value] = prePruning(subDataSet, subPruneSet, sublabels)
        else:
            # 计算D中样本最多的类
            myTree[bestFeatLabel][value] = mayorClass(classList)
    return myTree
```

后剪枝

```python
def postPruning(dataSet, pruneSet, labels):
    classList = dataSet[:, -1]
    # 如果基尼指数为0，即D中样本全属于同一类别，返回
    if len(set(classList)) == 1:
        return classList[0]
    # 属性值为空，只剩下类标签
    if len(dataSet[0]) == 1:
        return mayorClass(classList)

    # 得到增益最大划分的属性、值
    bestFeat = chooseBestFeature(dataSet, labels)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel: {}}  # 创建字典，即树的节点。
    # 生成子树的时候要将已遍历的属性删去。数值型不要删除。
    sublabels = np.delete(labels, bestFeat)
    featureList = dataSet[:, bestFeat]  # 最佳的特征列也选出来了
    uniqueVals = set(featureList)  # 取出特征下的属性集合
    for value in uniqueVals:  # 标称型的属性值有几种，就要几个子树。
        # Python中列表作为参数类型时，是按照引用传递的，要保证同一节点的子节点能有相同的参数。
        subPrune = splitDataSetByValue(pruneSet, bestFeat, value)
        subDataSet = splitDataSetByValue(dataSet, bestFeat, value)
        if len(subDataSet) != 0:
            myTree[bestFeatLabel][value] = postPruning(subDataSet, subPrune, sublabels)
        else:
            # 计算D中样本最多的类
            myTree[bestFeatLabel][value] = mayorClass(classList)
    # 后剪枝，如果到达叶子节点，尝试剪枝。
    # 计算未剪枝时，测试集的正确数
    numNoPrune = 0.0
    for value in uniqueVals:
        subDataSet = splitDataSetByValue(dataSet, bestFeat, value)
        if len(subDataSet) != 0:
            subPrune = splitDataSetByValue(pruneSet, bestFeat, value)
            numNoPrune += cntAccNums(subDataSet, subPrune)
    # 计算剪枝后，测试集正确数
    numPrune = cntAccNums(dataSet, pruneSet)
    # 比较决定是否剪枝，如果剪枝后该节点上测试集的正确数变多了，则剪枝。
    if numNoPrune < numPrune:
        return mayorClass(dataSet[:, -1])  # 直接返回节点上训练数据的多数类为节点类。
    return myTree
```