

# 机器学习个人补充报告

— lab02 分类实验



学 院 电子与信息工程学院

专 业 计算机科学与技术

授课老师 李 洁

学 号 1853790

姓 名 庄镇华

完成日期 2021.06.21

说明：原报告手写实现了基于最小风险的贝叶斯决策算法，本次补充报告手写实现了决策树，并分析了基于问题数据集特征的树生成与剪枝方法，应用于 cifar10 图片分类任务。

## 一、算法原理

决策树的三种主流算法是 ID3、C4.5、CART 算法，其中 ID3 和 C4.5 基于信息熵，而 CART 基于基尼系数。本次补充实验主要实现了 **ID3 算法和 C4.5 算法**。

### 1.1 ID3 算法

信息熵是度量样本集合纯度最常用的一种指标。假定当前样本集合  $D$  中第  $k$  类样本所占的比例为  $p_k$  ( $k = 1, 2, \dots, n$ )，则  $D$  的信息熵定义为

$$Entropy(D) = - \sum_{i=1}^n p_k \log_2 p_k$$

$Entropy(D)$  的值越小，则  $D$  的纯度越高。用属性  $A$  对样本集  $D$  进行划分所获得的“信息增益”为

$$infoGain(D | A) = Entropy(D) - Entropy(D | A)$$

### 1.2 C4.5 算法

ID3 算法根据信息增益值生成决策树。信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好可能带来的不利影响，C4.5 算法采用信息增益率来选择最优划分属性。信息增益率的定义为

$$GainRatio(D | A) = \frac{infoGain(D | A)}{IV(A)}$$

$$IV(A) = - \sum_{k=1}^K \frac{|D_k|}{|D|} \log_2 \frac{|D_k|}{|D|}$$

$IV(A)$  称为属性  $A$  的“固有值”，属性  $A$  的可能取值数目越多，则  $IV(A)$  的值通常会越大。需要注意的是，增益率准则对可取值数目较少的属性有所偏好，因此，C4.5 算法并不是直接选择增益率最大的候选划分属性，而是使用了一个启发式算法：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

同时，ID3 算法仅可以处理离散型特征值，而 C4.5 算法还可以处理连续型特征值。

## 二、生成与剪枝方法

### 2.1 生成方法

决策树学习的目的是为了产生一棵泛化能力强，处理未见示例能力强的决策树，其基本流程遵循“分而治之”策略。

决策树的生成是一个**递归过程**。在决策树基本算法中，有三种情形会导致递归返回：(1) 当前结点包含的样本全属于同一类别，无需划分；(2) 当前属性集为空，或是所有样本在所有属性上取值相同，无法划分；(3) 当前结点包含的样本集合为空，不能划分。

在第(2)种情形下，把当前结点标记为叶结点，并将其类别设定为该结点所含样本最多的类别；在第(3)种情形下，同样把当前结点标记为叶结点，但将其类别设定为其父结点所含样本最多的类别。注意这两种

情形的处理实质不同：情形(2)是在利用当前结点的后验分布，而情形(3)则是把父结点的样本分布作为当前结点的先验分布。

```
def create_tree(data_set, labels):
    class_split = [sample[-1] for sample in data_set]
    # 如果类别完全相同，则停止继续划分
    if class_split.count(class_split[0]) == len(class_split):
        return class_split[0]
        # return Node(_class=class_split[0], depth=0, leaf=True)
    # 遍历完所有特征时返回出现次数最多的类别
    if len(data_set[0]) == 1:
        return majority_count(class_split)
        # return Node(_class=majority_count(class_split), depth=0, leaf=True)
    best_feature, best_feature_value = choose_best_feature_to_split(data_set,
labels)
    # 如果无法选出最优分类特征，返回出现次数最多的类别
    if best_feature == -1:
        return majority_count(class_split)
        # return Node(_class=majority_count(class_split), depth=0, leaf=True)
    best_feature_label = labels[best_feature]
    tree = {best_feature_label: {}}
    # tree = Node(best_feature=best_feature,
best_feature_label=best_feature_label, depth=0)
    sub_labels = labels[:best_feature] + labels[best_feature + 1:]

    # 如果最佳切分特征是离散型
    if type(data_set[0][best_feature]).__name__ == 'str':
        feature_values = [sample[best_feature] for sample in data_set]
        unique_values = set(feature_values)
        for value in unique_values:
            sub_data_set = split_data_set(data_set, best_feature, value)
            tree[best_feature_label][value] = create_tree(sub_data_set,
sub_labels)
            # tree.children.append(create_tree(sub_data_set, sub_labels))

    # 如果最佳切分特征是连续型
    if type(data_set[0][best_feature]).__name__ == 'int' or type(
        data_set[0][best_feature]).__name__ == 'float':
        # 将数据集划分为两个子集，针对每个子集分别建树
        value = best_feature_value
        greater_sub_data_set = split_continuous_data_set(data_set,
best_feature, value, 0)
        smaller_sub_data_set = split_continuous_data_set(data_set,
best_feature, value, 1)
        # 针对连续型特征，在生成决策的模块，修改划分点的标签，如"> x.xxx", "<= x.xxx"
        tree[best_feature_label]['>' + str(value)] =
create_tree(greater_sub_data_set, sub_labels)
        tree[best_feature_label]['<=' + str(value)] =
create_tree(smaller_sub_data_set, sub_labels)
    return tree
```

## 2.2 剪枝方法

决策树是一个非常容易发生过拟合的模型，因为如果没有任何限制，在生成阶段，它将会为所有特征生成分支。这样的后果是一方面叶子节点过多，容易缺少泛化能力，另一方面训练速度会非常缓慢。可以通过剪枝的方法来降低树模型复杂度。剪枝可分为预剪枝、后剪枝两类。

本次补充实验主要采用**预剪枝**的方法。

---

## 预剪枝

预剪枝指在完全正确分类之前，决策树会较早地停止树的生长。停止生长的方法可以被总结为**一般情况下的停止**和**严格情况下的停止**两种。从根节点开始，从上至下判断收起当前节点的分支后，验证集性能是否变好。若是，剪去当前节点，否则保留。

**优点：**降低过拟合；显著减少训练时间和测试时间开销；

**缺点：**剪去的节点，虽然当前划分不能提升泛化性能，但在其基础上进行的后续划分却有可能导致性能显著提高。预剪枝基于“贪心”本质禁止这些分支展开给预剪枝决策树带来了欠拟合的风险。

---

### 一般情况下的停止

- ✚ 如果所有样本均属同一类。
- ✚ 如果样本的所有的特征值都相同。

---

### 严格情况下的停止

- ✚ 如果树到达一定高度。
- ✚ 如果节点下包含的样本点小于指定的阈值。
- ✚ 如果扩展当前节点不会改善信息增益，即信息增益小于指定的阈值。

本次补充实验选择的剪枝方法是一般情况下的停止中“如果所有样本均属同一类”、“样本的所有的特征值都相同”，严格情况下的停止中“如果树到达一定高度”、“如果节点下包含的样本点小于指定的阈值”和“如果扩展当前节点不会改善信息增益，即信息增益小于指定的阈值”。具体实现如下：

“如果所有样本均属同一类”：

```
if len(values) == 1:
    continue # entropy = 0
```

“如果树到达一定高度”：

```
if node.depth < self.max_depth:
    node.children = self._split(node)
    if not node.children: # leaf node
        self._set_label(node)
    queue += node.children
else:
    self._set_label(node)
```

“如果节点下包含的样本点小于指定的阈值”：

```
if min(map(len, splits)) < self.min_samples_split:
    continue
```

“如果扩展当前节点不会改善信息增益，即信息增益小于指定的阈值”：

```
if gain < self.min_gain:
    continue # stop if small gain
```

---

## 后剪枝

先从训练集生成一棵完整决策树，再从下往上剪枝，剪枝依据同上。

**优点：**欠拟合风险很小；泛化性能往往优于预剪枝

**缺点：**训练时间开销很大

由于后剪枝训练时间开销很大，由于时间限制，本次补充实验没有采用后剪枝方法。

### 三、实验结果

由于数据集为图片，每个数据样本的大小都是  $32 \times 32 \times 3 = 3072$ ，因此如果直接将像素点直接作为特征，一方面特征数量太多，训练速度比较慢，另一方面，一些干扰项混在特征里面，对最终分类结果有不良影响，所以最终我选择了 PCA 的方法，首先将数据降维。

```
# 数据降维
X_train, y_train, X_test, y_test = load_CIFAR10('./data')
pca = PCA(n_components=feature_num)
X_train, X_test = pca.fit_transform(X_train), pca.fit_transform(X_test)
```

降维之后，分别用手写的 ID3 模型、C4.5 模型和 sklearn 的决策树模型对验证集和测试集分别进行预测，得到结果如下表所示：

	验证集准确率	测试集准确率
手写的 ID3 模型	87.32%	17.32%
手写的 C4.5 模型	88.35%	26.93%
sklearn 决策树模型	86.52%	27.24%

可以看到，在验证集准确率方面，三种模型的准确率都相差不大。

在测试集准确率方面，相较于官方库的决策树，ID3 算法的测试集准确率较低，C4.5 算法测试集准确率与官方库相差无几，说明构造出来的决策树是正确的。

但总体而言，对于本数据集，相较于深度学习模型，无论是手写的 ID3、C4.5 模型，还是 sklearn 库的决策树模型，准确率都相差甚远。这恰恰说明了在 cv 方面，深度学习方法的优越性。