中山大学数据科学与计算机学院
计算机科学与技术专业-人工智能
本科生实验报告

（2018-2019 学年秋季学期）

学　　号：16337113

姓　　名：劳马东

教学班级：教务 2 班

专　　业：超算

1 实验题目

实现 ID3,C4.5,CART 三种决策树。不要求实现连续型数据的处理，不要求实现剪枝。

2 实验内容

2.1 算法原理

2.1.1 决策树

决策树是一棵多叉树，它通过执行一系列测试达到决策。树中内部节点代表对输入属性 A_i 的值的一个测试（如年龄），从结点射出的分支代表属性 A_i 的可能值（如 ≥ 18 岁），叶节点代表一种决策。

分类决策树模型是一种描述对实例进行分类的决策树，叶节点表示一个类。用决策树分类，从根节点开始，对实例的某一特征进行测试，根据测试结果，将实例分配到其子节点；这时，每一个子节点对应着该特征的一个取值。如此递归地对实例进行测试并分配，直到达到叶节点。最后将实例分到叶节点的类中。

2.1.2 决策树的建树过程

1、 初始化

创建根结点，它拥有全部数据集和特征。

2、 选择特征

遍历当前结点的数据集和特征，根某种原则选择一个特征。

3、 划分数据

根据这个特征的取值，将当前数集划分为若干子集。

4、 创建结点

为每个子数据集创建一个子结点，并删去选中的特征。

5、 递归建树

对每个子结点，回到第 2 步，直到达到边界条件，则回溯。

递归边界条件：

- 剩余样例属于同一类 C，该节点标记为 C 类叶节点，返回 C；
- 无样例，此时返回一个缺省值，该值是构造其父节点用到的所有样例中得票最多的类 C_p ，该节点标记为 C_p 类叶节点；
- 无属性可选择，根据多数投票原则选出剩余样例的类别 C_v，该节点标记为 C_v 类叶节点，返回 C_v。

6、 完成建树

叶子结点采用多数投票的方式判定自身类别。

2.1.3 特征选择指标

假设数据集 (X, Y) 的子数据集个数为 n，每个子数据集为 $(X_i, Y_i)(i \in [0, n))$，属性集 F 的每个属性为 F_j，则属性 F_j 在指标 M 上所得的分数为：

$$\underset{F_j \in F}{S(Y|F_j)} = \sum_{i=0}^{n-1}(\frac{|Y_i|}{Y} \times M(Y_i)) \tag{1}$$

1、 错误率

对于 Y_i 根据多数投票原则得到其类别 C_i，错误率计算 Y_i 中出现错误的频率。公式如下：

$$M(Y_i) = \frac{|\{y_k \in Y_i : y_k \neq C_i\}|}{|Y_i|} \tag{2}$$

我们希望最优属性使得错误率最小，即：

$$\begin{aligned} best(F) &= \underset{F_j \in F}{\arg\min}\, S(Y|F_j) \\ &= \underset{F_j \in F}{\arg\max}\, [S(Y) - S(Y|F_j)] \end{aligned} \tag{3}$$

2、 信息增益

信息增益采用熵计算，熵是随机变量不确定性的度量，其计算公式为：

$$M(Y_i) = H(Y_i) = -\sum_k p(y_k) \log_2 p(y_k) \tag{4}$$

$p(v_k)$ 表示 v_k 在 V 上的频率。随机变量 A 的信息增益表示 A 对 Y 的不确定性的减小程度，公式为：

$$I(A) = H(Y) - H(Y|A) \tag{5}$$

我们希望最优属性使得 Y 的不确定性最小，即：

$$\begin{aligned} best(F) &= \underset{F_j \in F}{\arg\max}\, [H(Y) - H(Y|F_j)] \\ &= \underset{F_j \in F}{\arg\max}\, [S(Y) - S(Y|F_j)] \end{aligned} \tag{6}$$

3、 信息增益率

信息增益的方法偏向于找出分支多的属性作为最优属性，因为分支越多代表不确定性越小，信息增益就越大。然而选择分支多的属性容易造成过拟合，因此该方法对其进行改进，在信息增益的基础之上乘上一个惩罚参数。特征取值较多时，惩罚参数较小；特征取值较少时，惩罚参数较大。该参数为特征的熵的倒数。公式如下：

$$\begin{aligned} best(F) &= \underset{F_j \in F}{\arg\max}\, \frac{H(Y) - H(Y|F_j)}{H(F_j)} \\ &= \underset{F_j \in F}{\arg\max}\, \frac{S(Y) - S(Y|F_j)}{S(F_j)} \end{aligned} \tag{7}$$

4、 基尼系数

基尼系数原本是用以衡量一个国家或地区居民收入差距的指标，介于 0-1 之间。基尼系数越大，表示不平等程度越高。对于一个随机变量，基尼系数代表其取值的差异程度，计算公式如下：

$$M(Y_i) = G(Y_i) = \sum_k p(y_k) \times (1 - p(y_k)) = 1 - \sum_k p(y_k)^2 \tag{8}$$

我们希望选择的最优属性使 Y 取值的差异程度最小，即：

$$\begin{aligned} best(F) &= \arg\min_{F_j \in F} G(Y|F_j) \\ &= \arg\min_{F_j \in F} S(Y|F_j) \\ &= \arg\max_{F_j \in F} [S(Y) - S(Y|F_j)] \end{aligned} \tag{9}$$

从以上分析可以看出，特征选择的公式可以统一化为一个公式：

$$best(F) = \arg\max_{F_j \in F} \frac{S(Y) - S(Y|F_j)}{punishment} \tag{10}$$

对于信息增益率方法，$punishment = H(F_j)$，其余方法为 1。

2.2 伪代码

1、 决策树建树过程基本框架

输入: 训练集 X，训练集 y，父节点 *parent*，分支的取值 *brach*

function $build\_tree(X, y, parent, brach)$
 $F \leftarrow feature(X)$
 $root \leftarrow Node(brach, parent)$
 if X *is empty* **then**
  $root.label \leftarrow parent.label$
 else if F *is empty* **then**
  $root.label \leftarrow vote(X)$
 else if $is\_same(y)$ **then**
  $root.label \leftarrow y[0]$
 else
  $root.label \leftarrow vote(y)$
  $S_y \leftarrow M(y)$
  for each $F_j \in F$ **do**
   $S_j \leftarrow 0$
   for each $(F_{ji}, X_i, y_i) \in divide(F_j, X, y)$ **do**
    $S_j \leftarrow S_j + \frac{|y_i|}{|y|} \times M(y_i)$
   end for
   $S_j \leftarrow \frac{S_y - S_j}{punishment}$
   $S_{best}, F_{best} \leftarrow \max(S_{best}, F_{best}, S_j, F_j)$
  end for
  $root.feature \leftarrow F_{best}$
  // 根据所选属性划分数据集
  for each $(branch, X_i, y_i) \in divide(F_{best}, X, y)$ **do**
   $sub\_tree = build\_tree(X_i, y_i, root, branch)$ // 递归建树
   $root.add\_child(sub\_tree)$
  end for
 end if
 return *root*
end function

2.3 关键代码

1、 特征选择：遍历所有属性（编号为数字），根据该属性划分数据集为若干子数据集，计算子数据集在某种指标上的得分，最后求这些子数据集得分的均值。如果该指标带有惩罚参数，得分均值需要乘上惩罚参数。

```python
score_best, feature_best = None, None
score_y = self._M(y)
# col为特征编号
for col in cols:
    score_j = 0
    # 根据特征划分数据集并计算得分（频率*指标得分的和）
    for sub_rows in self._divide(col, rows).values():
        score_j += len(sub_rows) / len(rows) \
                   * self._M(self._y.take(sub_rows, 0))
    score_j = score_y - score_j
    # 处理有惩罚的指标，如信息增益率
    if self._punish:
        feature = self._X[:, col].take(rows, 0)
        score_j /= self._M(feature)
    # 找出得分最高的属性
    if score_best is None or score_j > score_best:
        score_best = score_j
        feature_best = col
```

图 1: 特征选择

2、 数据集划分与递归：选择出最优属性后，需要将其从属性集合中删除，并创建一个内部结点，代表一种划分方法。该节点的类标签由数据集根据多数投票原则得出，当在测试集上无相应分支时返回这个内部结点的类标签。根据最优属性划分数据集递归建树，值得注意的地方是 cols 需要传副本而不是本身，否则在回溯时 cols 的值将发生改变。

```python
# 删除最优属性
cols.remove(feature_best)
# 根据所选属性创建新内部节点，其label根据多数投票原则得到
root = self.create_node(tag=(branch,
                             feature_best,
                             self._vote(y)),
                        parent=parent)
for branch, sub_rows in self._divide(feature_best, rows).items():
    self._build_tree(sub_rows, cols[:], root, branch)
```

图 2: 数据集划分与递归

3、 预测：预测的过程就是顺着决策树不断匹配的过程。取测试数据在对应属性上的值，如果有匹配的分支则进入分支继续重复该过程，否则返回结点的类标签。

```python
def _predict(self, test_data, node_id):
    while True:
        node = self.get_node(node_id)
        # tag[1]存储划分属性
        pro = node.tag[1]
        # 到了叶节点
        if pro is None:
            break
        # 取测试数据某一属性的值
        condition = test_data[pro]
        # 匹配当前节点的每个子节点
        for child in self.children(node_id):
            if child.tag[0] == condition:
                node_id = child.identifier
                break
        else:
            break
    return node.tag[2], node.is_leaf()
```

图 3: 预测

3 优化 & 创新

1、 数据集划分优化

数据集的划分涉及到许多次拷贝，例如，要将 $m \times n$ 的 X 分成 $[0, a)$、$[a, b)$、$[b, m)$ 的三部分，若采用简单的划分方式，即返回 $X[0:a]$、$X[a:b]$、$X[b:n]$，将会出现非常多的拷贝，时间复杂度为 $(O(m \times n))$, 尤其是当 n 较大时，拷贝将会成为瓶颈。其实，只需要记录当前子树的数据集对应的行和列的集合，在划分时对行或列的集合进行划分而不是对数据集本身划分，就能将时间复杂度降低为 $O(m + n)$。

4 实验结果及分析

4.1 实验结果展示示例

4.2 评测指标展示及分析

5 思考题