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）

课程名称：**Artificial Intelligence**

教学班级	计科 2 班	专业（方向）	计算机科学与技术
学号	16337341	姓名	朱志儒

实验题目

决策树

实验内容

- ### 算法原理

决策树

决策树是一种树形结构，可以是二叉树或非二叉树，树中每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，每个叶节点存放一个类别。使用决策树进行分类的过程就是从根节点开始，测试待分类项中相应的特征属性，按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为分类结果。

ID3

ID3 模型是以信息熵和信息增益作为衡量标准的分类模型。

熵是指信息的混乱程度，熵值越大，变量的不确定性也就越大，计算信息熵的公式：

$$\mathrm{Entropy(S)} = -\sum_{i=1}^{m} p(u_i)log_2(p(u_i))$$

其中，$\mathrm{p}(u_i) = \frac{|u_i|}{|S|}$，$\mathrm{p}(u_i)$为类别$u_i$在样本S中出现的概率。

条件熵是指在已知第二个随机变量 X 的值的前提下，随机变量 Y 的信息熵。计算特征 A 对数据集 S 的条件熵的公式：

$$\mathrm{H(S|A)} = \sum_{V\in Value(A)} \frac{|S_V|}{|S|} Entropy(S_V)$$

其中，A 表示样本特征，Value(A)是特征 A 所有的取值集合，V 是 A 中一个特征值，S_V是 S 中 A 的值为 V 的样例集合。

信息增益是指在某个条件下，信息复杂度，即不确定性，减少的程度。计算信息增益的公式：

$$\mathrm{infoGain(S,A) = Entropy(S) - H(S|A)}$$

其中，A 表示样本特征。

在构建决策树时，选择信息增益最大的特征作为决策点。

C4.5

C4.5 模型在 ID3 模型的基础上稍作改进，C4.5 是以信息增益率作为衡量标准的分类模型。C4.5 克服了 ID3 的一个缺点：用信息增益选择特征时偏向于选择分枝比较多的特征值；

计算特征 A 对数据集的信息增益的公式：

$$\mathrm{infoGain(D,A) = Entropy(D) - H(D|A)}$$

计算数据集 D 关于特征 A 的值的熵的公式：

$$\mathrm{SplitInfo(D,A)} = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \log(\frac{|D_j|}{|D|})$$

这个值表示通过将数据集 D 划分成对应于特征 A 测试的 v 个输出的 v 个划分产生的信息。

计算信息增益率的公式：

$$\mathrm{GainRatio(A)} = \frac{infoGain(D,A)}{SplitInfo(D,A)}$$

在构建决策树时，选择信息增益率最大的特征作为决策点。

CART

CART 模型生成的决策树是二叉树，CART 是以 GINI 指数作为衡量标准的分类模型。

GINI 指数是一种不等性度量，通常用来度量收入不平衡，也可用来度量任何不均匀分布，与熵的概念相似，总体内包含的类别越多，GINI 指数就越大。GINI 指数为介于 0~1 之间的数，0 表示完全相等，1 表示完全不相等。

对于一个数据集 D 包含来自 n 个类的样本，计算 GINI 指数的公式：

$$\mathrm{gini(D)} = \sum_{j=1}^{n} p_j(1-p_j) = 1 - \sum_{j=1}^{n} p_j^2$$

其中，p_j是类 j 在 D 中的相对频率。

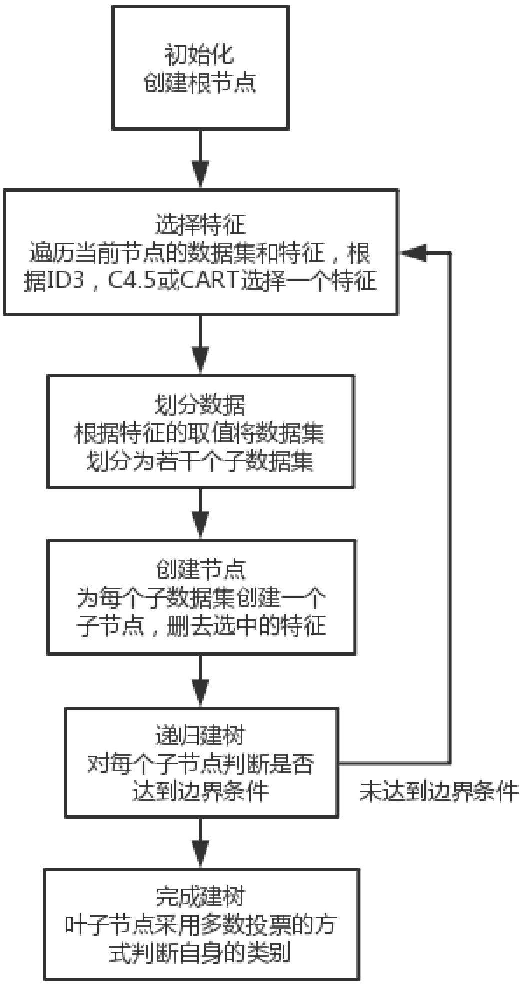
如果一个数据集 D 被分成两个子集 D_1和 D_2大小分别为 N_1和 N_2，数据包含来自 n 个类的样本，则计算 GINI 指数的公式：

$$gini_{split}(D) = \frac{N_1}{N} gini(D_1) + \frac{N_2}{N} gini(D_2)$$

选择具有最小$gini_{split}(D)$的属性作为分裂节点的属性，对每个属性均需要遍历所有可能的分裂位置点。

- **流程图&伪代码**

构建决策树的流程图：



- **关键代码**

1. **计算数据集 D 的信息熵：**

```
def empirical_entropy(train_set, D):
    '''根据数据集 train_set 计算类别 D 的信息熵'''
    dict_of_kinds = {}
    for i in range(len(train_set)):
        label = train_set[i][D]
        if label not in dict_of_kinds.keys():
            dict_of_kinds[label] = 1
        else:
            dict_of_kinds[label] += 1
    summ = len(train_set)
```

```
    for key in dict_of_kinds.keys():
        pd = dict_of_kinds[key] / summ
        dict_of_kinds[key] = pd * math.log(pd)
    return -sum(list(dict_of_kinds.values()))
```

2. 计算特征 A 对数据集 D 的条件熵：

```
def condition_entropy(train_set, A, D):

    '''根据数据集 train_set，在已知条件 A 的前提下，计算 D 的条件熵'''

    dict_of_kinds = {}
    for i in range(len(train_set)):
        label = train_set[i][A]
        if label not in dict_of_kinds.keys():
            dict_of_kinds[label] = [i]
        else:
            dict_of_kinds[label].append(i)
    for key in dict_of_kinds.keys():
        dict_of_acct = {}
        for j in dict_of_kinds[key]:
            label = train_set[j][D]
            if label not in dict_of_acct.keys():
                dict_of_acct[label] = 1
            else:
                dict_of_acct[label] += 1
        summ = len(dict_of_kinds[key])
        for keyy in dict_of_acct.keys():
            pd = dict_of_acct[keyy] / summ
            dict_of_acct[keyy] = pd * math.log(pd)
        dict_of_kinds[key] = len(dict_of_kinds[key]) / len(train_set) * (-
sum(list(dict_of_acct.values())))
    return sum(list(dict_of_kinds.values()))
```

3. 计算信息增益：

```
def informatin_gain(train_set, D, A):

    '''计算信息增益'''

    return empirical_entropy(train_set, D) - condition_entropy(train_set, A, D)
```

4. 计算信息增益率：

```
def information_gain_ratio(train_set, D, A):

    '''计算信息增益率'''
```

```
    return informatin_gain(train_set, D, A) / empirical_entropy(train_set, A)
```

5. 计算 GINI 指数：

```
def gini_index(train_set, D, A, set):
    '''计算特征 A 的条件下，标签 D 的 GINI 指数'''
    dict_of_kinds = {0: [], 1: []}
    for i in range(len(train_set)):
        if train_set[i][A] in set:
            dict_of_kinds[0].append(i)
        else:
            dict_of_kinds[1].append(i)
    for key in dict_of_kinds.keys():
        dict_of_acct = {}
        for j in dict_of_kinds[key]:
            label = train_set[j][D]
            if label not in dict_of_acct.keys():
                dict_of_acct[label] = 1
            else:
                dict_of_acct[label] += 1
        summ = len(dict_of_kinds[key])
        for kk in dict_of_acct.keys():
            pd = dict_of_acct[kk] / summ
            dict_of_acct[kk] = pd ** 2
        dict_of_kinds[key] = len(dict_of_kinds[key]) / len(train_set) * (1 -
sum(list(dict_of_acct.values())))
    return sum(list(dict_of_kinds.values()))
```

6. 定义节点类

```
class decision_node:
    '''定义节点类'''
    def __init__(self, col, value=None, child_node=None):
        self.col = col
        self.value = value
        self.child_node = child_node
```

7. 构建决策树

```
def build_tree(self, remain_set, used_col, function):
    '''构建决策树'''
```

```
    if self.is_same(remain_set):
        '''数据集 D 中的样本属于同一类别 C，则将当前结点标记为 C 类叶结点'''
        return decision_node(-1, value=remain_set[0][self.D])

    if len(used_col) == len(self.dict_of_labels.keys()):
        '''特征集 A 为空集，或数据集 D 中所有样本在 A 中所有特征上取值相同，则将当前结点标记为叶结点，类别为 D 中出现最多的类'''
        return decision_node(-1, value=self.find_mode(remain_set))

    if function == gini_index:
        '''构建 CART 模型决策树'''
        gini = {}
        gini_num = []
        for i in self.labels:
            if i not in used_col:
                child_set = get_child_set(self.dict_of_labels[i])
                child_gini = []
                for set in child_set:
                    child_gini.append(gini_index(remain_set, self.D, i, set))
                minn = min(child_gini)
                index = child_gini.index(minn)
                gini[minn] = child_set[index]
                gini_num.append(minn)
            else:
                gini[9999] = []
                gini_num.append(9999)
        '''选择 GINI 指数最小的特征作为决策点'''
        choose = gini_num.index(min(gini_num))
        left_set = gini[min(gini_num)]
        right_set = []
        for value in self.dict_of_labels[choose]:
            if value not in left_set:
                right_set.append(value)
        set = [left_set, right_set]
        new_used_col = used_col + [choose]
```

```
            child_node = {}
            for i in range(len(set)):
                new_remain_set = []
                for row in remain_set:
                    if row[choose] in set[i]:
                        new_remain_set.append(row)
                strings = ''
                for item in set[i]:
                    strings += item + '|'
                if len(new_remain_set) == 0:
         '''数据集 D 为空集，则将当前结点标记为叶结点，类别为父结点中出现最多的类'''

                    child_node[strings] = decision_node(-1,
value=self.find_mode(remain_set))
                else:
                    child_node[strings] = self.build_tree(new_remain_set, new_used_col,
function)


        else:
            '''构建 ID3 或 C4.5 模型决策树'''

            entropies = []
            for i in self.labels:
                if i not in used_col:
                    entropies.append(function(remain_set, self.D, i))
                else:
                    entropies.append(-1)
            '''选择信息增益或信息增益率最大的特征作为决策点'''

            choose = entropies.index(max(entropies))
            new_used_col = used_col + [choose]
            child_node = {}
            for label in self.dict_of_labels[choose]:
                new_remain_set = []
                for row in remain_set:
                    if row[choose] == label:
                        new_remain_set.append(row)
                if len(new_remain_set) == 0:
         '''数据集 D 为空集，则将当前结点标记为叶结点，类别为父结点中出现最多的类'''
```

```
                child_node[label] = decision_node(-1, value=self.find_mode(remain_set))
            else:
                child_node[label] = self.build_tree(new_remain_set, new_used_col,
function)

    return decision_node(choose, child_node=child_node)
```

8. 根据生成的决策树分类数据

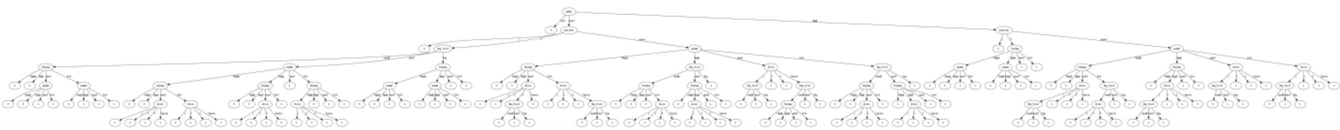
```
def classify(self, test_set):
    '''根据已生成的决策树将 test_set 中的数据分类'''

    results = []
    for row in test_set:
        head = self.decision_tree
        while head.col != -1:
            if self.function != gini_index:
                head = head.child_node[row[head.col]]
            else:
                for key in head.child_node.keys():
                    if key.find(row[head.col]) != -1:
                        head = head.child_node[key]
                        break
        results.append(head.value)
    return results
```

实验结果及分析

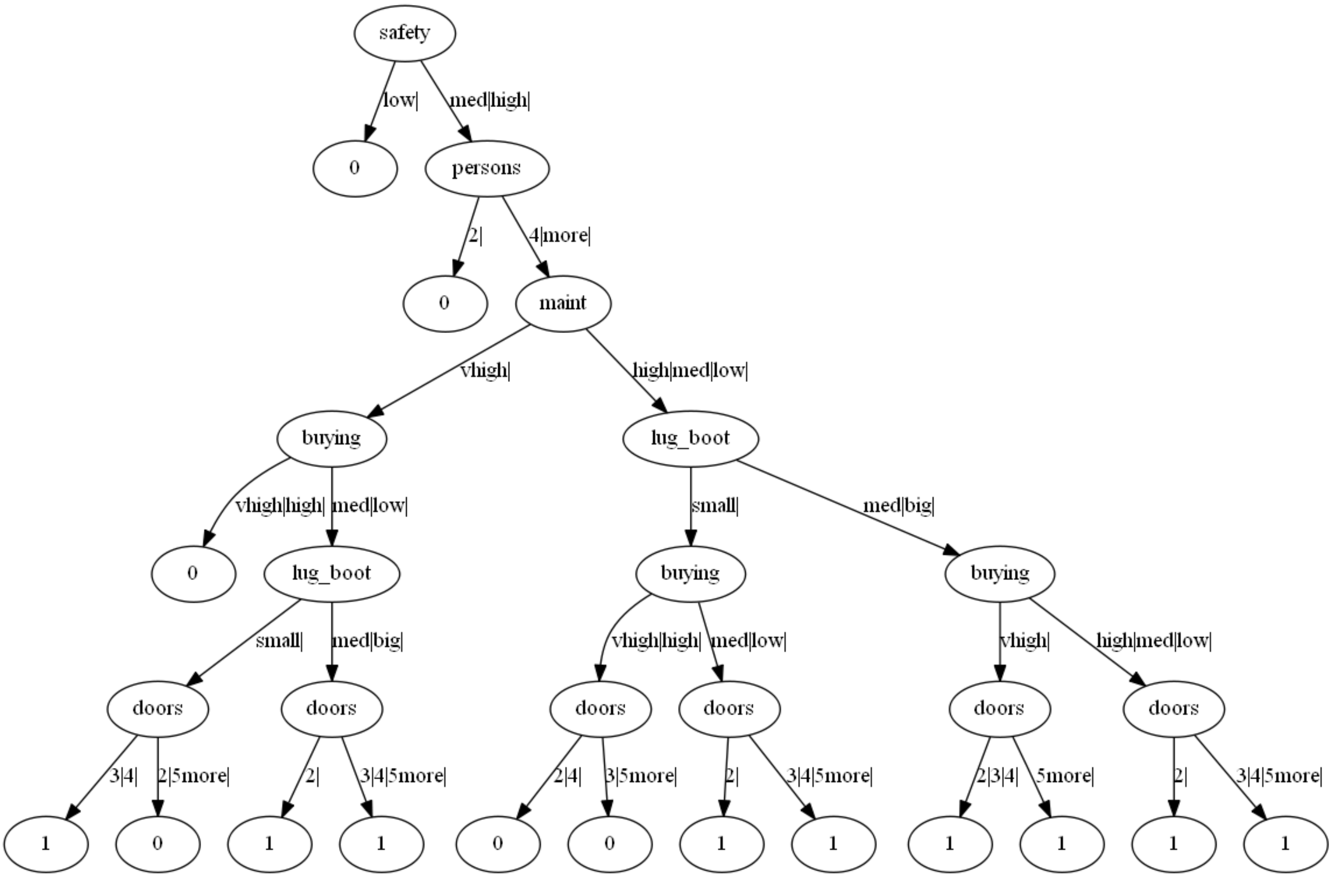
· 实验结果展示

ID3 模型决策树如图所示（高清图见文件 ID3_Decision_Tree.gv.png）:



C4.5 模型决策树如图所示（高清图见文件 C4.5_Decision_Tree.gv.png）:

CART 模型决策树如图所示（高清图见文件 CART_Decision_Tree.gv.png）：



注：ID3_Decision_Tree.gv.png、C4.5_Decision_Tree.gv.png、CART_Decision_Tree.gv.png 三个图片文件均可放大

· 评测指标展示

本次试验将 Car_train.csv 中的数据分为两类，前 1000 条数据作为训练集，其余部分作为验证集，三种模型的准确率如下：

```
ID3准确率： 0.9529983792544571
C4.5准确率： 0.9594813614262561
CART准确率： 0.9319286871961102
```

由此可以看出使用 C4.5 模型效果最佳，准确率高达 95.9%。

思考题

· 决策树有哪些避免过拟合的方法?

避免决策树过拟合的方法:

1、获取更多的数据。

2、预剪枝——在决策树生成过程中进行，对于当前的结点，判断是否应当继续划分。如果无需划分，则直接将当前结点设置为叶子结点。

3、后剪枝——先生成完整的决策树，再自底向上地对非叶结点进行考察。对于某个非叶结点,假如将它变成叶子结点,决策树在验证集上的准确率不降低,则将它变成叶子结点。

4、生成随机森林。

· C4.5 相比于 ID3 的优点是什么，C4.5 又可能有什么缺点?

C4.5 相比于 ID3 的优点:

1. 用信息增益率来选择属性,克服了用信息增益选择属性时偏向选择取值多的属性的不足;
2. 能够完成对连续属性的离散化处理;
3. 能够对不完整数据进行处理。

C4.5 的缺点:

在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。

· 如何用决策树来进行特征选择（判断特征的重要性）？

决策树最基础的算法基于贪心算法,也就是说决策树的每次特征选择均是基于当前情况选择最优的特征作为分类节点。

在 ID3 模型中，我们以信息熵和信息增益作为衡量标准，由于数据集本身的信息熵是一定的，所以只要计算各种特征对数据集的条件熵，然后找到信息增益最大的特征作为分类

节点即可。

在 C4.5 模型中，我们以信息增益率作为衡量标准选择最优的特征。在 ID3 模型中，特征的取值越多，条件熵越小，这样将会使得 ID3 模型偏向于选择取值较多的特征，导致模型的泛化性能下降。而使用信息增益率可避免该问题，我们使用信息增益率最大的特征作为决策点。

在 CART 模型中，以 GINI 指数作为衡量标准，选择 GINI 指数最小的特征作为决策点。与 ID3 和 C4.5 不同的是，CART 模型的决策树是一个二叉树，所以需要将特征的各种取值排列组合，计算它们的 GINI 系数并选取最小的系数作为该特征的 GINI 系数，最后选择 GINI 系数最小的特征作为决策点。