第3章

C4.5

3.1 C4.5 算法原理

心理测试好做，但是关键这些题怎么出才准确？答案是将被测试的群体量化得当并且区别有方。有没有方法挑选最具区别性的问题？答案之一是条件熵——C4.5。

3.1.1 C4.5 算法引入

在一些杂志或者网站上，我们经常会看到一些“心理测试”，通过这些“测试”可以测试性格、测试运势等。虽然这些心理测试的可靠性没有依据，但能够起到一定的娱乐作用。表 3-1 是截取的“心理测试”片段。

表 3-1 “心理测试”片段

1. 如果两个朋友各自在你面前指责对方，你会随他们的意思，表现出对另一方的不满吗？ 是的→2 题　　不是→7 题 2. 你觉得自己是那种藏不住心事的人吗？ 是的→8 题　　不是→3 题

续表

…… 6. 朋友们都喜欢跟你聊心事，发泄情绪吗？ 是的→B　　不是→A …… 11. 与别人相处的时候，你更多地注重对方做事的一些细节，而不是他整个人的性格。这样说法对吗？ 正确→D　　不对→C
【心理类型】
A 圆形　你是个非常圆滑的人…… **B 六边形**　你非常开朗大方…… **C 三角形**　你的个性很冲动…… **D 菱形**　你的好奇心特别旺盛……

我们可以看到，这种类型的心理测试并不需要将测试题从头到尾做一遍，而是根据上一题的选项答案进行跳转，直到给出心理类型，这就是一个决策的过程。我们可以将测试题的跳转形式以树状图来表示，每一个节点表示一个题目，树的分支表示对应题目的选项答案，树的叶子节点表示测试的结果，即测试心理类型。树状图如图 3-1 所示。

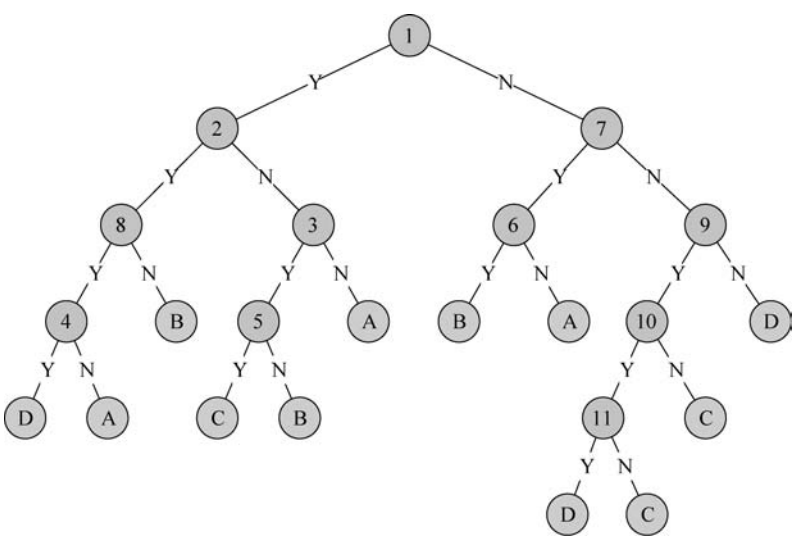

图 3-1　心理测试树状图

由图 3-1 可以看出，不同的选择代表着不同的决策过程，会得到不同的决策结果。想要设计这样的心理测试，我们除了需要收集大量的调查结果，即不同性格的人对不同问题的回答情况的数据，同时需要解决两个问题：①我们设计的心理测试题的顺序是什么，即根据上一题的选项答案应该跳转到哪一题；②在哪些心理测试题的选项答案后面设置测试结果。这其实也就是一棵决策树的设计过程。接下来，我们将围绕这两个问题来学习决策树 C4.5 算法。

3.1.2 科学问题

1. 相关理论

决策树是一种常见的分类和回归方法，本章我们主要针对分类决策树进行讨论。分类决策树描述的是一种对实例样本进行分类的树状结构，即基于特征对实例进行分类的过程，决策树算法的目的是从数据样本集中归纳出一组具有分类能力的分类规则。由于能对训练数据进行基本正确分类的决策树可能有多个或者没有，因此，我们的目标是找到与数据样本集矛盾最小的决策树。

上述示例即属于分类决策树，将上述示例中心理测试的题目、答案选项以及测试的心理类型整体看成是数据样本集，心理测试的题目看成是特征集合，测试的心理类型看成是分类类别，心理测试题的设计就是我们需要构建的决策树。示例中所需要解决的两个问题也是决策树 C4.5 算法所需解决的问题。给定数据样本集合和特征集合，我们要解决以下两个问题：

(1) 怎样选择特征来划分特征空间？从特征集合中挑选出能最大化减小数据样本集不确定性程度的特征，将其作为最优特征来划分特征空间。

(2) 如何构建决策树？递归地选择最优特征，并根据该特征对数据样本集进行分割，使得对每个子集都有一个最好的分类，得到一个与数据样本集矛盾最小的决策树。

2. 问题定义

决策树算法的输入是数据样本集合 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_M,y_M)\}$和特

征集合 $A=\{a_1,a_2,\cdots,a_N\}$。其中，M 表示数据样本集合中样本的个数；N 表示特征集合中特征的个数；$\boldsymbol{x}_m=[x_1,x_2,\cdots,x_N]$表示一个样本的特征向量，其维度为 N；y_m 表示样本的分类标签；a_n 表示一个特征，$\boldsymbol{x}_m$ 中的 x_n 表示特征 a_n 的取值。决策树算法的输出是一个与数据样本集矛盾最小的决策树 $Tree=TreeGenerate(D,A)$。

3.1.3 算法流程

1. 特征选择

通过从特征集合 $A=\{a_1,a_2,\cdots,a_d\}$中挑选出能最大化减小数据样本集不确定性程度的特征，即 $a_n^*=\arg\max_{a_n}G_{a_n}(D,a_n)$。其中 $G_{a_n}(D,a_n)$表示挑选特征 a_n 使数据样本集减少的不确定性程度。在 C4.5 算法中，我们用信息增益比来表示 $G_{a_n}(D,a_n)$。为了更好地说明信息增益比，我们先了解一下有关熵和信息增益的概念。

在信息论与数理统计中，熵表示的是对随机变量不确定性的度量，条件熵表示的是在已知某个随机变量的条件下，对另一随机变量不确定性的度量。我们分别用 $H(D)$表示数据样本集合 D 的熵，$H(D|a_n)$表示集合 D 在条件 a_n 下的条件熵。假设数据样本集合 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_M,y_M)\}$中的样本个数为 M；D 中样本的分类为 $C_k,k=1,2,\cdots,K$；属于每个类别 C_k 的样本个数为 M_{C_k}。假设特征集合 $A=\{a_1,a_2,\cdots,a_N\}$中特征的个数为 N；根据特征 a_n 的取值将 D 划分成 I 个子集，$D_1,D_2,\cdots,D_I$，子集 D_i 中的样本个数为 M_i。记子集 D_i 中属于类别 C_k 的样本集合为 D_{ik}，其样本个数为 M_{ik}。则 $H(D_i)$和 $H(D|a_n)$的计算公式如下：

$$H(D)=-\sum_{k=1}^{k}\frac{M_{C_k}}{M}\log_2\frac{M_{C_k}}{M} \tag{3-1}$$

$$H(D\mid a_n)=\sum_{i=1}^{I}\frac{M_i}{M}H(D_i)=-\sum_{i=1}^{I}\frac{M_i}{M}\sum_{k=1}^{K}\frac{M_{ik}}{M}\log_2\frac{M_{ik}}{M} \tag{3-2}$$

由于我们需要挑选出能最大化减小数据样本集不确定性程度的特征，根据熵和条件熵的概念，我们可以知道熵与条件熵之差即是数据样本集不确定性程度的减少量，我们称这个差值为信息增益，计算公式如下：

$$Gain(D,a_n)=H(D)-H(D\mid a_n) \tag{3-3}$$

$Gain(D,a_n)$表示由于特征a_n而使得对数据样本集D的分类的不确定性减小的程度，则我们应该选择的特征为信息增益最大的特征，即：

$$a_n^*=\underset{a_n}{\operatorname{argmax}}\, Gain(D,a_n) \tag{3-4}$$

如果以信息增益准则划分数据样本集，存在的问题是偏向于选择可取值数目较多的特征。为了减少这种偏好带来的负面影响，C4.5算法采用信息增益比来选择最佳划分特征。信息增益比，即信息增益$Gain(D,a_n)$与数据样本集D关于特征a_n的熵$H_{a_n}(D)$之比，计算公式如下：

$$Gain\_ratio(D,a_n)=\frac{Gain(D,a_n)}{H_{a_n}(D)} \tag{3-5}$$

$$H_{a_n}(D)=-\sum_{i=1}^{I}\frac{M_i}{M}\log_2\frac{M_i}{M} \tag{3-6}$$

根据信息增益比准则的特征选择方法是：对数据样本集D，计算每个特征的信息增益比并比较它们的大小，选择信息增益比最大的特征来划分D，即：

$$a_n^*=\underset{a_n}{\operatorname{argmax}}\, Gain\_ratio(D,a_n) \tag{3-7}$$

值得一提的是，信息增益并不是决策树选取特征的唯一方法。基尼不纯度通常也被用于决策树中特征的选取。尽管两种方法有截然不同的理论背景，但是大量实践经验表明，这两种方法的表现并没有显著差异。除了基尼不纯度之外，特征的分割度也可以用于选取特征。

2. 决策树的生成

通过递归地选择最优特征$a_n^*=\underset{a_n}{\operatorname{argmax}}\, G_{a_n}(D,a_n)$，并根据该特征对数据样本集进行分割，使得对每个子集都有一个最好的分类，得到一个与数据样本矛盾最小的决策树$Tree=CreateTree(D,A)$。

对于这一过程，具体的方法是：首先，构造根节点，将所有的数据样本都放在根节点；然后根据信息增益比准则选择一个最优特征进行数据样本集的分割，使得分割后的各个子集在当前条件下有最好的分类。如果子集中所有样本均被正确分类，则对此子集构造叶子节点；若仍有部分子集中的样本不能被正确分类，则对这部分

子集选择新的最优特征，并继续对其分割子集，构造相应的节点。对各个节点递归地调用上述方法，直到所有数据样本都能被正确分类或者所有特征都已被使用。最后生成了一棵决策树，数据样本集中每个样本都被分到对应的叶子节点中。当特征数量远远超过构建决策树所需特征数时，在构建决策树之前可以先进行一次特征筛选，挑选出对数据样本集有足够分类能力的特征数。

3. 决策树的使用

上述内容主要介绍了如何从原始数据样本集中创建决策树，下面将重点放在如何利用决策树进行数据分类上。

依靠训练数据创建决策树以后，利用这棵决策树以及特征集可对测试数据进行分类，决策树的使用同生成过程类似，依旧是一个递归过程。对每一个测试样本，比较样本在决策树中特征节点上的取值，从而跳转到下一个节点，递归执行该过程，直到跳转到叶子节点，最后将测试数据类别定义为该叶子节点所属类别。由于递归构造决策树的过程很耗时，为了提高时间性能，需要将创建好的决策树存储成一个对象，在对测试样本每次执行分类时调用已经创建好的决策树直接进行分类。

3.1.4 算法描述

根据上述算法流程，决策树 C4.5 算法的核心思想是：在决策树的各个节点上使用信息增益比准则来选择特征，递归地构建决策树。在 C4.5 算法中，有三种情形会导致递归返回：①当前节点中的所有样本都具有相同类别，则停止划分，递归返回；②当前没有剩余特征可供划分，则停止划分，递归返回；③当前节点中的数据样本集合为空，则停止划分，递归返回。算法描述如下：

算法 3-1 C4.5 算法。

输入：数据样本集合 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_M,y_M)\}$；

特征集合 $A=\{a_1,a_2,\cdots,a_N\}$；

过程：$CreateTree(D,A)$

1： 生成树节点 node；
2： **for** A 的每一个未被用来划分数据集的特征 a_n **do**
3： 按照式(3-5)计算信息增益比 $Gain\_ration(D,a_n)$，并比较它们的大小
4： 保存信息增益比最大的特征为最优特征 a_*
5： **end for**
6： **if** D 中所有样本属于同一类别 C_k **then**
7： 将 node 标记为 C_k 类别的叶子节点；**return**
8： **end if**
9： **if** $A=\varnothing$ **or** D 中样本在 A 上取值完全相同 **then**
10： 将 node 标记为叶子节点，其类别标记为包含样本数最多的类 $(C_k)_{\max}$；**return**
11： **end if**
12： **if** $D=\varnothing$ **then**
13： **return**
14： **else**
15： 根据 a_* 的每一个取值 a_*^i 分割样本集 D，得到样本子集 D_i
16： 调用 $CreateTree(D_i,A\backslash\{a_*\})$ 生成分支节点
17： **end if**
输出：以 node 为根节点的一棵决策树

3.1.5 补充说明

在决策树的学习中将已生成的树进行简化的过程称为剪枝。为了尽可能地正确分类样本，节点的划分过程可能会不断重复，有时会造成决策树的分支过多，以至于将一些不具有分类能力的特征用户划分为数据样本集，从而导致过拟合，降低分类的准确性。因此，可以通过主动剪掉一些分支来降低过拟合的影响。决策树剪枝的一种方法是通过极小化决策树整体的损失函数来实现。通过递归地从树的叶子节点向上回缩，比较叶子节点回缩到父亲节点之前和之后的决策树的损失函数值；若损失

函数值减小，则进行剪枝并将其父亲节点作为新的叶子节点；重复上述过程直到不能剪枝为止，则可以得到损失函数最小的决策树。

3.2 C4.5 算法实现

3.2.1 简介

对于 C4.5 算法的实现，我们采用 Python 语言进行。算法的实现流程如图 3-2 所示，具体的类和文件的描述如表 3-2 所示。

表 3-2 类和文件的描述

类和文件	描　　述

Example	（描述数据样本集合中的数据样本点）
	私有变量：
	`feature_list　　# 特征列表`
	`feature_index　　# 所属类别索引`

Node	（描述决策树中的树节点）
	私有变量：
	`feature_name　　# 划分树节点的特征取值`
	`feature_index　　# 划分树节点的特征`
	`data_list　　# 树节点存储的数据`
	`children_list　　# 孩子列表`
	`type　　# 节点分类(只有叶子节点有节点分类)`

file_operate.py	（描述文件的读入）
	函数：
	`def load_data(data_path, label):`
	`　　""" 读取数据样本集 """`

续表

类和文件	描　　述

algorithm_util.py	（描述算法的工具）
	函数：
	`def data_processing(train_rate, data_list):`
	`"""将数据集按比例分配给训练样本列表和测试样本列表"""`

Configuration	（描述工程的配置）
	成员变量：
	`DATA_PATH = "../data/nursery.data"　　# 数据存放路径`
	`TRAIN_RATE = 0.7　　# 训练集比例`

DecisionTree	（描述决策树 C4.5 算法）
	函数：
	`def creat_tree(node):`
	`"""构造决策树"""`
	`def test(test_list, classifier):`
	`"""测试决策树分类器"""`
	`def select_best_feature(temp_data_list):`
	`"""选择最佳划分方式"""`
	`def get_info_gain_ratio(temp_data_list, feature_index):`
	`"""计算给定特征的信息增益比"""`
	`def calc_ent(temp_data_list):`
	`"""计算给定数据集的熵"""`
	`def get_label_count(temp_data_list):`
	`"""获取给定数据集可能分类的出现个数"""`
	`def get_feature_key_map(temp_data_list, feature_index):`
	`"""获取给定特征的不同取值"""`
	`def get_split_data_list(temp_data_list, feature_index, feature_name):`
	`"""依据给定特征的取值划分数据集"""`

续表

类和文件	描　　述

DecisionTree	def judge_whole_cnt(node):
	　　""" 判断子节点的样本类别是否为同一分类 """
	def judge_feature_used(node):
	　　""" 判断分类特征是否用完 """
	def majority_vote(temp_data_list):
	　　"""
	　　如果所有分类特征都被用完，则采用多数表决方法决定节点类别
	　　"""
	def print_tree_path(node):
	　　""" 打印决策树路径(先序) """
	def judge_type(data, node):
	　　""" 判断一条测试数据的类别 """
	def calc_accuracy(truth, prediction):
	　　""" 计算决策树分类器的准确率 """

main. py	(算法的入口) 主函数： def main():

Example 用来描述数据，数据是指一些具有多种特征及标签的样本。Node 用来描述决策树中的树节点，用于存储树结构。file_operate. py 文件用来进行文件的读入操作。algorithm_util. py 用来描述算法工具，包括将数据集划分成训练集和测试集。Configuration 用来描述算法的相关配置，如读入的文件路径和训练集的比例。DecisionTree 用来描述决策树 C4. 5 算法，包括特征选择以及决策树的生成。main. py 是算法的入口，主函数在该文件中。

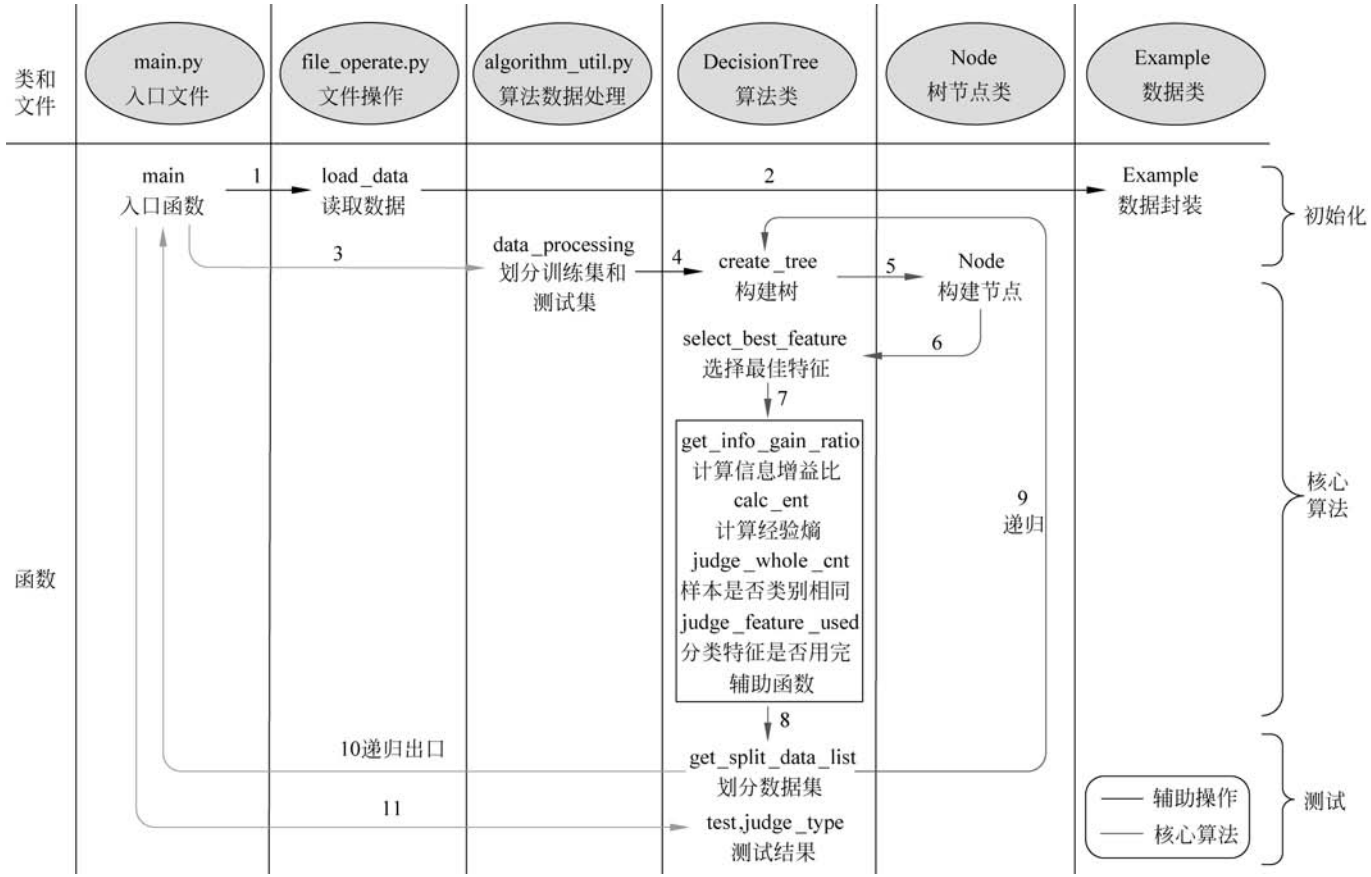
类和
文件
main.py
入口文件
file_operate.py
文件操作
algorithm_util.py
算法数据处理
DecisionTree
算法类
Node
树节点类
Example
数据类
函数
main
入口函数
1
load_data
读取数据
2
Example
数据封装
初始化
3
data_processing
划分训练集和
测试集
4
create_tree
构建树
5
Node
构建节点
6
select_best_feature
选择最佳特征
7
get_info_gain_ratio
计算信息增益比
calc_ent
计算经验熵
judge_whole_cnt
样本是否类别相同
judge_feature_used
分类特征是否用完
辅助函数
9
递归
核心
算法
8
10递归出口
get_split_data_list
划分数据集
11
test,judge_type
测试结果
辅助操作
核心算法
测试

图 3-2　算法设计流程图

3.2.2 核心代码

DecisionTree 类中有计算熵的方法，根据式(3-1)计算熵，具体如下：

```
def calc_accuracy(self, truth: list, prediction: list):
    """
    计算决策树分类器的准确率
    :param truth: 真实分类
    :param prediction: 预测分类
    :return: 准确率
    """
    # 准确率
    if len(truth) == 0:
        return 0
    else:
        return sum(map(lambda x, y: 1 if x == y else 0, truth, prediction)) /
                        len(truth)
```

在 DecisionTree 类中，计算过熵和条件熵后，根据式(3-6)计算一个特征 a_i 的熵，然后根据式(3-7)计算并选择信息增益比最大的特征作为划分数据集的特征。具体如下：

```
def get_info_gain_ratio(self, temp_data_list: list, feature_index: int):
    """
    计算给定特征的信息增益比
    :param temp_data_list:临时数据集
    :param feature_index:特征索引
    :return:给定特征的信息增益比
    """
    # 计算熵
    ent = self.calc_ent(temp_data_list)
    condition_ent = 0
    feature_ent = 0
    feature_key_map = self.get_feature_key_map(temp_data_list, feature_index)
    for feature_name in feature_key_map:
```

```
            split_data_list = self.get_split_data_list(temp_data_list,
                                feature_index, feature_name)
            p = len(split_data_list) / len(temp_data_list)
            # 计算划分后数据集的条件熵
            split_data_list_ent = p * self.calc_ent(split_data_list)
            # 计算划分数据集的特征值的熵
            feature_ent_for_one = -p * math.log(p) / math.log(2.0)
            condition_ent += split_data_list_ent
            feature_ent += feature_ent_for_one
        # 计算信息增益比
        if (ent - condition_ent) == 0:
            info_gain_ratio = 0
        else:
            info_gain_ratio = (ent - condition_ent) / feature_ent
        return info_gain_ratio

```

在 DecisionTree 类中，最主要的函数是递归生成决策树，具体如下：

```
    def create_tree(self, node: model.Node):
        """
        构造决策树
        :param node: 树节点
        :return:
        """
        # 选择最优划分特征
        best_feature = self.select_best_feature(node.data_list)
        # 当前节点中包含的样本完全属于同一类别，无须划分(递归返回出口 1)
        if self.judge_whole_cnt(node):
            type = node.data_list[0].feature_index
            node.type = type
            return
        # 当前特征集合为空，无法划分(递归返回出口 2)
        if self.judge_feature_used(node):
            # 取样本数最多的类别
            type = self.majority_vote(node.data_list)
            node.type = type
```

```
            return
        # 当前节点包含的样本集合为空,不能划分,此时不会生成新的节点(递归
                                返回出口 3)
        if len(node.data_list) == 0:
            return
        else:
            children_list = []
            feature_key_map = self.get_feature_key_map(node.data_list, best_feature)
            for feature_name in feature_key_map:
                data_list = self.get_split_data_list(node.data_list,
                                                     best_feature, feature_name)
                # 删除用过的属性,用"null"替代
                for data in data_list:
                    data.feature_list[best_feature] = "null"
                child_tree = model.Node(feature_name, best_feature, data_list)
                # 递归
                self.create_tree(child_tree)
                children_list.append(child_tree)
            node.children_list = children_list

```

决策树构造完成后,对新给出的样本数据进行测试并判别分类,具体如下：

```
    def judge_type(self, data: model.Example, node: model.Node):
        """
        判断一条测试数据的类别
        :param data: 一条测试数据
        :param node: 树节点
        :return: 数据类别
        """
        # 如果是叶子节点,则返回叶子结点类别
        if node.type is not None:
            return node.type
        # 遍历节点的孩子列表,找到对应特征以及对应的值
        else:
            feature_list = data.feature_list
            children_list = node.children_list
```

```
15            for child_node in children_list:
16                feature_index = child_node.feature_index
17                feature_name = child_node.feature_name
18                if feature_list[feature_index] == feature_name:
19                    return self.judge_type(data, child_node)
20
```

3.3 实验数据

我们的实验数据选自 UCI 数据库(网址链接:http://archive.ics.uci.edu/ml/),数据集"nursery.data"是一个幼儿园数据集(数据下载地址:http://archive.ics.uci.edu/ml/machine-learning-databases/nursery/nursery.data)。该数据集包含 12 960 个入学儿童的自身及家庭状况以及是否推荐他们入学,其具体统计信息如表 3-3 所示。

表 3-3 数据集统计信息

parents	usual, pretentious, great_pret
has_nurs	proper, less_proper, improper, critical, very_crit
form	complete, completed, incomplete, foster
children	1, 2, 3, more
housing	convenient, less_conv, critical
finance	convenient, inconv
social	non-prob, slightly_prob, problematic
health	recommended, priority, not_recom
5 classes	not_recom, recommend, very_recom, priority, spec_prior

3.4 实验结果

3.4.1 结果展示

使用以上数据集对 C4.5 算法进行测试,抽取的训练数据集和测试数据集的比例

为 7∶3，运行结果如表 3-4 所示。

表 3-4　测试结果

生成决策树（部分）	{7—priority—>{1—critical—>{0—usual—>{4—less_conv—>{3—3—>:spec_prior}{3—2—>{2—complete—>:priority}{2—foster—>:spec_prior}{2—completed—>:priority}……{3—more—>:spec_prior}}{5—convenient—>:priority}}}}{7—not_recom—>:not_recom}
测试正确率	0.974022633744856
运行时间	546ms

3.4.2　结果分析

观察上述数据集的实验结果可发现，通过决策树构建的分类规则直观且易于理解，程序运行时间较短且正确率较高，较好地表现出 C4.5 算法计算速度快和准确性较高的特点。

然而在有噪声的情况下，C4.5 算法会对训练数据完全拟合，产生过拟合现象。对训练数据的完全拟合反而不具有很好的预测性能，会降低对测试数据的分类准确性。为了克服过拟合问题，决策树的剪枝是解决该问题的主要手段。