

中山大学移动信息工程学院本科生实验报告

(2017年秋季学期)

课程名称：人工智能

年级	专业方向	学号	姓名
1501	移动（互联网）	15352005	蔡景皓

一、实验题目

- 决策树

二、实验内容

1. 算法原理

- 决策树 (Decision Tree)

- 决策树是一种常见的基于机器学习的分类方法，与KNN分类和NB分类同属于有监督的机器学习模型
- 决策树和PLA等算法是大类别的区别
- 决策树是一种树形模型，可以实现非线性分割。它对每一个特征做一个划分，相较于线性分类器，决策树可以实现非线性分割，且树形模型更加接近人的思维方式，可以产生可视觉化的分类规则
- 而线性分类器（包括感知机，线性回归，逻辑回归）是所有特征权重相加得到一个新的值，并通过阈值进行划分。线性分类器只能实现线性分割（除非对x进行多维映射）

- 决策树代表的是对象属性与对象值之间的一种映射关系

- 树中每个节点表示一个对象属性

- 而每个节点路径则表示一个节点的属性值

- 而每个节点路径则对应于从根节点所经历的路径（对象属性的情况）所表示的对象的值。

- 决策树通常用ID3,CART, Quest和C4.5等算法进行构建：

- CART (Classification And Regression Trees) 是结合分类树分析和回归树分析的一种算法，它基于GINI系数进行决策选择属性
- C4.5算法是C4.5算法的商业化版本，因此该算法因版权问题尚未公开。
- ID3 (Iterative Dichotomiser 3) 是针对离散属性取值的一种算法，它根据信息增益进行决策属性选择。
- C4.5选择ID3的一种优化算法，它根据信息增益率进行决策选择属性，且适用于对连续数据的处理（能够通过信息增益率选择最佳的离散划分）

- 本次实验使用ID3, C4.5, CART进行建树

- 决策树的建树思想

- 其实就是寻找最纯净的划分方法，即纯度最大，使目标变量要分得足够开，如ID3使用信息增益作为不纯度、C4.5使用信息增益率作为不纯度。

- 决策树建树过程中，每次都选取最优的属性作为划分，采取的是一种自顶向下的贪心策略

- 0. 以资料属性为根节点

- 1. 遍历每个决策条件，对当前结果集进行划分

- 2. 做单因子变量分析，找出信息量最大的变量作为分割（使用上述的算法进行分析）

- 3. 递归执行1、2步，直至达到边界条件，标记为叶子节点

- 1. 结果集中所有的数据属于同一个label，预测结果为这个label

- 2. 当前特征集为空集，预测结果为结果集中的众数label

- 3. 结果集中所有特征中特征的取值相同时，预测结果为结果集中的众数label

- 0. 决策树的优化

- 目的：为了提升决策树的泛化性能（对新鲜样本的适应能力）与解决对训练集过拟合问题

- 决策树的主要参数都是防止过拟合的，主要有两个参数：叶子节点的样本数和深度

- 叶子节点样本数：经验值必须大于100，如果一个节点没有100个样本支持它的决策，一般被认为是过拟合的

- 深度：控制决策树的规模，深度太深一方面会导致过拟合，另一方面对于我们的是有难度的

- 剪枝

- 原因：为了尽可能正确的分类训练样本，节点划分过程将不断重复，有时会分成决策树分支太多，这时就可能因训练样本的太过于复杂，以至于把训练集自身的一些特点当作所有数据的特点而导致过拟合。

- 剪枝是决策树学习算法减少“过拟合”情况的主要手段。

- 剪枝：

- 过程：在树的生长过程中设定一个指标（如决策树深度、纯度等），当达到该指标时就停止生长。

- 缺点：容易产生“视界局限”，一旦停止分支，断绝了其后继续进行“好”分支操作的任何可能性。

- 后剪枝

- 过程：首先让决策树充分生长，因而在避免“视界局限”，然后对所有粗俗的成对叶节点考虑是否消去它们，如果消去会引起令人满意的結果（如错误率、不纯度、模型复杂度等），那么继续消去，令它们的公共父节成为新叶节点

- 优点：克服“视界局限”效应

- 缺点：计算代价大，在大批样本集中不适用

- 随机森林

- 原因：尽管有剪枝的方法，但一棵树的生成肯定不如多棵树，因此就有了随机森林

- 随机森林可以解决决策树泛化能力弱的缺点

- 过程：

- 1. 样本的随机：从训练集中随机选取n个样本（未划分子集的训练集）

- 2. 特征的随机：从所有属性中随机选取k个属性，建立决策树（K的经验值为 \sqrt{A} ：A是属性的个数）

- 3. 重复以上两步m次，从m棵决策树中选取最佳的一棵

- 4. 随机森林的训练：每棵树的样本通过m次辨别，并进行投票表决

- 随机森林的调参有：棵数n、最大深度（经验值不超过8层）等

• 信息熵和信息增益

◦ 信息熵

- 信息熵是信息论中的基本概念

- 在实际应用前，信息熵对于信息会发出什么信息不可知道，称为信息熵对信息状态具有不确定性，由于这种不确定性是发生在通知之前的，故称为先验不确定性。在收到信息后后验不确定性，如果先前通知后验不确定性，则表示信息量为零；如果后验不确定性等于零，则表示信息收到了确定性的全部信息。

- 所以信息熵是指对不确定性的消除

- 信息量由消除的不确定性决定

- 表达式定义为： $-\log_2 p$

- 信息量单位是bit

- 信息熵是信息的字典期量

- 于1948年由香农引入，将其定义为随机事件出现的概率

- 一个系统越是有序，信息熵就越低，反之一个系统越是混乱，它的信息熵就越高。所以信息熵可以被看作是系统有序程度的一个衡量量

◦ 信息增益

- 信息熵又称先验熵，是在信息发送前信息量的数学期望

- 后验熵指在信息发送后，信息熵对信息量的数学期望

- 一般情况下后验熵大于先验熵，先验熵与后验熵偏差，即所谓的信息增益

- 信息增益，反映的是信息增阻对不确定性的程度

• ID3算法

- ID3算法的核心思想是以信息增益来度量属性的选择，选择分裂后信息增益最大的属性进行分裂

◦ ID3算法需要计算的两个熵值

- 原始信息熵：所有样本中各种label出现的不确定性之和。根据熵的概念，熵越大，不确定性就越大，如果信息熵越小，则表示该属性越纯。

- 属性熵：属性的不确定性。如果属性本身不确定很大（每个样本有一个id），则表示该属性的不确定性也很大。属性的不确定性越小，表示该属性越纯。

◦ 算法过程：

- 1. 计算数据集D的原始信息熵（本次实验中只有两种label，所以求和公式只有两项）

- $H(D) = -\sum_{d \in D} p(d) \log_2 p(d)$

- 2. 遍历所有属性，选出信息增益最大的属性作为决策点（划分）

- 1. 计算属性X对数据集D的条件熵（x是属性X的所有取值）（本次实验中只有两种label，所以D的求和公式只有两项）

- $H(D|X) = \sum_{x \in X} p(x) H(D|X=x)$

- $= \sum_{x \in X} p(x) \left[-\sum_{d \in D_x} p(d|x) \log_2 p(d|x) \right]$

- 2. 计算信息增益

- $Gain(D, X) = H(D) - H(D|X)$

◦ 决策点：

- 只考虑属性变量是离散型

- 会偏向选择取值多的属性

- 如：当属性id为连续类型的属性时，计算出条件熵为0，根据程序发现是最优决策点，然而实际上毫无意义

• C4.5算法

◦ C4.5算法是基于ID3算法进行改进的一种算法

- 使用信息增益率选择属性，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

- 能够适应非离散数据（能够通过信息增益率选择最佳的离散划分）（本次实验没有用到）

◦ 分类信息度量 (SplitInfo)

- 用分类信息来考虑某种属性进行分裂时分支的数量信息和尺寸信息，我们把这些信息称为属性的内信息

- 信息度量公式： $\frac{1}{N} \sum_{i=1}^N \frac{1}{S_i} \log_2 \frac{1}{S_i}$

- 信息增益率（属性的重要性）会随着内在信息的增长而减小（也就是说，如果这个属性本身不确定性就很大，那就越不倾向选择它）

- 对于上文ID3的缺点，如果出现id类的属性，由于这个属性本身不确定很大（每个样本有一个id），会导致信息量很大，从而导致信息增益率减小，程序就不倾向选择

- 由此可以看出，引入分裂信息得到的信息增益率对单纯使用信息增益有所补偿

◦ 算法过程：

- 1. 计算数据集D的原始信息熵（本次实验中只有两种label，所以求和公式只有两项）

- $H(D) = -\sum_{d \in D} p(d) \log_2 p(d)$

- 2. 遍历所有属性，选出信息增益最大的属性作为决策点（划分）

- 1. 计算属性X对数据集D的条件熵（x是属性X的所有取值）（本次实验中只有两种label，所以D的求和公式只有两项）

- $H(D|X) = \sum_{x \in X} p(x) H(D|X=x)$

- $= \sum_{x \in X} p(x) \left[-\sum_{d \in D_x} p(d|x) \log_2 p(d|x) \right]$

- 2. 计算信息增益

- $Gain(D, X) = H(D) - H(D|X)$

◦ 决策点：

- 在构造树的过程中，需要多次对数据集进行多次的顺序扫描和排序，导致算法的低效（ID3亦是同样的缺点）

• CART (Classification And Regression Tree) 算法

◦ CART算法使用GINI指数进行决策选择属性

◦ GINI指数

- 定义：在样本集合中一个随机选中的样本被分错的概率，即

- 基尼指数 = $1 - \sum_{i=1}^k p_i^2$ 其中 p_i 是第i类的样本数 / 总样本数

- 公式： $Gain(p) = 1 - \sum_{i=1}^k p_i^2$ 其中 p_i 是第i类的样本数 / 总样本数

- CART算法使用GINI指数作为划分指标的原因：（引用自知乎）

- ID3算法使用了信息增益作为划分特征，信息增益大的优先选择。

- C4.5算法中，采用了信息增益率来选择特征，解决ID3容易选择特征值多的特征。

- 但是无论ID3还是C4.5，都是基于信息论的模型的，这里会涉及大量的对数运算。能不能简化模型同时又不至于完全丢掉模型的优点呢？有！GART分类树算法使用基尼系数来代替信息增益比

- 基尼系数 = $1 - \sum_{i=1}^k p_i^2$ 其中 p_i 是第i类的样本数 / 总样本数

- 信息增益比 = $\frac{1}{N} \sum_{i=1}^N \frac{1}{S_i} \log_2 \frac{1}{S_i}$ 其中 S_i 是第i类的子集大小

- 信息增益 = $Gain = \sum_{i=1}^N p_i \cdot Gain_i$ 其中 $Gain_i = \sum_{j=1}^{N_i} p_{ij} \log_2 \frac{p_{ij}}{p_i}$ 其中 N_i 是第i类的样本数

- 信息增益率 = $SplitInfo = \sum_{i=1}^N p_i \cdot SplitInfo_i$ 其中 $SplitInfo_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{1}{S_{ij}} \log_2 \frac{1}{S_{ij}}$ 其中 S_{ij} 是第i类的第j个子集大小

- 信息增益 = $Gain / SplitInfo$

- 缺点：

- 只考虑属性变量是离散型

- 会偏向选择取值多的属性

- 如：当属性id为连续类型的属性时，计算出条件熵为0，根据程序发现是最优决策点，然而实际上毫无意义

- 根据决策树预测

```
void predict(const vector<vector<int>> &data) {
    Predict.clear();
    int Size = data.size();
    Node* p=root;
    for (int i=1; i<Size; i++) {
        if (p->isleaf()){
            child = NULL;
            for (int j=0; j<p->children_number; j++)
                if (data[i][j] == p->children[j]->attributeValue)
                    child = p->children[j];
            break;
        }
        if (child==NULL) p = child;
        else Predict.push_back(p->mostlabel);
        break;
    }
    if ( p->isleaf) Predict.push_back(p->ans);
}
```

- 随机森林算法

```
void random_forest() {
    vector<vector<int>> train;
    train.push_back(Train.begin()); Train.end();
    for (int i=0; i<train.size(); i++) {
        vector<vector<int>> test;
        int Size = train[i].size();
        for (int i=0; i<Size; i++) test.push_back(train[i]);
        Attribute* tree;
        Attribute* root;
        tree = new Attribute();
        root = tree->root();
        for (int i=0; i<10; i++) {
            int attr = rand()%(Attribute::attrNum);
            if (attr == root->attribute) {
                Attribute* tree1 = tree->choose(attr);
                break;
            }
        }
        recursive(root,tree);
    }
}

void predict(const vector<vector<int>> &data) {
    Predict.clear();
    Node* p=root;
    Node* child;
    for (int i=0; i<data.size(); i++) {
        for (int j=0; j<data[i].size(); j++) {
            if (p->isleaf) {
                child = NULL;
                for (int k=0; k<p->children_number; k++)
                    if (data[i][j] == p->children[k]->attributeValue)
                        child = p->children[k];
                break;
            }
            if (child==NULL) p = child;
            else Predict.push_back(p->mostlabel);
            if (p->mostlabel > 0) pos++;
            else neg++;
            break;
        }
        if (p->isleaf) {
            if (pos > neg) Predict.push_back(1);
            else Predict.push_back(-1);
        }
    }
}
```

4. 创新点&优化

- 使用random.shuffle()对数据集随机洗牌，再抽取3/4作为训练集，1/4作为验证集
- 处理多属性取值的属性时，当属性取值个数超过10种时，该属性所有值都除以 \log_2 (取值个数)
- 利用随机森林

三、实验结果及分析

1. 实验结果展示示例（使用小数据集）

- 训练集：

age	income	student	credit_rating	buy_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

- 转化为输入格式的训练集

```
0,2,0,0,-1
0,2,0,1,-1
1,2,0,0,1
2,1,0,0,1
2,0,1,0,1
2,1,0,1,-1
1,0,1,1,1
0,1,0,0,-1
0,0,1,0,1
2,1,1,0,1
0,0,1,1,1
1,1,0,1,1
1,2,1,0,1
2,1,0,1,-1
```

- 验证集：对所有属性进行全排，并添加几个不存在的属性值，label列写出正确答案

- 可以划分训练集，并画出决策树

- age + student

age	income	student	credit_rating	buy_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
<=30	medium	no	fair	no
age	income	student	credit_rating	buy_computer
1---	1----	1-----	1-----	1-----
<=30	low	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	high	yes	fair	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes

- age + credit_rating

age	income	student	credit_rating	buy_computer
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	medium	yes	fair	yes
age	income	student	credit_rating	buy_computer
1---	1----	1-----	1-----	1-----
<=30	medium	yes	excellent	yes
31...40	high	yes	fair	yes

- 决策树



- 执行程序，打印出决策树与验证集的准确率

- 叶子节点输出属性列数，叶子节点输出属性值*label

```
[1,1,2,1,1,0,1]
```

```
accuracyrate: 1
```

- 可以看到与画出的决策树相同，且对全排后的验证集准确率为1

- 打印树代码

```
1 void printTree(Node* root){
2     Node* p ;
3     int curDepth=0 ;
4
5     queue<Node*> que,coutQue ;
6     que.push(root) ;
7
8     while ( !que.empty() ){
9         p = que.front(); que.pop() ;
10
11         if ( p->children!=NULL )
12             for ( int i=0 ; i<p->children_number ; i++ )
13                 que.push(p->children[i]) ;
14
15         coutQue.push(p) ;
16     }
17     while ( !coutQue.empty() ){
18         p = coutQue.front(); coutQue.pop() ;
19         if ( curDepth!=p->depth ) cout << "\n" , curDepth++ ;
20         if ( p->isleaf ) cout << p->Attribute << "*" << p->ans ;
21         else cout << p->Attribute ;
22         cout << " " ;
23     }
24 }
```

2. 评测指标展示即分析（如果实验室题目有特殊要求，否则使用准确率）

- 使用限制最深深度为8层，随机划分3/4, 1/4分别为训练集与验证集

- 分别对ID3、C4.5与CART算法执行100次独立实验，结果如下

- 平均准确率为

ID3	C4.5	CART
0.586751	0.598985	0.600254

- 比较

- 可以看到，随机森林对于平均准确率三个模型都有所提升，且在100次测试中，准确率更是有所上升至0.7

四、思考题

1. 决策树有哪些避免过拟合的方法

- 调节决策树的参数，防止过拟合

- 对单个属性的样本数：经验值必须大于100，如果一个节点没有100个样本支撑它的决策，一般被认为是从拟合的

- 深度：控制决策树的规模，深度太深一方面会导致过拟合，另一方面对于我们的理解是有难度的

- 进行剪枝也可以防止过拟合

- 剪枝是决策树学习算法减少“过拟合”情况的主要手段。

- 弱剪枝

- 使用随机森林也能防止过拟合

- 随机森林不仅能防止过拟合，还可以解决决策树泛化能力弱的缺点

2. C4.5相比于ID3的优点是什么？

- 决策树中超常过拟节点的属性节点最重要

```
1 void printTree(Node* root){
2     Node* p ;
3     int curDepth=0 ;
4
5     queue<Node*> que,coutQue ;
6     que.push(root) ;
7
8     while ( !que.empty() ){
9         p = que.front(); que.pop() ;
10
11         if ( p->children!=NULL )
12             for ( int i=0 ; i<p->children_number ; i++ )
13                 que.push(p->children[i]) ;
14
15         coutQue.push(p) ;
16     }
17     while ( !coutQue.empty() ){
18         p = coutQue.front(); coutQue.pop() ;
19         if ( curDepth!=p->depth ) cout << "\n" , curDepth++ ;
20         if ( p->isleaf ) cout << p->Attribute << "*" << p->ans ;
21         else cout << p->Attribute ;
22         cout << " " ;
23     }
24 }
```



```
1 void printTree(Node* root){
2     Node* p ;
3     int curDepth=0 ;
4
5     queue<Node*> que,coutQue ;
6     que.push(root) ;
7
8     while ( !que.empty() ){
9         p = que.front(); que.pop() ;
10
11         if ( p->children!=NULL )
12             for ( int i=0 ; i<p->children_number ; i++ )
13                 que.push(p->children[i]) ;
14
15         coutQue.push(p) ;
16     }
17     while ( !coutQue.empty() ){
18         p = coutQue.front(); coutQue.pop() ;
19         if ( curDepth!=p->depth ) cout << "\n" , curDepth++ ;
20         if ( p->isleaf ) cout << p->Attribute << "*" << p->ans ;
21         else cout << p->Attribute ;
22         cout << " " ;
23     }
24 }
```



```
1 void printTree(Node* root){
2     Node* p ;
3     int curDepth=0 ;
4
5     queue<Node*> que,coutQue ;
6     que.push(root) ;
7
8     while ( !que.empty() ){
9         p = que.front(); que.pop() ;
10
11         if ( p->children!=NULL )
12             for ( int i=0 ; i<p->children_number ; i++ )
13                 que.push(p->children[i]) ;
14
15         coutQue.push(p) ;
16     }
17     while ( !coutQue.empty() ){
18         p = coutQue.front(); coutQue.pop() ;
19         if ( curDepth!=p->depth ) cout << "\n" , curDepth++ ;
20         if ( p->isleaf ) cout << p->Attribute << "*" << p->ans ;
21         else cout << p->Attribute ;
22         cout << " " ;
23     }
24 }
```



```
1 void printTree(Node* root){
2     Node* p ;
3     int curDepth=0 ;
4
5     queue<Node*> que,coutQue ;
6     que.push(root) ;
7
8     while ( !que.empty() ){
9         p = que.front(); que.pop() ;
10
11         if ( p->children!=NULL )
12             for ( int i=0 ; i<p->children_number ; i++ )
13                 que.push(p->children[i]) ;
14
15         coutQue.push(p) ;
16     }
17     while ( !coutQue.empty() ){
18         p = coutQue.front(); coutQue.pop() ;
19         if ( curDepth!=p->depth ) cout << "\n" , curDepth++ ;
20         if ( p->isleaf ) cout << p->Attribute << "*" << p->ans ;
21         else cout << p->Attribute ;
22         cout << " " ;
23     }
24 }
```



```
1 void printTree(Node* root){
2     Node* p ;
3     int curDepth=0 ;
4
5     queue<Node*> que,coutQue ;
6     que.push(root) ;
7
8     while ( !que.empty() ){
9         p = que.front(); que.pop() ;
10
11         if ( p->children!=NULL )
12             for ( int i=0 ; i<p->children_number ; i++ )
13                 que.push(p->children[i]) ;
14
15         coutQue.push(p) ;
16     }
17     while ( !coutQue.empty() ){
18         p = coutQue.front(); coutQue.pop() ;
19         if ( curDepth!=p->depth ) cout << "\n" , curDepth++ ;
20         if ( p->isleaf ) cout << p->Attribute << "*" << p->ans ;
21         else cout << p->Attribute ;
22         cout << " " ;
23
```