中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级		专业（方向）	**移动互联网**
学号		姓名	**Jw**

一、 实验题目

决策树 -------- Decision Tree

二、 实验内容

1. 算法原理

简介：决策树就是带有判决规则（if-then）的一种树，代表对象属性和对象值之间的一种映射关系，由结点和有向边组成。假设分类样本有 n 种属性，决策树的每一个非叶子结点表示一种特征或者属性，结点 i 到子结点 j 的边值 val 表示当属性 i 取值 val 时，下一个要根据属性 j 去分类。决策树的叶子结点表示一个分类类别。分类预测的过程可以看成从树的根一直往下走的过程，这里体现出决策树模型可读性很强，易于理解。

决策树生成的特征选择算法：由简介我们知道生成一颗决策树就是决定先选择哪种特征作为决策点。这里特征选择算法一般有 3 种：ID3、C4.5 和 CART。

ID3：

ID3 算法的思想就是用较少的东西做更多的事情，即希望生成最小的树形结构来完成决策树的构建。在信息论中，期望信息越小，那么信息增益就越大，从而纯度就越。高 ID3 算法的核心思想是以信息增益来度量属性的选择，它是通过选择分裂后信息增益最大的属性作为决策点。

在信息增益中，重要性的衡量标准就是看特征能够为分类系统带来多少信息，带来的信息越多，该特征越重要。度量这个东西的一个方法就是使用信息熵。一个系统越是有序，它的信息熵越低，反之一个系统越混乱，它的信息熵越高。信息熵定义如下：

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

其中 X 为随机变量，pi 为 X 的每种取值的概率。则上式表示 X 的信息熵。

ID3 算法步骤：

1.计算数据集 D 的熵

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

2.计算特征 A 对数据集 D 的条件熵

$$H(D|A) = \sum_{a \in A} p(a) H(D|A = a)$$

3.计算信息增益

$$g(D, A) = H(D) - H(D|A)$$

4.选择信息增益最大的特征作为决策点，这个增量就是特征给系统带来的信息量。

C4.5：

C4.5 是 ID3 算法的改进，使用最大信息增益率的特征作为决策点。

数据集 D 关于特征 A 的熵：

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

信息增益率：

$$\text{gRatio}(D, A) = (H(D) - H(D|A))/\text{SplitInfo}(D, A)$$

使用信息增益率是因为 ID3 算法利用增益来选择会偏向选择数据取值较多的特征，当除以一个 splitinfo 之后，可以削弱这种作用。

CART：

CART 算法采用 Gini 指数来度量分裂时的不纯度。采用 Gini 指数可以比熵的计算更快一些。

计算特征 A 的条件下，数据集 D 的 GINI 系数

$$\text{gini}(D, A) = \sum_{j=1}^{v} p(A_j) \times gini(D_j|A = A_j)$$

其中：

$$gini(D_j|A = A_j) = \sum_{i=1}^{n} p_i(1 - p_i) = 1 - \sum_{i=1}^{n} p_i^2$$

选择 GINI 系数最小的特征作为决策点。GINI 系数越小表示不确定性越小。

决策树分类时的特殊情况：

1.生成决策树时发现剩下的特征类别全都一样，此时不再继续划分，直接使用该标签作为叶子结点。

2.生成决策树时发现当没有特征可分下去，采用多数投票原则决定分类类别

3.预测测试样本分类时，发现没有办法继续分下去，即特征取值出现缺失值，此时采用多数投票原则决定分类类别。

2. 伪代码

建树部分：

```
traindata = getTraindata()
Features = get the features list of traindata
```

```
Set method to be ID3、C4.5 or CART
buildTree(traindata, Features, method){
   if nowIsLeaf() then
        return MooreVoting for traindata's labels
   DecisionPoint = getBestFeature with method
   Features = Features that remove the best features
   Tree = {bestFeature:{}}
   For i = 1,2,3…len(traindata)
        valueSet = valueSet | traindata[i][DecisionPoint]
   For value in valueSet:
        subdata = splitdata of traindata by the DecisionPoint axis' value
        Tree[bestFeature][value] = buildTree(subdata, Features, method)
}
```

分类部分：

```
classify(Tree, labels, testVector){
   if Tree is leaf then
         return Tree.label
   feature = Tree.root
   idx = labels.index(feature)
   value = testVector[idx]
   if Tree not has child[value] then
         return MooreVoting for traindata's labels
   return classify(Tree.child[value],labels,testVector)
}
```

3. 关键代码截图（带注释）

1.计算熵

```python
def getEntropy(traindata):
    """
    calculate the enrtopy of traindata
    :param traindata: the data that you need to calculate entropy
    :return: the entropy of traindata
    """
    classify = [x[-1] for x in traindata]
    num = len(classify)
    classify = Counter(classify)
    entropy = 0.0
    for key, value in classify.items():
        entropy -= (value/num) * log(value/num, 2)
    return entropy
```

2.计算 gini

```python
def getGini(traindata):
    """
    calcuulate the gini number of traindata
    :param traindata: the data that you need to calculate gini number
    :return: the gini number of traindata
    """
    classify = [x[-1] for x in traindata]
    num = len(classify)
    classify = Counter(classify)
    gini = 1.0
    for key, value in classify.items():
        gini -= ((value/num) * (value/num))
    return gini
```

3.选择最佳特征

```python
def getBestFeature(traindata, wayToChooseFeature):
    """
    :param traindata: traindata that waiting to be dividedy
    :param wayToChooseFeature: ID3、C4.5、CART
    :return bestFeature: the Feature that used to be the decision point
    """
    allEntropy = getEntropy(traindata)
    num = len(traindata[0]) - 1
    bestGain = 0.0
    bestGini = 10
    bestFeature = -1
    for i in range(num):
        featureData = [x[i] for x in traindata]
        valSet = set(featureData)
        nowEntropy = 0.0
        splitInfo = 0.0
        gini = 0.0
        for val in valSet:
            subData = splitData(traindata, i, val)
            p = len(subData) / len(traindata)
            if wayToChooseFeature == "ID3" or wayToChooseFeature == "C4.5":
                nowEntropy += p * getEntropy(subData)
                splitInfo -= p * log(p, 2)
            if wayToChooseFeature == "CART":
                gini += p * getGini(subData)

        if wayToChooseFeature == "ID3":
            gain = allEntropy - nowEntropy
            if gain > bestGain:
                bestGain = gain
                bestFeature = i
        elif wayToChooseFeature == "C4.5":
            if splitInfo == 0.0:
                continue
            gRatio = (allEntropy - nowEntropy) / splitInfo
            if gRatio > bestGain:
                bestGain = gRatio
                bestFeature = i
        elif wayToChooseFeature == "CART":
            if gini < bestGini:
                bestGini = gini
                bestFeature = i
    return bestFeature
```

4.建树

```python
def buildTree(traindata, labels, wayToChooseFeature):
    """
    :param traindata: data to create decision tree
    :param labels: feature name
    :param wayToChooseFeature: the way you use to choose best feature
                            it include ID3、C4.5 and CART
    :return Dtree: the decition tree you built
    """
    # 当剩下的特征的分类类别完全一样时，没必要继续分下去
    classify = [x[-1] for x in traindata]
    if classify.count(classify[0]) == len(classify):
        return classify[0]

    # 当没有特征可分下去，采用多数投票原则返回出现次数最多的标签
    if len(traindata) == 1:
        return MooreVoting(classify)

    # 选择最好的feature作为决策点，递归建树
    Dpoint = getBestFeature(traindata, wayToChooseFeature)
    label = labels[Dpoint]
    Tree = {label: {}}
    del(labels[Dpoint])
    values = [x[Dpoint] for x in traindata]
    valSet = set(values)
    for val in valSet:
        labelsCopy = labels[:]
        Tree[label][val] = buildTree(
                splitData(traindata, Dpoint, val),
                labelsCopy,
                wayToChooseFeature
        )
    return Tree
```

5. 分类是对测试数据每一行单独分类，所以分类函数每次只传进测试数据的一行

```python
def classifyVec(Tree, labels, testVec, traindata, testlabels):
    """
    :param Tree: The Descision tree
    :param labels: Labels of the testdata
    :param testdata: The data vector that need to be classified
    :param traindata: The data used to train
    :return: The classify result
    """
    root = list(Tree.keys())[0]
    choice = Tree[root]
    featureIdx = labels.index(root)
    feaidx2 = testlabels.index(root)
    del(testlabels[feaidx2])
    flag = 0
    #根据test向量feature的取值选择要走的树权
    for key in choice.keys():
        if testVec[featureIdx] == key:
            flag = 1
            if type(choice[key]).__name__ == "dict":
                labelsCopy = testlabels[:]
                subtraindata = splitData(traindata, feaidx2, key)
                result = classifyVec(choice[key], labels, testVec, subtraindata, labelsCopy)
            else:
                result = choice[key]
            break
    #如果无法继续分下去，采用多数投票原则决定类别
    if flag == 0:
        nowlabels = [x[-1] for x in traindata]
        result = MooreVoting(nowlabels)
    return result
```

4. **创新点&优化（如果有）**

一点预剪枝：

建树时当剩下的特征分类类别完全一样时，没必要继续分下去，直接使用该分类类别将当前结点变为叶子结点。。

```
# 当剩下的特征的分类类别完全一样时，没必要继续分下去
classify = [x[-1] for x in traindata]
if classify.count(classify[0]) == len(classify):
    return classify[0]
```

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

一个简单的小数据集：

Id	Feature0	Feature1	Label
Train1	1	1	1
Train2	0	1	0
Train3	1	0	0
Train4	0	0	0
Test1	1	0	?

手动计算：

ID3:

经验熵 H(D) = $-\frac{1}{4}*\log_2\frac{1}{4}-\frac{3}{4}*\log_2\frac{3}{4}=0.811278$

条件熵：

H(D|A=”Fea0”) = $\frac{2}{4}*\left(-\frac{1}{2}*\log_2\frac{1}{2}-\frac{1}{2}*\log_2\frac{1}{2}\right)+\frac{2}{4}*\left(0-\frac{2}{2}*\log_2\frac{2}{2}\right)=0.5$

H(D|A=”Fea1”) = $\frac{2}{4}*\left(-\frac{1}{2}*\log_2\frac{1}{2}-\frac{1}{2}*\log_2\frac{1}{2}\right)+\frac{2}{4}*\left(0-\frac{2}{2}*\log_2\frac{2}{2}\right)=0.5$

信息增益：

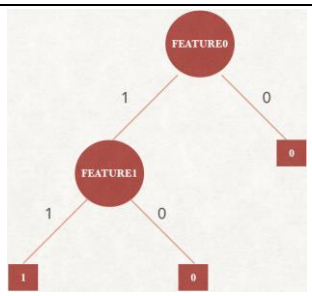
g (D|A=”Fea0”) = H(D) – H(D|A=”Fea0”) = 0.311278

g (D|A=”Fea1”) = H(D) – H(D|A=”Fea1”) = 0.311278

信息增益一样的情况下选择先出现的 feature 作为决策点

这里选择 Fea0 作为第一个决策点

可得决策树如下：



由决策树可知测试数据的分类结果是 0

C4.5:

计算每个特征的熵：

splitinfo(D|A="Fea0")= $-\frac{2}{4}*\log_2\frac{2}{4}-\frac{2}{4}*\log_2\frac{2}{4}=1$

splitinfo(D|A="Fea1")= $-\frac{2}{4}*\log_2\frac{2}{4}-\frac{2}{4}*\log_2\frac{2}{4}=1$

计算信息增益率：

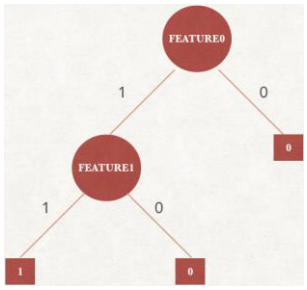
gainRatio(D|A="Fea0") = g(D|A="Fea0") / splitinfo(D|A="Fea0") = 0.311278

gainRatio(D|A="Fea1") = g(D|A="Fea1") / splitinfo(D|A="Fea1") = 0.311278

信息增益率一样的情况下选择先出现的 feature 作为决策点

这里选择 Fea0 作为第一个决策点

可得决策树如下：



由决策树可知测试数据的分类结果是 0

CART:

计算在每个特征的条件下，数据集的 GINI 系数：

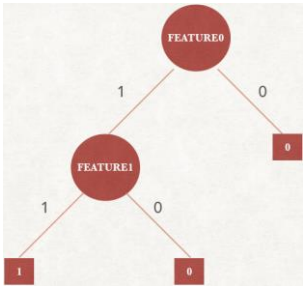
gini(D|A="Fea0") = $\frac{2}{4}*\left[1-\left(\frac{1}{2}\right)^2-\left(\frac{1}{2}\right)^2\right]+\frac{2}{4}*\left[1-\left(\frac{2}{2}\right)^2-0^2\right]=0.25$

gini(D|A="Fea1") = $\frac{2}{4}*\left[1-\left(\frac{1}{2}\right)^2-\left(\frac{1}{2}\right)^2\right]+\frac{2}{4}*[1-(1)^2-0^2]=0.25$

gini 一样的情况下选择先出现的 feature 作为决策点

这里选择 Fea0 作为第一个决策点

可得决策树如下：



由决策树可知测试数据的分类结果是 0

ID3 算法结果:

```
Algorithm: ID3
{'feature0': {'0': '0', '1': {'feature1': {'0': '0', '1': '1'}}}}
test0: 0
```

C4.5 算法结果:

```
Algorithm: c4.5
{'feature1': {'0': '0', '1': '1'}}
test0: 0
```

CART 算法结果:

```
问题  输出  调试控制台  终端
Algorithm: CART
{'feature0': {'1': {'feature1': {'1': '1', '0': '0'}}, '0': '0'}}
test0: 0
```

可以看到三种算法结果都是正确的

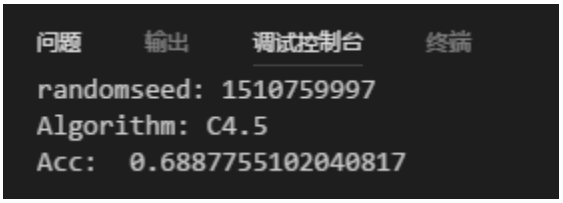
2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

我将 train.csv 的数据按照 4:1 划分成测试集和验证集，方法是随机划分，并记录下随机种子，下面的结果分别是三种算法跑了几次表现最好的结果。

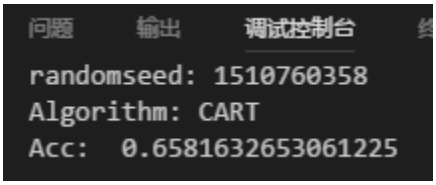
ID3：

```
问题  输出  调试控制台  终端
randomseed: 1510759881
Algorithm: ID3
Acc:  0.6581632653061225
```

C4.5：



CART：



这里 C4.5 的表现比较好。

四、 思考题

1. 决策树有哪些避免过拟合的方法？

答：1. 合理抽样，去掉训练样本中噪音大的数据

2. 预剪枝，提前停止树的增长

3. 后剪枝，对过度拟合的树进行修剪

4. 通过交叉验证来测试模型并作修改。

2. C4.5 相比于 ID3 的优点是什么？

答：1. 克服了 ID3 用信息增益选择属性时偏向选择取值多的属性的缺点。

2. 能够完成对连续属性的离散化处理。将连续值区间进行 N 等分，算 N 个等分点的增益率，选择最大增益率的那个点为连续属性的断点，并将区间以该点断开成两个区间，分别递归继续处理。

3. 能够对不完整数据进行处理，用类似连续属性离散化的方式去处理。

3. 如何用决策树来判断特征的重要性？

答：按照决策树生成时要生成最小结构的原则，越靠近根结点的特征的重要性越大。