机器学习实验六 --决策树

> 朱浩泽 1911530 计算机科学与技术

实验要求

- 基本要求
  1. 基于 Watermelon-train1数据集（只有离散属性），构造ID3决策树；
  2. 基于构造的 ID3 决策树，对数据集 Watermelon-test1进行预测，输出分类精度；
- 中级要求
  1. 对数据集Watermelon-train2，构造C4.5或者CART决策树，要求可以处理连续型属性；
  2. 对测试集Watermelon-test2进行预测，输出分类精度；

导入数据

In [ ]:

```python
import numpy as np
import pandas as pd
import math
import copy
def readfile(name):
    df = pd.read_csv(name, error_bad_lines = False, encoding = 'gbk')
    temp = np.array(df).tolist()
    for i in temp:
        i.pop(0)
    return temp

train1 = readfile("Watermelon-train1.csv")
train2 = readfile("Watermelon-train2.csv")
test1 = readfile("Watermelon-test1.csv")
test2 = readfile("Watermelon-test2.csv")
```

计算信息增益

- 统计某结果数据发生的频率，每项的信息以字典的形似存储
- 计算信息熵
- 返回信息熵和分类信息

In [ ]:

```python
def information(data):
    dic = {}
    for i in data:
        current = i[-1] #取出最后的结果
        if current not in dic.keys():
            dic[current] = 1 #创建一个新的类别
        else:
            dic[current] += 1 #原有类别+1
    result = 0.0
    for key in dic:
        prob = float(dic[key]) / len(data)
        result -= prob * math.log(prob,2)
    return result, dic
```

- 将数据按照某一种类的属性重新分类，并将该行属性删除

In [ ]:

```python
def split(data, index, kind):
    ls = []
    for temp in data:
        if temp[index] == kind:
            t = temp[0: index]
            t = t + temp[index + 1: ]
            ls.append(t)
    return ls
```

- 计算信息熵和信息增益并返回最优的分类方式，对数据的每项指标做以下的过程
  1. 抽取该项数据的所有信息
  2. 按照该项数据的类别信息将数据集划分成多个子数据集
  3. 计算每个数据集的信息熵
  4. 计算该项数据的信息增益
  5. 根据信息增益选择最好的分类项目，返回该项目的类别号

In [ ]:

```python
def chooseBestFeatureToSplit(data):
    base, mm= information(data) #原始的信息熵
    best = 0
    bestindex = -1
    for i in range(len(data[0]) - 1):
        #抽取该列数据的所有信息
        ls = [index[i] for index in data]
        feture = set(ls)
        #计算该列的信息增益
        temp = 0.0
        for value in feture:
            datatemp = split(data, i, value)
            prob = len(datatemp) / float(len(data))
            t, mm = information(datatemp)
            temp += prob * t
        infoGain = base - temp
        #根据信息增益挑选
        if infoGain > best:
            best = infoGain
            bestindex = i
    return bestindex
```

递归构建$ID3$决策树

- 决策树的生成
  1. 从根节点开始，计算所有可能特征的信息增益，选择信息增益最大的特征作为划分该节点的特征，根据该特征的不同取值建立子节点；
  2. 在对子节点递归地调用以上方法，直到达到停止条件，得到一个决策树。
- 迭代停止条件
  1. 当前结点所有样本都属于同一类别；
  2. 当前结点的所有属性值都相同，没有剩余属性可用来进一步划分样本；
  3. 达到最大树深；
  4. 达到叶子结点的最小样本数；
- 具体实现
  1. 首先取出该数据集的类别信息
  2. 统计处类别信息以及数量，如果只有一个类别，返回该类别
  3. 计算最优划分的索引
  4. 初始化子树
  5. 当前已经选择的特征不再参与分类，将该特征删除
  6. 计算剩余特征的集合
  7. 对于每个分支，进行递归
  8. 返回值为子树

In [ ]:

```python
def classify1(data, labels):
    typelist = [index[-1] for index in data] #取出该数据集的分类
    nothing, typecount = information(data) #计算出类别种类以及数量
    if len(typecount) == 1: #如果只有一个类别
        return typelist[0]
    bestindex = chooseBestFeatureToSplit(data)  # 最优划分属性的索引
    bestlabel = labels[bestindex]
    Tree = {bestlabel: {}}
    temp = labels[:]
    del (temp[bestindex])  # 已经选择的特征不再参与分类，将该类别删除
    feature = [example[bestindex] for example in data]
    unique = set(feature)  # 该属性所有可能取值，也就是节点的分支
    for i in unique:
        temp = temp[:]
        Tree[bestlabel][i] = classify1(split(data, bestindex, i), temp)
    return Tree
```

对测试集利用$ID3$决策树进行分类

- 取出当前树的根节点
- 利用跟节点信息查询当前输入数据输入内容
- 如果查询出来的分支是叶节点，返回该值
- 如果不是叶节点，递归查询子树

In [ ]:

```python
def run1(testdata, tree, labels):
    firstStr = list(tree.keys())[0]
    secondDict = tree[firstStr]
    featIndex = labels.index(firstStr)
    result = ''
    for key in list(secondDict.keys()):
         if testdata[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':  # 该分支不是叶子节点，递归
                result = run1(testdata, secondDict[key], labels)
            else:
                result = secondDict[key]
    return result

def getresult(train, test, labels):
    ls = []
    tree = classify1(train, labels)
    print("生成决策树如下:", tree)
    for index in test:
        ls.append(run1(index, tree, labels))
    return ls

labels1 = ['色泽', '根蒂', '敲声', '纹理', '好瓜']
result1 = getresult(train1, test1, labels1)
```

```
生成决策树如下: {'纹理': {'模糊': '否', '清晰': {'根蒂': {'稍蜷': '是', '蜷缩': '是', '硬挺': '否'}}, '稍糊': {'色泽': {'青绿': '否', '乌黑': {'敲声': {'沉闷': '否', '浊响': '是'}}, '浅白
': '否'}}}}
```

计算准确率

In [ ]:

```python
def simrate(data, predict):
    num = 0
    for i in range(len(data)):
        if data[i][-1] == predict[i]:
            num +=1
    return format(num / len(data), '.2%')
print("ID3分类器对于test1数据集的准确率是：", simrate(test1, result1))
```

```
ID3分类器对于test1数据集的准确率是： 70.00%
```

可以看出，用这种方法得到树分类树对于小规模数据的分类效果相对较好。

利用$C4.5$算法构建决策树

信息增益作为划分准则存在的问题：
信息增益偏向于选择取值较多的特征进行划分。比如学号这个特征，每个学生都有一个不同的学号，如果根据学号对样本进行分类，则每个学生都属于不同的类别，这样是没有意义的。而C4.5在生成过程中，用信息增益比来选择特征，可以校正这个问题。

信息增益比 = 惩罚参数 * 信息增益，即 $g_R(D,A)=\frac{g(D,A)}{H_A(D)}$，其中的$H_A(D)$，对于样本集合D，将当前特征A作为随机变量（取值是特征A的各个特征值），求得的经验熵。

信息增益比本质： 是在信息增益的基础之上乘上一个惩罚参数。特征个数较多时，惩罚参数较小；特征个数较少时，惩罚参数较大。
惩罚参数：数据集D以特征A作为随机变量的熵的倒数，即：将特征A取值相同的样本划分到同一个子集中，$\text{惩罚参数}=\frac{1}{H_A(D)}=\frac{1}{-\sum_{(i=1)}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}}$

- 将连续属性离散化
  采用二分法，把数据由小到大排列，选择每个区间的中位点位取值，左侧区间为不大于该点的值，右侧区间为大于该点的值

In [ ]:

```python
def Division(data):
    ls = data[:]
    ls.sort()
    result = []
    for i in range(len(ls) - 1):
        result.append((data[i + 1] + data[i]) / 2)
    return result
```

- 将数据按照某一种类的属性或者数值的区间重新分类；method = 0 按照区间左侧分类，method = 1 按照区间右侧分类

In [ ]:

```python
def split2(data, index, kind, method):
    ls = []
    if method == 0:
        for temp in data:
            if temp[index] <= kind:
                t = temp[0 : index]
                t = t + temp[index + 1 : ]
                ls.append(t)
    else:
        for temp in data:
            if temp[index] > kind:
                t = temp[0 : index]
                t = t + temp[index + 1 : ]
                ls.append(t)
    return ls
```

- 计算信息熵和信息增益并返回最优的分类方式，对数据的每项指标做以下的过程
  1. 抽取该项数据的所有信息
  2. 计算每一类的信息熵
  3. 按照该项数据的类别信息将数据集划分成多个子数据集
  4. 计算每个数据集的信息熵
  5. 计算该项数据的信息增益比
  6. 根据信息增益选择最好的分类项目，返回该项目的类别号
  7. 对连续型数据，计算分割值，求出信息增益比最小的点作为分割点返回

In [ ]:

```python
def chooseBestFeatureToSplit2(data):
    base, mm= information(data) #原始的信息熵
    info = []
    for j in range(len(data[0]) - 1):
        dic = {}
        for i in data:
            current = i[j] #取出最后的结果
            if current not in dic.keys():
                dic[current] = 1 #创建一个新的类别
            else:
                dic[current] += 1 #原有类别+1
        result = 0.0
        for key in dic:
            prob = float(dic[key]) / len(data)
            result -= prob * math.log(prob,2)
        info.append(result)
    best = 0
    bestindex = -1
    bestpartvalue = None #如果是离散值，使用该值进行分割
    for i in range(len(data[0]) - 1):
        #抽取该列数据的所有信息
        ls = [index[i] for index in data]
        feture = set(ls)
        #计算该列的信息增益
        temp = 0.0
        if type(ls[0]) == type("a"):#判断是否时离散的
            for value in feture:
                datatemp = split(data, i, value)
                prob = len(datatemp) / float(len(data))
                t, mm = information(datatemp)
                temp += prob * t
        else:
            ls.sort()
            min = float("inf")
            for j in range(len(ls) - 1):
                part = (ls[j + 1] + ls[j]) / 2 #计算划分点
                left = split2(data, i, part, 0)
                right = split2(data, i, part, 1)
                temp1, useless = information(left)
                temp2, useless = information(right)
                temp = len(left) / len(data) * temp1 + len(right) / len(data) * temp2
                if temp < min:
                    min = temp
                    bestpartvalue = part
            temp = min
        infoGain = base - temp
        #根据信息增益挑选
        if info[i] != 0:
            if infoGain / info[i] >= best:
                best = infoGain / info[i]
                bestindex = i
    return bestindex, bestpartvalue
```

递归构建$C4.5$决策树

- 决策树的生成
  1. 从根节点开始，计算所有可能特征的信息增益，选择信息增益最大的特征作为划分该节点的特征，根据该特征的不同取值建立子节点；
  2. 在对子节点递归地调用以上方法，直到达到停止条件，得到一个决策树。
- 迭代停止条件
  1. 当前结点所有样本都属于同一类别；
  2. 当前结点的所有属性值都相同，没有剩余属性可用来进一步划分样本；
  3. 达到最大树深；
  4. 达到叶子结点的最小样本数；
- 具体实现
  1. 首先取出该数据集的类别信息
  2. 统计处类别信息以及数量，如果只有一个类别，返回该类别
  3. 计算最优划分的索引
  4. 初始化子树
  5. 当前已经选择的特征如果是离散的便不再参与分类，将该特征删除；如果是连续的不删除该特征，以分界点构建左子树和右子树
  6. 计算剩余特征的集合
  7. 对于每个分支，进行递归
  8. 返回值为子树

In [ ]:

```python
def classify2(data, labels):
    typelist = [index[-1] for index in data] #取出该数据集的分类
    nothing, typecount = information(data) #计算出类别种类以及数量
    if typecount == len(typelist): #如果只有一个类别
        return typelist[0]
    bestindex, part = chooseBestFeatureToSplit2(data)  # 最优划分属性的索引
    if bestindex == -1:
        return "是"
    if type([t[bestindex] for t in data][0]) == type("a"):#判断是否时离散的
        bestlabel = labels[bestindex]
        Tree = {bestlabel: {}}
        temp = copy.copy(labels)
        feature = [example[bestindex] for example in data]
        del (temp[bestindex])  # 已经选择的特征不再参与分类，将该类别删除
        unique = set(feature)  # 该属性所有可能取值，也就是节点的分支
        for i in unique:
            s = temp[:]
            Tree[bestlabel][i] = classify2(split(data, bestindex, i), s)
    else: #连续的变量
        bestlabel = labels[bestindex] + "<" + str(part)
        Tree = {bestlabel: {}}
        temp = labels[:]
        del(temp[bestindex])
        leftdata = split2(data, bestindex, part, 0)
        Tree[bestlabel]["是"] = classify2(leftdata, temp)
        rightdata = split2(data, bestindex, part, 1)
        Tree[bestlabel]["否"] = classify2(rightdata, temp)
    return Tree
```

对测试集利用$C4.5$决策树进行分类

- 取出当前树的根节点
- 利用跟节点信息查询当前输入数据输入内容
- 如果是离散值
  1. 如果查询出来的分支是叶节点，返回该值
  2. 如果不是叶节点，递归查询子树
- 如果是非离散值
  1. 取出节点值与现在数据进行比较
  2. 如果大于以“否”为标签，否则以“是”为标签
  3. 如果查询出来的分支是叶节点，返回该值
  4. 如果不是叶节点，递归查询子树

In [ ]:

```python
def run2(data, tree, labels):
    firstStr = list(tree.keys())[0]  # 根节点
    firstLabel = firstStr
    t = str(firstStr).find('<') #查看是否是连续型
    if t > -1:  # 如果是连续型的特征
        firstLabel = str(firstStr)[ : t]
    secondDict = tree[firstStr]
    featIndex = labels.index(firstLabel)  # 跟节点对应的属性
    result = ''
    for key in list(secondDict.keys()):  # 对每个分支循环
        if type(data[featIndex]) == type("a"):
            if data[featIndex] == key:  # 测试样本进入某个分支
                if type(secondDict[key]).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    result = run2(data, secondDict[key], labels)
                else:  # 如果是叶子，  返回结果
                    result = secondDict[key]
        else:
            value = float(str(firstStr)[t + 1 : ])
            if data[featIndex] <= value:
                if type(secondDict['是']).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    result = run2(data, secondDict['是'], labels)
                else:  # 如果是叶子，  返回结果
                    result = secondDict['是']
            else:
                if type(secondDict['否']).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    result = run2(data, secondDict['否'], labels)
                else:  # 如果是叶子，  返回结果
                    result = secondDict['否']
    return result

def getresult2(train, test, labels):
    ls = []
    tree = classify2(train, labels)
    print("生成决策树如下:", tree)
    for index in test:
        ls.append(run2(index, tree, labels))
    return ls

labels2 = ["色泽", "根蒂", "敲声", "纹理", "密度", "好瓜"]
result2 = getresult2(train2, test2, labels2)
```

```
生成决策树如下: {'纹理': {'模糊': {'密度<0.294': {'是': '是', '否': '是'}}, '清晰': {'根蒂': {'稍蜷': {'密度<0.38149999999999995': {'是': '是', '否': {'色泽': {'青绿': '是', '乌黑': '
是'}}}}, '蜷缩': {'密度<0.5820000000000001': {'是': '是', '否': {'敲声': {'沉闷': {'色泽': {'青绿': '是', '乌黑': '是'}}, '浊响': {'色泽': {'青绿': '是', '乌黑': '是'}}}}}}, '硬挺': '
是'}}, '稍糊': {'敲声': {'沉闷': {'密度<0.6615': {'是': '是', '否': {'根蒂': {'稍蜷': '是', '蜷缩': '是'}}}}, '浊响': {'密度<0.56': {'是': '是', '否': '是'}}}}}}
```

计算准确率

In [ ]:

```python
print("C4.5分类器对于test2数据集的准确率是：", simrate(test2, result2))
```

```
C4.5分类器对于test2数据集的准确率是： 60.00%
```

可以看出，在不剪枝的情况下C4.5预测的准确率稍较为一般