决策树

算法的伪代码

输入： 训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
　　属性集 $A=\{a_1,a_2,\ldots,a_d\}$.
过程： 函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3: 　将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A=\varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6: 　将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10: 　为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11: 　**if** D_v 为空 **then**
12: 　　将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13: 　**else**
14: 　　以 TreeGenerate(D_v, $A\setminus\{a_*\}$)为分支结点
15: 　**end if**
16: **end for**
输出： 以 node 为根结点的一棵决策树

数据集

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.46	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.36	0.37	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

Python 实现

由于基于信息增益的决策树算法一般适用于离散属性，若要处理连续属性则必须将其按照一定规则转为离散属性。所以在接下来的代码实现当中并未处理连续属性

1.信息熵的计算

```
def entropy(D):
    count=D.shape[0]
    Ent=0.0
    temp=D['好瓜'].value_counts() #获取剩余的类别数量
    for i in temp:
        Ent-=i/count*np.log2(i/count)
    return round(Ent, 3)
```

2.信息增益的计算

```
def cal_gain(D,Ent,A):
```

```
    '''
    D:剩余的样本集
    Ent：对应的信息熵
    A：剩余的属性集合
    '''
    # print("gain:",A)
    gain = []
    count=D.shape[0]
    for i in A:
        temp=0
        for j in attribute[i]:
            temp+=D[(D[i]==j)].shape[0]/count*entropy(D[(D[i]==j)])
        # print(temp)
        gain.append(round(Ent-temp,3))
        # print(i,round(Ent-temp,3))
    return np.array(gain)
```

3.决策树的生成

```
def same_value(D, A):
    for key in A:
        if key in attribute and len(D[key].value_counts()) > 1:
            return False
    return True


# 叶节点选择其类别为 D 中样本最多的类
def choose_largest_example(D):
    count = D['好瓜'].value_counts()
    return '是' if count['是'] >= count['否'] else '否'


def choose_best_attribute(D,A):
    Ent=entropy(D)
    gain=cal_gain(D,Ent,A)
    return A[gain.argmax()]


#A:剩余的属性集
```

```
def TreeGenerate(D, A):
    Count = D['好瓜'].value_counts()
    if len(Count) == 1:  #情况一，如果样本都属于一个类别
        return D['好瓜'].values[0]

    if len(A) == 0 or same_value(D, A): #情况二：如果样本为空或者样本的所有属性取值相同，则取类别较多的为分类标准
        return choose_largest_example(D)

    node = {}
    best_attr = choose_best_attribute(D,A)  #情况三：选择一个最佳属性作为分类节点
    D_size = D.shape[0]
    # 最优划分属性为离散属性时
    for value in attribute[best_attr]:  #对最佳属性当中的每个属性值进行分析
        Dv = D[D[best_attr] == value]
        if Dv.shape[0] == 0:
            node[value] = choose_largest_example(D)
        else:
            new_A = [key for key in A if key != best_attr]
            node[value] = TreeGenerate(Dv, new_A)
    return {best_attr: node}
```

4.树型结构可视化

```
def drawtree(tree,coordinate,interval):
    '''
    tree：决策树
    coordinate: 当前节点的坐标
    interval：分支节点间的间隔
    '''
    now_A=list(tree.keys())[0]
    plt.text(coordinate[0], coordinate[1], now_A, size=20,
             ha="center", va="center",
             bbox=dict(boxstyle="circle",
                       ec=(0.5, 0.8, 0.5),
                       fc=(0.5, 0.9, 0.5),
```

```
                        )
                )
        split_num=len(tree[now_A].values())
        next_coordinate=coordinate-[(split_num-1)*interval,5]
        for i in tree[now_A]:
                plt.plot([coordinate[0],next_coordinate[0]],[coordinate[1],next_coordinate[1]])

plt.text((coordinate[0]+next_coordinate[0])/2,(coordinate[1]+next_coordinate[1])/2,s=i,size=15)
                if tree[now_A][i] in labels:
                        plt.text(next_coordinate[0], next_coordinate[1],tree[now_A][i] , size=20,
                                ha="center", va="center",
                                bbox=dict(boxstyle="circle",
                                        ec=(0.5, 0.5, 0.8),
                                        fc=(0.5, 0.5, 0.9),
                                        )
                                )
                else:
                        drawtree(tree[now_A][i],next_coordinate,interval-4)
                next_coordinate+=[interval*2,0]
```

实验结果

字典形式存储的决策树

{‘纹理’: {‘清晰’: {‘根蒂’: {‘蜷缩’: ‘是’, ‘稍蜷’: {‘色泽’: {‘青绿’: ‘是’, ‘乌黑’: {‘触感’: {‘硬滑’: ‘是’, ‘软粘’: ‘否’}}, ‘浅白’: ‘是’}}, ‘硬挺’: ‘否’}}, ‘稍糊’: {‘触感’: {‘硬滑’: ‘否’, ‘软粘’: ‘是’}}, ‘模糊’: ‘否’}}

可视化结果