决策树实现

1.概述

决策树（decision tree）算法基于特征属性进行分类，其主要的优点：模型具有可读性，计算量小，分类速度快。对于某一决策树，其类规则是**互斥并且完备**的，这实质上是对特征空间的划分，即根据不同的性质取值，在特征空间上划分数据集，最终完成对整体的分类任务：

从实现步骤上来说，决策树可以分为3个阶段：

- 特征选择
- 决策树生成
- 决策树的修剪

在代码实现部分，将进行说明。

2.初级实现

在初级阶段，针对离散的属性值，可以直接采用ID-3方法逐一计算每一特征值的信息增益，从中选取恰当的特征作为下一步分类的依据：

其基本实现如下：

首先完成基本的数据准备工作：

```
water_train1=pd.read_csv("./Watermelon-train1.csv",encoding='gb2312')
labels_origin=['色泽','根蒂','敲声','纹理','好瓜']

water_train1=water_train1[labels_origin]
water_label=water_train1["好瓜"]
print(water_train1)
watertest1=pd.read_csv("./Watermelon-test1.csv",encoding='gb2312')
label_test=labels_origin[:]
watertest1=watertest1[label_test]
water_train2=pd.read_csv("./Watermelon-train2.csv",encoding='gb2312')
watertest2=pd.read_csv("./Watermelon-test2.csv",encoding='gb2312')
labels_origin2=['色泽','根蒂','敲声','纹理','密度','好瓜']
watertest2=watertest2[labels_origin2]
water_train2=water_train2[labels_origin2]
print(watertest2)
```

而后针对训练数据，进行熵的计算以及特征的选取，其基本方法如下：

```
def entropy(dataset):
    label_values = dataset[dataset.columns[-1]]
    counts =  label_values.value_counts()
    Ent = 0
    for c in label_values.unique():
        freq = counts[c]/len(label_values)
        Ent =Ent- freq*math.log(freq,2)
    return Ent
def ration(group,length_data):
    ratio=len(group)/length_data
    return (ratio)*entropy(group)
```

```
def gain_col(dataset,col):
    ent_gross=entropy(dataset)
    #print(dataset.groupby(col))
    data_grounp=dataset.groupby(col)
    ent_sum=0
    for name,item in data_grounp:
        ent_sum+=ration(item,len(dataset))
    return ent_gross-ent_sum
def decisiontree_fit(dataset):
    data=dataset[labels_origin]
    label=dataset.iloc[:,-1]
def get_type(dataset,loc):
    return  len(Counter([element for element in dataset[loc]]))
def get_most_label(labels):
    return (labels.value_counts()).idxmax()
def get_mid_rep(dataset,col):
    to_get=sorted(dataset[col])
    lis=[]
    for item in np.arange(len(to_get)-1):
        lis.append((to_get[item]+to_get[item+1])/2)
    return lis
```

通过对每一个属性的信息增益进行计算，可以得到各个情况下所应当进行选取的属性，最终构建出一颗决策树：

```
def decision_tree(dataset,label_list):
    feature = dataset.columns[:-1]
    label_list = dataset.iloc[:, -1]
    if len(feature)==0 or get_type(dataset,feature)==1:
        return get_most_label(label_list)
    elif len(pd.unique(label_list)) == 1:
        return label_list.values[0]
    bestAttr = max_col(dataset)
    tree = {bestAttr: {}}
    for attri_value,tree_data in dataset.groupby(by=bestAttr):
        if len(tree_data) == 0:
            tree[bestAttr][attri_value] = get_most_label(label_list)
        else:
            new_data = tree_data.drop(bestAttr,axis=1)
            tree[bestAttr][attri_value] = decision_tree(new_data,water_label)
    return tree
decision_tree = decision_tree(water_train1,water_label)
```

基于此，即可完成最终的预测操作：

```
def predict(tree, data):
    feature = list(tree.keys())[0]
    #print(feature)
    label = data[feature]
    next_tree = tree[feature][label]
    if type(next_tree) == str:
        return next_tree
    else:
        return predict(next_tree, data)
print(watertest1.columns)
labels=watertest1['好瓜']
```

```
acc=0
for num in np.arange(len(watertest1)):
    if predict(decision_tree ,watertest1.loc[num,:])==labels[num]:
        acc+=1
print('Accuracy: %f'%(acc/len(watertest1)))
```

可以计算出，在这一方法下，构建出的树的字典为：

```
{'纹理': {'模糊': '否', '清晰': {'根蒂': {'硬挺': '否', '稍蜷': '是', '蜷缩': '是'}},
'稍糊': {'色泽': {'乌黑': {'敲声': {'沉闷': '否', '浊响': '是'}}, '浅白': '否', '青绿':
'否'}}}}
```

而这一方法在测试集上的准确率为70%,基本符合预期。

中级-C4.5实现

在初级要求的基础上，针对第二个数据集，除了需要对特征选取方法进行更新外，还需要针对连续值进行处理，在本次实现中，以信息增益作为属性选取的依据，而将连续值进行排序后，不断以两数的平均值作为切分依据，进行切分处理，并从中选取信息增益最大的值作为对连续值进行切分的分界点，其基本实现如下：

```
def type_attri(dataset,attr):
    if dataset[attr].dtype == 'float64':
        return 1
    return 0
def gain(dataset,attr):
    if type_attri(dataset,attr):
        split_p = get_mid_rep(dataset,attr)
        return contin(dataset,attr,split_p)
    else:
        return discrete(dataset,attr)
def discrete(dataset,attr):
    ent_parent=entropy(dataset)
    attribution = dataset[attr].unique()
    entD = 0
    for attri in attribution:
        value = dataset[dataset[attr]== attri]
        entD += len(value)/len(dataset)*entropy(value)
    return (entropy(dataset)/ent_parent - entD/ent_parent,None)

def contin(dataset,attr,mid_term):
    entD = entropy(dataset)
    gains = []
    for item in mid_term:
        data_posi = dataset[dataset[attr] > item]
        data_neg = dataset[dataset[attr] <= item]
        sum_Ent=ration(data_posi,len(data_posi))+ration(data_neg,len(data_neg))
        gain_ratio = (entD - sum_Ent)/entD
        gains.append((gain_ratio,item))
    return max(gains)
```

而针对树的生成采用与1类似的方法：

```
def generateTree(data,label_list):
```

```
    label_values =  data.iloc[:, -1]
    if len(label_values.unique()) == 1:
        return label_values.values[0]
    if data.shape[1] == 1 or (data.shape[1]==2 and
len(data.T.ix[0].unique())==1):
        return get_most_label(label_values)
    bestAttr,(gain_attri,t_data) = split(data)
    if t_data is None:
        DecesionTree = {bestAttr:{}}
        values = data[bestAttr].unique()
        for val in values:
            attrsAndLabel = data.columns.tolist()
            data_cur = data[data[bestAttr]== val]
            attrsAndLabel.remove(bestAttr)
            data_cur  = data_cur [attrsAndLabel]
            DecesionTree[bestAttr][val] = generateTree(data_cur ,label_list)
        return DecesionTree
    else:
        node = bestAttr+'<'+str(t_data)
        DecesionTree = {node:{}}
        values = ['是','否']
        for val in values:
            if val == '是':
                data_cur  = data[data[bestAttr] <= t_data]
            elif val=="否":
                data_cur =data[data[bestAttr] > t_data]
            DecesionTree[node][val] = generateTree(data_cur ,label_list)
        return DecesionTree
```

最终可以得到基于C4.5的决策树，如下：

```
{'纹理': {'清晰': {'密度<0.3815': {'是': '否', '否': '是'}}, '稍糊': {'密度<0.56':
{'是': '是', '否': '否'}}, '模糊': '否'}}
```

对其实验结果进行验证，可以计算出其准确率为80%。

总结

在这一实验中，基于ID3方法与C4.5方法完成了对决策树的基本实现操作，针对信息增益与信息增益比进行属性的选择以完成分类任务，针对西瓜数据集均取得了理想的实验结果，**完成了基本要求与中级要求**；

在本次实验中，也遇到了一些问题，即：如何通过字典的形式完成对树的构建，以及进行针对连续值属性与离散属性进行属性的选择，在这些问题均耗费了较长的时间，也是本次实验的难点所在。

通过本次实验，基本了解了不同属性情况下对决策树的计算方法与属性的选择问题，从头构建出了一颗决策树，这也为后续的学习打下了基础。