实验　ID3 决策树课堂案例实现

学号：21030031009　姓名：惠欣宇　电话：18149045867

本实验要求独立完成。实验报告正文要求采用小四号字，中文使用宋体，英文使用 Times New Roman。对于实验中出现的代码文字，可使用 Consolas 字体。实验报告要求提交 pdf 电子版。电子版文件命名格式要求为：学号-姓名.pdf。

1. 实验目的

1) 深入理解 ID3 原理。
2) 掌握 ID3 的基本用法。
3) 利用 python 实现 ID3 算法。

2. 实验预备知识及实验要求

2.1. 实验预备知识

1) 决策树的概念
2) ID3 基本原理

2.2. 实验要求：

1）信息增益是针对一个一个特征而言的，就是看一个特征，系统有它和没有它时的信息量各是多少，两者的差值就是这个特征给系统带来的信息量，即信息增益。

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。

ID3 算法的核心思想就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。该算法采用自顶向下的贪婪搜索遍历可能的决策空间。那么我就可以通过信息增益的不同，决定不同特征出现的先后顺序，我们就可以计算其他特征。

下面给出部分未介绍到的代码：

数据集：

```
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],  # 数据集
               [0, 0, 0, 1, 'no'],
               [1, 0, 0, 0, 'yes'],
               [2, 1, 0, 0, 'yes'],
               [2, 2, 1, 0, 'yes'],
               [2, 2, 1, 1, 'no'],
               [1, 2, 1, 1, 'yes'],
               [0, 1, 0, 0, 'no'],
               [0, 2, 1, 0, 'yes'],
               [2, 1, 1, 0, 'yes'],
               [0, 1, 1, 1, 'yes'],
               [1, 1, 0, 1, 'yes'],
               [1, 0, 1, 0, 'yes'],
               [2, 1, 0, 1, 'no']]
    labels = ['天气', '气温', '湿度', '风']  # 分类属性
    return dataSet, labels  # 返回数据集和分类属性
```

统计 classlist 中出现次数最多的元素：

```
def majorityCnt(classList):
    classCount = {}
    for vote in classList:  # 统计classList中每个元素出现的次数
        if vote not in classCount.keys(): # 如果不在classList中
            classCount[vote] = 0 # 将对应元素数量置为0
        classCount[vote] += 1 # 如果在classList中，数量+1
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]  # 返回classList中出现次数最多的元素
```

使用决策树执行分类：

```
def classify(inputTree, featLabels, testVec):
    firstStr = next(iter(inputTree))  # 获取决策树结点
    secondDict = inputTree[firstStr]  # 下一个字典
    featIndex = featLabels.index(firstStr)
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':
                classLabel = classify(secondDict[key], featLabels, testVec)
            else:
                classLabel = secondDict[key]
    return classLabel
```

2）展示整个实验过程，可适当截图，并标注说明。

①主函数中首先调用创建训练数据集的函数 `createDataSet` 函数，然后初始化一个我们选取过的最优的属性值，因为初始我们并没有进行选取，所以我们初始化为空，接着调用 `createTree` 函数来创建决策树。然后构建测试数据集进行测试。

```
if __name__ == '__main__':
    dataSet, labels = createDataSet() # 获取数据集和相
    featLabels = [] # 已经选取过的最优属性
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
    testVec = [0,0,0,0]  # 测试数据
    result = classify(myTree, featLabels, testVec)
    if result == 'yes':
        print('打网球')
    if result == 'no':
        print('不打网球')
```

②我们主要分析创建决策树函数，这是最重要的一部分。它接收的是传入的原始数据集以及数据集现在会被划分的属性和已经使用过的属性。首先遍历

dataset 取它的最后一列，获取这个分类标签，我们再进行判断。因为这个函数是个递归函数，我们需要先进行判断他的条件是否满足递归终止的条件。如果是类别完全相同那就停止遍历，当 dataset 等于 0，说明所有特征遍历完毕。然后调用 chooseBestFeatureToSplit 函数选择最优特征，得到最优特征的下标。通过最优下标获取属性值，最后根据最优特征生成决策树。最后使用 splitDataSet 函数进行子集的拆分。

```
def createTree(dataSet, labels, featLabels):
    classList = [example[-1] for example in dataSet]  # 取分类标签(是否打网球:yes or no)
    if classList.count(classList[0]) == len(classList):  # 如果类别完全相同则停止继续划分
        return classList[0]
    if len(dataSet[0]) == 0:  # 遍历完所有特征时返回出现次数最多的类标签
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)  # 选择最优特征
    bestFeatLabel = labels[bestFeat]  # 最优特征的标签
    featLabels.append(bestFeatLabel)# 存储选择的最优特征标签
    print('bestFeat',bestFeat)
    print('bestFeatLabel',bestFeatLabel)
    print('featLabels',featLabels)
    myTree = {bestFeatLabel: {}}  # 根据最优特征的标签生成树
    del (labels[bestFeat])  # 删除已经使用特征标签
    featValues = [example[bestFeat] for example in dataSet]  # 得到训练集中所有最优特征的属性值
    uniqueVals = set(featValues)  # 去掉重复的属性值
    for value in uniqueVals:
        subLabels = labels[:]
        # 递归调用函数createTree(),遍历特征，创建决策树。
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels, featLabels)
    return myTree
```

③下面我们来看看划分数据集的函数，他首先是获取数据集，然后再获取最优特征它所对应的下标值，以及最优特征所取的属性值。我们遍历数据集，如果遇到与最优特征所取到的属性值相同的属性值时，就把这个最优特征从数据集中去掉。最后将符合条件的添加到返回的数据集，形成一个划分，在对划分好的数据集继续进行递归，最终生成整个决策树。

```
def splitDataSet(dataSet, axis, value):
    retDataSet = []  # 创建返回的数据集列表
    for featVec in dataSet:  # 遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]  # 去掉axi
            reducedFeatVec.extend(featVec[axis + 1:])
            retDataSet.append(reducedFeatVec) # 加入到
    return retDataSet  # 返回划分后的数据集
```

④首先获取特征数量，如果初始四个特征都未被选择，则 dataset 为 5，需要减去一列获得特征数量。然后计算这个数据集最原始的信息熵，初始化信息增益 bestInfoGain 与最优特征索引值 bestFeature，然后根据特征数

量遍历数据集，依次在天气、气温、湿度和风这四个变量中进行计算。计算它们的信息增益，然后选择信息增益大的进行划分。首先我们遍历所有特征，选取第 i 个特征，根据第一个元素的特征获取到第一个属性的所有取值，然后初始化一个经验条件熵 newEntropy 设为 0，然后根据每一个特征不同的取值，首先划分他每一个特征不同的取值，对应便是子数据集。根据五条数据集计算子集的概率，再计算信息熵与条件熵，计算出条件熵之后，根据天气属性其他的几个子集的条件熵进行加权求和，最后求得平均条件熵与信息增益。依次计算所有的属性，得到四个不同的信息增益，根据信息增益选出最优的特征，以及特征的索引值，进行返回。

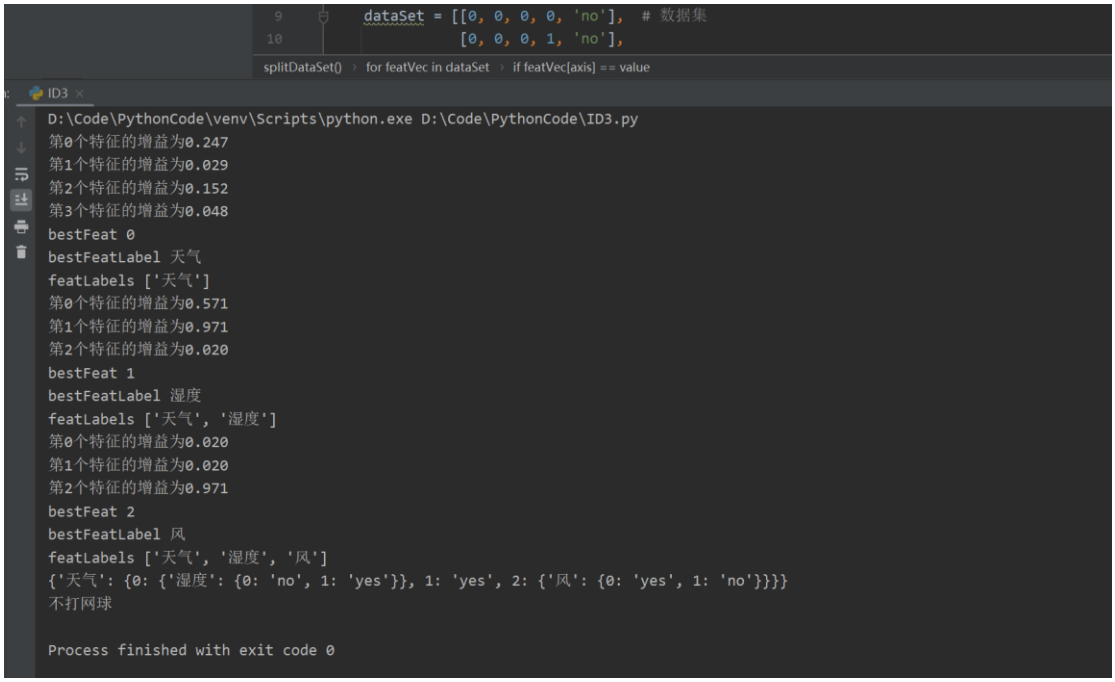
```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1  #特征数量
    baseEntropy = calcShannonEnt(dataSet)  # 计算数据集的香农
    bestInfoGain = 0.0  # 信息增益
    bestFeature = -1  # 最优特征的索引值
    for i in range(numFeatures):  # 征遍历所有特征
        # 获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)  # 创建set集合{},元素不可重
        newEntropy = 0.0  # 经验条件熵
        for value in uniqueVals:  # 计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)  #
            prob = len(subDataSet) / float(len(dataSet))  #
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy  # 信息增益
        print("第%d个特征的增益为%.3f" % (i, infoGain))  # 打
        if (infoGain > bestInfoGain):  # 计算信息增益
            bestInfoGain = infoGain  # 更新信息增益，找到最大
            bestFeature = i  # 记录信息增益最大的特征的索引值
    return bestFeature  # 返回信息增益最大的特征的索引值
```

⑤计算信息熵的函数

```
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)  # 返回数据集的行数
    labelCounts = {}  # 保存每个标签(Label)出现次数的
    for featVec in dataSet:  # 对每组特征向量进行统计
        currentLabel = featVec[-1]  # 提取标签(Label)
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1  # 如果标签(La
    shannonEnt = 0.0  # 经验熵(香农熵)
    for key in labelCounts:  # 计算香农熵
        prob = float(labelCounts[key]) / numEntires
        shannonEnt -= prob * log(prob, 2)  # 利用公式
    return shannonEnt  # 返回经验熵(香农熵)
```

3）实验结果。

第一步是根据每一个特征进行选择，选择最大信息增益的作为第一个划分的特征进行划分子数据集。这里选择到“天气”，对于天气来说有三个取值，分别为 0、1、2。对于 0，便是根据天气取晴的情况进行计算信息增益的，然后再选择最大信息增益作为第二个划分特征进行划分子数据集。这次选择到了“湿度”，到了“湿度”便是到了第二层决策树的构建，与上述过程相同，再递归的进行构建第三层决策树。我们发现当“天气”取值为 1（多云）时，所有数据的 label 值都为 yes，说明我们已经可以完全划分出该数据集了，该数据集没有不确定性了，此时终止递归。



4）总结分析。

决策树是一种类似于流程图的树结构，由内部结点、叶节点、有向边组成。一个内部结点表示一个特征上的测试，一个叶节点表示类别标号。决策树分类就是从根节点开始，对实例的某一特征进行测试，根据测试结果将实例划分子节点，再对子结点上的剩余特征进行递归测试并分配，直到产生叶子节点。但是决策树容易产生过拟合，可以通过剪枝来缓解。不能保证建立全局最优的决策树，可以通过随机森林来缓解。