**实验五 决策树**

分类决策树模型是一种描述对实例进行分类的树形结构。决策树由结点和有向边组成。结点有两种类型：内部节点和叶节点，内部节点表示一个特征或属性，叶节点表示一个类。

分类的时候，从根节点开始，对实例的某一个特征进行测试，根据测试结果，将实例分配到其子结点；此时，每一个子结点对应着该特征的一个取值。如此递归向下移动，直至达到叶结点，最后将实例分配到叶结点的类中。

决策树的创建可包括三部分：特征选择、决策树的生成以及决策树的剪枝；决策树的应用包括：分类、回归以及特征选择。决策树最经典的算法包括：ID3、C4.5以及CART算法。

熵表示随机变量的不确定性，熵值越大表示随机变量含有的信息越少，变量的不确定性越大。克劳德·香农提出，如果待分类的事物可能划分在多个分类之中，则符号的信息定义为：

其中表示选择该分类的概率。而“熵”定义为信息的期望值，因此其计算公式为：

其中n为分类的数目。

特征A对数据集D的信息增益计算公式为：

其中a为特征A的所有取值，表示根据特征A的取值把数据集D划分成的a份中的一份，表示该子集的样本数，表示的熵。

**实验要求：**实现决策树ID3算法，该算法在特征选择时使用的是信息增益。不要求对决策树进行剪枝。

**实验步骤：**

创建一个trees.py文件，添加如下代码：

*from math import log*

*import operator*

①创建一个简单的数据集。这个数据集根据两个属性来判断一个海洋生物是否属于鱼类，第一个属性是不浮出水面是否可以生存，第二个属性是是否有鳍。数据集中的第三列是分类结果。

*def createDataSet():*

*dataSet = [[1, 1, 'yes'],*

*[1, 1, 'yes'],*

*[1, 0, 'no'],*

*[0, 1, 'no'],*

*[0, 1, 'no']]*

# 这里的labels保存的是属性名称。

*labels = ['no surfacing','flippers']*

*return dataSet, labels*

②编写函数计算熵。计算公式看前面。

*def calcEntropy(dataSet):*

# 获取总的训练数据数

*numEntries = len(dataSet)*

# 创建一个字典统计各个类别的数据量

*labelCounts = {}*

*for featVec in dataSet:*

# 使用下标-1获取所属分类保存到currentLabel

**#【代码待补全】**

# 若获得的类别属于新类别，则初始化该类的数据条数为0

*if currentLabel not in labelCounts.keys():*

*labelCounts[currentLabel] = 0*

*labelCounts[currentLabel] += 1*

*entropy = 0.0*

*for key in labelCounts.keys():*

# 计算p(xi)

**#【代码待补全】**

# 计算熵

**#【代码待补全】**

*return entropy*

③编写函数，实现按照给定特征划分数据集。

*def splitDataSet(dataSet, axis, value):*

*returnDataSet = []*

*for featVec in dataSet:*

*if featVec[axis] == value:*

# 隔开axis这一列提取其它列的数据

# 保存到变量reducedFeatVec中

**#【代码待补全】**

*returnDataSet.append(reducedFeatVec)*

*return returnDataSet*

④实现特征选择函数。遍历整个数据集，循环计算熵和splitDataSet()函数，找到最好的特征划分方式。

*def chooseBestFeatureToSplit(dataSet):*

# 获取属性个数，保存到变量numFeatures

# 注意数据集中最后一列是分类结果

**#【代码待补全】**

*baseEntropy = calcEntropy(dataSet)*

*bestInfoGain = 0.0; bestFeature = -1*

*for i in range(numFeatures):*

# 获取数据集中某一属性的所有取值

*featList = [example[i] for example in dataSet]*

# 获取该属性所有不重复的取值，保存到uniqueVals中

# 可使用set()函数去重

**#【代码待补全】**

*newEntropy = 0.0*

*for value in uniqueVals:*

*subDataSet = splitDataSet(dataSet, i, value)*

# 计算按照第i列的某一个值分割数据集后的熵

# 参考文档开始部分介绍的公式

**#【代码待补全】**

*infoGain = baseEntropy - newEntropy*

*if (infoGain > bestInfoGain):*

*bestInfoGain = infoGain*

*bestFeature = i*

*return bestFeature*

⑤决策树创建过程中会采用递归的原则处理数据集。递归的终止条件为：程序遍历完所有划分数据集的属性；或者每一个分支下的所有实例都具有相同的分类。如果数据集已经处理了所有属性，但是类标签依然不是唯一的，此时我们需要决定如何定义该叶子节点，在这种情况下，通常会采用多数表决的方法决定分类。

*def majorityCnt(classList):*

*classCount = {}*

*for vote in classList:*

*if vote not in classCount.keys():*

*classCount[vote] = 0*

*classCount[vote] += 1*

*sortedClassCount = sorted(classCount.items(), \*

*key=operator.itemgetter(1), reverse=True)*

*return sortedClassCount[0][0]*

⑥创建决策树。

*def createTree(dataSet, labels):*

# 获取类别列表，类别信息在数据集中的最后一列

# 使用变量classList

**#【代码待补全】**

# 以下两段是递归终止条件

# 如果数据集中所有数据都属于同一类则停止划分

# 可以使用classList.count(XXX)函数获得XXX的个数，

# 然后那这个数和classList的长度进行比较，相等则说明

# 所有数据都属于同一类，返回该类别即可

**#【代码待补全】**

# 如果已经遍历完所有属性则进行投票，调用上一步的函数

# 注意，按照所有属性分割完数据集后，数据集中会只剩下

# 一列，这一列是分类结果

**#【代码待补全】**

# 调用特征选择函数选择最佳分割属性，保存到bestFeat

# 根据bestFeat获取属性名称，保存到bestFeatLabel中

**#【代码待补全】**

# 初始化决策树，可以先把第一个属性填好

*myTree = {bestFeatLabel:{}}*

# 删除最佳分离属性的名称以便递归调用

*del(labels[bestFeat])*

# 获取最佳分离属性的所有不重复的取值保存到uniqueVals

**#【代码待补全】**

*for value in uniqueVals:*

# 复制属性名称，以便递归调用

*subLabel = labels[:]*

# 递归调用本函数生成决策树

*myTree[bestFeatLabel][value] = createTree\*

*(splitDataSet(dataSet, bestFeat, value), subLabel)*

*return myTree*

⑦利用构建好的决策树进行分类。

*def classify(inputTree, featLabels, testVec):*

# 获取树的第一个节点，即属性名称

*firstStr = inputTree.keys()[0]*

# 获取该节点下的值

*secondDict = inputTree[firstStr]*

# 获取该属性名称在原属性名称列表中的下标

# 保存到变量featIndex中

# 可使用index(XXX)函数获得XXX的下标

**#【代码待补全】**

# 获取待分类数据中该属性的取值，然后在secondDict

# 中寻找对应的项的取值

# 如果得到的是一个字典型的数据，说明在该分支下还

# 需要进一步比较，因此进行循环调用分类函数；

# 如果得到的不是字典型数据，说明得到了分类结果

*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*

可以在python命令行中测试结果：

*import trees*

*data,labels=trees.createDataSet()*

*myTree=trees.createTree(data,labels)*

*trees.classify(myTree,labels,[1,0])*

上面的代码中[1,0]为新数据。

**\*\*\*延伸拓展\*\*\***

一、决策树的存储

决策树的构建是很耗时的任务，但用创建好的决策树解决分类问题时可以很快完成，因此为了节省时间，最好在每次执行分类时直接调用已经构造好的决策树。因此延伸出了决策树存储的问题。如有需要可使用以下代码：

# 存储决策树

*def storeTree(inputTree,filename):*

*import pickle*

*fw = open(filename,’w’)*

*pickle.dump(inputTree,fw)*

*fw.close()*

# 根据文件名读取决策树

*def grabTree(filename):*

*import pickle*

*fr = open(filename)*

*return pickle.load(fr)*

二、使用python绘制决策树

决策树相比于k近邻算法的优势就在于数据形式非常容易理解，因此可视化决策树是很有帮助的。这一部分的内容不在本次实验的要求内，大家有兴趣可以参考**《机器学习实战》**这本书的**3.2**节。