决策树

**def** calcShannonEnt(dataSet): *# 计算数据的熵(entropy)* numEntries=len(dataSet) *# 数据条数*  labelCounts={}  
 **for** featVec **in** dataSet:  
 currentLabel=featVec[-1] *# 每行数据的最后一个字（类别）*  **if** currentLabel **not in** labelCounts.keys():  
 labelCounts[currentLabel]=0  
 labelCounts[currentLabel]+=1 *# 统计有多少个类以及每个类的数量* shannonEnt=0  
 **for** key **in** labelCounts:  
 prob=float(labelCounts[key])/numEntries *# 计算单个类的熵值*

shannonEnt-=prob\*log(prob,2) *# 累加每个类的熵值* **return** shannonEnt

输入所需数据

**def** splitDataSet(dataSet,axis,value): *# 按某个特征分类后的数据* retDataSet=[]

**for** featVec **in** dataSet:  
 **if** featVec[axis]==value:  
 reducedFeatVec =featVec[:axis]  
 reducedFeatVec.extend(featVec[axis+1:])  
 retDataSet.append(reducedFeatVec)  
 **return** retDataSet  
 **def** chooseBestFeatureToSplit(dataSet): *# 选择最优的分类特征* numFeatures = len(dataSet[0])-1  
 baseEntropy = calcShannonEnt(dataSet) *# 原始的熵* bestInfoGain = 0  
 bestFeature = -1  
 **for** i **in** range(numFeatures):  
 featList = [example[i] **for** example **in** dataSet]  
 uniqueVals = set(featList)  
 newEntropy = 0  
 **for** value **in** uniqueVals:  
 subDataSet = splitDataSet(dataSet,i,value)  
 prob =len(subDataSet)/float(len(dataSet))  
 newEntropy +=prob\*calcShannonEnt(subDataSet) *# 按特征分类后的熵* infoGain = baseEntropy - newEntropy *# 原始熵与按特征分类后的熵的差值* **if** (infoGain>bestInfoGain): *# 若按某特征划分后，熵值减少的最大，则次特征为最优分类特征* bestInfoGain=infoGain  
 bestFeature = i  
 **return** bestFeature  
**def** majorityCnt(classList): *#按分类后类别数量排序，比如：最后分类为2男1女，则判定；* 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]  
  
 **def** createTree(dataSet,labels):  
 classList=[example[-1] **for** example **in** dataSet] *# 类别：男或女* **if** classList.count(classList[0])==len(classList):  
 **return** classList[0]  
 **if** len(dataSet[0])==1:  
 **return** majorityCnt(classList)  
 bestFeat=chooseBestFeatureToSplit(dataSet) *#选择最优特征* bestFeatLabel=labels[bestFeat]  
 myTree={bestFeatLabel:{}} *#分类结果以字典形式保存* **del**(labels[bestFeat])  
 featValues=[example[bestFeat] **for** example **in** dataSet]  
 uniqueVals=set(featValues)  
 **for** value **in** uniqueVals:  
 subLabels=labels[:]  
 myTree[bestFeatLabel][value]=createTree(splitDataSet\  
 (dataSet,bestFeat,value),subLabels)  
 **return** myTree