import math as m

def createDataSet():

#数据集用字符串表示

dataSet = [['yes', 'yes', 'high', 'france', 'some', '0-10', 'yes'],

['yes', 'yes', 'low', 'china', 'full', '30-60', 'no'],

['no', 'no', 'low', 'fast', 'some', '0-10', 'yes'],

['yes', 'yes', 'low', 'china', 'full', '10-30', 'yes'],

['yes', 'no', 'high', 'france', 'full', '>60', 'no'],

['no', 'yes', 'mid', 'italy', 'some', '0-10', 'yes'],

['no', 'no', 'low', 'fast', 'nobody', '0-10', 'no'],

['no', 'yes', 'mid', 'china', 'some', '0-10', 'yes'],

['no', 'no', 'low', 'fast', 'full', '>60', 'no'],

['yes', 'yes', 'high', 'italy', 'full', '10-30', 'no'],

['yes', 'no', 'low', 'china', 'nobody', '0-10', 'no'],

['no', 'yes', 'low', 'fast', 'full', '30-60', 'yes'],

]

return dataSet

# 计算熵

def calcShannonEnt(dataSet):

numEntries = len(dataSet)

labelCounts = {}

for featVec in dataSet:

currentLable = featVec[-1]

if currentLable not in labelCounts.keys():

labelCounts[currentLable] = 0

labelCounts[currentLable] += 1

#计算熵

Ent = 0.0

for feat in labelCounts:

prob = float(labelCounts[feat]) / numEntries

Ent -= prob \* m.log(prob, 2)

return Ent

# 划分数据集

def splitDataSet(dataSet, axis, value):

retDataSet = []

for featVec in dataSet:

if featVec[axis] == value:

reducedFeatVec = featVec[:axis]

reducedFeatVec.extend(featVec[axis + 1:])

retDataSet.append(reducedFeatVec)

return retDataSet # 返回一个矩阵

# 根据熵，选择最优的划分方式

def chooseBestFeatureToSplit(dataSet):

baseEntropy = calcShannonEnt(dataSet)

numFeatures = len(dataSet[0]) - 1

bestInfoGain = 0.0

bestFeature = 0

for i in range(0, numFeatures):

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

uniqueVals = set(featList)

newEntorpy = 0.0

for value in uniqueVals:

subDataSet = splitDataSet(dataSet, i, value)

prob = len(subDataSet) / float(len(dataSet))

newEntorpy += prob \* calcShannonEnt(subDataSet)

infoGain = baseEntropy - newEntorpy

if (infoGain > bestInfoGain):

bestInfoGain = infoGain

bestFeature = i

return bestFeature

def majorityCnt(classList):

classCount = {}

for vote in classList:

if vote not in classCount.keys():

classCount[vote] = 0

classCount[vote] += 1

sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)

return sortedClassCount[0][0]

# 创建树

def createTree(dataSet, labels):

classList = [example[-1] for example in dataSet]

if classList.count(classList[0]) == len(classList):

return classList[0]

if len(dataSet[0]) == 1:

return majorityCnt(classList)

bestFeat = chooseBestFeatureToSplit(dataSet)

bestFeatLabel = labels[bestFeat]

myTree = {bestFeatLabel: {}}

del (labels[bestFeat])

# 构建树

featValues = [example[bestFeat] for example in dataSet]

uniqueVals = set(featValues)

for value in uniqueVals:

subLabels = labels[:]

subDataSet = splitDataSet(dataSet, bestFeat, value)

myTree[bestFeatLabel][value] = createTree(subDataSet, subLabels)

return myTree

# 测试

if \_\_name\_\_ == '\_\_main\_\_':

dataSet = createDataSet()

labels = ['choice', 'hungry', 'price', 'types', 'people', 'waittimes']

labelsForCreateTree = labels[:]

Tree = createTree(dataSet, labelsForCreateTree)

print(Tree)

结果：

