**《机器学习与数据挖掘》实验四**

实验题目： 编程实现基于信息增益进行划分选择的决策树算法

实验目的： 掌握构建决策树的基本流程

实验环境（硬件和软件） Anaconda/Jupyter notebook/Pycharm

实验内容：

编码实现基于信息增益进行划分选择的决策树算法，为给定数据生成一棵决策树；

**要求：**

**一、**已经给定部分代码，补充完整的代码，需要补充代码的地方已经用红色字体标注，包括：

1. 在第（2）部分；

**#补充计算信息熵的代码**

2. 在第（3）部分；

**#补充按给定特征和特征值划分好的数据集的代码**

3. 在第（4）部分：

**#补充计算条件熵的代码**

4. 在第（6）部分：

**#遍历uniqueVals中的每个值，生成相应的分支**

**二、**将补充完整的第（2）（3）（4）（6）部分的代码提交，并提交实验结果；（**也可以自己重写这四部分的代码提交**）

from math import log

import numpy as np

import operator

import csv

**（1）加载数据集；**

#方式1；

def loaddata ():

dataSet = [[0, 0,0,0,0,0, 'yes'],

[1, 0,1,0,0,0,'yes'],

[1, 0,0,0,0,0,'yes'],

[0, 0,1,0,0,0,'yes'],

[2, 0,0,0,0,0,'yes'],

[0, 1,0,0,1,1,'yes'],

[1, 1,0,1,1,1,'yes'],

[1, 1,0,0,1,0, 'yes'],

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

[0, 2,2,0,2,1,'no'],

[2, 2,2,2,2,0,'no'],

[2, 0,0,2,2,1,'no'],

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

[2, 1,1,1,0,0,'no'],

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

[2, 0,0,2,2,0,'no'],

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

feature\_name = ['a1','a2','a3','a4','a5','a6']

return dataSet, feature\_name

#方式2

def loaddata\_new():

# 定义文件路径

csv\_path = 'watermelon2.csv'

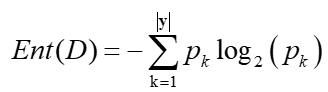
with open(csv\_path,'r',encoding='utf-8-sig')as fp:

dataSet = [i for i in csv.reader(fp)] # csv.reader 读取到的数据是list类型

feature\_name = ['a1','a2','a3','a4','a5','a6']

return dataSet, feature\_name

**（2）计算数据集的熵；**



def entropy(dataSet):

#数据集条数

m = len(dataSet)

#保存所有的类别及属于该类别的样本数

labelCounts = {}

for featVec in dataSet:

currentLabel = featVec[-1]

if currentLabel not in labelCounts.keys():

labelCounts[currentLabel] = 0

labelCounts[currentLabel] += 1

#保存熵值

e = 0.0

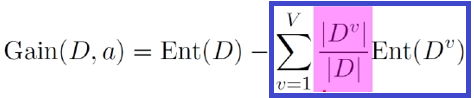
**#补充计算信息熵的代码**

for key in labelCounts:

prob=float(labelCounts[key])/m

e-=prob\*log(prob,2)return e

**（3）划分数据集；**



def splitDataSet(dataSet, axis, value):

**#补充按给定特征和特征值划分好的数据集的代码**

**# axis对应的是特征的索引;**

retDataSet = []

#遍历数据集

for featVec in dataSet:

if featVec[axis]==value:

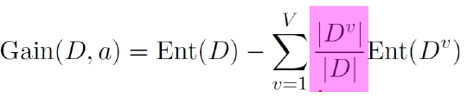
reducedFeatVec=featVec[:axis]

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

retDataSet.append(reducedFeatVec)

return retDataSet

**（4）选择最优特征；**



def chooseBestFeature(dataSet):

n = len(dataSet[0]) - 1

#计数整个数据集的熵

baseEntropy = entropy(dataSet)

bestInfoGain = 0.0; bestFeature = -1

#遍历每个特征

for i in range(n):

#获取当前特征i的所有可能取值

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

uniqueVals = set(featList)

newEntropy = 0.0

#遍历特征i的每一个可能的取值

for value in uniqueVals:

#按特征i的value值进行数据集的划分

subDataSet = splitDataSet(dataSet, i, value)

**#补充计算条件熵的代码**

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

newEntropy+=prob\*entropy(subDataSet)

#计算信息增益

infoGain = baseEntropy - newEntropy

#保存当前最大的信息增益及对应的特征

if (infoGain > bestInfoGain):

bestInfoGain = infoGain

bestFeature = i

return bestFeature

**（5）类别投票表决；**

def classVote(classList):

#定义字典，保存每个标签对应的个数

classCount={}

for vote in classList:

if vote not in classCount.keys():

classCount[vote] = 0

classCount[vote] += 1

#排序

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

return sortedClassCount[0][0]

**（6）递归训练一棵树；**

def trainTree(dataSet,feature\_name):

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

#所有类别都一致

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

return classList[0]

#数据集中没有特征

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

return classVote(classList)

#选择最优划分特征

bestFeat = chooseBestFeature(dataSet)

bestFeatName = feature\_name[bestFeat]

myTree = {bestFeatName:{}}

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

uniqueVals = set(featValues)

**#遍历uniqueVals中的每个值，生成相应的分支**

for value in uniqueVals:

sub\_feature\_name = feature\_name[:]

# 生成在dataSet中bestFeat取值为value的子集；

sub\_dataset = feature\_name[:] #补充代码；

# 根据得到的子集，生成决策树

myTree[bestFeatName][value] = myTree[bestFeatName][value] =trainTree(splitDataSet(dataSet,bestFeat,value),sub\_dataset) #补充代码；

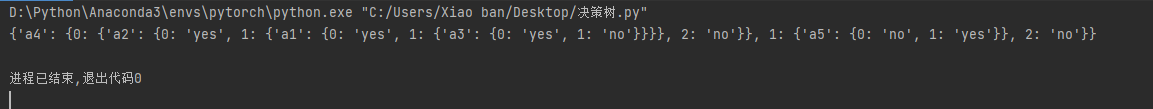
return myTree

**（7）测试代码**

myDat,feature\_name = loaddata()

myTree = trainTree(myDat,feature\_name)

print(myTree)



补充：预测代码，自己练习，不需要提交到实验报告中；

（1）给定新样本，预测类别；

def predict(inputTree,featLabels,testVec):

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

secondDict = inputTree[firstStr]

featIndex = featLabels.index(firstStr)

key = testVec[featIndex]

valueOfFeat = secondDict[key]

if isinstance(valueOfFeat, dict):

classLabel = predict(valueOfFeat, featLabels, testVec)

else: classLabel = valueOfFeat

return classLabel

（2）测试代码

print(predict(myTree,feature\_name,[1,1,0,1,0,0]))