**'''**

**DecisionTree Algorithm**

**Created by PyCharm**

**Date: 2018/7/31**

**'''**

**from math import log**

**import operator**

**import matplotlib.pyplot as plt**

**import numpy as np**

**def loadDataSet(path,training\_sample):**

**'''**

**从文件中读入训练样本的数据，同上面给出的示例数据**

**下面第20行代码中的1.0表示x0 = 1**

**@param filename 存放训练数据的文件路径**

**@return dataMat 存储训练数据的前两列**

**@return labelMat 存放给出的标准答案（0,1）**

**'''**

**dataMat = []; labelMat = [] #定义列表**

**filename = path+training\_sample**

**fr = open(filename)**

**for line in fr.readlines():**

**line = line.strip('\n')**

**lineArr = line.strip().split(' ') #文件中数据的分隔符**

**dataMat.append([float(lineArr[0]), float(lineArr[1]),float(lineArr[2])]) #前两列数据和一列标签**

**labelMat.append(float(lineArr[2])) #标准答案**

**return dataMat,labelMat**

**def calcShannonEnt(dataSet): #计算数据的熵(entropy)**

**'''**

**计算给定数据集的香农熵**

**@:param dataSet 数据集**

**@:return shannonEnt 返回香农熵值**

**'''**

**numEntries = len(dataSet) #数据条数**

**labelCounts = {}**

**for featVec in dataSet: #统计每一类的数量**

**currentLabel = featVec[-1] #取最后一列的键值**

**if currentLabel not in labelCounts.keys(): #当前键值不存在，初始化当前键值**

**labelCounts[currentLabel] = 0**

**labelCounts[currentLabel] += 1 #统计当前键值出现的次数**

**shannonEnt = 0**

**for key in labelCounts: #计算所有键值的熵**

**prob = float(labelCounts[key])/numEntries #计算单个键值的熵值**

**shannonEnt -= prob\*log(prob,2) #累加单个键值的熵值**

**return shannonEnt**

**def createDataDic(feat): #创建分支条件**

**'''**

**定义数据集，画图用**

**@:param dataSet 数据集**

**@:param labels 特征值**

**'''**

**dataSet = [['<'+str(feat[0]),'<'+str(feat[1]),'false'],**

**['>'+str(feat[0]),'<'+str(feat[1]),'false'],**

**['<'+str(feat[0]),'>'+str(feat[1]),'false'],**

**['>'+str(feat[0]),'>'+str(feat[1]),'true']]**

**labels = ['feature1','feature2']**

**return dataSet,labels**

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

**"""**

**统计数据集中该特征值value的数量**

**@:param dataSet 待划分数据集**

**@:param axis 划分数据集的特征,指出是第几类特征**

**@:param value 特征的返回值，指出是哪一类特征的那个值**

**@return retDataSet 划分后的数据集**

**"""**

**retDataSet = []**

**for featVec in dataSet: #取一行**

**if featVec[axis] == value: #该列值是否为所要值**

**reducedFeatVec = featVec[:axis] #取0到axis的值**

**#reducedFeatVec = featVec[:]**

**reducedFeatVec.extend(featVec[axis+1:]) #取axis+1之后的值**

**retDataSet.append(reducedFeatVec)**

**return retDataSet**

**def chooseBestFeatureToSplit(dataSet): #选择最优的分类特征**

**"""**

**选择特征划分的优先次序，画图用**

**@:param dataSet 初始数据集**

**@:return bestFeature 最优划分方式**

**"""**

**numFeatures = len(dataSet[0])-1 #数据集中的特征数量**

**baseEntropy = calcShannonEnt(dataSet) #根据标签计算的初始熵**

**bestInfoGain = 0**

**bestFeature = -1**

**for i in range(numFeatures): #寻找最优分类特征**

**featList = [example[i] for example in dataSet] #第i类特征**

**uniqueVals = set(featList) #去除重复的特征值**

**newEntropy = 0 #初始化信息熵**

**for value in uniqueVals:**

**subDataSet = splitDataSet(dataSet,i,value) #第i列特征中value值在dataSet的数量**

**prob = len(subDataSet)/float(len(dataSet)) #该特征值数除特征值总数量**

**newEntropy += prob\*calcShannonEnt(subDataSet) #累加该列特征各特征值的信息熵**

**infoGain = baseEntropy - newEntropy #信息增益=熵（总）- 熵（某个特征）**

**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 getSubCol(dataSet,col1,col2):**

**"""**

**取列表的部分列**

**@:param dataSet 数据列表**

**@:param col1 第col1列**

**@:param col2 第col2列**

**@:return list 返回列表子集**

**"""**

**rownum = len(dataSet)**

**list = []**

**for featVec in dataSet: # 统计每一类的数量**

**list.append([featVec[col1],featVec[col2]])**

**return list**

**def getSubRow(dataSet,row1,row2):**

**"""**

**取列表的部分行**

**@:param dataSet 数据列表**

**@:param row1 第row1行**

**@:param row2 第row2行**

**@:return list 返回列表子集**

**"""**

**rownum = len(dataSet) #数据行数**

**list = []**

**for i in range(row1,row2+1): #取部分数据集**

**list.append(dataSet[i])**

**return list**

**def chooseBestNumberToSplit(baseEntropy,featList):**

**"""**

**获取每个特征属性的最佳分割点**

**@:param dataSet 数据列表**

**@:return bestNumber 返回最佳分割点**

**"""**

**rownum = len(featList) #行数**

**bestInfoGain = 0 #最佳信息增益**

**bestNumber = -1 #最佳分割点的下标**

**featList.sort() #递增排序**

**for i in range(rownum):**

**subList = getSubRow(featList,0,i) #获取0到i行的数据**

**EntD0 = calcShannonEnt(subList) #前部分信息熵**

**temp = rownum - (i+1)**

**subList = getSubRow(featList,i+1,rownum-1) #获取i+1到最后一行的数据**

**EntD1 = calcShannonEnt(subList) #后部分信息熵**

**Gain = baseEntropy - (((i+1)/rownum)\*EntD0+(temp/rownum)\*EntD1) #计算信息增益**

**if Gain > bestInfoGain: #是否大于当前最大信息增益**

**bestNumber = i**

**bestInfoGain = Gain**

**return featList[bestNumber][0] #返回最佳分割点**

**def majorityCnt(classList):**

**"""**

**按分类后类别数量排序，比如：最后分类为2男1女，则判定为男**

**@:param classList 数据字典**

**@:return sortedClassCount[0][0] 返回出现次数最多的分类名称**

**"""**

**classCount={}**

**for vote in classList: #统计各键值的频率**

**if vote not in classCount.keys(): #若不存在初始化为0**

**classCount[vote]=0**

**classCount[vote]+=1 #频率加1**

**#利用operator操作键值排序字典**

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

**return sortedClassCount[0][0]**

**def createTree(dataSet,treeSet,labels):**

**"""**

**创建树**

**@:param dataSet 原始数据集**

**@:param labels 特征值**

**@:param myTree 返回创建好的决策树**

**"""**

**classList=[example[-1] for example in treeSet] #最后一列值**

**if classList.count(classList[0])==len(classList): #类别完全相同则停止继续划分**

**return classList[0]**

**if len(treeSet[0])==1: #遍历完所有特征时返回出现次数最多的特征值**

**return majorityCnt(classList)**

**bestFeat=chooseBestFeatureToSplit(dataSet) #选择最优特征**

**bestFeatLabel=labels[bestFeat] #取最优特征值**

**myTree={bestFeatLabel:{}} #创建树，以字典类型存储树的信息**

**del(labels[bestFeat]) #删除该特征**

**featValues=[example[bestFeat] for example in treeSet] #得到列包含的所有特征值**

**uniqueVals=set(featValues) #除去重复的特征值**

**for value in uniqueVals: #递归创建树(构造数据字典的过程)**

**subLabels=labels[:]**

**myTree[bestFeatLabel][value]=createTree(dataSet,splitDataSet\**

**(treeSet,bestFeat,value),subLabels)**

**return myTree**

**'''**

**-------------**

**构造注解树**

**-------------**

**'''**

**def getNumLeafs(myTree):**

**"""**

**获取叶节点的数目**

**@:param myTree 创建后的树**

**@:return numLeafs 返回叶节点的数目**

**"""**

**numLeafs = 0**

**firstStr = list(myTree.keys())[0]**

**secondDict = myTree[firstStr]**

**for key in secondDict.keys():**

**if type(secondDict[key]) is dict: #不是子节点**

**numLeafs += getNumLeafs(secondDict[key])**

**else:**

**numLeafs += 1 #统计子节点**

**return numLeafs**

**def getTreeDepth(myTree):**

**"""**

**获取树的层数**

**@:param myTree 创建的树**

**@:return maxDepth 树的最大深度**

**"""**

**maxDepth = 0**

**firstStr = list(myTree.keys())[0]**

**secondDict = myTree[firstStr]**

**for key in secondDict.keys():**

**if type(secondDict[key]) is dict: #还有子节点**

**thisDepth = 1 + getTreeDepth(secondDict[key])**

**else:**

**thisDepth = 1**

**if thisDepth > maxDepth: #是否为最深点**

**maxDepth = thisDepth**

**return maxDepth**

**def plotMidText(cntrPt,parentPt,txtString):**

**"""**

**计算父节点和子节点的中间位置，并在此处添加简单的文本标签信息**

**@:param cntrPt 子节点**

**@:param parentPt 父节点**

**@:param txtString 标签值**

**"""**

**xMid = (parentPt[0] - cntrPt[0])/2.0 + cntrPt[0] #计算标签的横值**

**yMid = (parentPt[1] - cntrPt[1])/2.0 + cntrPt[1] #计算标签的纵值**

**plotBestFit.ax1.text(xMid,yMid,txtString) #插值操作**

**dicisionNode = {'boxstyle': "sawtooth", 'fc': "0.8"}**

**leafNode = {'boxstyle': "round4", 'fc': "0.8"}**

**arrow\_args = {'arrowstyle': "<-"}**

**def plotNode(nodeTxt,centerPt,parentPt,nodeType):**

**"""**

**执行了实际的绘图功能**

**@:param nodeTxt 节点值**

**@:param centerPt 起始点**

**@:param parentPt 终止点**

**@:param nodeType 节点类型**

**"""**

**plotBestFit.ax1.annotate(nodeTxt,xy=parentPt,**

**xycoords='axes fraction',**

**xytext=centerPt,textcoords='axes fraction',**

**va="center",ha="center",bbox=nodeType,arrowprops=arrow\_args)**

**def plotTree(myTree,parentPt,nodeTxt):**

**"""**

**创建树图**

**@:param myTree 数据字典**

**@:param parentPt 起始位置**

**"""**

**numLeafs = getNumLeafs(myTree) #获取叶节点的数目**

**depth = getTreeDepth(myTree) #获取树的层数**

**firstStr = list(myTree.keys())[0]**

**cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW,\**

**plotTree.yOff) #计算位置**

**plotMidText(cntrPt,parentPt,nodeTxt) #插入标签**

**plotNode(firstStr,cntrPt,parentPt,dicisionNode) #实现绘图功能**

**secondDict = myTree[firstStr]**

**plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD #更新纵值**

**for key in secondDict.keys():**

**if type(secondDict[key]) is dict: #是数据字典**

**plotTree(secondDict[key],cntrPt,str(key)) #递归调用**

**else: #是叶节点**

**plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW #更新横值**

**plotNode(secondDict[key],(plotTree.xOff,plotTree.yOff),cntrPt,leafNode) #实现绘图功能**

**plotMidText((plotTree.xOff,plotTree.yOff),cntrPt,str(key)) #插入标签**

**plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD #更新纵值**

**def plotBestFit(dataArr,inTree,labelMat1,labelMat2):**

**"""**

**分类效果展示**

**@:param weights 回归系数**

**@:param path 数据文件路径**

**@:return null**

**"""**

**n = len(dataArr) #取行数**

**xcord1 = []; ycord1 = []**

**xcord2 = []; ycord2 = []**

**xcord3 = []; ycord3 = []**

**xcord4 = []; ycord4 = []**

**for i in range(n): #将训练前的数据分类存储**

**if int(labelMat1[i])== 1:**

**xcord1.append(dataArr[i][0]); ycord1.append(dataArr[i][1])**

**else:**

**xcord2.append(dataArr[i][0]); ycord2.append(dataArr[i][1])**

**for i in range(n): #将训练后的数据分类存储**

**if int(labelMat2[i])== 1:**

**xcord3.append(dataArr[i][0]); ycord3.append(dataArr[i][1])**

**else:**

**xcord4.append(dataArr[i][0]); ycord4.append(dataArr[i][1])**

**"""**

**创建树图**

**"""**

**fig = plt.figure('DecisionTree1')**

**fig.clf()**

**axprops = {'xticks': [], 'yticks': []}**

**plotBestFit.ax1 = plt.subplot(111, frameon=False, \*\*axprops)**

**plotTree.totalW = float(getNumLeafs(inTree)) # 存储树的宽度**

**plotTree.totalD = float(getTreeDepth(inTree)) # 存储树的深度**

**plotTree.xOff = -0.5 / plotTree.totalW;**

**plotTree.yOff = 1.0 # 追踪已经绘制的节点位置**

**plotTree(inTree, (0.5, 1.0), '') # 显示字典数据**

**"""**

**决策树预测结果**

**"""**

**fig = plt.figure("DecisionResult") #新建一个画图窗口**

**ax = fig.add\_subplot(111) #添加一个子窗口**

**ax.set\_title('Forecast')**

**ax.scatter(xcord3, ycord3, s=30, c='red', marker='s')**

**ax.scatter(xcord4, ycord4, s=30, c='green')**

**plt.xlabel('X1'); plt.ylabel('X2')**

**plt.figure("DecisionBefore")**

**plt.title('Original')**

**plt.scatter(xcord1, ycord1, s=30, c='red', marker='s')**

**plt.scatter(xcord2, ycord2, s=30, c='green')**

**plt.xlabel('X1');plt.ylabel('X2')**

**plt.show()**

**def getResult(dataArr,feat):**

**h = []**

**for featVec in dataArr:**

**if((featVec[0]>feat[0]) and (featVec[1]>feat[1])):**

**h.append(0)**

**else:**

**h.append(1)**

**return h**

**def featuerSplit(trainingSet):**

**"""**

**对每一类特征求最佳分割点**

**:param trainingSet:训练集**

**:return: 返回每个特征的分割点**

**"""**

**baseEntropy = calcShannonEnt(trainingSet) # 求初始香农熵**

**featList = getSubCol(trainingSet, 0, 2) #取一和三列**

**feat1 = chooseBestNumberToSplit(baseEntropy, featList) # 求特征1最佳分割点**

**featList = getSubCol(trainingSet, 1, 2) #取二和三列**

**feat2 = chooseBestNumberToSplit(baseEntropy, featList) # 求特征2最佳分割点**

**return [feat1, feat2] #返回特征分割点**