|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名：张晨龙 | | 学号：1925050069 | 专业年级：2019级 | 班级：软工1班 |
| 分组： 无 | | 实验室： 曾宪梓楼505 | 指导教师：李霞 | 实验日期：2022.04.04 |
| **实验的准备阶段** | **课程名称** | 机器学习及与智能数据处理 | | |
| **实验名称** | 决策树-CART算法实现 | | |
| **实验目的** | 1. 掌握决策树算法的基本过程 2. 编程实现CART算法，并应用于具体案例 | | |
| **实验内容** | 编程实现CART算法，并根据如下训练集输出分类决策树：  C:\Users\Administrator\AppData\Roaming\Tencent\Users\8306743\QQ\WinTemp\RichOle\I}W60)(QGZ695J6H37PY`WY.png  并利用决策树预测一个无房产、单身、年收入55k的样本的分类结果。给出代码与运行结果图。 | | |
| **实验类型**  （打☑） | □验证性 □演示性 🗹设计性 □综合性 | | |
| **实验的重点、难点** | 决策树算法的算法内容，递归调用，决策树的特征选择指标 | | |
| **实验环境** | Python 3.8 | | |
| **实验的实施阶段** | **实验步骤及实验结果** | 1. 构造CART\_Tree.py中的CART决策树： class CARTTree(object):  def \_\_init\_\_(self):  self.tree = {} *# CART Tree* self.dataSet = [] *# 数据集* self.labels = [] *# 标签集* def getDataSet(self, dataset, labels):  self.dataSet = dataset  self.labels = labels   def train(self):  *# labels = copy.deepcopy(self.labels)* labels = self.labels[:]  self.tree = self.buildTree(self.dataSet, labels)   def buildTree(self, dataSet, labels):  classList = [ds[-1] for ds in dataSet] *# 提取样本的类别* if classList.count(classList[0]) == len(classList): *# 单一类别* return classList[0]  if len(dataSet[0]) == 1: *# 没有属性需要划分了* return self.classify(classList)  bestFeat, bestGroup = self.findBestSplit(dataSet) *# 选取最大增益的属性序号和分组(如果多于两个特征值)* bestFeatLabel = labels[bestFeat]  tree = {bestFeatLabel: {}} *# 构造一个新的树结点* del (labels[bestFeat]) *# 从总属性列表中去除最大增益属性* featValues = list(bestGroup) *# 抽取最大增益属性的取值列表* for value in featValues: *# 对于每一个属性类别* subLabels = labels[:]  subDataSet = self.splitDataSet(dataSet, bestFeat, value) *# 分裂结点* subTree = self.buildTree(subDataSet, subLabels) *# 递归构造子树* tree[bestFeatLabel][value] = subTree  return tree   *# 计算出现次数最多的类别标签* def classify(self, classList):  items = dict([(classList.count(i), i) for i in classList])  return items[max(items.keys())]   *# 判断元组中是否均为字符串* def dete\_str(self, data\_Tuple):  for elem in data\_Tuple:  if type(elem) != str:  return False  return True   def splitGroup(self, vals):  *# 判断属性是否有多个特征值,如果小于等于2个则直接返回* if len(vals) <= 2:  return [vals]  *# 否则应当对特征进行分组* vals = list(vals)  retList = []  ls\_len = len(vals)  if self.dete\_str(vals):  for i in range(ls\_len):  *# 如果是字符串，说明特征为婚姻状况，定义分组列表* retList.append((vals[i], tuple(elem for elem in vals if elem is not vals[i])))  else:  vals.sort()  *# 如果不是字符串，则是收入，定义收入的分组列表* for i in range(ls\_len - 1):  retList.append((tuple(vals[j] for j in range(i + 1)), tuple(vals[k] for k in range(i + 1, ls\_len))))  *# 返回分组的列表* return retList   *# 计算最优特征* def findBestSplit(self, dataset):  numFeatures = len(dataset[0]) - 1  baseGini = self.calcGini(dataset) *# 基础基尼系数* num = len(dataset) *# 样本总数* bestInfoGain = 0.0  bestGroup = None  bestFeat = -1 *# 初始化最优特征向量轴  # 遍历数据集各列，寻找最优特征轴* for i in range(numFeatures):  featValues = [ds[i] for ds in dataset]  uniqueFeatValues = set(featValues)  **"""  最重要的地方：对于CART决策树来说，多个特征值需要分组，从而形成二叉树，因此需提前先把某个属性的多个特征分为两组  """** groups = self.splitGroup(uniqueFeatValues)  *# 按列和唯一值，计算基尼系数* for group in groups:  newGini = 0.  for val in group:  subDataSet = self.splitDataSet(dataset, i, val)  prob = len(subDataSet) / float(num) *# 子集中的概率* newGini += prob \* self.calcGini(subDataSet)  infoGain = baseGini - newGini *# 信息增益* if infoGain > bestInfoGain: *# 挑选最大值* bestInfoGain = baseGini - newGini  bestFeat = i  *# 记录最佳的分组* bestGroup = group  *# 由于重新分组的缘故，需要把最佳的分组返回，便于生成二叉树* return bestFeat, bestGroup *# 返回最佳分组   # 从dataset数据集的feat特征中，选取值为value的数据* def splitDataSet(self, dataset, feat, values):  retDataSet = []  for featVec in dataset:  if type(values) is int:  values = (values,)  if featVec[feat] in values:  reducedFeatVec = featVec[:feat]  reducedFeatVec.extend(featVec[feat + 1:])  retDataSet.append(reducedFeatVec)  return retDataSet   *# 计算dataSet的基尼系数* def calcGini(self, dataSet):  num = len(dataSet) *# 样本集总数* classList = [c[-1] for c in dataSet] *# 抽取分类信息* labelCounts = {}  for cs in set(classList): *# 对每个分类进行计数* labelCounts[cs] = classList.count(cs)   Gini = 1.  for key in labelCounts:  prob = labelCounts[key] / float(num)  Gini -= prob \*\* 2  return Gini   *# 预测。对输入对象进行Cart Tree分类* def predict(self, tree, newObject):  *# 判断输入值是否为“dict”* while type(tree).\_\_name\_\_ == **'dict'**:  key = list(tree.keys())[0]  *# 对outcome进行一些操作* if key == **'income'**:  digi\_str = list(tree[key].keys())[0]  digit = float(digi\_str[1:] if digi\_str.startswith(**'<'**) else digi\_str[2:])  if newObject[**'income'**] < digit:  newObject[**'income'**] = **'<'** + str(digit)  else:  newObject[**'income'**] = **'>='** + str(digit)  for val in tree[key]:  if type(val) is tuple:  newObject[key] = val  tree = tree[key][newObject[key]]  return tree 2. 构造主函数和一些操作函数：   from CART\_Tree import CARTTree import treePlotter   def create\_dataset():  *# 定义训练集* dataset\_columns = [**'house'**, **'marriage'**, **'income'**, **'delinquency'**]  dataset\_samples = [[1, **"single"**, 125, **'no'**],  [0, **'married'**, 100, **'no'**],  [0, **'single'**, 70, **'no'**],  [1, **'married'**, 120, **'no'**],  [0, **'divorced'**, 95, **'yes'**],  [0, **'married'**, 60, **'no'**],  [1, **'divorced'**, 220, **'no'**],  [0, **'single'**, 85, **'yes'**],  [0, **'married'**, 75, **'no'**],  [0, **'single'**, 90, **'yes'**]]  return dataset\_samples, dataset\_columns   import numpy as np   def divide\_tree\_group(node, key):  *# 对于income做一些操作，使得结果更加符合任务* org\_ls = [tp for tp in node[key]]  group\_ls = [np.asarray(tp) for tp in org\_ls]  if group\_ls[0].min() > group\_ls[1].min():  div\_num = (group\_ls[0].min() + group\_ls[1].max()) / 2  new\_dict = {**'>='** + str(div\_num): {node[key][org\_ls[0]]}, **'<'** + str(div\_num): {node[key][org\_ls[1]]}}  else:  div\_num = (group\_ls[1].min() + group\_ls[0].max()) / 2  new\_dict = {**'<'** + str(div\_num): {node[key][org\_ls[0]]}, **'>='** + str(div\_num): {node[key][org\_ls[1]]}}  node[key] = new\_dict   def tree\_postprocess(Tree):  if type(Tree) is not dict:  return  for key in Tree.keys():  *# 递归深度优先遍历二叉树，找到'income'结点，并将其修改为中间值* if key == **'income'**:  divide\_tree\_group(Tree, key)  break  tree\_postprocess(Tree[key])   def main():  Cart\_Tree = CARTTree()  ds, labels = create\_dataset()  Cart\_Tree.getDataSet(ds, labels)  Cart\_Tree.train() *# 训练CART决策树* tree\_postprocess(Cart\_Tree.tree) *# 对于已生成的决策树进行一些后处理，使其符合题目要求* print(Cart\_Tree.tree) *# 输出CART决策树* print(Cart\_Tree.predict(Cart\_Tree.tree, {**'house'**: 0, **'marriage'**: **'single'**, **'income'**: 55}))  treePlotter.createPlot(Cart\_Tree.tree)   if \_\_name\_\_ == **'\_\_main\_\_'**:  main()   1. 构造画图函数treePlotter:   import matplotlib.pyplot as plt  decisionNode = dict(boxstyle=**"sawtooth"**, fc=**"0.8"**) leafNode = dict(boxstyle=**"round4"**, fc=**"0.8"**) arrow\_args = dict(arrowstyle=**"<-"**)   def plotNode(nodeTxt, centerPt, parentPt, nodeType):  createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords=**'axes fraction'**, \  xytext=centerPt, textcoords=**'axes fraction'**, \  va=**"center"**, ha=**"center"**, bbox=nodeType, arrowprops=arrow\_args)   def getNumLeafs(myTree):  numLeafs = 0  firstStr = list(myTree.keys())[0]  secondDict = myTree[firstStr]  for key in secondDict.keys():  if type(secondDict[key]).\_\_name\_\_ == **'dict'**:  numLeafs += getNumLeafs(secondDict[key])  else:  numLeafs += 1  return numLeafs   def getTreeDepth(myTree):  maxDepth = 0  firstStr = list(myTree.keys())[0]  secondDict = myTree[firstStr]  for key in secondDict.keys():  if type(secondDict[key]).\_\_name\_\_ == **'dict'**:  thisDepth = getTreeDepth(secondDict[key]) + 1  else:  thisDepth = 1  if thisDepth > maxDepth:  maxDepth = thisDepth  return maxDepth   def plotMidText(cntrPt, parentPt, txtString):  xMid = (parentPt[0] - cntrPt[0]) / 2.0 + cntrPt[0]  yMid = (parentPt[1] - cntrPt[1]) / 2.0 + cntrPt[1]  createPlot.ax1.text(xMid, yMid, txtString)   def plotTree(myTree, parentPt, nodeTxt):  numLeafs = getNumLeafs(myTree)  depth = getTreeDepth(myTree)  firstStr = list(myTree.keys())[0]  cntrPt = (plotTree.xOff + (1.0 + float(numLeafs)) / 2.0 / plotTree.totalw, plotTree.yOff)  plotMidText(cntrPt, parentPt, nodeTxt)  plotNode(firstStr, cntrPt, parentPt, decisionNode)  secondDict = myTree[firstStr]  plotTree.yOff = plotTree.yOff - 1.0 / plotTree.totalD  for key in secondDict.keys():  if type(secondDict[key]).\_\_name\_\_ == **'dict'**:  plotTree(secondDict[key], cntrPt, str(key))  else:  plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalw  plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)  plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))  plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD   def createPlot(inTree):  fig = plt.figure(1, facecolor=**'white'**)  fig.clf()  axprops = dict(xticks=[], yticks=[])  createPlot.ax1 = plt.subplot(111, frameon=False, \*\*axprops)  plotTree.totalw = float(getNumLeafs(inTree))  plotTree.totalD = float(getTreeDepth(inTree))  plotTree.xOff = -0.5 / plotTree.totalw  plotTree.yOff = 1.0  plotTree(inTree, (0.5, 1.0), **''**)  plt.show()   1. 运行结果： 2. CART决策树：      1. 预测结果： | | |
| **实验结果的处理阶段** | **实验结果的分析与总结** | 本次实验中我将ID3决策树修改为CART决策树，主要难点在于CART树作为一个二叉树，如果某个特征存在多于两个值，需要对该特征进行分组，来计算最大的基尼系数，我的思路是将分组作为特征一起输入到选择最优特征的函数中，这样可以避免大量代码的改动，只要在划分数据时增加划分的分组即可实现。第二个难点在于分组时不同的值含义不同，需要不一样的处理：如收入需要通过中值不断划分，而是婚姻状况则需要两两组合，需要对不同的组别进行判断再分组。在最后预测的时候因为输入和树的结构不同，需要再对CART树进行一次后处理，并对预测函数进行一些修改。  代码开源在github:  https://github.com/ZsirHenu/Machine\_Learning\_practice | | |