# 《人工智能》实验三报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 俞家宝 | 学号 | 2021117338 |
| 实验地点 | 实验室321 | 实验日期 | 2023/11/1 |

**一、实验内容**

Implement ID3 in a language of your choice and run it on the credit history example from  
the text。(自选一种语言实现ID3算法，数据库采用课本上预测个人财务风险的数据集)

**数据集如下：**





1. **实验原理**

**从根结点(root node)开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立子节点；再对子结点递归地调用以上方法，构建决策树；直到所有特征的信息增益均很小或没有特征可以选择为止，最后得到一个决策树。ID3相当于用极大似然法进行概率模型的选择。**

**递归返回条件：**

**（1）所有类的标签完全相同，则直接返回该类标签。**

**（2）使用完所有即当前属性集为空，仍不能将数据集划分成仅包含唯一类别的分组，则挑选出现次数最多的类别作为返回值**

1. **实验过程以及结果分析**

**Test.txt文件数据**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NO. | 风险 | 信用历史 | 债务 | 抵押 | 收入 |
| 1 | 高 | 坏 | 高 | 无 | 0-15000美元 |
| 2 | 高 | 未知 | 高 | 无 | 15000-35000美元 |
| 3 | 中等 | 未知 | 低 | 无 | 15000-35000美元 |
| 4 | 高 | 未知 | 低 | 无 | 0-15000美元 |
| 5 | 低 | 未知 | 低 | 无 | 超过35000美元 |
| 6 | 低 | 未知 | 低 | 充分 | 超过35000美元 |
| 7 | 高 | 坏 | 低 | 无 | 0-15000美元 |
| 8 | 中等 | 坏 | 低 | 充分 | 超过35000美元 |
| 9 | 低 | 好 | 低 | 无 | 超过35000美元 |
| 10 | 低 | 好 | 高 | 充分 | 超过35000美元 |
| 11 | 高 | 好 | 高 | 无 | 0-15000美元 |
| 12 | 中等 | 好 | 高 | 无 | 15000-35000美元 |
| 13 | 低 | 好 | 高 | 无 | 超过35000美元 |
| 14 | 高 | 坏 | 高 | 无 | 15000-35000美元 |

**代码:**

# -\*- coding: UTF-8 -\*-  
from math import log  
import operator  
  
"""  
进行标注  
"""  
credit = {"好": 0, "坏": 1, "未知": 2}  
debt = {"高": 0, "低": 1}  
mortgage = {"无": 0, "充分": 1}  
income = {"0-15000美元": 0, "15000-35000美元": 1, "超过35000美元": 2}  
C2E = {"风险": "risk", "信用历史": "credit", "债务": "debt", "抵押": "mortgage", "收入": "income"}  
  
"""  
函数说明:创建测试数据集  
读取本地文件test.txt  
"""  
  
  
def createDataSet():  
 data = []  
 with open('test.txt', 'r', encoding='utf-8') as file:  
 lines = file.readlines()  
 labels = lines[0].strip().split(' ')[1:]  
 for line in lines[1:]:  
 line = line.strip()  
 if line:  
 entry = line.split(' ')[1:]  
 entry = exchange(entry, labels)  
 data.append(entry)  
 dataSet = [row[1:] + [row[0]] for row in data]  
 labels = labels[1:] + [labels[0]]  
 return dataSet, labels # 返回数据集和分类属性  
  
  
"""  
对读入的数据进行标注  
"""  
  
  
def exchange(data, label):  
 size = len(data)  
 for i in range(1, size):  
 t = label[i]  
 data[i] = globals()[C2E[t]][data[i]]  
 return data  
  
  
"""  
函数说明:计算给定数据集的经验熵(香农熵)  
Parameters:  
 dataSet - 数据集  
Returns:  
 shannonEnt - 经验熵(香农熵)  
"""  
  
  
def calcShannonEnt(dataSet):  
 numEntires = len(dataSet) # 返回数据集的行数  
 labelCounts = {} # 保存每个标签(Label)出现次数的字典  
 for featVec in dataSet: # 对每组特征向量进行统计  
 currentLabel = featVec[-1] # 提取标签(Label)信息  
 if currentLabel not in labelCounts.keys(): # 如果标签(Label)没有放入统计次数的字典,添加进去  
 labelCounts[currentLabel] = 0  
 labelCounts[currentLabel] += 1 # Label计数  
 shannonEnt = 0.0 # 经验熵(香农熵)  
 for key in labelCounts: # 计算香农熵  
 prob = float(labelCounts[key]) / numEntires # 选择该标签(Label)的概率  
 shannonEnt -= prob \* log(prob, 2) # 利用公式计算  
 return shannonEnt # 返回经验熵(香农熵)  
  
  
"""  
函数说明:按照给定特征划分数据集  
Parameters:  
 dataSet - 待划分的数据集  
 axis - 划分数据集的特征  
 value - 需要返回的特征的值  
"""  
  
  
def splitDataSet(dataSet, axis, value):  
 retDataSet = [] # 创建返回的数据集列表  
 for featVec in dataSet: # 遍历数据集  
 if featVec[axis] == value:  
 reducedFeatVec = featVec[:axis] # 去掉axis特征  
 reducedFeatVec.extend(featVec[axis + 1:]) # 将符合条件的添加到返回的数据集  
 retDataSet.append(reducedFeatVec)  
 return retDataSet # 返回划分后的数据集  
  
  
"""  
函数说明:选择最优特征  
Parameters:  
 dataSet - 数据集  
Returns:  
 bestFeature - 信息增益最大的(最优)特征的索引值  
"""  
  
  
def chooseBestFeatureToSplit(dataSet):  
 numFeatures = len(dataSet[0]) - 1 # 特征数量  
 baseEntropy = calcShannonEnt(dataSet) # 计算数据集的香农熵  
 bestInfoGain = 0.0 # 信息增益  
 bestFeature = -1 # 最优特征的索引值  
 for i in range(numFeatures): # 遍历所有特征  
 # 获取dataSet的第i个所有特征  
 featList = [example[i] for example in dataSet]  
 uniqueVals = set(featList) # 创建set集合{},元素不可重复  
 newEntropy = 0.0 # 经验条件熵  
 for value in uniqueVals: # 计算信息增益  
 subDataSet = splitDataSet(dataSet, i, value) # subDataSet划分后的子集  
 prob = len(subDataSet) / float(len(dataSet)) # 计算子集的概率  
 newEntropy += prob \* calcShannonEnt(subDataSet) # 根据公式计算经验条件熵  
 infoGain = baseEntropy - newEntropy # 信息增益  
 print("第%d个特征的增益为%.3f" % (i, infoGain)) # 打印每个特征的信息增益  
 if (infoGain > bestInfoGain): # 计算信息增益  
 bestInfoGain = infoGain # 更新信息增益，找到最大的信息增益  
 bestFeature = i # 记录信息增益最大的特征的索引值  
 return bestFeature # 返回信息增益最大的特征的索引值  
  
  
"""  
函数说明:统计classList中出现此处最多的元素(类标签)  
Parameters:  
 classList - 类标签列表  
Returns:  
 sortedClassCount[0][0] - 出现此处最多的元素(类标签)  
"""  
  
  
def majorityCnt(classList):  
 classCount = {}  
 for vote in classList: # 统计classList中每个元素出现的次数  
 if vote not in classCount.keys():  
 classCount[vote] = 0  
 classCount[vote] += 1  
 sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True) # 根据字典的值降序排序  
 return sortedClassCount[0][0] # 返回classList中出现次数最多的元素  
  
  
"""  
函数说明:递归构建决策树  
Parameters:  
 dataSet - 训练数据集  
 labels - 分类属性标签  
 featLabels - 存储选择的最优特征标签  
Returns:  
 myTree - 决策树  
"""  
  
  
def createTree(dataSet, labels, featLabels):  
 classList = [example[-1] for example in dataSet] # 取分类标签(是否放贷:yes or no)  
 if classList.count(classList[0]) == len(classList): # 如果类别完全相同则停止继续划分  
 return classList[0]  
 if len(dataSet[0]) == 1: # 遍历完所有特征时返回出现次数最多的类标签  
 return majorityCnt(classList)  
 bestFeat = chooseBestFeatureToSplit(dataSet) # 选择最优特征  
 bestFeatLabel = labels[bestFeat] # 最优特征的标签  
 if bestFeatLabel not in featLabels:  
 featLabels.append(bestFeatLabel)  
 myTree = {bestFeatLabel: {}} # 根据最优特征的标签生成树  
 del (labels[bestFeat]) # 删除已经使用特征标签  
 featValues = [example[bestFeat] for example in dataSet] # 得到训练集中所有最优特征的属性值  
 uniqueVals = set(featValues) # 去掉重复的属性值  
 for value in uniqueVals:  
 subLabels = labels[:]  
 # 递归调用函数createTree(),遍历特征，创建决策树。  
 myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels, featLabels)  
 return myTree  
  
  
"""  
函数说明:使用决策树执行分类  
Parameters:  
 inputTree - 已经生成的决策树  
 featLabels - 存储选择的最优特征标签  
 testVec - 测试数据列表，顺序对应最优特征标签  
Returns:  
 classLabel - 分类结果  
"""  
  
  
def classify(inputTree, featLabels, testVec):  
 firstStr = next(iter(inputTree)) # 获取决策树结点  
 secondDict = inputTree[firstStr] # 下一个字典  
 featIndex = featLabels.index(firstStr)  
 for key in secondDict.keys():  
 if testVec[featIndex] == key:  
 if type(secondDict[key]).\_\_name\_\_ == 'dict':  
 classLabel = classify(secondDict[key], featLabels, testVec)  
 else:  
 classLabel = secondDict[key]  
 return classLabel  
  
  
def chooseInput(type):  
 print("请输入" + type + "等级:")  
 if type == "信用历史":  
 print("(0:好 or 1:坏 or 2:未知)")  
 if type == "收入":  
 print("(0:0-15000美元 or 1:15000-35000美元 or 2:超过35000美元)")  
 if type == "债务":  
 print("(0:高 or 1:低)")  
 if type == "抵押":  
 print("(0:无 or 1:充分)")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 dataSet, labels = createDataSet()  
 featLabels = []  
 myTree = createTree(dataSet, labels, featLabels)  
 print(myTree)  
 testVec = [] # 测试数据  
 for i in featLabels:  
 chooseInput(i)  
 testVec.append(int(input()))  
 result = classify(myTree, featLabels, testVec)  
 print('风险等级为：' + result)

**运行截图:**

**测试用例:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 风险 | 信用历史 | 债务 | 抵押 | 收入 |
| 高 | 坏 | 高 | 无 | 0-15000美元 |
| 高 | 未知 | 高 | 无 | 15000-35000美元 |
| 中等 | 未知 | 低 | 无 | 15000-35000美元 |

**测试结果:**

**决策树:**

- 收入: 高

- 1:

- 信用历史:

- 0: 中等

- 1: 高

- 2:

- 债务:

- 0: 高

- 1: 中等

- 2:

- 信用历史:

- 0: 低

- 1: 中等

- 2: 低

请输入收入等级: (0:0-15000美元 or 1:15000-35000美元 or 2:超过35000美元)

0

请输入信用历史等级:

(0:好 or 1:坏 or 2:未知)

1

请输入债务等级:

(0:高 or 1:低)

0

风险等级为：高

请输入收入等级:

(0:0-15000美元 or 1:15000-35000美元 or 2:超过35000美元)

1

请输入信用历史等级:

(0:好 or 1:坏 or 2:未知)

2

请输入债务等级:

(0:高 or 1:低)

1

风险等级为：中等

请输入收入等级:

(0:0-15000美元 or 1:15000-35000美元 or 2:超过35000美元)

1

请输入信用历史等级:

(0:好 or 1:坏 or 2:未知)

2

请输入债务等级:

(0:高 or 1:低)

0

风险等级为：高

1. **实验总结**

**实现了ID3决策树算法,在进行熵值计算还有标注等阶段时遇到很大困难,在执行递归的时候很大的拖延了进度,通过上网查阅资料,最终解决了问题,成功实现了该算法.**