

**人工智能原理作业报告**

**学号： 2018214789**

**姓名： 孙东东**

**班级： 软件18-1班**

**指导老师： 王浩**

2020/10/20

目录

[一、作业任务 3](#_Toc5930)

[二、 运行环境 4](#_Toc9594)

[三、算法介绍 4](#_Toc2402)

[四、程序分析 6](#_Toc27795)

[1. 制作数据集： 6](#_Toc9207)

[2. 输出决策树结果 6](#_Toc29136)

[3. 可视化决策树： 8](#_Toc10088)

[五、 界面截图与分析 9](#_Toc30637)

[1.通过图来大致观察一下不同属性的划分情况： 9](#_Toc2150)

[2.查看属性对于结果的划分影响： 12](#_Toc5011)

[3.程序运行控制台输出结果： 14](#_Toc13360)

[4.决策树可视化结果： 15](#_Toc23898)

[六、心得体会 15](#_Toc3215)

[七、参考资料 16](#_Toc23495)

[八、附录 16](#_Toc197)

[代码 16](#_Toc7754)

# 一、作业任务

1．编程实现ID3算法，针对下表数据，生成决策树。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | color | size | act | age | inflated |
| 1 | YELLOW | SMALL | STRETCH | ADULT | T |
| 2 | YELLOW | SMALL | STRETCH | CHILD | T |
| 3 | YELLOW | SMALL | DIP | CHILD | F |
| 4 | YELLOW | LARGE | STRETCH | ADULT | T |
| 5 | YELLOW | LARGE | DIP | ADULT | T |
| 6 | YELLOW | LARGE | DIP | CHILD | F |
| 7 | PURPLE | SMALL | STRETCH | CHILD | T |
| 8 | PURPLE | SMALL | DIP | ADULT | T |
| 9 | PURPLE | SMALL | DIP | CHILD | F |
| 10 | PURPLE | LARGE | STRETCH | CHILD | T |

**问题提示：**可设计数据文件格式，如color属性取值YELLOW：0，PURPLE：1等，程序从指定数据文件中读取训练集数据。

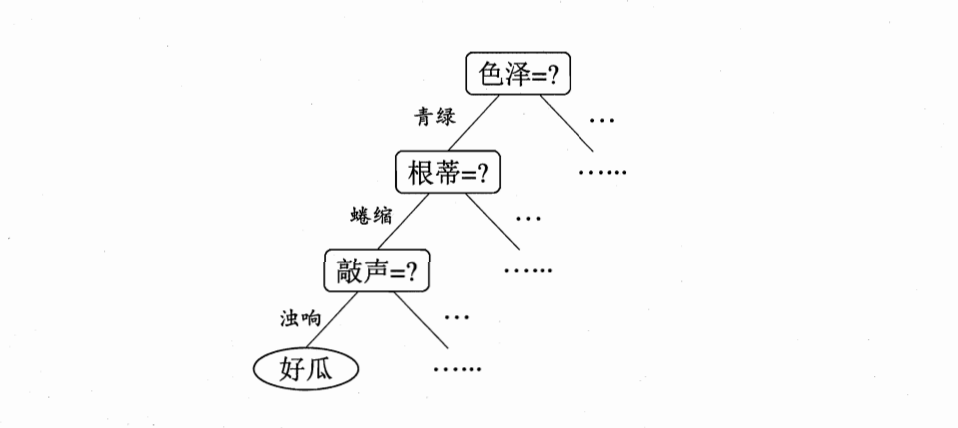
**问题拓展：**要求将计算各属性信息增益过程及决策树生成过程演示出来。

# 运行环境

1. 编程语言：Python
2. 使用第三方库：Numpy，Matplotlib，Scikit-learn
3. IDE：PyCharm
4. 操作系统：WIndows10

# 三、算法介绍

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。引用《机器学习》（西瓜书）中的例子：



图片来源：周志华《机器学习》

通过属性来逐渐地把西瓜分到对应的类别中去，决策树的学习过程就是在寻找最优决策方案的过程。

基本的算法流程：

1. 输入：

训练集D={(x1,y1),(x2,y2),(x3,y3),(x4,y4), ... , (xm,ym)}

属性集A={a1,a2,a3,a4, ... , a5}

1. 过程：

函数TreeGenerate(D, A)

生成节点

if D 中样本全属于同一类别C then

将node标记为C类叶子节点；return

end if

从A中选择最优划分属性a\*；

for a\* 的每一个a’ do

为node生成一个分支；另Dv表示D中在a\*上取值为a’的end样本子集；

if Dv为空 then

将分支结点标记为叶结点，其类别标记为D中样本最多的类；return

else

以TreeGenerate（Dv，A\{a\*}）为分支结点

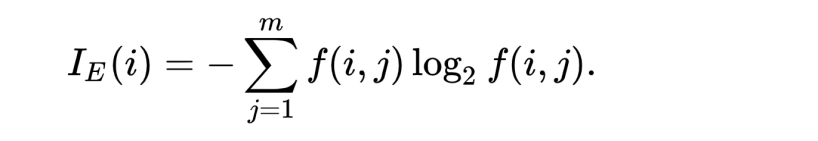
end if

end for

1. 输出：

以node为根结点的一棵决策树

ID3是决策树算法的一种，它是以信息熵和信息增益作为衡量标准



# 四、程序分析

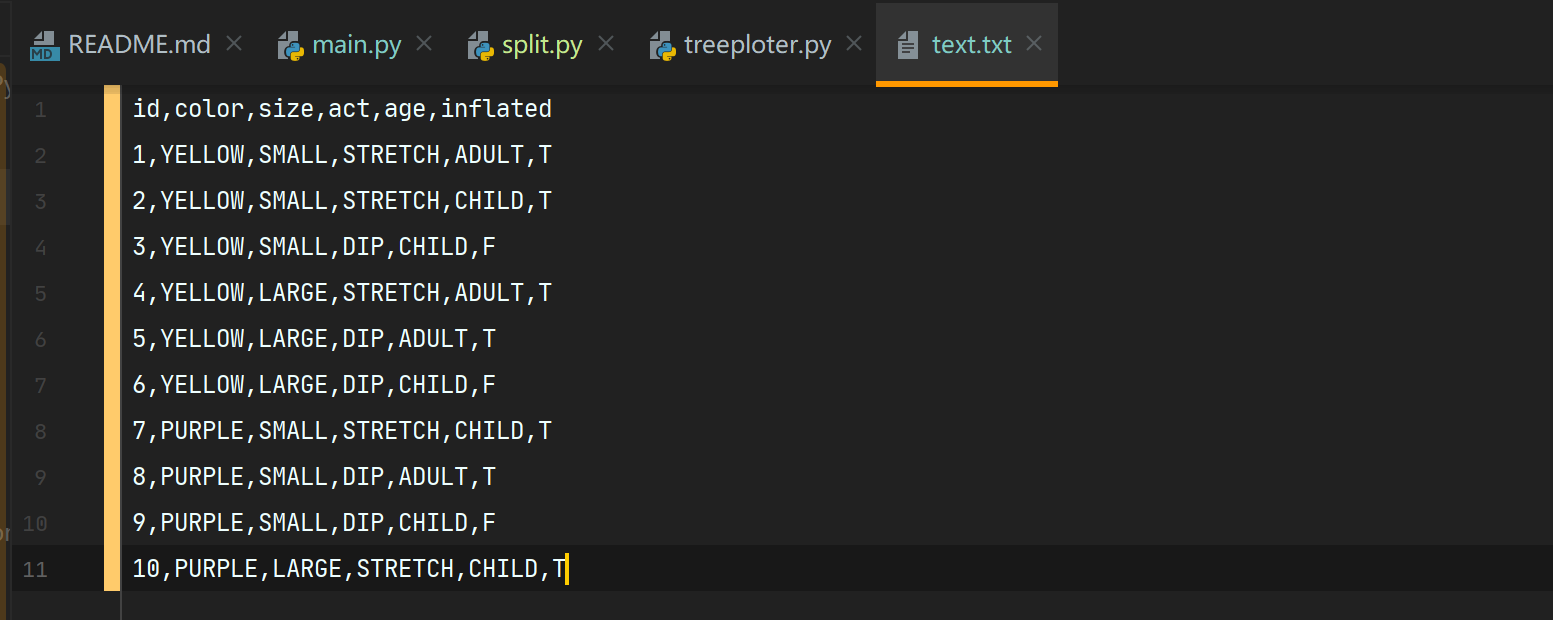
## 制作数据集：

因为ID对于实际环境并没有什么参考价值，所以这次作业数据集我将ID这个属性去掉

1. fr = open(r'data/text.txt')
2. listWm = [inst.strip().split(',')[1:] **for** inst **in** fr.readlines()[1:]]

通过python的切片功能，把每一行数据中的id去掉（通过slide[1:]来实现）

数据集截图：



## 输出决策树结果

1. **print**(json.dumps(Trees, ensure\_ascii=False))

Json格式 {

"act": {

"DIP": {

"age": {

"CHILD": "F",

"ADULT": "T"

}

},

"STRETCH": "T"

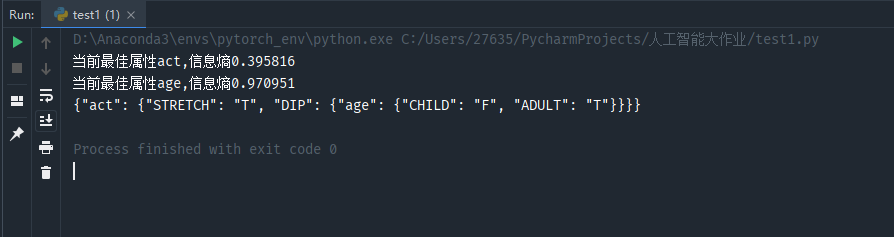
}

}

程序运行过程中得出 act的决策能力大于age（因为act的信息熵<age的信息熵）

1. **def** createTree(dataSet, labels):
2. classList = [example[-1] **for** example **in** dataSet]  # 类别向量
3. **if** classList.count(classList[0]) == len(classList):  # 如果只有一个类别，返回
4. **return** classList[0]
5. **if** len(dataSet[0]) == 1:  # 如果所有特征都被遍历完了，返回出现次数最多的类别
6. **return** majorityCnt(classList)
7. bestFeat,bestGain = chooseBestFeatureToSplit(dataSet)  # 最优划分属性的索引
8. bestFeatLabel = labels[bestFeat]  # 最优划分属性的标签
9. **print**("当前最佳属性%s,信息熵%f" % (bestFeatLabel,bestGain))
10. myTree = {bestFeatLabel: {}}
11. **del** (labels[bestFeat])  # 已经选择的特征不再参与分类
12. featValues = [example[bestFeat] **for** example **in** dataSet]
13. uniqueValue = set(featValues)  # 该属性所有可能取值，也就是节点的分支
14. **for** value **in** uniqueValue:  # 对每个分支，递归构建树
15. subLabels = labels[:]
16. myTree[bestFeatLabel][value] = createTree(
17. splitDataSet(dataSet, bestFeat, value), subLabels)
18. **return** myTree

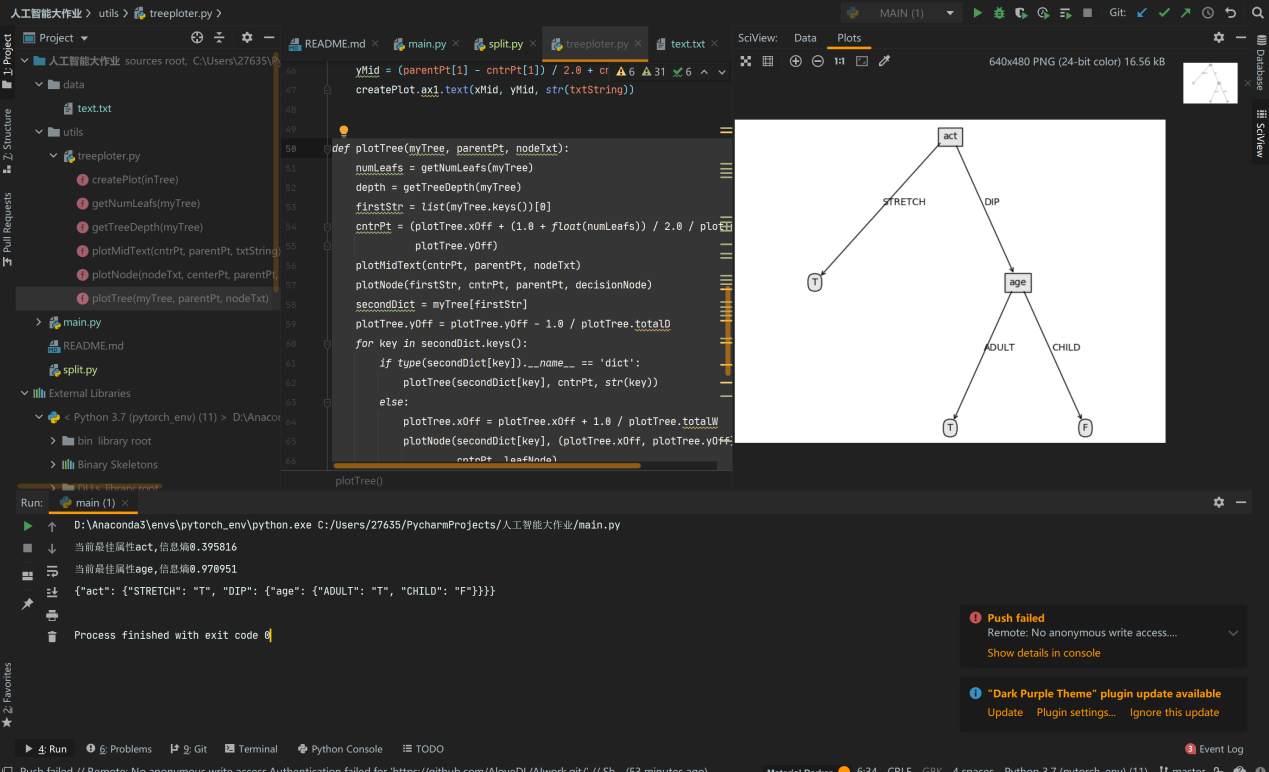
根据当前属性计算信息增益的值，并将结果输出出来



## 可视化决策树：

1. **def** plotTree(myTree, parentPt, nodeTxt):
2. numLeafs = getNumLeafs(myTree)
3. depth = getTreeDepth(myTree)
4. firstStr = list(myTree.keys())[0]
5. cntrPt = (plotTree.xOff + (1.0 + float(numLeafs)) / 2.0 / plotTree.totalW,
6. plotTree.yOff)
7. plotMidText(cntrPt, parentPt, nodeTxt)
8. plotNode(firstStr, cntrPt, parentPt, decisionNode)
9. secondDict = myTree[firstStr]
10. plotTree.yOff = plotTree.yOff - 1.0 / plotTree.totalD
11. **for** key **in** secondDict.keys():
12. **if** type(secondDict[key]).\_\_name\_\_ == 'dict':
13. plotTree(secondDict[key], cntrPt, str(key))
14. **else**:
15. plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalW
16. plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff),
17. cntrPt, leafNode)
18. plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
19. plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD

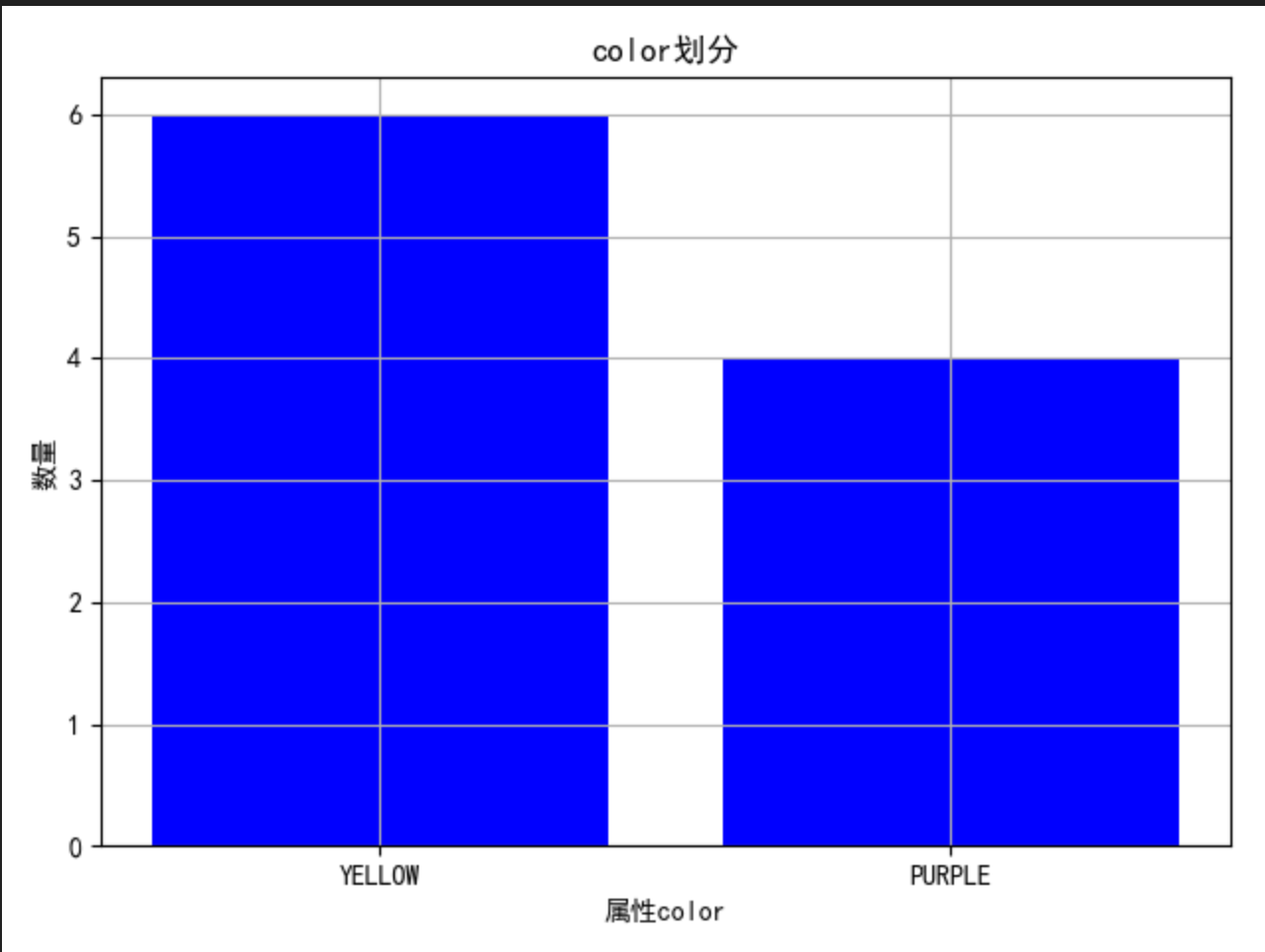
使用第三方库matplotlib画图



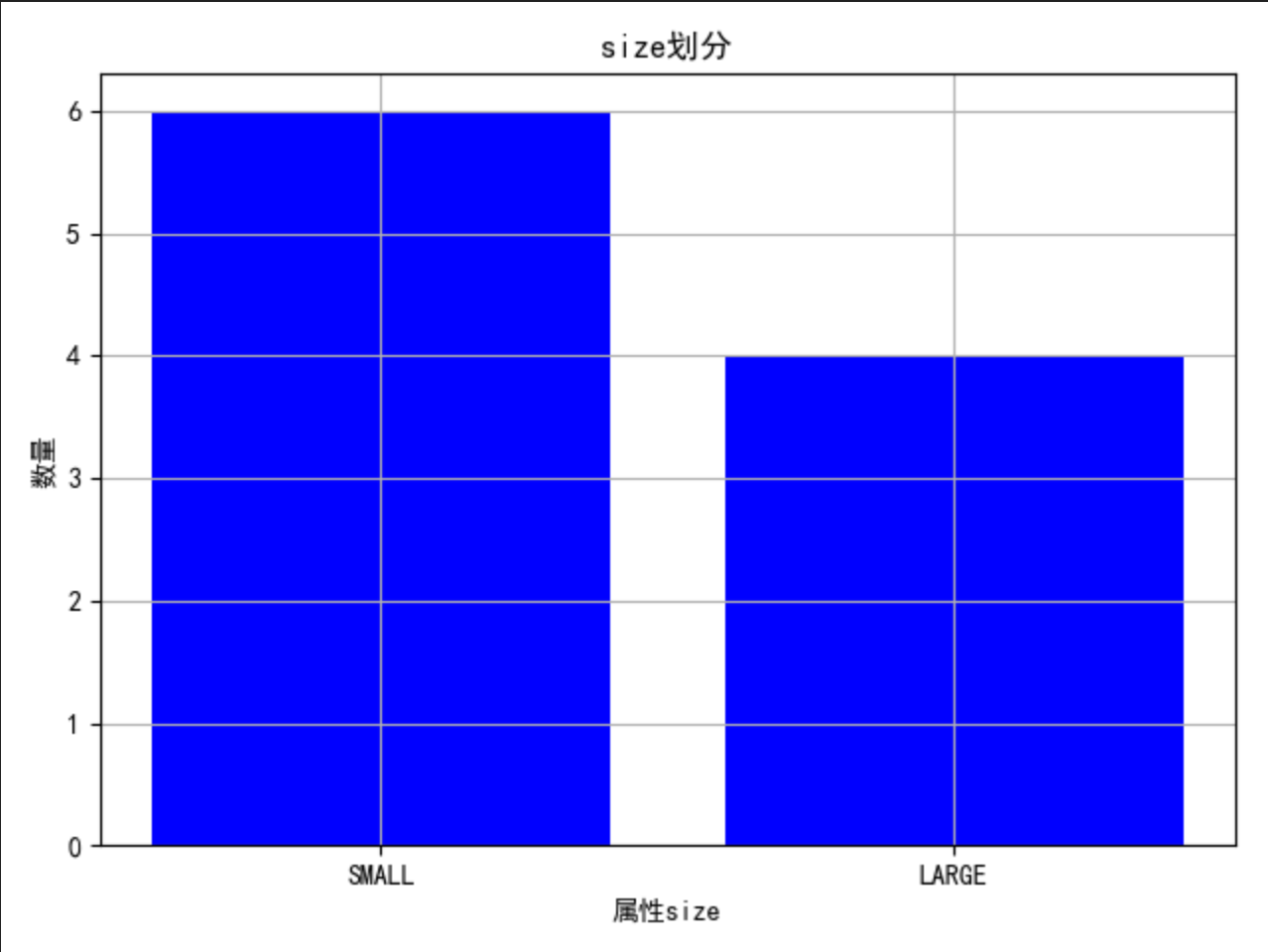
# 界面截图与分析

## 1.通过图来大致观察一下不同属性的划分情况：

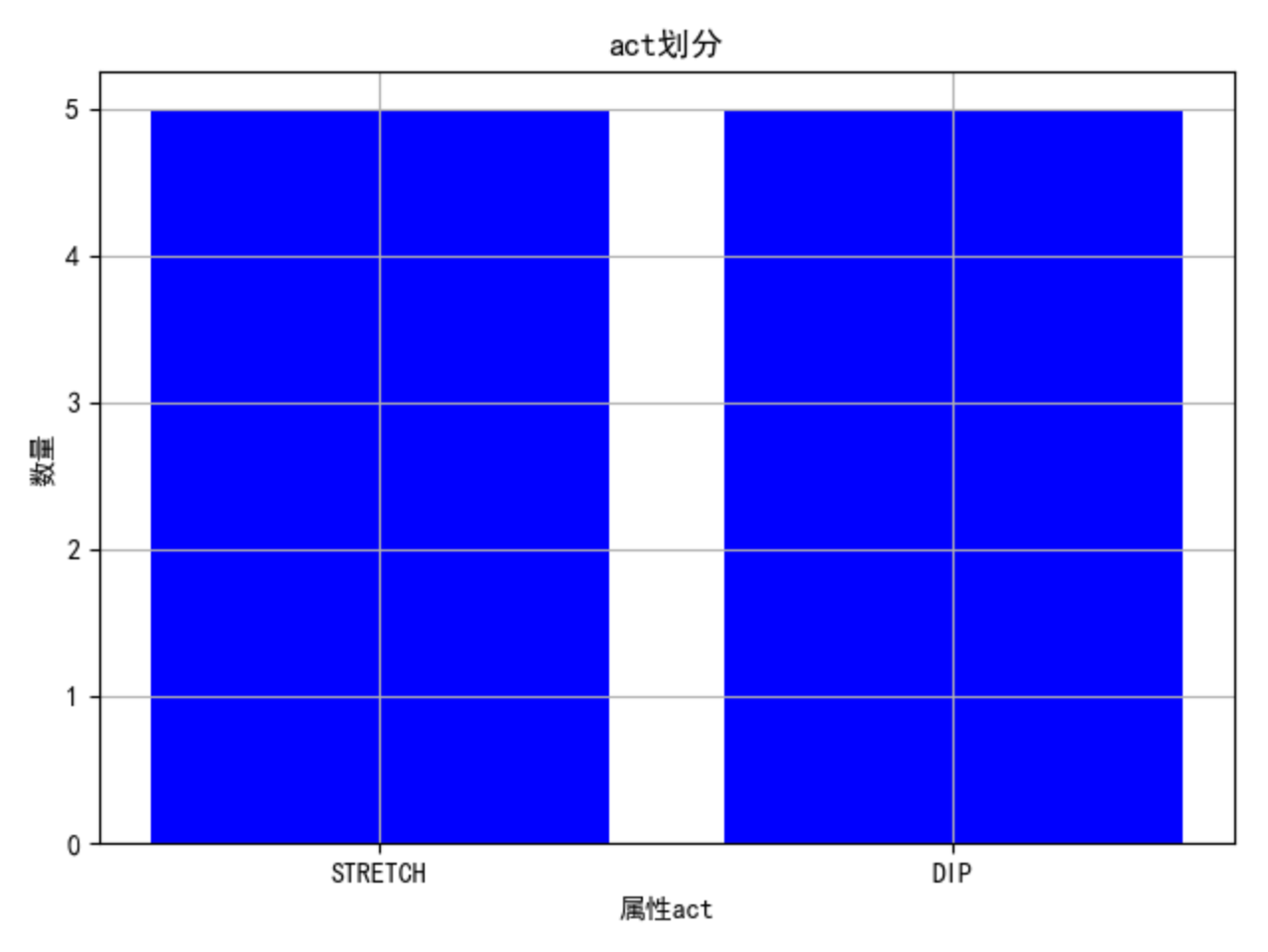
color：



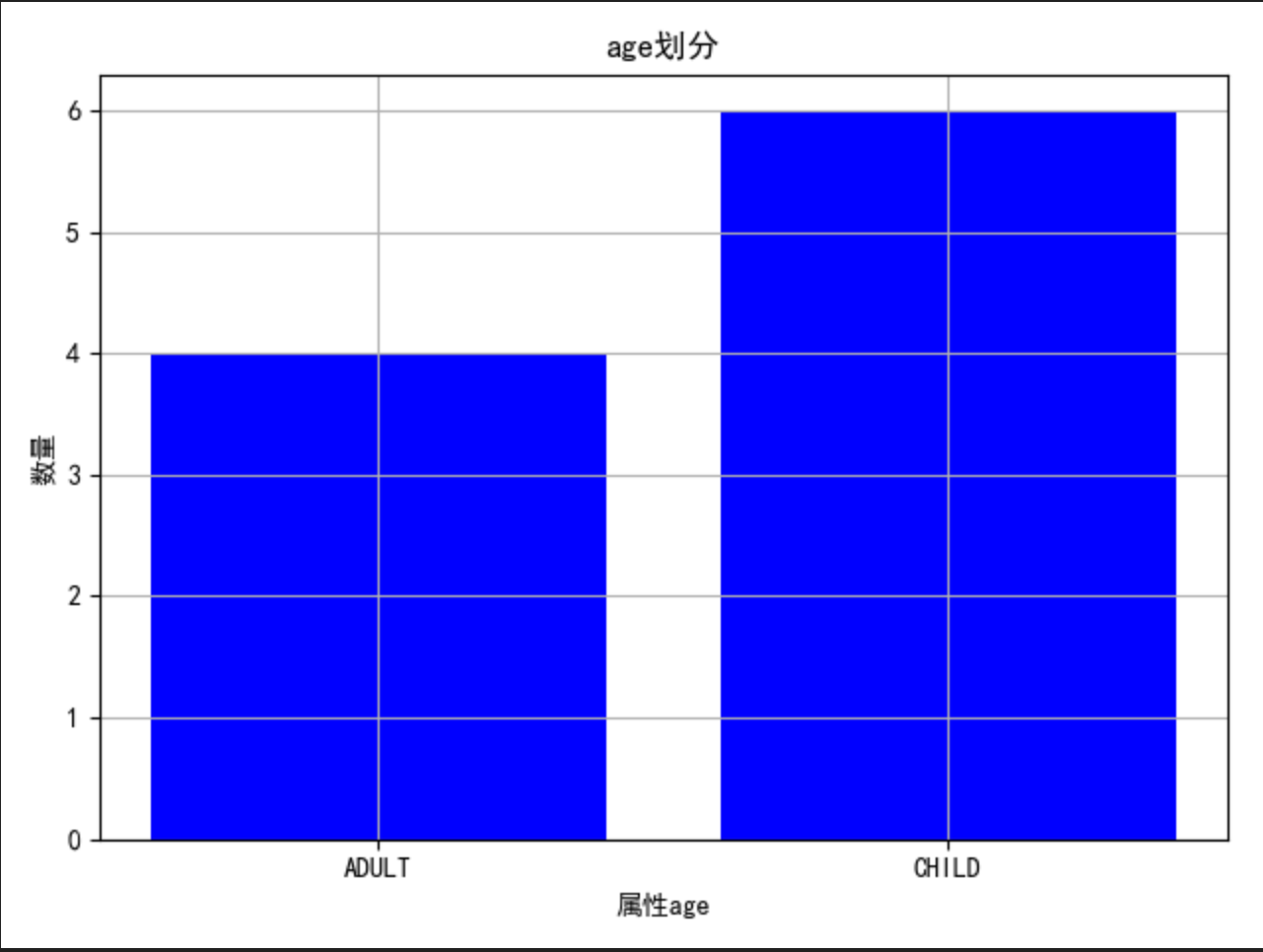
size：



act：

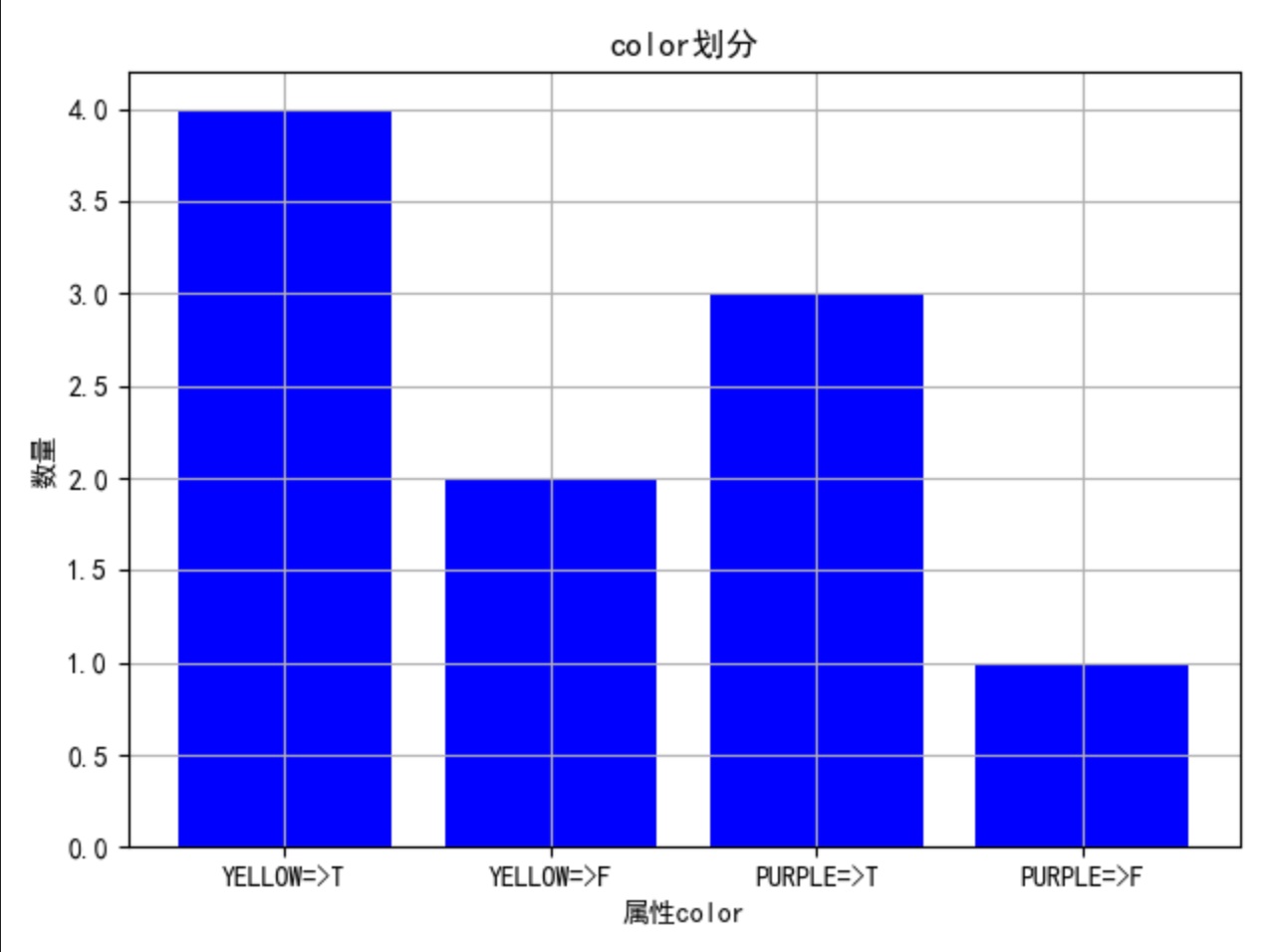


age：



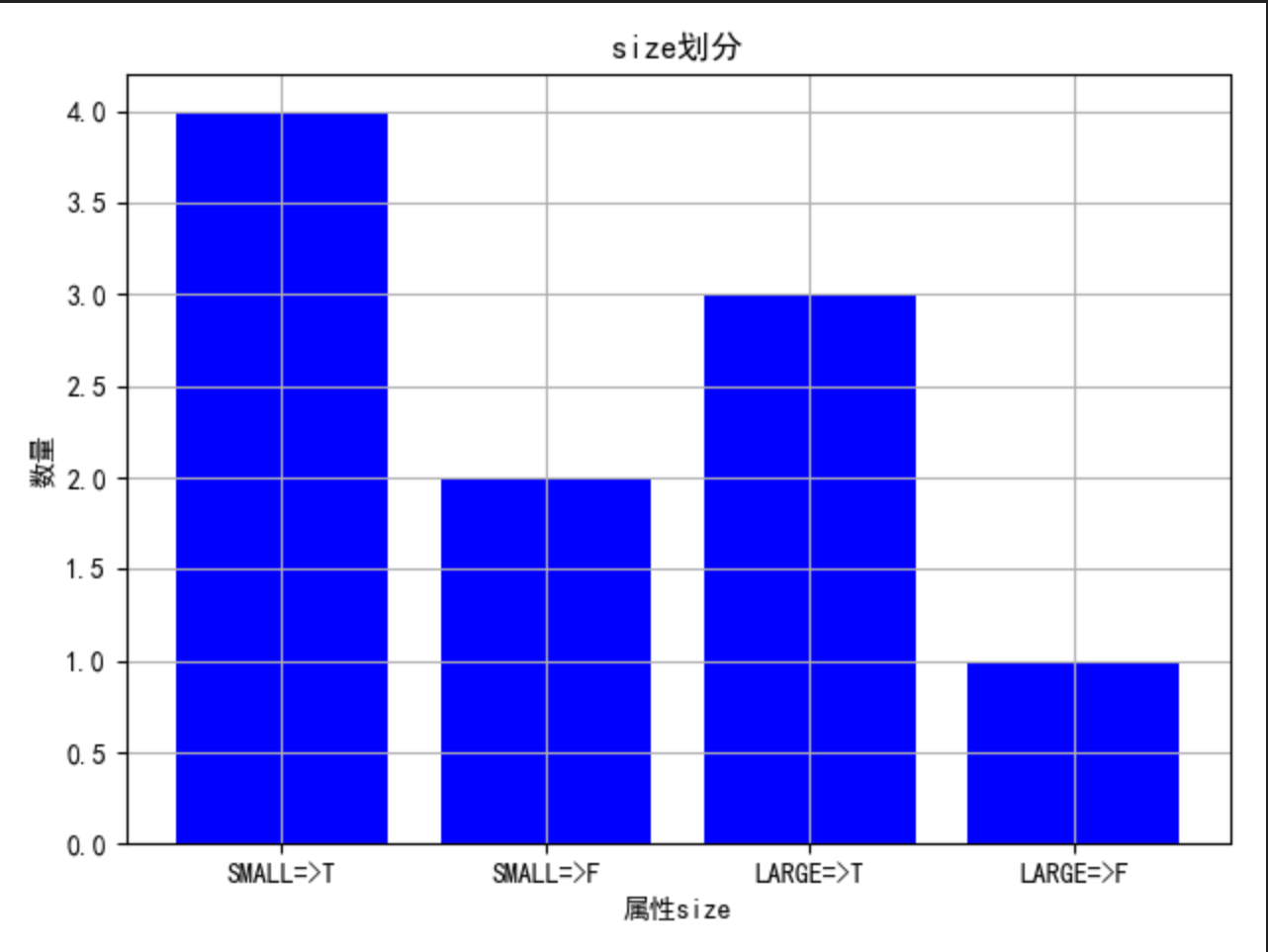
## 2.查看属性对于结果的划分影响：

color:



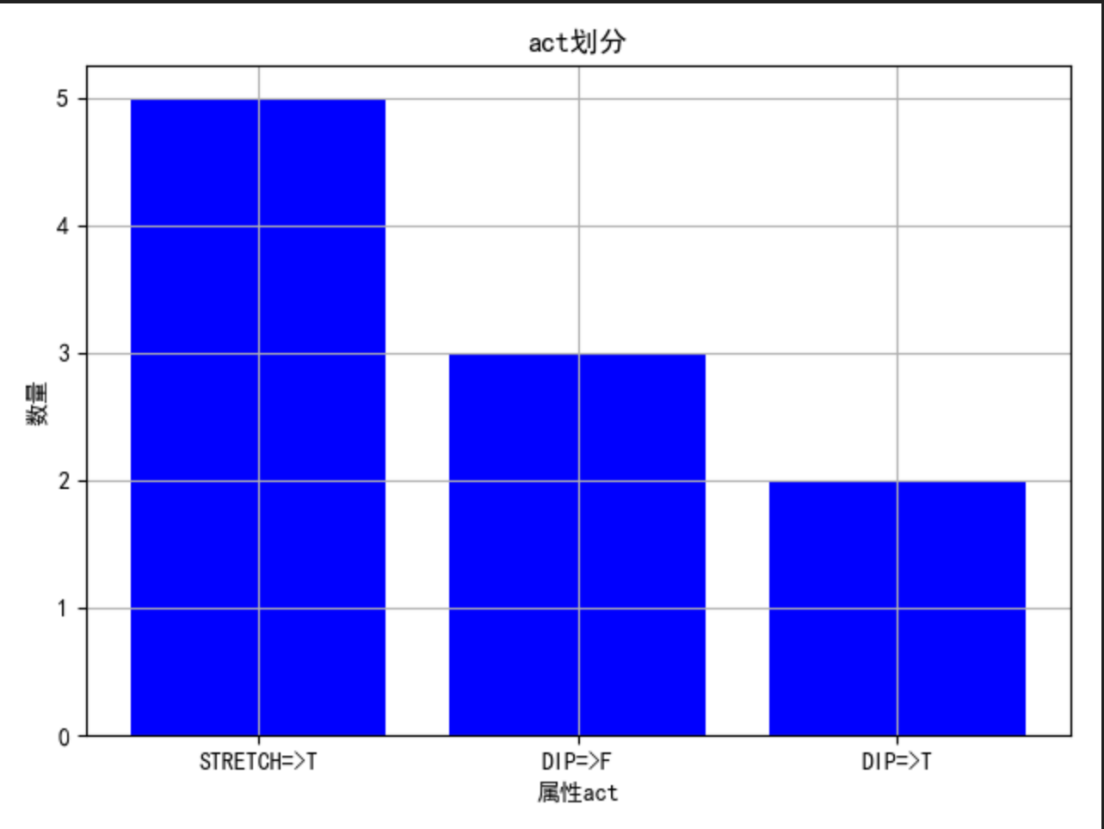
此属性出现了四个柱，说明不能完全划分

size:



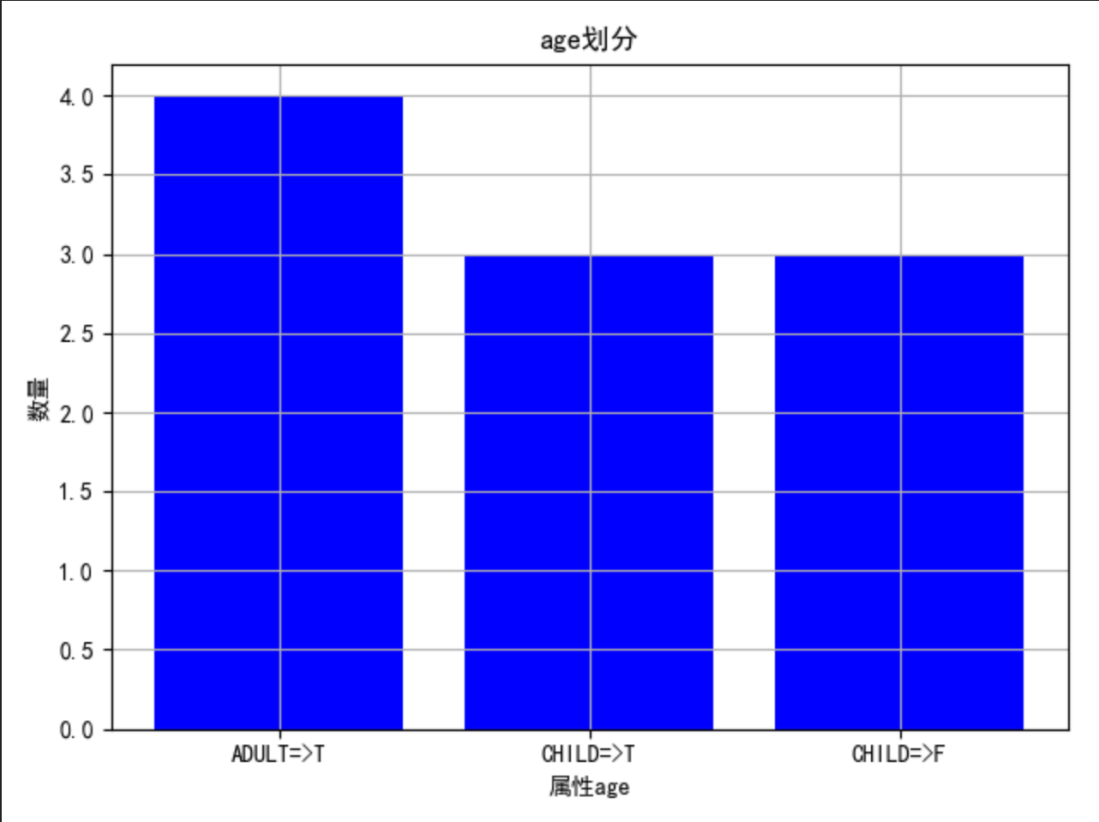
此属性同上，不能完全划分

act:



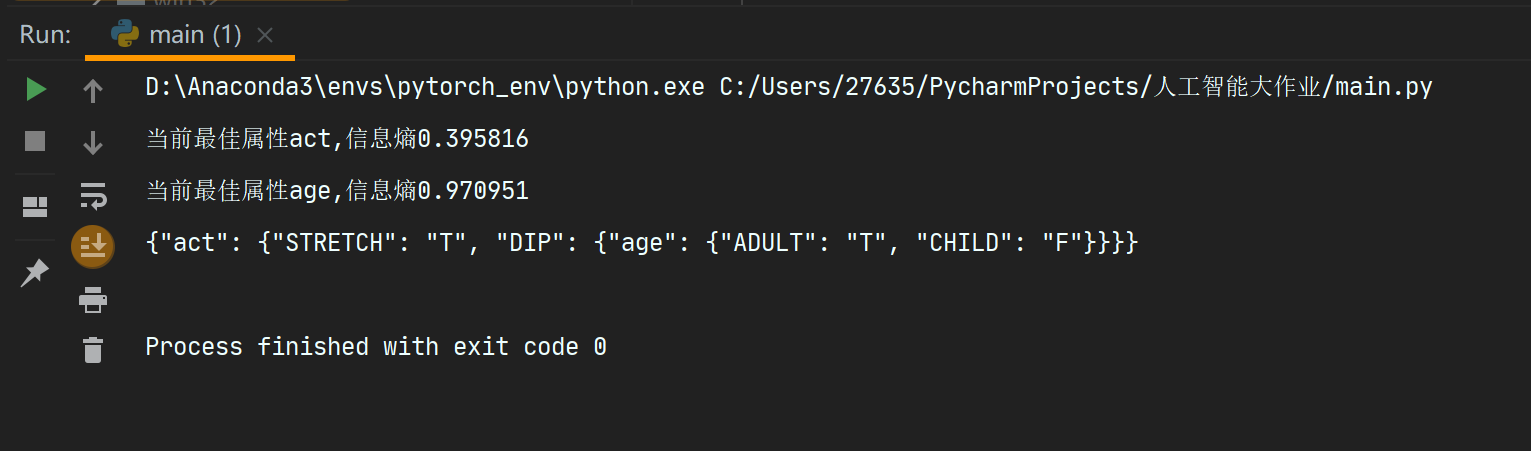
从图中可以看出，STRETCH全部为T，说明此属性可以连接叶子节点

age:

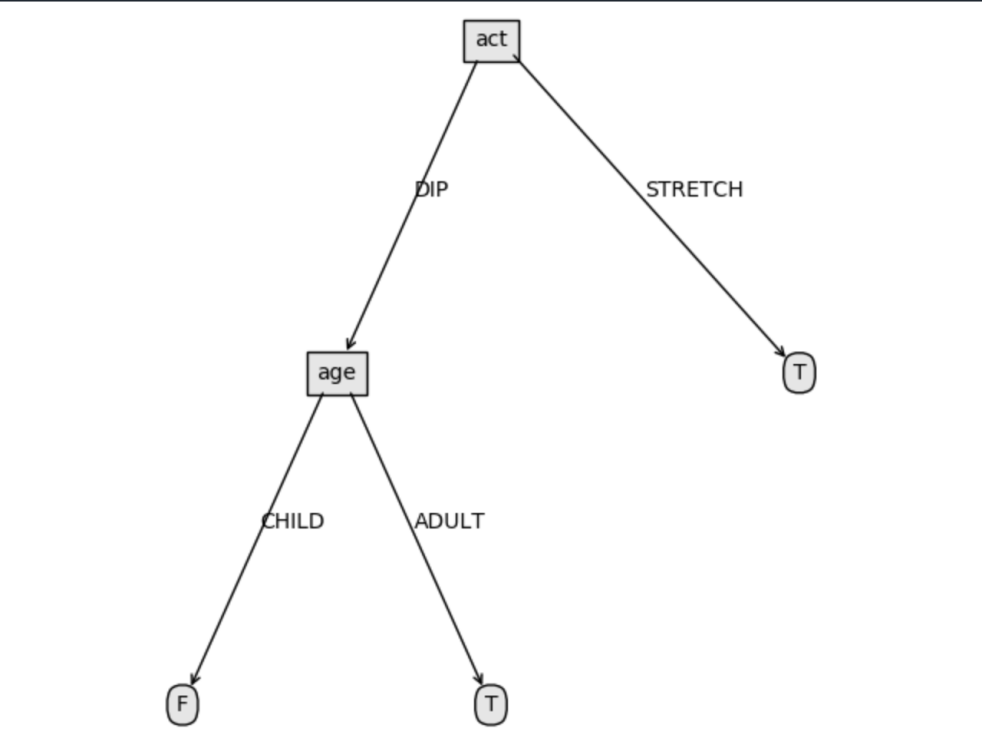


从图中可以看出，ADULT全部为T，说明此属性可以连接叶子节点

## 3.程序运行控制台输出结果：



## 4.决策树可视化结果：



通过输出的结果可以知道，act属性对于结果的划分最彻底，其次是age属性

根据可视化图可知

act = STRETCH => T

act = DIP and age = CHILD => F

act = DIP and age = ADULT => T

# 六、心得体会

1. 首先通过本次实验我学习了一下决策树的可视化的方式
2. 通过这次试验，我将以前学习过的决策树相关的知识复习了一遍
3. 这次实验的数据量有限，而且都是离散化数据，所以我并没有做整体的数据分布的分析
4. 因为决策树算法与分析线性相关性分析联系不大，所以我并没有衡量不同属性间的相关性
5. 通过这次实验，我对机器学习的相关项目的流程有了更加清晰的认识，这也对我日后做病理图像相关的机器学习打下了基础

# 七、参考资料

[1] 《DEEPLEARNING》，[美]lan Goodfellow [加]Yoshua Bengio [加]Aaron Courville 人民邮电出版社

[2] 《机器学习》，周志华，清华大学出版社

[3] 《Python数据可视化之matplotlib》，刘大成，电子工业出版社

[4] 《统计学习方法》，李航，清华大学出版社

[5] 《机器学习实战》，Peter Harrington，人民邮电出版社

[6] 《阿里云天池大赛赛题解析机器学习篇》，天池平台，电子工业出版社

[7] 《Python 袖珍指南》,Mark Lutz, 中国电力出版社

# 八、附录

## 代码

Github：<https://github.com/AloveDL/AIwork>

### main.py

1. **from** math **import** log
2. **import** operator  # 此行加在文件顶部
3. **import** json
4. **from** utils.treeploter **import** \*
5. # 计算信息熵
6. **def** calcShannonEnt(dataSet):
7. numEntries = len(dataSet)  # 样本数
8. labelCounts = {}
9. **for** featVec **in** dataSet:  # 遍历每个样本
10. currentLabel = featVec[-1]  # 当前样本的类别
11. **if** currentLabel **not** **in** labelCounts.keys():  # 生成类别字典
12. labelCounts[currentLabel] = 0
13. labelCounts[currentLabel] += 1
14. shannonEnt = 0.0
15. **for** key **in** labelCounts.keys():  # 计算信息熵
16. prob = float(labelCounts[key]) / numEntries
17. shannonEnt = shannonEnt - prob \* log(prob, 2)
18. # print("属性%s 信息熵为%f"%(key,shannonEnt))
19. **return** shannonEnt


23. # 划分数据集，axis:按第几个属性划分，value:要返回的子集对应的属性值
24. **def** splitDataSet(dataSet, axis, value):
25. retDataSet = []
26. featVec = []
27. **for** featVec **in** dataSet:
28. **if** featVec[axis] == value:
29. reducedFeatVec = featVec[:axis]
30. reducedFeatVec.extend(featVec[axis + 1:])
31. retDataSet.append(reducedFeatVec)
32. **return** retDataSet

35. # 选择最好的数据集划分方式
36. **def** chooseBestFeatureToSplit(dataSet):
37. numFeatures = len(dataSet[0]) - 1  # 属性的个数
38. baseEntropy = calcShannonEnt(dataSet)
39. bestInfoGain = 0.0
40. bestFeature = -1
41. **for** i **in** range(numFeatures):  # 对每个属性技术信息增益
42. featList = [example[i] **for** example **in** dataSet]
43. uniqueVals = set(featList)  # 该属性的取值集合
44. newEntropy = 0.0
45. **for** value **in** uniqueVals:  # 对每一种取值计算信息增益
46. subDataSet = splitDataSet(dataSet, i, value)
47. prob = len(subDataSet) / float(len(dataSet))
48. newEntropy += prob \* calcShannonEnt(subDataSet)
49. infoGain = baseEntropy - newEntropy
50. **if** (infoGain > bestInfoGain):  # 选择信息增益最大的属性
51. bestInfoGain = infoGain
52. bestFeature = i
54. **return** bestFeature,bestInfoGain




60. # 通过排序返回出现次数最多的类别
61. **def** majorityCnt(classList):
62. classCount = {}
63. **for** vote **in** classList:
64. **if** vote **not** **in** classCount.keys(): classCount[vote] = 0
65. classCount[vote] += 1
66. sortedClassCount = sorted(classCount.iteritems(),
67. key=operator.itemgetter(1), reverse=True)
68. **return** sortedClassCount[0][0]

71. # 递归构建决策树
72. **def** createTree(dataSet, labels):
73. classList = [example[-1] **for** example **in** dataSet]  # 类别向量
74. **if** classList.count(classList[0]) == len(classList):  # 如果只有一个类别，返回
75. **return** classList[0]
76. **if** len(dataSet[0]) == 1:  # 如果所有特征都被遍历完了，返回出现次数最多的类别
77. **return** majorityCnt(classList)
78. bestFeat,bestGain = chooseBestFeatureToSplit(dataSet)  # 最优划分属性的索引
79. bestFeatLabel = labels[bestFeat]  # 最优划分属性的标签
80. **print**("当前最佳属性%s,信息熵%f" % (bestFeatLabel,bestGain))
81. myTree = {bestFeatLabel: {}}
82. **del** (labels[bestFeat])  # 已经选择的特征不再参与分类
83. featValues = [example[bestFeat] **for** example **in** dataSet]
84. uniqueValue = set(featValues)  # 该属性所有可能取值，也就是节点的分支
85. **for** value **in** uniqueValue:  # 对每个分支，递归构建树
86. subLabels = labels[:]
87. myTree[bestFeatLabel][value] = createTree(
88. splitDataSet(dataSet, bestFeat, value), subLabels)
89. **return** myTree
91. **if** \_\_name\_\_ == '\_\_main\_\_':

94. fr = open(r'data/text.txt')
95. listWm = [inst.strip().split(',')[1:] **for** inst **in** fr.readlines()[1:]]
96. # print(listWm)
97. fr.close()
98. fr = open(r'data/text.txt')
99. labels = fr.readlines()[0].split(',')[1:]
100. Trees = createTree(listWm, labels)
101. fr.close()
102. **print**(json.dumps(Trees, ensure\_ascii=False))
104. createPlot(Trees)

### treeploter.py

1. # -\*- coding: cp936 -\*-
2. **import** matplotlib.pyplot as plt
4. # 设置决策节点和叶节点的边框形状、边距和透明度，以及箭头的形状
5. decisionNode = dict(boxstyle="square,pad=0.5", fc="0.9")
6. leafNode = dict(boxstyle="round4, pad=0.5", fc="0.9")
7. arrow\_args = dict(arrowstyle="<-", connectionstyle="arc3", shrinkA=0,
8. shrinkB=16)

11. **def** plotNode(nodeTxt, centerPt, parentPt, nodeType):
12. createPlot.ax1.annotate(str(nodeTxt), xy=parentPt,
13. xycoords='axes fraction',
14. xytext=centerPt, textcoords='axes fraction',
15. va="top", ha="center", bbox=nodeType,
16. arrowprops=arrow\_args)

19. **def** getNumLeafs(myTree):
20. numLeafs = 0
21. firstStr = list(myTree.keys())[0]
22. secondDict = myTree[firstStr]
23. **for** key **in** secondDict.keys():
24. **if** type(secondDict[key]).\_\_name\_\_ == 'dict':
25. numLeafs += getNumLeafs(secondDict[key])
26. **else**:
27. numLeafs += 1
28. **return** numLeafs

31. **def** getTreeDepth(myTree):
32. maxDepth = 0
33. firstStr = list(myTree.keys())[0]
34. secondDict = myTree[firstStr]
35. **for** key **in** secondDict.keys():
36. **if** type(secondDict[key]).\_\_name\_\_ == 'dict':
37. thisDepth = 1 + getTreeDepth(secondDict[key])
38. **else**:
39. thisDepth = 1
40. **if** thisDepth > maxDepth: maxDepth = thisDepth
41. **return** maxDepth

44. **def** plotMidText(cntrPt, parentPt, txtString):
45. xMid = (parentPt[0] - cntrPt[0]) / 2.0 + cntrPt[0]
46. yMid = (parentPt[1] - cntrPt[1]) / 2.0 + cntrPt[1]
47. createPlot.ax1.text(xMid, yMid, str(txtString))

50. **def** plotTree(myTree, parentPt, nodeTxt):
51. numLeafs = getNumLeafs(myTree)
52. depth = getTreeDepth(myTree)
53. firstStr = list(myTree.keys())[0]
54. cntrPt = (plotTree.xOff + (1.0 + float(numLeafs)) / 2.0 / plotTree.totalW,
55. plotTree.yOff)
56. plotMidText(cntrPt, parentPt, nodeTxt)
57. plotNode(firstStr, cntrPt, parentPt, decisionNode)
58. secondDict = myTree[firstStr]
59. plotTree.yOff = plotTree.yOff - 1.0 / plotTree.totalD
60. **for** key **in** secondDict.keys():
61. **if** type(secondDict[key]).\_\_name\_\_ == 'dict':
62. plotTree(secondDict[key], cntrPt, str(key))
63. **else**:
64. plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalW
65. plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff),
66. cntrPt, leafNode)
67. plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
68. plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD

71. **def** createPlot(inTree):
72. fig = plt.figure(1, facecolor='white')
73. fig.clf()
74. axprops = dict(xticks=[], yticks=[])
75. createPlot.ax1 = plt.subplot(111, frameon=False, \*\*axprops)
76. plotTree.totalW = float(getNumLeafs(inTree))
77. plotTree.totalD = float(getTreeDepth(inTree))
78. plotTree.xOff = -0.5 / plotTree.totalW
79. plotTree.yOff = 1.0
80. plotTree(inTree, (0.5, 1.0), '')
81. plt.show()

### split.py

1. **import** matplotlib as mpl
2. **import** matplotlib.pyplot as plt
4. mpl.rcParams["font.sans-serif"] = ["SimHei"]
5. mpl.rcParams["axes.unicode\_minus"] = False
7. **def** split():
8. fr = open(r'data/text.txt')
9. listWm = [inst.strip().split(',')[1:] **for** inst **in** fr.readlines()]
10. features = listWm[0][:-1]
11. **for** index,feature **in** enumerate(features):
13. plt.title(feature+"划分")
14. mydict = {}
15. **for** s **in** listWm[1:]:
16. **print**(s[index])
17. **if** s[index] **not** **in** mydict.keys():
18. mydict[s[index]] = 1
19. **else**:
20. mydict[s[index]] += 1
21. x = range(len(mydict.keys()))
22. y = list(map(int,mydict.values()))
23. **print**(mydict.keys())
24. plt.bar(x,y,align="center",color="b",tick\_label=list(mydict.keys()))
25. plt.xlabel("属性"+feature)
26. plt.ylabel("数量")
27. plt.grid(True)
28. plt.show()
30. **if** \_\_name\_\_ == '\_\_main\_\_':
31. split()

### split2.py

1. **import** matplotlib as mpl
2. **import** matplotlib.pyplot as plt
4. mpl.rcParams["font.sans-serif"] = ["SimHei"]
5. mpl.rcParams["axes.unicode\_minus"] = False
7. **def** split2(inputFeature):
8. fr = open(r'data/text.txt')
9. listWm = [inst.strip().split(',')[1:] **for** inst **in** fr.readlines()]
10. features = listWm[0][:-1]
11. **for** index,feature **in** enumerate(features):
12. **if** feature == inputFeature:
13. plt.title(feature+"划分")
14. mydict = {}
15. **for** s **in** listWm[1:]:
17. **if** s[index]+"=>"+s[-1] **not** **in** mydict.keys():
18. mydict[s[index]+"=>"+s[-1]] = 1
19. **else**:
20. mydict[s[index]+"=>"+s[-1]] += 1
21. x = range(len(mydict.keys()))
22. y = list(map(int,mydict.values()))
23. **print**(mydict.keys())
24. plt.bar(x,y,align="center",color="b",tick\_label=list(mydict.keys()))
25. plt.xlabel("属性"+feature)
26. plt.ylabel("数量")
27. plt.grid(True)
28. plt.show()

31. **if** \_\_name\_\_ == '\_\_main\_\_':
32. split2("size")