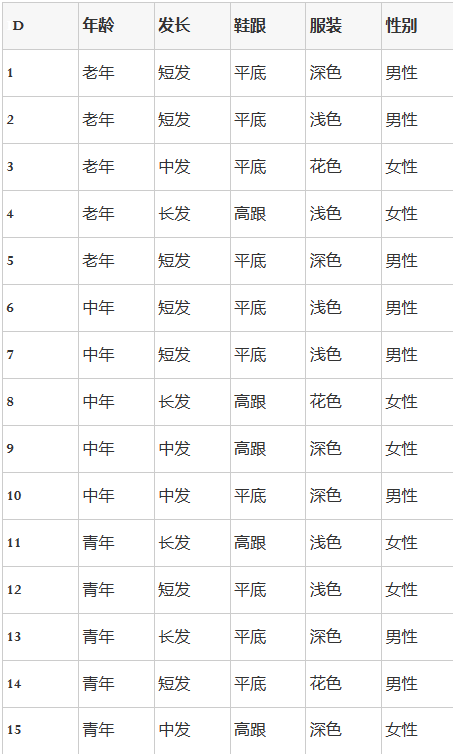
决策树

题目：

现有数据集15人，初始数据集D={1,2,…,15}



请按照决策树算法进行分类。默认ID3.

1）请写出计算过程，最后画出决策树。

2）请编程实现。注意，代码实现语言不限。

为了使用ID3算法进行分类，我们需要执行以下步骤：

1.计算数据集的香农熵。

2.对每个特征计算信息增益。

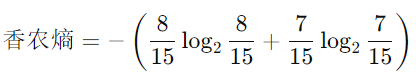
3.选择信息增益最大的特征作为节点，并划分数据集。

4.重复步骤1-3，直到所有数据均被分类或没有特征可用。

ID3算法的计算过程：计算信息增益，并选择最佳特征进行划分。从数据集中选择信息增益最高的特征，递归构建决策树。

1. 计算初始香农熵

初始数据集包含 15 条记录，我们首先计算整个数据集的香农熵：



计算结果为：香农熵 = 0.9968

2. 计算每个特征的信息增益

我们将计算每个特征的信息增益，并选择信息增益最大的特征进行划分。共有5个特征：年龄、发长、鞋跟、服装。对于每个值，计算子集的香农熵并求和：

年龄

中年香农熵=0.97095

中年香农熵=0.97095

青年香农熵=0.97095

信息增益=初始香农熵−年龄的香农熵=0.9968−0.97095=0.02585

发长

短发香农熵=0.9183

中发香农熵=0.9183

长发香农熵 = 1

信息增益=0.9968−0.9521=0.0447

鞋跟

平底香农熵=0.7642

高跟香农熵=0.65002

信息增益=0.9968−0.7177=0.2791

服装

深色香农熵=0.9183

浅色香农熵 = 0.97095

花色香农熵 = 1

信息增益=0.9968−0.9605=0.0363

进行信息熵值计算：

import math  
def calcShannonEnt(dataSet):  
 numEntries = len(dataSet)  
 labelCounts = {}  
 for featVec in dataSet:  
 currentLabel = featVec[-1]  
 if currentLabel not in labelCounts.keys():  
 labelCounts[currentLabel] = 0  
 labelCounts[currentLabel] += 1  
 shannonEnt = 0.0  
 for key in labelCounts:  
 prob = float(labelCounts[key]) / numEntries  
 shannonEnt -= prob \* math.log(prob, 2)  
 return shannonEnt  
  
def splitDataSet(dataSet, axis, value):  
 retDataSet = []  
 for featVec in dataSet:  
 if featVec[axis] == value:  
 reducedFeatVec = featVec[:axis]  
 reducedFeatVec.extend(featVec[axis+1:])  
 retDataSet.append(reducedFeatVec)  
 return retDataSet  
  
def chooseBestFeatureToSplit(dataSet):  
 numFeatures = len(dataSet[0]) - 1  
 baseEntropy = calcShannonEnt(dataSet)  
 bestInfoGain = 0.0  
 bestFeature = -1  
 for i in range(numFeatures):  
 featList = [example[i] for example in dataSet]  
 uniqueVals = set(featList)  
 newEntropy = 0.0  
 for value in uniqueVals:  
 subDataSet = splitDataSet(dataSet, i, value)  
 prob = len(subDataSet) / float(len(dataSet))  
 newEntropy += prob \* calcShannonEnt(subDataSet)  
 infoGain = baseEntropy - newEntropy  
 if infoGain > bestInfoGain:  
 bestInfoGain = infoGain  
 bestFeature = i  
 return bestFeature  
dataSet = [  
 ['老年', '短发', '平底', '深色', '男性'],  
 ['老年', '短发', '平底', '浅色', '男性'],  
 ['老年', '中发', '平底', '花色', '女性'],  
 ['老年', '长发', '高跟', '浅色', '女性'],  
 ['老年', '短发', '平底', '深色', '男性'],  
 ['中年', '短发', '平底', '浅色', '男性'],  
 ['中年', '短发', '平底', '浅色', '男性'],  
 ['中年', '长发', '高跟', '花色', '女性'],  
 ['中年', '中发', '高跟', '深色', '女性'],  
 ['中年', '中发', '平底', '深色', '男性'],  
 ['青年', '长发', '高跟', '浅色', '女性'],  
 ['青年', '短发', '平底', '浅色', '女性'],  
 ['青年', '长发', '平底', '深色', '男性'],  
 ['青年', '短发', '平底', '花色', '男性'],  
 ['青年', '中发', '高跟', '深色', '女性']  
]  
labels = ['年龄', '发长', '鞋跟', '服装']  
bestFeature = chooseBestFeatureToSplit(dataSet)  
labels[bestFeature]

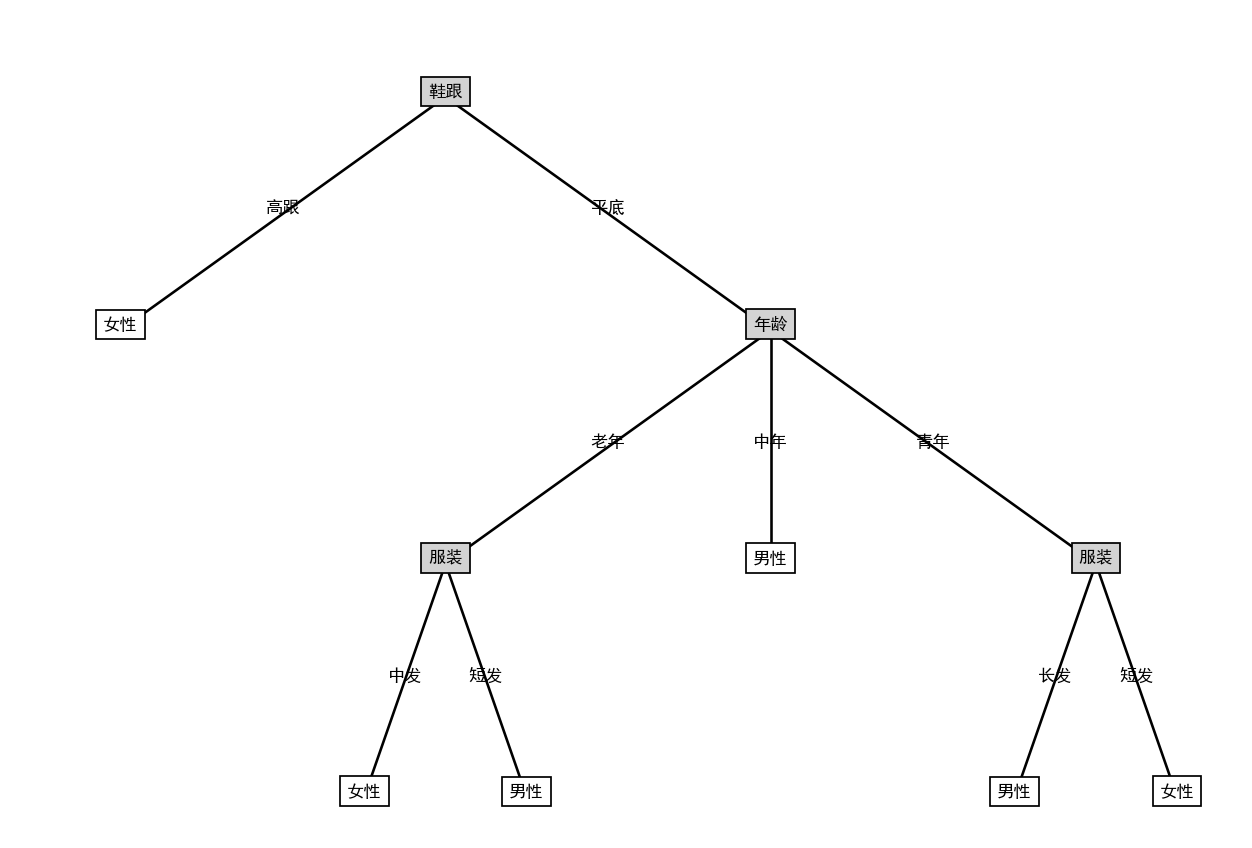
3. 选择信息增益最大的特征

根据计算结果，信息增益最大的特征是鞋跟。

4. 递归构建子树

对平底和高跟分别构建子树，重复上述计算过程，直到所有特征都用完或所有子集中的记录属于同一类。

1. 绘制最终决策树



绘制最终决策树的代码如下：

# 树节点  
class DecisionNode:  
 def \_\_init\_\_(self, attribute=None, value=None, results=None, branches=None):  
 self.attribute = attribute  
 self.value = value  
 self.results = results  
 self.branches = branches  
  
  
def majority\_count(class\_list):  
 class\_count = Counter(class\_list)  
 return class\_count.most\_common(1)[0][0]  
  
  
# 生成决策树  
def TreeGenerate(data, attributes):  
 class\_list = [row[-1] for row in data]  
  
 # 如果只有一个类，返回这个类  
 if class\_list.count(class\_list[0]) == len(class\_list):  
 return DecisionNode(results=class\_list[0])  
  
 # 如果没有更多属性进行划分，返回多数类  
 if len(attributes) == 0:  
 return DecisionNode(results=majority\_count(class\_list))  
  
 # 找到最佳划分属性  
 best\_gain = 0.0  
 best\_attribute = None  
  
 for attribute\_index in range(len(attributes)):  
 gain = information\_gain(data, attribute\_index)  
 if gain > best\_gain:  
 best\_gain = gain  
 best\_attribute = attribute\_index  
  
 # 如果信息增益为0，返回多数类  
 if best\_gain == 0:  
 return DecisionNode(results=majority\_count(class\_list))  
  
 best\_attribute\_name = attributes[best\_attribute]  
 tree = DecisionNode(attribute=best\_attribute\_name)  
 tree.branches = {}  
  
 # 删除已经选择的属性  
 remaining\_attributes = attributes[:best\_attribute] + attributes[best\_attribute + 1:]  
  
 # 创建分支  
 attribute\_values = set([row[best\_attribute] for row in data])  
 for value in attribute\_values:  
 subset = split\_data(data, best\_attribute, value)  
 if subset:  
 branch = TreeGenerate(subset, remaining\_attributes)  
 tree.branches[value] = branch  
  
 return tree  
  
# 定义数据集  
D = [  
 [1, '老年', '短发', '平底', '深色', '男性'],  
 [2, '老年', '短发', '平底', '浅色', '男性'],  
 [3, '老年', '中发', '平底', '花色', '女性'],  
 [4, '老年', '长发', '高跟', '浅色', '女性'],  
 [5, '老年', '短发', '平底', '深色', '男性'],  
 [6, '中年', '短发', '平底', '浅色', '男性'],  
 [7, '中年', '短发', '平底', '浅色', '男性'],  
 [8, '中年', '长发', '高跟', '花色', '女性'],  
 [9, '中年', '中发', '高跟', '深色', '女性'],  
 [10, '中年', '中发', '平底', '深色', '男性'],  
 [11, '青年', '长发', '高跟', '浅色', '女性'],  
 [12, '青年', '短发', '平底', '浅色', '女性'],  
 [13, '青年', '长发', '平底', '深色', '男性'],  
 [14, '青年', '短发', '平底', '花色', '男性'],  
 [15, '青年', '中发', '高跟', '深色', '女性']  
]  
  
A = ['年龄', '发长', '鞋跟', '服装']  
  
# 去掉ID列  
D = [row[1:] for row in D]  
  
# 构建决策树  
tree = TreeGenerate(D, A)  
  
# 下面用plt绘图展示决策树  
plt.rcParams['font.sans-serif'] = ['SimHei'] # 以黑体显示中文  
plt.rcParams['axes.unicode\_minus'] = False # 解决保存图像符号“-”显示为放块的问题  
  
  
def plot\_tree(node, depth=0, pos=(0.5, 1), parent\_pos=None, text=""):  
 if node.results:  
 plt.text(pos[0], pos[1], f"{node.results}", horizontalalignment='center',  
 bbox=dict(facecolor='white', edgecolor='black'))  
 else:  
 plt.text(pos[0], pos[1], f"{node.attribute}", horizontalalignment='center',  
 bbox=dict(facecolor='lightgray', edgecolor='black'))  
  
 if parent\_pos:  
 plt.plot([parent\_pos[0], pos[0]], [parent\_pos[1], pos[1]], 'k-')  
 x\_offset = (parent\_pos[0] + pos[0]) / 2  
 y\_offset = (parent\_pos[1] + pos[1]) / 2  
 plt.text(x\_offset, y\_offset, text, horizontalalignment='center')  
  
 if node.branches:  
 branch\_count = len(node.branches)  
 for i, (value, branch) in enumerate(node.branches.items()):  
 x\_new = pos[0] + (i - (branch\_count - 1) / 2) \* 1.0 / (2 \*\* depth)  
 y\_new = pos[1] - 0.1  
 plot\_tree(branch, depth + 1, (x\_new, y\_new), pos, value)  
  
  
# 设置绘图尺寸和初始位置  
plt.figure(figsize=(12, 8))  
plot\_tree(tree)  
plt.axis('off')  
plt.show()