**一、决策树原理**

决策树是用样本的属性作为结点，用属性的取值作为分支的树结构。   
决策树的根结点是所有样本中信息量最大的属性。树的中间结点是该结点为根的子树所包含的样本子集中信息量最大的属性。决策树的叶结点是样本的类别值。决策树是一种知识表示形式，它是对所有样本数据的高度概括决策树能准确地识别所有样本的类别，也能有效地识别新样本的类别。

决策树**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**ID3的基本思想：

首先找出最有判别力的属性，把样例分成多个子集，每个子集又选择最有判别力的属性进行划分，一直进行到所有子集仅包含同一类型的数据为止。最后得到一棵决策树。

J.R.Quinlan的工作主要是引进了信息论中的信息增益，他将其称为信息增益（information gain），作为属性判别能力的度量，设计了构造决策树的递归算法。

**举例子比较容易理解：**

**对于气候分类问题，属性为:**

**天气(A1) 取值为： 晴，多云，雨**

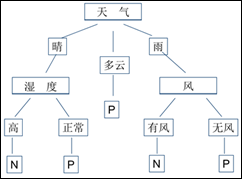
**气温(A2)  取值为： 冷 ，适中，热**

**湿度(A3)  取值为： 高 ，正常**

**风 (A4)  取值为： 有风， 无风**

**每个样例属于不同的类别，此例仅有两个类别，分别为P，N。P类和N类的样例分别称为正例和反例。将一些已知的正例和反例放在一起便得到训练集。**

**由ID3算法得出一棵正确分类训练集中每个样例的决策树，见下图。**

[](http://img.blog.csdn.net/20140714135742587)

决策树叶子为类别名，即P 或者N。其它结点由样例的属性组成，每个属性的不同取值对应一分枝。

若要对一样例分类，从树根开始进行[**测试**](http://lib.csdn.net/base/softwaretest)，按属性的取值分枝向下进入下层结点，对该结点进行测试，过程一直进行到叶结点，样例被判为属于该叶结点所标记的类别。

现用图来判一个具体例子，

某天早晨气候描述为:

天气：多云

气温：冷

湿度：正常

风： 无风

它属于哪类气候呢?-------------从图中可判别该样例的类别为P类。

**ID3就是要从表的训练集构造图这样的决策树。实际上，能正确分类训练集的决策树不止一棵。Quinlan的ID3算法能得出结点最少的决策树。**

**ID3算法：**

     ⒈ 对当前例子集合，计算各属性的信息增益；

     ⒉ 选择信息增益最大的属性Ak；

     ⒊ 把在Ak处取值相同的例子归于同一子集，Ak取几个值就得几个子集；

     ⒋ 对既含正例又含反例的子集，递归调用建树算法；

     ⒌ 若子集仅含正例或反例，对应分枝标上P或N，返回调用处。

一般只要涉及到树的情况，经常会要用到递归。

对于气候分类问题进行具体计算有：

⒈ **信息熵的计算**:[image](http://img.blog.csdn.net/20140714135743601) 其中S是样例的集合， P(ui)是类别i出现概率：

[image](http://img.blog.csdn.net/20140714135746846)

|S|表示例子集S的总数，|ui|表示类别ui的例子数。对9个正例和5个反例有：

P（u1）=9/14

P（u2）=5/14

H（S）=（9/14）log（14/9）+（5/14）log（14/5）=0.94bit

⒉ **信息增益的计算**:[image](http://img.blog.csdn.net/20140714135513156)

其中A是属性，Value(A)是属性A取值的集合，v是A的某一属性值，Sv是S中A的值为v的样例集合，| Sv |为Sv中所含样例数。

以属性A1为例，根据信息增益的计算公式，属性A1的信息增益为

[image](http://img.blog.csdn.net/20140714135749576)

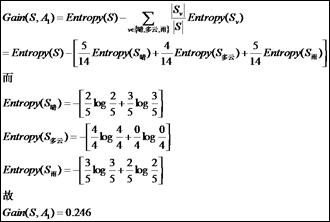
S=[9+,5-] //原样例集中共有14个样例，9个正例，5个反例

S晴=[2+,3-]//属性A1取值晴的样例共5个，2正，3反

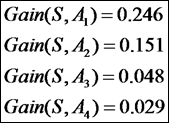
S多云=[4+,0-] //属性A1取值多云的样例共4个，4正，0反

S雨=[3+,2-] //属性A1取值晴的样例共5个，3正，2反

故



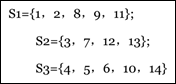
**3.结果为**

[](http://img.blog.csdn.net/20140714135750948)

属性A1的信息增益最大，所以被选为根结点。

**4.建决策树的根和叶子**

ID3算法将选择信息增益最大的属性天气作为树根，在14个例子中对天气的3个取值进行分枝，3 个分枝对应3 个子集，分别是:

[](http://img.blog.csdn.net/20140714135752711)

其中S2中的例子全属于P类，因此对应分枝标记为P，其余两个子集既含有正例又含有反例，将递归调用建树算法。

**5.递归建树**

分别对S1和S3子集递归调用ID3算法，在每个子集中对各属性求信息增益.

（1）对S1，湿度属性信息增益最大，以它为该分枝的根结点，再向下分枝。湿度取高的例子全为N类，该分枝标记N。取值正常的例子全为P类，该分枝标记P。

（2）对S3，风属性信息增益最大，则以它为该分枝根结点。再向下分枝，风取有风时全为N类，该分枝标记N。取无风时全为P类，该分枝标记P。

**二、PYTHON实现决策树算法分类**

本代码为machine learning in action 第三章例子，亲测无误。

 1、计算给定数据shangnon数据的函数：

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** calcShannonEnt(dataSet):
2. #calculate the shannon value
3. numEntries = len(dataSet)
4. labelCounts = {}
5. **for** featVec **in** dataSet:      #create the dictionary for all of the data
6. currentLabel = featVec[-1]
7. **if** currentLabel **not** **in** labelCounts.keys():
8. labelCounts[currentLabel] = 0
9. labelCounts[currentLabel] += 1
10. shannonEnt = 0.0
11. **for** key **in** labelCounts:
12. prob = float(labelCounts[key])/numEntries
13. shannonEnt -= prob\*log(prob,2) #get the log value
14. **return** shannonEnt

 2. 创建数据的函数

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** createDataSet():
2. dataSet = [[1,1,'yes'],
3. [1,1, 'yes'],
4. [1,0,'no'],
5. [0,1,'no'],
6. [0,1,'no']]
7. labels = ['no surfacing','flippers']
8. **return** dataSet, labels

3.划分数据集，按照给定的特征划分数据集

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** splitDataSet(dataSet, axis, value):
2. retDataSet = []
3. **for** featVec **in** dataSet:
4. **if** featVec[axis] == value:      #abstract the fature
5. reducedFeatVec = featVec[:axis]
6. reducedFeatVec.extend(featVec[axis+1:])
7. retDataSet.append(reducedFeatVec)
8. **return** retDataSet

4.选择最好的数据集划分方式

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** chooseBestFeatureToSplit(dataSet):
2. numFeatures = len(dataSet[0])-1
3. baseEntropy = calcShannonEnt(dataSet)
4. bestInfoGain = 0.0; bestFeature = -1
5. **for** i **in** range(numFeatures):
6. featList = [example[i] **for** example **in** dataSet]
7. uniqueVals = set(featList)
8. newEntropy = 0.0
9. **for** value **in** uniqueVals:
10. subDataSet = splitDataSet(dataSet, i , value)
11. prob = len(subDataSet)/float(len(dataSet))
12. newEntropy +=prob \* calcShannonEnt(subDataSet)
13. infoGain = baseEntropy - newEntropy
14. **if**(infoGain > bestInfoGain):
15. bestInfoGain = infoGain
16. bestFeature = i
17. **return** bestFeature

5.递归创建树

    用于找出出现次数最多的分类名称的函数

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** majorityCnt(classList):
2. classCount = {}
3. **for** vote **in** classList:
4. **if** vote **not** **in** classCount.keys(): classCount[vote] = 0
5. classCount[vote] += 1
6. sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
7. **return** sortedClassCount[0][0]

  用于创建树的函数代码

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** createTree(dataSet, labels):
2. classList = [example[-1] **for** example **in** dataSet]
3. # the type is the same, so stop classify
4. **if** classList.count(classList[0]) == len(classList):
5. **return** classList[0]
6. # traversal all the features and choose the most frequent feature
7. **if** (len(dataSet[0]) == 1):
8. **return** majorityCnt(classList)
9. bestFeat = chooseBestFeatureToSplit(dataSet)
10. bestFeatLabel = labels[bestFeat]
11. myTree = {bestFeatLabel:{}}
12. **del**(labels[bestFeat])
13. #get the list which attain the whole properties
14. featValues = [example[bestFeat] **for** example **in** dataSet]
15. uniqueVals = set(featValues)
16. **for** value **in** uniqueVals:
17. subLabels = labels[:]
18. myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)
19. **return** myTree

然后是在python 名利提示符号输入如下命令：

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. myDat, labels = trees.createDataSet()
2. myTree = trees.createTree(myDat,labels)
3. **print** myTree

结果是：

{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}

6.实用决策树进行分类的函数

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. **def** classify(inputTree, featLabels, testVec):
2. firstStr = inputTree.keys()[0]
3. secondDict = inputTree[firstStr]
4. featIndex = featLabels.index(firstStr)
5. **for** key **in** secondDict.keys():
6. **if** testVec[featIndex] == key:
7. **if** type(secondDict[key]).\_\_name\_\_ == 'dict':
8. classLabel = classify(secondDict[key], featLabels, testVec)
9. **else**: classLabel = secondDict[key]
10. **return** classLabel

在Python命令提示符，输入：

**[python]** [view plain](http://blog.csdn.net/alvine008/article/details/37760639) [copy](http://blog.csdn.net/alvine008/article/details/37760639)

[print?](http://blog.csdn.net/alvine008/article/details/37760639)

1. trees.classify(myTree,labels,[1,0])

得到结果：

'no'

Congratulation. Oh yeah. You did it.!!!