决策树学习与 ID3 算法

ID3 算法

决策树学习是一种归纳学习方法，当前国际上最有影响的示例学习方法首推的应当 ID3 算法。ID3 算法最早是由罗斯昆（J. Ross Quinlan）于 1975 年在悉尼大学提出的一种分类预测算法，算法的核心是“信息熵”。ID3 以信息熵的下降速度为选取测试属性的标准，通过计算每个属性的信息增益，认为信息增益高的是好属性，每次划分选取信息增益最高的属性为划分标准，重复这个过程，直至生成一个能完美分类训练样例的决策树。

决策树，又称为判定树，是一种类似二叉树或多叉树的树结构。树中的每个非叶节点（包括根节点）对应于训练样本集中一个非类别属性的测试，非叶节点的每个分支对应属性的一个测试结果，每个叶子节点则代表一个类或类分布。从根节点到叶子节点的一条路径形成一条分类规则。决策树可以很方便地转化为分类规则，是一种非常直观的分类模式表示形式。

ID3 算法是所有可能决策树空间中一种自顶向下、贪婪的搜索方法，以信息熵的下降速度为选取测试属性的标准，即在每个节点选取还尚未被用来划分的具有最高信息增益的属性作为划分标准，然后继续这个过程，直到生成的决策树能完美分类训练样例。

在决策树构造中，如何选取一个条件属性作为形成决策树的节点是建树的核心。一般情况下，选取的属性能最大程度反映训练样本集的分类特征。ID3 算法作为决策构造中的经典算法，引入了信息论的方法，应用信息论中的熵的概念，采用信息增益作为选择属性的标准来对训练样本集进行划分，选取信息增益最大的属性作为当前节点。计算信息增益还要涉及三个概念：信息熵、信息增益和信息条件熵。

信息熵

信息熵也称为香农熵，是随机变量的期望。度量信息的不确定程度。信息的熵越大，信息就越不容易搞清楚。处理信息就是为了把信息搞清楚，就是熵减少的过程。

信息熵是随机变量的期望。度量信息的不确定程度。信息的熵越大，信息就越不容易搞清楚。处理信息就是.为了把信息搞清楚，就是熵减少的过程。

对数据 D 中的元组分类所需要的期望信息如下:

$$\text{Entropy(D)} = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

其中p_i是 D 中任意元组属于c_i的非零概率

信息增益则用于度量属性 A 降低样本集合 X 熵的贡献大小。信息增益越大，越适于对 X 分类。

$$\text{Gain}(\text{D}_A) = \text{Entropy(D)} - \text{Entropy}_A\text{(D)}$$

算法思想

决策树是对数据进行分类，以此达到预测的目的。该决策树方法先根据训练集数据形成决策树，如果该树不能对所有对象给出正确的分类，那么选择一些例外加入到训练集数据中，重复该过程一直到形成正确的决策集。决策树代表着决策集的树形结构。决策树由决策结点、分支和叶子组成。决策树中最上面的结点为根结点，每个分支是一个新的决策结点，或者是树的叶子。每个决策结点代表一个问题或决策，通常对应于待分类对象的属性。每一个叶子结点代表一种可能的分类结果。沿决策树从上到下遍历的过程中，在每个结点都会遇到一个测试，对每个结点上问题的不同的测试输出导致不同的分支，最后会到达一个叶子结点，这个过程就是利用决策树进行分类的过程，利用若干个变量来判断所属的类别。

1．自顶向下的贪婪搜索遍历可能的决策树空间构造决策树

2．从“哪一个属性将在树的根节点被测试”开始；

3．使用统计测试来确定每一个实例属性单独分类训练样例的能力，分类能力最好的属性作为树的根结点测试(如何定义或者评判一个属性是分类能力最好的呢？这便是下文将要介绍的信息增益，or 信息增益率)。

4．然后为根结点属性的每个可能值产生一个分支，并把训练样例排列到适当的分支（也就是说，样例的该属性值对应的分支）之下。

5．重复这个过程，用每个分支结点关联的训练样例来选取在该点被测试的最佳属性。

示例代码

以下为我们使用的数据集

outlook	*temperature*	*humidity*	*windy*	*class*
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

我们使用函数 **calcShannonEnt** 来计算各个属性的信息熵,同时将计算的

结果记录下来，以备下一步优选。

```
#  计算给定数据集的熵
def calcShannonEnt(dataSet):
    """
    dataset : 待处理的数据集
    """
    numEntries = len(dataSet)  #得到数据集总数
    labelCounts = {}

    for featVec in dataSet:
        currentLabel = featVec[-1]  #-1表示最后一列

        #这边处理可能不好理解，初始不在字典中先设为0，再加1，以后再字典中就直接加1
        #我们可能处理是if no in dictionary，设为1，else +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*log(prob,2)
    return shannonEnt
```

随后寻找一个最优的属性进行划分，找到信息增益最大的那个属性，作为当前根节点的属性，然后对属性的各个取值，将数据划分为相互不交叉的子集，然后递归的进行信息增益的计算和属性的划分。

```
# 选择最好的数据集划分方式
def chooseBestFeatureToSplit(dataSet):

    numFeatures = len(dataSet[0]) - 1   #最后一个是类别，特征数是每一行数目减去1
    baseEntropy = calcShannonEnt(dataSet)#计算数据集原始信息熵
    bestInfoGain = 0.0; bestFeature = -1

    for i in range(numFeatures):

        featList = [example[i] for example in dataSet]#example[i]代表第i列的特征值，在for循
        uniqueVals = set(featList) #set是一个无序不重复集跟其他语言一样，这样得到i这一列不同
        newEntropy = 0.0

        for value in uniqueVals:#根据不同的特征值分类计算所占百分比
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)

        infoGain = baseEntropy - newEntropy #这是信息增益，是两个信息差

        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i

    return bestFeature
```

最后不断的递归划分，最后生成一棵决策树。我们采用字典的形式表示这个树形结构的。

```
the tree is next :
{'outlook': {'overcast': 'P', 'sunny': {'humidity': {'high': 'N', 'normal': 'P'}}, 'rain':
{'windy': {'false': 'P', 'true': 'N'}}}}
```

我们生成树的结果如下

```
84
85  def createTree(dataSet,Labels):
86      classList = [example[-1] for example in dataSet]#得到分类这边是yes no
87
88      if classList.count(classList[0]) == len(classList): #这边说明都是同一个属性了，不好再分了
89          return classList[0]
90
91      if len(dataSet[0]) == 1:
92          return majorityCnt(classList)
93
94      bestFeat = chooseBestFeatureToSplit(dataSet)
95      bestFeatLabel = labels[bestFeat]
96
97      myTree = {bestFeatLabel:{}}
98
99      del(labels[bestFeat])
100     featValues = [example[bestFeat] for example in dataSet]
101     uniqueVals = set(featValues)
102
103     for value in uniqueVals:#分类 递归
104         subLabels = labels[:]  #:表示全部
105         myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value),subLabels
106
107     return myTree
108
```

最后我们生成的决策树如下所示，我们发现我们的数据已经被很好的类。