机器学习实验 5. 决策树

决策树原理概述

决策树(decision tree)是一种有监督的分类和预测学习算法，以树状图为基础，其输出结果为一系列简单实用的规则，故得名决策树。为了表述方便，我们以分类问题为例。决策树模型基于特征对实例进行分类，它是一种树状结构，由节点和有向边组成。内部节点表示一个特征或者属性;叶子节点表示一个分类。使用决策树进行分类时，将实例分配到叶节点的类中，该叶节点所属的类就是该节点的分类。

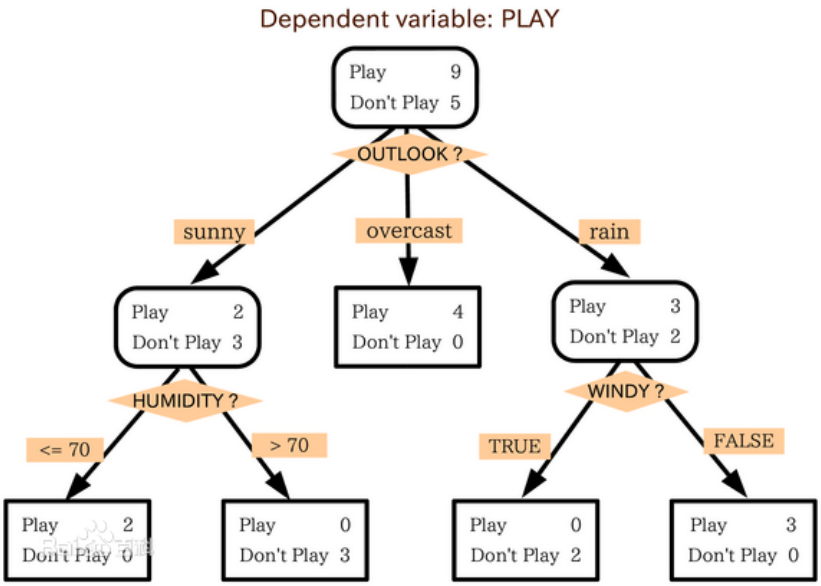

图 1 决策树模型示意图

决策树的一个重要任务是为了数据中所蕴含的知识信息，因此决策树可以使用不熟悉的数据集合，并从中提取出一系列规则，在这些机器根据数据集创建规则时，就是机器学习的过程。

决策树的构建

- 构建决策树的 3 个步骤:

(1) 特征选择;

(2) 决策树生成;

(3) 决策树剪枝。

假设给定训练集 $D=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N),\}$，其中 $x_i=(x_i^{(1)},x_i^{(2)},...,x_i^{(n)})$ 为输入实例，n 为特征个数；$y_i\in\{1,2,...,K\}$ 为类标记，$i=1,2,...,N$，N 为样本容量。构建决策树的目标是,根据给定的训练数据集学习一个决策树模型。

构建决策树时通常是将正则化的极大似然函数作为损失函数,其学习目标是损失函数 ·为目标函数的最小化。构建决策树的算法通常是递归地选择最优特征,并根据该特征对训练数据进行分割。

- 决策树的一般流程

(1) 收集数据：可以使用任何方法。

(2) 准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化。

(3) 分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期。

(4) 训练算法：构造树的数据结构。

(5) 测试算法：使用经验树计算错误率。

(6) 使用算法：此步骤可以适用于任何监督学习算法，而使用决策树可以更好地理解数据 的内在含义。

决策树分类算法在模拟数据中的一般实现

- 模拟数据集

表 1 海洋生物数据

	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

- 信息增益

在划分数据集之前之后信息发生的变化称为信息增益,知道如何计算信息增益，我们就可以计算每个特征值划分数据集获得的信息增益，获得信息增益最高的特征就是最好的选择。在可以评测哪种数据划分方式是最好的数据划分之前，我们必须学习如何计算信息增益。集合信息的度量方式称为香农熵或者简称为熵。

熵定义为信息的期望值，在明晰这个概念之前，我们必须知道信息的定义。如果待分类的事务可能划分在多个分类之中，则符号x_i的信息定义为 ：

$$l(x_i) = -\log_2 p(x_i)$$

其中 p(x_i)是选择该分类的概率。

为了计算熵，我们需要计算所有类别所有可能值包含的信息期望值，通过下面的公式得到:

$$H = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

其中 n 是分类的数目。

1)：计算给定数据集的香农熵

```
from math import log

def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet: #the the number of unique elements and their occurance
        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 * log(prob,2) #Log base 2
    return shannonEnt
```

首先，计算数据集中实例的总数。我们也可以在需要时再计算这个值，但是由于代码中多次用到这个值，为了提高代码效率，我们显式地声明一个变量保存实例总数。然后，创建一个数据字典，它的键值是最后一列的数值 。如果当前键值不存在，则扩展字典并将当前键值加入字典。每个键值都记录了当前类别出现的次数。最后，使用所有类标签的发生频率计算类别出现的概率。我们将用这个概率计算香农熵 ，统计所有类标签发生的次数。下面我们看看如何使用熵划分数据集。

2)：输入模拟的数据集

```
def createDataSet():
    dataSet = [[1,1,'yes'],
               [1,1,'yes'],
               [1,0,'no'],
               [0,1,'no'],
               [0,1,'no']]

    labels = ['no surfacing','flippers']
    return dataSet,labels
```

```
myDat, labels=createDataSet()
print(myDat)
print(calcShannonEnt(myDat))
```

运行我们会得到以下输出：

```
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
0.9709505944546686
```

代码是先输出训练数据集，然后第二行输出的是计算的经验熵。得到熵之后，我们就可以按照获取最大信息增益的方法划分数据集。

- 划分数据集

分类算法除了需要测量信息熵，还需要划分数据集，度量划分数据集的熵，以便判断当前是否正确地划分了数据集。我们将对每个特征划分数据集的结果计算一次信息熵，然后判断按照哪个特征划分数据集是最好的划分方式。

1)：按照给定特征划分数据集

```
def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]     #chop out axis used for splitting
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet
```

上述代码使用了三个输入参数：待划分的数据集、划分数据集的特征、需要返回的特征的值。因为该函数代码在同一数据集上被调用多次，为了不修改原始数据集，创建一个新的列表对象。数据集这个列表中的各个元素也是列表，我们要遍历数据集中的每个元素，一旦发现符合要求的值，则将其添加到新创建的列表中。在 if 语句中，程序将符合特征的数据抽取出来 。后面讲述得更简单，这里我们可以这样理解这段代码：当我们按照某个特征划分数据集时，就需要将所有符合要求的元素抽取出来。

2)：选择最好的数据集划分方式

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1      #the last column is used for the labels
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):        #iterate over all the features
        featList = [example[i] for example in dataSet]#create a list of all the examples of this feature
        uniqueVals = set(featList)       #get a set of unique values
        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     #calculate the info gain; ie reduction in entropy
        if (infoGain > bestInfoGain):       #compare this to the best gain so far
            bestInfoGain = infoGain         #if better than current best, set to best
            bestFeature = i
    return bestFeature                      #returns an integer
```

该函数实现选取特征，划分数据集，计算得出最好的划分数据集的特征。函数 chooseBestFeatureToSplit()使用了之前的两个函数。在函数中调用的数据需要

满足一定的要求：第一，数据必须是一种由列表元素组成的列表，而且所有的列表元素都要具有相同的数据长度；第二，数据的最后一列或者每个实例的最后一个元素是当前实例的类别标签。一旦满足上述要求，我们就可以在函数的第一行判定当前数据集包含多少特征属性。在开始划分数据集之前，上述函数第 3 行代码计算了整个数据集的原始香农熵，我们保存最初的无序度量值，用于与划分完之后的数据集计算的熵值进行比较。第 1 个 for 循环遍历数据集中的所有特征。使用列表推导来创建新的列表，将数据集中所有第 i 个特征值或者所有可能存在的值写入这个新 list 中 。然后使用 Python 语言原生的集合（set）数据类型。集合数据类型与列表类型相似，不同之处仅在于集合类型中的每个值互不相同。遍历当前特征中的所有唯一属性值，对每个特征划分一次数据集 ，然后计算数据集的新熵值，并对所有唯一特征值得到的熵求和。信息增益是熵的减少或者是数据无序度的减少，大家肯定对于将熵用于度量数据无序度的减少更容易理解。最后，比较所有特征中的信息增益，返回最好特征划分的索引值 。

现在我们测试上面代码的实际输出结果，输入：

```
myDat, labels=createDataset()
print (myDat)
print(chooseBestFeatureTosplit(myDat))
```

输出得到：

```
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
0
```

代码运行结果告诉我们，第 0 个特征是最好的用于划分数据集的特征。结果是否正确呢？这个结果又有什么实际意义呢？数据集中的数据来源于表 1，让我们看下变量 myDat 中的数据。如果我们按照第一个特征属性划分数据，也就是说第一个特征是 1 的放在一个组，第一个特征是 0 的放在另一个组，数据一致性如何？按照上述的方法划分数据集，第一个特征为 1 的海洋生物分组将有两个属于鱼类，一个属于非鱼类；另一个分组则全部属于非鱼类。如果按照第二个特征分组，结果又是怎么样呢？第一个海洋动物分组将有两个属于鱼类，两个属于非鱼类；另一个分组则只有一个非鱼类。第一种划分很好地处理了相关数据。

- 递归构建决策树

目前我们已经学习了从数据集构造决策树算法所需要的子功能模块，其工作原理如下：得到原始数据集，然后基于最好的属性值划分数据集，由于特征值可能多于两个，因此可能存在大于两个分支的数据集划分。第一次划分之后，数据将被向下传递到树分支的下一个节点，在这个节点上，我们可以再次划分数据。因此我们可以采用递归的原则处理数据集。

递归结束的条件是：程序遍历完所有划分数据集的属性，或者每个分支下的所有实例都具有相同的分类。如果所有实例具有相同的分类，则得到一个叶子节点或者终止块。任何到达叶子节点的数据必然属于叶子节点的分类。

如果数据集已经处理了所有属性，但是类标签依然不是唯一的，此时我们需要决定如何定义该叶子节点，在这种情况下，我们通常会采用多数表决的方法决定该叶子节点的分类。

1)：统计 classList 中出现此处最多的元素(类标签)

```
import operator

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

该函数使用分类名称的列表，然后创建键值为 classList 中唯一值的数据字典，字典对象存储了 classList 中每个类标签出现的频率，最后利用 operator 操作键值排序字典，并返回出现次数最多的分类名称。

2)：创建决策树

```
def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]#stop splitting when all of the classes are equal
    if len(dataSet[0]) == 1: #stop splitting when there are no more features in dataSet
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]       #copy all of labels, so trees don't mess up existing labels
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value),subLabels)
    return myTree
```

上述程序使用两个输入参数：数据集和标签列表。标签列表包含了数据集中所有特征的标签。代码首先创建了名为 classList 的列表变量，其中包含了数据集的所有类标签。递归函数的第一个停止条件是所有的类标签完全相同，则直接返回该类标签。递归函数的第二个停止条件是使用完了所有特征，仍然不能将数据集划分成仅包含唯一类别的分组。由于第二个条件无法简单地返回唯一的类标签，这里使用上述程序中 chooseBestFeatureToSplit()函数挑选出现次数最多的类别作为返回值。

下一步程序开始创建树，字典变量 myTree 存储了树的所有信息，这对于其后绘制树形图非常重要。当前数据集选取的最好特征存储在变量 bestFeat 中，得到列表包含的所有属性值。

最后代码遍历当前选择特征包含的所有属性值，在每个数据集划分上递归调用函数 createTree()，得到的返回值将被插入到字典变量 myTree 中，因此函数终止执行时，字典中将会嵌套很多代表叶子节点信息的字典数据。

现在我们可以测试上面代码的实际输出结果，输入：

```
myDat, labels = createDataset()
myTree = createTree(myDat,labels)
print(myTree)
```

输出得到：

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

上述输出结果可用 Matplotlib 绘制出来，如下图：

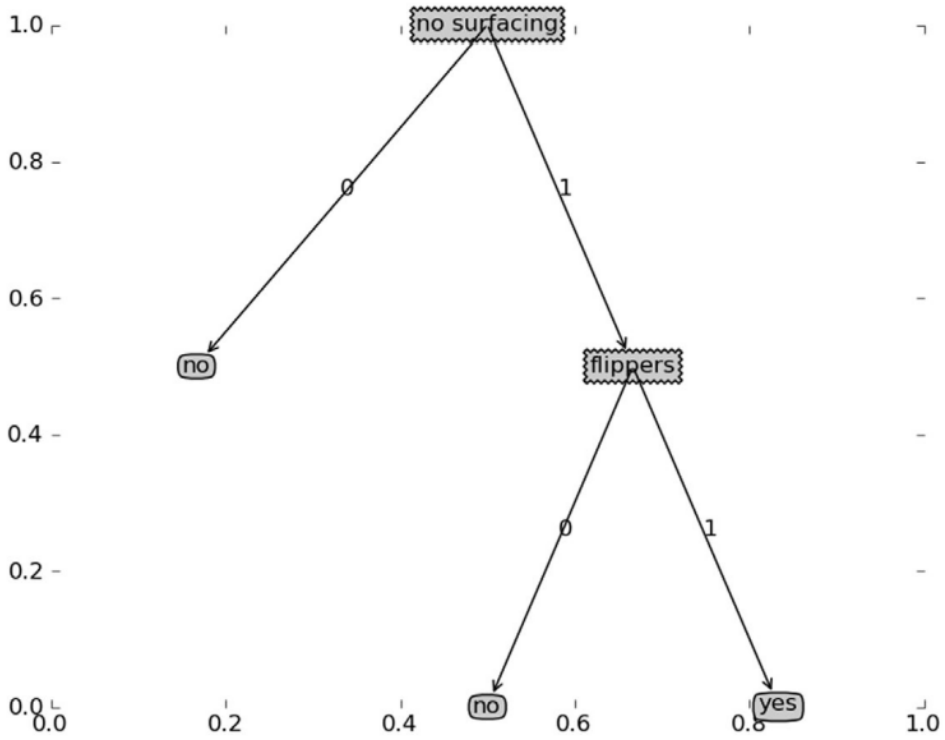

图 2 Matplotlib 下输出的树的结构图

变量 myTree 包含了很多代表树结构信息的嵌套字典，从左边开始，第一个关键字 no surfacing 是第一个划分数据集的特征名称，该关键字的值也是另一个数据字典。第二个关键字是 no surfacing 特征划分的数据集，这些关键字的值是 no surfacing 节点的子节点。这些值可能是类标签，也可能是另一个数据字典。如果值是类标签，则该子节点是叶子节点；如果值是另一个数据字典，则子节点是一个判断节点，这种格式结构不断重复就构成了整棵树。由此也能看出这棵树包含了 3 个叶子节点以及 2 个判断节点。