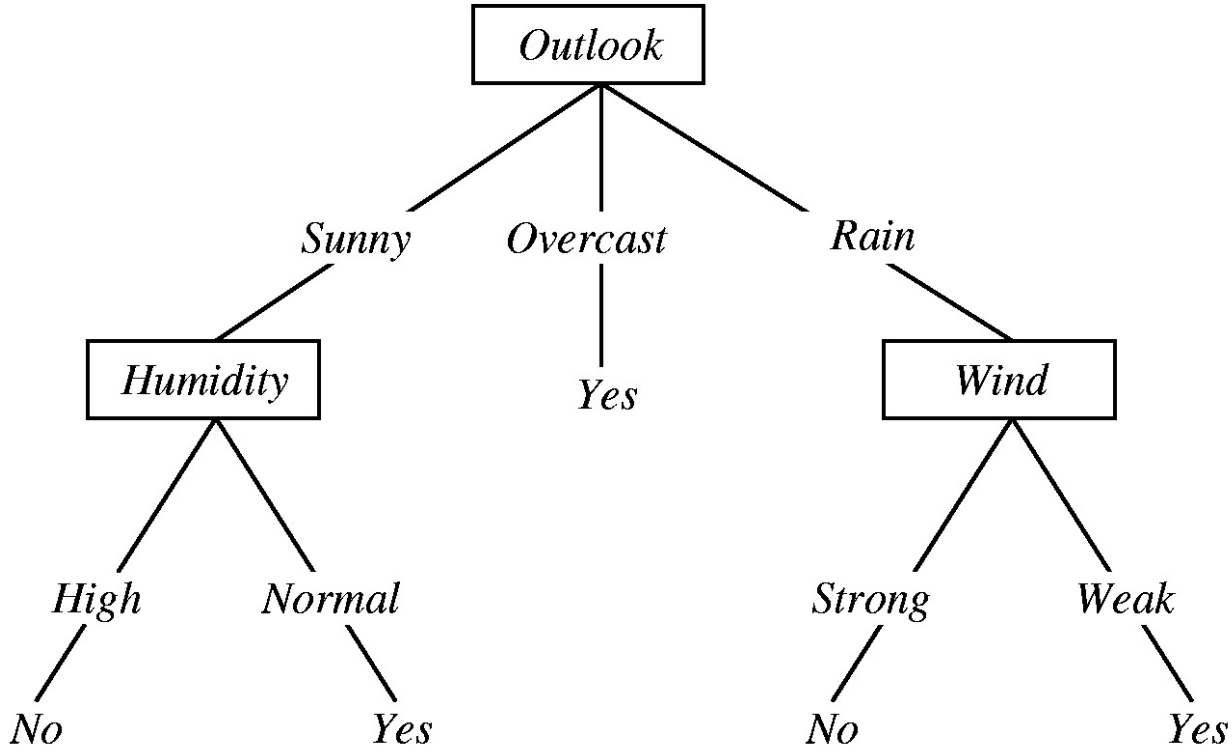
基本思想

决策树是基于树结构来进行决策的，这正是人类在面临决策问题时一种很自然的处理机制

一般的，一颗决策树包含一个根节点、若干个内部节点和若干个叶节点；叶节点对应于决策结果，其他的每个节点则对应于一个属性测试；每个节点包含的样本集合根据属性测试的结果被划分到子节点中；根节点包含样本全集。

> Wikipedia: A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules.

Example：



度量值

信息熵（Information Entropy）

信息熵是度量样本集合纯度最常用的一种指标

> Wikipedia: Information entropy is a concept from information theory. It tells how much information there is in an event. In general, the more uncertain or random the event is, the more information it will contain. The concept of information entropy was created by mathematician Claude Shannon.

假定当前样本集合D中第k类样本所占的比例为$p_k(k=1,2,...,|y|)$,则D的信息熵定义为

$$Entropy(D) = -\sum_{k=1}^{|y|}(p_k \cdot log_2 p_k)$$

$Entropy(D)$的值越小，则D的纯度越高

信息增益（Information Gain）

信息增益为总的熵减去某个分类标准对应的熵

假定属性a有V个可能的取值$\{a_1, a_2, a_3, \dots, a_V\}$，若使用a来对样本集合D进行划分，可以的到V个样本子集$\{D_1, D_2, D_3, \dots, D_V\}$，每个样本子集对应到一个分支节点上，考虑到不同分支节点样本数量不同，我们给每个子节点定义权重$|D_v|/|D|$，于是我们可以计算出用属性a对D进行划分所获的的信息增益

$$Gain(D, a) = Entropy(D) - \sum_{v=1}^{V}\frac{|D_v|}{|D|}Entropy(D_v)$$

一般而言，信息增益越大，则表明使用属性a来划分所获得的纯度提升越大。

选择属性时的目标函数

$$a_* = arg\,max_{a \in A}\ Gain(D, a)$$

增益率（Gain Ratio）

信息增益有一个缺点，它对取值数目较多的属性有所偏好，为了减少这种偏好的影响，我们引入增益率

$$Gain\ ratio(D, a) = \frac{Gain(D, a)}{IV(a)}$$

其中

$$IV(a) = -\sum_{v=1}^{V}\frac{|D_v|}{|D|} \cdot log_2 \frac{|D_v|}{|D|}$$

$IV(a)$称为属性a的固有值（Intrinsic value）属性a的可能取值越多，那个$IV(a)$的值通常越大

Note! 增益率准则对可取值数目较少的属性有所偏好，因此实际中，我们先从候选划分属性中找出信息增益高于平均水平的属性，再从中选出增益率最高的

基尼指数（Gini Index）

数据集D的纯度可以用基尼指数来度量，定义如下

$$Gini(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k} p_k p_k^{'} = 1 - \sum_{k=1}^{m} p_k^2$$

Gini(D)反应了从数据集中随机抽取两个样本，其类别标记不一致的概率

属性a的基尼指数定义为

$$Gini(D,a) = \sum_{v=1}^{V} \frac{|D_v|}{|D|} Gini(D_v)$$

选择属性时的目标函数

$$a_* = arg\,min_{a \in A}\ Gini(D,a)$$

剪枝处理

预剪枝

决策树从根节点开始以上文介绍的度量值为依据往外分枝延伸，分别计算分枝前和分枝后的验证集精度，然后判断是否分枝

优缺点：预剪枝可以有效降低过拟合的风险，还显著减少了决策树训练时间开销和测试时间开销，但是基于“贪心”使得某些分枝不能展开（后续展开可能导致性能显著提升的分枝），从而存在欠拟合的风险

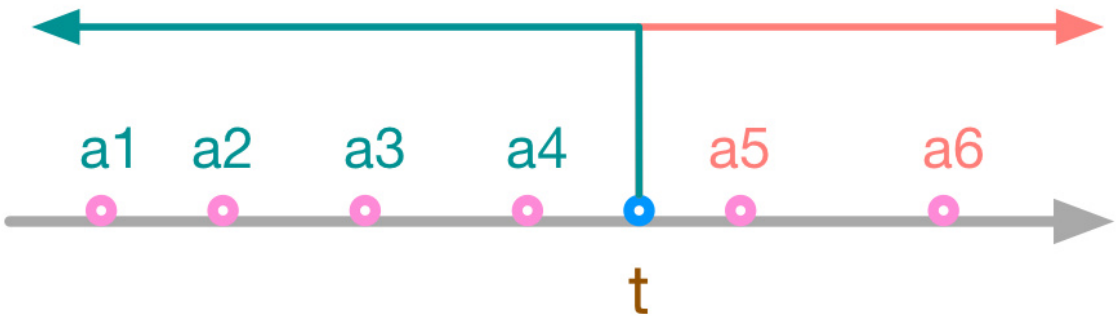
后剪枝

决策树从根节点开始以上文介绍的度量值为依据往外分枝延伸，生成整棵决策树，然后至底向上对各节点考虑是否要分枝

优缺点：能避免前欠拟合，但是训练开销较大

连续属性处理

当我们遇到属性值为连续数值时，我们可以利用连续属性离散化技术



给定样本D和连续属性a，假定a在D上出现n个不同的取值，将这些值从小到大排序，记为

$\{a_1, a_2, \dots, a_n\}$, 基于划分点t可以将集合D分为D_t^+, D_t^-，在区间$[a_i, a_{i+1}]$中任意取值t对划分的结果没有影响，故我们可以选取划分点集合如下

$$T_a = \{\frac{a_i + a_{i+1}}{2} | 1 \leq i \leq n-1\}$$

然后我们就能像离散值一样来考虑这些划分点并选取最优划分点，选取依据为

$$Gain(D, a) = max_{t \in T_a}\ Gain(D, a, t) = max_{t \in T_a}\ Entropy(D) - \sum_{\lambda \in \{+,-\}} \frac{|D_t^\lambda|}{|D|} Entropy(D_t^\lambda)$$

`Note!` 与离散属性不同，若当前节点划分属性为连续属性，该属性还可作为其后代节点的划分属性

缺失值处理

现实中我们经常会遇到数据存在缺失值的状况，如果直接把存在缺失值的数据给去除掉，那么我们的样本集合往往会受到较大影响，并且这样做对数据造成了很大的浪费。

数据缺失例子

编号	属性1	属性2	标签
1	A	1	N
2	-	2	Y
3	A	-	N
4	B	3	Y

划分属性的选择

给定数据集D和属性a，令$\hat{D}$表示D属性a上没有缺失值的样本子集，假定a有V个可取值

$\{a_1, a_2, \dots a_V\}$，令$\hat{D}^v$表示$\hat{D}$中在属性a上取值为a_v的样本子集，$\hat{D}_k$表示$\hat{D}$中属于第

$k(k = 1, 2, 3, \dots, |\ y\ |)$类的样本子集，假定我们为每一个样本$x$赋予一个权重$w_x$

据此我们定义如下变量

无缺失值样本所占比例

$$\rho = \frac{\sum_{x \in \hat{D}} w_x}{\sum_{x \in D} w_x}$$

无缺失样本中第k类所占比例

$$\hat{p_k} = \frac{\sum_{x \in \hat{D}_k} w_x}{\sum_{x \in \hat{D}} w_x}\ (1 \leq k \leq |y|)$$

无缺失样本中在属性a上取值a_v的样本所占的比例

$$\hat{r}_v = \frac{\sum_{x \in \hat{D}^v} w_x}{\sum_{x \in \hat{D}} w_x} \ (1 \leq v \leq V)$$

从而我们得到新的信息增益计算公式

$$Gain(D,a) = \rho \times Gain(\hat{D},a) = \rho \times (Entropy(\hat{D}) - \sum_{v=1}^{V} \hat{r}_v Entropy(\hat{D}^v))$$

其中

$$Entropy(\hat{D}) = -\sum_{k=1}^{|y|} \hat{p}_k \cdot log_2 \hat{p}_k$$

样本的划分

1. 若样本x在划分属性a上的属性已知，则将样本x划分入与其取值对应的子节点中，并保持权值不变
2. 若样本x在划分属性a上的属性未知，则将样本x同时划分入所有子节点中，且样本权值在与属性a_v对应的子节点中更新为$\hat{r} \cdot w_x$

多变量决策树

在n维空间考虑决策树可知，决策树的分类边界是由与若干个坐标轴平行的分段组成的，这样的分类边界能使学习结果有较好的解释性，但是在现实问题中，分类边界往往比较复杂，必须得用很多分段才能描述

为此引入多变量决策树，在这类决策树中，非叶节点不再是针对某个属性，而是针对属性的线性组合

进行测试，为此能使用斜的分类边界，每个非叶节点是一个形如$\sum_{i=1}^{n} w_i a_i = t$的线性分类器

Scalability

用比较合理的速度对具有上百万样本和上百个属性的数据集进行分类

数据挖掘领域为什么采用决策树呢?

- 和其他分类方法相比，决策树的学习速度较快
- 可以转换成简单的、易于理解的分类规则
- 可以采用SQL查询访问数据库
- 具有较高的分类准确性

`Code For Python`

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
>>> clf.predict([[2., 2.]])
array([1])
>>> clf.predict_proba([[2., 2.]])
array([[ 0.,  1.]])
```

[Decision trees can also be applied to regression problems]

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([ 0.5])
```