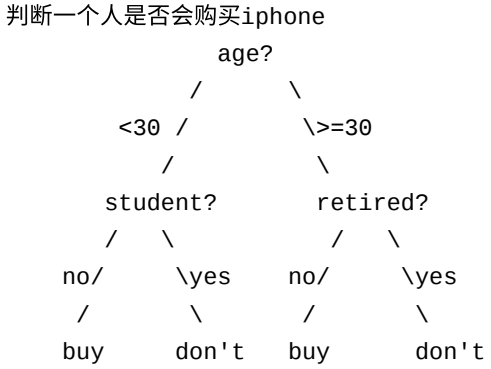
决策树 Decision Tree

概述

决策树是一种类似流程图的树结构，其中每个内部节点表示在一个属性上的测试，每个分支代表该测试的一个输出，而每个叶子结点存放一个分类标号。例如：



给定一个未知的X，在决策树上测试该元组的属性，跟踪到叶子结点，该叶子结点就存放着该X的分类的估计，比如一个元组 X=(age=25,student=false,retired=false)，那么根据上面决策树最终预测就是不购买iphone。

构造决策树，需要属性选择度量来选择将数据集最好地划分成不同类的属性，常见的三种决策树算法为ID3，C4.5和CART，其中C4.5可以认为是ID3的后继，CART与ID3大约同时独立的发明。

三个算法都采用贪心方法，自顶向下递归的分治方式构造。实际上大多数的决策树算法都沿用这种构造方法，随着树的构建，训练集递归地划分成较小的子集。

算法流程

输入：
数据分区D，可用属性列表List，属性选择度量Method
D可以理解为训练集的一个子集，分区D的子分区D_1,...D_k为D的一个划分，即子分区交集为空集，所有子分区的并集为D
输出：
决策树

```
GenerateDT(D，List，Method)
1．创建结点N;

2．if 数据分区D中所有的元组都属于同一个分类C then
        标记结点N的分类标号为C，返回N作为叶子结点;

3．if List 为空 then
        标记结点N的分类标号为分区中大多数元组所属的集合C，返回N作为叶子结点;

4．基于Method，获得最佳分裂属性split_attr，split_criterion;

5．if split_attr取离散值且允许多路划分(比如C4.5)
        remove split_attr from 可用属性列表List;
        // 如果是严格二路划分的树例如CART，那么此处就无需移除split_attr

6．for i in split_criterion
        // D_i 为根据split_criterion对 D 的第i个划分
        if D_i 为空 then
            加一个叶子结点到结点N，标记为D中的多数类;
        else
            加一个由GenerateDT(D_i，List，Method)产生的子节点到结点N;

7．return 结点N;
```

属性选择度量

属性选择度量是启发式地将给定类标记的数据划分成独立类的方法，又称分裂规则，因为它们决定在给定结点上的元组如何分裂。如果根据分裂准则把 D 划分成小的分区 D_i ，最理想的情况是小分区 D_i 里的元组都是拥有相同类标记的，“最好的分裂准则”是最接近理想情况的划分。对应ID3，C4.5，CART的度量是：

1. 信息增益
   ID3使用信息增益作为属性选择度量，选择具有最高信息增益的属性作为结点的分裂属性，该属性使结果分区中对元组分类所需要的信息量最少，反映这些分区中的最小随机性，使得对一个对象分类所需要的期望测试次数最少，确保找到一棵相对简单的树，假设分区D中有来自不同的 m 个分类的元组。
   对D中元组分类所需的信息为：

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

   其中 p_i 为D中元组属于分类 C_i 的概率，用 $|C_{i,D}|/|D|$ 估计，简单来说就是分类标识为 C_i 的元组在分区 D 中所占的比例。基于属性A对D的元组进行划分为 v 个分区后，分类所需的信息的加权平均为：

$$\mathrm{Info}_A(D)=\sum_{j=1}^{v}\frac{|D_j|}{|D|}\mathrm{Info}(D_j)$$

信息增益定义为：

$$\mathrm{Gain}(A)=\mathrm{Info}(D)-\mathrm{Info}_A(D)$$

上式可以认为是 划分前分类所需的信息 - 划分后分类所需的信息 ， $\mathrm{Gain}(A)$ 描述了以属性A对D划分所增加的信息。
对于离散值的划分无需赘述，而对于连续值的划分则需要指定 v 之后，在属性A的可能划分点上逐一测试，找到 $v-1$个分裂点 ，假设分区D中A的取值为$\{a_1,...,a_n\}$ ， 那么 $\frac{a_i+a_{i+1}}{2}$ 可以作为可能的分裂点，测试则变为 $split_i<A\leq split_{i+1}$。
信息增益度量偏向与多输出测试，亦即倾向于选择取值可能性多的属性，例如每个元组有一个独一无二的ID属性，那么选择ID属性做划分，分出来的每个子分区都只有一个元组，由于每个子分区都是纯的，因此$\mathrm{info}(D)=0$ ，必有信息增益最大，但这种划分毫无意义，因此，ID3的原作者Quinlan后续提出了改进的C4.5，使用信息增益率代替信息增益。

2. 信息增益率
信息增益率为信息增益的扩充，试图克服上述倾向，定义了规范化的信息增益———“分裂信息”：

$$\mathrm{SplitInfo}_A(D)=-\sum_{j=1}^{v}\frac{|D_j|}{|D|}\log_2\frac{|D_j|}{|D|}$$

该值代表由分区D划分成对应属性A测试的 v 个输出的子分区产生的信息。
信息增益率定义为：

$$\mathrm{GainRate}(A)=\frac{\mathrm{Gain}(A)}{\mathrm{SplitInfo}_A(D)}$$

上式可以认为是 获得的信息 / 产生的信息，因此信息增益率可以理解为，用属性A对D划分产生的信息的有效利用率。
需要注意的是，随着划分信息趋向于0,该比率变得不稳定，因此应增加一个约束：选取的测试的信息增益必须较大，至少与考察的所有测试的平均增益一样大。

3. 基尼指数
基尼指数可在CART中使用，度量分区D的不纯程度，定义为：

$$\mathrm{Gini}(D)=1-\sum_{i=1}^{m}p_i^2$$

p_i 与前面定义一致，为D中元组属于分类 C_i 的概率，用 $|C_{i,D}|/|D|$ 估计。
基尼指数考虑每个属性的二元划分。首先假设属性 A 取离散值，可能的取值有 v 个，记这个集合为S，考虑S的所有二元划分，对于任意划分$\{S_1, S_2 = S_1^c\}$，可以作出 $A \in S_1$ or $A \in S_2$ 这一测试，去掉S_1为全集和空集的情况，共有$2^v - 2$种划分。考虑二元划分时，计算每个结果分区的不纯度的加权和，假设测试将D划分为D_1, D_2：

$$\mathrm{Gini}_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2)$$

对于每个属性，考虑每种可能的二元划分，对于离散的属性，选择该属性产生最小基尼指数的子集作为它的分裂子集。
对于连续的属性，处理方法与前面类似，假设分区D中A的取值为$\{a_1, ..., a_n\}$，每个可能的划分点 $split_i = \frac{a_i + a_{i+1}}{2}$ 可以将取值范围划分为 $S_1 = \{x | x < split_i\}$ 和 $S_2 = \{x | x \geq split_i\}$ ，其余操作同离散值，S 的划分共有 n 种可能。
不纯度的降低量为：

$$\Delta\mathrm{Gini}(A) = \mathrm{Gini}(D) - \mathrm{Gini}_A(D)$$

最大化不纯度的降低量，等价于最小化$\mathrm{Gini}_A(D)$，该属性A与分裂子集$\{S_1, S_2\}$一起形成分裂准则。

其他属性选择度量
不同的选择度量有不同的倾向性，信息增益倾向多值属性，信息增益率倾向与不平衡的划分，基尼指数同样偏向多值属性且当类的数量很大时会有困难。

引用一个简易列举：

> CHAID (Chi-Squared Automatic Interaction Detection)：
> 是AID的后续版本， 在认识的AID的偏差性后， 利用χ2-Test和F-Test结合的方式来选择属性，来降低偏差。 同时CHAID也意识到二叉树的复杂性， 运行多叉树合并产生更小的决策树。
>
> FACT (Fast Algorithm for Classification Trees)：
> 是QUEST的一个早期版本， 因此在后面说明中不会提及。FACT讲符号的属性全部转换成数值型的， 然后，利用LDA（linear discriminant analysis）来将属性进行分割的。 它进行特征选择的时候， 利用ANOVA (analysis of variance)的F-Test， 优先选择最大的F Ratio的属性进行划分。
>
> QUEST (Quick, Unbiased, Efficient, Statistical Tree)：
> 是FACT的后续版本， 并且将FACT的基于LDA（linear discriminant analysis）的划分， 修改成了基于QDA（quadratic discrimination analysis）的划分。 并且在特征选择的时候， 连续特征和FACT一样基于F-Test， 但是离散特征是基于χ2-Test的。 在QUEST中Missing处理比较简单，就是用插值（Imputation）。

> CRUISE (Classification Rule with Unbiased Interaction Selection and Estimation)：
> 是QUEST的后续算法， 除了继承了QUEST的F-Test（连续值）和χ2-Test（离散值）的划分， 还引入了叫2D的特征划分方式。 2D的划分中， 两两属性之间都要进行5次测试（2次marginal tests和3次interaction tests）
> 并且划分也有改变， 先进行Box-Cox Transformation预处理， 再进行LDA划分。 另外还有一个重大改变是CRUISE经过Box-Cox变换后，可以将一个属性划分成多个分支（多叉树）。 CRUISE采用了CART树处理Missing的Surrogate Splitting办法。
>
> GUIDE （Generalized, Unbiased, Interaction Detection and Estimation）：
> GUIDE更像一个QUEST 和CART树的一个综合体的Bagging升级版。 它继承了QUEST和CRUISE的特征划分方式， 但是加入了Variable Importance的排序和按阈值选择部分特征集。 并且GUIDE和CART类似也可以用作Regression。 由于受到Random Forest成功的影响， GUIDE自带了Bagging的两种机制（Random Forest 和Extremely Randomized Trees）。 但是GUIDE里面Missing没有采用CART的方式， 而是把Missing看成一类特殊值，但是同时根据数据类型，具有插值的mean（连续型），或者是常量（符号型）。
>
> MARS（Multivariate Adaptive Regression Splines）：
> 是Frieman提出的一个多变量回归的线性样板。 基于多个Hinge Function把不同变量的回归部分拼接起来。 因此来说， MARS与传统的树还是差异蛮大的，因此在后面说明中不会提及。 但是这个过程中Friedman应用了Stepwise Regression的思想。这或许是他后来提出Gradient Boosting方法的一个基础。
>
> 作者：CodingFish
> 链接：https://www.jianshu.com/p/338939130b24
> 來源：简书
>

剪枝

创建决策树时，由于数据中的噪声和离群点，许多分支实际上是异常数据导致的，如果保留过多的这些分支，就会导致过拟合的问题。剪枝后的树更简单，运行速度更快，通常情况下在独立验证集上会有更好的表现。

先剪枝

先剪枝技术通过提前停止树的构建，例如在数据分区的不纯度低于一定阈值就不再分裂，直接作为叶子节点，属于Early Stopping的正则化方法，剪枝效果取决于阈值的选取。

后剪枝

在决策树中，后剪枝的方法更为常用。即先完全展开整个决策树（每个叶子节点都是纯净分区），再剪掉某些结点上的子树使之合并为一个叶子结点。
CART使用的代价复杂度剪枝，该方法将树的复杂度看作树中叶子结点的个数和树的错误率的函数，对每

个结点N的子树，评估其代价复杂度和对N剪枝后的代价复杂度，比较两者，若剪枝能降低代价复杂度则实施剪枝。该算法要使用一个标记类元组的剪枝集来评估代价复杂度，该集合独立于训练集和验证集，算法产生一个渐进的剪枝树的集合。
C4.5使用悲观剪枝，与代价复杂度剪枝类似，也使用错误率评估并对子树剪枝作出决定。但是，悲观剪枝不需要剪枝集，而是使用训练集估计错误率，但由于训练集的错误率过于乐观，因此悲观剪枝通过加上一个惩罚项来调节从训练集获得的错误率，以抵消偏倚。

用sklearn实现决策树

数据说明

Iris数据集，3种鸢尾花，每个元组有4个属性['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

模型说明

scikit-learn uses an optimised version of the CART algorithm.
sklearn的决策树模型为CART

class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

criterion : string, optional (default="gini")
Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
支持基尼系数和信息增益

splitter : string, optional (default="best")
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
best每次找最优的划分属性，random则随机地找几个属性试试，random比较快，best生成的树比较小

max_depth : int or None, optional (default=None)
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
最大树高限制，属于先剪枝

min_samples_split : int, float, optional (default=2)
The minimum number of samples required to split an internal node:
If int, then consider min_samples_split as the minimum number.
If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
最小划分元组数，如果结点元组太少，就不划分了，属于先剪枝

min_samples_leaf : int, float, optional (default=1)
The minimum number of samples required to be at a leaf node:
If int, then consider min_samples_leaf as the minimum number.
If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
叶结点最小元组数

min_weight_fraction_leaf : float, optional (default=0.)
The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
叶子结点最小权重比例，限制叶子结点的参数，要求叶子结点内元组权重总和大于一个值，默认每个样本权值一样

max_features : int, float, string or None, optional (default=None)

The number of features to consider when looking for the best split:
If int, then consider max_features features at each split.
If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
If “auto”, then max_features=sqrt(n_features).
If “sqrt”, then max_features=sqrt(n_features).
If “log2”, then max_features=log2(n_features).
If None, then max_features=n_features.
Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.
最多测试几个属性，实践中并不是真正测试每一个属性的话，那么要设置一个寻找次数限制，以优化性能

random_state : int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

max_leaf_nodes : int or None, optional (default=None)

Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
限制最大叶子结点数目

min_impurity_decrease : float, optional (default=0.)

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
最小不纯度减少值，如果划分后不纯度减少大于本值才能划分