决策树

1. 决策树是一种基本的分类与回归方法。这里主要讨论决策树用于分类。
2. 决策树模型是描述对样本进行分类的树形结构。树由结点和有向边组成：
    - 内部结点表示一个特征或者属性。
    - 叶子结点表示一个分类。
    - 有向边代表了一个划分规则。
3. 决策树从根结点到子结点的的有向边代表了一条路径。决策树的路径是互斥并且是完备的。
4. 用决策树分类时，对样本的某个特征进行测试，根据测试结果将样本分配到树的子结点上。此时每个子结点对应该特征的一个取值。

   递归地对样本测试，直到该样本被划分叶结点。最后将样本分配为叶结点所属的类。
5. 决策树的优点：可读性强，分类速度快。
6. 决策树学习通常包括3个步骤：
    - 特征选择。
    - 决策树生成。
    - 决策树剪枝。

一、 原理

1.1 基本概念

1. 决策树模型可以认为是 `if-then` 规则的集合，也可以认为是定义在特征空间与类空间上的条件概率分布。
2. 决策树将特征空间划分为互不相交的单元，在每个单元定义一个类的概率分布，这就构成了一个条件概率分布。
    - 决策树的每一条路径对应于划分中的一个基本单元。
    - 设某个单元 $\mathbb{S}$ 内部有 N_S 个样本点，则它定义了一个条件概率分布 $p(y \mid \vec{\mathbf{x}}), \vec{\mathbf{x}} \in \mathbb{S}$ 。
        - 单元上的条件概率偏向哪个类，则该单元划归到该类的概率较大。即单元的类别为：

          $$\arg\max_{y} p(y \mid \vec{\mathbf{x}}), \vec{\mathbf{x}} \in \mathbb{S}$$

        - 决策树所代表的条件概率分布由各个单元给定条件下类的条件概率分布组成。
3. 决策树的学习目标是：根据给定的训练数据集构造一个决策树模型，使得它能够对样本进行正确的分类。
4. 决策树最优化的策略是：损失函数最小化。决策树的损失函数通常是正则化的极大似然函数。
5. 选择最优决策树的问题是个 `NP` 完全问题。一般采用启发式方法近似求解这个最优化问题，这时的解为次最优的。
6. 决策树学习的算法通常递归地选择最优特征，并根据该特征对训练数据进行分割，使得对各个子数据集有一个最好的分类。

   这一过程对应着特征空间的划分，也对应着决策树的构建。

1.2 生成

1. 决策树生成算法：
    - 构建根结点：将所有训练数据放在根结点。
    - 选择一个最优特征，根据这个特征将训练数据分割成子集，使得各个子集有一个在当前条件下最好的分类。
        - 若这些子集已能够被基本正确分类，则将该子集构成叶结点。
        - 若某个子集不能够被基本正确分类，则对该子集选择新的最优的特征，继续对该子集进行分割，构建相应的结点。
    - 如此递归下去，直至所有训练数据子集都被基本正确分类，或者没有合适的特征为止。
2. 上述生成的决策树可能对训练数据有很好的分类能力，但是对于未知的测试数据却未必有很好要的分类能力，即可能发生过拟合的现象。
    - 解决的方法是：对生成的决策树进行剪枝，从而使得决策树具有更好的泛化能力。
    - 剪枝是去掉过于细分的叶结点，使得该叶结点中的子集回退到父结点或更高层次的结点并让其成为叶结点。
3. 决策树的生成对应着模型的局部最优，决策树的剪枝则考虑全局最优。
4. 如果学习任意大小的决策树，则可以将决策树算法视作非参数算法。但是实践中经常有大小限制（如限制树的高度、限制树的叶结点数量），从而使得决策树成为有参数模型。

二、 特征选择

1. 特征选择的关键是：选取对训练数据有较强分类能力的特征。若一个特征的分类结果与随机分类的结果没有什么差别，则称这个特征是没有分类能力的。
2. 通常特征选择的指标是：信息增益或者信息增益比。这两个指标刻画了特征的分类能力。
3. 对于分布 $p(y)$，熵为 $H(y)=\sum_y -p(y)\log p(y)$ 。

   定义数据集 $\mathbb{D}$ 的经验熵为：$H(\mathbb{D})=-\sum_{k=1}^{K}\frac{N_k}{N}\log\frac{N_k}{N}$ 。

   其中：
    - 样本的类别分别为 $c_1,c_2,\cdots,c_K$ 。
    - 类别 c_k 的样本的数量为 N_k ，所有样本的总数为 N 。

      > 因此有：$\sum_{k=1}^{K}N_k=N$ 。
    - $\frac{N_k}{N}$ 是概率 $p(y=c_k)$ 的估计。
    - $H(\mathbb{D})$ 就是熵 $H(y)$ 的估计。它刻画了数据集 $\mathbb{D}$ 中样本的类别分布情况。
4. 对于特征 A ，定义数据集 $\mathbb{D}$ 在 A 上的经验熵为：$H_A(\mathbb{D})=-\sum_{i=1}^{n_A}\frac{N_{a_i}}{N}\log\frac{N_{a_i}}{N}$ 。

   其中：
    - 特征 A 的取值范围为 $\{a_1,a_2,\cdots,a_{n_A}\}$ 。
    - 属性$A=a_i$ 的样本的数量为 N_{a_i} 。

      > 因此有：$\sum_{i=1}^{n_A}N_{a_i,k}=N_k$
    - $\frac{N_{a_i}}{N}$ 是概率 $p(A=a_i)$ 的估计。
    - $H_A(\mathbb{D})$ 刻画了数据集 $\mathbb{D}$ 中的样本在属性 A 上的取值分布情况。
5. 对于特征 A ，其条件熵为：$H(y\mid A)=\sum_A p(A)H(y\mid A)$ 。

定义数据集 $\mathbb{D}$ 关于特征 A 的经验条件熵为：

$$H(\mathbb{D}\mid A)=\sum_{i=1}^{n_A}p(A=a_i)H(y\mid A=a_i)=\sum_{i=1}^{n_A}\frac{N_{a_i}}{N}\left[-\sum_{k=1}^{K}\frac{N_{a_i,k}}{N_{a_i}}\log\frac{N_{a_i,k}}{N_{a_i}}\right]$$

其中：

- 属性 $A=a_i$ 且类别为 c_k 的样本的数量为 $N_{a_i,k}$，所有样本的总数为 N 。

  > 因此有：$\sum_{k=1}^{K}N_{a_i,k}=N_{a_i}$，$\sum_{i=1}^{n_A}N_{a_i}=\sum_{k=1}^{K}N_k=N$。

- $-\sum_{k=1}^{K}\frac{N_{a_i,k}}{N_{a_i}}\log\frac{N_{a_i,k}}{N_{a_i}}$ 是条件熵 $H(y\mid A=a_i)$ 的估计。它刻画了数据集 $\mathbb{D}$ 中，属性 $A=a_i$ 中的那些样本中的类别的分布情况。
- $H(\mathbb{D}\mid A)$ 是条件熵 $H(y\mid A)$ 的估计。

2.1 信息增益

1. 特征 A 对训练数据集 $\mathbb{D}$ 的信息增益 $g(\mathbb{D},A)$ 定义为：集合 $\mathbb{D}$ 的经验熵 $H(\mathbb{D})$ 与关于特征 A 经验条件熵 $H(\mathbb{D}\mid A)$ 之差。即： $g(\mathbb{D},A)=H(\mathbb{D})-H(\mathbb{D}\mid A)$ 。

   由于熵 $H(y)-H(y\mid A)$ 也称作互信息，因此信息增益也等于训练数据集中类与特征的互信息。

2. 决策树学习可以应用信息增益来选择特征。给定训练集 $\mathbb{D}$ 和特征 A ：
   - 经验熵 $H(\mathbb{D})$ 刻画了对数据集 $\mathbb{D}$ 进行分类的不确定性。
   - 经验条件熵 $H(\mathbb{D}\mid A)$ 刻画了在特征 A 给定条件下，对数据集 $\mathbb{D}$ 分类的不确定性。
   - 信息增益 $H(\mathbb{D})-H(\mathbb{D}\mid A)$ 刻画了由于特征 A 的确定，从而使得对数据集 $\mathbb{D}$ 的分类的不确定性减少的程度。
3. 不同的特征往往具有不同的信息增益。
   - 信息增益大的特征具有更强的分类能力 。
   - 如果一个特征的信息增益为0，则表示该特征没有什么分类能力。

2.2 信息增益比

1. 以信息增益作为划分训练集的特征选取方案，存在偏向于选取值较多的特征的问题。

   公式 $H(\mathbb{D}\mid A)=\sum_{i=1}^{n_A}\frac{N_{a_i}}{N}\left[-\sum_{k=1}^{K}\frac{N_{a_i,k}}{N_{a_i}}\log\frac{N_{a_i,k}}{N_{a_i}}\right]$ 中：

   - 当极限情况下 ， 特征 A 在每个样本上的取值都不同，即 $n_A=N$ 。
     - 此时特征 A 将每一个样本都划分到不同的子结点。即：$N_{a_i}=1,\quad i=1,\cdots,n_A$ 。
     - 由于 $\sum_{k=1}^{K}N_{a_i,k}=N_{a_i}=1$ ，因此有：$N_{a_i,k}\le 1,\quad i=1,\cdots,n_A;\quad k=1,\cdots,K$ 。

       即：$N_{a_i,k}$ 取值为 0 或者 1 。因此有：$\frac{N_{a_i,k}}{N_{a_i}}\log\frac{N_{a_i,k}}{N_{a_i}}=0$ 。
     - 最终使得$H(\mathbb{D}\mid A)=0$ 。
   - 条件熵的最小值为 0，这意味着该情况下的信息增益达到了最大值。

     然而很显然这个特征 A 显然不是最佳选择，因为它并不具有任何分类能力。

2. 可以通过定义信息增益比来解决该问题。

   特征 A 对训练集 $\mathbb{D}$ 的信息增益比 $g_R(\mathbb{D},A)$ 定义为：信息增益 $g(\mathbb{D},A)$ 与关于特征 A 的熵 $H_A(\mathbb{D})$ 之比：

$$g_R(\mathbb{D},A)=\frac{g(\mathbb{D},A)}{H_A(\mathbb{D})}$$

$H_A(\mathbb{D})$ 表征了特征 A 对训练集 $\mathbb{D}$ 的拆分能力。

因为 $H_A(\mathbb{D})$ 只考虑样本在特征 A 上的取值，而不考虑样本的标记 y ，所以这种拆分并不是对样本的分类。

3. 信息增益比本质上是对信息增益乘以一个加权系数：
    - 当特征 A 的取值集合较大时，加权系数较小，表示抑制该特征。
    - 当特征 A 的取值集合较小时，加权系数较大，表示鼓励该特征。

三、生成算法

1. 决策树有两种常用的生成算法：
    - `ID3` 生成算法。
    - `C4.5` 生成算法。
2. `ID3` 生成算法和 `C4.5` 生成算法只有树的生成算法，生成的树容易产生过拟合：对训练集拟合得很好，但是预测测试集效果较差。

3.1 ID3 生成算法

1. `ID3` 生成算法核心是在决策树的每个结点上应用信息增益准则选择特征，递归地构建决策树。
    - 从根结点开始，计算结点所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征划分出子结点。
    - 再对子结点递归地调用以上方法，构建决策树。
    - 直到所有特征的信息增益均很小或者没有特征可以选择为止，最后得到一个决策树 。

      如果不设置特征信息增益的下限，则可能会使得每个叶子都只有一个样本点，从而划分得太细。
2. `ID3` 生成算法：
    - 输入：
        - 训练数据集 $\mathbb{D}$
        - 特征集合 $\mathbb{A}=\{A_1,A_2,\cdots,A_n\}$
        - 特征信息增益阈值 $\varepsilon>0$
    - 输出：决策树 T
    - 算法步骤：
        - 若 $\mathbb{D}$ 中所有样本均属于同一类 c_k，则 T 为单结点树，并将 c_k 作为该结点的类标记，算法终止。
        - 若 $\mathbb{A}=\phi$，则 T 为单结点树，将 $\mathbb{D}$ 中样本数最大的类 c_k 作为该结点的类标记，算法终止。
        - 否则计算 $g(\mathbb{D},A_j),\ j=1,2,\cdots,n$，选择信息增益最大的特征 A_g ：
            - 若 $g(\mathbb{D},A_g)<\varepsilon$ ，则置 T 为单结点树，将 $\mathbb{D}$ 中样本数最大的类 c_k 作为该结点的类标记，算法终止 。
            - 若 $g(\mathbb{D},A_g)\geq\varepsilon$ ，则对 A_g 特征的每个可能取值 $a_i,i=1,2,\cdots,n_g$ ，根据 $A_g=a_i$ 将 $\mathbb{D}$ 划分为若干个非空子集 $\mathbb{D}_i$ 。

              将 $\mathbb{D}_i$ 中样本数最大的类作为该子集的标记，构建子结点。
        - 对第 i 个子结点， 以 $\mathbb{D}_i$ 为训练集， 以 $\mathbb{A}-\{A_g\}$ 为特征集，递归地调用前面的步骤来构建子树。

3.2 C4.5 生成算法

1. `C4.5` 生成算法与 `ID3` 算法相似，但是 `C4.5` 算法在生成过程中用信息增益比来选择特征。

四、剪枝算法

1. 决策树生成算法生成的树往往对于训练数据拟合很准确，但是对于未知的测试数据分类却没有那么准确。即出现过拟合现象。

   过拟合产生得原因是决策树太复杂。解决的办法是：对决策树剪枝，即对生成的决策树进行简化。

2. 决策树的剪枝是从已生成的树上裁掉一些子树或者叶结点，并将根结点或者其父结点作为新的叶结点。

   剪枝的依据是：极小化决策树的整体损失函数或者代价函数。

3. 决策树生成算法是学习局部的模型，决策树剪枝是学习整体的模型。即：生成算法仅考虑局部最优，而剪枝算法考虑全局最优。

4.1 原理

1. 设树 T 的叶结点个数为 $|T_f|$ 。设 $T_t, t=1,2,\cdots,|T_f|$ 为树的叶结点，该叶结点有 N_t 个样本点，其中属于 c_k 类的样本点有 $N_{t,k}$ 个， $k=1,2,\cdots,K$ 。

   > 这意味着： $\sum_{k=1}^K N_{t,k}=N_t$ 。

   令 $H(t)$ 为叶结点 T_t 上的经验熵，令 $\alpha\geq 0$为参数，则决策树 T 的损失函数定义为：

   $$C_\alpha(T)=\sum_{t=1}^{|T_f|}N_tH(t)+\alpha|T_f|$$

   其中： $H(t)=-\sum_{k=1}^K\frac{N_{t,k}}{N_t}\log\frac{N_{t,k}}{N_t}$ 。

   - 叶结点个数越多，表示决策树越复杂，则损失函数越大。
   - 叶结点经验熵越大，表示叶结点的样本类别分布很分散，则损失函数越大。
   - 叶结点经验熵还需要加权，权重为叶结点大小。即：越大的叶结点，其分类错误的影响越大。

2. 令 $C(T)=\sum_{t=1}^{|T_f|}N_tH(t)=-\sum_{t=1}^{|T_f|}\sum_{k=1}^K N_{t,k}\log\frac{N_{t,k}}{N_t}$ ，则： $C_\alpha(T)=C(T)+\alpha|T_f|$ 。

   其中 $\alpha|T_f|$ 为正则化项，$C(T)$ 表示预测误差。

3. $C(T)=0$ 意味着 $N_{t,k}=N_t$ ，即每个结点 T_t 内的样本都是纯的，即单一的分类。

   决策树划分得越细致，则 T 的叶子结点越多。当叶结点的数量等于样本集的数量时，树 T 的每个叶子结点只有一个样本点。此时每个叶结点内的样本都是纯的，从而 $C(T)=0$ 。

   这样的决策树其实是没有什么实用价值的，所以必须使用正则化项来约束决策树的复杂程度。

4. 参数 α 控制预测误差与模型复杂度之间的关系。

   - 较大的 α 会选择较简单的模型 。
   - 较小的 α 会选择较复杂的模型。
   - $\alpha=0$ 只考虑对训练集的拟合，不考虑模型复杂度。

5. 决策树剪枝的准则是：考虑当 α 确定时， $C_\alpha(T)$ 最小化。这等价于正则化的极大似然估计。

4.2 算法

1. 决策树的剪枝算法：

   - 输入：
     - 生成算法产生的决策树 T

        - 参数 α
    - 输出： 修剪后的决策树 T_α
    - 算法步骤：
        - 对树 T 每个结点 $T_t, t=1,2,\cdots,|T_f|$ ，计算其经验熵 $H(t)$ 。
        - 递归地从树的叶结点向上回退：

          设一组叶结点回退到父结点之前与之后的整棵树分别为 T 与 T'，对应的损失函数值分别为 $C_\alpha(T)$ 与$C_\alpha(T')$ 。若 $C_\alpha(T')\leq C_\alpha(T)$， 则进行剪枝，将父结点变成新的叶结点。
        - 递归回退直到不能继续为止，得到损失函数最小的子树 T_α 。

五、CART 树

1. `CART:classfification and regression tree` ： 学习在给定输入随机变量 $\vec{\mathbf{x}}$ 条件下，输出随机变量 y 的条件概率分布的模型。
    - 它同样由特征选取、树的生成、剪枝组成。
    - 它既可用于分类，也可用于回归。
2. `CART` 假设决策树是二叉树：
    - 内部结点特征的取值为 `是` 与 `否` 。其中：左侧分支取 `是`，右侧分支取 `否` 。
    - 它递归地二分每个特征，将输入空间划分为有限个单元。
3. `CART` 树与 `ID3` 决策树和 `C4.5` 决策树的重要区别：
    - `CART` 树是二叉树，而后两者是 `N` 叉树。

      由于是二叉树，因此 `CART` 树的拆分不依赖于特征的取值数量。因此 `CART` 树也就不像 `ID3` 那样倾向于取值数量较多的特征。
    - `CART` 树的特征可以是离散的，也可以是连续的。

      而后两者的特征是离散的。如果是连续的特征，则需要执行分桶来进行离散化。

      > `CART` 树处理连续特征时，也可以理解为二分桶的离散化。
4. `CART` 算法分两步：
    - 决策树生成：用训练数据生成尽可能大的决策树。
    - 决策树剪枝：用验证数据基于损失函数最小化的标准对生成的决策树剪枝。

5.1 CART 生成算法

1. `CART` 生成算法有两个生成准则：
    - `CART` 回归树：用平方误差最小化准则。
    - `CART` 分类树：用基尼指数最小化准则。

5.1.1 CART 回归树

5.1.1.1 划分单元和划分点

1. 一棵 `CART` 回归树对应着输入空间的一个划分，以及在划分单元上的输出值。

   设输出 y 为连续变量，训练数据集 $\mathbb{D}=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\}$。

   设已经将输入空间划分为 M 个单元 $R_1,R_2,\cdots,R_M$，且在每个单元 R_m 上有一个固定的输出值 c_m。则 `CART` 回归树模型可以表示为：

$$f(\vec{\mathbf{x}}) = \sum_{m=1}^{M} c_m I(\vec{\mathbf{x}} \in R_m)$$

其中 $I(\cdot)$ 为示性函数。

2. 如果已知输入空间的单元划分，基于平方误差最小的准则，则 `CART` 回归树在训练数据集上的损失函数为：

$$\sum_{m=1}^{M} \sum_{\vec{\mathbf{x}}_i \in R_m} (\tilde{y}_i - c_m)^2$$

根据损失函数最小，则可以求解出每个单元上的最优输出值 $\hat{c}_m$ 为： R_m 上所有输入样本 $\vec{\mathbf{x}}_i$ 对应的输出 $\tilde{y}_i$ 的平均值。

即：$\hat{c}_m = \frac{1}{N_m} \sum_{\vec{\mathbf{x}}_i \in R_m} \tilde{y}_i$ ，其中 N_m 表示单元 R_m 中的样本数量。

3. 问题是输入空间的单元划分是未知的。如何对输入空间进行划分？

设输入为 n 维： $\vec{\mathbf{x}} = (x_1, x_2, \cdots, x_n)^T$ 。

- 选择第 j 维 x_j 和它的取值 s 作为切分变量和切分点。定义两个区域：

$$R_1(j, s) = \{\vec{\mathbf{x}} \mid x_j \le s\}$$
$$R_2(j, s) = \{\vec{\mathbf{x}} \mid x_j > s\}$$

- 然后寻求最优切分变量 j 和最优切分点 s 。即求解：

$$(j^*, s^*) = \min_{j,s} \left[\min_{c_1} \sum_{\vec{\mathbf{x}}_i \in R_1(j,s)} (\tilde{y}_i - c_1)^2 + \min_{c_2} \sum_{\vec{\mathbf{x}}_i \in R_2(j,s)} (\tilde{y}_i - c_2)^2 \right]$$

其意义为：

- 首先假设已知切分变量 j ，则遍历最优切分点 s，则到：

$$\hat{c}_1 = \frac{1}{N_1} \sum_{\vec{\mathbf{x}}_i \in R_1(j,s)} \tilde{y}_i, \quad \hat{c}_2 = \frac{1}{N_2} \sum_{\vec{\mathbf{x}}_i \in R_2(j,s)} \tilde{y}_i$$

其中 N_1 和 N_2 分别代表区域 R_1 和 R_2 中的样本数量。

- 然后遍历所有的特征维度，对每个维度找到最优切分点。从这些 `(切分维度,最优切分点)` 中找到使得损失函数最小的那个。

4. 依次将输入空间划分为两个区域，然后重复对子区域划分，直到满足停止条件为止。这样的回归树称为最小二乘回归树。

5.1.1.2 生成算法

1. `CART` 回归树生成算法：
   - 输入：
     - 训练数据集 $\mathbb{D}$
     - 停止条件
   - 输出： `CART` 回归树 $f(\vec{\mathbf{x}})$
   - 步骤：
     - 选择最优切分维度 j 和切分点 s。

即求解：

$$(j^*,s^*)=\min_{j,s}\left[\min_{c_1}\sum_{\vec{\mathbf{x}}_i\in R_1(j,s)}(\tilde{y}_i-c_1)^2+\min_{c_2}\sum_{\vec{\mathbf{x}}_i\in R_2(j,s)}(\tilde{y}_i-c_2)^2\right]$$

- 用选定的 (j,s) 划分区域并决定响应的输出值：

$$R_1(j,s)=\{\vec{\mathbf{x}}\mid x_j\le s\},\quad R_2(j,s)=\{\vec{\mathbf{x}}\mid x_j>s\}$$
$$\hat{c}_1=\frac{1}{N_1}\sum_{\vec{\mathbf{x}}_i\in R_1(j,s)}\tilde{y}_i,\quad \hat{c}_2=\frac{1}{N_2}\sum_{\vec{\mathbf{x}}_i\in R_2(j,s)}\tilde{y}_i$$

  其中 N_1 和 N_2 分别代表区域 R_1 和 R_2 中的样本数量。
- 对子区域 R_1,R_2 递归地切分，直到满足停止条件。
- 最终将输入空间划分为 M 个区域 $R_1,R_2,\cdots,R_M$，生成决策树：
  $f(\vec{\mathbf{x}})=\sum_{m=1}^M\hat{c}_mI(\vec{\mathbf{x}}\in R_m)$ 。

5.1.2 CART 分类树

5.1.2.1 基尼系数

1. CART 分类树采用基尼指数选择最优特征。
2. 假设有 K 个分类，样本属于第 k 类的概率为 $p_k=p(y=c_k)$ 。则概率分布的基尼指数为：

$$Gini(p)=\sum_{k=1}^K p_k(1-p_k)=1-\sum_{k=1}^K p_k^2$$

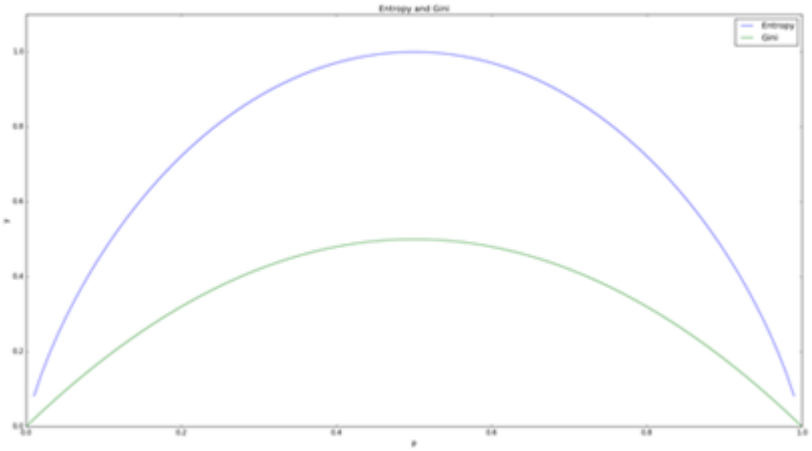
   基尼指数表示：样本集合中，随机选中一个样本，该样本被分错的概率。基尼指数越小，表示越不容易分错。

   > 样本被选错概率 = 样本被选中的概率 p_k * 样本被分错的概率 $1-p_k$ 。

3. 对于给定的样本集合 $\mathbb{D}$，设属于类 c_k 的样本子集为 $\mathbb{D}_k$，则样本集的基尼指数为：

$$Gini(\mathbb{D})=1-\sum_{k=1}^K(\frac{N_k}{N})^2$$

   其中 N 为样本总数， N_k 为子集 $\mathbb{D}_k$ 的样本数量。
4. 对于最简单的二项分布，设 $p(X=1)=p,P(X=0)=1-p$，则其基尼系数与熵的图形见下图。
   - 可以看到基尼系数与熵一样，也是度量不确定性的度量。
   - 对于样本集 $\mathbb{D}$， $Gini(\mathbb{D})$ 越小，说明集合中的样本越纯净。



5. 若样本集 $\mathbb{D}$ 根据特征 A 是否小于 a 而被分为两个子集: $\mathbb{D}_1$ 和 $\mathbb{D}_2$，其中：

$$\mathbb{D}_1 = \{(\vec{\mathbf{x}}, y) \in \mathbb{D} \mid x_A \leq a\}$$
$$\mathbb{D}_2 = \{(\vec{\mathbf{x}}, y) \in \mathbb{D} \mid x_A > a\} = \mathbb{D} - \mathbb{D}_1$$

则在特征 $A:a$ 的条件下，集合 $\mathbb{D}$ 的基尼指数为：

$$Gini(\mathbb{D}, A:a) = \frac{N_1}{N} Gini(\mathbb{D}_1) + \frac{N_2}{N} Gini(\mathbb{D}_2)$$

其中 N 为样本总数， N_1, N_2 分别子集 $\mathbb{D}_1, \mathbb{D}_2$ 的样本数量。它就是每个子集的基尼系数的加权和，权重是每个子集的大小（以子集占整体集合大小的百分比来表示）。

5.1.2.2 划分单元和划分点

1. 一棵 CART 分类树对应着输入空间的一个划分，以及在划分单元上的输出值。

   设输出 y 为分类的类别，是离散变量。训练数据集 $\mathbb{D} = \{(\vec{\mathbf{x}}_1, \tilde{y}_1), (\vec{\mathbf{x}}_2, \tilde{y}_2), \cdots, (\vec{\mathbf{x}}_N, \tilde{y}_N)\}$。

   设已经将输入空间划分为 M 个单元 $R_1, R_2, \cdots, R_M$，且在每个单元 R_m 上有一个固定的输出值 c_m。则 CART 分类树模型可以表示为：

$$f(\vec{\mathbf{x}}) = \sum_{m=1}^{M} c_m I(\vec{\mathbf{x}} \in R_m)$$

   其中 $I(\cdot)$ 为示性函数。

2. 如果已知输入空间的单元划分，基于分类误差最小的准则，则 CART 分类树在训练数据集上的损失函数为：

$$\sum_{m=1}^{M} \sum_{\vec{\mathbf{x}}_i \in R_m} I(\tilde{y}_i \neq c_m)$$

   根据损失函数最小，则可以求解出每个单元上的最优输出值 $\hat{c}_m$ 为： R_m 上所有输入样本 $\vec{\mathbf{x}}_i$ 对应的输出 $\tilde{y}_i$ 的众数。

   即： $\hat{c}_m = \arg\max_{c_m} \sum_{\vec{\mathbf{x}}_i \in R_m} I(c_m = \tilde{y}_i)$ 。

3. 问题是输入空间的单元划分是未知的。如何对输入空间进行划分？

   类似 CART 回归树， CART 分类树遍历所有可能的维度 j 和该维度所有可能的取值 s ，取使得基尼系数最小的那个维度 j 和切分点 s 。

   即求解： $(j^*, s^*) = \min_{j,s} Gini(\mathbb{D}, j:s)$ 。

5.1.2.3 生成算法

1. `CART` 分类树的生成算法：
   - 输入：
     - 训练数据集 $\mathbb{D}$
     - 停止计算条件
   - 输出： `CART` 决策树
   - 步骤：
     - 选择最优切分维度 j 和切分点 s。

       即求解：$(j^*,s^*)=\min_{j,s}Gini(\mathbb{D},A_j=s)$ 。

       它表示：遍历所有可能的维度 j 和该维度所有可能的取值 s ，取使得基尼系数最小的那个维度 j 和切分点 s 。
     - 用选定的 (j,s) 划分区域并决定响应的输出值：

       $$R_1(j,s)=\{\vec{\mathbf{x}}\mid x_j\le s\},\quad R_2(j,s)=\{\vec{\mathbf{x}}\mid x_j>s\}$$
       $$\hat{c}_1=\arg\max_{c_1}\sum_{\vec{\mathbf{x}}_i\in R_1}I(c_1=\tilde{y}_i),\quad \hat{c}_2=\arg\max_{c_2}\sum_{\vec{\mathbf{x}}_i\in R_2}I(c_m=\tilde{y}_i)$$

       其中 N_1 和 N_2 分别代表区域 R_1 和 R_2 中的样本数量。
     - 对子区域 R_1,R_2 递归地切分，直到满足停止条件。
     - 最终将输入空间划分为 M 个区域 $R_1,R_2,\cdots,R_M$，生成决策树：$f(\vec{\mathbf{x}})=\sum_{m=1}^M\hat{c}_mI(\vec{\mathbf{x}}\in R_m)$ 。

5.1.3 其它讨论

1. `CART` 分类树和 `CART` 回归树通常的停止条件为：
   - 结点中样本个数小于预定值，这表示树已经太复杂。
   - 样本集的损失函数或者基尼指数小于预定值，表示结点已经非常纯净。
   - 没有更多的特征可供切分。
2. 前面讨论的 `CART` 分类树和 `CART` 回归树都假设特征均为连续值。
   - 实际上 `CART` 树的特征可以为离散值，此时切分区域定义为：

     $$R_1(j,s)=\{\vec{\mathbf{x}}\mid x_j=s\}$$
     $$R_2(j,s)=\{\vec{\mathbf{x}}\mid x_j\ne s\}$$

   - 连续的特征也可以通过分桶来进行离散化，然后当作离散特征来处理。

5.2 CART 剪枝

1. `CART` 树的剪枝是从完全生长的 `CART` 树底端减去一些子树，使得 `CART` 树变小（即模型变简单），从而使得它对未知数据有更好的预测能力。

5.2.1 原理

1. 定义 `CART` 树 T 的损失函数为（$\alpha\ge 0$）：$C_\alpha(T)=C(T)+\alpha|T_f|$ 。其中：
   - $C_\alpha(T)$ 为参数是 α 时树 T 的整体损失。
   - $C(T)$ 为树 T 对训练数据的预测误差。

- $|T_f|$ 为子树的叶结点个数。

2. 对固定的 α ，存在使 $C_\alpha(T)$ 最小的子树，令其为 T_α^* 。可以证明 T_α^* 是唯一的。

   - 当 α 大时，T_α^* 偏小，即叶结点偏少。
   - 当 α 小时，T_α^* 偏大，即叶结点偏多。
   - 当 $\alpha=0$ 时，未剪枝的生成树就是最优的，此时不需要剪枝。
   - 当 $\alpha=\infty$ 时，根结点组成的一个单结点树就是最优的。此时剪枝到极致：只剩下一个结点。

3. 令从生成树 T_0 开始剪枝。对 T_0 任意非叶结点 t ，考虑：需不需要对 t 进行剪枝？

   - 以 t 为单结点树：因为此时只有一个叶结点，即为 t 本身，所以损失函数为：$C_\alpha(t)=C(t)+\alpha$ 。
   - 以 t 为根的子树 T_t：此时的损失函数为： $C_\alpha(T_t)=C(T_t)+\alpha|T_t|$ 。

4. 可以证明：

   - 当 $\alpha=0$ 及充分小时，有 $C_\alpha(T_t)<C_\alpha(t)$。即此时倾向于选择比较复杂的 T_t，因为正则化项的系数 α 太小。
   - 当 α 增大到某个值时，有 $C_\alpha(T_t)=C_\alpha(t)$ 。
   - 当 α 再增大时，有 $C_\alpha(T_t)>C_\alpha(t)$ 。即此时倾向于选择比较简单的 t，因为正则化项的系数 α 太大。

5. 令 $\alpha=\frac{C(t)-C(T_t)}{|T_t|-1}$ ，此时 T_t 与 t 有相同的损失函数值，但是 t 的结点更少。

   因此 t 比 T_t 更可取，于是对 T_t 进行剪枝。

6. 对 T_0 内部的每一个内部结点 t ，计算 $g(t)=\frac{C(t)-C(T_t)}{|T_t|-1}$ 。

   它表示剪枝后整体损失函数增加的程度（可以为正，可以为负）。则有：

$$\begin{aligned}C_\alpha(t)-C_\alpha(T_t)&=C(t)+\alpha-C(T_t)-\alpha|T_t|\\&=C(t)-C(T_t)-\alpha(|T_t|-1)\\&=(g(t)-\alpha)(|T_t|-1)\end{aligned}$$

   因为 t 是个内部结点，所以 $|T_t|>1$，因此有：

   - $g(t)>\alpha$ 时，$C_\alpha(t)-C_\alpha(T_t)>0$ ，表示剪枝后，损失函数增加。
   - $g(t)=\alpha$ 时，$C_\alpha(t)-C_\alpha(T_t)=0$ ，表示剪枝后，损失函数不变。
   - $g(t)<\alpha$ 时，$C_\alpha(t)-C_\alpha(T_t)<0$ ，表示剪枝后，损失函数减少。

7. 对 T_0 内部的每一个内部结点 t ，计算最小的 g ：

$$g*=\min g(t),t\in T_0 \text{ and } t \text{ is not a leaf}$$

   设 g^* 对应的内部结点为 t^*，在 T_0 内减去 t^* ，得到的子树作为 T_1。

   令 $\alpha_1=g*$ ，对于 $t\neq t^*$，有：$g(t)>g(t^*)=\alpha_1,\ t\in T_0 \text{ and } t \text{ is not a leaf and } t\neq t^*$ 。

   对于 $\alpha>\alpha_1$，有：

   - 对于 t^* 剪枝，得到的子树的损失函数一定是减少的。

     它也是所有内部结点剪枝结果中，减少的最多的。因此 T_1 是 $\alpha\in[\alpha_1,)$ 内的最优子树。

   - 对任意一个非 t^* 内部结点的剪枝，得到的子树的损失函数有可能是增加的，也可能是减少的。

     如果损失函数是减少的，它也没有 T_1 减少的多。

8. 如此剪枝下去，直到根结点被剪枝。

   - 此过程中不断产生 α_i 的值，产生新区间 $[\alpha_1,\alpha_2),[\alpha_2,\alpha_3),\cdots$
   - 此过程中不断产生 最优子树 $T_1,T_2,\cdots$
   - 其中 T_1 是由 T_0 产生的、$\alpha\in[\alpha_1,\alpha_2)$ 内的最优子树； T_2 是由 T_1 产生的、$\alpha\in[\alpha_2,\alpha_3)$ 内的最优子树；...

9. 上述剪枝的思想就是用递归的方法对树进行剪枝：计算出一个序列 $0=\alpha_0<\alpha_1<\cdots<\alpha_n<\infty$，同时剪枝得到一系列最优子树序列 $\{T_0,T_1,\cdots,T_n\}$。其中 T_i 是 $\alpha\in[\alpha_i,\alpha_i+1)$ 时的最优子树。
10. 上述剪枝的结果只是对于训练集的损失函数较小。
    - 需要用交叉验证的方法在验证集上对子树序列进行测试，挑选中出最优子树。

      交叉验证的本质就是为了挑选超参数 α 。
    - 验证过程：用独立的验证数据集来测试子树序列 $\{T_0,T_1,\cdots,T_n\}$ 中各子树的平方误差或者基尼指数。

      由于 $\{T_1,\cdots,T_n\}$ 对应于一个参数序列 $\{\alpha_1,\alpha_2,\cdots,\alpha_n\}$，因此当最优子树 T_k 确定时，对应的区间 $[\alpha_k,\alpha_{k+1})$ 也确定了。

5.2.2 算法

1. `CART` 剪枝由两步组成：
   - 从生成算法产生的决策树 T_0 底端开始不断的剪枝：
     - 每剪枝一次生成一个决策树 $T_i,i=1,2...$
     - 这一过程直到 T_0 的根结点，形成一个子树序列 $\{T_0,T_1,\cdots,T_n\}$ 。
   - 用交叉验证的方法在独立的验证集上对子树序列进行测试，挑选中出最优子树。
2. `CART` 剪枝算法：
   - 输入： `CART` 算法生成的决策树 T_0
   - 输出： 最优决策树 T^*
   - 算法步骤：
     - 初始化： $k=0,T=T_0,\alpha=\infty$
     - 自下而上的对各内部结点 t 计算： $C(T_t),\ |T_t|,\ g(t)=\frac{C(t)-C(T_t)}{|T_t|-1},\ \alpha=\min(\alpha,g(t))$ 。
     - 自下而上地访问内部结点 t： 若有$g(t)=\alpha$,则进行剪枝，并确定叶结点 t 的输出，得到树 T 。
       - 如果为分类树，则叶结点 t 的输出采取多数表决法：结点 t 内所有样本的标记的众数。
       - 如果为回归树，则叶结点 t 的输出为平均法：结点 t 内所有样本的标记的均值。
     - 令 $k=k+1,\alpha_k=\alpha,T_k=T$ 。
     - 若 T 不是由根结点单独构成的树，则继续前面的步骤。
     - 采用交叉验证法在子树序列 $T_0,T_1,\cdots,T_n$ 中选取最优子树 T^* 。
3. `CART` 剪枝算法的优点是：不显式需要指定正则化系数 α 。
   - `CART` 剪枝算法自动生成了一系列良好的超参数 $\alpha_1,\alpha_2,\cdots$ ，然后利用验证集进行超参数选择。
   - 虽然传统剪枝算法也可以用验证集来进行超参数选择，但是`CART` 剪枝算法的效率更高。

     因为`CART` 剪枝算法只需要搜索超参数 α 的有限数量的区间即可，而传统剪枝算法需要搜索整个数域 $[0,\infty)$ 。

六、连续值、缺失值处理

6.1 连续值

1. 现实学习任务中常常会遇到连续属性，此时可以使用连续属性离散化技术将连续特征转换为离散特征。
2. 最常用的离散化技术为二分法`bi-partition` ：

给定样本集 $\mathbb{D}$ 和连续属性 A，假设该属性在 $\mathbb{D}$ 中出现了 M 个不同的取值。

- 将这些值从小到大进行排列，记作 $a_1, a_2, \cdots, a_M$。
- 选取 $M-1$ 个划分点，依次为：$\frac{a1+a2}{2}, \frac{a2+a3}{2}, \cdots, \frac{a_{M-1}+a_M}{2}$ 。
- 然后就可以像离散属性值一样来考察这些划分点，选取最优的划分点进行样本集和的划分。

  这也是 `C4.5` 算法采取的方案。

3. 事实上划分点的数量可以是任意正整数，划分点的位置也可以采取其它的形式。

   事实上，划分点的数量、划分点的位置都是超参数，需要结合验证集、具体问题来具体分析。

6.2 缺失值

1. 现实任务中经常会遇到不完整样本，即样本的某些属性值缺失。

   如果简单地放弃不完整样本，仅使用无缺失值的样本来进行学习，则是对数据信息的极大浪费。

2. 这里有两个问题：
   - 如何在属性值缺失的情况下选择划分属性？
   - 给定划分属性，如果样本在该属性上的值缺失，则如何划分样本？

6.2.1 划分属性选择

1. 给定训练集 $\mathbb{D}$ 和属性 A， 令 $\tilde{\mathbb{D}}$ 表示 $\mathbb{D}$ 中在属性 A 上没有缺失的样本子集。则可以仅根据 $\tilde{\mathbb{D}}$ 来判断属性 A 的优劣 。

2. 假定属性 A 有 n 个可能的取值 $a_1, a_2, \cdots, a_n$。令：
   - $\tilde{\mathbb{D}}^i$ 表示： $\tilde{\mathbb{D}}$ 中在属性 A 上取值为 a_i 的样本的子集
   - $\tilde{\mathbb{D}}_k$ 表示： $\tilde{\mathbb{D}}$ 中属于第 k 类的样本子集（一共有 K 个分类）。

     > 根据定义有：$\tilde{\mathbb{D}}=\bigcup_{k=1}^{K}\tilde{\mathbb{D}}_k=\bigcup_{i=1}^{n}\tilde{\mathbb{D}}^i$

3. 为每个样本 $\vec{\mathbf{x}}$ 赋予一个权重 $w_{\vec{\mathbf{x}}}$，定义：

$$\rho=\frac{\sum_{\vec{\mathbf{x}}\in\tilde{\mathbb{D}}}w_{\vec{\mathbf{x}}}}{\sum_{\vec{\mathbf{x}}\in\mathbb{D}}w_{\vec{\mathbf{x}}}},\ \tilde{p}_k=\frac{\sum_{\vec{\mathbf{x}}\in\tilde{\mathbb{D}}_k}w_{\vec{\mathbf{x}}}}{\sum_{\vec{\mathbf{x}}\in\tilde{\mathbb{D}}}w_{\vec{\mathbf{x}}}},\ \tilde{r}_i=\frac{\sum_{\vec{\mathbf{x}}\in\tilde{\mathbb{D}}^i}w_{\vec{\mathbf{x}}}}{\sum_{\vec{\mathbf{x}}\in\tilde{\mathbb{D}}}w_{\vec{\mathbf{x}}}}$$
$$k=1,2,\cdots,K;\quad i=1,2,\cdots,n$$

   其物理意义为：
   - ρ 表示：无缺失值样本占总体样本的比例。
   - $\tilde{p}_k$ 表示：无缺失值样本中，第 k 类所占的比例。
   - $\tilde{r}_i$ 表示：无缺失值样本中，在属性 A 上取值为 a_i 的样本所占的比例。

4. 将信息增益的计算公式推广为：$g(\mathbb{D},A)=\rho\times G(\tilde{\mathbb{D}},A)=\rho\times\left(H(\tilde{\mathbb{D}})-\sum_{i=1}^{n}\tilde{r}_iH(\tilde{\mathbb{D}}^i\mid A)\right)$ 。

   其中：$H(\tilde{\mathbb{D}})=-\sum_{k=1}^{K}\tilde{p}_k\log\tilde{p}_k$ 。

6.2.2 样本划分

1. 基于权重的样本划分：
   - 如果样本 $\vec{\mathbf{x}}$ 在划分属性 A 上的取值已知，则：
     - 将 $\vec{\mathbf{x}}$ 划入与其对应的子结点。
     - $\vec{\mathbf{x}}$ 的权值在子结点中保持为 $w_{\vec{\mathbf{x}}}$ 。

- 如果样本 $\vec{\mathbf{x}}$ 在划分属性 A 上的取值未知，则：
  - 将 $\vec{\mathbf{x}}$ 同时划入所有的子结点。
  - $\vec{\mathbf{x}}$ 的权值在所有子结点中进行调整：在属性值为 a_i 对应的子结点中，该样本的权值调整为 $\tilde{r}_i \times w_{\vec{\mathbf{x}}}$ 。

2. 直观地看，基于权重的样本划分就是让同一个样本以不同的概率分散到不同的子结点中去。
   - 这一做法依赖于：每个样本拥有一个权重，然后权重在子结点中重新分配。
   - `C4.5` 使用了该方案。

七、多变量决策树

1. 由于决策树使用平行于坐标轴的拆分，使得它对于一些很简单的问题很费力。

   比如：当 $x_2 > x_1$ 时为正类；否则为反类。这种拆分边界并不平行于坐标轴，使得决策树会用许多层的拆分来逼近这个边界。

2. 解决方案是：多变量决策树 `multivariate decision tree` 。
   - 传统的单变量决策树的分类边界每一段是与坐标轴平行的，每一段划分都直接对应了某个属性的取值。
   - 多变量决策树的分类边界可以为斜线，它可以大大简化了决策树的模型。
3. 多变量决策树中，内部结点不再是针对某个属性，而是对属性的线性组合。即：每个内部结点是一个 $\sum_{i=1}^{d} w_i x_i = t$ 的线性分类器。其中：
   - w_i 是属性 x_i 的权重。
   - d 为变量的数量。
   - t 表示这些变量的约束。
4. 与传统的单变量决策树不同，在多变量决策树的学习过程中，不是为每内部结点寻找一个最优划分属性，而是试图建立一个合适的线性分类器。