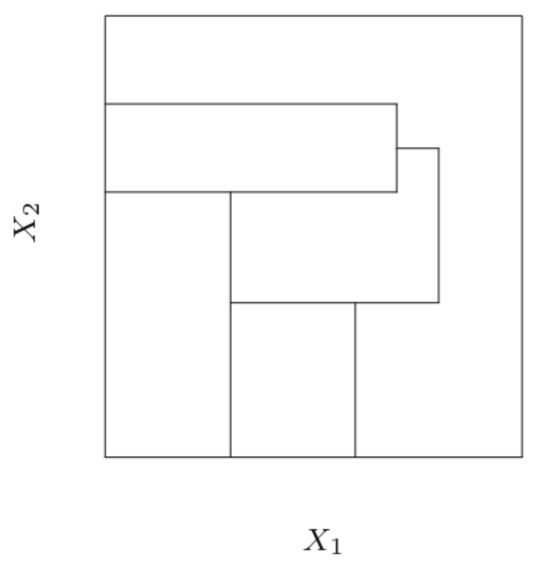
Decision Trees

Decision tress can be applied to both **regression** and **classification** problems, which makes it very flexible and interpretable.

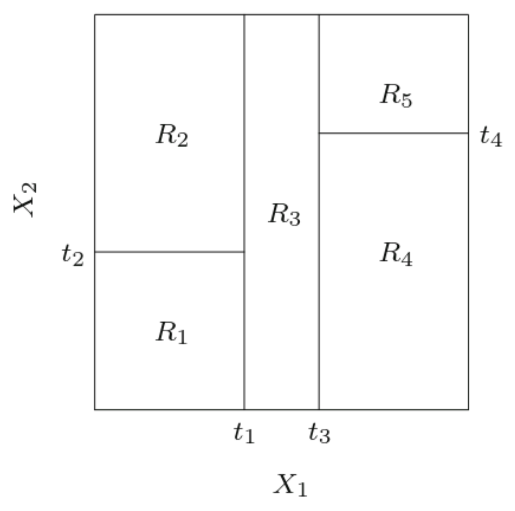
CART

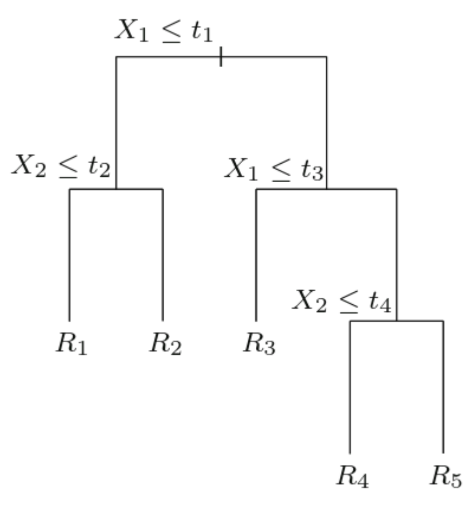
Stands for **Classification and Regression Trees** which is an (algorithm/method) for tree based methods.

Considering a response Y and predictor X_1, X_2



By restricting it to **recursive binary partitions** by splitting the region into two and model the response by the mean of Y





Regression Trees

To construct a **regression tress**, the algorithm needs to automatically decide on splitting variables(feature) and splitting point s also what **shape** the tree should have.

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

- This models the response as a constant c_m in each region R_m.
  Using the Residual Sum of Squares :

$$\sum_{i=1} (y_i - \sum^{M} c_m I(x \in R_m))^2$$

  For one **Region** we get :

$$\mathcal{L}(c) = \sum^{n} (y_i - c)^2$$

  Deriving w.r.t. c :

$$\frac{d\,\mathcal{L}}{d\,c} = \sum^{n} 2(c - y_i)$$

  Setting it to zero results in:

$$\hat{c} = \frac{1}{N_m} \sum_{i=1}^{n} y_i \equiv \hat{c}_m = \text{ave}(y_i | x_i \in R_m)$$

  Note:

- $\sum_{i \in R_m} c = N_m . c$
- The constant $\hat{c}$ represent the mean of $\bar{y}$ on that region m
- $\text{avg}(y_i | x_i \in R_m)$ means the average of y_i given that x_i is in the region m

To find the best binary partition in terms of minimum sum of squares is computationally infeasible. Hence **regression trees** use a greedy algorithm.
At a given node we consider all possible splits (j, s) by :

- Consider all features p of X given by X_j
  - For every possible threshold s
  - We evaluate the $R_1(j, s) = \{X | X_j \le s\}$ and $R_2(j, s) = X | X_j > s$
- Repeat for each **feature** resulting in pairs $(j, s) \to (\text{feature,split point})$
  ex : $(age, 50), (age, 40), (height, 170), (height, 167) \ldots$

Then we seek the **splitting variable** j and **split point** s that (minimize/solves) this :

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

The inner minimization is solved by:

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s))$$

The greedy algorithm Summary :

- Select splits which are pairs (j, s)
- Calculate the constant for the split c_1, c_2
- Evaluate the split by calculating the Residual Sum of Squares
- Choose the split (j, s) that yields the smallest RSS

The question now is how large should we grow the tree? , very large tree might overfit the data easily while small might not learn the data and the underline structure.

Tree Pruning

The size of tree is a **tuning parameter** which correspond to the model complexity, the **greedy** strategy is to grow a large tree T_0 and then stop growing after the minimum **node size** is reached(4 observations per region).

The large tree T_0 is pruned using $\text{cost-complexity prunning}$:

- The number of Observations in a region m is denoted :

$$N_m = \#\{x_i \in R_m\},$$

- The constant for region m is denoted :

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

- The **MSE** for region m is denoted(**impurity measure**) :

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$$

The **Cost complexity criterion** :

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

With :

- $|T|$ the number of terminal nodes(leafs)
- α penalty parameter

The idea is to find for each α a subtree $T_\alpha \subseteq T_0$ that minimize $C_\alpha(T)$,$\alpha \geq 0$ results in a tradeoff between the tree size and it's **goodness of fit**.

- Large values results in smaller trees T_α
- $\alpha = 0$ results in T_0

Weakest Link Pruning

To find T_α that minimize $C_\alpha(T)$ we use **weakest link pruning**:

First the starting point for the pruning is not T_0 , but rather $T_1 = T(0)$ which is the smallest subtree of T_0 that satisfy:

$$R(T_1) = R(T_0)$$

With $R(T) = \sum^{|T|} N_m Q_m(T)$

To Obtain T_1 First, we look at T_0 the biggest tree and for any **two terminal nodes(leafs)** from the same parent node, if we sum the error rate and it's the same as their parent node, we prune off these two terminal nodes :

$$R(t) = T(t_L) + R(t_R)$$

- Parent node t
- Two terminal nodes t_L and t_R

This process is applied recursively. which results in a pruned T_0 while having the same error rate.

```
T_0:          T₁ (after pruning):
    A              A
   / \            / \
  B   C          B   C
 / \
D   E
```

- Since $R(B) = R(D) + R(E)$
- Making a prediction using the region B or in D and E will results in the same error rate
- The **child nodes** doesn't provide any improvements over their **parents**

The **weakest link** method not only find the next α which results in different optimal subtree, but find that optimal subtree.

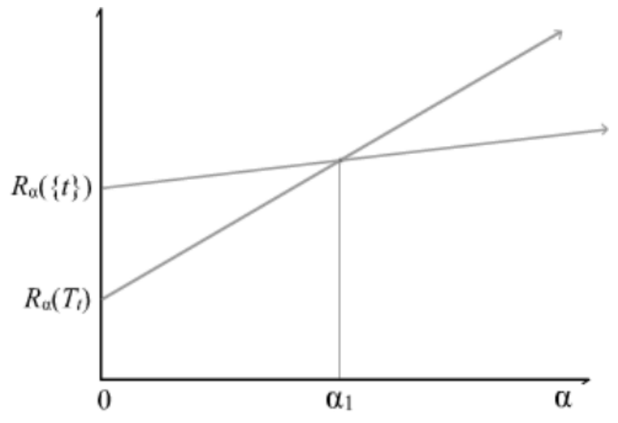
Let $t \in T_1$ is any node $\rightarrow R_\alpha(t) = R(t) + \alpha$.
and T_t any branch $\rightarrow R_\alpha(T_t) = R(T_t) + \alpha|T_t|$.
For more details on these formulas check Weakest link lagrangian derivation

When $\alpha = 0$:

$$R_0(T_t) < R_0(t)$$

- That is the **penalty error rate** of the node is bigger than it's branch

Increasing α leads to a faster increase in $R_\alpha(T_t)$ since it's $\alpha|T_t|$ at a certain α_1 we will have $R_{\alpha_1}(T_t) = R_{\alpha_1}(t)$.



Solving the inequality $R_\alpha(T_t) < R_\alpha(t)$:

$$\alpha < \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}$$

- The numerator is the increase in **error rate** if we prune
- The denominator is the number of leaves removed

$$g_1(t) \begin{cases} \frac{R(t)-R(T_t)}{|\tilde{T}_t|-1}, & t \notin \tilde{T}_1 \\ +\infty, & t \in \tilde{T}_1 \end{cases}$$

With $\tilde{T}$ is set of **terminal nodes**

- If $t \in \tilde{T}_1$ means t is a **leaf/terminal node**, that's why we set it to $+\infty$ to exclude it
  The **weakest link** t^* in T_1 achieves the minimum of $g_1(t)$

  $$g_1(t^*) = \min_{t \in T_1} g_1(t)$$

  put $\alpha_2 = g_1(t^*)$, to get the optimal subtree corresponding to α_2 and removing the branch growing out of t^* since it increase the error rate the least if removed , keep in mind there can be several nodes that reach or achieve the minimum of $g_1(t)$

Steps Summary :

- For T_1 Tree compute for every internal node $t \in T$, $g(t)$ interpreted as the increase in the error rate if that node t is pruned
- Find $t^* = \min g_1(t)$, Let $\alpha^* = g(t^*)$ so that the next subtree is smaller
- Prune by replacing the subtree T_{t^*} by a single leaf t^* resulting in a new tree T_k
- Save the pair(α_k, T_k), and set T_k as the main tree and repeat it
  Resulting in guaranteed nested sub trees :

  $$T_1 \supset T_2 \supset \cdots \supset T_k$$

  With :

  $$\alpha_1 < \alpha_2 < \cdots < \alpha_{k+1}$$

Intuition :

The weakest link pruning **iteratively** removes internal nodes(non-terminal) whose pruning causes the **smallest increase in error rate**. Simply :

- Remove the nodes to reduce complexity but only the one that effort the error rate the least $\min_{t\in T_1} g_i(t)$
- α is the threshold computed to decide on the optimal prune using $g(t)$
- Running **weakest link** to completion will results in the **root** node only, that's why the results is guaranteed **nested sub trees**
- **Weakest link** pruning main goal is the reduce complexity

Classification Trees

A **classification tree** is very similar to regression tree, except that is used to predict **qualitative** response , for regression tree the predicted response for an observation is given by the mean response $\hat{c}$ of the training observations.

For a **classification tree** we predict each observation belongs to the **most common occurring class** of training observations in the region m.
To interpret the results of a classification tree we are often interested not only in class prediction on node/region, but also in the class **proportions** among the training observations.

$$\hat{p}_{mk} = \frac{1}{N_m}\sum_{x_i\in R_m} I(y_i = k)$$

- $I(y_i = k)$ is indication function returns 1 if the condition is met , 0 otherwise
  To perform the binary split so we grow the classification tree we use **Missclassification error**

  $$\text{Missclassificaion error} = \frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$$

  With :

  $$k(m) = \arg\max_k(\hat{p}_{mk})$$

- Representing the majority class in node m

A simpler form of **Missclassification error**

$$E = 1 - \max_k(\hat{p}_{mk})$$

The problem with missclassification error is not **sufficiently sensitive for tree-growing**, for example :

- **Node** X has 400 **observation** from class A and 380 from class B

  $$E = 1 - \max(\hat{p}_A, \hat{p}_B)$$

  $$\hat{p}_A = \frac{400}{780} = 0.51$$
  $$\hat{p}_A = \frac{380}{780} = 0.49$$
  $$\text{Results in: } E = 1 - \max(0.51, 0.49)$$
  $$E = 1 - 0.51 = 0.49$$

- **Node** Y had 700 **observation** from class A and 80 from class B

  $$E = 1 - \max(0.89, 0.11)$$

  $$E = 1 - 0.89 = 0.11$$

  For missclassification both node X and node Y are the same and both have class A **majority**, not taking into **consideration** that node Y is more pure and that the probabilities in node X are closer and almost the same, that's what not **sufficiently sensitive** means.

In practice **Gini index** and **Cross-entropy** are more preferable :

Gini index

The $Gini\ index$ is defined by :

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Or the **Computationally efficient** form :

$$G = 1 - \sum_{k=1}^{K} (\hat{p}_{mk})^2$$

It's measure of total **variance** across the K classes, takes values between $[0, 1]$.

- Gini index of 1 represent impure region/dataset
- While Gini index of 0 represent pure dataset

Calculating the **Gini index** for each feature, help us decide on which feature to pick as the root node

- Fast computation than both $entropy$ and $missclassification\ error$
- Create splits quickly
- Efficient for large high-dimensional datasets

Cross-entropy/deviance

The $Cross-entropy$ is defined by :

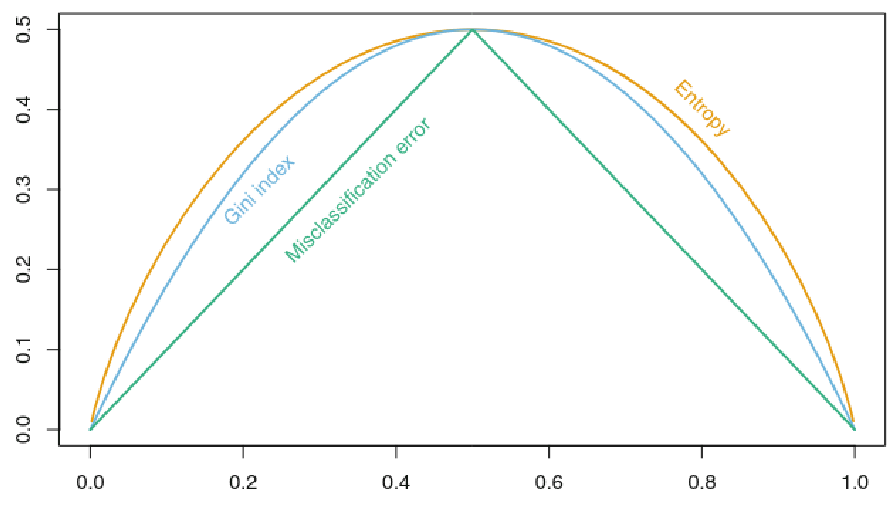
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Entropy measures uncertainty in a node's class distribution, derived from **information gain** lower entropy indicates that the node m is pure.

- Sensitive to probability changes
- Produce more balanced nodes partitions
- Suited for more balanced datasets

Use Cases

Split Evaluation :
Both of **entropy** and **gini index** are numerically similar, they are used to evaluate the quality of a particular split since they are sensitive to **node purity** more than **missclassification error**.

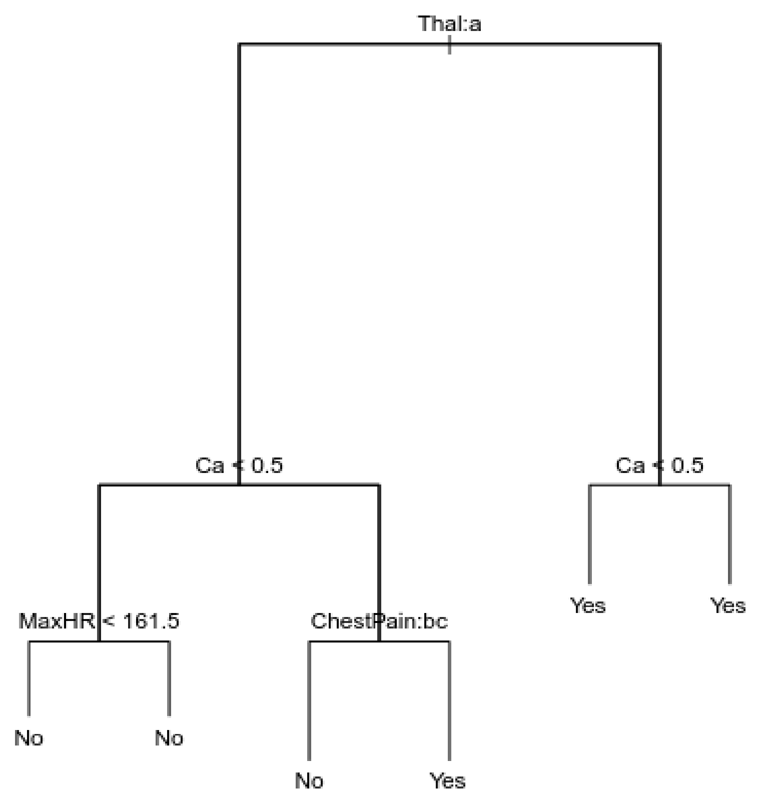


Why node purity important ?

an important aspect in **Decision Tress** is that the leaves should represent regions with strong **confident predictions**, and that is determined by **node purity**.

- The higher the purity $\rightarrow$ the node gives you confident predictions

- The lower the purity $\rightarrow$ the node is uncertain even tho it gives the same results
  for example :



- The right hand node `Ca<0.5` both of it's leaves are yes , which seems useless why not prune them
- The **right hand leaf** is more certain and confident on it's prediction than the **left hand leaf** which is less certain and *probably yes*
- Splitting `Ca<0.5` does not reduce the **missclassification error** but improves **gini index** and **entropy** which are more sensitive to node purity

More explanation :
Say we have a node/region with :

- $Yes \rightarrow 80\%$
- $No \rightarrow 20\%$
  After the split :
- $Right\ Leaf \rightarrow Yes\ 90\%$, $No\ 10\% \implies$ very pure , results in a Yes
- $Left\ Leaf \rightarrow Yes\ 55\%$, $No\ 45\% \implies$ still mixed and uncertain but still results in a Yes
  For **missclassification** both leaves are the same, but for **gini index/entropy** the right leaf is more certain and can be trusted.

For more details and considerations when using **Decision Trees** Read :

- Other Considerations on Decision Trees
- Trees Versus Linear Models
  For more advance **Decision Trees** see:
- Random Forests
- Boosting
- Bayesian Additive Regression Tress