Data Mining
Classification Trees (2)

Ad Feelders

Universiteit Utrecht

Basic Tree Construction Algorithm (control flow)

Construct tree

nodelist $\leftarrow$ {{training data}}

Repeat

current node $\leftarrow$ select node from nodelist

nodelist $\leftarrow$ nodelist $-$ current node

if impurity(current node) > 0 impurity > 0，表示還有混合類別

then

S $\leftarrow$ set of candidate splits in current node 找出這個節點所有可能的切分方式（候選 splits）

s* $\leftarrow$ arg max$_{s \in S}$ impurity reduction(s,current node)

child nodes $\leftarrow$ apply(s*,current node) 用這個最佳分裂 s* 把 current node 切成左右子節點

nodelist $\leftarrow$ nodelist $\cup$ child nodes 把新產生的子節點加入 nodelist，等之後再處理

fi (end if)

Until nodelist $= \varnothing$

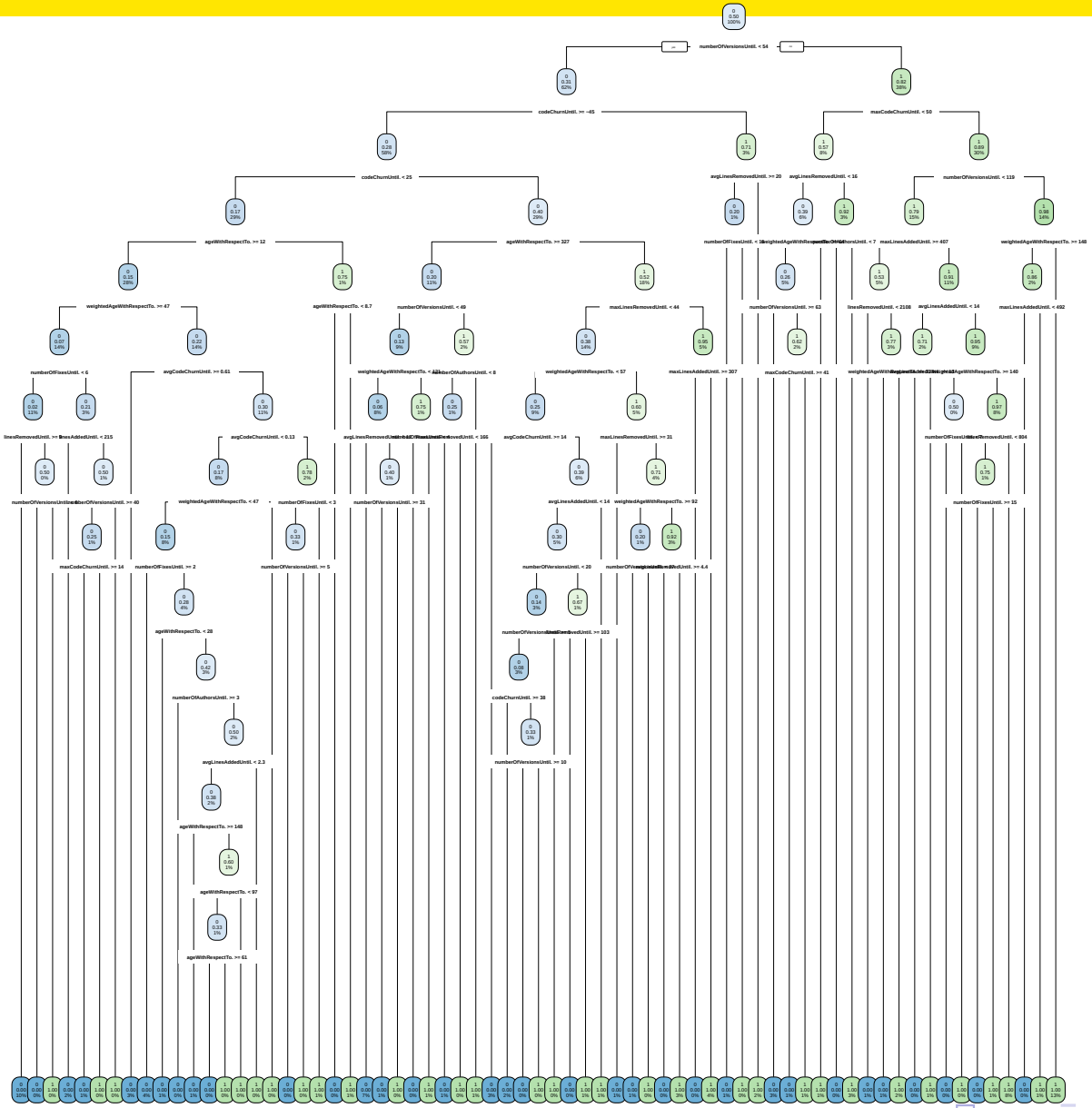
Overfitting and Pruning

- The tree growing algorithm continues splitting until all leaf nodes contain examples of a single class.
- This results in a tree with zero resubstitution error. i(t) = 1 – maxj p(jlt)
- Is this a good tree for predicting the class of new examples?
- Not unless the problem is truly "deterministic"!
- Problem of *overfitting*.

決定性

deterministic: 資料沒有雜訊，每個特徵組合都唯一對應一個類別

An Overfitted Tree on Bug Prediction Data



Proposed Solutions

How can we prevent overfitting?

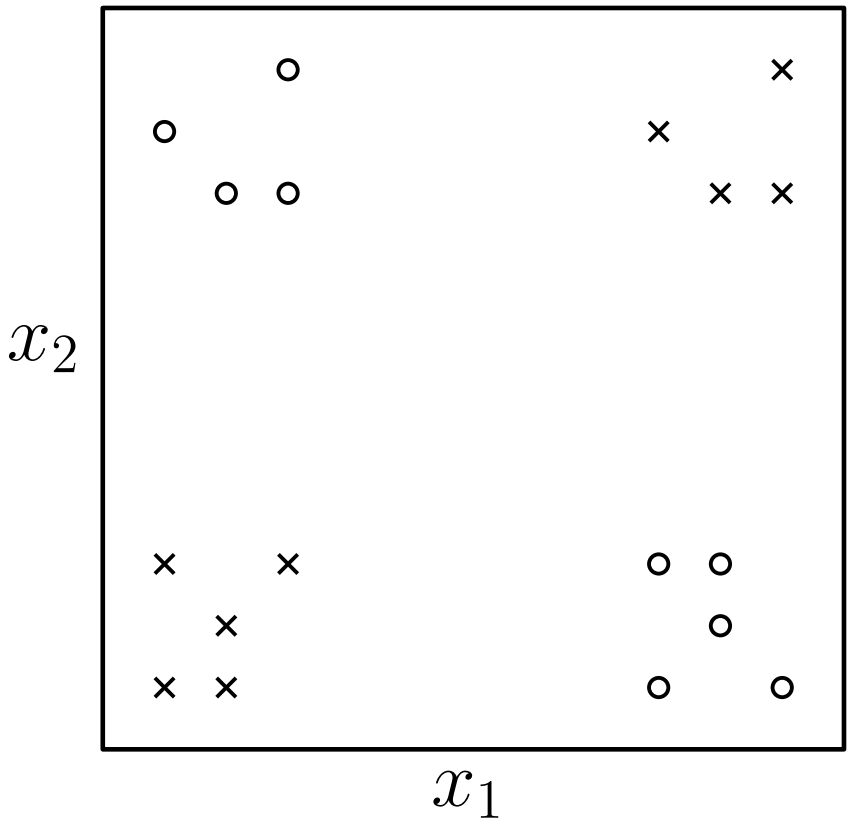
impurity reduction: $\Delta i(s,t) = i(t) - \{\pi(\ell) i(\ell) + \pi(r) i(r)\}$

- Stopping rules, e.g. don't expand a node if:
  - the impurity reduction of the best split is below some threshold, or
  - the number of training examples falling into that node is too small.
- Pruning: grow a very large tree T_{max} and merge back nodes.

Stopping Rules

Disadvantage: sometimes you first have to make a weak split to be able to follow up with a good split.

Since we only look one step ahead we may miss the good follow-up split.



Pruning

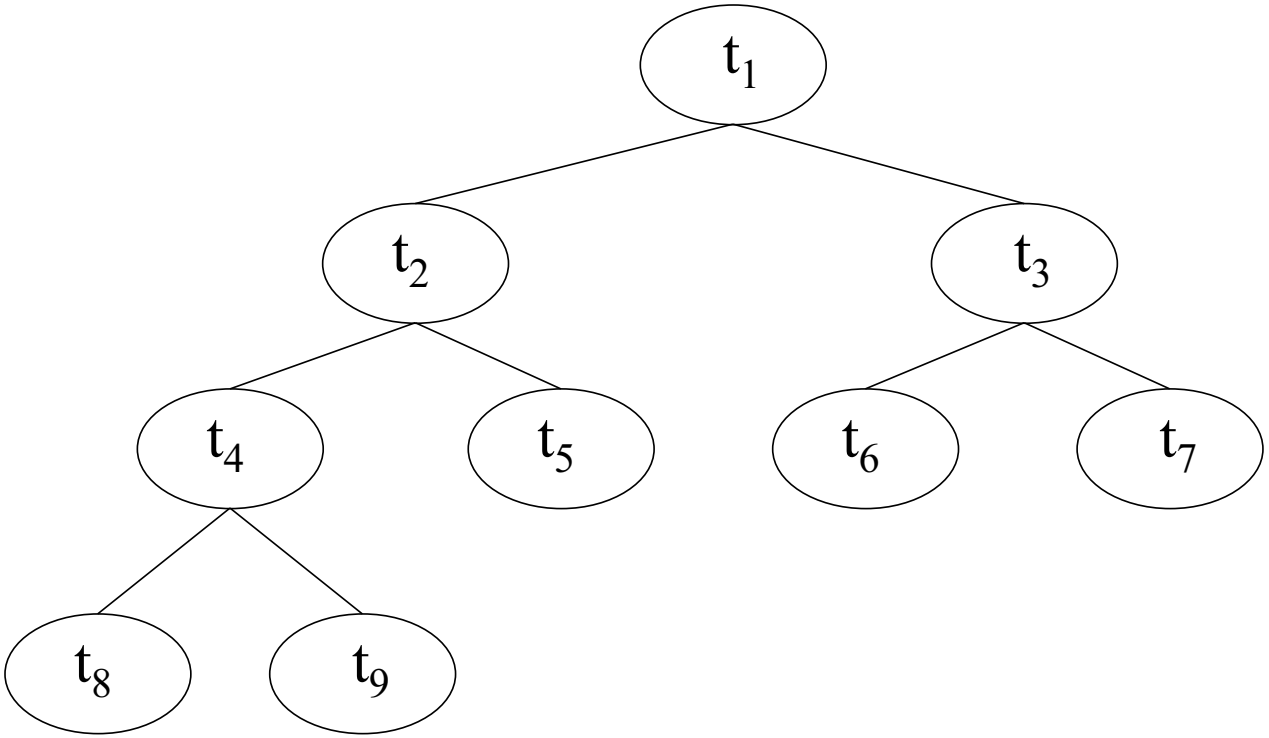
- To avoid the problem of stopping rules, we first grow a very large tree $T_{\max}$ on the training sample, and then *prune* this large tree.
- Objective: select the pruned subtree that has lowest *true* error rate.
- Problem: how to find this pruned subtree?
- Cost-complexity pruning (Breiman et al.; CART), also called *weakest link* pruning.

P. 12

Note: in the practical assignment we don't use pruning, but we use a stopping rule based on the `nmin` and `minleaf` parameters.

true error rate (真實錯誤率)
指模型在「新的、未見過的資料」上的錯誤率。
不只是訓練集的錯誤率（因為那會低估）。
實際上我們用 測試集 或 交叉驗證 (cross-validation) 來近似這個 true error rate。

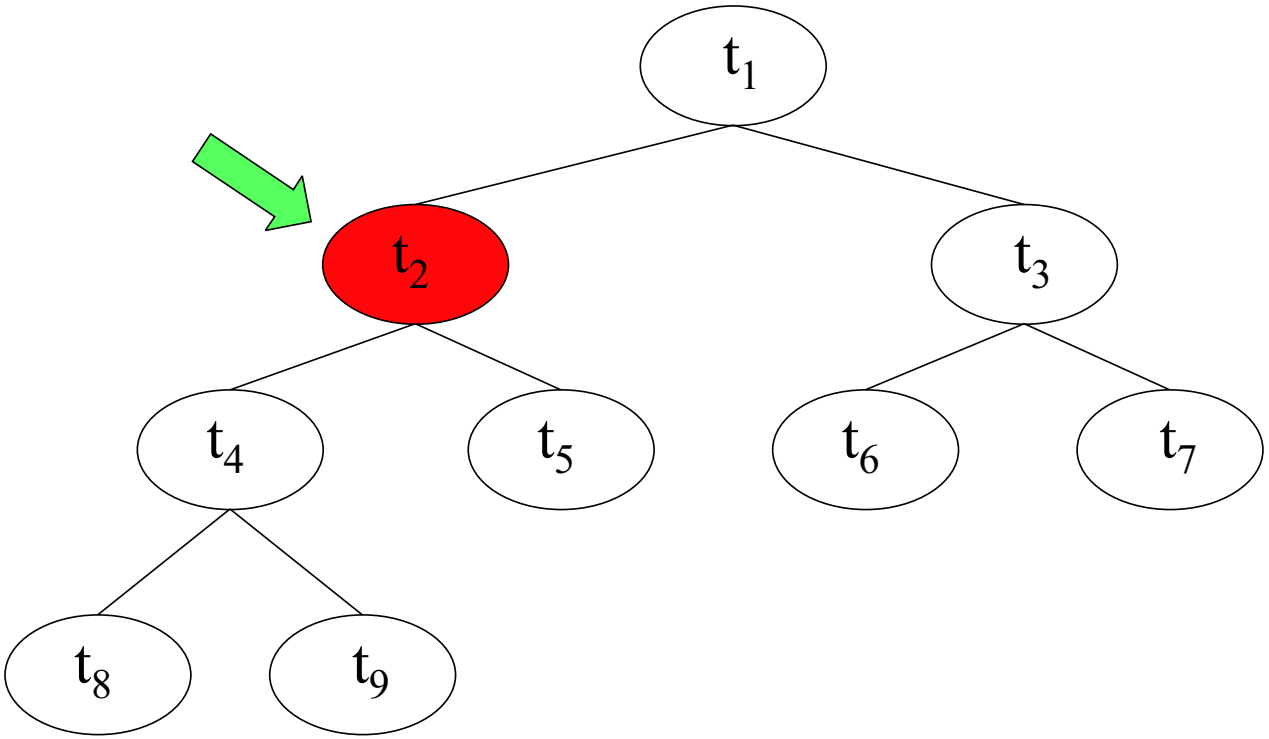
Terminology: Tree T



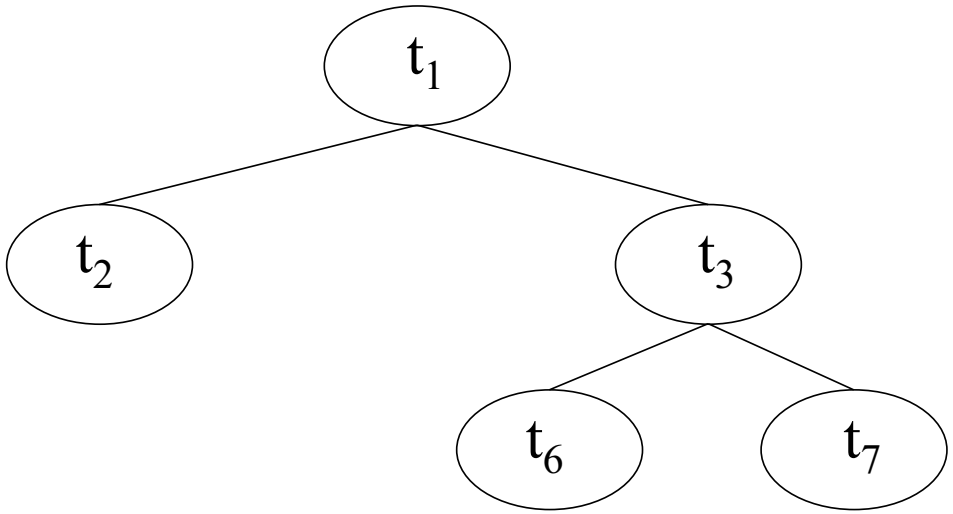
$\tilde{T}$ denotes the collection of leaf nodes of tree T.

$\tilde{T} \overset{\text{表示}}{=} \{t_5, t_6, t_7, t_8, t_9\}, |\tilde{T}| = 5$

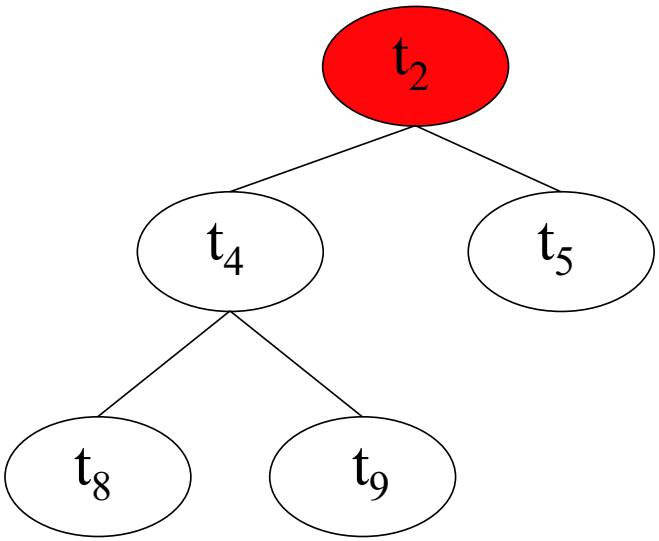
Terminology: Pruning T in node t_2



Terminology: T after pruning in t_2: $T - T_{t_2}$



Terminology: Branch T_{t_2}



$\tilde{T}_{t_2} = \{t_5, t_8, t_9\}, |\tilde{T}_{t_2}| = 3$

Cost-complexity pruning

- A pruned subtree of T is a tree obtained by pruning T in 0 or more nodes.
- The total number of pruned subtrees of a balanced binary tree with ℓ leaf nodes is

$$\lfloor 1.5028369^{\ell} \rfloor$$

- With just 40 leaf nodes we have approximately 12 million pruned subtrees.
- Exhaustive search not recommended.
- Basic idea of cost-complexity pruning: reduce the number of pruned subtrees we have to consider by selecting the ones that are the "best of their kind" (in a sense to be defined shortly...)

Total cost of a tree

Strike a balance between fit and complexity. Total cost $C_\alpha(T)$ of tree T

$$C_\alpha(T) = R(T) + \alpha|\tilde{T}|$$

$\tilde{T}$: the collection of leaf nodes of tree T

Total cost consists of two components:

- resubstitution error $R(T)$, and
- a penalty for the complexity of the tree $\alpha|\tilde{T}|, (\alpha \geq 0)$.

Note: $R(T) = \dfrac{\text{number of wrong classifications made by } T}{\text{number of examples in the training sample}}$

Tree with lowest total cost

- Depending on the value of α, different pruned subtrees will have the lowest total cost.
- For $\alpha = 0$ (no complexity penalty) the tree with smallest resubstitution error wins.
- For higher values of α, a less complex tree that makes a few more errors might win.

nested sequence：意思是嵌套序列，也就是每個剪枝後的子樹都是前一個子樹的子集。
例如：T0 ⊃ T1 ⊃ T2 ⊃ ··· ⊃ Tk

As it turns out, we can find a nested sequence of pruned subtrees of T_{max}, such that the trees in the sequence minimize total cost for consecutive intervals of α values.

剪枝後的子樹

連續的

我們可以找到一個從最大樹 Tmax 出發的剪枝樹序列，這個序列是嵌套的，且對每個 α 區間，序列中的某棵樹是最佳的（使成本函數最小）。

Smallest minimizing subtree

T(α): 對應特定 α 的最佳剪枝樹（pruned subtree），也就是能最小化成本函數的樹。

Ca(T): Total Cost, Ca(T) = R(T) + α|˜T |

T(α)≤T: T(α) 是所有成本最小的樹的最小剪枝樹，意思是如果其他樹 T 也達到相同成本 Ca(T) =Ca(T(α))，那麼 T(α) 是它的子樹。

Theorem:

For any value of α, there exists a smallest minimizing subtree $T(\alpha)$ of T_{max} that satisfies the following conditions:

1. $T(\alpha)$ minimizes total cost for that value of α:
   $C_\alpha(T(\alpha)) = \min_{T \leq T_{\text{max}}} C_\alpha(T)$
2. $T(\alpha)$ is a pruned subtree of all trees that minimize total cost:
   if $C_\alpha(T) = C_\alpha(T(\alpha))$ then $T(\alpha) \leq T$.

Note: $T' \leq T$ means T' is a pruned subtree of T.

Sequence of subtrees

Construct a *decreasing sequence* of pruned subtrees of $T_{\max}$

$$T_{\max} > T_1 > T_2 > T_3 > \ldots > \{t_1\}$$

(where t_1 is the root node of the tree) such that T_k is the smallest minimizing subtree for $\alpha \in [\alpha_k, \alpha_{k+1})$.

Note: From a computational viewpoint, the important property is that T_{k+1} is guaranteed to be a pruned subtree of T_k. No backtracking is required.

Decomposition of total cost
分解

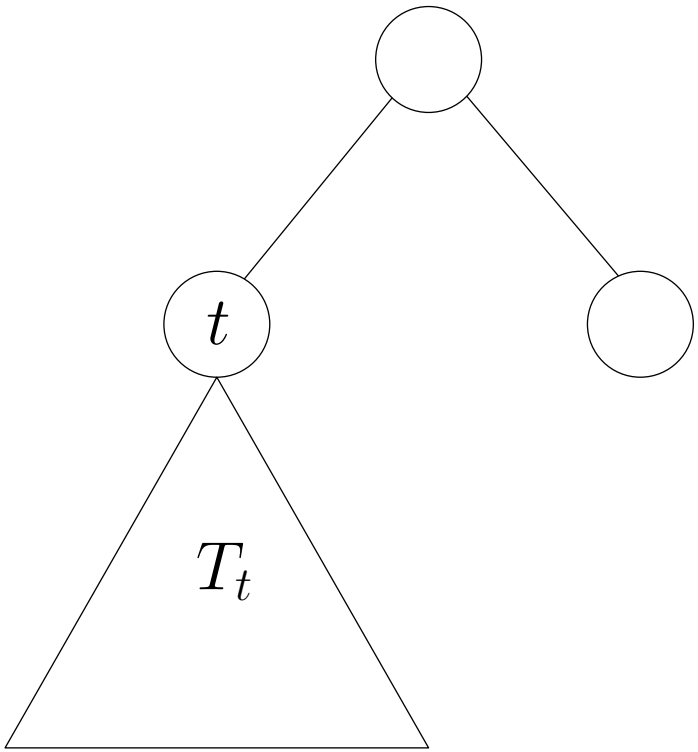
The total cost of a tree is the sum of the contributions of its leaf nodes:

$$C_\alpha(T) = R(T) + \alpha|\tilde{T}| \stackrel{?}{=} \sum_{t \in \tilde{T}} (R(t) + \alpha)$$

$R(t)$ is the number of errors we make in node t if we predict the majority class, divided by the total number of observations in the training sample.

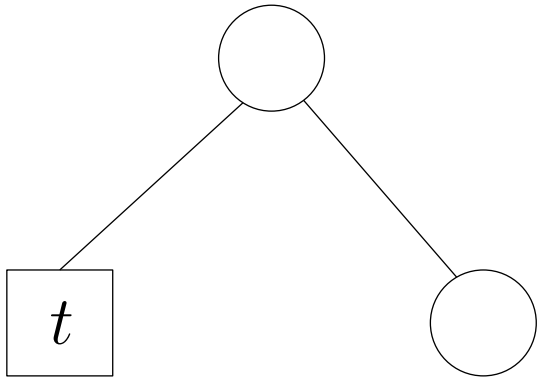
Effect of pruning in node t on cost

Before pruning in t



$C_\alpha(T_t) = \Sigma_{t' \in \tilde{T}_t}(R(t') + \alpha)$

After pruning in t



$C_\alpha(\{t\}) = R(t) + \alpha$

Finding the T_k and corresponding α_k

T_t: branch of T with root node t.
After pruning in t, its contribution to total cost is:

$$C_\alpha(\{t\}) = R(t) + \alpha,$$ 只保留根節點 t，剪掉 Tt 的其餘部分的成本

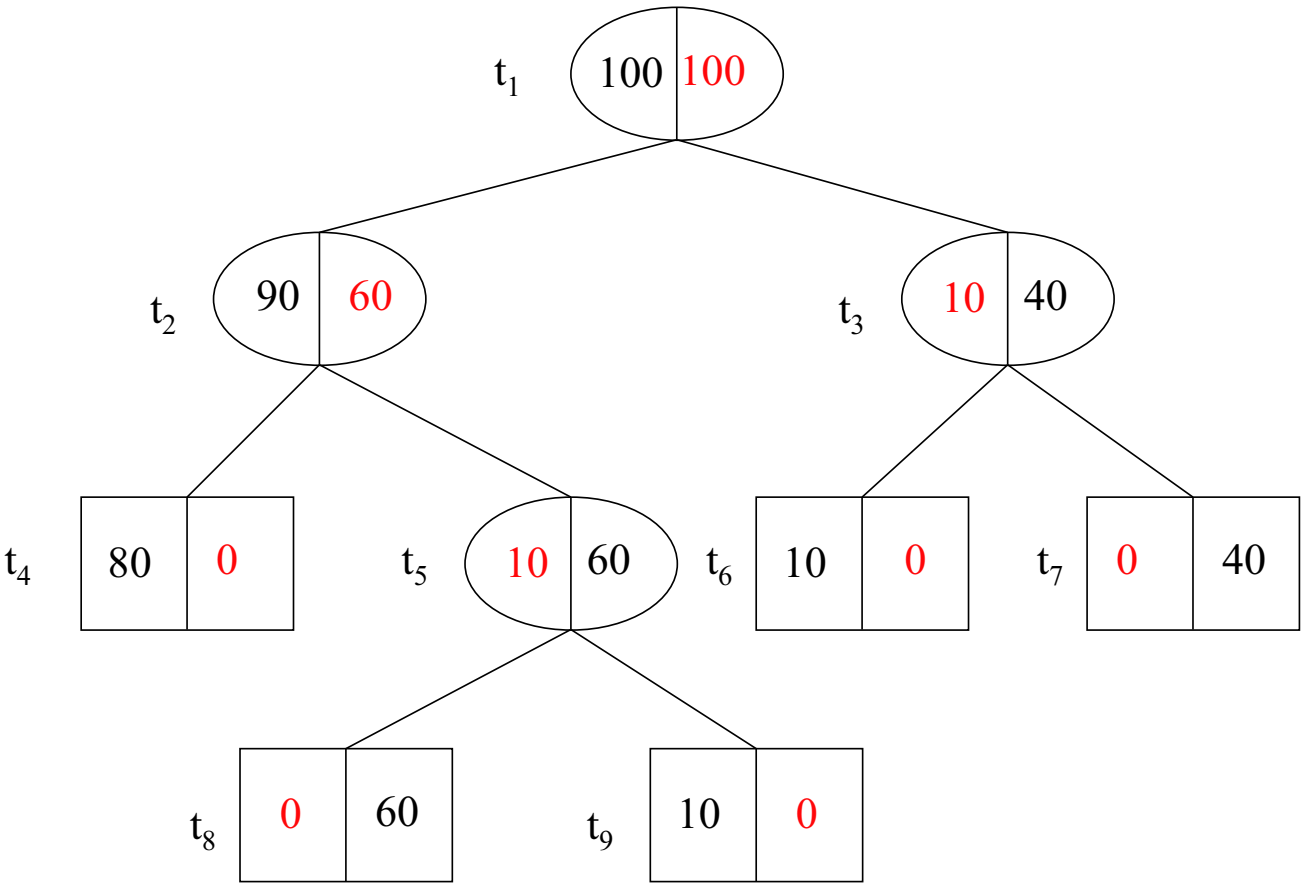
The contribution of T_t to the total cost is:

$$C_\alpha(T_t) = \sum_{t' \in \tilde{T}_t} (R(t') + \alpha) = R(T_t) + \alpha|\tilde{T}_t|$$ 保留整個子樹 Tt 的成本

The tree obtained by pruning in t becomes better than T when

$$C_\alpha(\{t\}) = C_\alpha(T_t)$$

Computing contributions to total cost of T



$$C_\alpha(\{t_2\}) = R(t_2) + \alpha = \frac{3}{10} + \alpha$$
$$C_\alpha(T_{t_2}) = R(T_{t_2}) + \alpha|\tilde{T}_{t_2}| = \alpha|\tilde{T}_{t_2}| + \sum_{t' \in \tilde{T}_{t_2}} R(t') = 3\alpha + 0$$

Solving for α

The total cost of T and $T - T_t$ become equal when

$$C_\alpha(\{t\}) = C_\alpha(T_t),$$

At what value of α does this happen?

$$R(t) + \alpha = R(T_t) + \alpha|\tilde{T}_t|$$

Solving for α we get

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

Note: for this value of α total cost of T and $T - T_t$ is the same, but $T - T_t$ is preferred because we want the *smallest* minimizing subtree.

Computing $g(t)$: the "critical" α value for node t

- For each non-terminal node t we compute its "critical" *alpha* value:

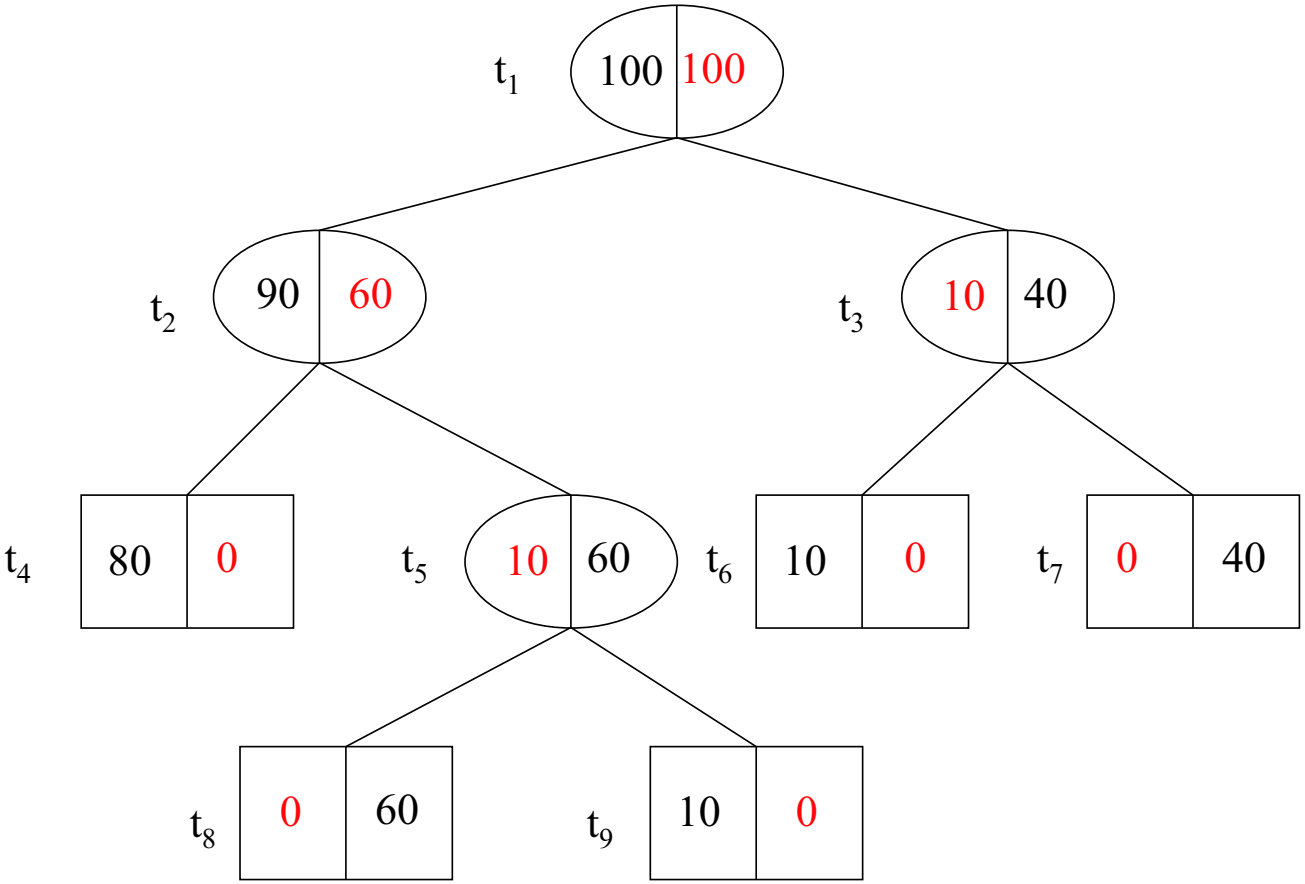
$$g(t) = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}$$

In words:

$$g(t) = \frac{\text{increase in error due to pruning in } t}{\text{decrease in } \# \text{ leaf nodes due to pruning in } t}$$

- Subsequently, we prune in the nodes for which $g(t)$ is the smallest (the "weakest links").
- This process is repeated until we reach the root node.

Computing $g(t)$: the "critical" α value for node t



$$g(t_1) = \frac{1}{8}, g(t_2) = \frac{3}{20}, g(t_3) = \frac{1}{20}, g(t_5) = \frac{1}{20}.$$

Computing $g(t)$: the "critical" α value for node t

Calculation examples:

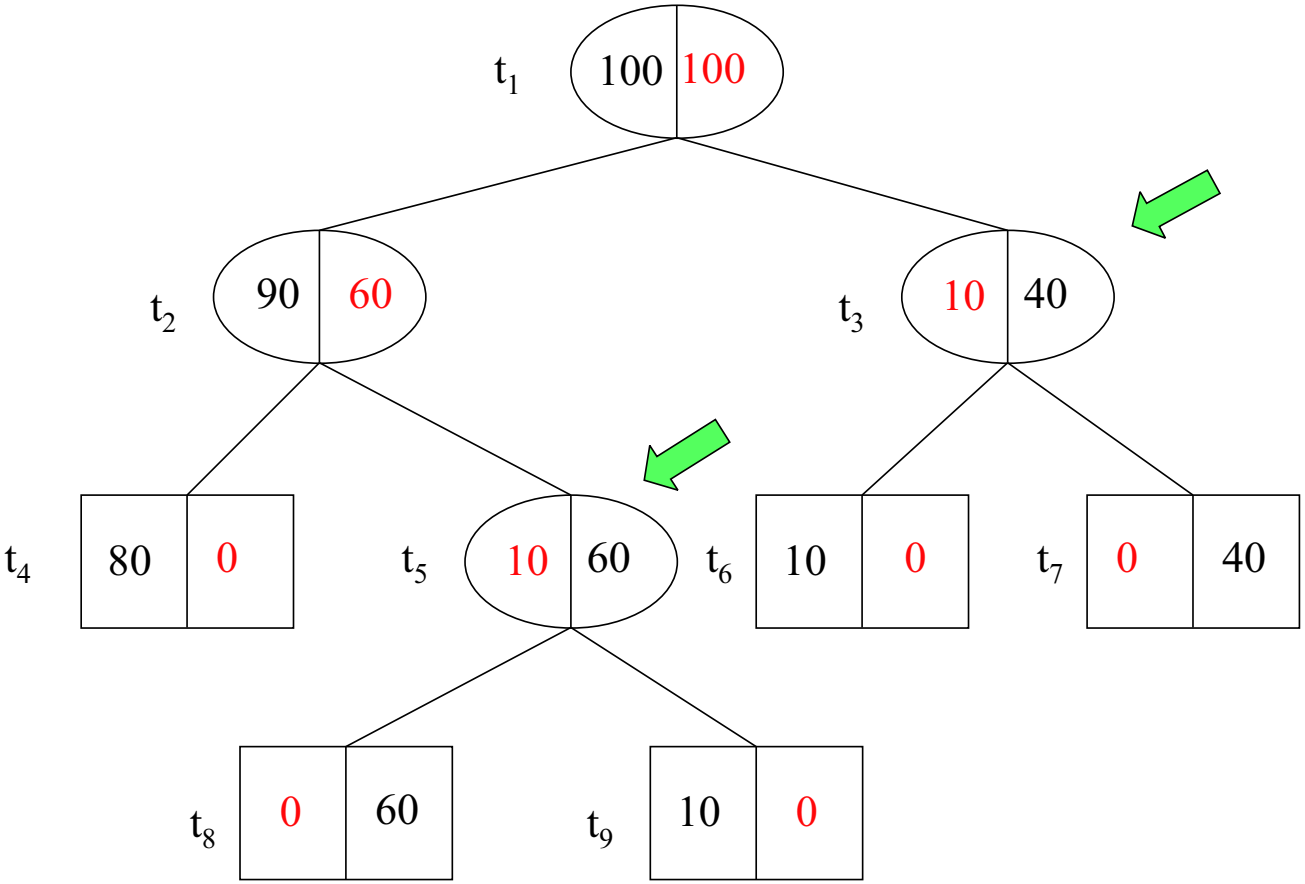
$$g(t_1) = \frac{R(t_1) - R(T_{t_1})}{|\tilde{T}_{t_1}| - 1} = \frac{1/2 - 0}{5 - 1} = \frac{1}{8}$$

$$g(t_2) = \frac{R(t_2) - R(T_{t_2})}{|\tilde{T}_{t_2}| - 1} = \frac{3/10 - 0}{3 - 1} = \frac{3}{20}$$

$$g(t_3) = \frac{R(t_3) - R(T_{t_3})}{|\tilde{T}_{t_3}| - 1} = \frac{1/20 - 0}{2 - 1} = \frac{1}{20}$$

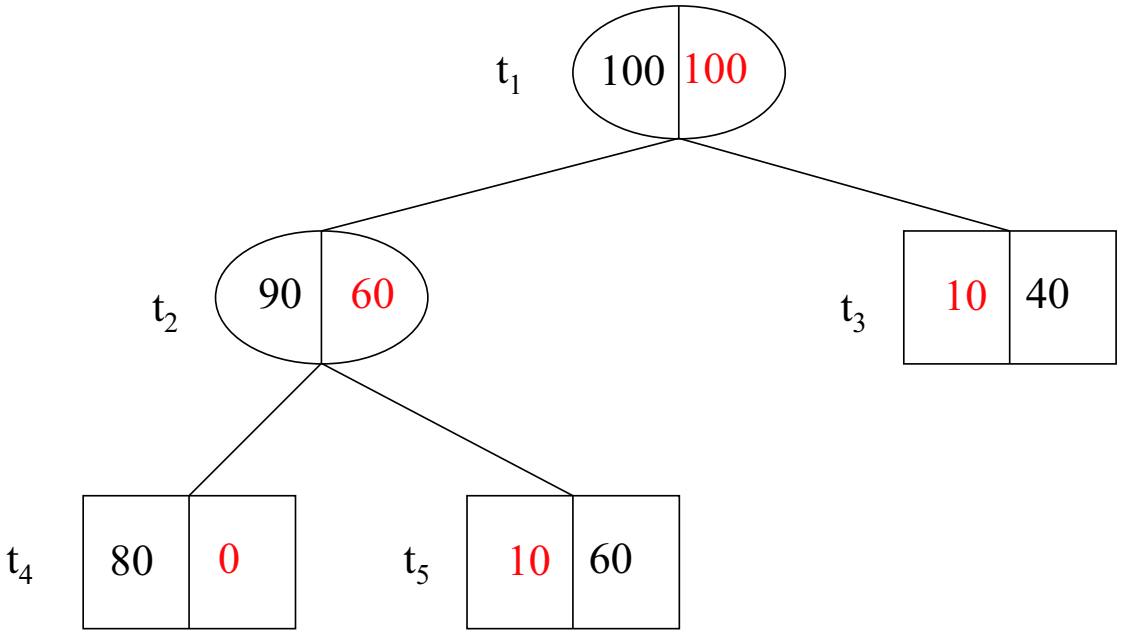
$$g(t_5) = \frac{R(t_5) - R(T_{t_5})}{|\tilde{T}_{t_5}| - 1} = \frac{1/20 - 0}{2 - 1} = \frac{1}{20}$$

Finding the weakest links



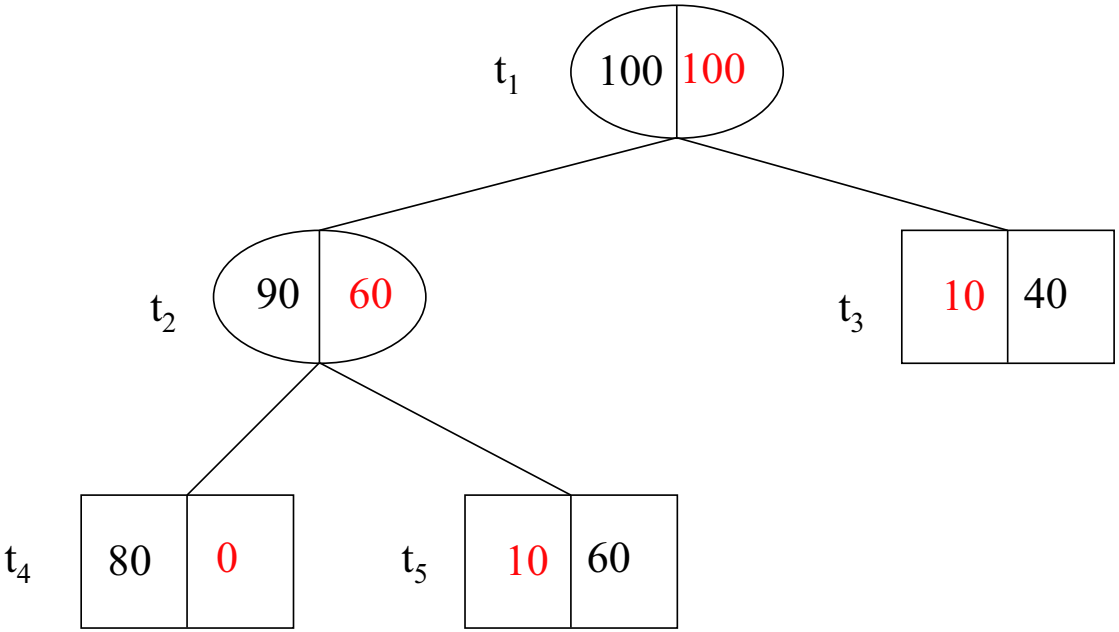
$$g(t_1) = \frac{1}{8}, g(t_2) = \frac{3}{20}, g(t_3) = \frac{1}{20}, g(t_5) = \frac{1}{20}.$$

Pruning in the weakest links



By pruning the weakest links we obtain the next tree in the sequence.

Repeating the same procedure



$g(t_1) = \frac{2}{10}, g(t_2) = \frac{1}{4}.$

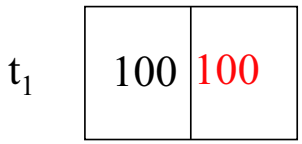
Computing $g(t)$: the "critical" α value for node t

Calculation examples:

$$g(t_1) = \frac{R(t_1) - R(T_{t_1})}{|\tilde{T}_{t_1}| - 1} = \frac{1/2 - 1/10}{3 - 1} = \frac{2}{10}$$

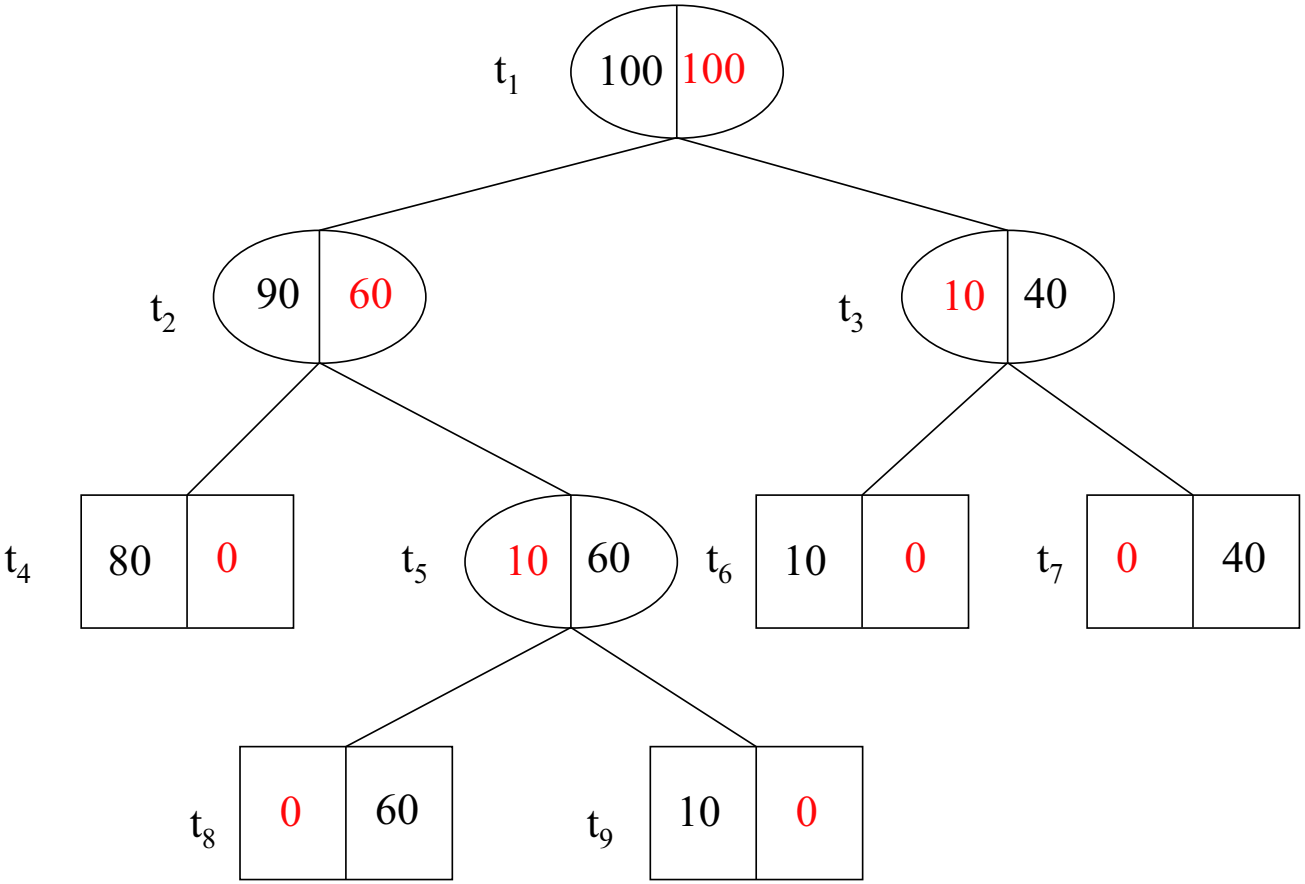
$$g(t_2) = \frac{R(t_2) - R(T_{t_2})}{|\tilde{T}_{t_2}| - 1} = \frac{3/10 - 1/20}{2 - 1} = \frac{1}{4}$$

Going back to the root



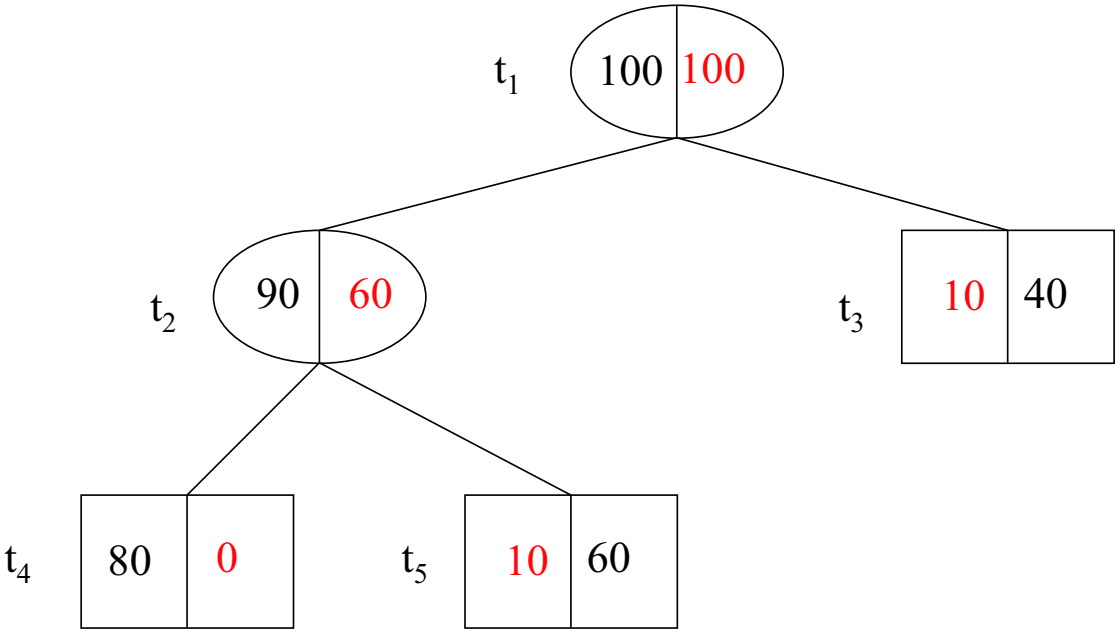
We have arrived at the root so we're done.

The best tree for $\alpha \in [0, \frac{1}{20})$



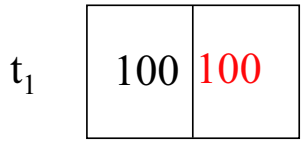
The big tree is the best for values of α below $\frac{1}{20}$.

The best tree for $\alpha \in [\frac{1}{20}, \frac{2}{10})$



When α reaches $\frac{1}{20}$ this tree becomes the best.

The best tree for $\alpha \in [\frac{2}{10}, \infty)$



When α reaches $\frac{2}{10}$ the root wins and we're done.

Computing the Pruning Sequence

讀書
post-order

$T_1 \leftarrow T(\alpha = 0);\ \alpha_1 \leftarrow 0;\ k \leftarrow 1$

T(α=0) = T max ：從最大樹開始（不剪枝）。
α1 = 0 : 初始成本複雜度參數為 0。
k = 1：迭代計數器。

While $T_k > \{t_1\}$ do t1 : 根節點

For all non-terminal nodes $t \in T_k$

$$g_k(t) \leftarrow \frac{R(t) - R(T_{k,t})}{|\tilde{T}_{k,t}| - 1}$$

$\alpha_{k+1} \leftarrow \min_t g_k(t)$ 找到當前樹中所有非終端節點的 最小臨界 α

Visit the nodes in post-order and prune whenever $g_k(t) = \alpha_{k+1}$ to obtain T_{k+1}

Post-order 遍歷：先處理子節點，再處理根節點，保證先剪掉子節點再剪根節點。

$k \leftarrow k + 1$

od

Note: $T_{k,t}$ is the branch of T_k with root node t, and T_k is the pruned tree in iteration k.

Algorithm to compute T_1 from $T_{\max}$

If we don't continue splitting until all nodes are pure, then $T_1 = T(\alpha = 0)$ may not be the same as $T_{\max}$.

Tmax：最大樹，通常是「完全生長的樹」，但如果你沒有分裂到所有節點都純淨（pure nodes），最大樹可能還有些不必要的分支。

T1 = T(α=0)：α=0 對應成本函數只關注誤差，不考慮複雜度。
如果最大樹不是完全純淨，直接取 Tmax 就不一定等於 T1，需要剪掉一些多餘的枝節。

Compute T_1 from $T_{\max}$

$T' \leftarrow T_{\max}$

Repeat

Pick any pair of terminal nodes ℓ and r with common parent t in T' such that $R(t) = R(\ell) + R(r)$, and set $T' \leftarrow T' - T_t$ (i.e. prune T' in t)

節點 t 的誤差等於它兩個子節點誤差之和
→ 則保留子節點 ℓ、r 並沒有降低誤差

Until no more such pair exists

$T_1 \leftarrow T'$

Selection of the final tree: using a test set

唸書
standard error

Pick the tree T from the sequence with the lowest error rate $R^{ts}(T)$ on the test set.

剪枝樹序列：之前生成的嵌套樹序列T 1 ⊃T 2 ⊃···⊃T k
測試集誤差 Rts(T)：將每棵樹在測試集上分類或預測，計算錯誤比例（error rate）。
選擇在測試集上 錯誤率最低 的樹作為最終樹

This is an *estimate* of the true error rate $R^*(T)$ of T.

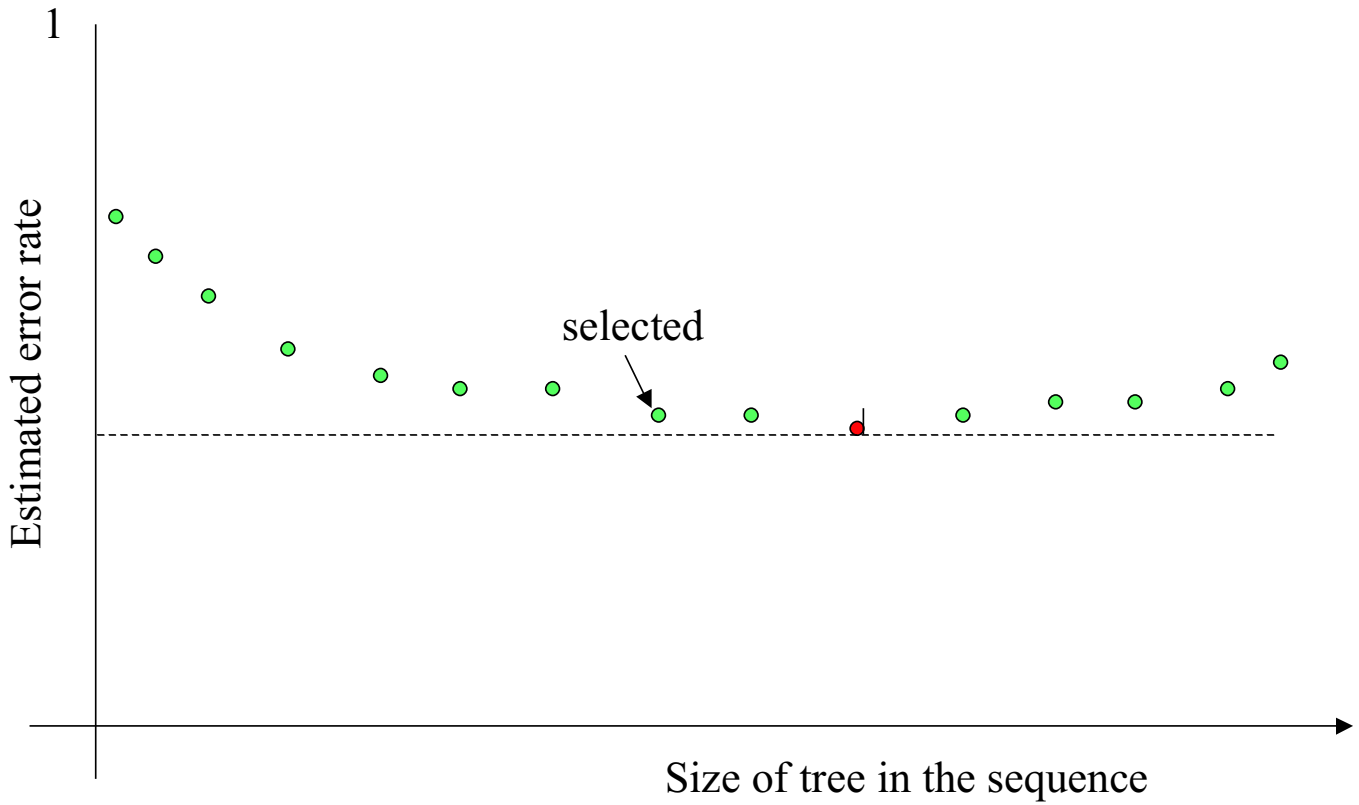
R∗ (T)：樹在未來、實際應用中的真實錯誤率（population error rate）。
R ts (T)：用測試集估計的錯誤率，因為測試集只是樣本，所以這是 對 R∗ (T) 的一個估計。

The standard error of this estimate is

$$SE(R^{ts}) = \sqrt{\frac{R^{ts}(1-R^{ts})}{n_{test}}},$$

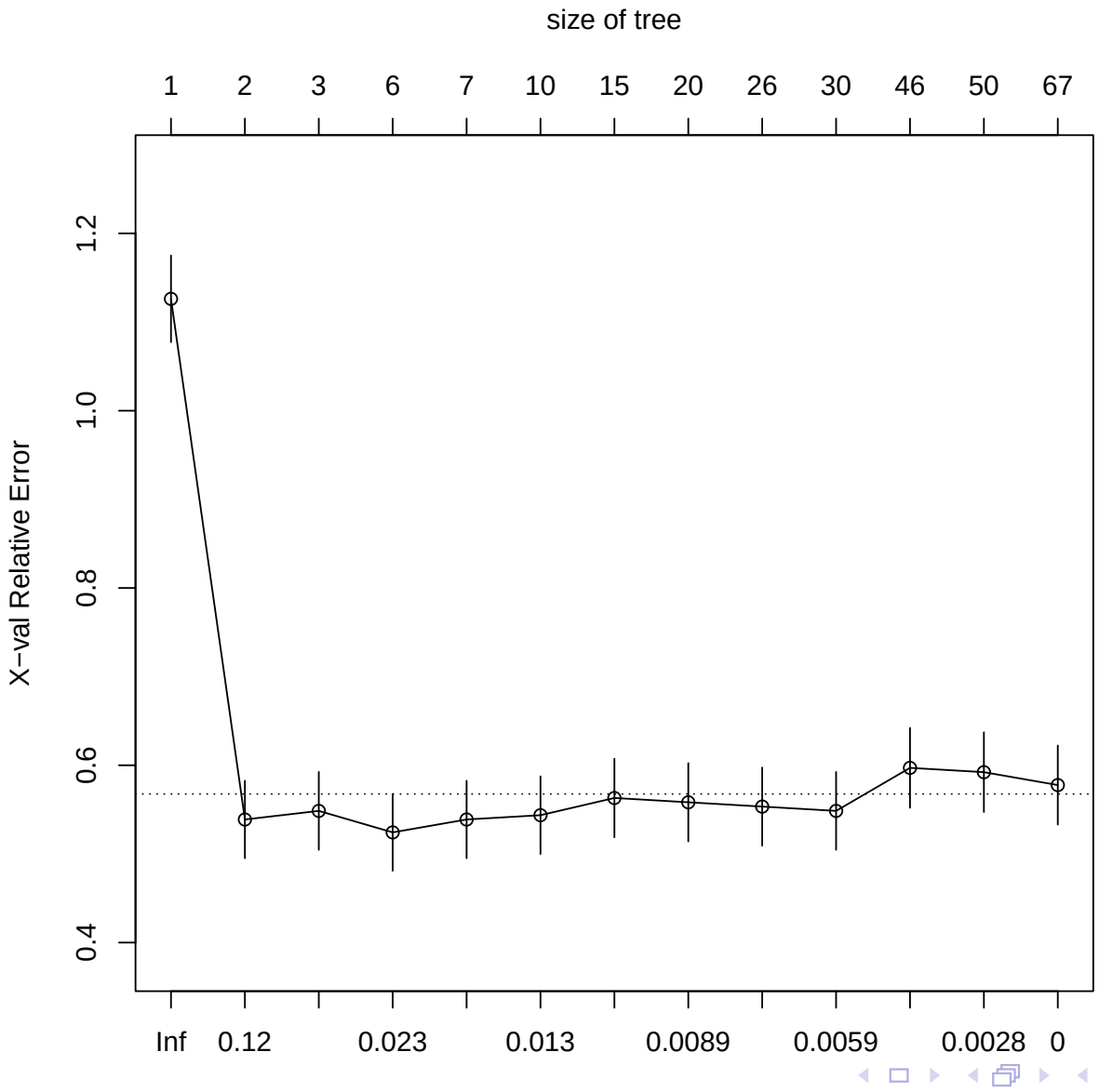
where n_{test} is the number of observations in the test set.

Selection of the final tree: the 1-SE rule

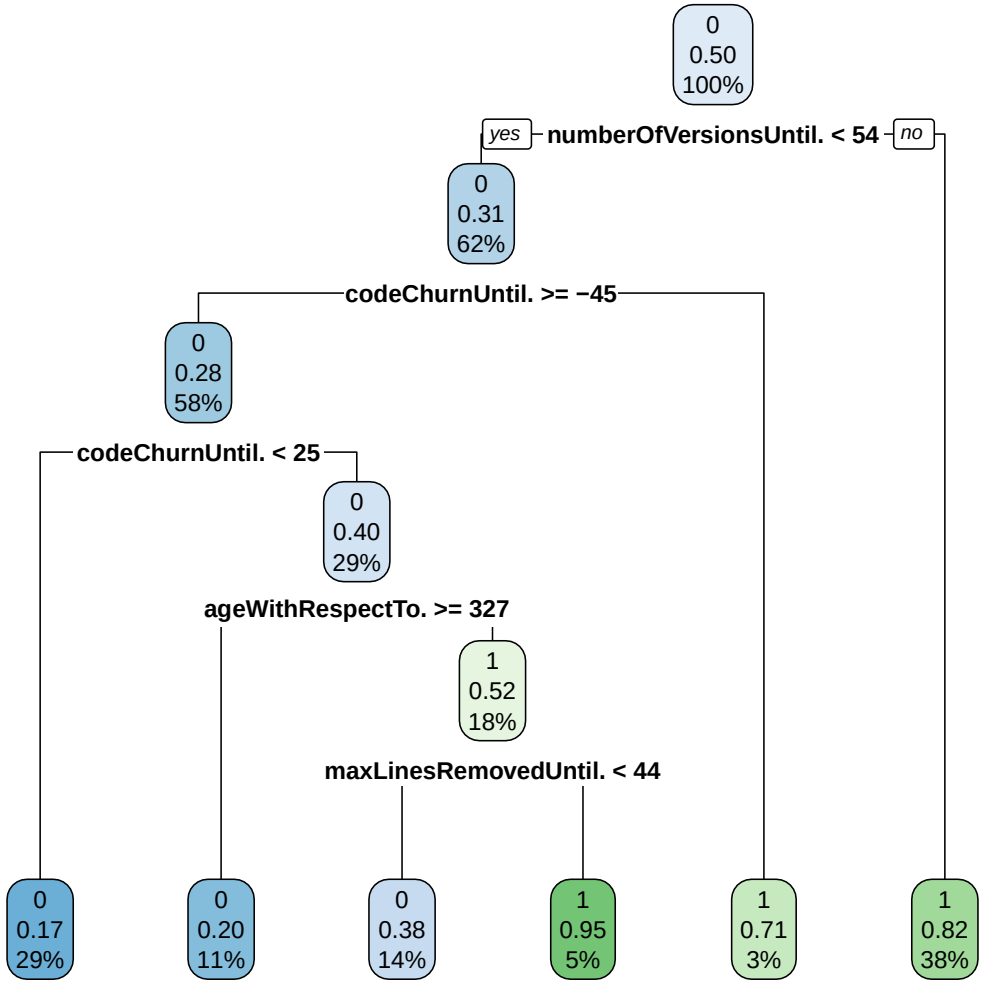


1-SE rule: select the smallest tree with R^{ts} within one standard error of the minimum.

Bug Prediction Tree Pruning Sequence



Bug Prediction Tree after Pruning



Cross-Validation

- When the data set is relatively small, it is a bit of a waste to set aside part of the data for testing.
- A way to avoid this problem is to use *cross-validation*.

Cross-Validation

1. Divide data into v folds.
2. Train on $v - 1$ folds.
3. Predict on the remaining fold.
4. Leave out each of the v folds in turn.

Cross-Validation

First iteration:

fold	X	Y	$\hat{Y}$
1			
2			
3			
4			
5			$\hat{Y}^{(5)}$

Cross-Validation

Second iteration:

fold	X	Y	$\hat{Y}$
1			
2			
3			
4			$\hat{Y}^{(4)}$
5			

Cross-Validation

Third iteration:

fold	X	Y	$\hat{Y}$
1			
2			
3			$\hat{Y}^{(3)}$
4			
5			

Cross-Validation

Fourth iteration:

fold	X	Y	$\hat{Y}$
1			
2			$\hat{Y}^{(2)}$
3			
4			
5			

Cross-Validation

Fifth iteration:

fold	X	Y	$\hat{Y}$
1			$\hat{Y}^{(1)}$
2			
3			
4			
5			

Cross-Validation

In the end we have out-of-sample predictions for all cases!

fold	X	Y	$\hat{Y}$
1			$\hat{Y}^{(1)}$
2			$\hat{Y}^{(2)}$
3			$\hat{Y}^{(3)}$
4			$\hat{Y}^{(4)}$
5			$\hat{Y}^{(5)}$

1. Perform cross-validation for different hyper-parameter settings (e.g. nmin and minleaf). n min：節點最小樣本數，決定何時停止分裂 minleaf：葉子節點最小樣本數
2. Compute prediction error for each parameter setting.
3. Pick setting with lowest error.
4. Train with selected setting on complete data set.

v-fold cross-validation (general)

複雜度參數 C：控制模型複雜度的參數

Let C be a complexity parameter of a learning algorithm (like α in the classification tree algorithm). To select the best value of C from a range of values $c_1, \ldots, c_m$ we proceed as follows. 在給定的候選值 c 1 ,...,cm 中選出最適合的 C

1. Divide the data into v groups $G_1, \ldots, G_v$.
2. For each value c_i of C
   1. For $j = 1, \ldots, v$
      1. Train with $C = c_i$ on all data *except* group G_j.
      2. Predict on group G_j. 在 Gj 上測試模型，計算誤差
   2. Compute the CV prediction error for $C = c_i$.
3. Select the value c^* of C with the smallest CV prediction error.
4. Train on the complete training sample with $C = c^*$

Using cross-validation: Step 1

Grow a tree on the full data set, and compute $\alpha_1, \alpha_2, \ldots, \alpha_K$ and $T_1 > T_2 > \ldots > T_K$.

Recall that T_k is the smallest minimizing subtree for $\alpha \in [\alpha_k, \alpha_{k+1})$.

Estimate the error of a tree T_k from this sequence as follows.
Set
$\beta_1 = 0$,
$\beta_2 = \sqrt{\alpha_2 \alpha_3}$,
$\beta_3 = \sqrt{\alpha_3 \alpha_4}$,
$\ldots$,
$\beta_{K-1} = \sqrt{\alpha_{K-1} \alpha_K}$,
$\beta_K = \infty$.

β_k is the "representative" value for T_k.

βk 落在區間 [α k ,α k+1) 之內。
也就是說，當 α 在這個範圍時，對應的最小化子樹就是 Tk 。
所以 βk 只是選一個「代表值」，用來代表這個區間的子樹 Tk 。

Using cross-validation: Step 2

Divide the data set into v groups $G_1, G_2, \ldots, G_v$ (of approximately equal size) and for each group G_j

1. Grow a tree on all data *except* G_j, and determine the smallest minimizing subtrees $T^{(j)}(\beta_1), T^{(j)}(\beta_2), \ldots, T^{(j)}(\beta_K)$ for this reduced data set.
2. Compute the error of $T^{(j)}(\beta_k)$ $(k = 1, \ldots, K)$ on G_j.

Using cross-validation: Step 3

1. For each β_k, sum the errors of $T^{(j)}(\beta_k)$ over G_j $(j = 1, \ldots, v)$.
2. Let β_h be the one with the lowest overall error.
   Select T_h as the best tree.
3. Use the error rate computed with cross-validation as an estimate of its error rate.

Remark: Alternatively, we could again use the 1-SE rule in the final step to select the final tree from the sequence.

有時候「誤差最小的樹」可能太複雜。
1-SE rule：選擇一棵「比最佳樹小，但誤差在一個標準誤 (1-SE) 範圍內」的樹。
好處：讓模型更簡單，更有泛化能力。

Using cross-validation: Step 1

Tree sequence constructed on *full* data set:

- T_1 is the best tree for $\alpha \in [0, \frac{1}{20})$.
- T_2 is the best tree for $\alpha \in [\frac{1}{20}, \frac{2}{10})$.
- T_3 is the best tree for $\alpha \in [\frac{2}{10}, \infty)$.

Set

$\beta_1 = 0$, value corresponding to T_1

$\beta_2 = \sqrt{\frac{1}{20}\ \frac{2}{10}} = \frac{1}{10}$, value corresponding to T_2

$\beta_3 = \infty$, value corresponding to T_3 (root).

Using cross-validation: Step 2

Divide the data set in $v = 4$ groups G_1, G_2, G_3, G_4 of size 50 each.

First CV-run

1. Build a tree on all data *except* G_1, and determine the smallest minimizing subtrees $T^{(1)}(0)$, $T^{(1)}(\frac{1}{10})$ and $T^{(1)}(\infty)$.
2. Compute the error of those trees on G_1.

Repeat this procedure for G_2, G_3 and G_4.

Using cross-validation: Step 3

CV-run	$\beta_1 = 0$	$\beta_2 = \frac{1}{10}$	$\beta_3 = \infty$
1	20	10	25
2	18	8	25
3	22	9	25
4	20	13	25
Total	80	40	100

β_2 wins (40 errors), so T_2 gets selected.

We estimate the error rate of T_2 at 20%.

Building Trees in R: Rpart

Pima Indians Diabetes Database from the UCI ML Repository

1. Number of times pregnant
2. Plasma glucose concentration in a glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Class Value	Number of instances
0	500
1	268

Building Trees in R: Rpart

```
> pima.dat[1:5,]
  npreg plasma bp triceps serum  bmi pedigree age class
1     6    148 72      35     0 33.6    0.627  50     1
2     1     85 66      29     0 26.6    0.351  31     0
3     8    183 64       0     0 23.3    0.672  32     1
4     1     89 66      23    94 28.1    0.167  21     0
5     0    137 40      35   168 43.1    2.288  33     1

> library(rpart)
> pima.tree <- rpart(class ~.,data=pima.dat,cp=0,minbucket=1,minsplit=2,method="class")
> printcp(pima.tree)

Classification tree:
rpart(formula = class ~ ., data = pima.dat, method = "class",
    cp = 0, minbucket = 1, minsplit = 2)

Variables actually used in tree construction:
[1] age      bmi      bp       npreg    pedigree plasma   serum    triceps

Root node error: 268/768 = 0.34896

n= 768
```

cptable: the pruning sequence

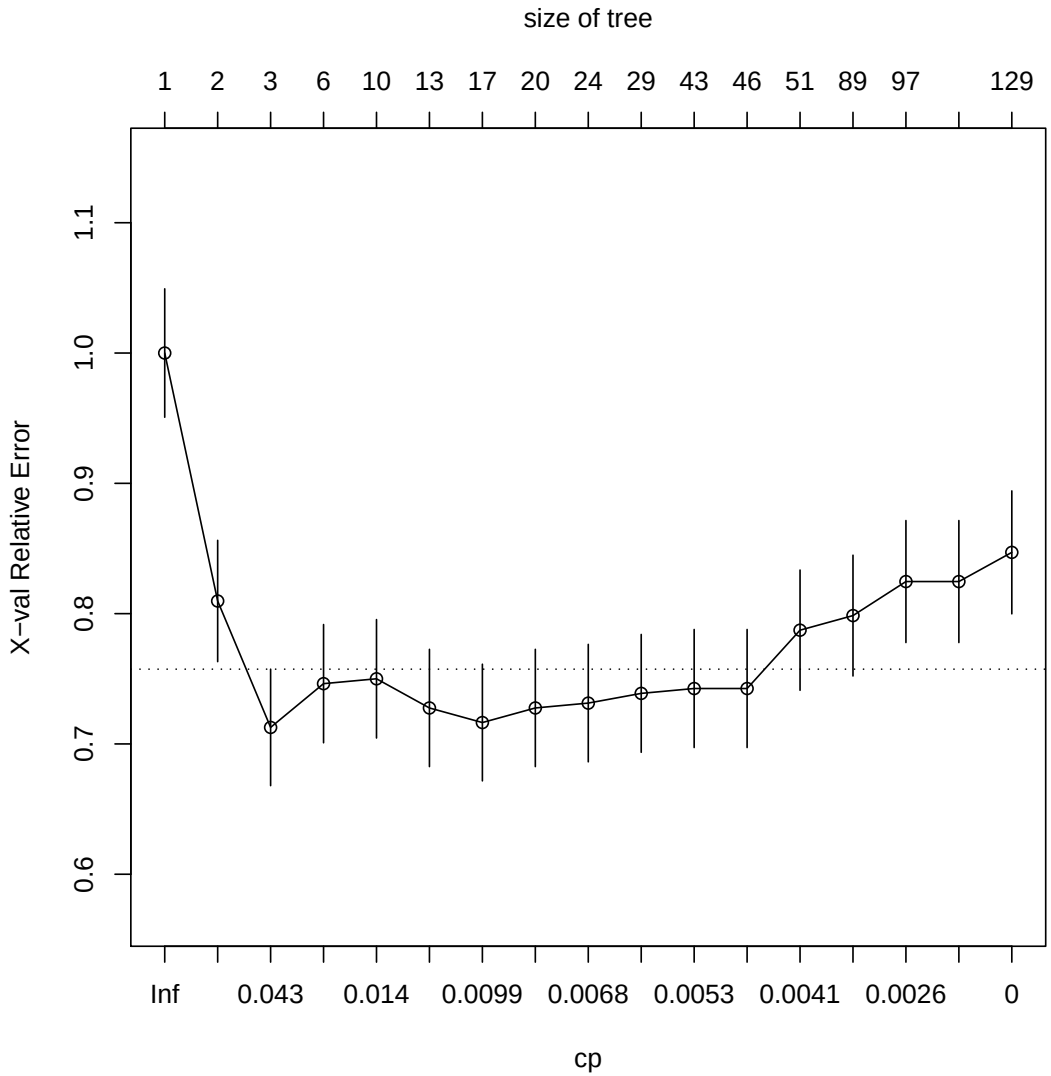
	CP	nsplit	rel error	xerror	xstd
1	0.2425373	0	1.000000	1.00000	0.049288
2	0.1044776	1	0.757463	0.80970	0.046558
3	0.0174129	2	0.652985	0.71269	0.044698
4	0.0149254	5	0.600746	0.74627	0.045381
5	0.0130597	9	0.541045	0.75000	0.045454
6	0.0111940	12	0.492537	0.72761	0.045007
7	0.0087065	16	0.447761	0.71642	0.044776
8	0.0074627	19	0.421642	0.72761	0.045007
9	0.0062189	23	0.391791	0.73134	0.045083
10	0.0055970	28	0.358209	0.73881	0.045233
11	0.0049751	42	0.272388	0.74254	0.045307
12	0.0044776	45	0.257463	0.74254	0.045307
13	0.0037313	50	0.235075	0.78731	0.046159
14	0.0027985	88	0.093284	0.79851	0.046360
15	0.0024876	96	0.070896	0.82463	0.046814
16	0.0018657	109	0.037313	0.82463	0.046814
17	0.0000000	128	0.000000	0.84701	0.047184

CP is α divided by the resubstitution error in the root node.

Example: tree with 2 splits is best for CP $\in [0.0174129, 0.1044776)$.

Tree with 2 splits has cross-validation error of $0.34896 \times 0.71269 = 0.2487$.

Plot of pruning sequence



Selecting the best tree

```
> pima.pruned <- prune(pima.tree,cp=0.02)
> post(pima.pruned)
```