Data Mining
Classification Trees (2)

Ad Feelders

Universiteit Utrecht

Basic Tree Construction Algorithm (control flow)

```
Construct tree
    nodelist ← {{training data}}
    Repeat
        current node ← select node from nodelist
        nodelist ← nodelist − current node
        if impurity(current node) > 0
        then
            S ← set of candidate splits in current node
            s* ← $\arg\max_{s \in S}$ impurity reduction(s,current node)
            child nodes ← apply(s*,current node)
            nodelist ← nodelist ∪ child nodes
        fi
    Until nodelist = ∅
```

Overfitting and Pruning

- The tree growing algorithm continues splitting until all leaf nodes of T contain examples of a single class
(i.e. resubstitution error $R(T) = 0$).
- Is this a good tree for predicting the class of new examples?
- Not unless the problem is truly “deterministic”!
- Problem of *overfitting*.

Proposed Solutions

How can we prevent overfitting?

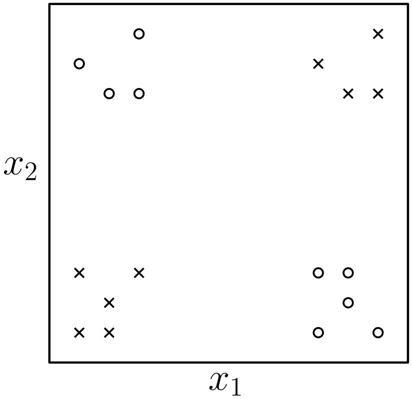
- Stopping Rules: e.g. don't expand a node if the impurity reduction of the best split is below some threshold.
- Pruning: grow a very large tree $T_{\max}$ and merge back nodes.

Note: in the practical assignment we do use a stopping rule based on the `nmin` and `minleaf` parameters.

Stopping Rules

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

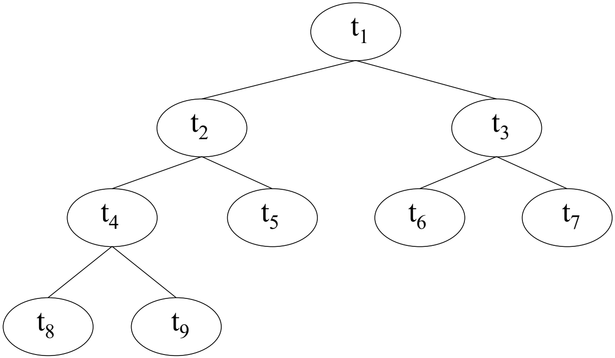
Since we only look one step ahead we may miss the good follow-up split.



Pruning

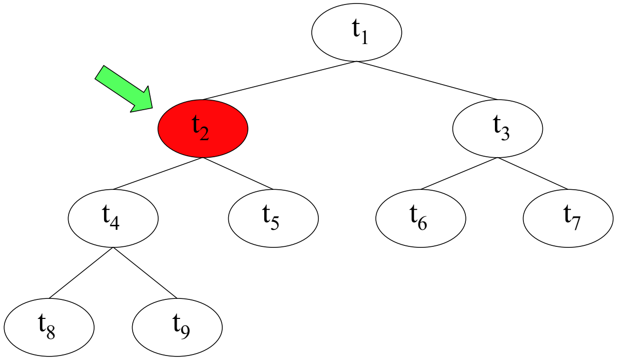
- To avoid the problem of stopping rules, we first grow a very large tree on the training sample, and then *prune* this large tree.
- Objective: select the pruned subtree that has lowest *true* error rate.
- Problem: how to find this pruned subtree?
- Cost-complexity pruning (Breiman et al.; CART), also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

Terminology: Tree T

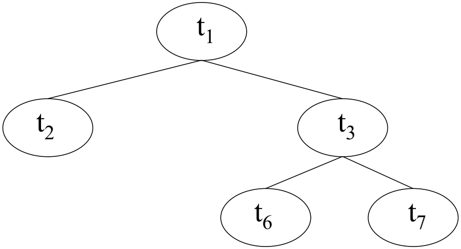


$\tilde{T}$ denotes the collection of leaf nodes of tree T.
$\tilde{T} = \{t_5, t_6, t_7, t_8, t_9\}, |\tilde{T}| = 5$

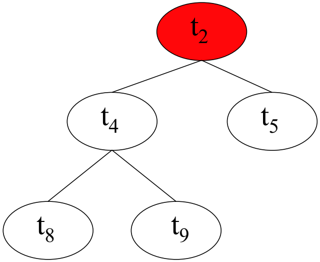
Terminology: Pruning T in node t_2



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

- The total number of pruned subtrees of a balanced binary tree with ℓ leaves 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

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

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.

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.

Smallest minimizing subtree

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,
i.e. it can be obtained by pruning T in 0 or more nodes.

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 a pruned subtree of T_k. No backtracking is required.

Decomposition of total cost

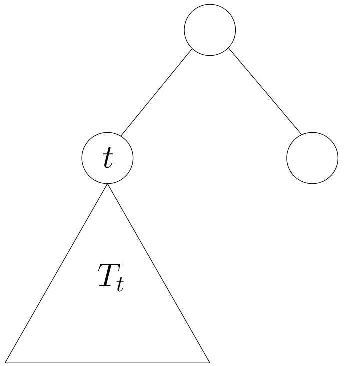
Total cost has an additive decomposition over the leaf nodes of a tree:

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

$R(t)$ 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.

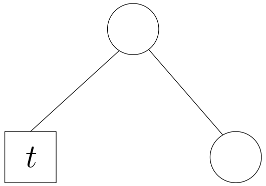
Effect on cost of pruning in node t

Before pruning in t



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

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:

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

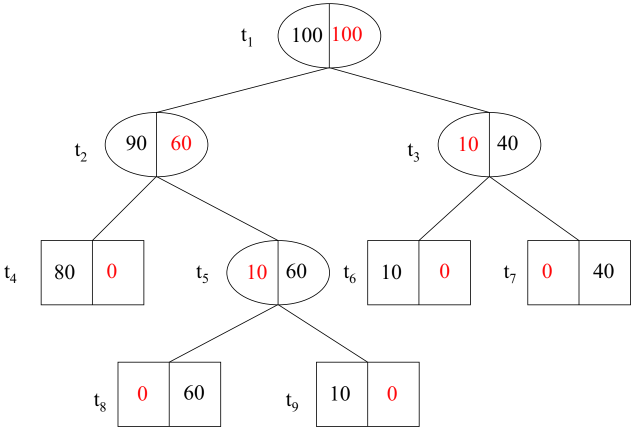
The contribution of T_t to the total cost is:

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

$T - T_t$ 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:

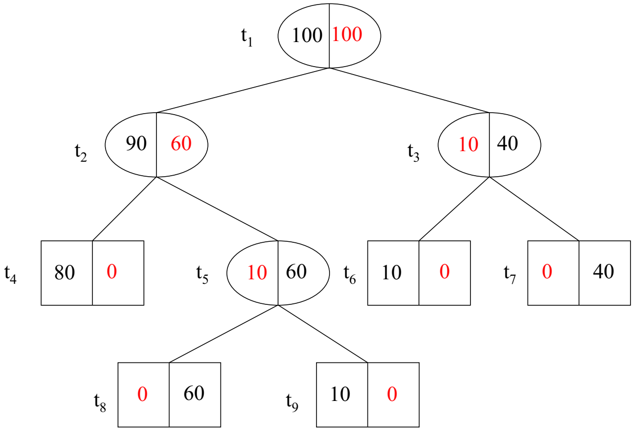
$$\boxed{g(t) = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}}$$

In words:

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

- 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:

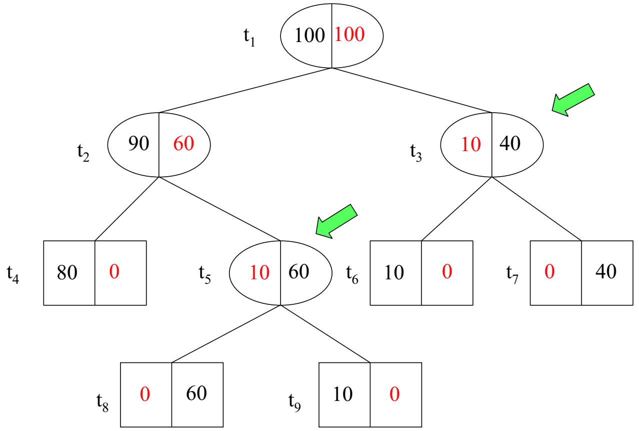
$$g(t_1) = \frac{R(t_1) - R(T_{t_1})}{|\tilde{T}_{t_1}| - 1} = \frac{1/2 - 0}{5 - 1} = \frac{1}{8}$$

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

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

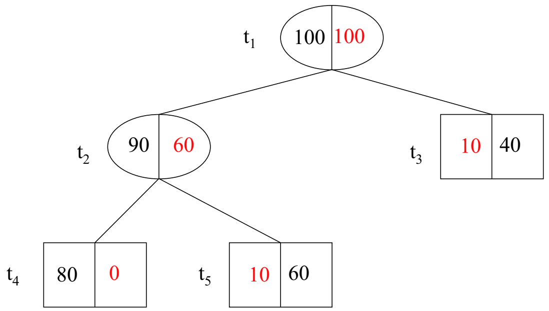
$$g(t_5) = \frac{R(t_5) - R(T_{t_5})}{|\tilde{T}_{t_5}| - 1} = \frac{1/20 - 0}{2 - 1} = \frac{1}{20}$$

Finding the weakest links



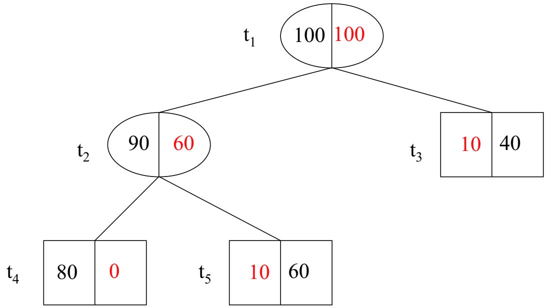
$$g(t_1) = \frac{1}{8}, g(t_2) = \frac{3}{20}, g(t_3) = \frac{1}{20}, g(t_5) = \frac{1}{20}.$$

Pruning in the weakest links



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

Repeating the same procedure



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

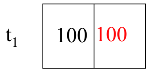
Computing $g(t)$: the "critical" α value for node t

Calculation examples:

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

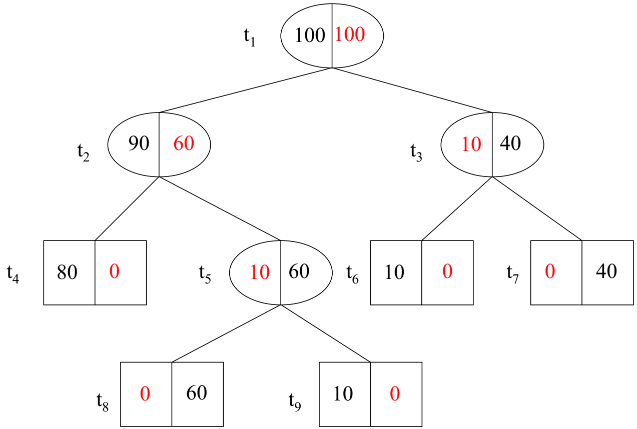
$$g(t_2) = \frac{R(t_2) - R(T_{t_2})}{|\tilde{T}_{t_2}| - 1} = \frac{3/10 - 1/20}{2 - 1} = \frac{1}{4}$$

Going back to the root



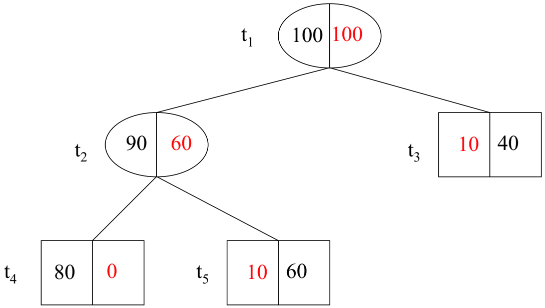
We have arrived at the root so we're done.

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



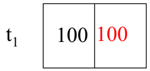
The big tree is the best for values of α below $\frac{1}{20}$.

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

```
$T_1 \leftarrow T(\alpha = 0)$; $\alpha_1 \leftarrow 0$; $k \leftarrow 1$
While $T_k > \{t_1\}$ do
    For all non-terminal nodes $t \in T_k$
        $g_k(t) \leftarrow \frac{R(t) - R(T_{k,t})}{|\tilde{T}_{k,t}| - 1}$
    $\alpha_{k+1} \leftarrow \min_t g_k(t)$
    Visit the nodes in post-order and prune
    whenever $g_k(t) = \alpha_{k+1}$ to obtain $T_{k+1}$
    $k \leftarrow k + 1$
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 $T_{\max}$.

Compute T_1 from $T_{\max}$

```
    $T' \leftarrow T_{\max}$
    Repeat
        Pick any pair of terminal nodes $\ell$ 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 $t$)
    Until no more such pair exists
    $T_1 \leftarrow T'$
```

Selection of the final tree: using a test set

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

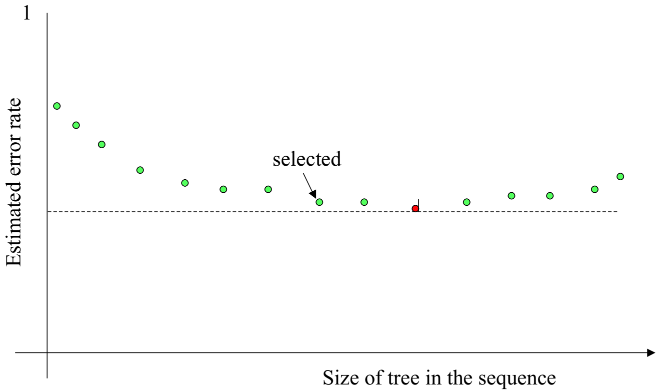
This is an *estimate* of the true error rate $R^*(T)$ of T.

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.

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.

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).
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)

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.

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.
    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 $[\alpha_k, \alpha_{k+1})$.

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.

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 a 2 hours in an oral glucose tolerance tes
   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

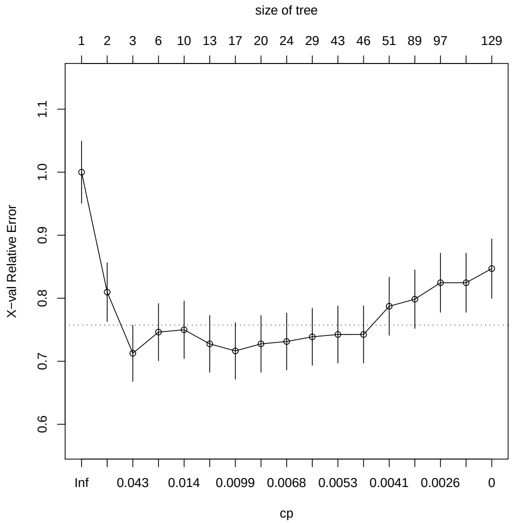
```
          CP nsplit rel error  xerror     xstd
1  0.2425373      0  1.000000 1.00000 0.049288
2  0.1044776      1  0.757463 0.80970 0.046558
3  0.0174129      2  0.652985 0.71269 0.044698
4  0.0149254      5  0.600746 0.74627 0.045381
5  0.0130597      9  0.541045 0.75000 0.045454
6  0.0111940     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 $\in [0.0174129, 0.1044776)$.

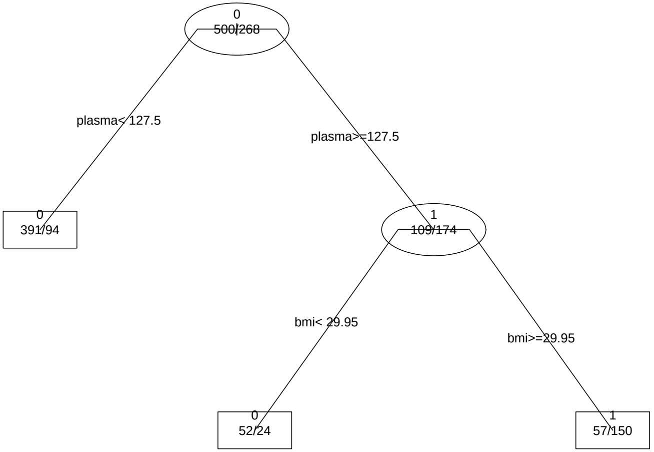
Tree with 2 splits has cross-validation error of $0.34896 \times 0.71269 = 0.2487$.

Plot of pruning sequence



Selecting the best tree

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


0
500/268
plasma< 127.5
plasma>=127.5
0
391/94
1
109/174
bmi< 29.95
bmi>=29.95
0
52/24
1
57/150