

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

 then

$S \leftarrow$ set of candidate splits in current node

$s^* \leftarrow \arg \max_{s \in S} \text{impurity reduction}(s, \text{current node})$

 child nodes $\leftarrow$ apply(s^* , current node)

 nodelist $\leftarrow$ nodelist $\cup$ child nodes

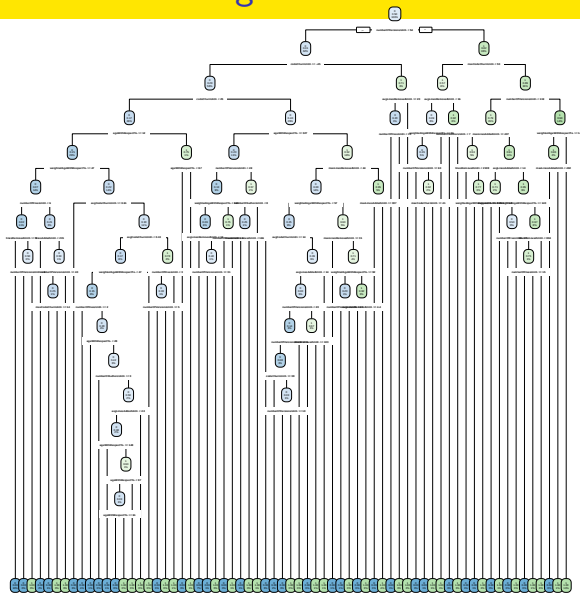
 fi

Until nodelist = $\emptyset$

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.
- Is this a good tree for predicting the class of new examples?
- Not unless the problem is truly “deterministic”!
- Problem of *overfitting*.

An Overfitted Tree on Bug Prediction Data



Proposed Solutions

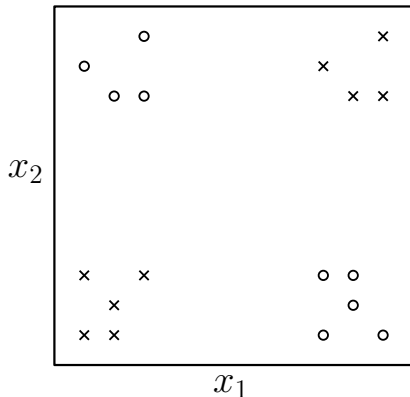
How can we prevent overfitting?

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

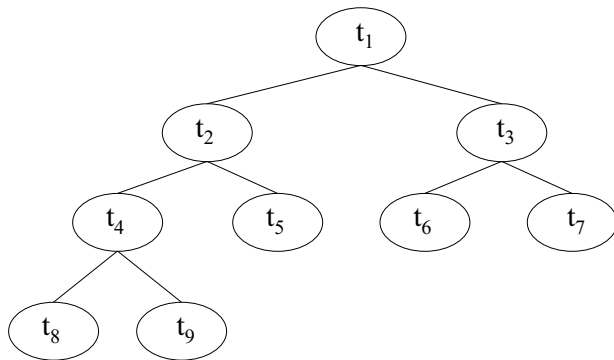
Since we only look one step ahead we may miss the good follow-up split.



Pruning

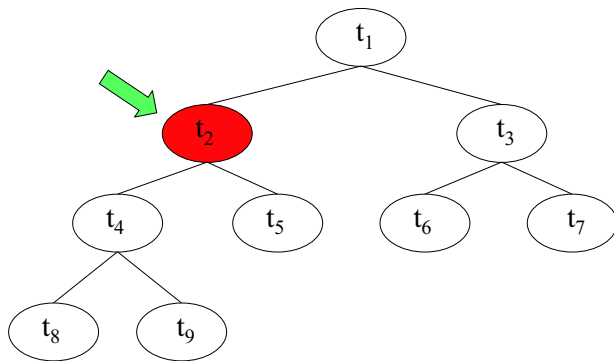
- To avoid the problem of stopping rules, we first grow a very large tree $T_{\max}$ on the training sample, and then *prune* this large tree.
- Objective: select the pruned subtree that has lowest *true* error rate.
- Problem: how to find this pruned subtree?
- Cost-complexity pruning (Breiman et al.;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

Terminology: Tree T

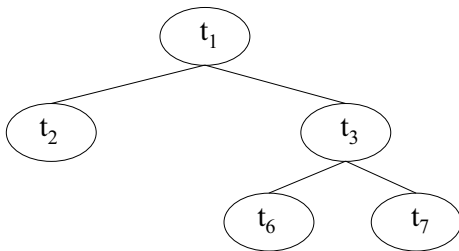


$\tilde{T}$ denotes the collection of leaf nodes of tree T .
 $\tilde{T} = \{t_5, t_6, t_7, t_8, t_9\}, |\tilde{T}| = 5$

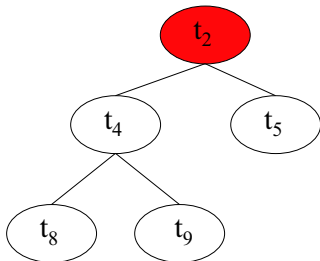
Terminology: Pruning T in node t_2



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

- 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

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

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

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

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 :
$$C_{\alpha}(T(\alpha)) = \min_{T \leq T_{\max}} C_{\alpha}(T)$$
- 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 $T_{\max}$

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

(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

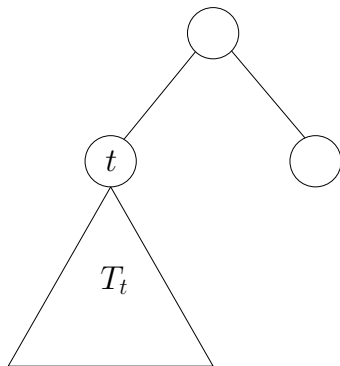
The total cost of a tree is the sum of the contributions of its leaf nodes:

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

$R(t)$ 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.

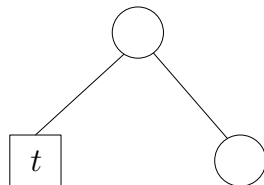
Effect on cost of pruning in node t

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:

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

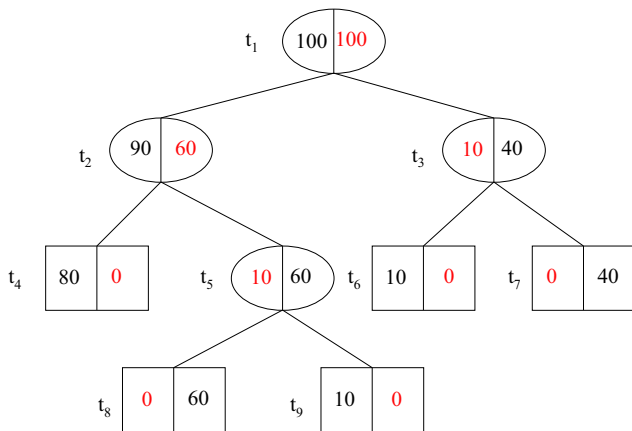
The contribution of T_t to the total cost is:

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

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:

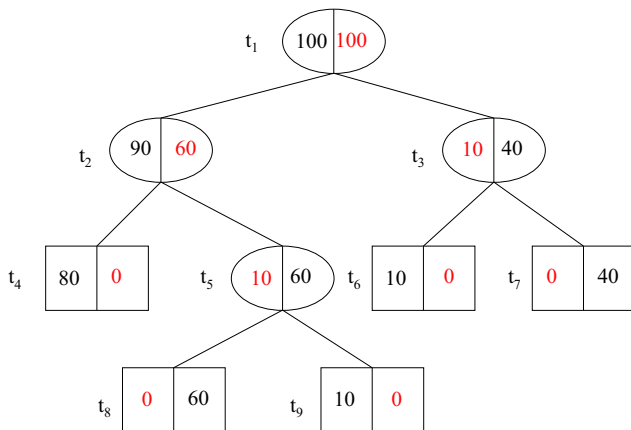
$$g(t) = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}$$

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:

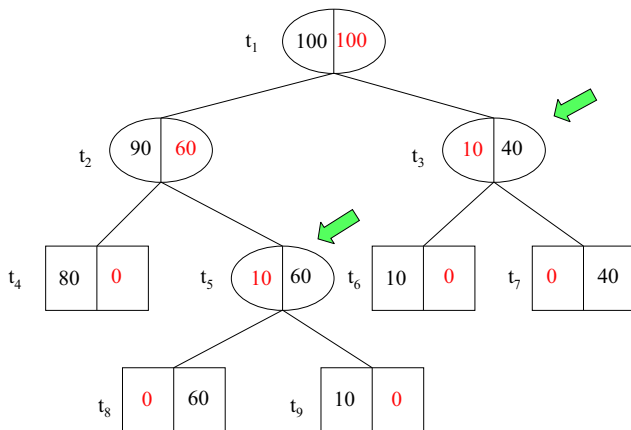
$$g(t_1) = \frac{R(t_1) - R(T_{t_1})}{|\tilde{T}_{t_1}| - 1} = \frac{1/2 - 0}{5 - 1} = \frac{1}{8}$$

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

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

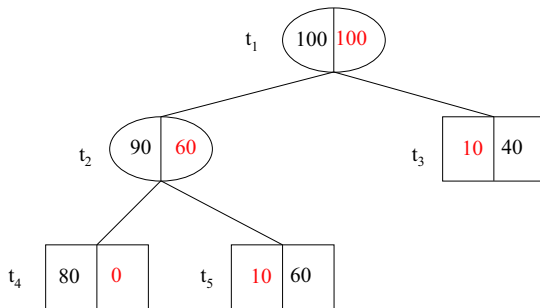
$$g(t_5) = \frac{R(t_5) - R(T_{t_5})}{|\tilde{T}_{t_5}| - 1} = \frac{1/20 - 0}{2 - 1} = \frac{1}{20}$$

Finding the weakest links



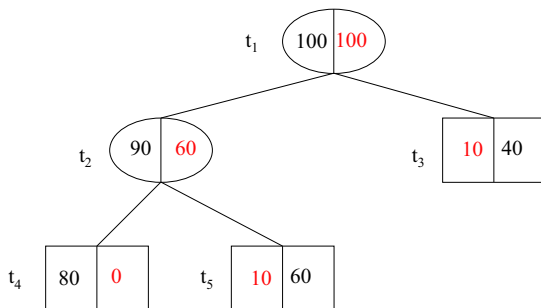
$$g(t_1) = \frac{1}{8}, g(t_2) = \frac{3}{20}, g(t_3) = \frac{1}{20}, g(t_5) = \frac{1}{20}.$$

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:

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

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

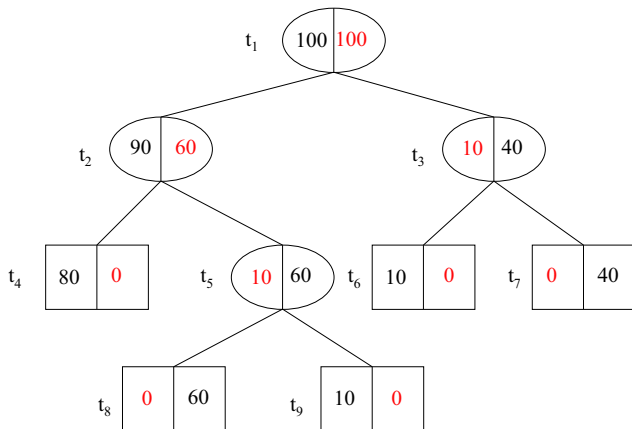
Going back to the root

t_1

100	100
-----	-----

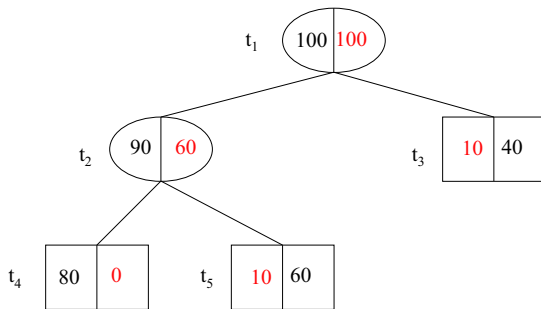
We have arrived at the root so we're done.

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 $\alpha \in [\frac{2}{10}, \infty)$

t_1	<table><tr><td>100</td><td>100</td></tr></table>	100	100
100	100		

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 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)

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.

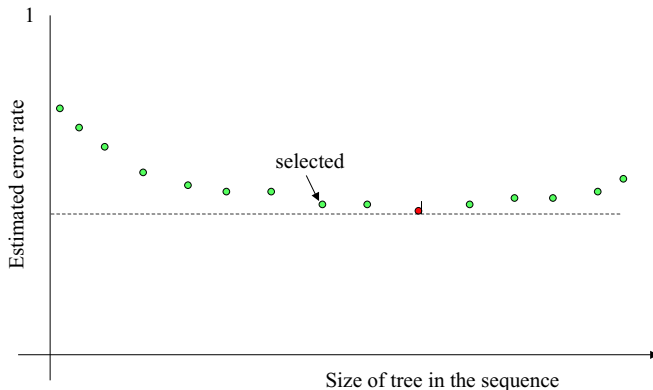
This is an *estimate* of the true error rate $R^*(T)$ of T .

The standard error of this estimate is

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

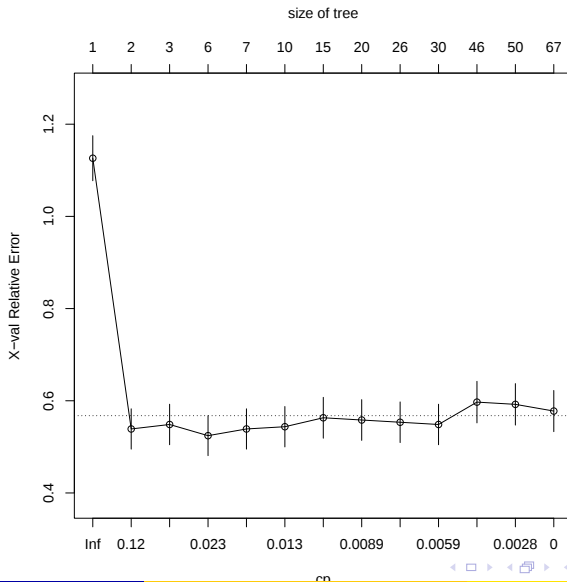
where n_{test} is the number of observations in the test set.

Selection of the final tree: the 1-SE rule

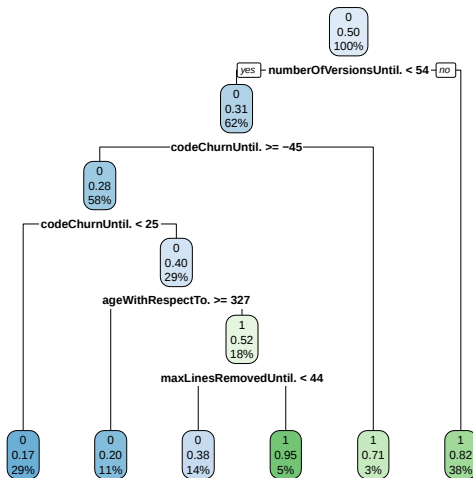


1-SE rule: select the smallest tree with R^{ts} within one standard error of the minimum.

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: train on folds 1-4, predict on fold 5

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

Cross-Validation

Second iteration:

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

Cross-Validation

Third iteration:

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

Cross-Validation

Fourth iteration:

fold	X	Y	$\hat{Y}$
1			$\hat{Y}^{(2)}$
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. different values for α).
- 2 Compute prediction error for each parameter setting.
- 3 Pick setting with lowest error.
- 4 Train with selected setting on complete data set.

ν -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 $c_1, \dots, c_m$ we proceed as follows.

- ➊ Divide the data into ν groups $G_1, \dots, G_\nu$.
- ➋ For each value c_i of C
 - ➊ For each group $j = 1, \dots, \nu$
 - ➊ Train with $C = c_i$ on all data *except* group G_j .
 - ➋ Predict on group G_j .
 - ➋ Compute the CV prediction error for $C = c_i$.
- ➌ Select the value c^* of C with the smallest CV prediction error.
- ➍ Train on the complete training sample with $C = c^*$

Selecting the best pruned subtree with cross-validation

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

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

Determine the grid of complexity values as follows:

$$c_1 = \sqrt{\alpha_1 \alpha_2} = 0,$$

$$c_2 = \sqrt{\alpha_2 \alpha_3},$$

$$c_3 = \sqrt{\alpha_3 \alpha_4},$$

$\dots$, c_k is the “representative” value for T_k .

$$c_{K-1} = \sqrt{\alpha_{K-1} \alpha_K},$$

$$c_K = \infty.$$

Selecting the best pruned subtree with cross-validation

Divide the data set into v groups $G_1, G_2, \dots, G_v$ and for each group G_j

- 1 Grow a tree on all data *except* G_j , and determine the smallest minimizing subtrees $T^{(j)}(c_1), T^{(j)}(c_2), \dots, T^{(j)}(c_K)$ for this reduced data set.
- 2 Compute the error of $T^{(j)}(c_k)$ ($k = 1, \dots, K$) on G_j .

From among $c_1, \dots, c_K$, determine the value c^* that minimizes cross-validation error, and select the tree $T(\alpha = c^*)$ from the original pruning sequence.

Regression Trees

We can also apply tree-based models to problems with numeric targets.

Three elements are necessary to specify a tree growing algorithm:

- ① A way to select a split at every non-terminal node.
- ② A rule for determining when a node is terminal.
- ③ A rule for assigning a predicted value $\hat{y}(t)$ to every terminal node t .

Prediction Rule for Regression Trees

In leaf nodes, we predict the average target value of all cases falling into that node.

$$\hat{y}(t) = \bar{y}(t) = \frac{1}{N(t)} \sum_{i \in t} y_i,$$

where $N(t)$ is the number of cases falling into node t .

Predict the value of c that minimizes the residual sum of squares (RSS):

$$RSS(t) = \sum_{i \in t} (y_i - c)^2.$$

Exercise: show that $c = \bar{y}(t)$ minimizes RSS.

Splitting Rule

The mean squared error (MSE) of a tree T is given by:

$$R(T) = \frac{1}{N} \sum_{t \in \tilde{T}} \sum_{i \in t} (y_i - \bar{y}(t))^2$$

where N is the size of the learning sample.

The contribution of node t to the MSE of T is

$$R(t) = \frac{1}{N} \sum_{i \in t} (y_i - \bar{y}(t))^2,$$

so we can write

$$R(T) = \sum_{t \in \tilde{T}} R(t).$$

Splitting Rule

The best split s^* of t is that split which most decreases $R(T)$.

The decrease in $R(T)$ of a (binary) split s in node t is given by:

$$\Delta R(s, t) = R(t) - R(\ell) - R(r),$$

where ℓ and r denote the left and right child created by the split respectively.

Stopping and Pruning

Continue until all nodes are pure? Not likely!

Don't split node t if $N(t) < nmin$, where $nmin$ is some small number (e.g. $nmin = 5$).

Pruning is identical to cost-complexity pruning for classification problems, using cost function

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

Note that in classification problems $R(T)$ denoted the classification error on the training sample, whereas in regression problems $R(T)$ is the mean squared error on the training sample.

Bug Prediction Data of Eclipse Classes

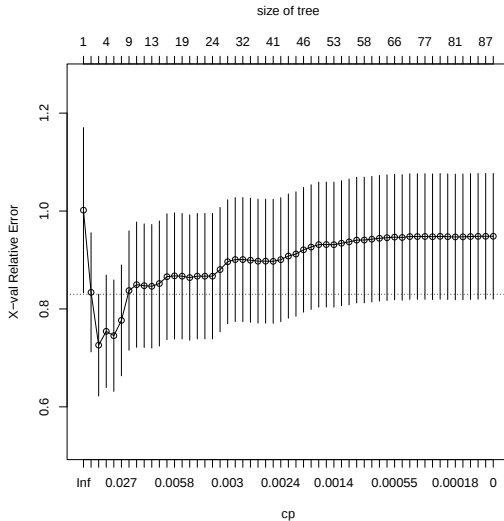
Change metrics:

numberOfVersionsUntil	numberOfFixesUntil	numberOfRefactoringsUntil
numberOfAuthorsUntil	linesAddedUntil	maxLinesAddedUntil
avgLinesAddedUntil	linesRemovedUntil	maxLinesRemovedUntil
avgLinesRemovedUntil	codeChurnUntil	maxCodeChurnUntil
avgCodeChurnUntil	ageWithRespectTo	weightedAgeWithRespectTo

Distribution of number of bugs ($N = 997$):

bugs	0	1	2	3	4	5	6	7	8	9
count	791	138	31	15	8	2	4	3	3	2

Bug Prediction Regression Tree Pruning Sequence



Bug Prediction Pruned Regression Tree

Top: average number of bugs

Bottom: percentage of training examples

