1). grow a tree first

2) prune it

cost complexity pruning

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

$|T|$ = number of leaves in T

for each node, check how much the error increase (by e) and how many leaves (n) would disappear

$$e - \alpha n = 0$$

$$\alpha = \frac{e}{n}$$

Select node with the smallest ratio $\frac{e}{n}$

its subtree gets pruned

Repeat that for the pruned tree.

For $\alpha = 0$: no pruning

α very large: only root will be left

How to select a good value of α?

1) check on the testing set

2) use K-fold cross-correlation: We train the tree as usually, on full training data.

To select a parameter $\alpha \in \{\alpha_1, \alpha_2, \dots, \alpha_r\}$, do the following:

divide the training set into K-folds:

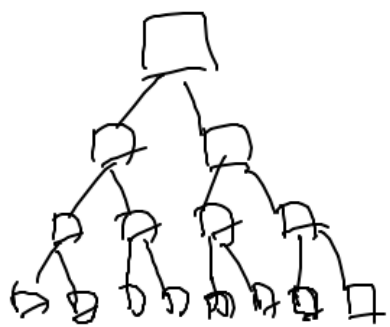
for each $k = 1, 2, \dots, K$:

- train a tree on the training data $\bigcup_{i \neq k} A_i$
- prune it using different values of α
- check its accuracy on A_k

Take α which corresponds to the best average accuracy

Why we first grow a large tree and then prune, instead of growing a smaller tree?

This is because of the greedy algorithm used for growing the tree.



Bootstrap aggregation (bagging)

A general method, but often used to decision trees.

1) Take B training sets
2) build a separate model for each of the training sets
3) construct the final model:
   - by taking the average of all B models (for regression problems)
   - -11- majority vote -11- (for classification problems)

Ad 1) The problem: we usually have a limited number of training samples
Suppose we have n training samples, we construct new training sets of size $\tilde{n}$ each in the following way:
- draw $\tilde{n}$ times (with replacement) a sample from the initial n training samples
  (usually $\tilde{n} = n$)

$\{1, 2, \ldots, n\}$ – set of (indices of) samples

we draw n times with replacement

E(number of different numbers that we are going to draw) =

$$X_i := \begin{cases} 1 & \text{if we draw „}i\text{''} \text{ at least once} \\ 0 & \text{if we never draw „}i\text{''} \end{cases}$$

$$P(X_i = 0) = \left(1 - \frac{1}{n}\right)^n \qquad P(X_i = 1) = 1 - \left(1 - \frac{1}{n}\right)^n$$

$$= E(X_1 + \ldots + X_n) = \sum_{i=1}^{n} E(X_i) = n \cdot \left(1 - \underbrace{\left(1 - \frac{1}{n}\right)^n}_{\downarrow n \to \infty \; \frac{1}{e}}\right) \approx \underbrace{n\left(1 - \frac{1}{e}\right)}_{0.632}$$

Ad 2) Pruning may be omitted. (In case of trees).

Bagging is a special case of "ensemble" methods.

0.59
0.65 ó

Contest: guess the weight of an ox (Galton)

Crowd wisdom (?)

Random forests

As in bagging, but at step 2):

2') build a model for each training set, using only a random sample of m from p predictors (each time drawn randomly)
($m = \lfloor \sqrt{p} \rfloor$ for classification trees and $m = \lfloor p/3 \rfloor$ for regression trees is suggested, but should be treated as a tuning parameter)

That way the trees will look more different one from another than in the bagging procedure.

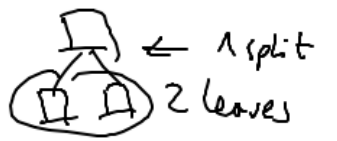
- Bagging, random forests usually have better accuracy than a single decision tree, but the simplicity (interpretability) of the model is lost.

Boosting (for regression tree)

} not only for trees

parameters:

- B – number of trees
- λ – shrinking parameter ($\rightarrow$ regularisation) λ – small $\Rightarrow$ B must be large ~~$B > 1$~~
  e.g. $\lambda = 0.01$
- d – number of splits in each tree ($d+1$ leaves)



1) Set $\hat{f}(x) = 0$ and $r_i = y_i$ for i ~~in the training set~~ $\in \{1, \dots, N\}$ ($i = 1, \dots, N$, N – number of samples)

2) for $b = 1, 2, \dots, B$:
   (a) fit a tree $\hat{f}^{(b)}$ with d splits to the training data $(X_i, r_i)_{i=1,\dots N}$
   (b) update $\hat{f}$:
   $$\text{new } \hat{f}(x) = (\text{old } \hat{f}(x)) + \lambda \hat{f}^{(b)}(x)$$
   (c) update the residuals:
   $$(\text{new } r_i) = (\text{old } r_i) - \lambda \hat{f}^{(b)}(x_i)$$

3) The final model is the last $\hat{f}$.

$(b=1) \Rightarrow$ e.g. $r_i = y_i = 3.5$, $\hat{f}^{(1)}(x_i) = 3.2$, new $r_i = 0.3$

$(b=2)$ fit to e.g. $(x_i, 0.3)$

$\hat{f}^{(2)}(x_i) = 0.4$

$r_i = -0.1$

for classification: e.g. AdaBoost