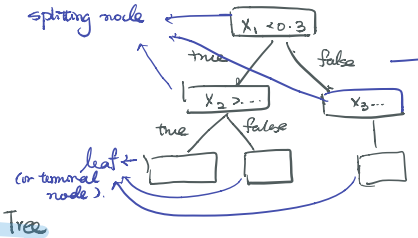
07 Trees



→ for now we only focus on **binary split**. because it's more interpretable and faster to compute and it can also replicate a multi-split tree using a series of binary splits.

(1) Regression Tree

$$\hat{Y} = \hat{f}(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \longrightarrow X \text{ is categorised in } R_1, R_2 \dots R_m.$$

where the choice of c_m depends on **loss function**: $\sum_{m=1}^{M} \sum_{i: x_i \in R_m} L(y_i, c_m) = \sum_{i=1}^{n} \sum_{m=1}^{M} L(y_i, c_m) I(x_i \in R_m)$.

e.g. **square loss**: $\sum_{m=1}^{M} \sum_{i: x_i \in R_m} (y_i - c_m)^2 = \sum_{i=1}^{n} \sum_{m=1}^{M} (y_i - c_m)^2 I(x_i \in R_m)$

minimising it $\Rightarrow \sum_{m=1}^{M} \sum_{i=1}^{n} 2(y_i - c_m) I(x_i \in R_m) = 0 \Rightarrow \sum_{i=1}^{n} \sum_{m=1}^{M} y_i I(x_i \in R_m) = \sum_{i=1}^{n} \sum_{m=1}^{M} c_m I(x_i \in R_m) \Rightarrow \sum_{m=1}^{M} \sum_{i=1}^{n} y_i I(x_i \in R_m) = \sum_{m=1}^{M} c_m \sum_{i=1}^{n} I(x_i \in R_m)$

$\Rightarrow c_m = \dfrac{\sum_{i=1}^{n} y_i I(x_i \in R_m)}{\sum_{i=1}^{n} I(x_i \in R_m)} = \dfrac{1}{N_m} \sum_{i=1}^{n} y_i I(x_i \in R_m)$ for $m = 1, 2, \dots M$ $\Rightarrow$ sample mean in each region.

absolute loss: $\sum_{m=1}^{M} \sum_{i: x_i \in R_m} |y_i - c_m| = \sum_{m=1}^{M} \sum_{i=1}^{n} |y_i - c_m| I(x_i \in R_m)$.

minimising it give c_m = sample median in each region

steps:

(1) Deciding on splits by using a **greedy approach**

i.e. at each splitting node we choose to use the ①best predictor and ②threshold to give the smallest loss.

Let R_k be the subset of predictors data represent by the k^{th} split, then in a binary split,

$$R_{k1}(j, s) = \{X_i : X_{ij} < s\} \text{ and } R_{k2}(j, s) = \{X_i : X_{ij} > s\}$$

k^{th} split $X_j \le s$ (true / false) → R_{k1}, R_{k2}

$\Rightarrow$ using square loss.

$$\sum_{x_i \in R_{k1}(j,s)} (y_i - c_1)^2 + \sum_{x_i \in R_{k2}(j,s)} (y_i - c_2)^2$$

Minimising it,

$$\min_{j,s} \left\{ \min_{c_1} \sum_{x_i \in R_{k1}(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_{k2}(j,s)} (y_i - c_2)^2 \right\}.$$

As sample mean minimises it

$$\min_{j,s} \left\{ \sum_{x_i \in R_{k1}(j,s)} (y_i - \bar{y}_{R_{k1}})^2 + \sum_{x_i \in R_{k2}(j,s)} (y_i - \bar{y}_{R_{k2}})^2 \right\}$$

i.e. we need try a few predictor (X_j) and a few threshold values (s) to find the best predictor and threshold to do the split at each split.

(2) Then decide on tree size.

Let T_0 be the full tree (i.e. only a few observations in each leaf). Then if $T \subset T_0$, T is a subtree.

We decide on the tree size by **pruning** the full tree T_0 that minimises:

cost complexity: $C_\alpha(T) = \underbrace{\sum_{m=1}^{|T|} N_m Q_m(T)}_{Q(T)} + \alpha |T|$ for a given α.

$|T|$ → number of leaves of tree T (penalty on the size of tree).

$\frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2$: average square loss.

$Q(T)$: total loss

$\Rightarrow$ $\alpha = 0$, $T \to T_0$

$\alpha \to \infty$, $T \to$ root tree (no splits).

$\Rightarrow C_\alpha(T) = Q(T) + \alpha |T|$

↳ determined by CV

how to choose the tree that minimises $C_\alpha(T)$?

(3) How to prune an optimal tree for a given α:

(1) Grow a full tree T_0, compute $C_\alpha(T_0) = Q(T_0)$ and set $\alpha_0 = 0$

(2) Prune off the weakest terminal split from T_0 to get T_{α_1}:

$$T_{\alpha_1} = \operatorname{argmin}_{T \subset T_0, |T_0| - |T| = 1} Q(T) - Q(T_0).$$ i.e. remove the split with the least loss reduction

and set $\alpha_1 = Q(T_{\alpha_1}) - Q(T_0)$ $\Rightarrow$ improvement of error.

(3) Prune off the weakest terminal split from T_{α_1} to get T_{α_2}:

$$T_{\alpha_2} = \operatorname{argmin}_{T \subset T_0, |T_0| - |T| = 1} Q(T) - Q(T_{\alpha_1})$$

and set $\alpha_2 = Q(T_{\alpha_2}) - Q(T_{\alpha_1})$

(4) Repeat until the root tree is left

$\Rightarrow$ This gives a sequence of trees $\{T_0, T_{\alpha_1}, \dots\}$ with increasing $0 \le \alpha_1 \le \alpha_2 \le \dots$

$\Rightarrow$ then there must be an optimal tree T_{α_k} in the sequence $\{T_0, T_{\alpha_1}, T_{\alpha_2}, \dots\}$ with α_k being the smallest one that is $\ge \alpha$.

proof: $C_\alpha(T_{\alpha_k}) = Q(T_{\alpha_k}) + \alpha |T_{\alpha_k}| = Q(T_0) + \alpha_1 + \alpha_2 + \dots + \alpha_k + \alpha(|T_0| - k)$ since $Q(T_0) - Q(T_{\alpha_1}) = \alpha_1$, $Q(T_{\alpha_1}) - Q(T_{\alpha_2}) = \alpha_2, \dots$

as $k \uparrow$, $+\alpha_k$ and $-\alpha$ $\Rightarrow C_\alpha(T_{\alpha_k})$ will start to increase once $\alpha_k > \alpha$ so we stop when $\alpha_k \ge \alpha$

⇒ it's equivalent to growing the strongest terminal node each time, which gives a sequence of optimals trees $\{T_0, T_1, \dots\}$ with decreasing α ⇒ so we choose T_{d+1} where $d+1$ is the first one that is $\neq d$.

note: tree size controls the bias-variance tradeoff

- overly large tree → complex/overfitted model (low bias and high variance)
- too small tree → over-simplify/underfitted model (high bias and low variance).

(2) Classification Trees

$$\hat{Y} = \hat{f}(x) = \sum_{m=1}^{M} k(m)\, I(x \in R_m) \longrightarrow X \text{ is categorized in } R_1, R_2 \dots R_m.$$

where Y has K levels/classes; $k(m)$ is the most commonly occurring class in R_m.

define $\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$, $k = 1, 2, \dots K$ ⇒ proportion of observation in R_m that belong to level k.

(m: region, k: class)

⇒ $k(m)$ can be written as $\arg\max_k \hat{p}_{mk}$ i.e. the most common class in region m.

classification tree uses different loss function to prune/grow trees: e.g.

Misclassification error:

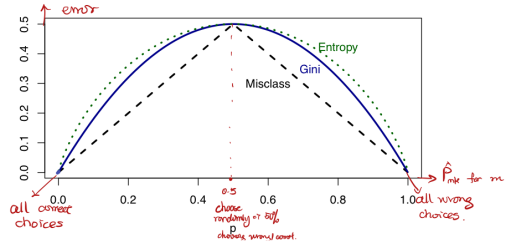
$$Q_m = \frac{1}{N_m} \sum_{x_i \in R_m} I\{y_i \neq k(m)\} = 1 - \max_k \hat{p}_{mk}. \text{ i.e. proportion of observation that are not in the most common class}$$

$$Q_m = 1 - \max(\hat{p}_m, 1-\hat{p}_m) \text{ for 2 classes situations}$$

Gini index:

$$Q_m = \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}) \longrightarrow \text{if } \hat{p}_{mk} \text{ is close to 1 or 0 then Gini} \to 0.$$

$$Q_m = 2\hat{p}_m(1-\hat{p}_m) \text{ for 2 classes situations}$$



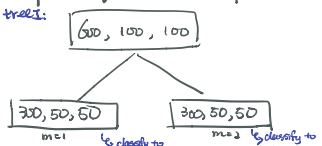
Cross-entropy (binomial deviance):

$$Q_m = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \longrightarrow \text{numerically similar to Gini.}$$

$$Q_m = -\hat{p}_m \log \hat{p}_m - (1-\hat{p}_m)\log(1-\hat{p}_m)$$

example: given a sample with 600, 100, 100 in 3 classes:

tree 1:

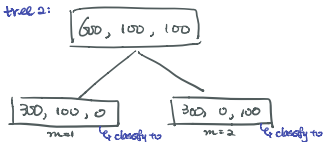


$N_1 = 400$, $N_2 = 400$

$P_{11} = \frac{300}{400} = 0.75$, $P_{12} = \frac{50}{400} = 0.125$, $P_{13} = \frac{50}{400} = 0.125$

$P_{21} = \frac{300}{400} = 0.75$, $P_{22} = \frac{50}{400} = 0.125$, $P_{23} = \frac{50}{400} = 0.125$

tree 2:



$N_1 = 400$, $N_2 = 400$

$P_{11} = \frac{300}{400} = 0.75$, $P_{12} = \frac{100}{400} = 0.25$, $P_{13} = \frac{0}{400} = 0$

$P_{21} = \frac{300}{400} = 0.75$, $P_{22} = \frac{0}{400} = 0$, $P_{23} = \frac{100}{400} = 0.25$

total Misclassification rate:

Tree 1: $400(1-0.75) + 400(1-0.75) = 200$ Tree 2: $400(1-0.75) + 400(1-0.75) = 200$

total Gini loss:

Tree 1: $400[1-0.75)0.75 + (1-0.125)0.125 + (1-0.125)0.125] \approx 325$ Tree 2: $400[(1-0.75)0.75 + (1-0.25)0.25 + 1.0] = 300$

note:
- some types misclassification are worse than other types (e.g. false positive or false negative).
  ⇒ we can adjust loss expression Q_m to upweight some types of losses relative to others.
- we can also use trees for feature selection.
- nice properties.

Speed: Running through the possible splits is fairly fast, and there are efficient algorithms to do this. Trees scale well for large n, compared to other approaches.

Interpretability: It is very easy to see what variables are driving the model, to make interpretations and to explain to a less numerate person.

- Problems:

In a word - **ACCURACY**.

It turns out that even for large dataset, decision trees give good but not great accuracy. The main reasons are

Handles all types of variables well: We do not have to massage ordinal and categorical variables into continuous variables, as with some other approaches. All types of variables can be handled in the single framework.

Treatment of missing variable observations: For a given split, it is (often) possible to find splits using other variables that replicate the main split fairly well. These are referred to as **"surrogate splits"**. If the observation for the main split variable is missing, we can use a surrogate split to assign the appropriate direction to an observation, greatly improving the ability to make predictions when observations are missing.

Robustness: Some modelling approaches will give extreme predictions for new response observations that lie outside the original data domain. By contrast trees give piecewise constant predictions, so will be very stable near boundaries. Similarly the fits are relatively insensitive to outliers (of the response variable).

Insensitivity to predictor transformations: A monotonic transformation of a continuous predictor variable will not affect the tree in any way (since all we do is separate high and low values), so we need not worry about scaling or transforming our data.

Nonparametric: Besides some smoothness, the model does not assume any function form for the underlying model structure. In particular, given enough data a decision tree can approximate the true model structure to arbitrary accuracy.

Curse of dimensionality: Since splits use just one variable at a time and ignore the rest, trees are relatively insensitive to extra dimensionality. Predictions will be almost as good, even if you add extra nonsense variables.

① ▸ Piecewise constant structure is probably not true in "reality".
② ▸ It struggles to fit main effects.
③ ▸ Model complexity can grow only slowly with sample size (see below). ⇒ adding a split would require twice the amount of data.
④ ▸ The greedy procedure does not guarantee optimal splits (see below). ⇒
⑤ ▸ Tree stability can be an issue (see below). ⇒ cause a different split even there's a slight change in data
⑥ ▸ Categorical variables with many levels can distort trees (see below). ⇒ the tree stop growing before it has more meaning splits

Note that the first two points suggest that trees will struggle when linear models will do well. Indeed, the linear model is almost the "chief enemy" of a decision tree.

* we can overcome some of these problems with the following methods:

(3) **Bagging**
- many bootstrap samples.
- grow but dont prune (high variance in each tree)
- ↓ variance due to average.
- but trees are correlated

methods:

grow deep, unpruned trees →
- regression tree: average the prediction.
- classification tree: majority vote from all trees.

steps:

(1) Generate bootstrap sample $\mathcal{Z}^*_b = \{(x_{b1}, y_{b1}), \dots (x_{bn}, y_{bn})\}$, for $b = 1, 2, \dots B$

(2) Use the same prediction method on these bootstrap dataset to give $\hat{f}_1(x), \dots \hat{f}_B(x)$.

(3) The bagged model prediction is:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x). \text{ or majority vote.}$$

(4) **Random Forest**
- many bootstrap samples
- grow but dont prune (high variance in each tree)
- ↓ variance due to average.
- de-correlate trees due to feature bagging.
- feature bagging: at each split consider only m randomly selected predictors → more randomness → double bagging.

steps:

(1) Generate bootstrap sample $\mathcal{Z}^*_b = \{(x_{b1}, y_{b1}), \dots (x_{bn}, y_{bn})\}$, for $b = 1, 2, \dots B$

(2) For each bootstrap sample $\mathcal{Z}^*_b$, grow a random forest tree T_b by repeating:

2.1 Select m predictors randomly
2.2 Pick the best predictor and split-threshold among the m predictors
2.3 Split the node into two nodes
2.4 Repeat until minimum node size n_{min} is reached

} greedy approach among m predictors.

(3) Output the ensemble trees $\{T_b\}_1^B$

(4) The bagged model prediction is:

$$\hat{f}^B_{rf}(x) = \frac{1}{B}\sum_{b=1}^{B}T_b(x) \quad \text{for regression tree.} \qquad m = p/3$$

$$\hat{C}^B_{rf}(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B \text{ for classification tree} \qquad m = \sqrt{p}.$$

(5) **Boosting**
- one original tree
- trees grow sequentially
- each tree is fitted on the residuals
- improve the prediction for the observations that the model a poor at estimating
- slow-learner: each tree is small.

these are many algorithms but the main difference is the method of trainy points weighting.

Adaboost:

Methods:

- Produce a collection of classifier $G_m(x_i) \in \{-1, 1\}$ for $m = 1, 2, \dots M$. These are each produced from the training data, re-weighting each time.
- Then the error is:

$$\overline{err} = \frac{1}{n} \sum_i I\{y_i \neq G_m(x_i)\} \quad \text{for classifier } m.$$

- Then combine these to obtain a weighted majority vote:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$ ← should be larger for models that are more accurate

⇒ Trainy sample → weighted sample → weighted sample → ..... → weighted sample
↓ $G_1(x)$ ↓ $G_2(x)$ ↓ $G_3(x)$ ↓ $G_M(x)$. ⇒ $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

Algorithm:

1. Initialise weights $w_i = 1/n$ for all i
2. for $m = 1, \dots M$
   - 2.1 Fit classifier $G_m(x)$ to the training data, with weights w_i
   - 2.2 Compute weighted error,
   $$err_m = \frac{\sum_{i=1}^{n} w_i I\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$
   - 2.3 Set $\alpha_m = \log\{(1 - err_m)/err_m\}$ ⇒ $err_m \uparrow$ ⇒ $\log\{\frac{1-err_m}{err_m}\} \downarrow$ ⇒ $\alpha_m \downarrow$ ⇒ less/more weight for poor/good models
   - 2.4 Update w_i with $w_i \exp\{\alpha_m I\{y_i \neq G_m(x_i)\}\}$ for each observation. ⇒ increase the weight of observations that the model misclassified
3. Output $G(x) = \text{sign}\left[\sum_m \alpha_m G_m(x)\right]$.

note:

- Adaboost doesn't specify what classifiers to use to create $G_m(x)$ at each step rather it reweights the observations each time ⇒ so we can use different models at each stage, such as decision trees, simple spline models etc.
- Adaboost can take a class of weak classifiers and produce an ensemble model the predicts accurately

Forward Stagewise Modelling

Methods:

Suppose $f(x) = \sum_{m=1}^{M} \beta_m \underbrace{b(x; \gamma_m)}_{\text{basic function}}$, where β_m and γ_m are determined to minimise:

$$\sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

forward stagewise adds a basic function one at a time.

Algorithm:

1. Initialise $f_0(x) = 0$
2. For $m = 1, \dots M$,
   - 2.1 Compute $(\beta_m, \gamma_m) = \text{argmin}_{\beta, \gamma} \sum_{i=1}^{n} L\{y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)\}$.
   - 2.2 Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
3. $f(x) = \sum_{m=1}^{M} \beta_m f_m(x)$

note:

- If γ represent the choice of the number of splits and level for each terminal node, Adaboost is equivalent to (i.e. $\alpha_m = 2\beta_m$) forward stagewise modelling with exponential loss function:

$$L\{y_i, f(x_i)\} = \exp\{-y_i f(x_i)\} = \begin{cases} \exp\{1\} & \text{if misclassified} \\ \exp\{-1\} & \text{if corrected.} \end{cases}$$

- So we can:
  - build stagewise models for both classification and regression.
  - choose any loss function we like
  - Not be restricted to classifiers taking values in $\{-1, 1\}$.
  - modify the basis on which new terms are added to the model (e.g. shrinkage).

For regression:

$(Y_i - f(x_i))^2$ (square loss),

$|Y_i - f(x_i)|$ (absolute loss),

$h(Y_i - f(x_i))$, where $h(z) = \begin{cases} z^2 & \text{if } |z| < \delta \\ 2\delta(|z| - \delta/2) & \text{if } |z| \geq \delta \end{cases}$ (Huber loss)

For classification:

$I[Y_i \neq \text{sign} f(x_i)]$ (misclassification rate).

$-\left[I(Y_i = 1) \log\left\{\frac{e^{2f(x)}}{1+e^{2f(x)}}\right\} + I(Y_i = -1) \log\left\{\frac{1}{1+e^{2f(x)}}\right\}\right]$ (binomial loss)

$\exp\{-Y_i f(x_i)\}$ (exponential loss)

$\left\{Y_i - \frac{e^{2f(x)}}{1+e^{2f(x)}}\right\}^2$ (squared loss)

boosting with forward stagewise.

1. Initialise $f_0(x) = 0$
2. For $m = 1, \ldots M$,
   - 2.1 Compute $(\beta_m, \gamma_m) = \text{argmin}_{\beta,\gamma} \frac{1}{n} \sum_{i=1}^{n} L\{Y_i, f_{m-1}(x_i) + \beta b_m(x_i; \gamma)\}$. → can set new basic function each time.
   - 2.2 Set $f_m(x) = f_{m-1}(x) + \beta_m b_m(x; \gamma_m)$.
3. $f(x) = \sum_{m=1}^{M} \beta_m f_m(x)$