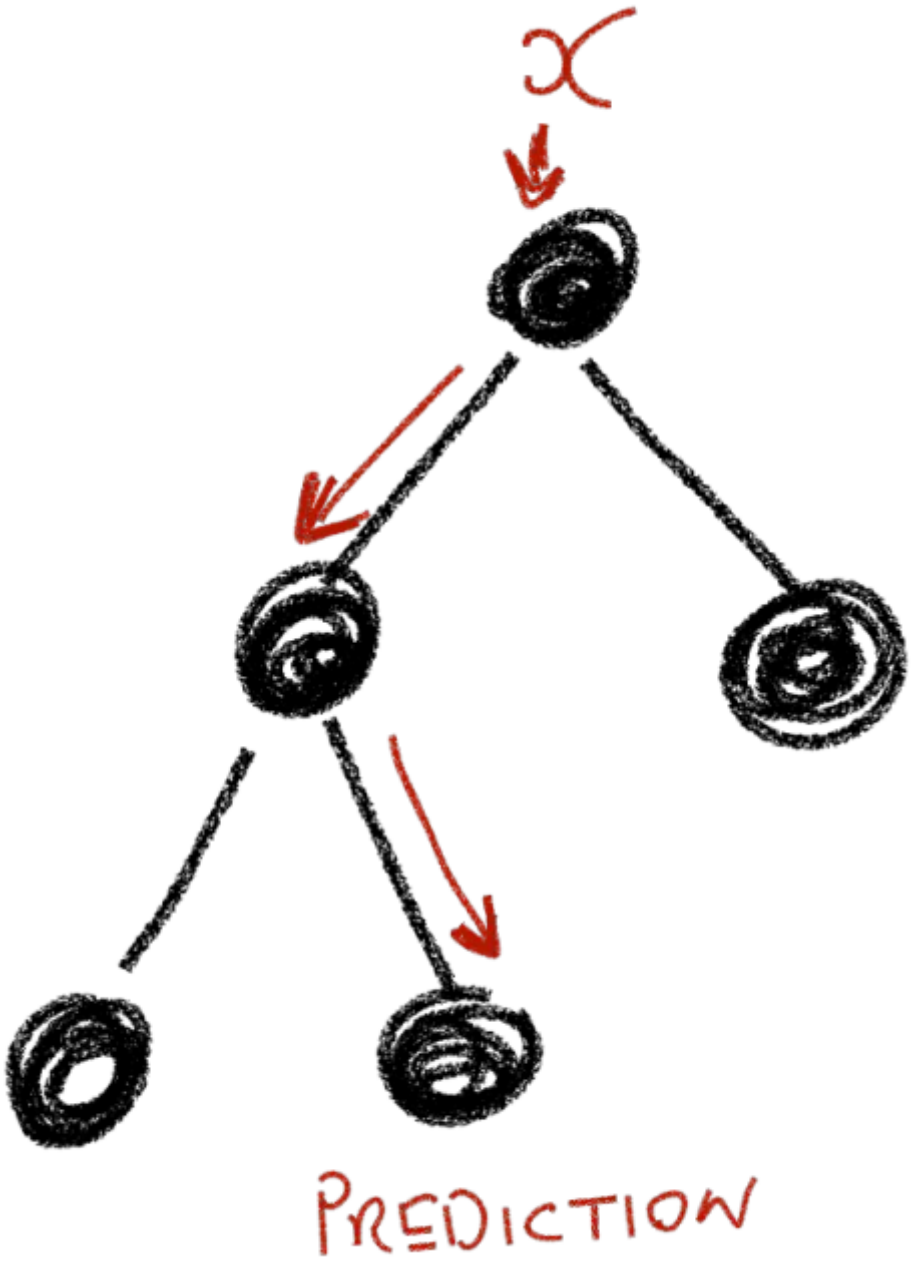
Decision Trees

A #decision-tree is a tree-structured prediction model composed of terminal[1] nodes and non-terminal nodes. Non-terminal nodes have two or more children and implement a routing function, whereas leaf nodes have no children and implement a prediction function. There are no cycles, and all nodes have at most one parenct except the root node.

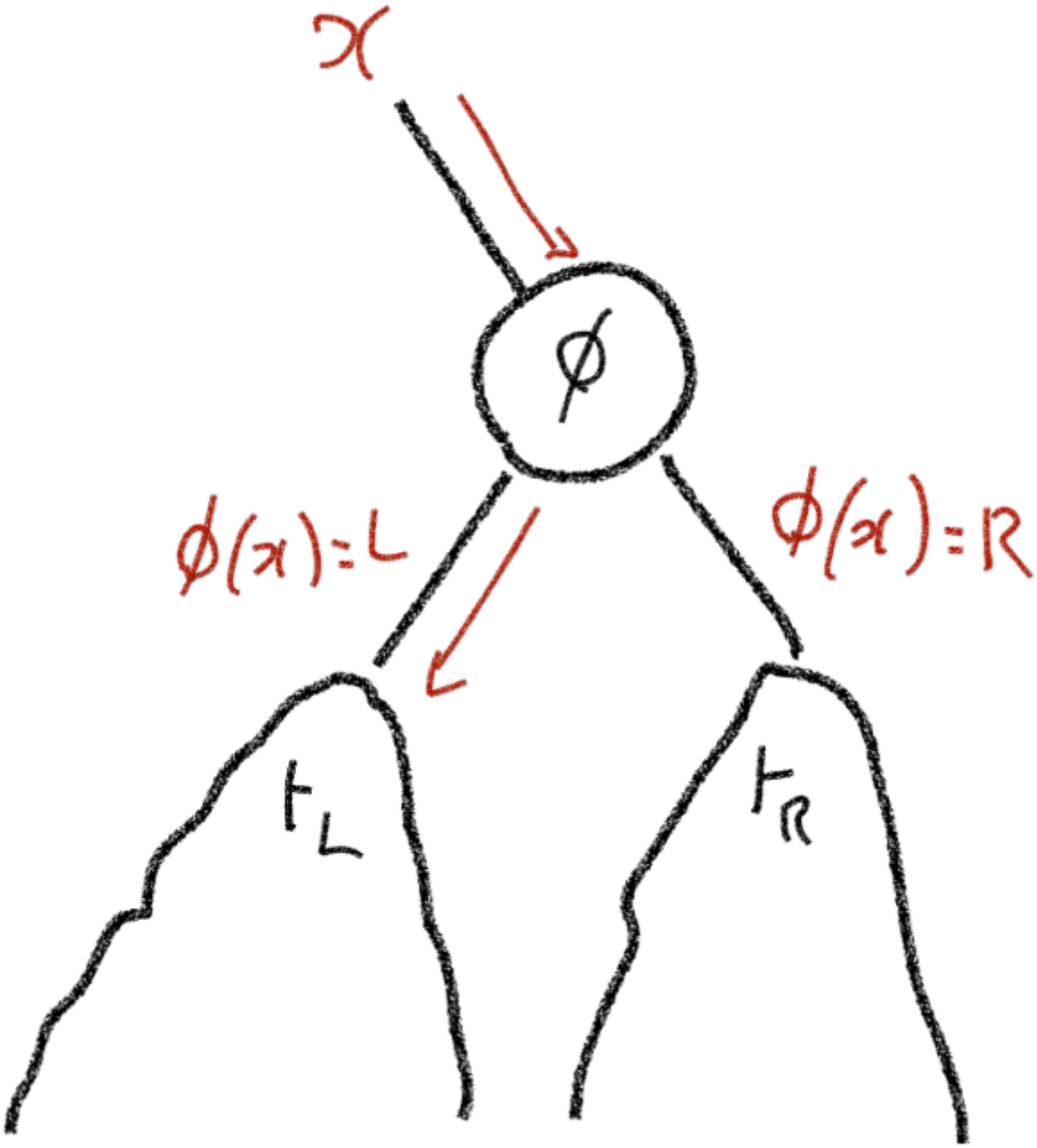
How #Decision-Trees Work

A #decision-tree takes an input $x \in X$ and routes it through its nodes until it reaches a leaf node. In the leaf a prediction takes place.



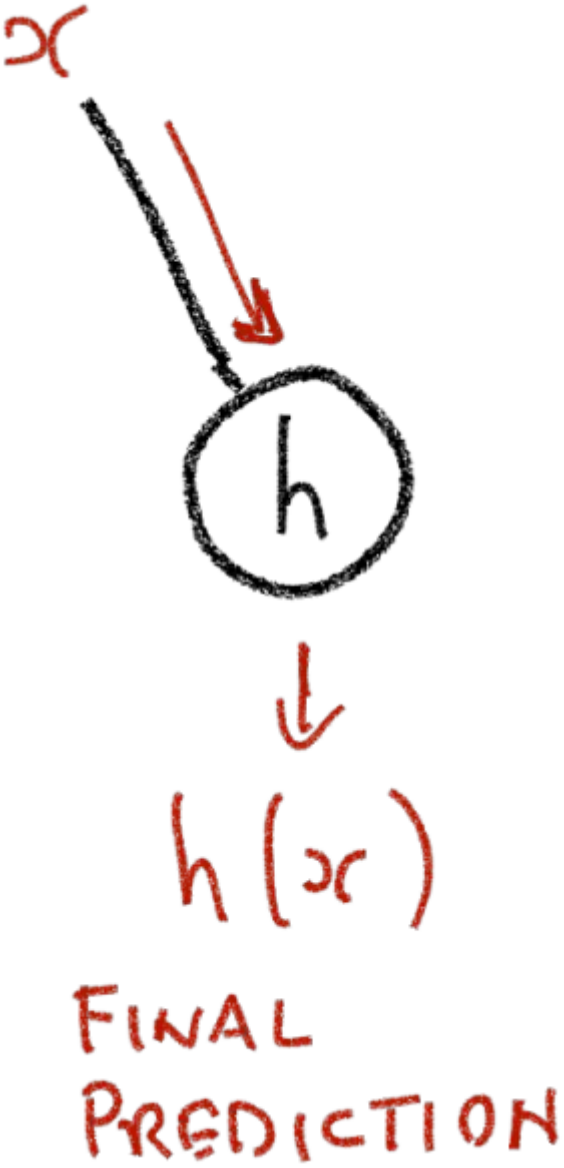
Non-Terminal Nodes

Each non-terminal node $Node(\phi, t_L, t_R)$ holds a routing function $\phi \in \{L, R\}^X$, a left child t_L and a right child t_R. When x reaches a node it will go to its left or right child depending on the value of $\phi(x) \in \{L, R\}$[2].



Terminal/Leaf Nodes

Each leaf node $Leaf(h)$ holds a prediction function $h \in F_{task}$[3]. Depending on the task we want to solve it could be $h \in Y^X$[4], $h \in \Delta(X)$[5], or something else. Once x reaches a leaf node the final prediction is given by $h(x)$.



Inference

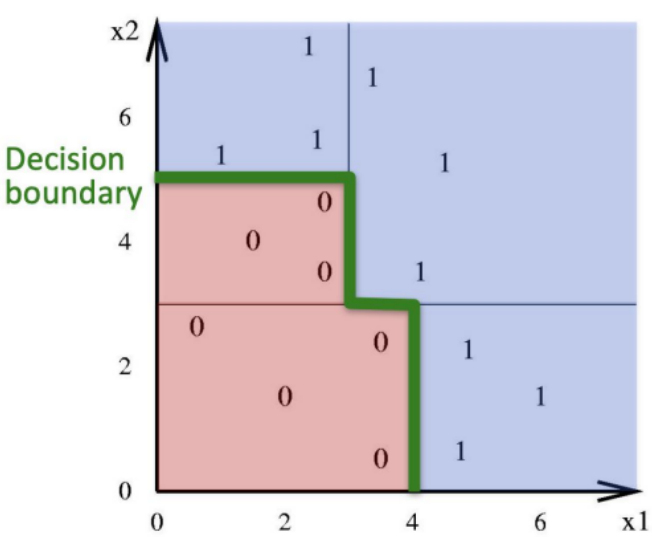
f_t is the function returning the prediction for the input $x \in X$ according to the decision tree t. It is recursively defined as $f_t(x) = h(x)$ if $t = Leaf(h)$, $f_{t_{\phi(x)}}(x)$ if $t = Node(\phi, t_L, t_R$

Decision Boundaries

#Decision-trees divide the feature space into axis parallel (hyper-)rectangles, where each rectangular region is labelled with one label.

Example

For a binary classification problem, we could have something like the following:



Learning Algorithm

Given a training set $D_n = \{z_1, \ldots, z_n\}$ find f_{t^*} where $t^* \in argmin_{t \in T} E(f_t; D_n)$ and T is a set of decision trees.

The optimisation problem is easy[6] as long as we don't impose constraints, otherwise it could be NP-hard. A solution can be found using a simple, greedy strategy.

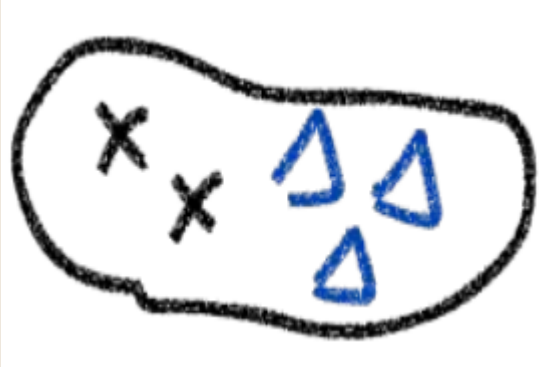
From now on we'll assume $E(f_t; D) = \frac{1}{|D|} \sum_{z \in D} l(f; z)$

Training is performed in batch mode.

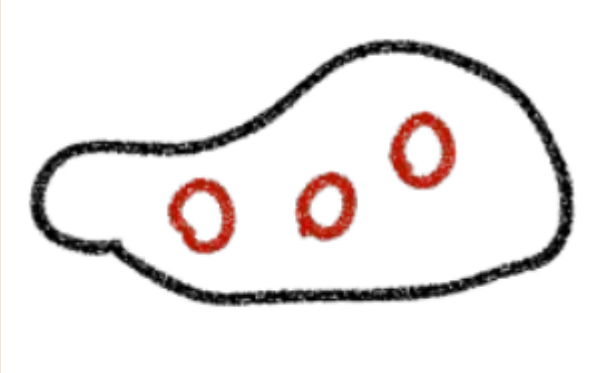
First, fix a set of leaf predictions $H_{leaf} \subset F_{task}$[7]. Then, fix a set of possible routing functions $\Phi \subset \{L, R\}^X$. Now, to "train" the model, we employ a recursive strategy that repetitively partitions the training set and decides whether to grow leaves or non-terminal nodes.

Impurity of a set

An impure group is a group containing multiple different classes, for example:



A group with minimum impurity is a group containing exactly one class, for example:



Growing a Leaf

Given $D = \{z_1, \ldots, z_m\}$ the training set reaching the current node:

- The optimal leaf predictor can be computed as $h_D^* \in argmin_{h \in H_{leaf}} E(h; D)$
- The optimal error value, also known as impurity measure, can be computed as $I(D) = E(h_D^*; D)$

If some criterion is met we grow a leaf $Leaf(h_D^*)$

Growing a Node

If no stopping criterion is met we have to find an optimal split function, $\phi_D^* \in argmin_{\phi \in \Phi} I_\phi(D)$
The impurity $I_\phi(D)$ of a split function ϕ given a training set D is computed in terms of the impurity of the split data: $I_\phi(D) = \sum_{d \in \{L,R\}} \frac{|D_d^\phi|}{|D|} I(D_d^\phi)$ where $D_d^\phi = \{(x, y) \in D; \phi(x) = d\}$

Impurity of a split function

The impurity of a split function is the lowest training error that can be attained by a tree consisting of a root and two leaves.

Growth in a Nutshell

$Grow(D) = Leaf(h^*_D)$ if the stopping criterion is met, $Node(\phi^*_D, Grow(D^*_L), Grow(D^*_R))$[8] otherwise. That is, if the stopping criterion is met we grow a leaf and terminate this recursive branch, but otherwise we create a node with split function ϕ^*_D and two children, each created by a further recursive call to the $Grow()$ function.

Split Selection

Sometimes the metric for choosing the best split function isn't the impurity of the split, which we'd like to minimise, but the information gain which we would rather maximise. This is given by $\Delta_\phi(D) = I(D) - I_\phi(D)$

> **Note**
>
> Information gain is non-negative, that is $\Delta_\phi(D) \geq 0$ for any $\phi \in \{L, R\}^X$ and any training set $D \subset X \times Y$

With this criterion the impurity will never increase for any randomly chosen split.

Leaf Predictions

The leaf prediction provides a solution to a simplified problem involving only data reaching the leaf. This solution can be an arbitrary function $h \in F_{task}$, but in practice we restrict it to a subset H_{leaf} of simple ones. The simplest predictor one can construct is a function returning a constant, for example a fixed class label.

> **Note**
>
> The set of all possible constant functions can be written as $H_{leaf} = \bigcup_{y \in Y} \{y\}^X$

Impurity Measures for Classification

For a classification $Y = \{c_1, \ldots, c_k\}$ and $D \subset X \times Y$, let $D^y = \{(x, y) \in D : y = y'\}$ denote the subset of training samples in D with class label y.

Consider, then, the following error function $E(F; D) = \frac{1}{|D|} \sum_{z \in D} l(f; z)$[9]

If $l(f; (x, y)) = 1_{f(x) \neq y}$ and $H_{leaf} = \bigcup_{y \in Y} \{y\}^x$, then the impurity measure is the classification error: $I(D) = 1 - max_{y \in Y} \frac{|D^y|}{|D|}$

If $l(f;(x,y)) = -log(f_y(x))$ and $H_{leaf} = \bigcup_{\pi \in \Delta(Y)} \{\pi\}^X$[10] then the impurity measure is the entropy: $I(D) = -\sum_{y \in Y} (\frac{|D^Y|}{|D|} log(\frac{|D^Y|}{|D|}))$

If $l(f;(x,y)) = \sum_{c \in Y} [f_c(x) - 1_{c=y}]^2$ and $H_{leaf} = \bigcup_{\pi \in \Delta(Y)} \{\pi\}^X$ then the impurity measure is the Gini impurity: $I(D) = 1 - \sum_{y \in Y} (\frac{|D^Y|}{|D|})^2$

Impurity Measures for Regression

For regression, $Y \subset \mathbb{R}^d$ and $D \subset X \times Y$.

If $l(f;(x,y)) = ||f(x) - y||_2$ and $H_{leaf} = \bigcup_{y \in Y} \{y\}^x$ then the impurity measure is the variance: $I(D) = \frac{1}{|D|} \sum_{(x,y) \in D} ||x - \mu_D||^2$[11]

Split/Routing Functions

The routing or split $\phi \in \{L, R\}^X$ determines whether a data point $x \in X$ should move left[12] or right[13]. The possible split functions are restricted to some predefined set $\Phi \subset \{L, R\}^X$ depending on the nature of the feature space. The prototypical split function for a d-dimensional input first selects one dimension then applies a 1-dimensional splitting criterion.

Discrete, Nominal Features

Assume discrete, nominal features[14] taking values in K. The split function can be implemented given a partition of K into K_R and K_L: $\phi(x) = L$ if $x \in K_L$, R if $x \in K_R$.

Finding the optimal split requires testing $2^{|K|-1} - 1$ bi-partitions.

Ordinal Features

Assume ordinal features[15] taking values in K. The split function can be implemented given a threshold $r \in K$: $\phi(x) = L$ if $x \leq r$, R if $x > r$.

If $|K| \leq |D|$, finding the optimal split requires testing $|K| - 1$ thresholds. Otherwise, it requires sorting the input values in D, where D is the training set reaching the node, and testing $|D| - 1$ thresholds.

Oblique

Sometimes it is convenient to split using multiple features at once. Such split functions work with continuous features and are called oblique, because they can generate oblique decision boundaries.

If $x \in \mathbb{R}^d$ then the split function can be implemented given $w \in \mathbb{R}^d$ and $r \in \mathbb{R}$. $\phi(x) = L$ if $w^T x \leq r$, R otherwise.

> **Note**
>
> This is harder to optimise.

Overfitting

Decision trees have a structure that is determined by the data, and as a result they are flexible and can easily fit the training set with a high risk of #overfitting . Standard techniques to improve generalisation apply also to #decision-trees [16]. A technique to reduce complexity after-the-fact is called #pruning . #Pruning of the #decision-tree is done by replacing a whole subtree with a leaf node.

Many kinds of "noise" can arise in the training data, for example:

- Some values of attributes are incorrect because of errors in the data acquisition process
- The example was labelled incorrectly
- Some attributes are irrelevant to the decision making process

Irrelevant attributes can result in overfitting the training data. If we have a large number of attributes we may find meaningless regularity in the data, so we should also remove irrelevant attributes[17].

Random Forests

Random forests are sets of decision trees. Each tree is typically trained on a #bootstrapped version of the training set[18].

Split functions are optimised on randomly sampled features, or are sampled completely at random[19]. This helps us obtain decorrelated decision trees. The final prediction of the forest is obtained by averaging the prediction of each tree in the set $Q = \{t_1, \ldots, t_T\}$:
$f_Q(x) = \frac{1}{T} \sum_{j=1}^{T} f_t(x)$

Algorithm

- For $b = 1$ to B
  - Draw a #bootstrap sample Z^* of size N from the training data
  - Grow a random-forest tree T_b from the #bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached
    - Select m variables at random from the p variables
    - Pick the best variable/split-point among the m variables
    - Split the node into two child nodes
- Output the set of trees $\{T_b\}_1^B$

To make a prediction at a new point x, in the case of #regression we evaluate the following: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$. In the case of #classification we should first let $\hat{C}_B(x)$ be the class prediction of the bth random-forest tree. Then, let $\hat{C}_{rf}^{B}(x) = majority\ vote\{\hat{C}_b(x)\}_1^B$.

---

1. leaf ↩
2. assuming binary trees ↩
3. typically constant ↩
4. classification/regression ↩
5. density estimation ↩
6. there are many solutions ↩
7. constant functions ↩
8. where $D_L^* = \{(x, y) \in D; \phi_D^*(x) = L\}$, or rather where L is the set of examples in the training set that fall on the left-hand side of the #decision-tree according to the split function ϕ_D^* ↩
9. average loss over the training set ↩
10. constant label distribution as leaf prediction ↩
11. where $\mu_D = \frac{1}{|D|}\sum_{(x,y)\in D} x$ ↩
12. $\phi(x) = L$ ↩
13. $\phi(x) = R$ ↩
14. e.g., colours, marital status, etc ↩
15. sortable, e.g., age, height, etc ↩
16. early stopping, regularisation, data augmentation, complexity reduction, ensembling, … ↩
17. this is a manual process, and is not always possible ↩
18. sampled with replacement, i.e. independent samples ↩
19. extremely randomised trees ↩