Lesson 2

Classification trees

Carlo Mannino
SINTEF Digital and University of Oslo

Outline

1. Classification and regression trees (CART)
2. The CART algorithm
3. A MILP formulation for CART

Classification problem

- Training data $(\mathbf{X}, \mathbf{y})$ with n observations $(\mathbf{x}_i, y_i)$, $i \in [n]$, each with p features $\mathbf{x}_i \in \mathbb{R}^p$ and class label $y_i \in [K]$.
- Partition the feature space into disjoint regions with homogeneous label.

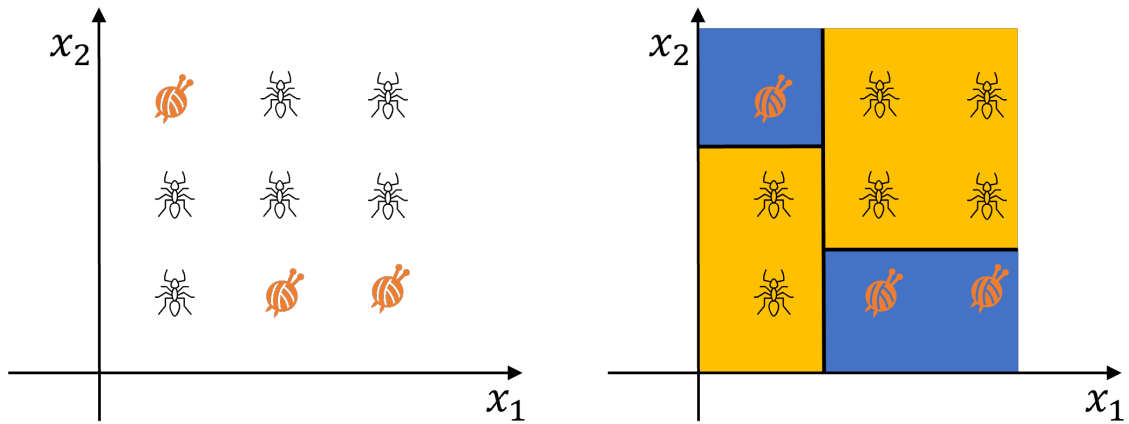


Figure: Each point is either labelled ladybug, or ant. The region is partitioned using only lines parallel to axis. Every region contains only ladybugs or ants.

Iris Data

From: Bertsimas, D. and Dunn, J., 2019. Machine learning under a modern optimization lens. Dynamic Ideas LLC.

- Another example: the Iris data.
- The feature space is partitioned by means of a *classification tree*.
- Classification trees correspond to greedy, recursive partition of the feature space

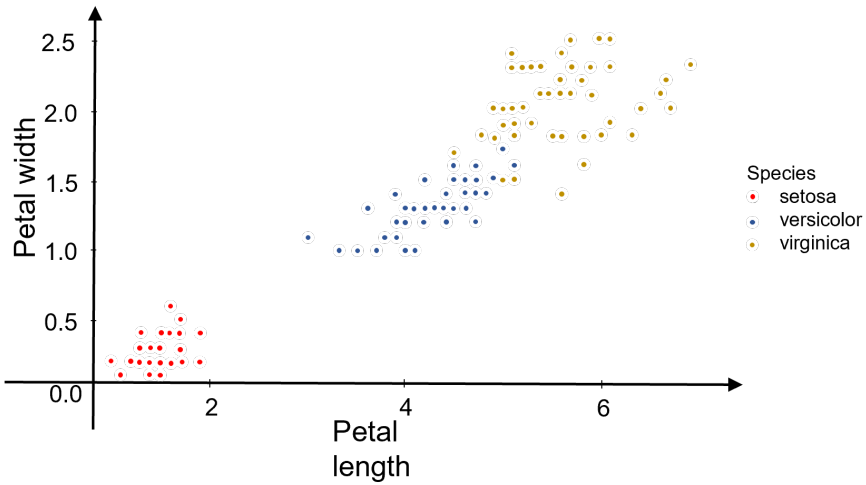


Figure: Iris Data.

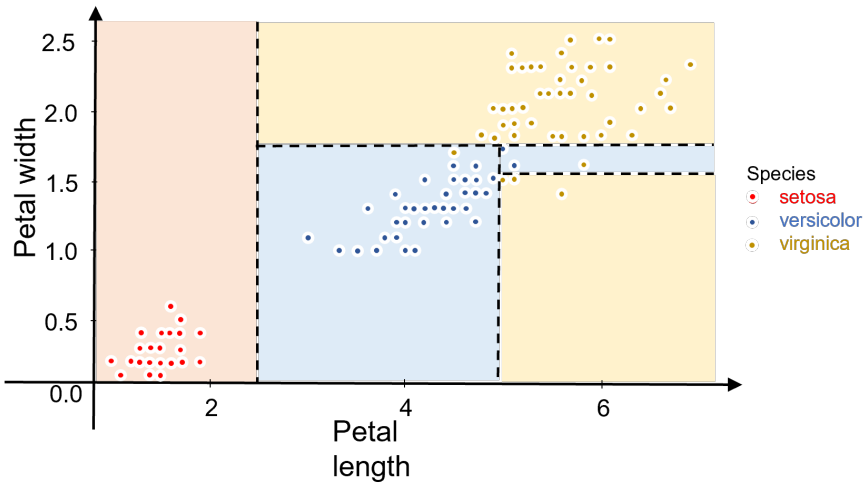
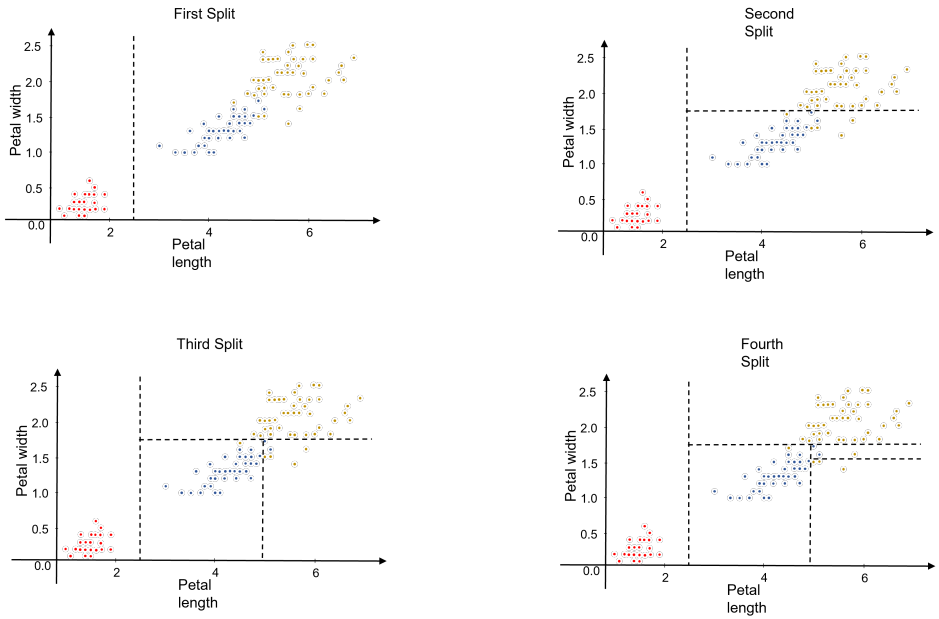
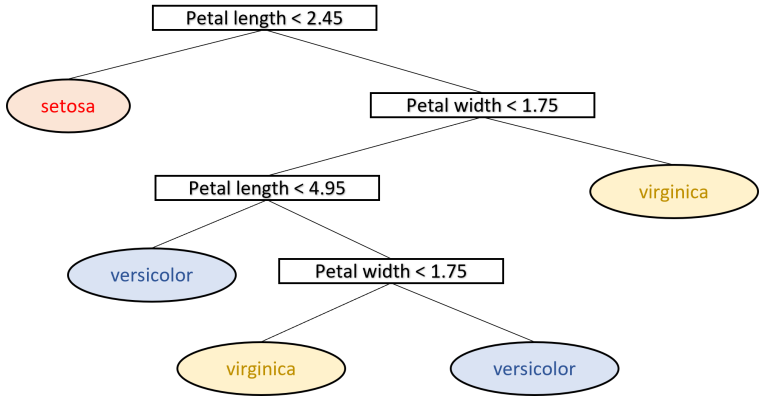


Figure: A partition of the Iris Data

Classification tree: example

- Each node of the tree is associated with a region. The regions of the final partition correspond to the leaves of the tree.





- Every non-leaf node has 2 children, corresponding to the two regions of a split.
- Regions are partitioned by hyperplanes parallel to the axis.

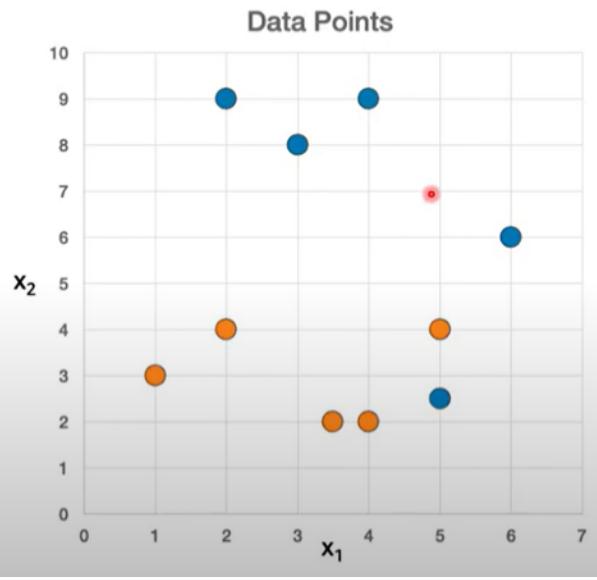
Features of decision trees

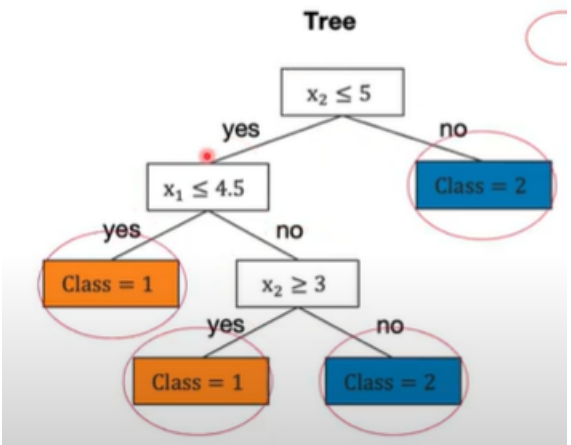
- Positive features
  - Easy to explain
  - Similar to human decision process
  - Easy to display graphically
  - Fit to handle qualitative predictors
- Negative features
  - Less predictive accuracy than other methods
  - Lack of robustness. Small changes in input data may produce big changes in the tree.

Another example

From Machine Learning University Channel on YouTube

x_1	x_2	y
3.5	2	1
5	2.5	2
1	3	1
2	4	1
4	2	1
6	6	2
2	9	2
4	9	2
5	4	1
3	8	2





- Ten observations. Two features, two labels.

The CART algorithm: impurity

- The CART (Classification And Regression Trees) is a greedy method to build a hierarchical partition of the feature space.
- The tree (and the partition) are built iteratively by bi-partitioning (at each iteration) an existing class.
- Regions are split so as to minimize the *impurity*, which is a measure of misclassification.
- For a region r with n^r data-points $(\mathbf{x}_i^r, y_i^r)$, $i \in [n^r]$, we let n_q^r be the number of points labelled by $q \in [K]$
- We let $p_q^r = n_q^r \backslash n^r$ be the probability of label $q \in [K]$ in region r.
- *Gini impurity* of region r: $i(r) = \sum_{q \in [K]} p_q^r (1 - p_q^r)$.
- *Entropy* of region r: $i(r) = -\sum_{q \in [K]} p_q^r \log_2 p_q^r)$.

Note: the impurity is 0 if $p_q^r = 1$ for a $q \in [K]$.

The CART algorithm: description

- At each iteration, CART selects a region r, a feature $j \in [p]$ and a threshold b_j.
- It creates a new partition by splitting region r into the two regions r^1 and r^2 separated by the hyperplane $x_j = b_j$.
- r^1 contains the n^1 data-points $d \in r$ with $x_{d,j} < b_j$. r^2 contains the n^2 data-points of $p \in r$ with $x_{d,j} \geq b_j$. Clearly $n^1 + n^2 = n^r$
- The average decrease of impurity is defined as

$$i(r) - \frac{n^1 i(r^1) + n^2 i(r^2)}{n^r}$$

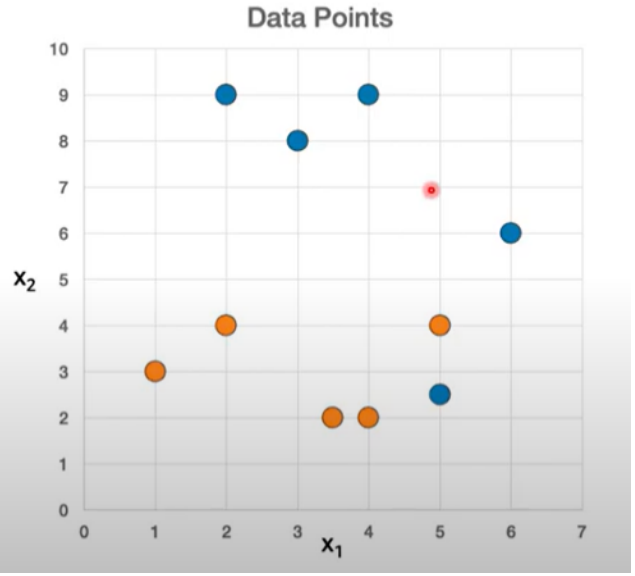
- At each iteration, CART selects the region, the feature and the threshold which maximises the average impurity decrease.

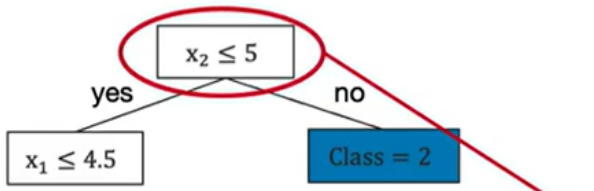
The CART algorithm: termination

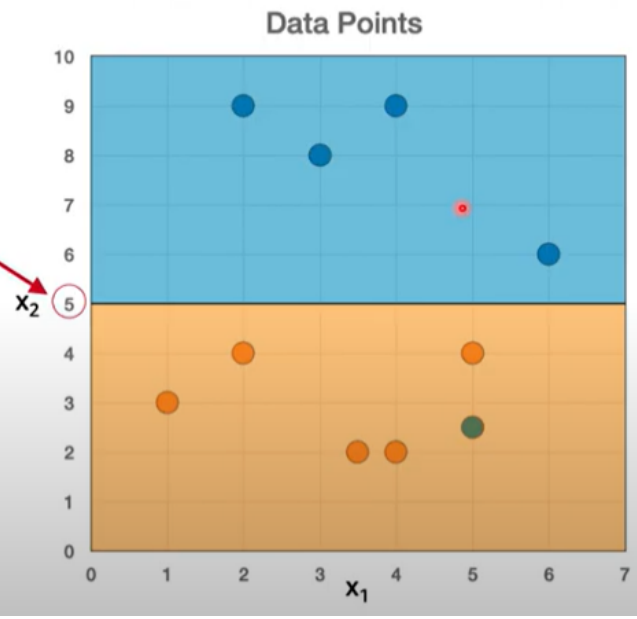
- There can be several termination criteria
  - Any split produces a region with too few data-points
  - Every region contains data-points with the same class label.
  - ... other
- After termination, the tree can be pruned to avoid over-fitting
- We want a tree which does not misclassify too much, and does not overfit too much.

An example from ML University channel

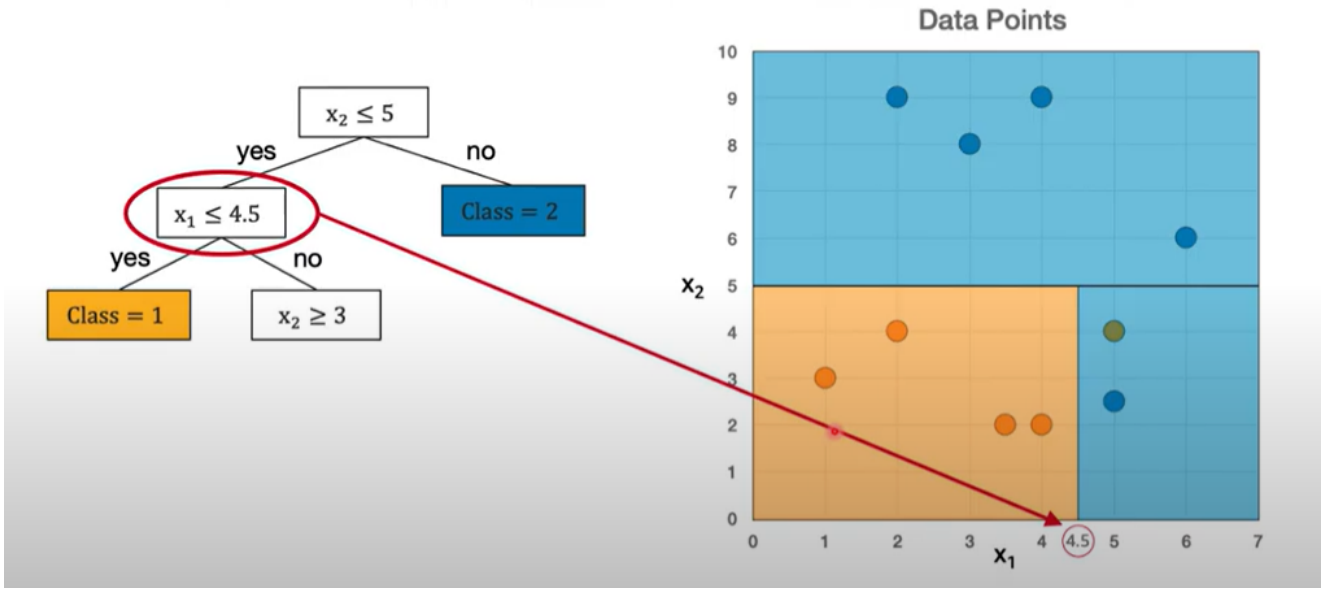
x_1	x_2	y
3.5	2	1
5	2.5	2
1	3	1
2	4	1
4	2	1
6	6	2
2	9	2
4	9	2
5	4	1
3	8	2



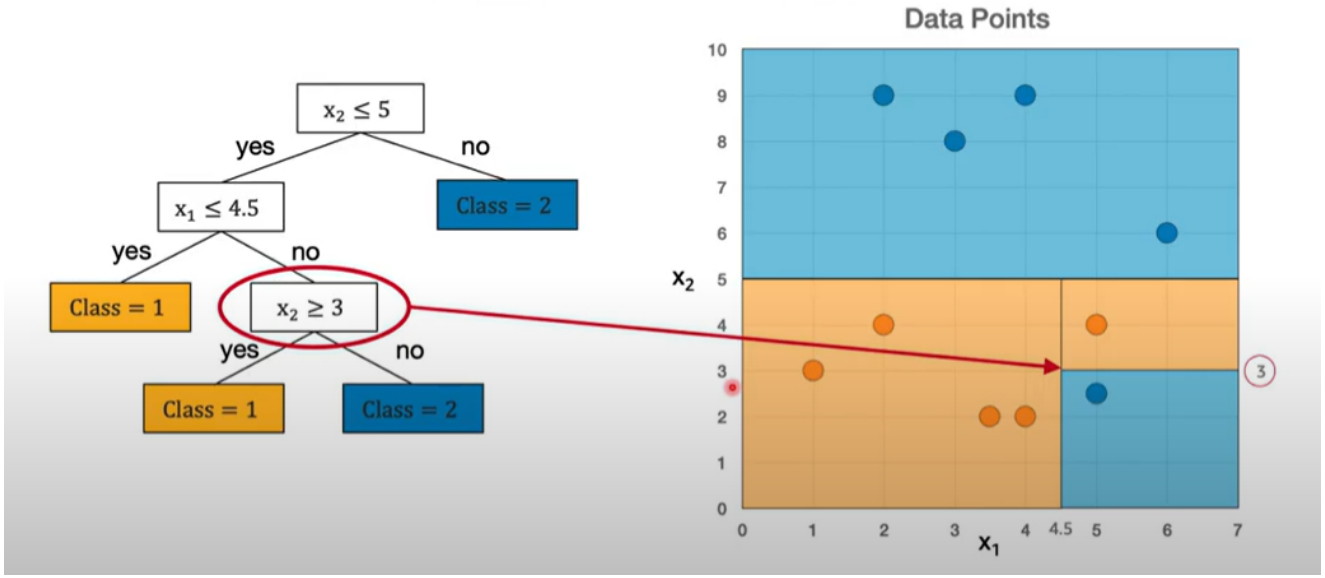




The CART algorithm: an example



Step2



Step3

What do we actually want?

- CART algorithms are often presented without stating exactly what we actually want to obtain.
- The generic CART algorithm creates a partition associated with a partial binary tree, i.e. every node has at most two children.
- Each region of the partition corresponds to a leaf of the tree.
- We want a tree $\mathcal{T}$ not "too big", with small misclassification error $R(\mathcal{T})$, and "enough" data points $N(\ell)$ in each leaf ℓ.
- That is, we want to find a tree $\mathcal{T}$ that solves:

$$\begin{aligned} \min \quad & R(\mathcal{T}) + \alpha|\mathcal{T}| \\ s.t. \quad & N(\ell) \geq N_{min} \qquad \forall \ell \in leaves(\mathcal{T}) \end{aligned} \tag{1}$$

- We can formulate the problem of finding an optimal tree as a MILP, following again Bertsimas and Dunn (2019).

Representing the problem

- Training data $(\mathbf{X}, \mathbf{y})$ are **normalized**, i.e. $\mathbf{x}_i \in [0,1]^p$, $i \in [n]$.
- Rather then minimizing $|\mathcal{T}|$, with fix the depth D of the tree.
- In principle, a solution could be represented by associating to each node of the tree suitable variables and constants.
- However, we do not know in advance the partial tree, i.e. which nodes it contains.
- To cope with this, we will consider a complete binary tree of depth D, which contains exactly $T = 2^{(D+1)} - 1$ nodes.
- Variables and constants are associated to every node (indexed by) $t \in [T]$ of the tree.
- To represent any partial binary tree of depth D we will allow some nodes not to be active.

Representing a solution

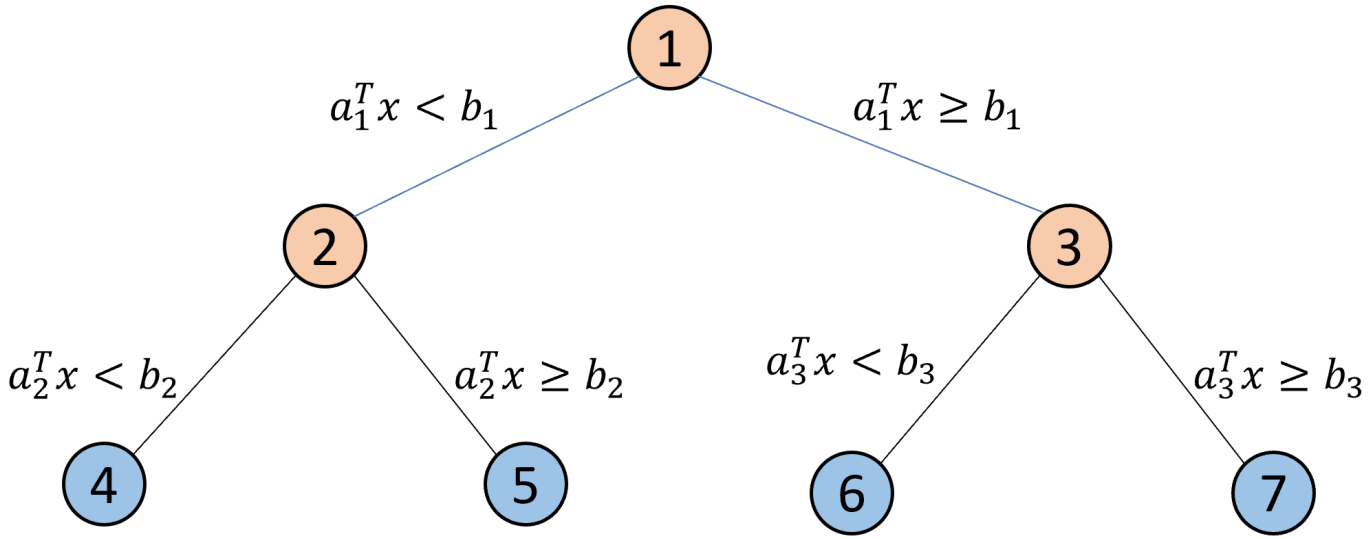
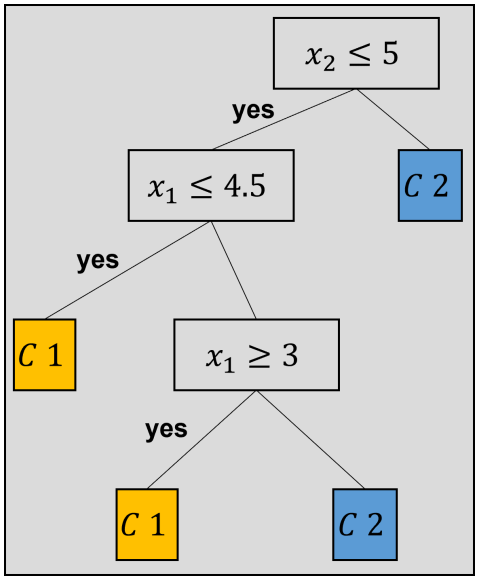


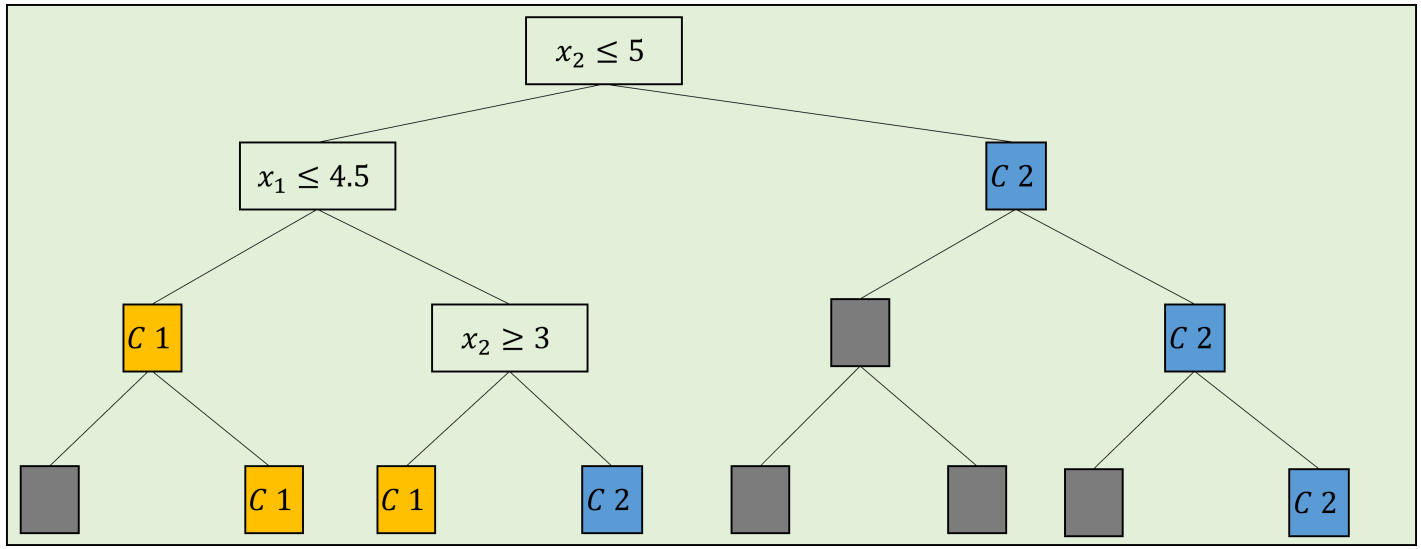
Figure: The complete binary tree of depth $D = 2$.

- The nodes of the tree are partitioned into *branch nodes* $\mathcal{T}_B = \{1, \ldots, 2^D - 1\}$ and *leaf nodes* $\mathcal{T}_L = \{2^D, \ldots, 2^{D+1} - 1\}$.
- The split hyperplane (parallel to an axis) associated with $t \in \mathcal{T}_B$ is denoted by $\mathbf{a}_t^T x = b_t$, where $\mathbf{a}^T = (0 \ldots 0\ 1\ 0 \ldots 0)$ is a vector with a single 1 and all other components are 0. $b_t \in [0, 1]$.
- A solution is then given by $\mathbf{a}_t, b_t$, for all $t \in \mathcal{T}_B$.

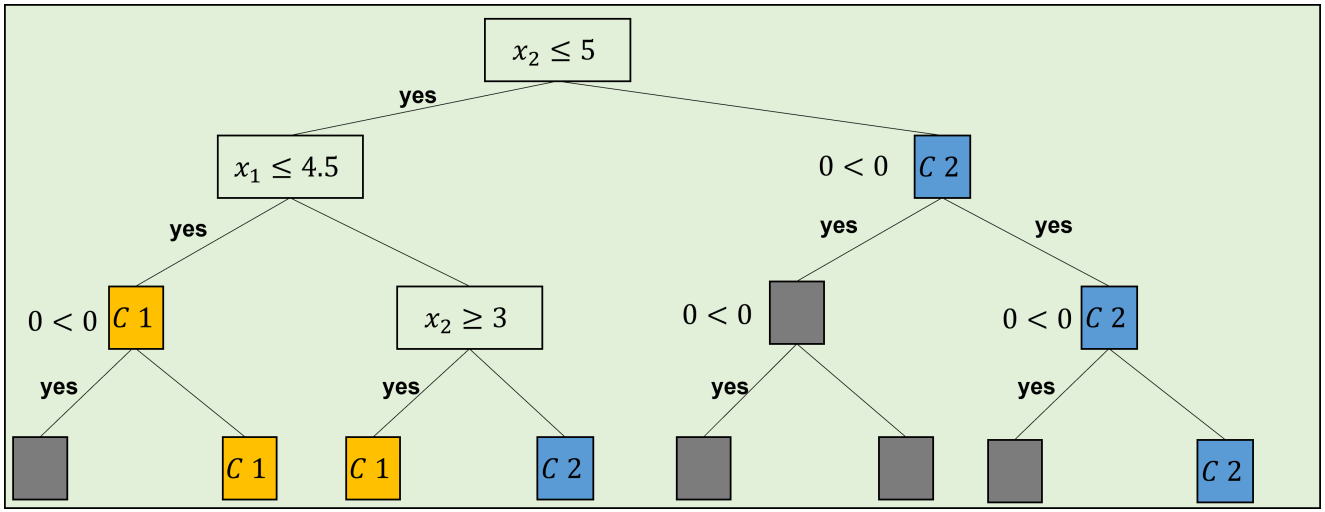
Representing a partial tree

- How do we map a partial (binary) tree into a complete binary tree?
- We add fictitious nodes!





Representing a partial tree



- How do we map a partial (binary) tree into a complete binary tree?

- Idea: (the region R_u of) a leaf u of the original partial tree is mapped into exactly one leaf $\ell(u)$ of the complete tree.
- $\ell(u)$ is the rightmost leaf of the subtree of the complete tree routed in (the node corresponding to the original leaf) u.
- We obtain this through fictitious splits (*non-split*).
- The left child ($0 < 0$) of a non-split corresponds to the empty region $R_u^1 = R_u \cap \{x : 0 < 0\} = \emptyset$.
- The right child ($0 \geq 0$) corresponds to the parent region $R_u^2 = R_u \cap \{x : 0 \geq 0\} = R_u$.

MILP model: split variables and constraints

- With every branching node $t \in \mathcal{T}_B$ and every feature $j \in [p]$, we introduce variables:
  - $a_{jt} \in \{0, 1\}$, $b_t \in [0, 1]$: coefficients of hyperplane $\mathbf{a}_t^T x = b_t$
  - $d_t \in \{0, 1\}$ identifies the type of node: $d_t = 0$ non-split node, $d_t = 1$ split node.
- If node $t \in \mathcal{T}_B$ is split, then we have:
  - $\sum_{j=1}^{p} a_{jt} = 1$ (in vector $\mathbf{a}_t$, exactly one component is 1)
  - $0 \leq b_t \leq 1$ (feature values are normalized)
- If $t \in \mathcal{T}_B$ is non-split, then the associated hyperplane reduces to the identity $0 = 0$, and so we have $\sum_{j=1}^{p} a_{jt} = 0$ and $b_t = 0$, for every $t \in \mathcal{T}_B$.
- The condition $t \in \mathcal{T}_B$ is an actual split depends on d_t, and so the above constraints can be represented as

$$\begin{aligned} \sum_{j=1}^{p} a_{jt} &= d_t \quad t \in \mathcal{T}_B \\ 0 \leq b_t &\leq d_t \quad t \in \mathcal{T}_B \end{aligned} \tag{2}$$

Non-split subtree

- All nodes in the sub-tree rooted in a non-split node are non-split.
- For $t \in \mathcal{T}_B \setminus \{1\}$, let $p(t)$ denote the parent node of t.
- Note that if $p(t)$ is a non-split node (i.e. $d_{p(t)} = 0$), so is t ($d_t = 0$). This implication is ensured by the constraints:

$$d_t \leq d_{p(t)} \qquad t \in \mathcal{T}_B \setminus \{1\}. \tag{3}$$

- Remark: the above valid constraints are not strictly necessary, but they strengthen the formulation.

Mapping data-points to leaves

- We now need to associate each data-point to exactly one leaf $t \in \mathcal{T}_L$.
- Let $r(t)$ be the region of the feature space associated with $t \in \mathcal{T}$.
- Note that every node in $t \in \mathcal{T}_B \setminus \{1\}$ is either a left child or a right child (of $p(t)$).
- For every $t \in \mathcal{T}$, we call *ancestors* of t the set of nodes on the unique path from t to the root $\{1\}$.
- For every $t \in \mathcal{T}_L$, we let $\mathcal{L}(t)$ and $\mathcal{R}(t)$ be the set of ancestors of t which are left and right children, respectively.
- For instance, in the complete tree of the ML University example, we have $\mathcal{L}(10) = \{2\}, \mathcal{R}(10) = \{5\}$, $\mathcal{L}(14) = \{\}, \mathcal{R}(14) = \{3, 7\}$.

Mapping data-points to leaves

- The region $r(t)$ associated to any leaf t is the set of points $\mathbf{x} \in \mathbb{R}^m$ satisfying by the system of inequalities $\begin{cases} \mathbf{a}_q^T \mathbf{x} < b_q & q \in \mathcal{L}(t) \\ \mathbf{a}_q^T \mathbf{x} \geq b_q & q \in \mathcal{R}(t) \end{cases}$
- With each data-point $(\mathbf{x}_i, y_i)$, $i \in [n]$ and each leaf $t \in \mathcal{T}_L$ we associated a binary variable z_{it} which is 1 if point i is assigned to t.
- Every data-point is assigned to exactly one leaf:

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1 \qquad i \in [n] \tag{4}$$

- If the i-th data-point $(\mathbf{x}_i, y_i)$ is assigned to leaf t, then $\mathbf{x}_i \in r(t)$ and we have:

$$z_{it} = 1 \rightarrow \begin{cases} \mathbf{a}_q^T \mathbf{x}_i < b_q & q \in \mathcal{L}(t) \\ \mathbf{a}_q^T \mathbf{x}_i \geq b_q & q \in \mathcal{R}(t) \end{cases}$$

Mapping data-points to leaves: the big-M trick

- The implication can be represented by means of the notorious big-M trick. Assume M is a (sufficiently) large constant.

$$\begin{array}{ll} (i) & \mathbf{a}_q^T \mathbf{x}_i < b_q + M(1 - z_{it}) \quad q \in \mathcal{L}(t) \\ (ii) & \mathbf{a}_q^T \mathbf{x}_i \geq b_q - M(1 - z_{it}) \quad q \in \mathcal{R}(t) \end{array}$$

- If $z_{it} = 1$, then the big-M terms disappear and the above system reduces to the conditions which must be satisfied for $\mathbf{x}_i \in r(t)$.
- If $z_{it} = 0$ and M is sufficiently large, since $\mathbf{a}_q \in \{0, 1\}^m$, $0 \leq \mathbf{a}_q^T \mathbf{x}_i \leq 1$ and $0 \leq b_q \leq 1$ for any $q \in \mathcal{T}_B$, $i \in [n]$, then (i) and (ii) are always satisfied and become redundant.
- We need to transform constraint (i) (not in MILP form).
- We need to transform constraint (i) into a standard form MILP inequality (i.e. '$\leq$' instead for '¡').

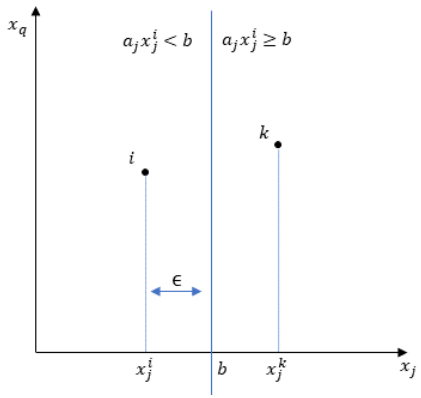
Mapping data-points to leaves: choosing ϵ

- We need to include a small, positive constant $\epsilon > 0$.

$$\begin{aligned} &(i) \quad \mathbf{a}_q^T \mathbf{x}_i \leq b_q - \epsilon + M(1 - z_{it}) \quad && q \in \mathcal{L}(t) \\ &(ii) \quad \mathbf{a}_q^T \mathbf{x}_i \geq b_q - M(1 - z_{it}) && q \in \mathcal{R}(t) \end{aligned} \tag{5}$$

- ϵ should be sufficiently small not to alter the feasible solutions set but as large as possible to avoid numerical instabilities.

- Let x_j^i denote the j-feature value of data-point $i \in [n]$ and let
$\epsilon_j = \min \left\{ |x_j^i - x_j^k| : k, i \in [n], x_j^i \neq x_j^k \right\}$



- $\epsilon = \min_{j \in [p]} \epsilon_j$ is a feasible choice for ϵ

Exercise

Exercise

Show that $\epsilon = \min_{j \in [p]} \epsilon_j$ is a feasible choice for ϵ in constraint (5.i).

Exercise

Find a small value for M which ensures that constraint (5.ii) is satisfied for any $b_q \in [0, 1]$ and any $\mathbf{a}_q \in \{0, 1\}^p$ such that $\mathbf{a}_q^T \mathbb{1} = 1$.

Objective function

- The objective is to minimize the misclassification error, so an incorrect label prediction has cost 1, and a correct one has cost 0.
- To find the number of misclassified data-points in a leaf $t \in \mathcal{T}_L$ we need to know the total number of data-points, the label class with most data points (the *representative class* of the node) and the number of points belonging to the other classes.
- We introduce decision variables:
  - $N_{kt} \in \mathbb{Z}_+$ number of data-points of class $k \in [K]$ assigned to t
  - $N_t \in \mathbb{Z}_+$, total number of data-points in class t
  - $c_{kt} \in \{0, 1\}$ indicator variable which is 1 if $k \in [K]$ is the representative class of t, and is 0 for all other classes.
  - $L_t \in \mathbb{Z}_+$ number of misclassified points in t.
- The objective amounts to minimizing the total number of misclassified points:

$$\min \sum_{t \in \mathcal{T}_L} L_t \tag{6}$$

Objective function: computing main quantities

- We need to choose a representative class for each node

$$\sum_{k \in [K]} c_{kt} = 1 \qquad t \in \mathcal{T}_L \tag{7}$$

- Number of data-points of class k assigned to t

$$N_{kt} = \sum_{i : y_i = k} z_{it} \qquad k \in [K], t \in \mathcal{T}_L \tag{8}$$

- Total number of data-points assigned to t

$$N_t = \sum_{k \in [K]} N_{kt} \qquad t \in \mathcal{T}_L \tag{9}$$

Objective function: computing L_t

- If $\bar{k}$ is the representative class of node t, then $L_t = N_t - N_{\bar{k}t}$, i.e. the total number of data-points in t minus the data-points well classified. $\bar{k}$ is known through variable vector $\mathbf{c}_t$
- We need to use again the big-M trick, with $\overline{M}$ large constant.

$$L_t \geq N_t - N_{kt} - \overline{M}(1 - c_{kt}) \qquad k \in [K], t \in \mathcal{T}_L \tag{10}$$

- When $c_{kt} = 0$ the constraint becomes redundant as $L_t \geq 0$.
- When $c_{kt} = 1$, the constraint reduces to $L_t \geq N_t - N_{kt}$.
- Because of constraint (7), there is only one such non-redundant constraint for each $t \in \mathcal{T}_L$.
- Let $\bar{k} \in [K]$ be the unique label such that $c_{\bar{k}t} = 1$. The objective function then forces then $L_t = N_t - N_{\bar{k}t}$ (any other feasible value for L_t would lead to a worsening of the objective).

Another way to represent misclassification error

- Alternative to Bertsimas and Dunn approach, proposed by PhD student Johan Sokrates (during my lectures) in 2022.
- It avoids the use of annoying big-M constraints.
- We drop variables L_t, N_t, N_{kt} and associated constraints.
- We introduce a binary variable m_i, which is 1 if data point i is misclassified, 0 otherwise.
- The objective function becomes: $\min \sum_{i \in [n]} m_i$
- The following constraint forces $m_i = 1$ when data-point $(\mathbf{x}_i, y_i)$ is assigned to a leaf t with the wrong label (i.e. different from y_i).

$$m_i \geq z_{it} - c_{y_i t} \qquad i \in [n], t \in \mathcal{T}_L$$

- If m_i is not forced to 1, it will be forced to 0 by the objective.

Exercise

Exercise

What is a good value for $\overline{M}$ in (10)?

Exercise

The complexity of the tree is somehow related to the number of splits. We can add a penalty term in the objective function (6) proportional to such number (by a parameter α).

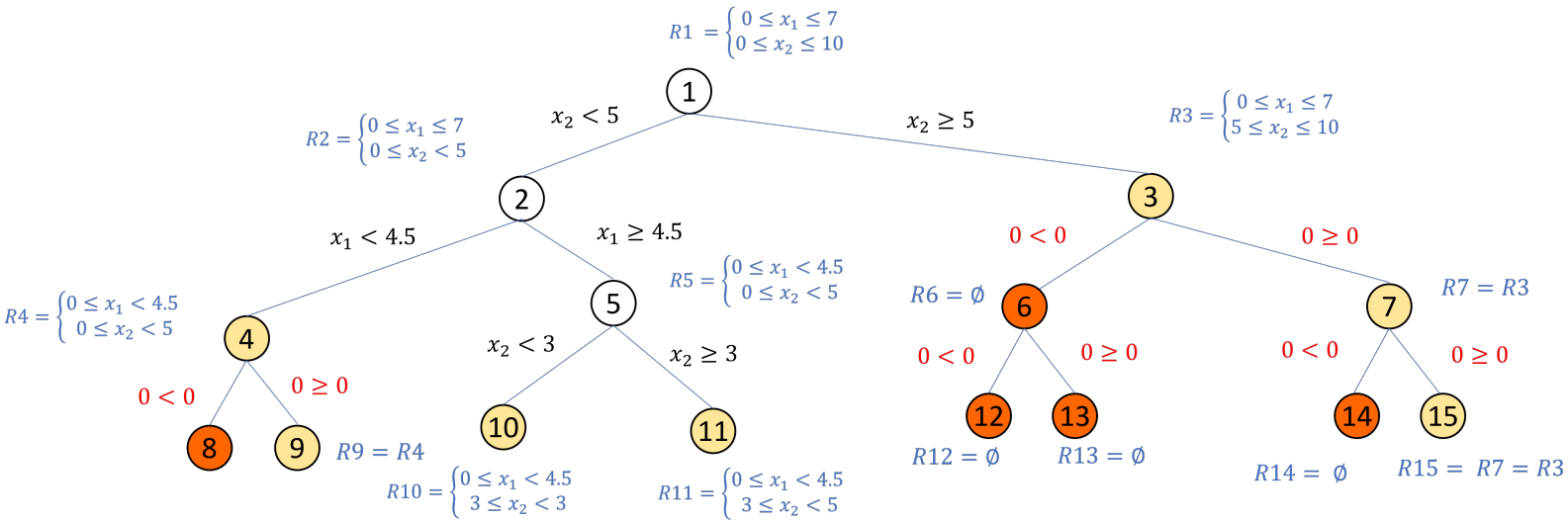
Exercise

Extend the model to cope with generic splitting hyperplanes (not necessarily parallel to one of the axes).

Exercise

Write the constraints to impose a lower bound N_l on the number of data points associated to any (non-empty) leaf.

The complete tree of the ML University example



- Yellow nodes correspond to original leaves.
- Red nodes correspond to empty regions.
- Note how the original leaf region is pushed downward the sub-tree of the complete to the rightmost leaf.