

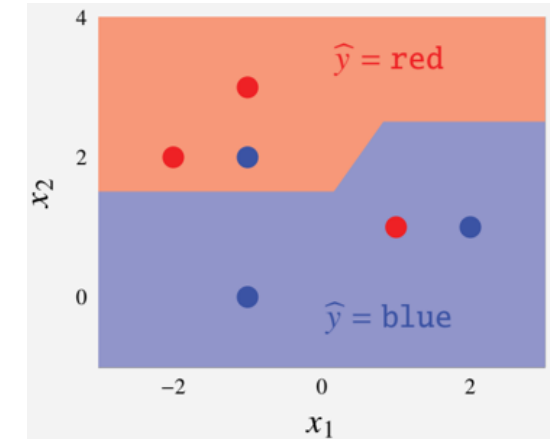
# Fundamentals of AI/ML and Applications

## Decision Trees

Sponsored by  
Software Technology Parks of India (STPI)

# Supervised Learning: Recap of $k$ NN

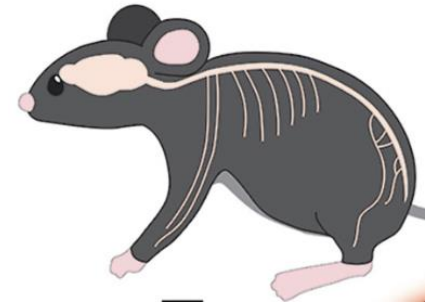
- In both regression and classification settings, we seek a function  $f(\mathbf{x}_*)$  that maps the test input  $\mathbf{x}_*$  to a prediction
- The  $k$ -NN method results in a prediction  $\hat{y}(\mathbf{x}_*)$  that is a **piecewise constant function** of the input  $\mathbf{x}_*$ 
  - The method partitions the input space into **disjoint regions**
  - Each region is associated with a certain **constant** prediction
  - These regions are described by the  **$k$ -neighbourhood of each possible test input**
- Another way is to come up with **a set of rules that defines the regions explicitly: Decision Trees**
  - Also known as Classification and Regression Trees (CART)



# Idea of decision trees (CART)



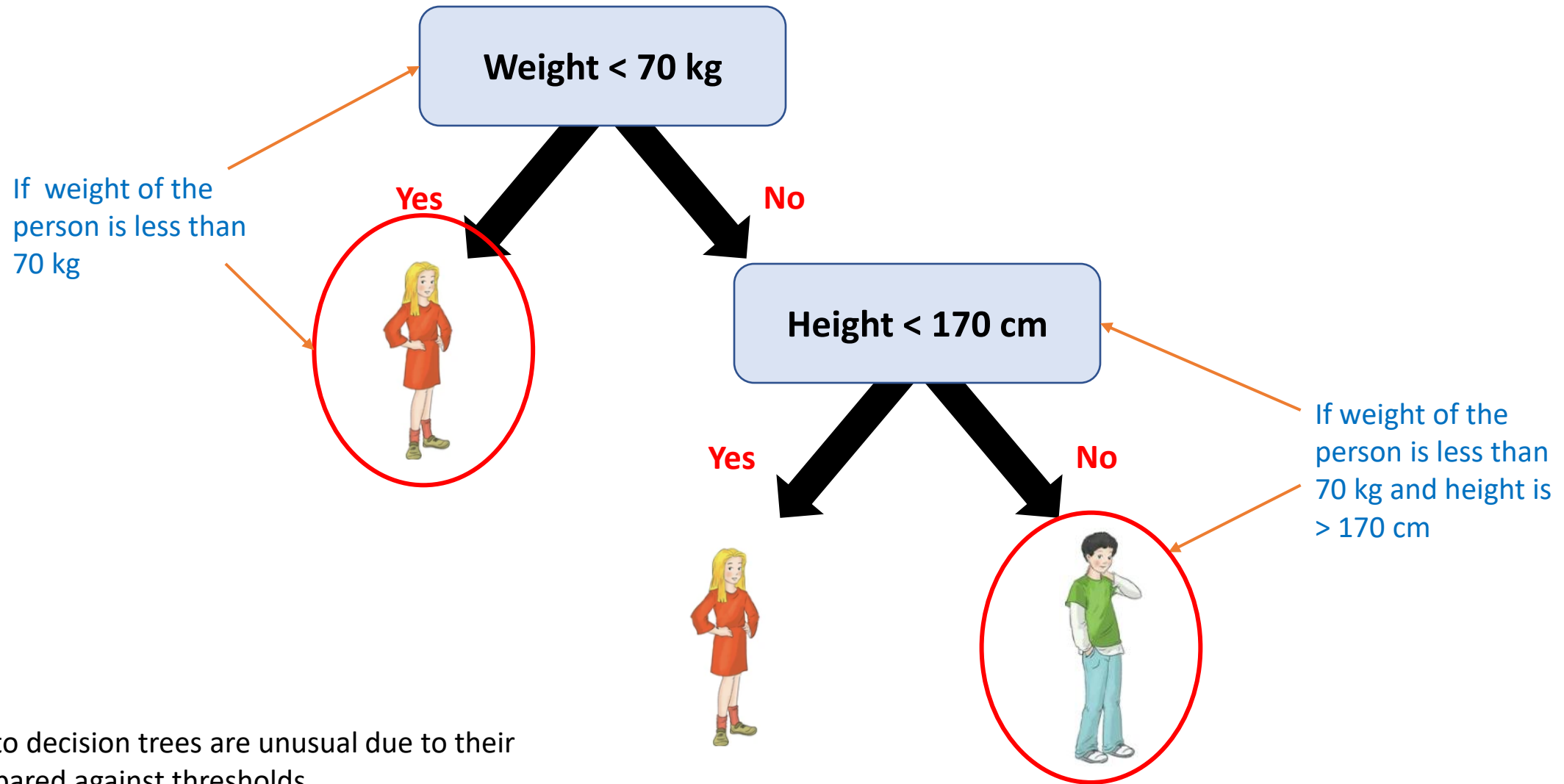
- Features measured: height and weight
- **Scenario:** Decide **Male** or **Female**?



- Feature considered: Diet portion size, nerve activity
- **Scenario:** Decide **rat weight**?



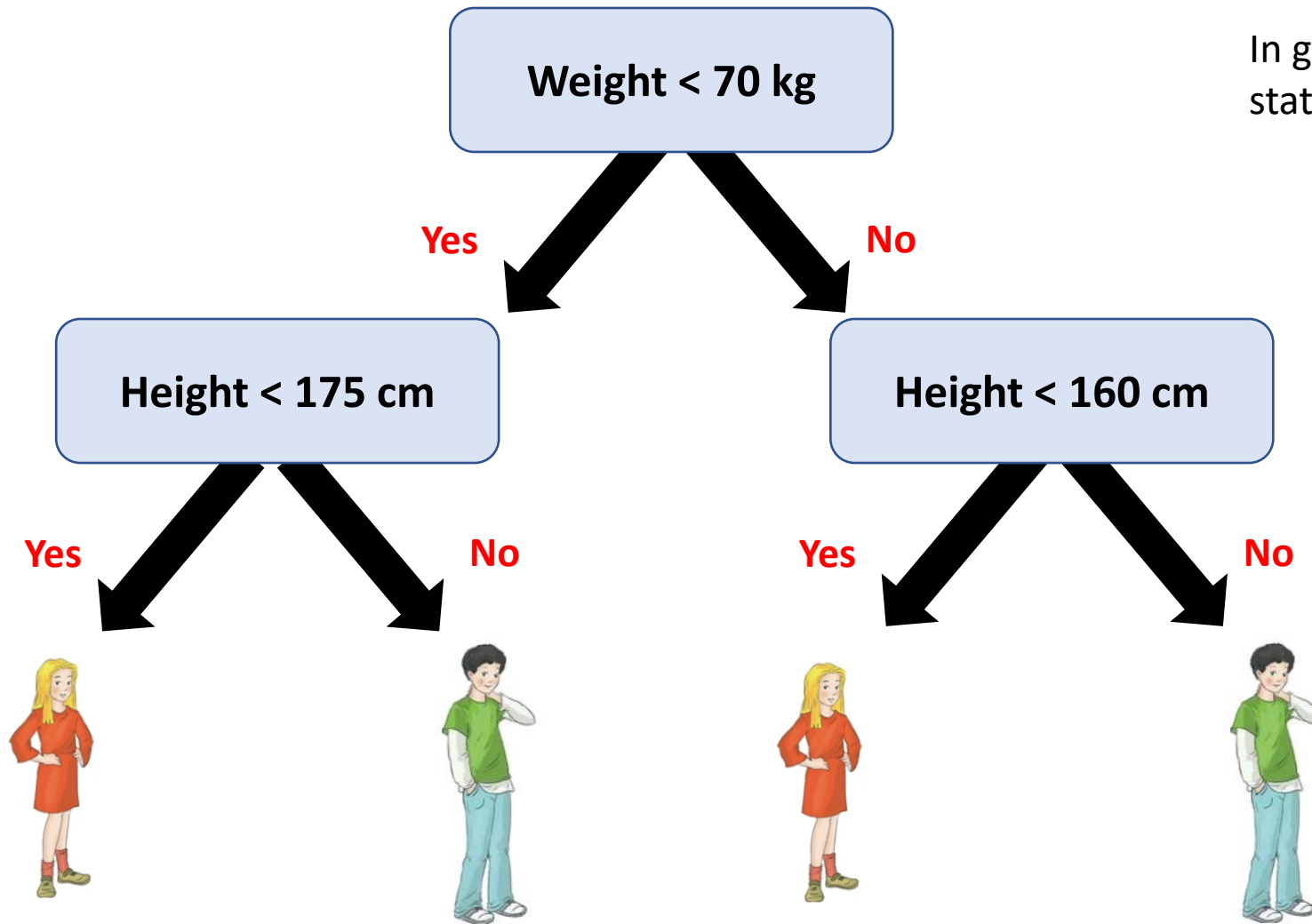
# A simple decision tree based on rules



Categorical inputs to decision trees are unusual due to their inability to be compared against thresholds

Split **numeric** input features by checking whether that feature is greater than or less than **some threshold**

# Another possible decision tree

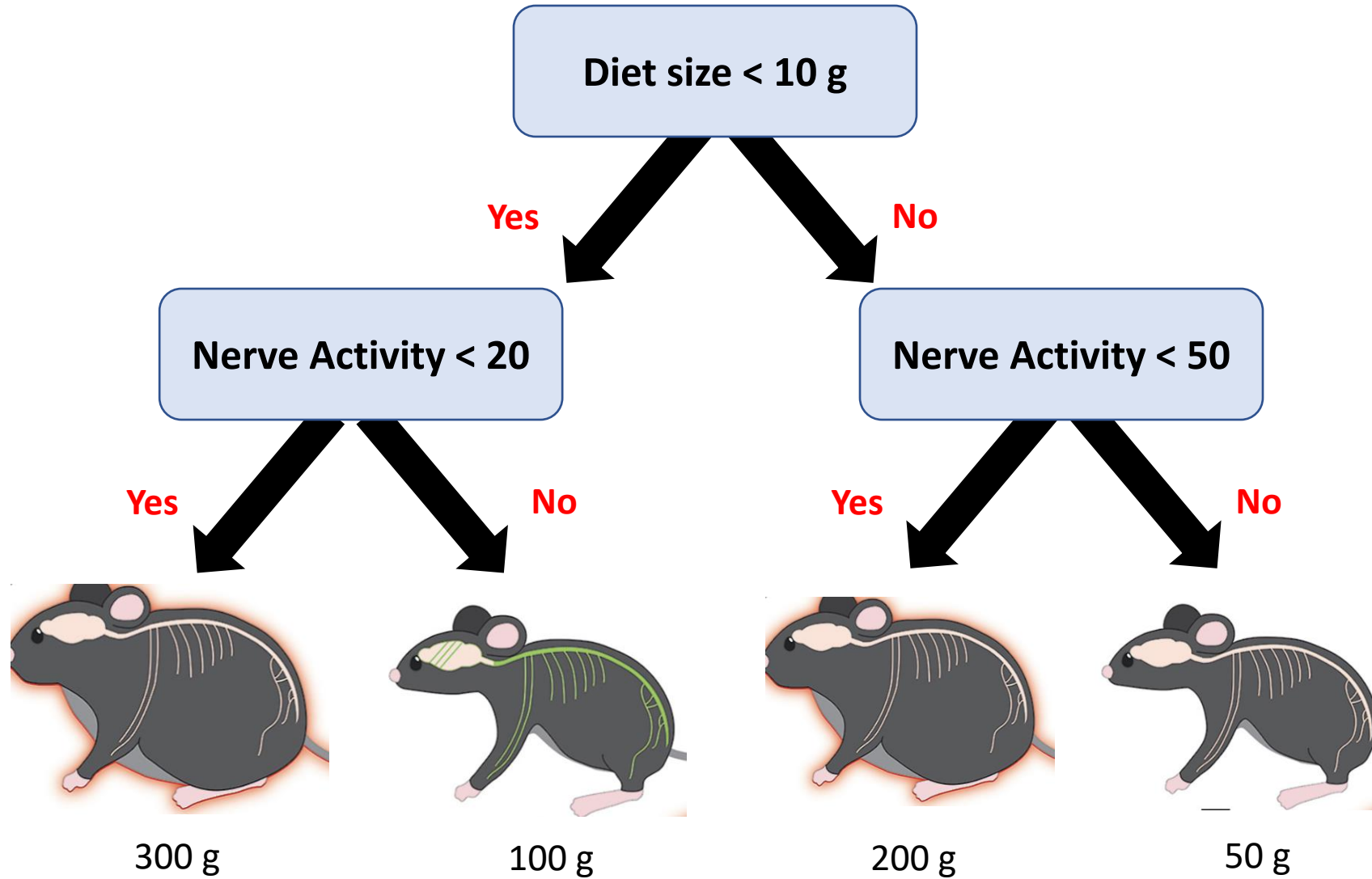


In general, a decision tree makes a statement...

... and then makes a decision based on whether or not the statement is true or false


The outputs are categorical → Classification problem → **Classification** tree

# Regression tree



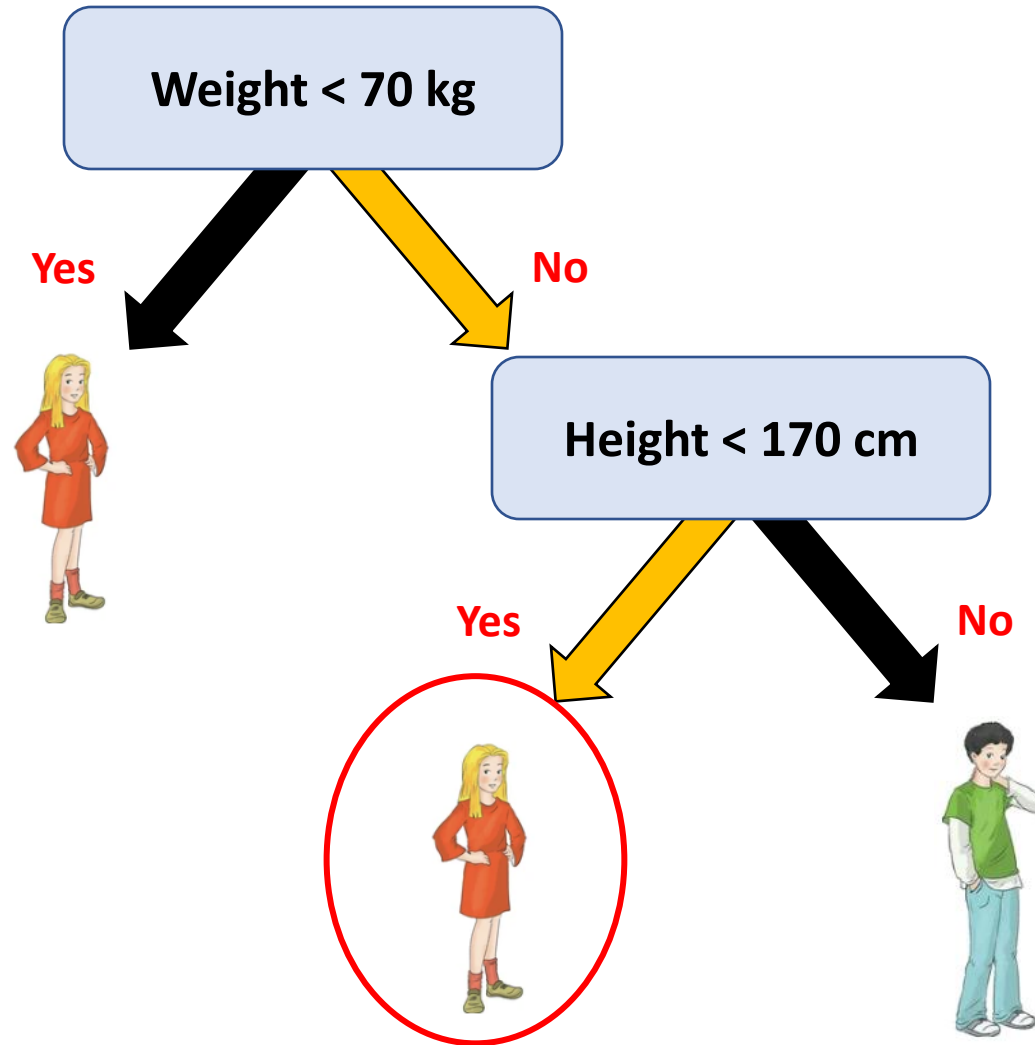
The outputs are numerical values → Regression problem → **Regression tree**

# Prediction using decision trees (CART)



**Test input**

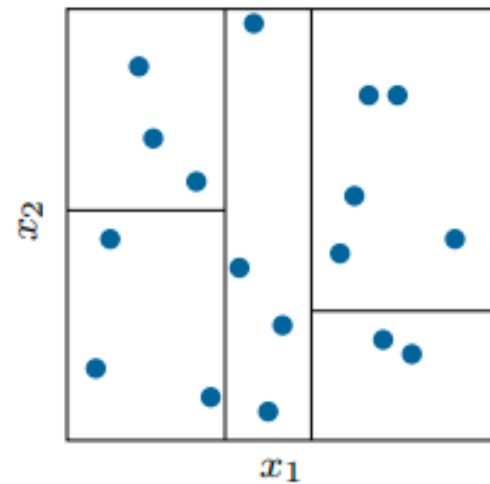
- Weight = 70 kg
- Height = 168 cm



Make predictions by splitting on features according to a [tree structure](#)

# Rule-based method: Decision Trees

- In both regression and classification settings, we seek a function  $f(\mathbf{x}^*)$  that maps the test input  $\mathbf{x}^*$  to a prediction
- One **flexible** way of designing this function is to partition the input space into multiple disjoint regions and fit a simple model in each region
  - For  $k$ -NN, these regions are given implicitly by the  $k$ -neighbourhood of each possible test input
  - Another way is to come up with a set of rules that defines the regions explicitly: **Decision Trees**



● = Training data

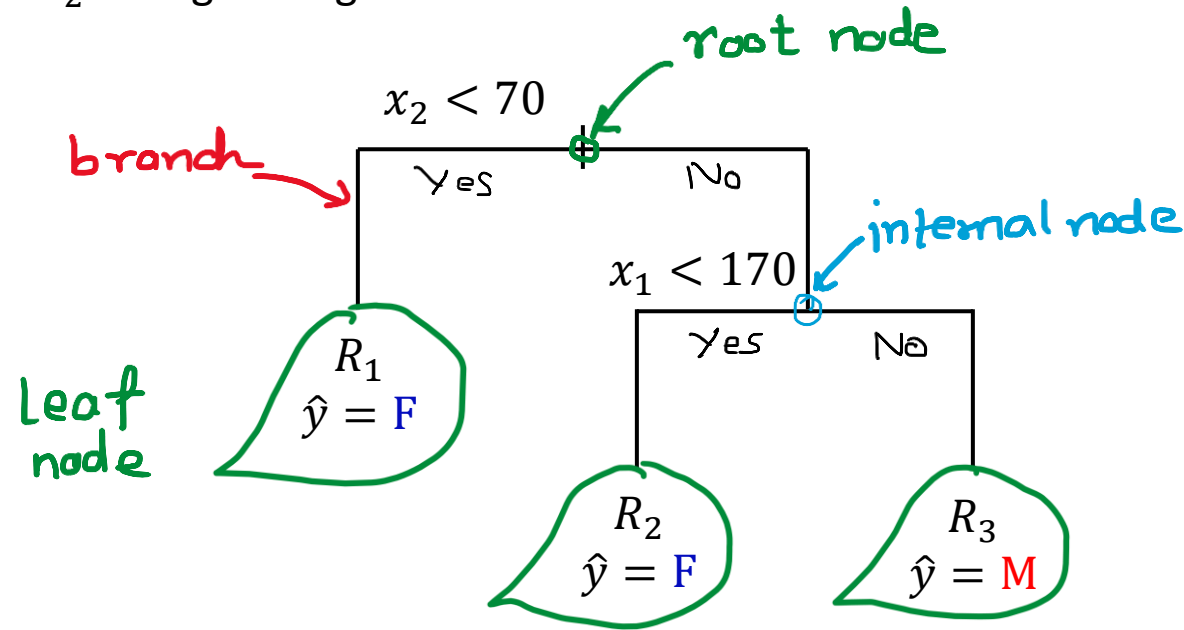


# Structure of a Decision Tree looks like an Inverted Tree



# Structure of a Decision Tree

- Consider the classification example with two numerical inputs  $\mathbf{x} = [x_1 \ x_2]^T$  and a categorical output  $y \in \{F, M\}$ 
  - $x_1$  - height in centimeters
  - $x_2$  - weight in kg



Structure of a **decision** tree

- Internal nodes (which includes root node) check an input variable for a condition
- Left/Right branch is determined by value of input variable  $\rightarrow x_j < s_k$
- Leaf nodes are outputs (predictions)

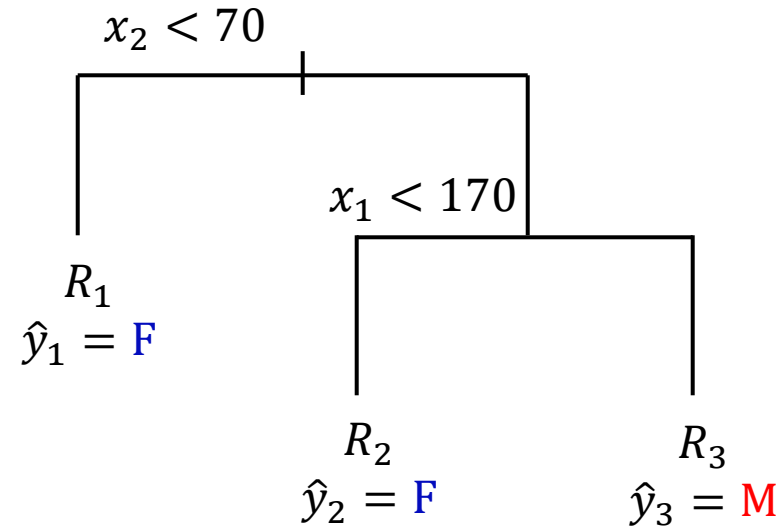
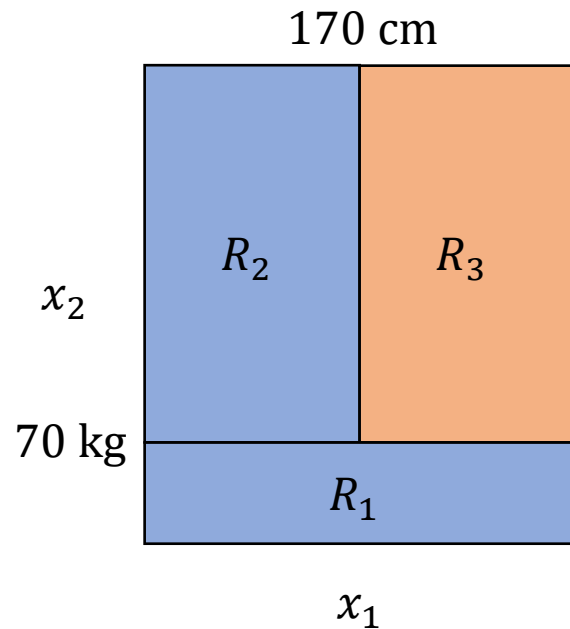
The tree shown has two internal nodes (including the root) and three leaf nodes

## Set of Rules

```
If Height < 170 cm Then Female
If Height >= 170 cm AND Weight < 70 kg Then Female
If Height <= 170 cm AND Weight >= 70 kg Then Male
```

# Decision boundaries of a Decision Tree

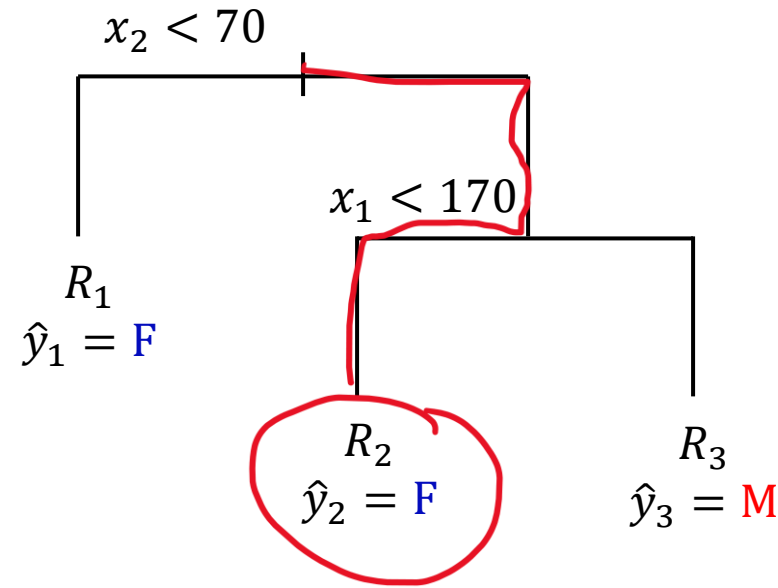
- Each path from **root node** to a **leaf** defines a **region  $R_m$**  of input space



- The decision tree partitions the input space into **axis-aligned 'boxes' or rectangles**. So the **decision boundaries** are in the shape of **rectangles**

# Make a prediction using a Classification tree

- Consider a test input  $\mathbf{x}^* = [70 \quad 168]^T$ 
  - $x_1$  - height in centimeters
  - $x_2$  - weight in kg

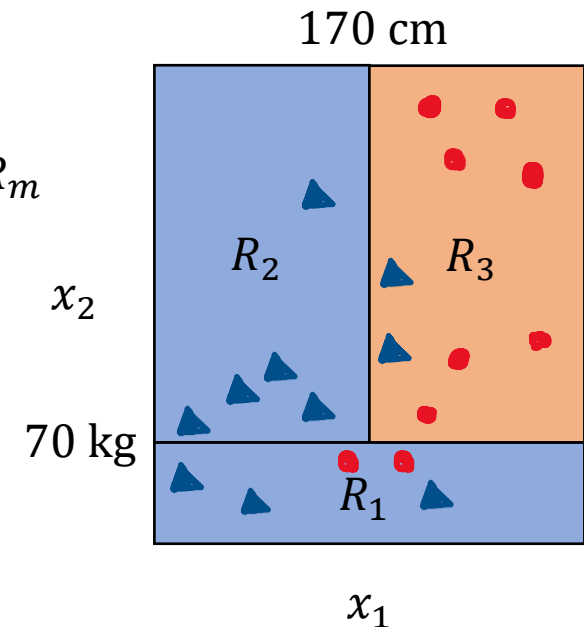


## Set of Rules

```
If Height < 170 cm Then Female
If Height >= 170 cm AND Weight < 70 kg Then Female
If Height <= 170 cm AND Weight >= 70 kg Then Male
```

# Structure of generic decision trees

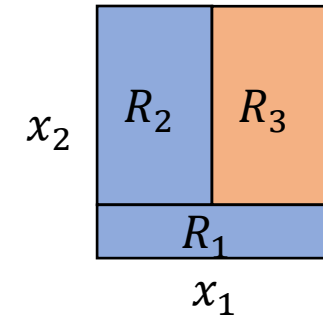
- The same partitioning can be done for a general input vector  $\mathbf{x} = [x_1 \ x_2 \ \cdots \ x_p]^T$ , however, the partitioned space is difficult to illustrate
- For a  $p$ -dimensional input space, the decision boundaries would be represented by **hyper-rectangles (boxes in higher dimensions)**
- Let  $\{(\mathbf{x}_{m_1}, y_{m_1}), (\mathbf{x}_{m_2}, y_{m_2}), \dots, (\mathbf{x}_{m_k}, y_{m_k})\}$  be the training examples that fall into  $R_m$ 
  - $m = 3, k = 9$  for the top-right reddish box
- **Regression tree**
  - Numerical output
  - Leaf value  $\hat{y}_m$  typically set to the mean value in  $\{(\mathbf{x}_{m_1}, y_{m_1}), \dots, (\mathbf{x}_{m_k}, y_{m_k})\}$
- **Classification tree**
  - Categorical output
  - Leaf value  $\hat{y}_m$  typically set to the most common value (mode) in  $\{(\mathbf{x}_{m_1}, y_{m_1}), \dots, (\mathbf{x}_{m_k}, y_{m_k})\}$



# Learning Decision Trees

- We saw how a decision tree can be used to make a prediction. **Now, how a tree can be learned from training data?**

- Learning a decision tree involves deciding the **shape of the tree**
  - Finding the **number of regions (boxes)**, say  $L$  regions,  $R_1, R_2, \dots, R_L$ , and
  - Finding the **partitions of the boxes**



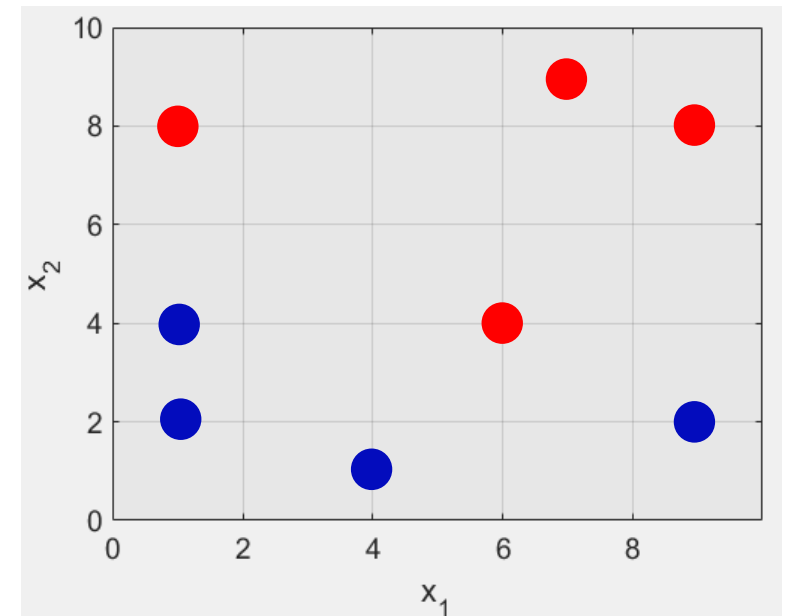
- We should select the number of regions and partitions such that the tree fits the training data well
  - the output predictions from the tree should match the output values in the training data
- However, finding the tree (a collection of splitting rules) that optimally partitions the input space to fit the training data is **computationally infeasible due to combinatorial explosion** in the number of ways you can partition the input space
- Searching through all possible binary trees is not possible in practice unless the tree size is very small
- To handle this situation, a heuristic algorithm known as **recursive binary splitting** is used for learning decision trees.

# Learning **Classification** Trees using Recursive Binary Splitting

Start with the first split at the root and then build the tree from top to bottom

- When determining the splitting rule at the root node, *the objective is to obtain a model that best explains the training data after a single split*, without taking into consideration that additional splits may be added afterwards

$i$	$x_1$	$x_2$	$y$
1	9	2	Blue
2	4	1	Blue
3	1	2	Blue
4	1	4	Blue
5	1	8	Red
6	6	4	Red
7	7	9	Red
8	9	8	Red



# Learning **Classification** Trees using Recursive Binary Splitting

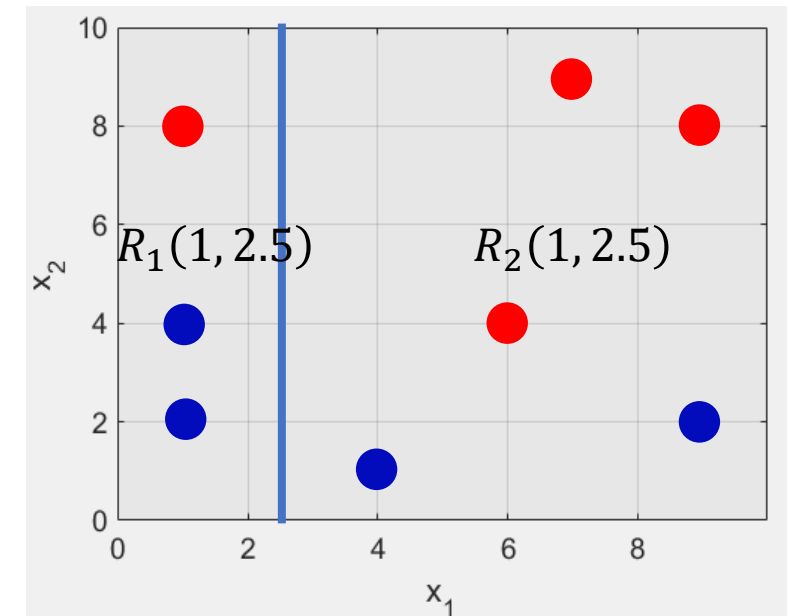
Start with the first split at the root and then build the tree from top to bottom

- When determining the splitting rule at the root node, *the objective is to obtain a model that best explains the training data after a single split*, without taking into consideration that additional splits may be afterwards
  - Select one of the  $p$  input variables  $x_1, \dots, x_j, \dots, x_p$  and a corresponding cutpoint  $s$  which divide the input space into two half-spaces

$$R_1(j, s) = \{ \mathbf{x} \mid x_j < s \} \quad \text{and} \quad R_2(j, s) = \{ \mathbf{x} \mid x_j \geq s \}$$

$i$	$x_1$	$x_2$	$y$
1	9	2	Blue
2	4	1	Blue
3	1	2	Blue
4	1	4	Blue
5	1	8	Red
6	6	4	Red
7	7	9	Red
8	9	8	Red

Split at say  $x_1 = 2.5$





# Learning **Classification** Trees using Recursive Binary Splitting

Start with the first split at the root and then build the tree from top to bottom

- When determining the splitting rule at the root node, *the objective is to obtain a model that best explains the training data after a single split*, without taking into consideration that additional splits may be added afterwards
  - Select one of the  $p$  input variables  $x_1, \dots, x_j, \dots, x_p$  and a corresponding cutpoint  $s$  which divide the input space into two half-spaces

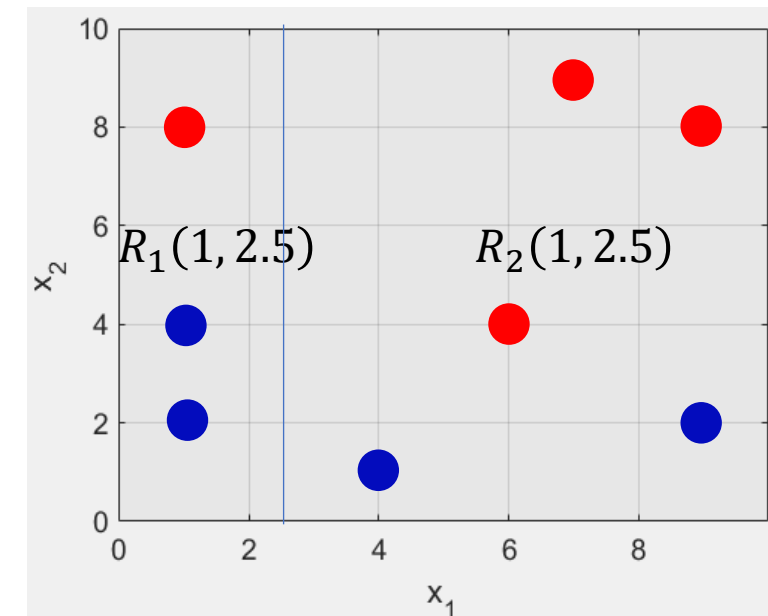
$$R_1(j, s) = \{ \mathbf{x} \mid x_j < s \} \quad \text{and} \quad R_2(j, s) = \{ \mathbf{x} \mid x_j \geq s \}$$

- The predictions associated with the two regions will be

$$\hat{y}_1(j, s) = \text{Mode}\{y_i : \mathbf{x}_i \in R_1(j, s)\} \quad \text{and} \quad \hat{y}_2(j, s) = \text{Mode}\{y_i : \mathbf{x}_i \in R_2(j, s)\}$$

**Blue**

**Red**



# Learning Classification Trees using Recursive Binary Splitting

Start with the first split at the root and then build the tree from top to bottom

- When determining the splitting rule at the root node, *the objective is to obtain a model that minimizes the loss function on the training data after a single split*, without taking into consideration that additional splits may achieve a better fit

- Select one of the  $p$  input variables  $x_1, \dots, x_p$  and a corresponding cutpoint  $s$  with  $s \in [0, 1]$  to split the data into two half-spaces

$$R_1(j, s) = \{\mathbf{x} \mid x_j < s\} \quad \text{and} \quad R_2(j, s) = \{\mathbf{x} \mid x_j \geq s\}$$

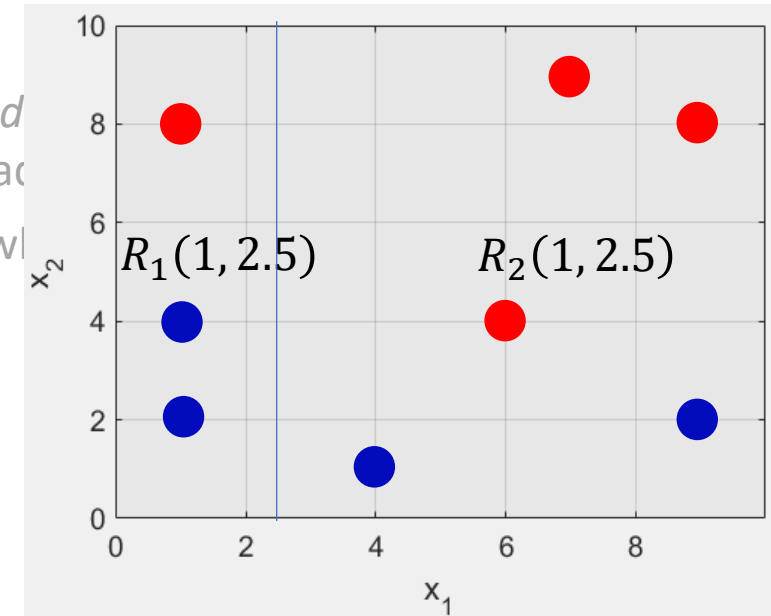
- The predictions associated with the two regions will be

$$\hat{y}_1(j, s) = \text{Mode}\{y_i : \mathbf{x}_i \in R_1(j, s)\} \quad \text{and} \quad \hat{y}_2(j, s) = \text{Mode}\{y_i : \mathbf{x}_i \in R_2(j, s)\}$$

- Compute prediction loss for all training data points  $(\mathbf{x}_i, y_i)_{i=1}^N$  at the node to determine goodness-of-fit

$$\text{Loss} = n_1 Q_{\ell,L}(j, s) + n_2 Q_{\ell,R}(j, s)$$

where,  $n_1, n_2$  are the number of data points in left and right nodes of current split and  $Q_{\ell,L}, Q_{\ell,R}$  are the associated prediction errors of the left ( $L$ ) branch and right ( $R$ ) branch for the  $\ell$ th region



**Loss** → When learning a model, we use a **scalar number** to assess whether we are on track; **low is good**, high is bad

# Prediction errors for **classification** trees

- Define  $\hat{\pi}_{\ell,m}$  as the proportion of training observations in the  $\ell$ th region belong to the  $m$ th class

$$\hat{\pi}_{\ell,m} = \frac{1}{n_{\ell}} \sum_{i: \mathbf{x}_i \in R_{\ell}(j,s)} \mathbb{I}(y_i = m)$$

Indicator function

- Misclassification rate:** proportion of data points in region  $R_{\ell}$  which do not belong to the most common class

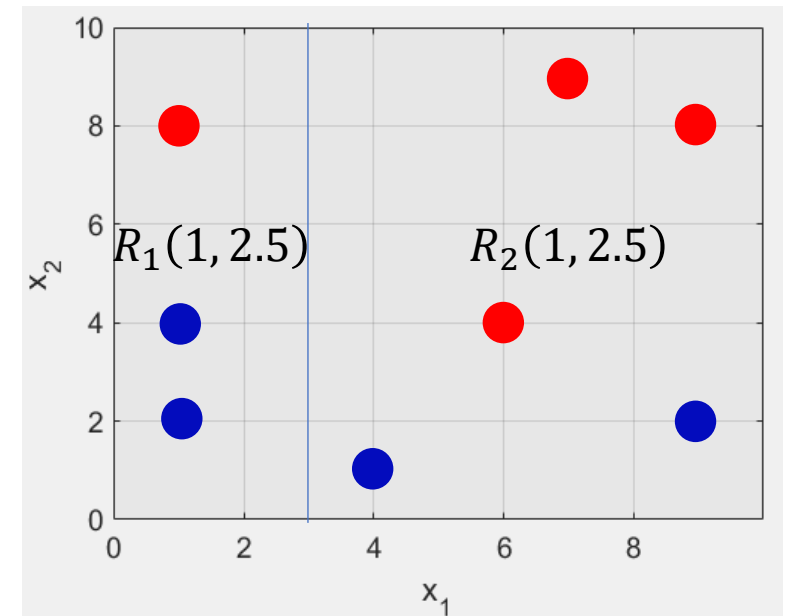
$$Q_{\ell} = 1 - \max_m \hat{\pi}_{\ell,m} ,$$

- Gini index**

$$Q_{\ell} = \sum_{m=1}^M \hat{\pi}_{\ell,m} (1 - \hat{\pi}_{\ell,m})$$

- Entropy criteria** (commonly used)

$$Q_{\ell} = - \sum_{m=1}^M \hat{\pi}_{\ell,m} \ln \hat{\pi}_{\ell,m}$$

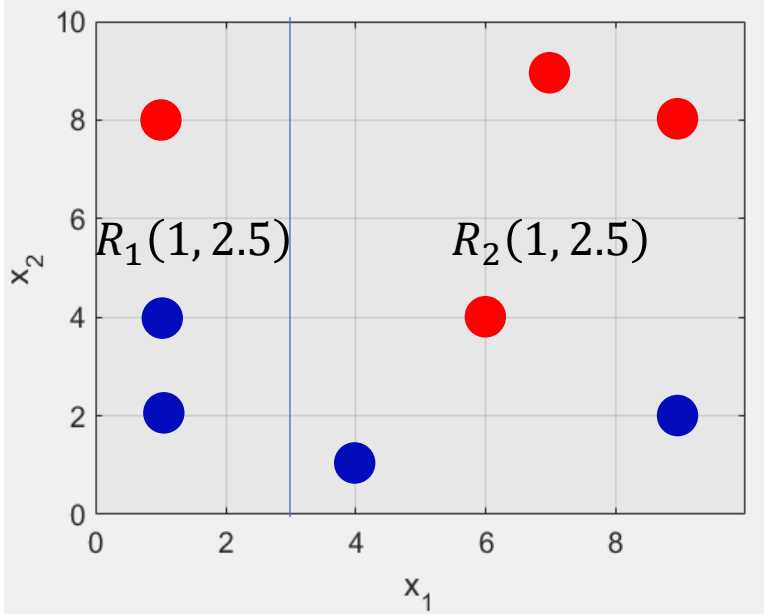


# Calculating the loss for 1<sup>st</sup> split

- Define  $\hat{\pi}_{\ell,m}$  as the proportion of training observations in the  $\ell$ th region of that belong to the  $m$ th class

$$\hat{\pi}_{\ell,m} = \frac{1}{n_{\ell}} \sum_{i:\mathbf{x}_i \in R_{\ell}(j,s)} \mathbb{I}(y_i = m)$$

- Let's use **Misclassification rate**:  $Q_{\ell} = 1 - \max_m \hat{\pi}_{\ell,m}$



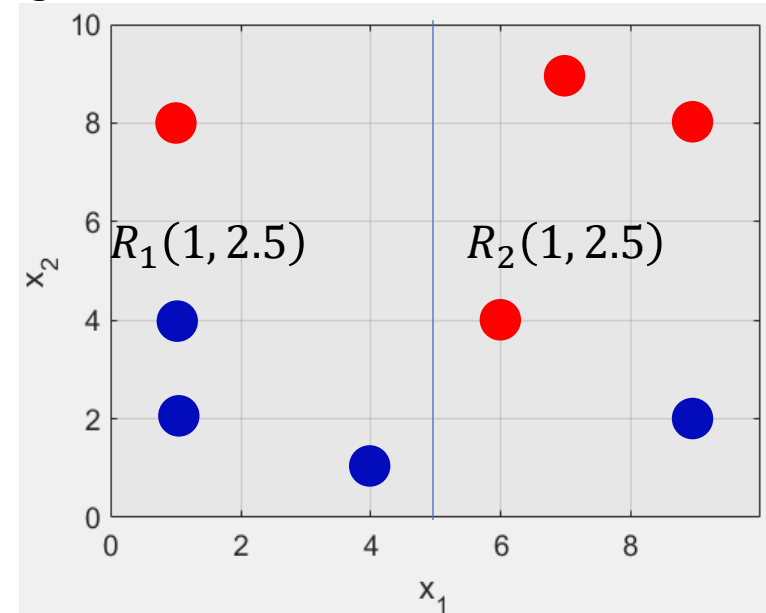
Splits ( $R_1$ )	$n_1$	$\hat{\pi}_{1,B}$	$\hat{\pi}_{1,R}$	$Q_1$	$n_2$	$\hat{\pi}_{2,B}$	$\hat{\pi}_{2,R}$	$Q_2$	$n_1 Q_1 + n_2 Q_2$
$x_1 < 2.5$	3	2/3	1/3	1/3	5	2/5	3/5	2/5	3

# Calculating the loss for 1<sup>st</sup> split

- Define  $\hat{\pi}_{\ell,m}$  as the proportion of training observations in the  $\ell$ th region of that belong to the  $m$ th class

$$\hat{\pi}_{\ell,m} = \frac{1}{n_{\ell}} \sum_{i: \mathbf{x}_i \in R_{\ell}(j,s)} \mathbb{I}(y_i = m)$$

- Let's use **Misclassification rate**:  $Q_{\ell} = 1 - \max_m \hat{\pi}_{\ell,m}$



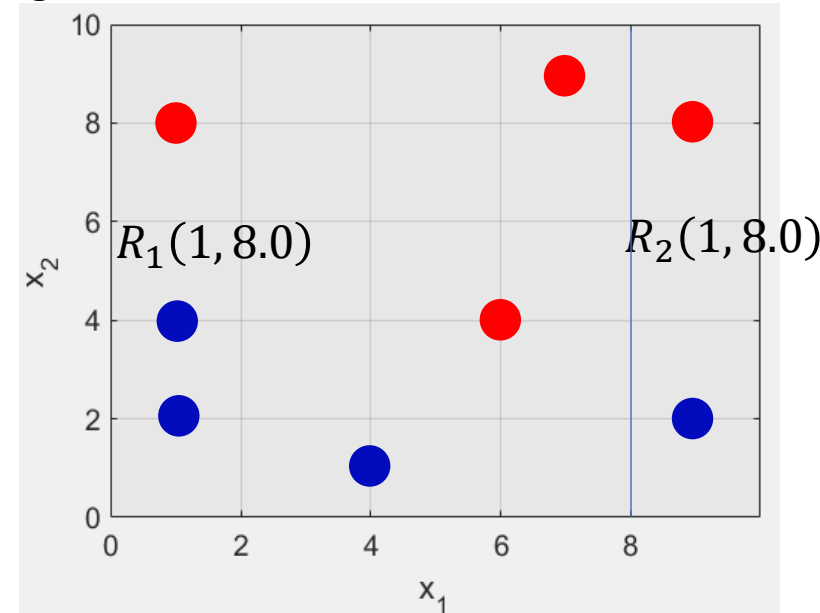
Splits ( $R_1$ )	$n_1$	$\hat{\pi}_{1,B}$	$\hat{\pi}_{1,R}$	$Q_1$	$n_2$	$\hat{\pi}_{2,B}$	$\hat{\pi}_{2,R}$	$Q_2$	$n_1 Q_1 + n_2 Q_2$
$x_1 < 2.5$	3	2/3	1/3	1/3	5	2/5	3/5	2/5	3
$x_1 < 5.0$	4	3/4	1/4	1/4	4	1/4	3/4	1/4	2

# Calculating the loss for 1<sup>st</sup> split

- Define  $\hat{\pi}_{\ell,m}$  as the proportion of training observations in the  $\ell$ th region of that belong to the  $m$ th class

$$\hat{\pi}_{\ell,m} = \frac{1}{n_{\ell}} \sum_{i: \mathbf{x}_i \in R_{\ell}(j,s)} \mathbb{I}(y_i = m)$$

- Let's use **Misclassification rate**:  $Q_{\ell} = 1 - \max_m \hat{\pi}_{\ell,m}$



Splits ( $R_1$ )	$n_1$	$\hat{\pi}_{1,B}$	$\hat{\pi}_{1,R}$	$Q_1$	$n_2$	$\hat{\pi}_{2,B}$	$\hat{\pi}_{2,R}$	$Q_2$	$n_1 Q_1 + n_2 Q_2$
$x_1 < 2.5$	3	2/3	1/3	1/3	5	2/5	3/5	2/5	3
$x_1 < 5.0$	4	3/4	1/4	1/4	4	1/4	3/4	1/4	2
$x_1 < 8.0$	6	3/6	3/6	3/6	2	1/2	1/2	1/2	4

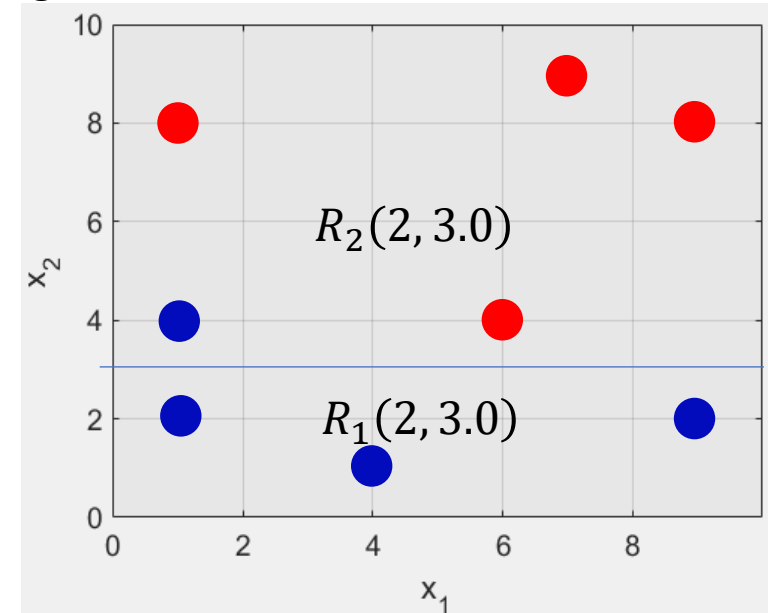
# Calculating the loss for 1<sup>st</sup> split

- Define  $\hat{\pi}_{\ell,m}$  as the proportion of training observations in the  $\ell$ th region of that belong to the  $m$ th class

$$\hat{\pi}_{\ell,m} = \frac{1}{n_{\ell}} \sum_{i: \mathbf{x}_i \in R_{\ell}(j,s)} \mathbb{I}(y_i = m)$$

- Let's use **Misclassification rate**:  $Q_{\ell} = 1 - \max_m \hat{\pi}_{\ell,m}$

- To find the optimal split, we select the values for  $j$  and  $s$  that minimize the loss



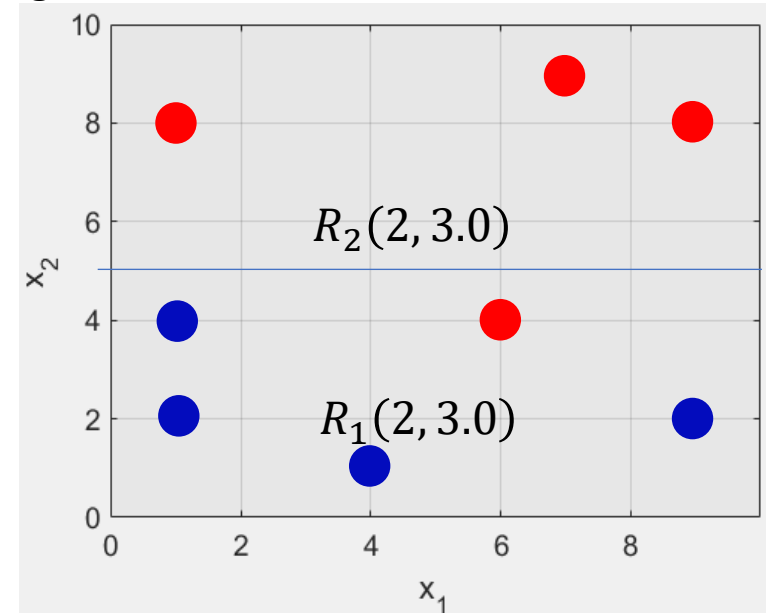
Splits ( $R_1$ )	$n_1$	$\hat{\pi}_{1,B}$	$\hat{\pi}_{1,R}$	$Q_1$	$n_2$	$\hat{\pi}_{2,B}$	$\hat{\pi}_{2,R}$	$Q_2$	$n_1 Q_1 + n_2 Q_2$
$x_1 < 2.5$	3	2/3	1/3	1/3	5	2/5	3/5	2/5	3
$x_1 < 5.0$	4	3/4	1/4	1/4	4	1/4	3/4	1/4	2
$x_1 < 8.0$	6	3/6	3/6	3/6	2	1/2	1/2	1/2	4
$x_2 < 3.0$	3	3/3	0/3	0/3	5	1/5	4/5	1/5	1

# Calculating the loss for 1<sup>st</sup> split

- Define  $\hat{\pi}_{\ell,m}$  as the proportion of training observations in the  $\ell$ th region of that belong to the  $m$ th class

$$\hat{\pi}_{\ell,m} = \frac{1}{n_{\ell}} \sum_{i: \mathbf{x}_i \in R_{\ell}(j,s)} \mathbb{I}(y_i = m)$$

- Let's use **Misclassification rate**:  $Q_{\ell} = 1 - \max_m \hat{\pi}_{\ell,m}$



Splits ( $R_1$ )	$n_1$	$\hat{\pi}_{1,B}$	$\hat{\pi}_{1,R}$	$Q_1$	$n_2$	$\hat{\pi}_{2,B}$	$\hat{\pi}_{2,R}$	$Q_2$	$n_1 Q_1 + n_2 Q_2$
$x_1 < 2.5$	3	2/3	1/3	1/3	5	2/5	3/5	2/5	3
$x_1 < 5.0$	4	3/4	1/4	1/4	4	1/4	3/4	1/4	2
$x_1 < 8.0$	6	3/6	3/6	3/6	2	1/2	1/2	1/2	4
$x_2 < 3.0$	3	3/3	0/3	0/3	5	1/5	4/5	1/5	<b>1</b>
$x_2 < 5.0$	5	4/5	1/5	1/5	3	0/3	3/3	0/3	<b>1</b>



# Learning Classification Trees using Recursive Binary Splitting

Start with the first split at the root and then build the tree from top to bottom

- When determining the splitting rule at the root node, the objective is to obtain a model that best explains the training data after a single split, without taking into consideration that additional splits may be added before arriving at the final model
  - Select one of the  $p$  input variables  $x_1, \dots, x_p$  and a corresponding cutpoint  $s$  which divide the input space into two half-spaces

$$R_1(j, s) = \{ \mathbf{x} \mid x_j < s \} \text{ and } R_2(j, s) = \{ \mathbf{x} \mid x_j \geq s \}$$

- The predictions associated with the two regions will be

$$\hat{y}_1(j, s) = \text{Mode}\{y_i : \mathbf{x}_i \in R_1(j, s)\} \text{ and } \hat{y}_2(j, s) = \text{Mode}\{y_i : \mathbf{x}_i \in R_2(j, s)\}$$

- Compute *loss* (squared error) for all training data points  $(\mathbf{x}_i, y_i)_{i=1}^N$  at the node to determine goodness-of-fit

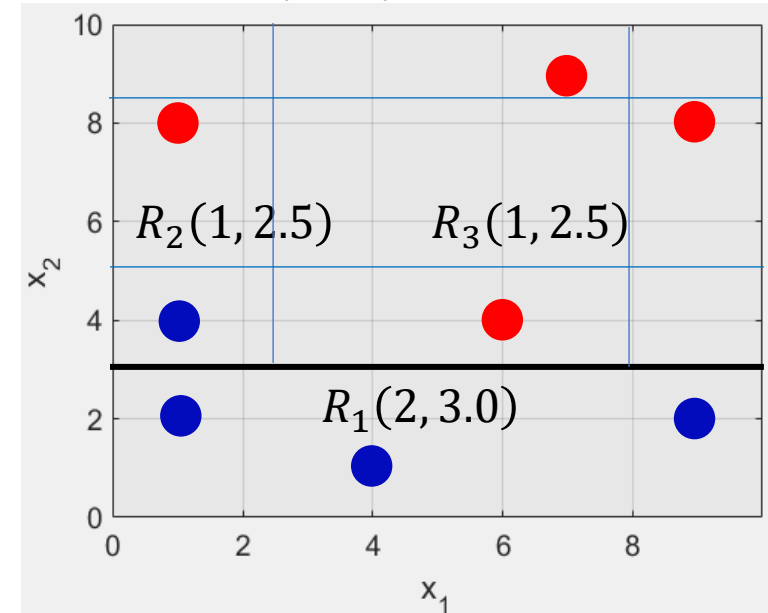
$$n_1 Q_1 + n_2 Q_2$$

where,  $n_1, n_2$  are # of data points in left and right nodes of current split and  $Q_1, Q_2$  are associated prediction errors

- To find the optimal split, we select the values for  $j$  and  $s$  that minimize the loss
- When we have decided on the first split of the input space (corresponding to the root node of the tree), this split is kept fixed, and we continue in a similar way for the two resulting half-spaces (corresponding to the two branches of the tree)

# Calculating the loss for 2<sup>nd</sup> split

- There is no point splitting region  $R_1$  further since it only contains data points from the same class (Blue)
- We therefore split the upper region into two new regions,  $R_2$  and  $R_3$
- We split in the same manner as before



Splits ( $R_2$ )	$n_2$	$\hat{\pi}_{2,B}$	$\hat{\pi}_{2,R}$	$Q_2$	$n_3$	$\hat{\pi}_{3,B}$	$\hat{\pi}_{3,R}$	$Q_3$	$n_2Q_2 + n_3Q_3$
$x_1 < 2.5$	2	1/2	1/2	1/2	3	0/3	3/3	0/3	1
$x_1 < 8.0$	4	1/4	3/4	1/4	1	0/1	1/1	0/1	1
$x_2 < 5.0$	2	1/2	1/2	1/2	3	0/3	3/3	0/3	1
$x_2 < 8.5$	4	1/4	3/4	1/4	1	0/1	1/1	0/1	1

# Learning Regression Trees using Recursive Binary Splitting

Start with the first split at the root and then build the tree from top to bottom

- When determining the splitting rule at the root node, the objective is to obtain a model that best explains the training data after a single split, without taking into consideration that additional splits may be added before arriving at the final model
  - Select one of the  $p$  input variables  $x_1, \dots, x_p$  and a corresponding cutpoint  $s$  which divide the input space into two half-spaces

$$R_1(j, s) = \{ \mathbf{x} \mid x_j < s \} \text{ and } R_2(j, s) = \{ \mathbf{x} \mid x_j \geq s \}$$

- The predictions associated with the two regions will be

$$\hat{y}_1(j, s) = \text{Mean}\{y_i : \mathbf{x}_i \in R_1(j, s)\} \text{ and } \hat{y}_2(j, s) = \text{Mean}\{y_i : \mathbf{x}_i \in R_2(j, s)\}$$

- Compute *loss* (squared error) for all training data points  $(\mathbf{x}^{(i)}, y^{(i)})_{i=1}^N$  at the node to determine goodness-of-fit

$$n_1 Q_1 + n_2 Q_2 = \sum_{i: \mathbf{x}_i \in R_1(j, s)} (y_i - \hat{y}_1(j, s))^2 + \sum_{i: \mathbf{x}_i \in R_2(j, s)} (y_i - \hat{y}_2(j, s))^2$$

where,  $n_1, n_2$  are # of data points in left and right nodes of current split and  $Q_1, Q_2$  are associated prediction errors

- To find the optimal split, we select the values for  $j$  and  $s$  that minimise the loss
- When we have decided on the first split of the input space (corresponding to the root node of the tree), this split is kept fixed, and we continue in a similar way for the two resulting half-spaces (corresponding to the two branches of the tree)

# Algorithm for CART using Recursive Binary Splitting

**Data:**  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$

**Output:** Decision tree with regions  $R_1, R_2, \dots, R_L$  and corresponding predictions  $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_L$

1. Let  $R$  denote the entire input space
2. Compute the regions  $(R_1, \dots, R_L) = \mathbf{split}(R, \mathcal{T})$
3. Compute the predictions  $\hat{y}_\ell$  for  $\ell = 1, 2, \dots, L$  as

$$\hat{y}_\ell = \begin{cases} \text{Mean}\{y_i: \mathbf{x}_i \in R_\ell\} & \text{(Regression)} \\ \text{Mode}\{y_i: \mathbf{x}_i \in R_\ell\} & \text{(Classification)} \end{cases}$$

**Test Data:**  $\mathbf{x}_*$

**Output:**  $\hat{y}(\mathbf{x}_*)$

1. Find the region  $R_\ell$  in which  $\mathbf{x}_*$  belongs to
2. Return the prediction  $\hat{y}(\mathbf{x}_*) = \hat{y}_\ell$

function **split**( $R_\ell, \mathcal{T}_\ell$ )

*if stopping criterion fulfilled*

**return**  $R_\ell$

*else*

Go through all possible splits  $x_j < s$  for all input variables

Pick pair  $(j, s)$  that minimizes the chosen loss

Split the region  $R_\ell$  into two half-spaces  $R_\ell$  and  $R_{\ell+1}$

Split the data  $\mathcal{T}_\ell$  into two parts,  $\mathcal{T}_\ell$  and  $\mathcal{T}_{\ell+1}$ , respectively

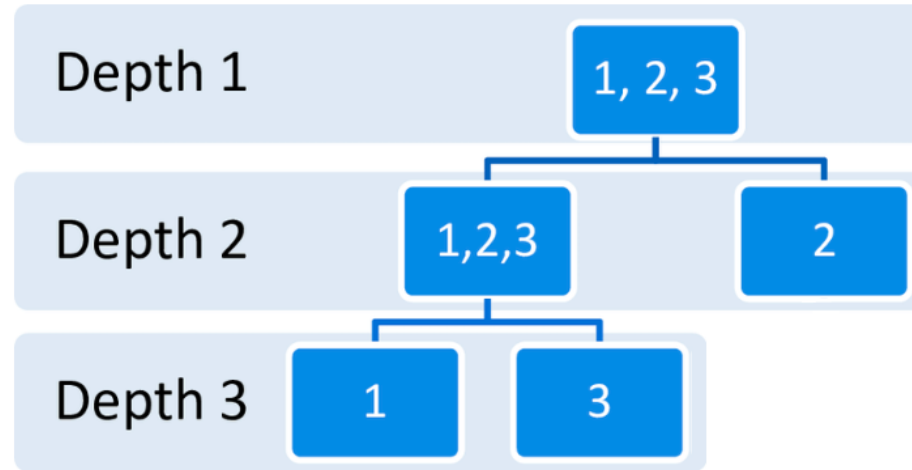
**return**  $(R_\ell, \mathcal{T}_\ell)$  and  $(R_{\ell+1}, \mathcal{T}_{\ell+1})$

**Example of stopping criterion**

No further splits if there are less than a certain number of training data points in the corresponding region

# How deep should be a decision tree?

- The depth of a decision tree (the maximum distance between the root node and any leaf node) has a big impact on the final predictions



- The tree depth impacts the predictions in a somewhat similar way to the hyperparameter  $k$  in  $k$ -NN
- **Not too shallow (or small):** Underfits. Need to fit more subtle distinctions in data
- **Not too deep (or big):** Avoid over-fitting training examples
- **Optimal tree depth (or size):** Is a trade-off between flexibility and rigidity of the final model
- Typically, we desire small trees with informative nodes near the root

# $k$ NN vs Decision Trees

- **Advantages of Decision trees over  $k$ NNs**

- Simple to deal with poorly scaled data
- Fast at test time (no need to calculate distances like in  $k$ NN)
- More interpretable

- **Advantages of  $k$ NNs over Decision trees**

- Fewer hyperparameters (need to decide on just the value of  $k$ )
- Can incorporate interesting distance measures