INTRODUCTION TO MACHINE LEARNING METHODS

Toni Šćulac

Faculty of Science, University of Split, Croatia

Johns Hopkins University, Baltimore, USA

tCSC Machine Learning 2025, Malmo, Sweden

LECTURES OUTLINE

1) Introduction to Statistics
2) Statistics and Machine Learning
3) Classical Machine Learning
4) Introduction to Deep Learning
5) Advanced Deep Learning

CLASSICAL MACHINE LEARNING

*inspiration and examples from [M. Donega] and [MM]

ML GOAL REMINDER

- The optimal test statistic for hypothesis testing:
  - $$t(x) = \frac{L(x\,|\,H_1)}{L(x\,|\,H_0)} > c$$
- Very often unobtainable in real-life cases. One of the biggest problem is that x^N is very often multidimensional (N >> 1)
- Machine Learning is our effort to approximate the likelihood ratio (LR)
  - One of key ideas is to build an algorithm that can “learn” the likelihood from training data and then apply the LR test statistic to distinguish data from different hypothesis with (close to) optimal performance

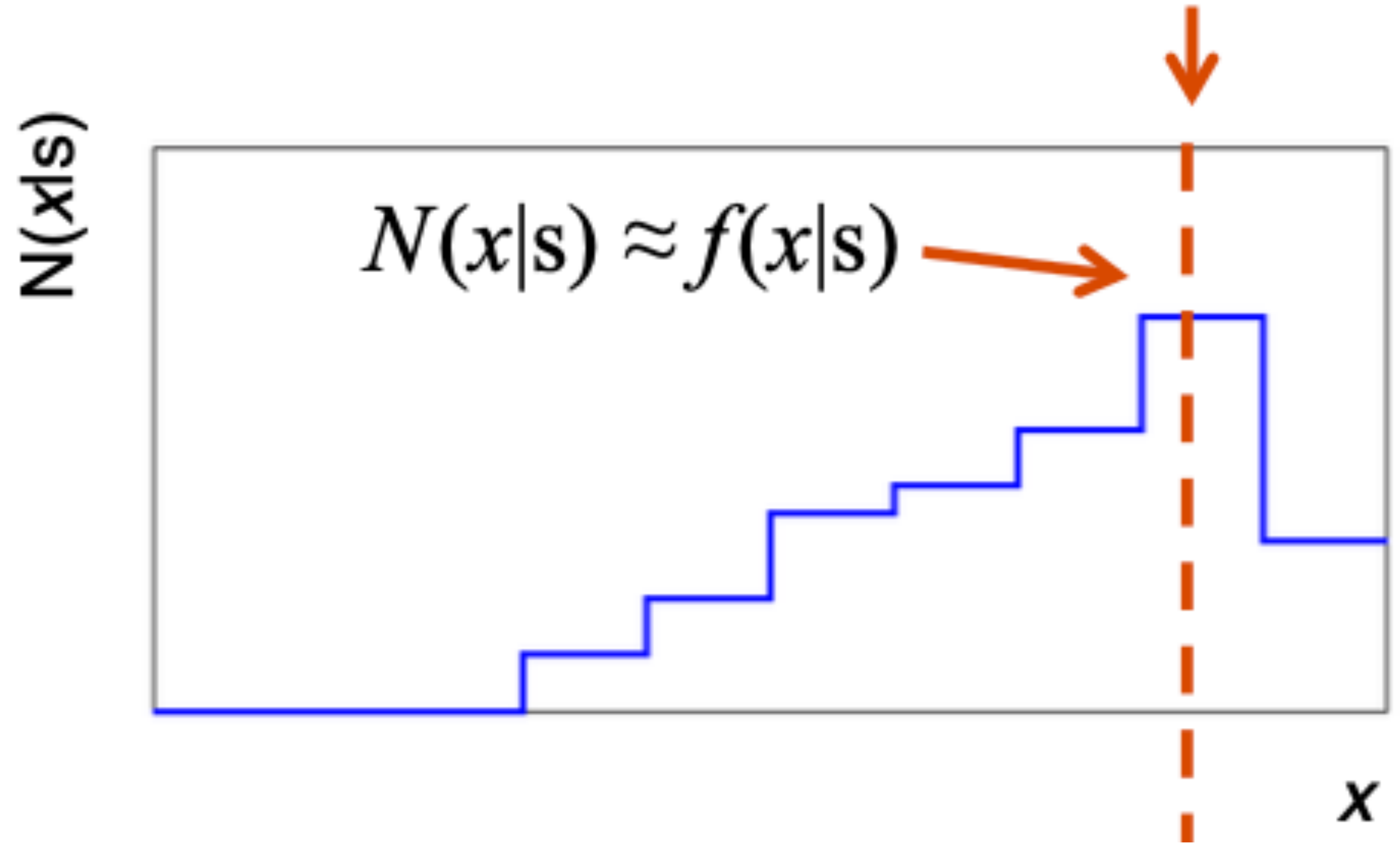
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
f(x|H₀)
critical region W
f(x|H₁)
x

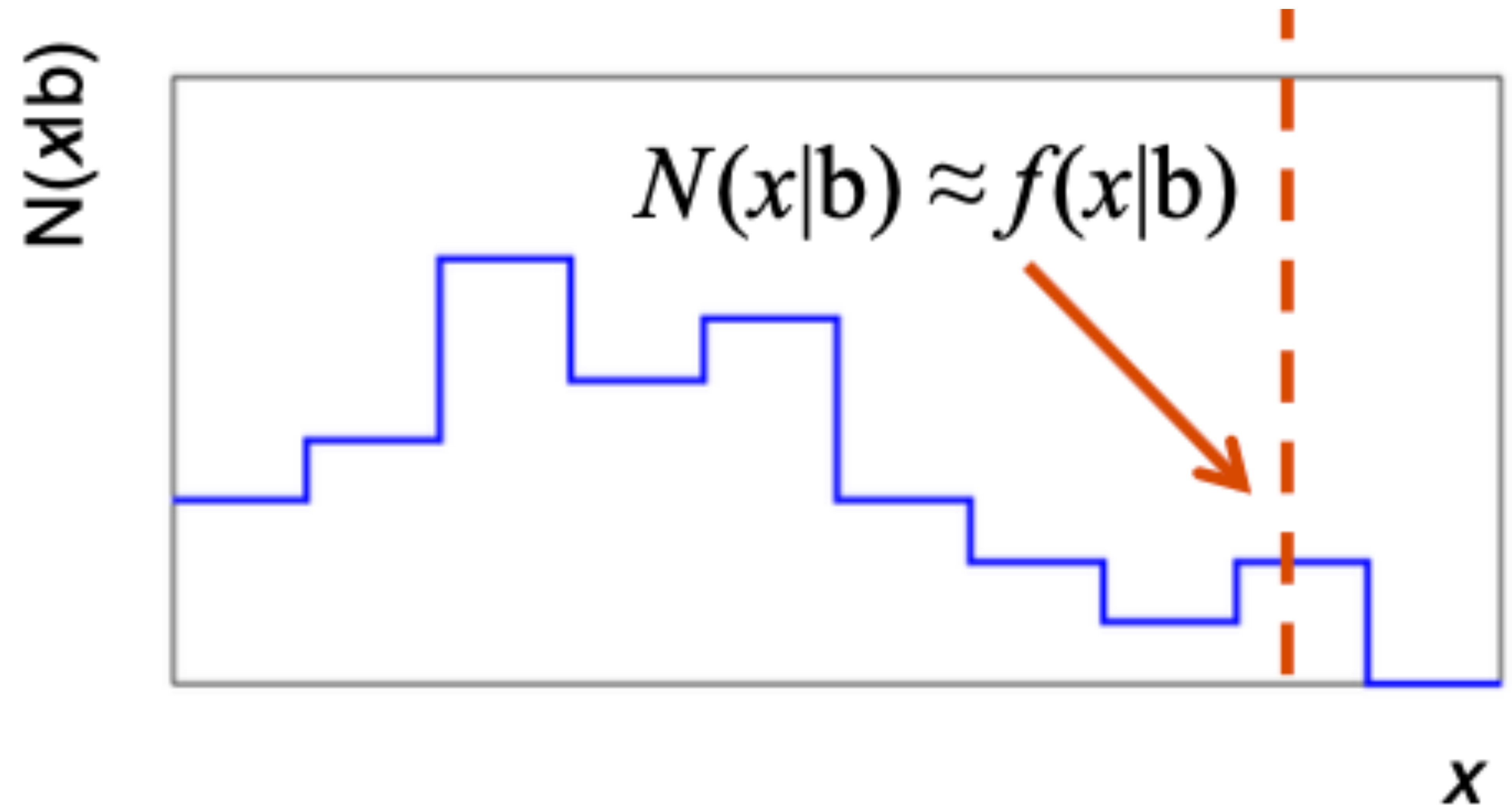
BLACK BOX

Machine Learning
AI
BDT
Neural Networks

LIKELIHOOD RATIO - HISTOGRAMS

- We usually don't have explicit formulae for the pdfs $f(x\,|\,s)$, $f(x\,|\,b)$, so for a given x we can't evaluate the LR
- Instead we may have MC models for signal and background processes, so we can produce simulated data $x^s(x_1^s, \ldots, x_N^s)$ and $x^b(x_1^b, \ldots, x_N^b)$
  - Can be expensive (1 fully simulated LHC event ~ 1 CPU minute).
- One possibility is to generate MC data and construct histograms for both signal and background and use them to approximate the Likelihood ratio

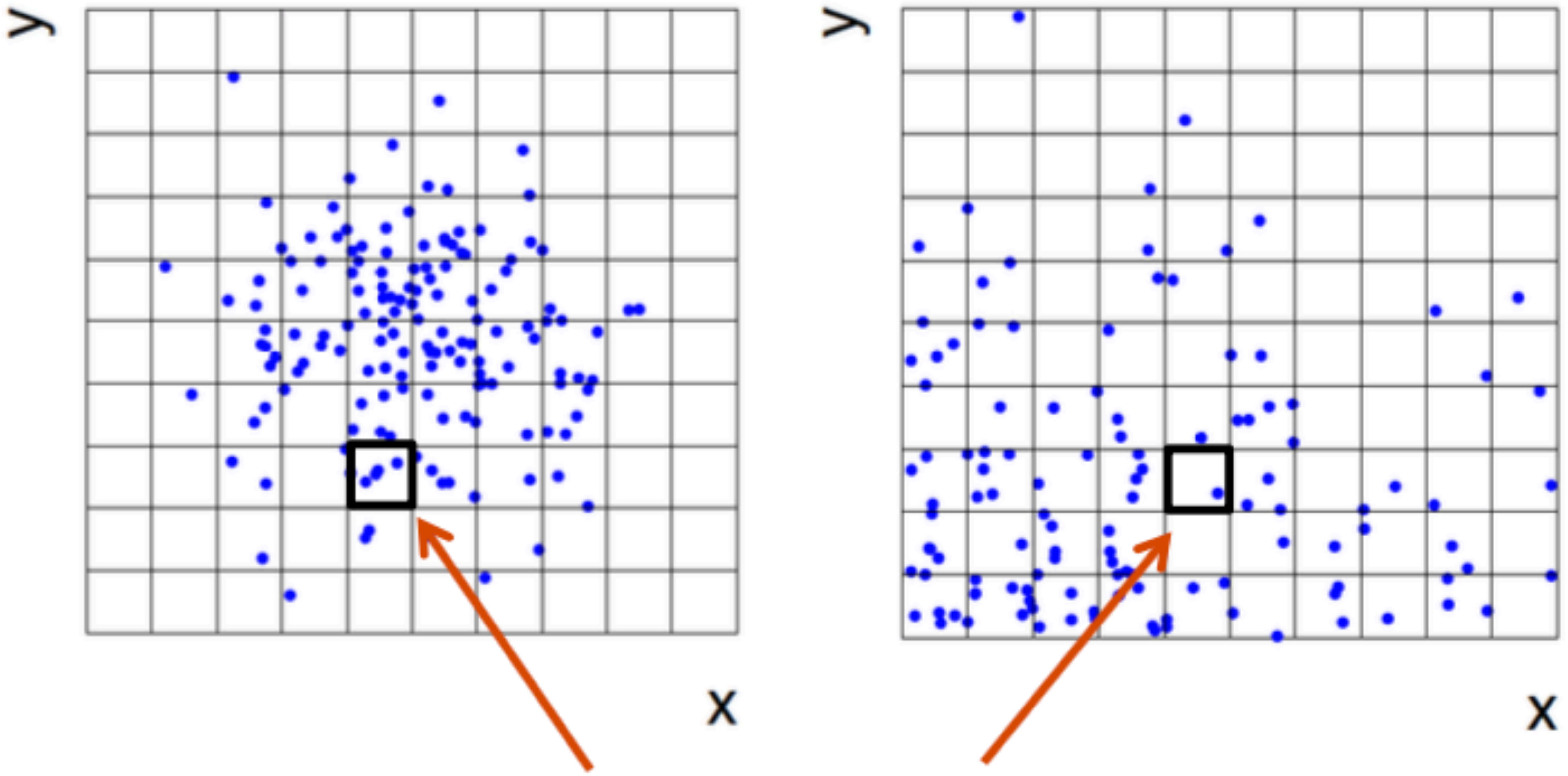




$$t(x) = \frac{N(x\,|\,s)}{N(x\,|\,b)}$$

LIKELIHOOD RATIO - MULTIDIMENSIONAL

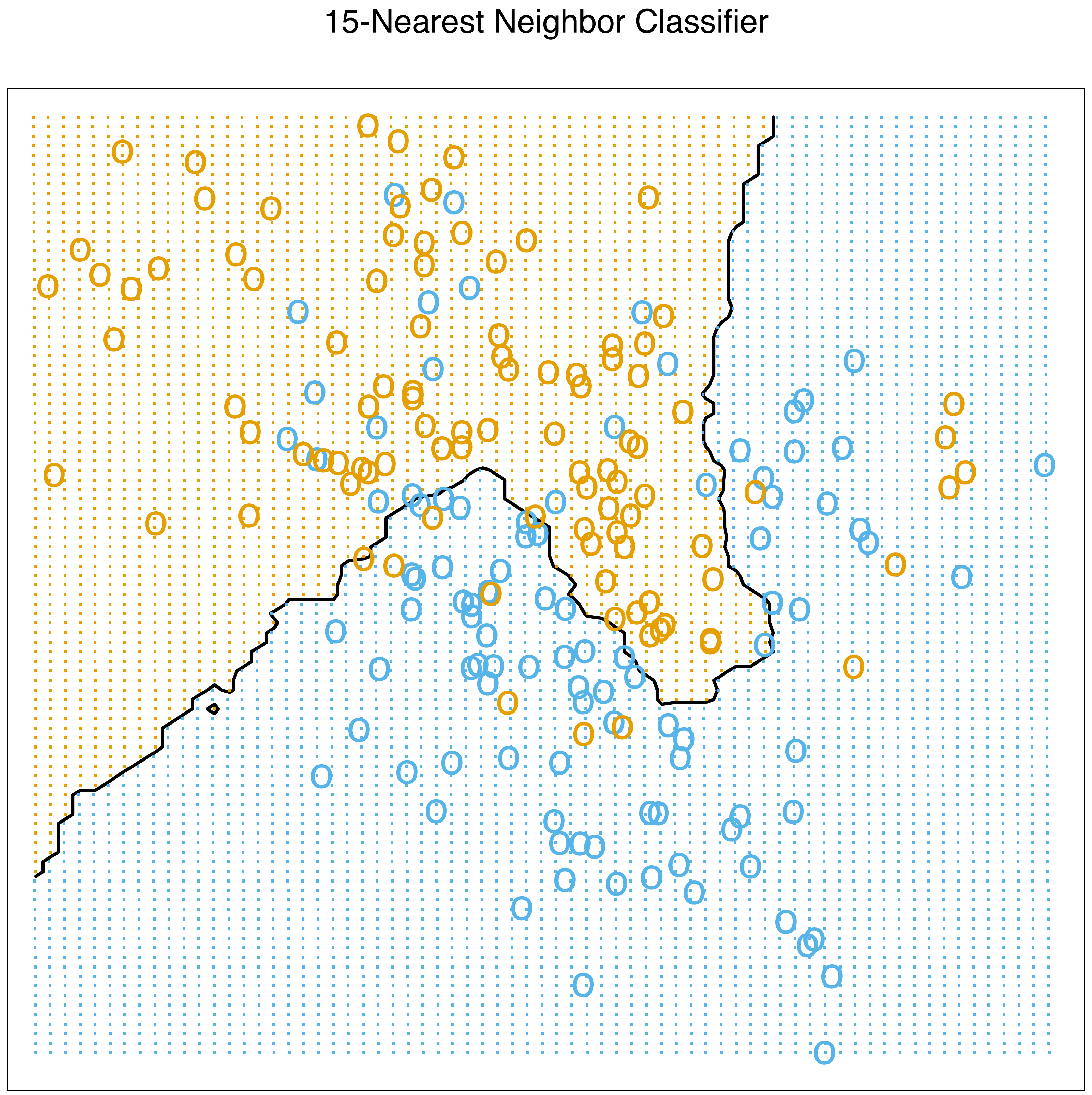
- Suppose problem has 2 variables:
  - Approximate pdfs using $N(x, y\,|\,s)$, $N(x, y\,|\,b)$ in corresponding bins
  - But if we want M bins for each variable, then in N-dimensions we have M^N cells; can't generate enough training data to populate



- The idea is to try to estimate the probability densities $f(x\,|\,s)$ and $f(x\,|\,b)$ with something better than histograms and use the estimated PDFs to construct an approximate likelihood ratio.

K-CLOSEST NEIGHBOURS

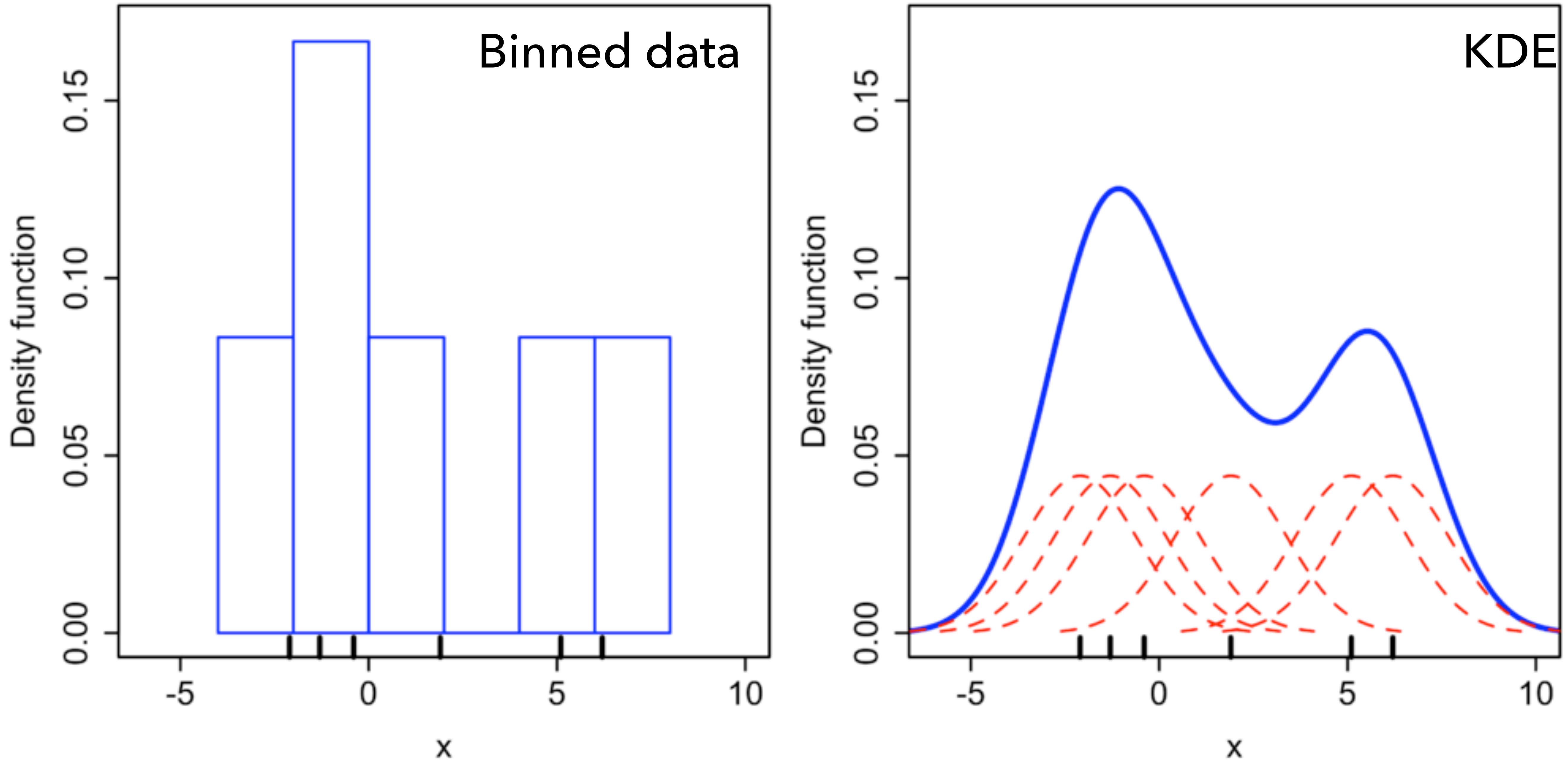
- We count the number of events in local neighbourhood of x and do a **majority vote**
- The distance definition is important
  - can be tricky in case of multidimensional problems with variables in different units and ranges
- Consider a small volume **V** centered around $x = (x_1, \cdots, x_D)$
- Suppose from **N** events we find **K** inside volume **V**
- We can estimate PDF as:
  - $\hat{p}(x) = \dfrac{K}{NV}$
- There are 2 free parameters that we introduced (**K**,**V**). If you **fix K** and determine V from data it is called **K-nearest neighbours**



KERNEL DENSITY ESTIMATOR

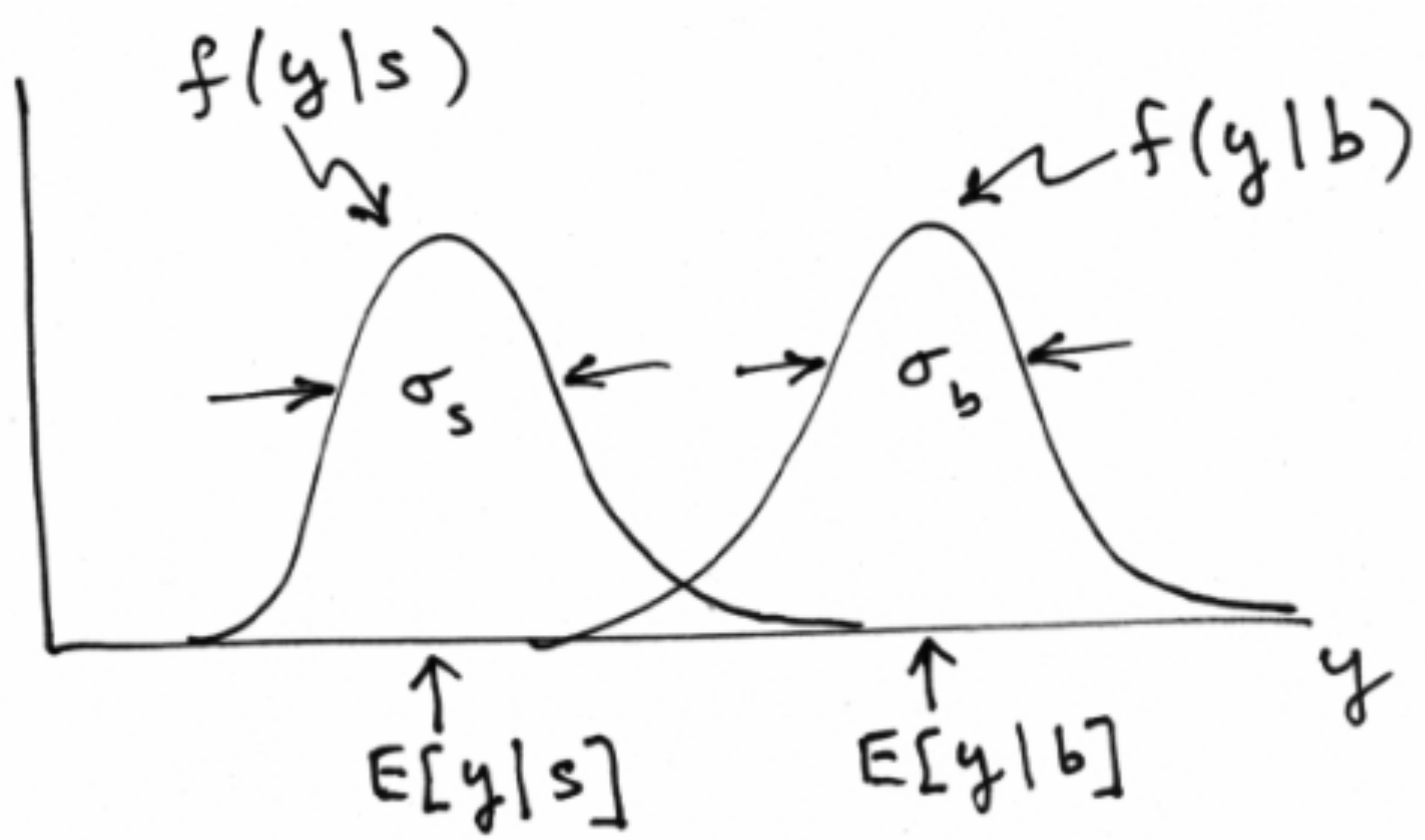
- There are 2 free parameters that we introduced (**K**,**V**). If you **fix V** and determine K from data it is called **Kernel Density Estimator**
- We have to define a volume around a point
- Let’s look at a simple example of placing a 1d Gaussian “kernel” with a standard deviation **h** centred around each point

- We can estimate PDF as:
- $$p(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\sqrt{2\pi h^2}}e^{-\frac{(\vec{x}-\vec{x_i})^2}{2h^2}}$$
- This very smart method again suffers from the problem of scaling with dimensionality…



FISHER DISCRIMINANT

- Consider a linear function of **n** input variables $y(x) = \sum_{i=1}^{n} w_i x_i$
- For a given choice of the coefficients $w(w_1, \ldots, w_n)$ and two different samples (signal and background) we will get PDFs $f(y\,|\,s)$ and $f(y\,|\,b)$



- The Fisher method aims at finding an hyperplane in the **n-dimensional space** which gives the best expected separation between two random variables sets whose PDFs in the multivariate space are known.

FISHER DISCRIMINANT

- To get large difference between means and small widths for $f(y \mid s)$ and $f(y \mid b)$, maximise the difference squared of the expectation values divided by the sum of the variances:

$$J(w) = \frac{(E(y \mid s) - E(y \mid b))^2}{V(y \mid s) + V(y \mid b)} \qquad \frac{\partial J(w)}{\partial w} = 0$$

- The resulting coefficients w_i define **a Fisher discriminant**.

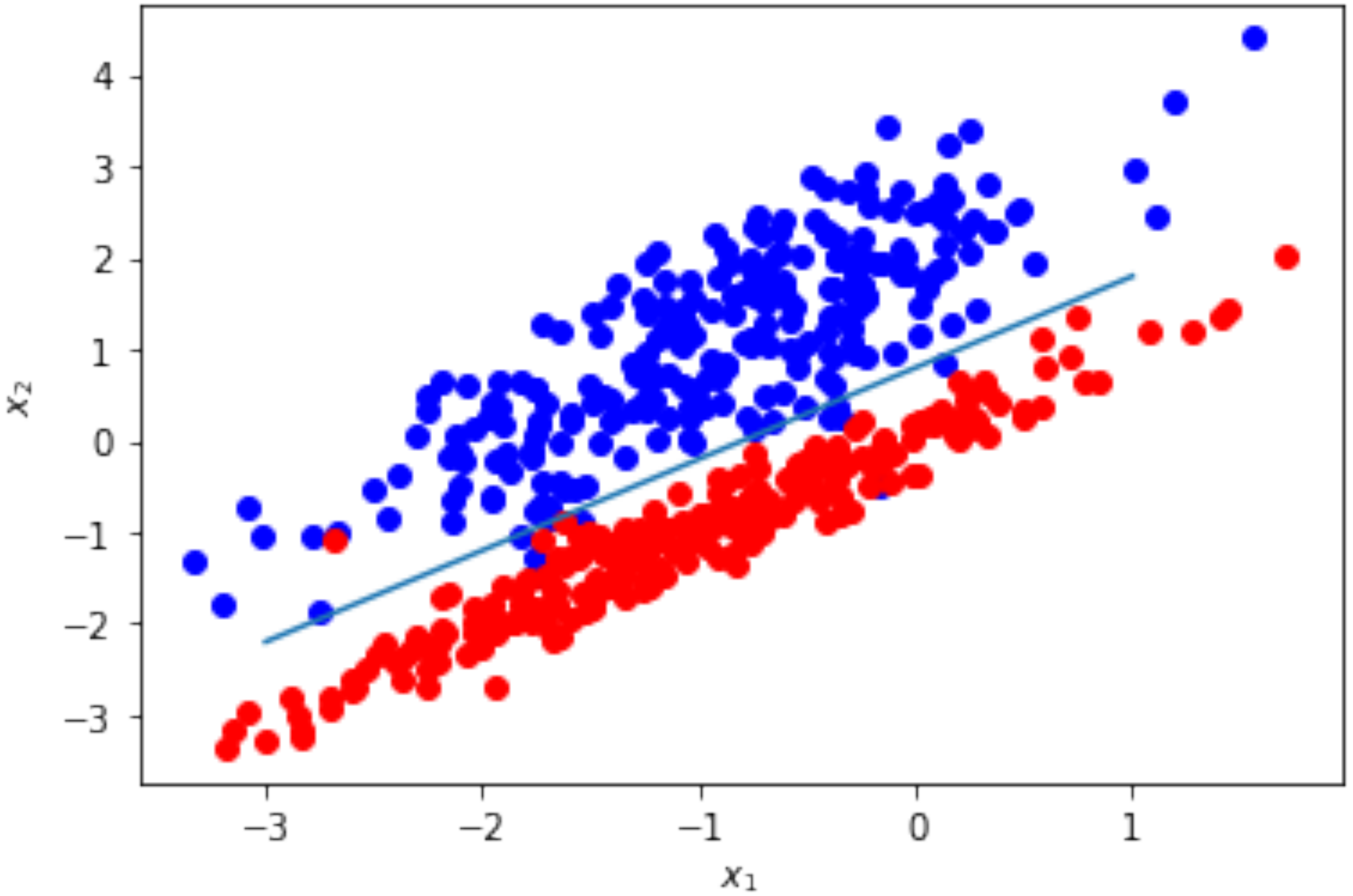
- In the case when $y(x) = w_1 x_1 + w_2 x_2$, the solution for weights that minimizes the Fisher discriminant is:

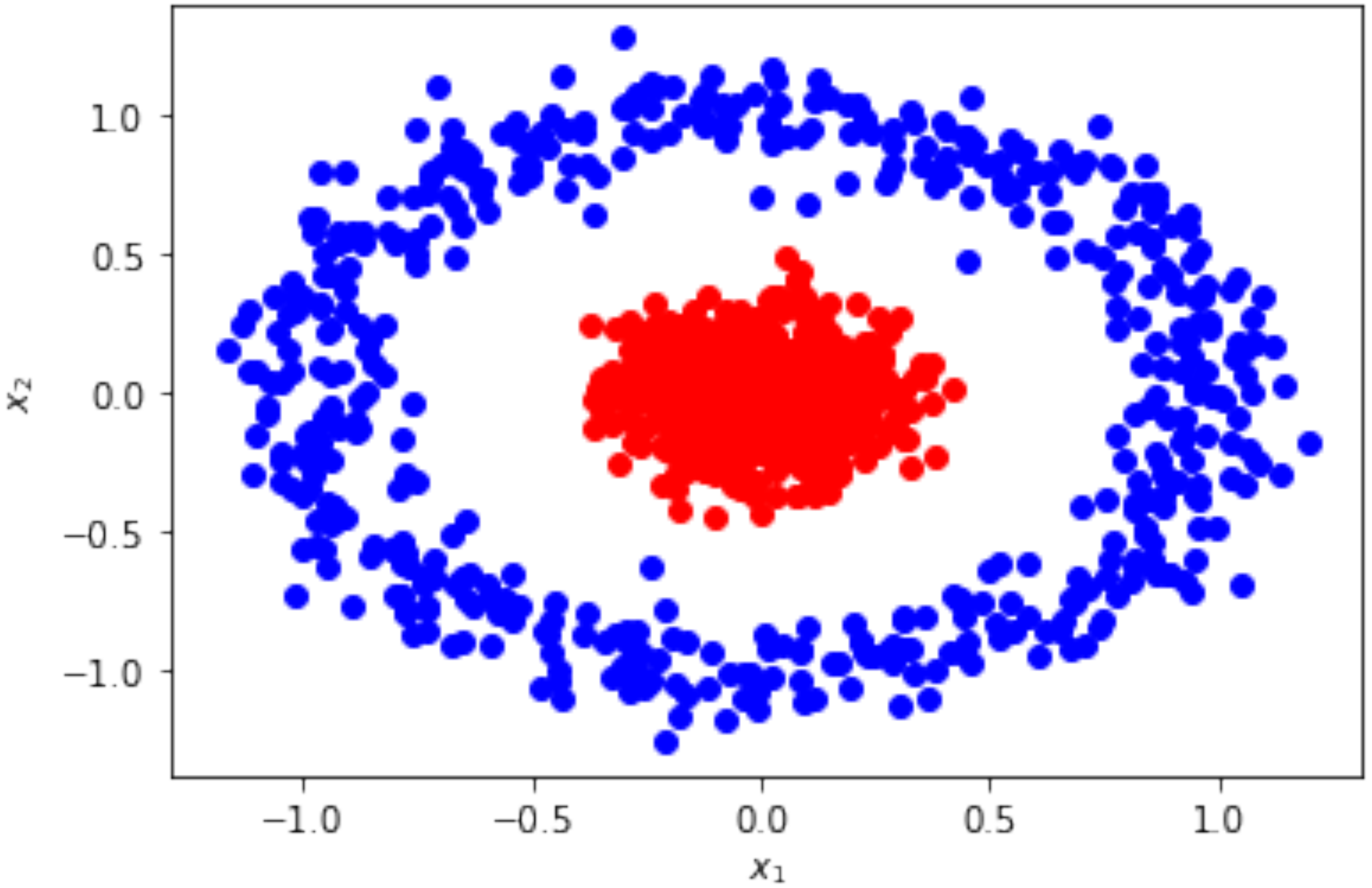
$$\overrightarrow{w} = (E(\vec{x} \mid s) - E(\vec{x} \mid b)) \cdot \mathrm{cov}^{-1}(\vec{\mathrm{x}}) = (\mathrm{w}_1, \mathrm{w}_2)$$

$$\mathrm{cov}(\vec{\mathrm{x}}) = \mathrm{cov}(\vec{\mathrm{x}} \mid \mathrm{S}) + \mathrm{cov}(\vec{\mathrm{x}} \mid \mathrm{B})$$

FISHER DISCRIMINANT

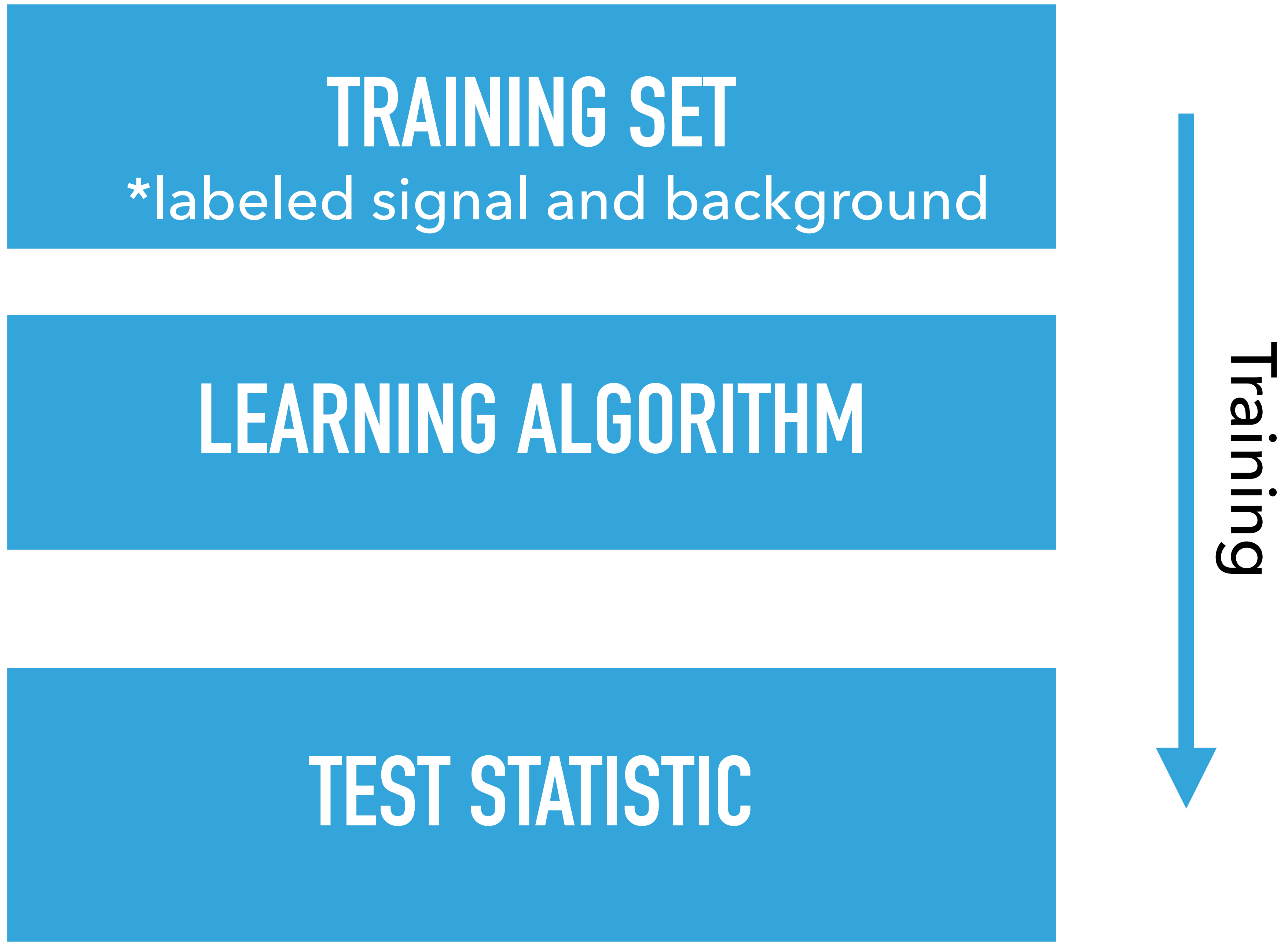
- Fisher discriminant is only optimal when signal and background are Gaussians with same covariance and different means
- Can be almost useless in other cases
- We can try finding a transformation $x(x_1, \ldots, x_N) \rightarrow \Phi(x)(\Phi_1(x), \ldots, \Phi_N(x))$ that transformed variables can be separated with the Fisher discriminants
    - Unreliable and often impossible to guess





LEARNING ALGORITHM

- General idea is to simply learn the PDF from given data



TRAINING SET

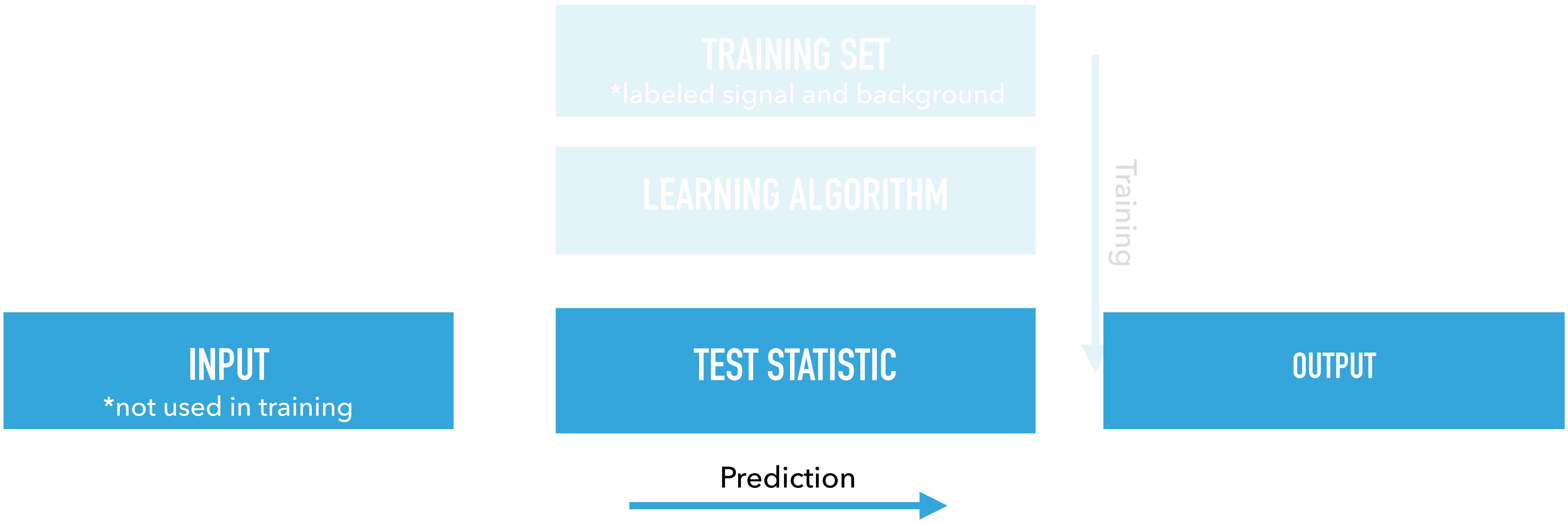
*labeled signal and background

LEARNING ALGORITHM

TEST STATISTIC

LEARNING ALGORITHM

- General idea is to simply learn the PDF from given data
- And use it to find (close to) optimal test statistic for future hypothesis testing



TRAINING SET

*labeled signal and background

LEARNING ALGORITHM

Training

INPUT

*not used in training

TEST STATISTIC

OUTPUT

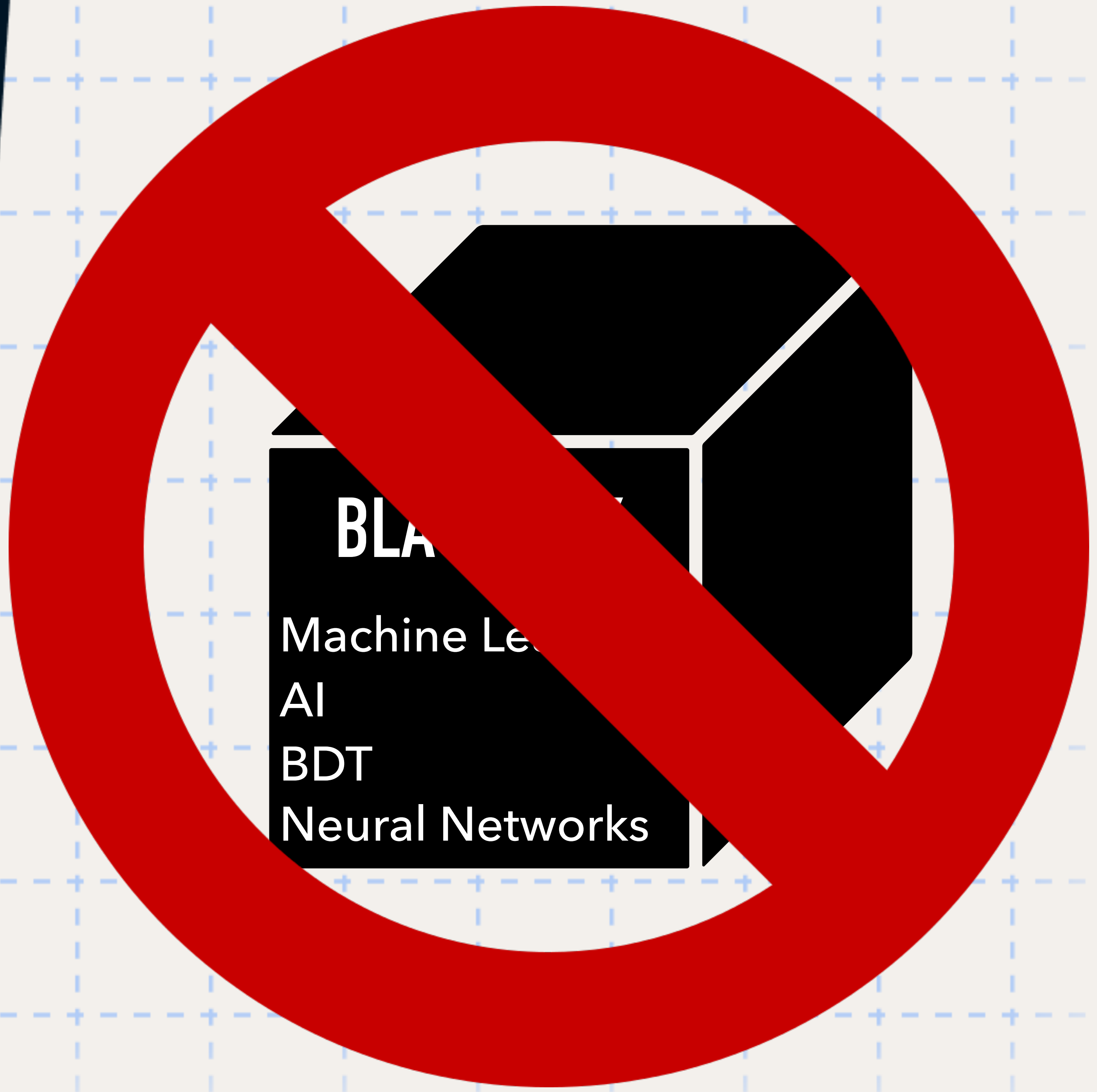
Prediction

DECISION TREES

- **Classification tree** analysis is when the predicted outcome is the class (discrete) to which the data belongs
  - For example if a particle is electron or not
- **Regression tree** analysis is when the predicted outcome can be considered a real number
  - For example, four-momentum of electron
- Training and testing data sample needs to be provided
- During training different weights are assigned to each node according to importance (discriminating power) of different variables
  - Decision trees are a complicated combination of linear cuts that achieve optimal separation power
- Additional techniques like **boosting**, **gradient boosting**, **pruning** used to further improve stability and performance

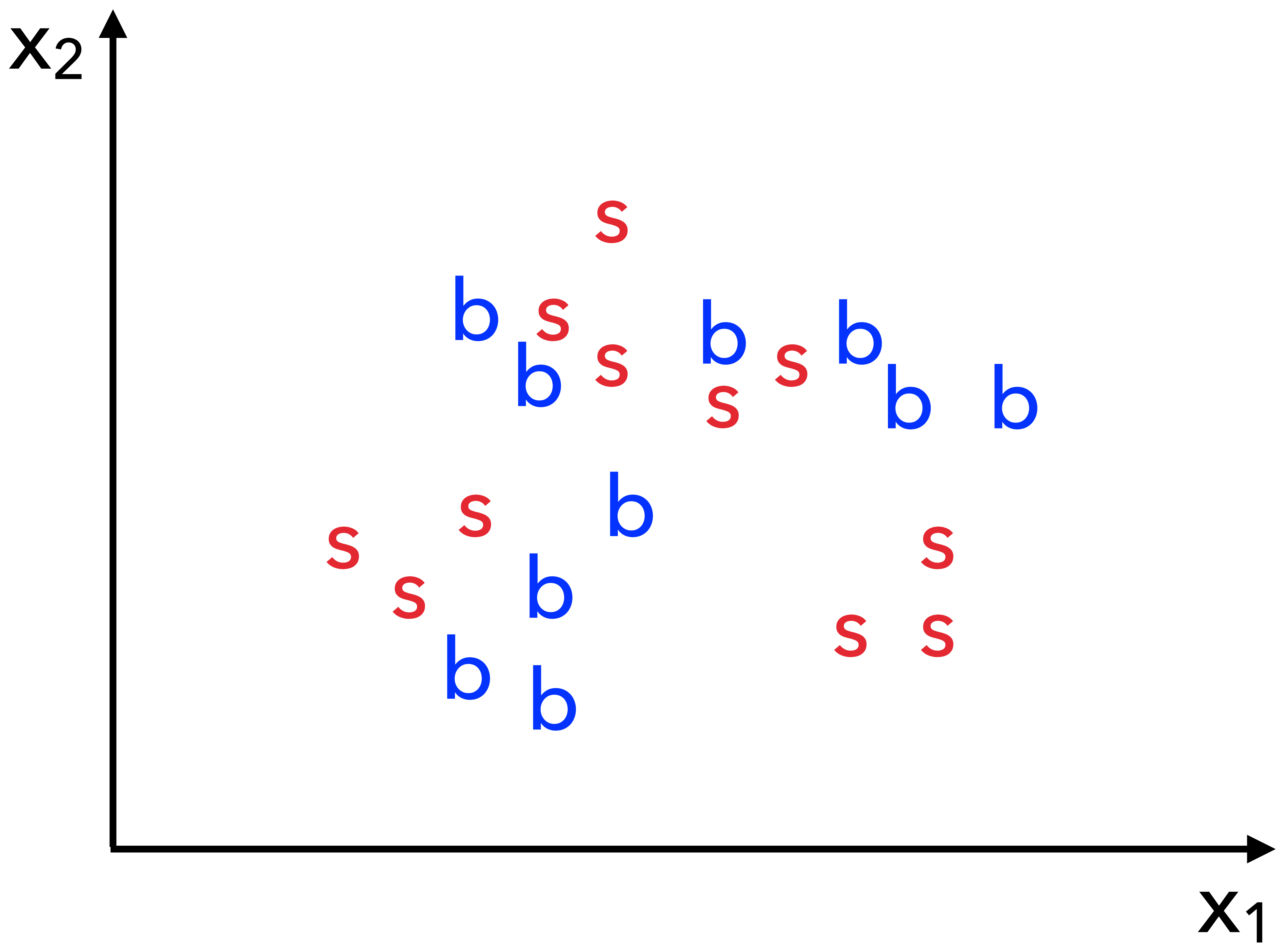
DATASHEET

Machine learning demystified



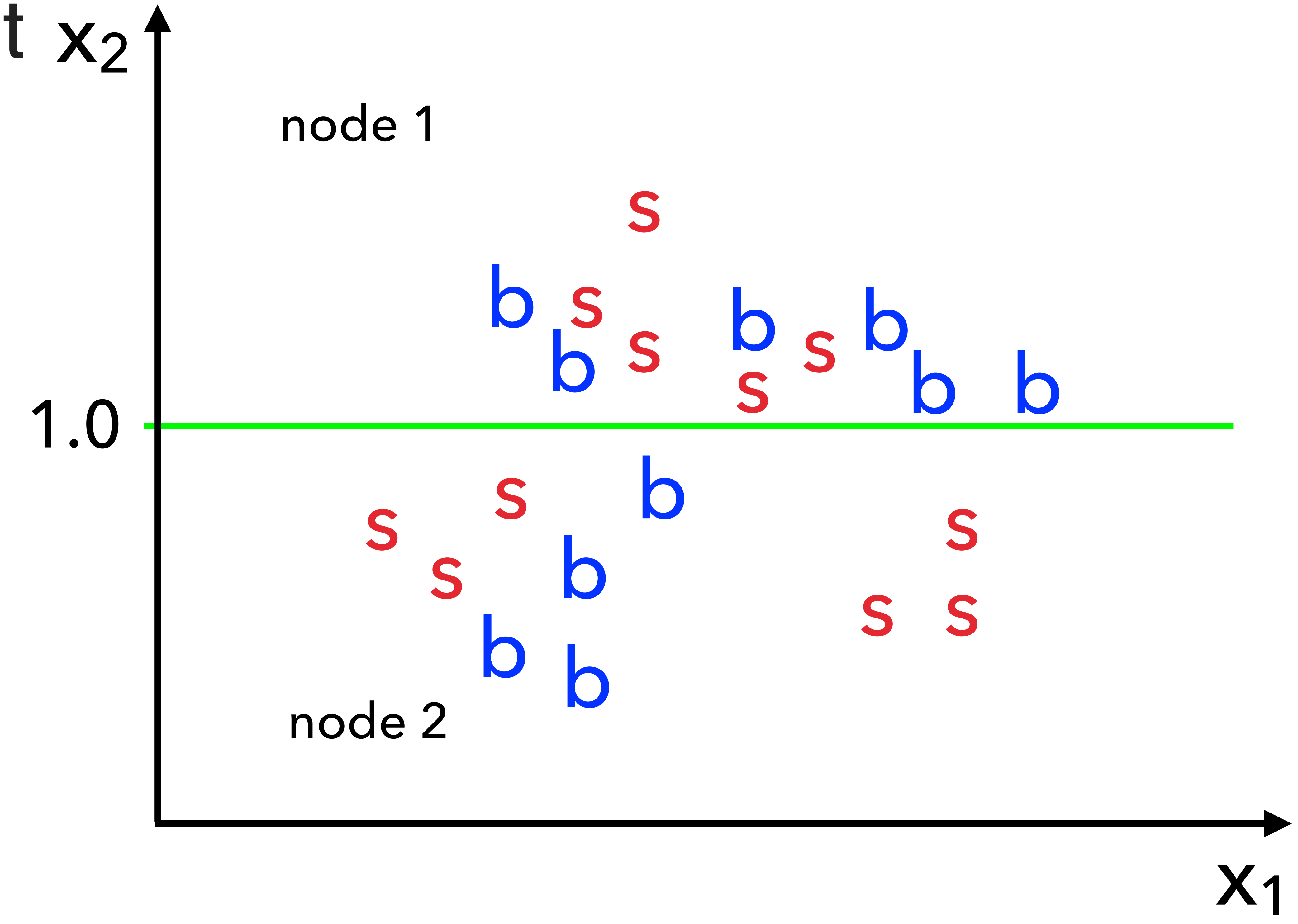
DECISION TREES CLASSIFICATION

- We are given a 2D training dataset (x_1,x_2) and each point is labeled as signal or background
- Idea is to separate the classes placing simple cuts (binary decisions):
  - $x_i > \text{cut}$ or $x_i < \text{cut}$
- Do this in several steps making sure each time misclassification is minimised



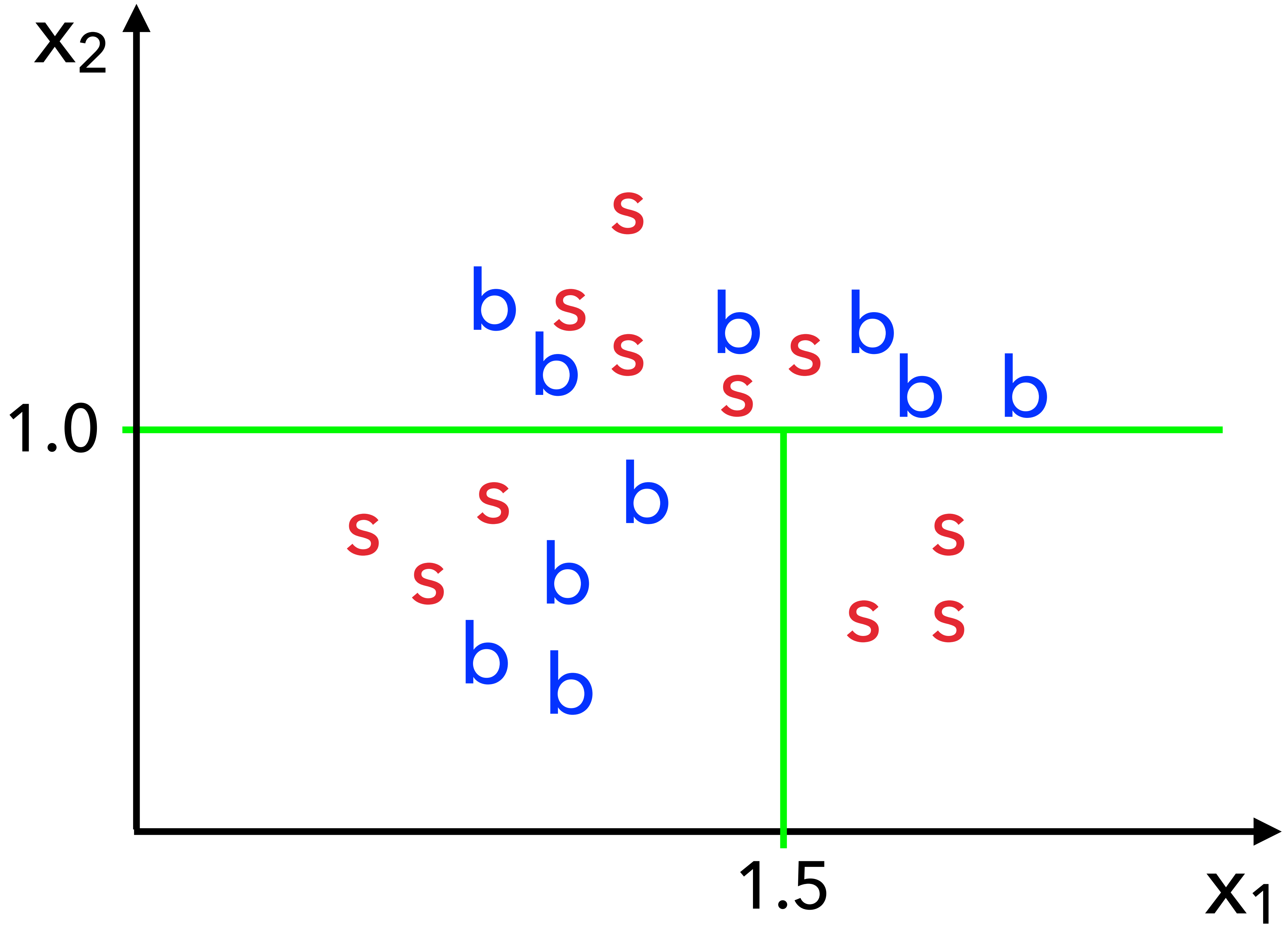
DECISION TREES CLASSIFICATION

- Choose the variable that provides the greatest increase in the separation measured in the two daughter nodes relative to the parent.
  - The same variable may be used at several nodes or ignored
- Define a metric for the separation:
  - the Gini index= P(1-P); $P = \frac{\sum_{\text{sig}} w_i}{\sum_{\text{sig}} w_i + \sum_{\text{bkg}} w_i}$
- The Gini index has a maximum for P=0.5 (random separation) and minimum for P=1 or 0 (maximum separation).
- In this example:
  - Gini index of parent node is **0.249**
  - Node 1 = **0.248** and Node 2 = **0.240**



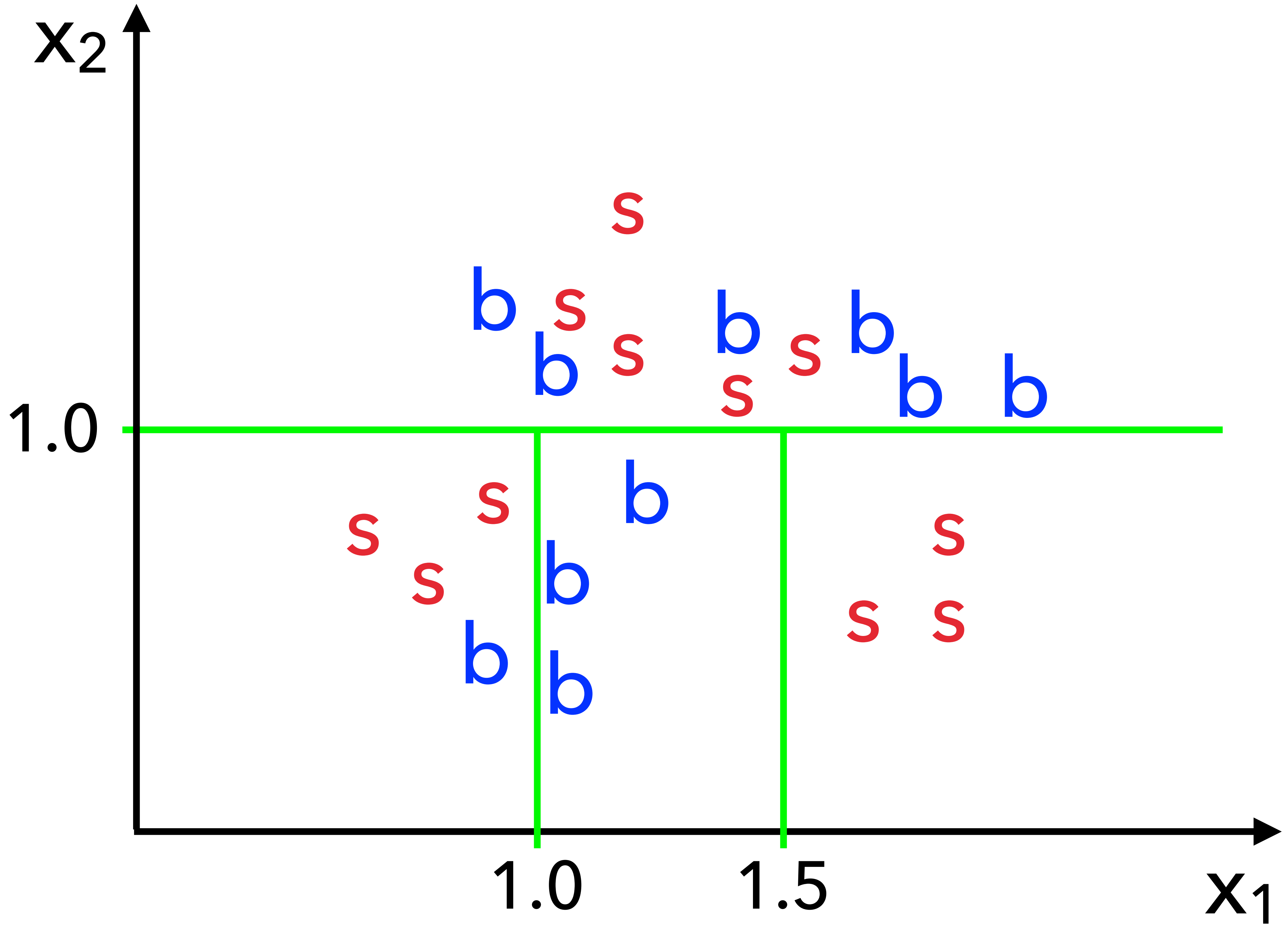
DECISION TREES CLASSIFICATION

- Continue by dividing further new nods with additional simple (binary) cuts that minimise the misclassification



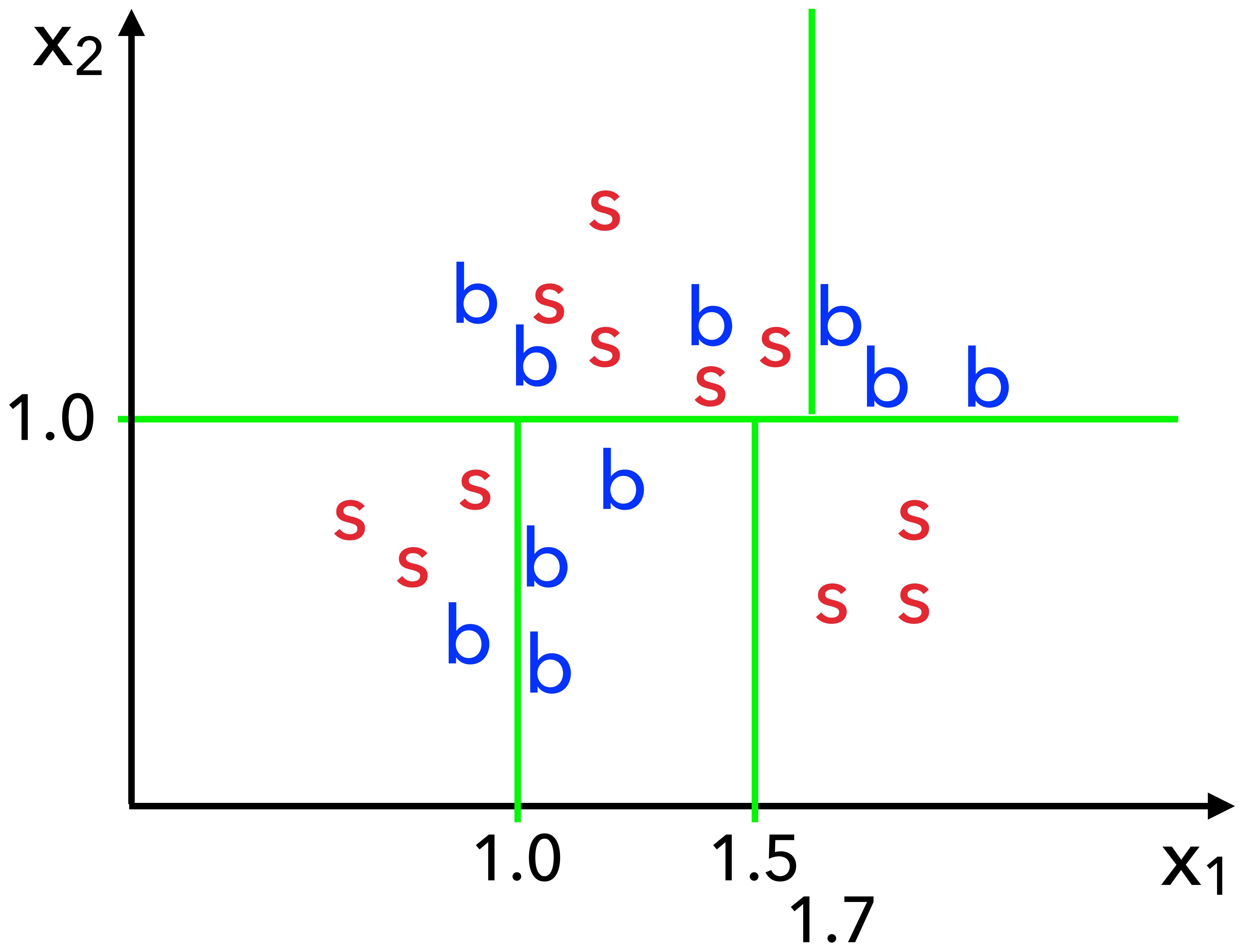
DECISION TREES CLASSIFICATION

- Continue by dividing further new nods with additional simple (binary) cuts that minimise the misclassification



DECISION TREES CLASSIFICATION

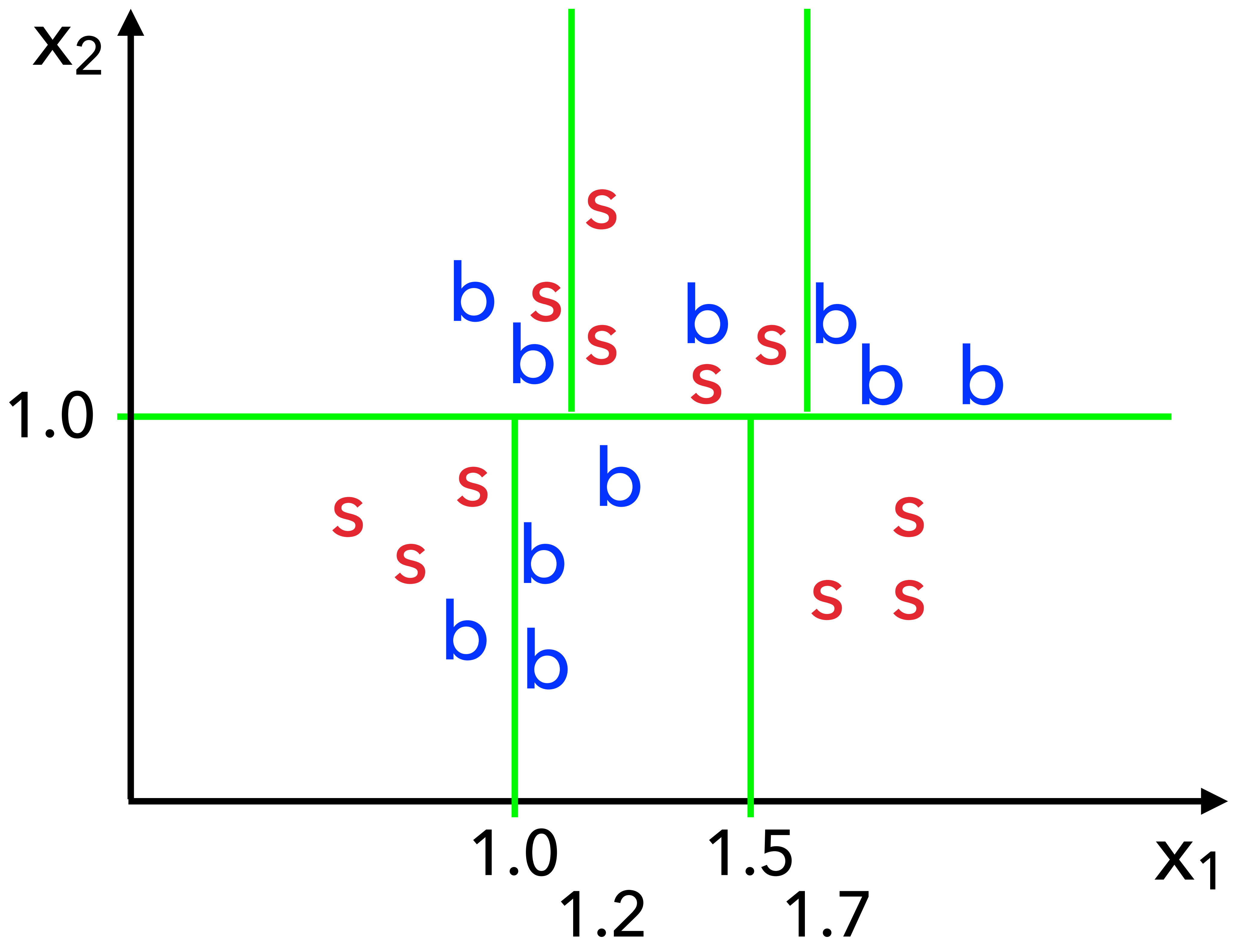
- Continue by dividing further new nods with additional simple (binary) cuts that minimise the misclassification



DECISION TREES CLASSIFICATION

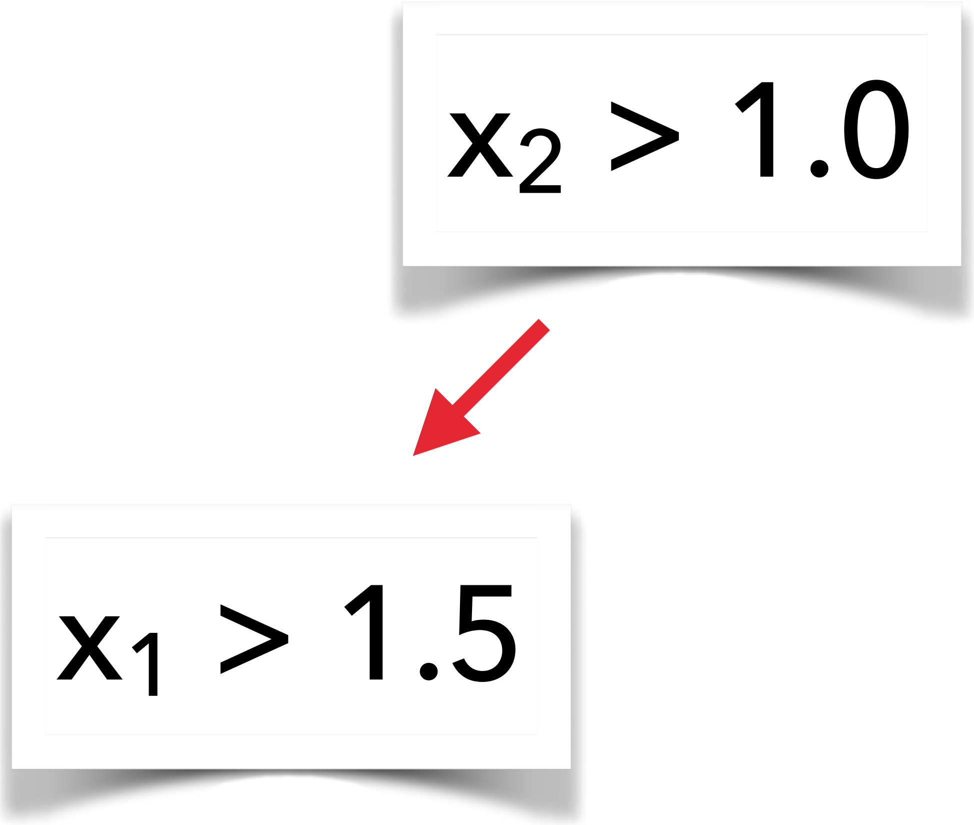
- Finish once every region contains a “minimum” (predefined) number of points
- Now we can build a binary tree:

$x_2 > 1.0$



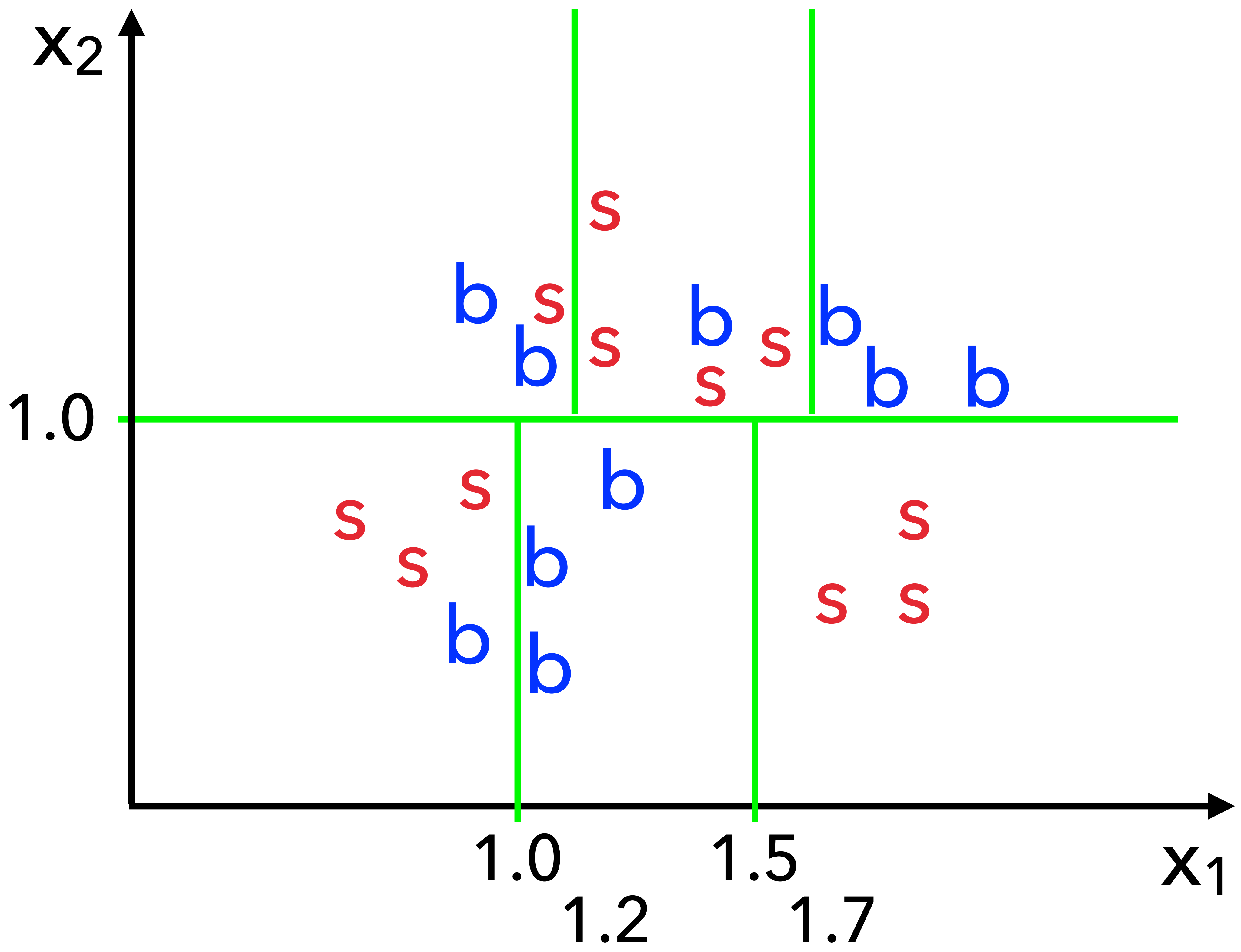
DECISION TREES CLASSIFICATION

- Finish once every region contains a “minimum” (predefined) number of points
- Now we can build a binary tree:



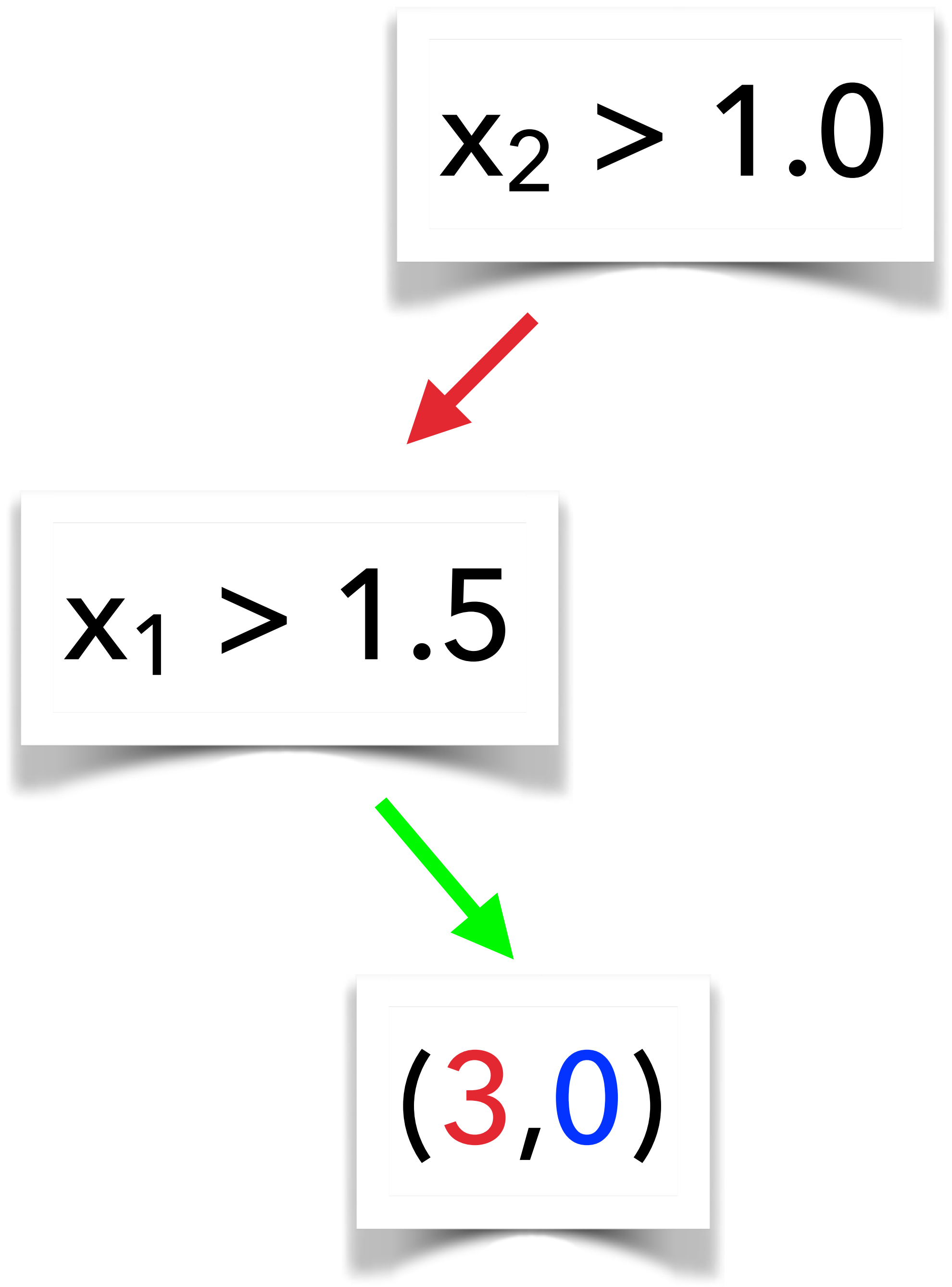
$x_2 > 1.0$

$x_1 > 1.5$



DECISION TREES CLASSIFICATION

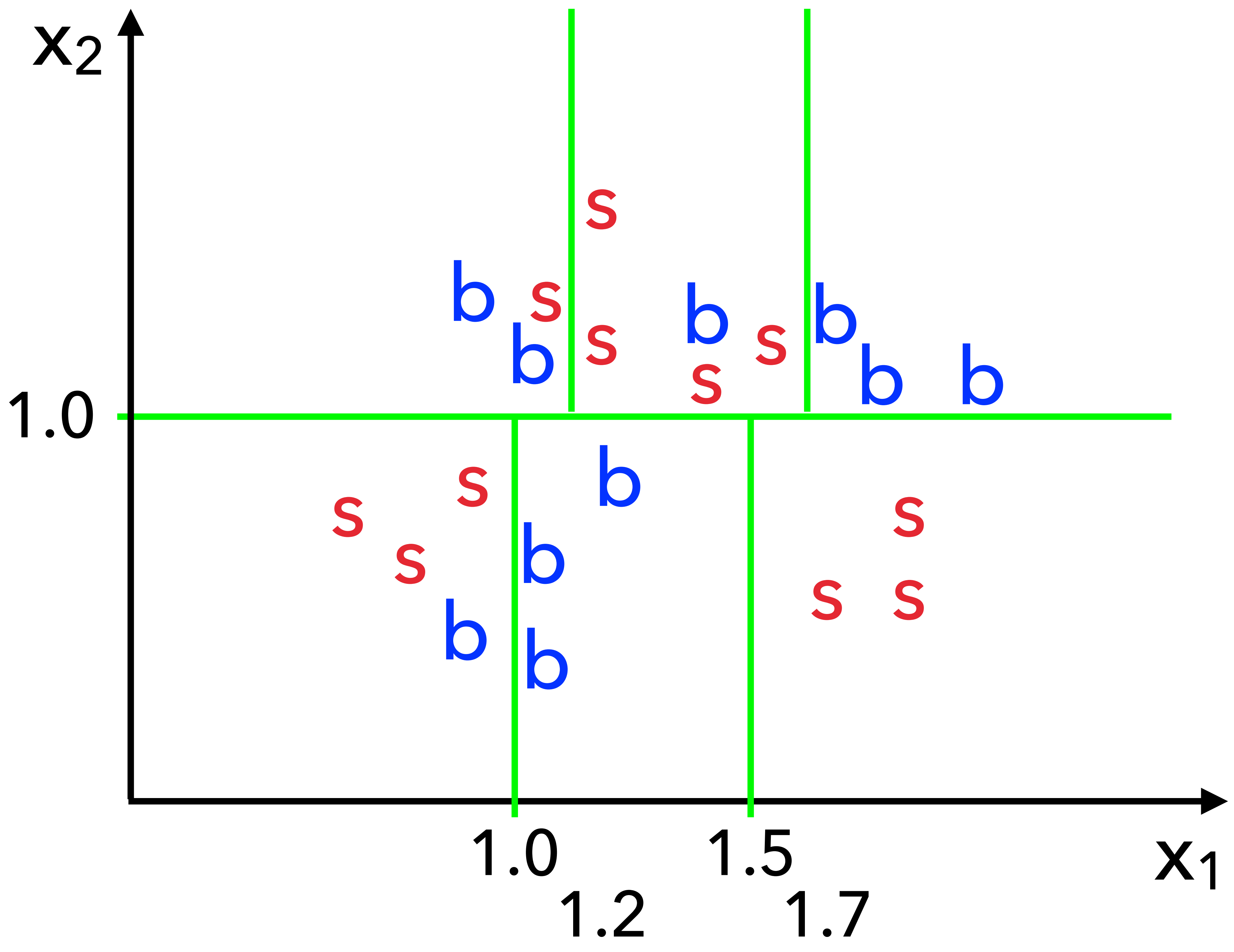
- Finish once every region contains a “minimum” (predefined) number of points
- Now we can build a binary tree:



$x_2 > 1.0$

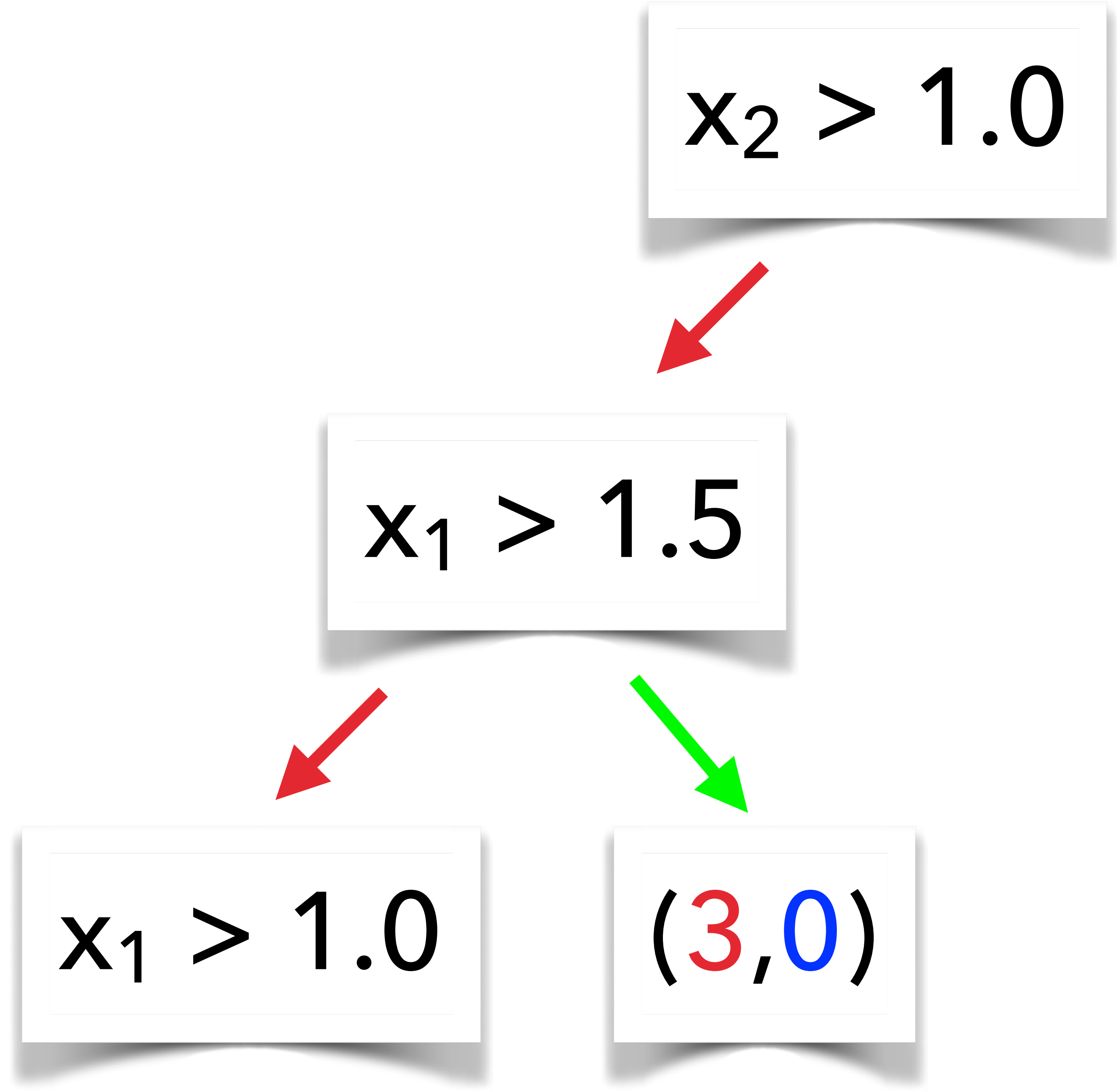
$x_1 > 1.5$

(3,0)



DECISION TREES CLASSIFICATION

- Finish once every region contains a “minimum” (predefined) number of points
- Now we can build a binary tree:

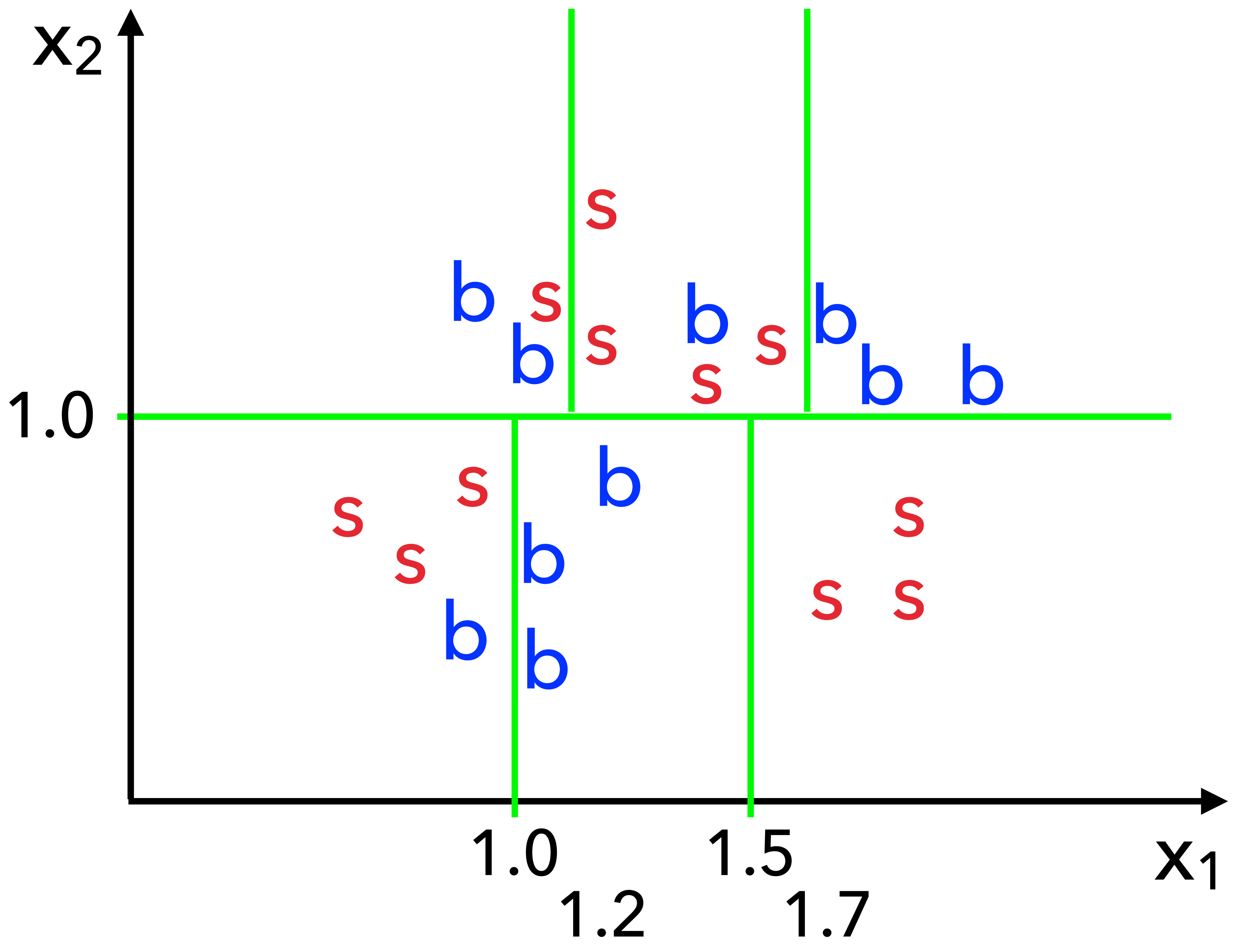


$x_2 > 1.0$

$x_1 > 1.5$

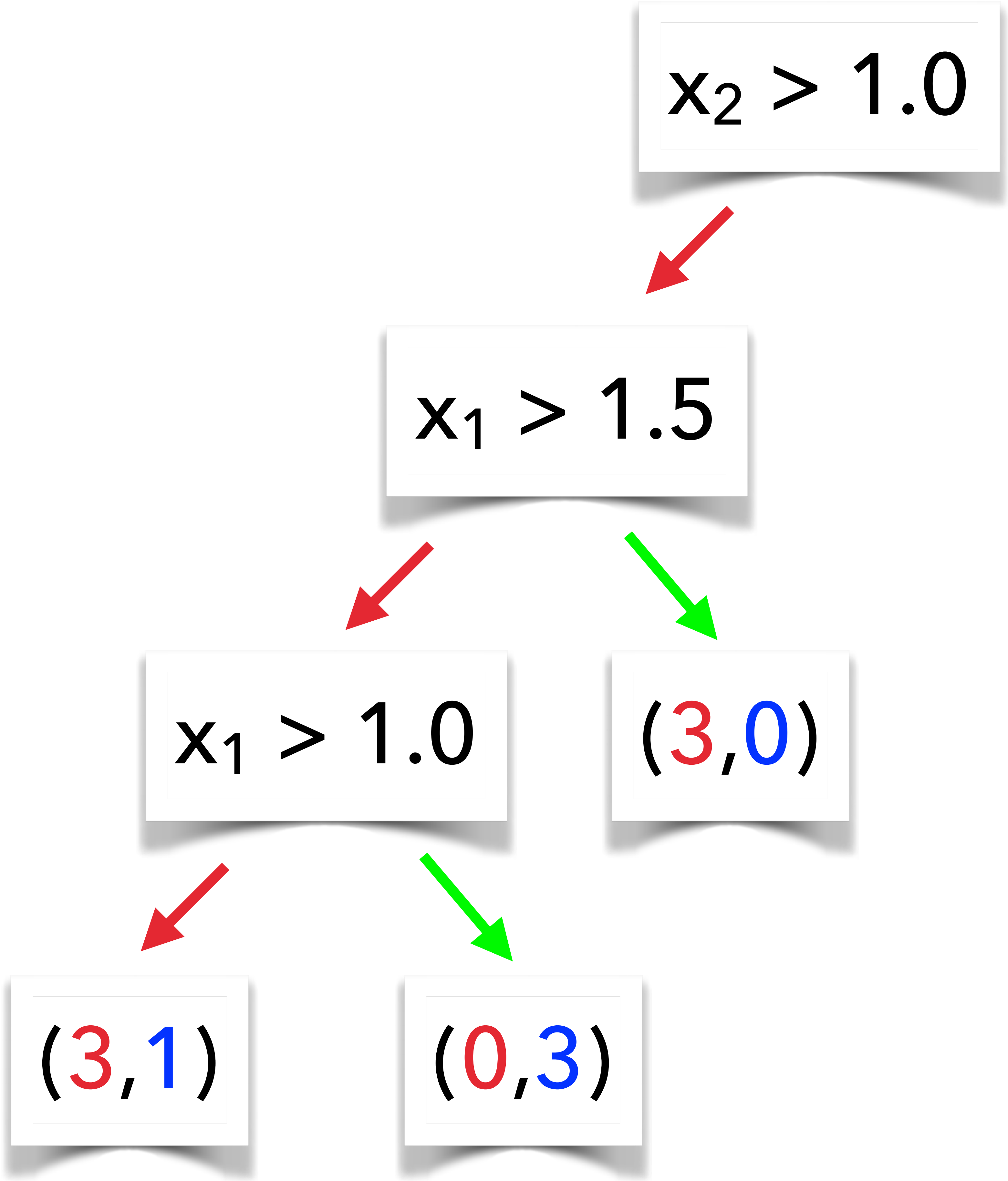
$x_1 > 1.0$

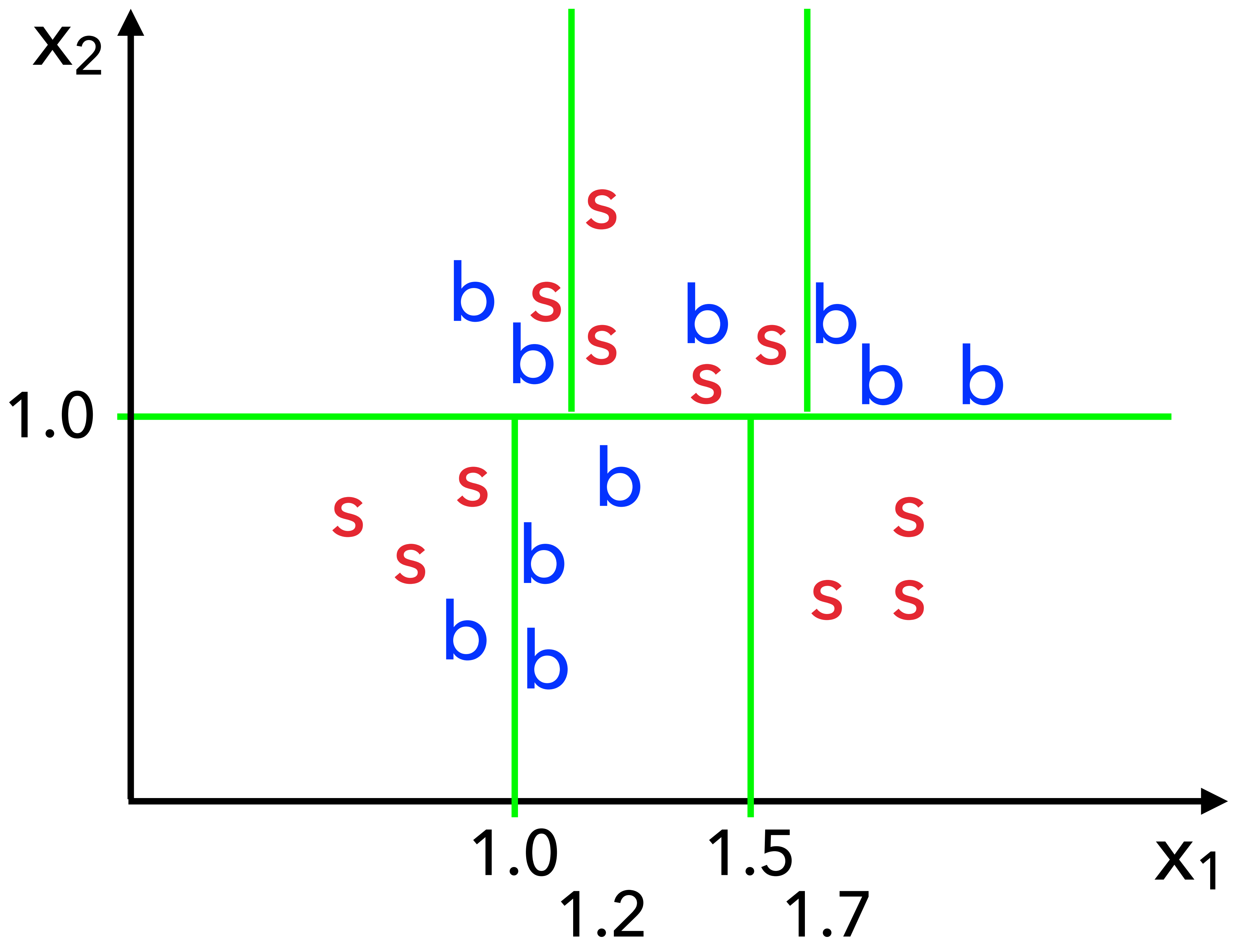
(3,0)



DECISION TREES CLASSIFICATION

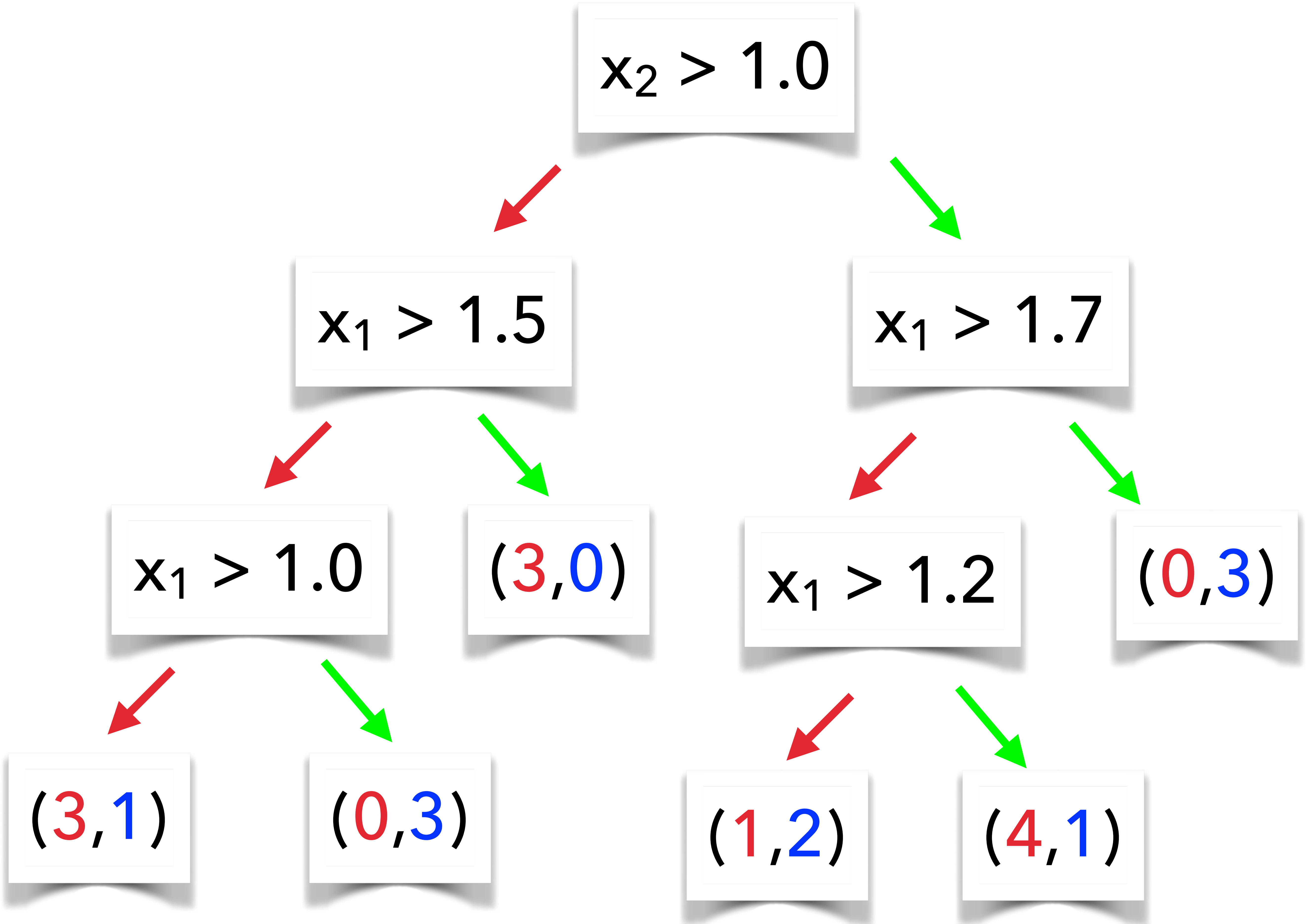
- Finish once every region contains a "minimum" (predefined) number of points
- Now we can build a binary tree:





DECISION TREES CLASSIFICATION

- Classify the tree leaves using the **majority vote**



$x_2 > 1.0$

$x_1 > 1.5$

$x_1 > 1.7$

$x_1 > 1.0$

(3,0)

$x_1 > 1.2$

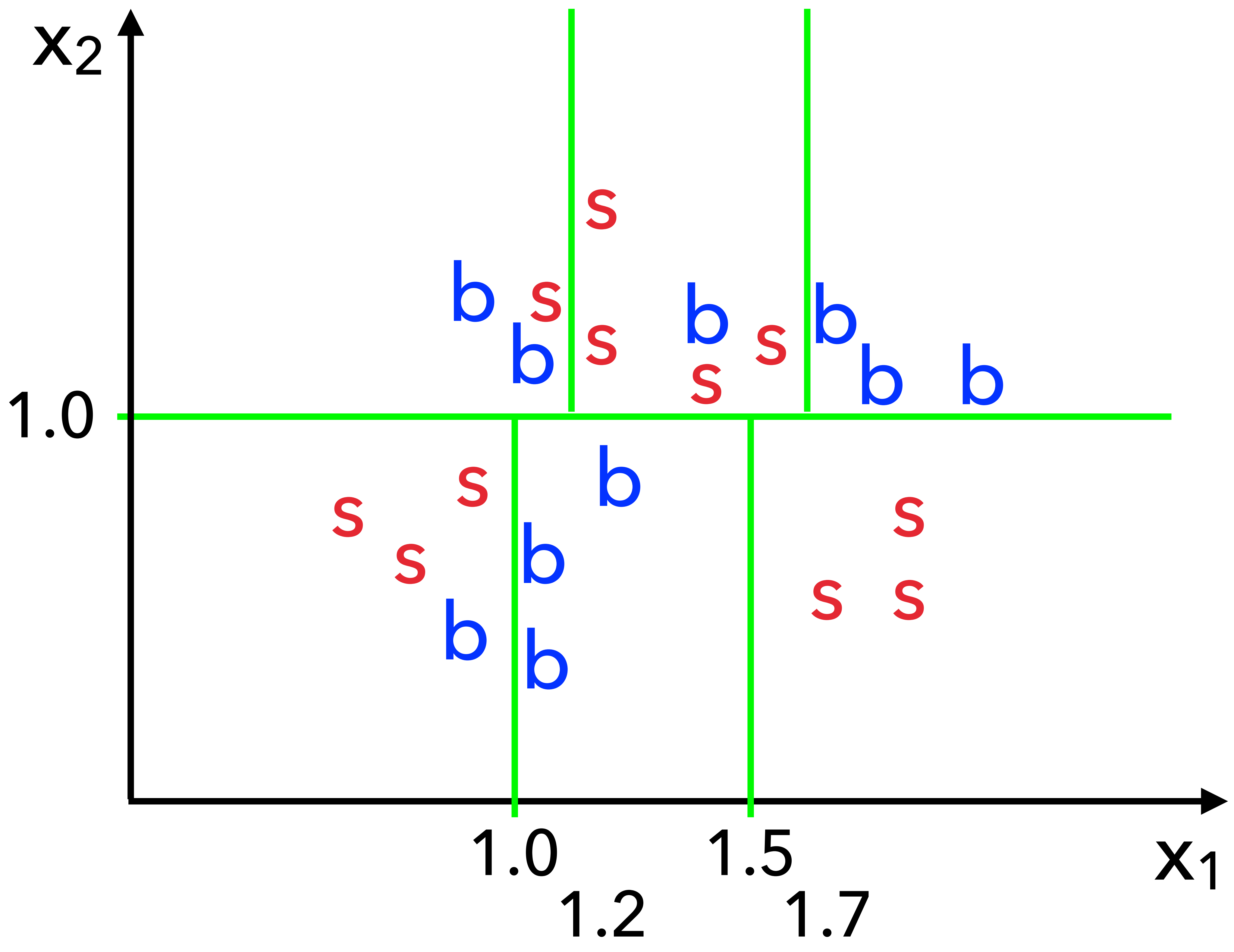
(0,3)

(3,1)

(0,3)

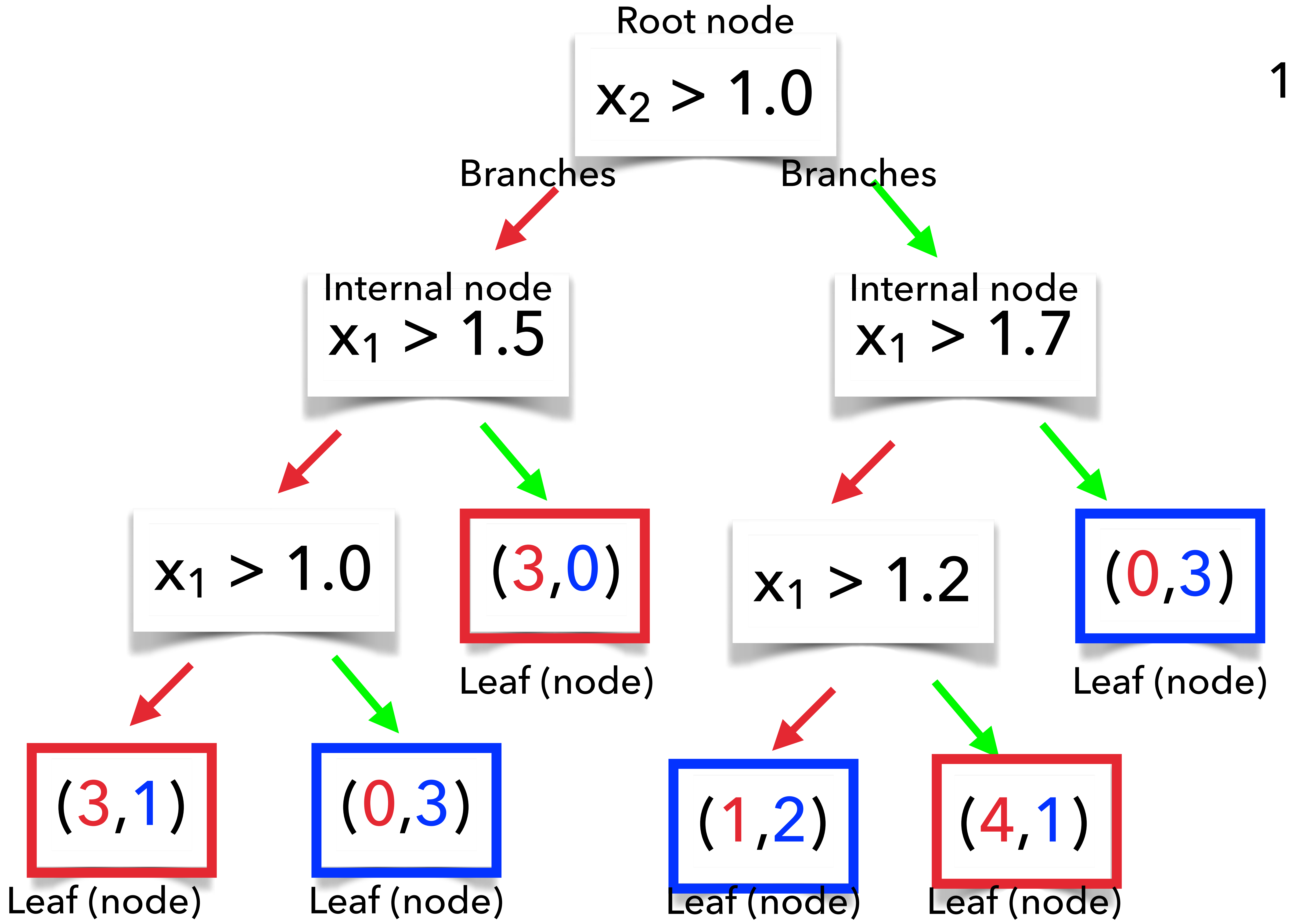
(1,2)

(4,1)



DECISION TREES CLASSIFICATION

- Classify the tree leaves using the **majority vote**



Root node

$x_2 > 1.0$

Branches

Branches

Internal node

$x_1 > 1.5$

Internal node

$x_1 > 1.7$

$x_1 > 1.0$

(3,0)

Leaf (node)

$x_1 > 1.2$

(0,3)

Leaf (node)

(3,1)

Leaf (node)

(0,3)

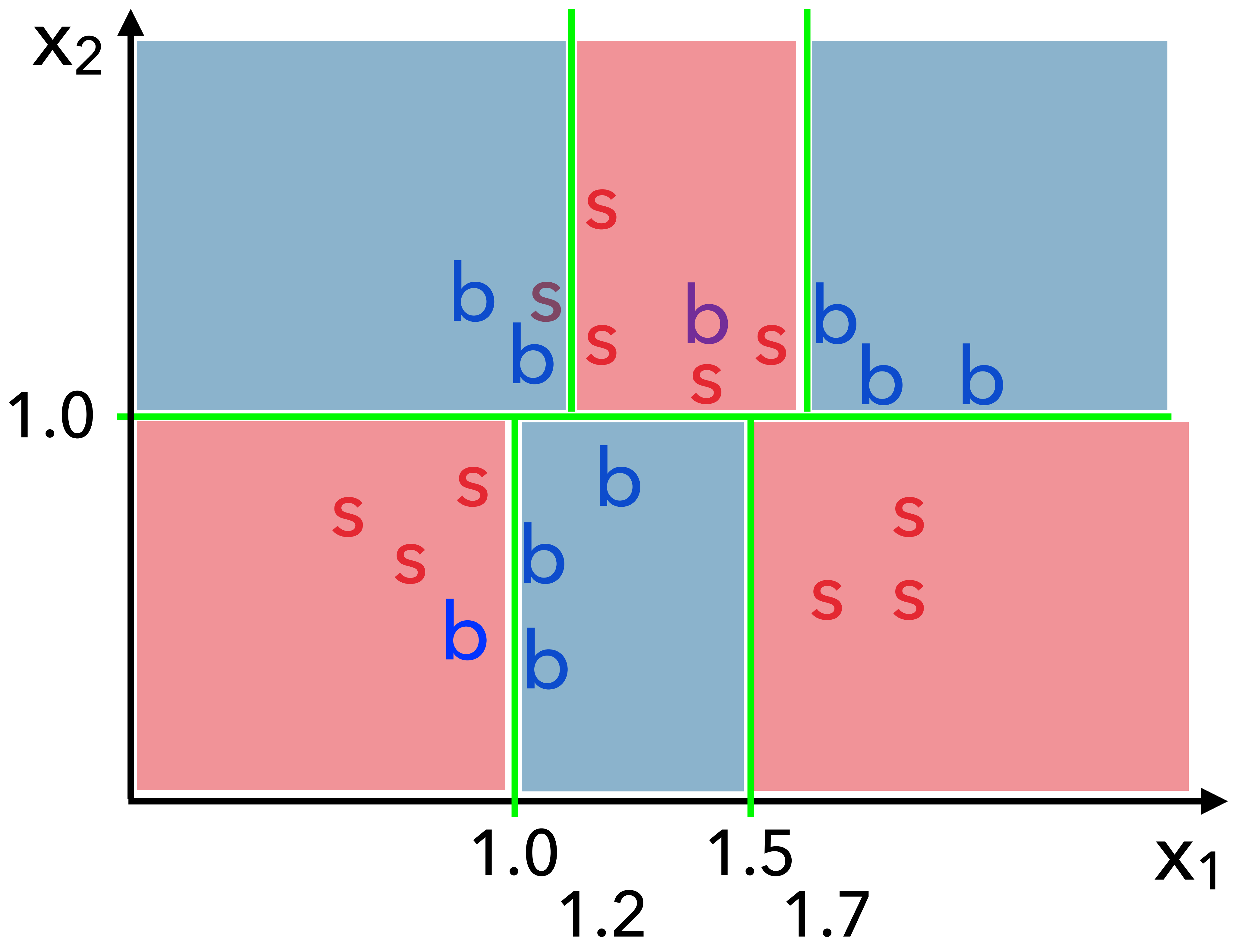
Leaf (node)

(1,2)

Leaf (node)

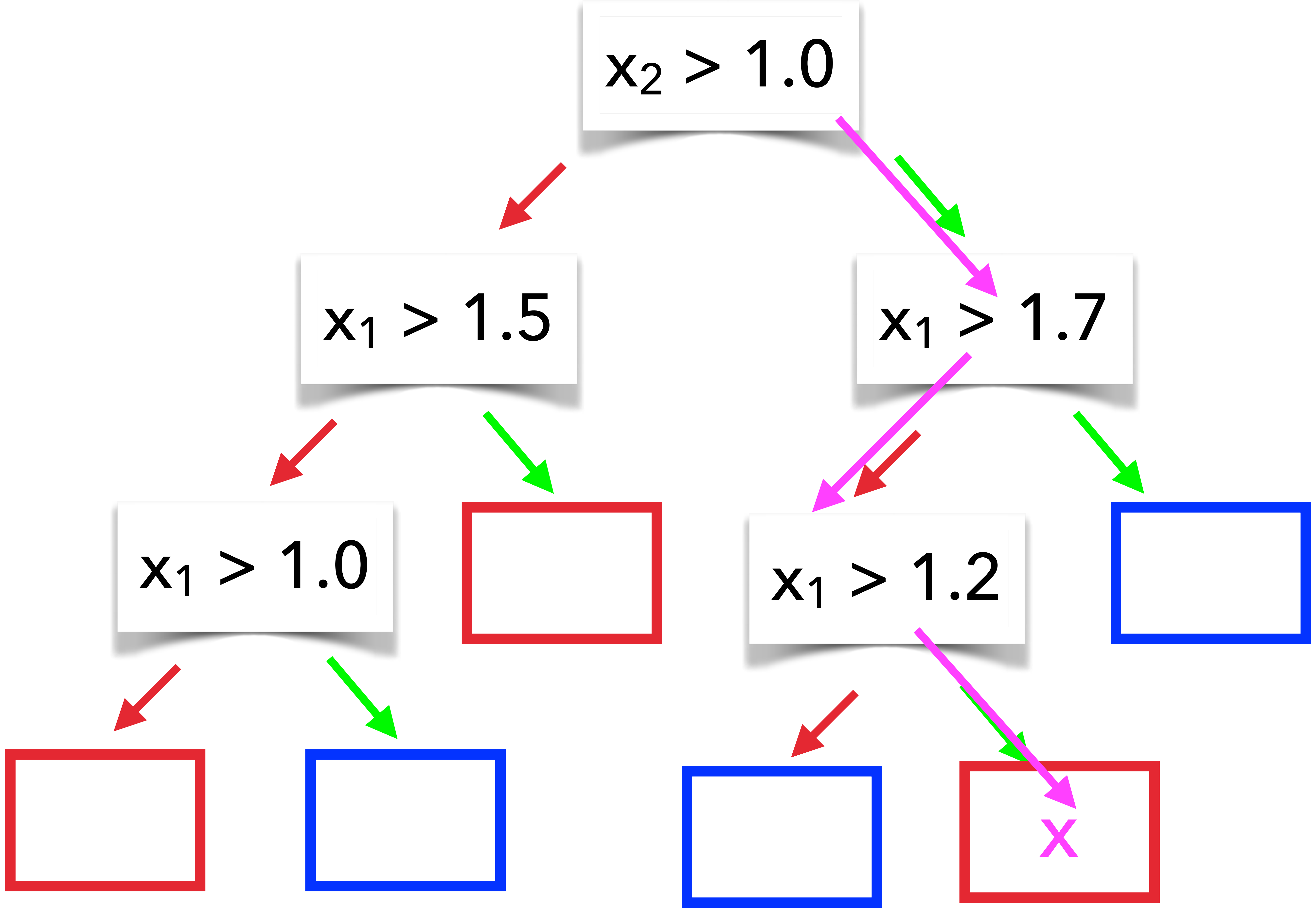
(4,1)

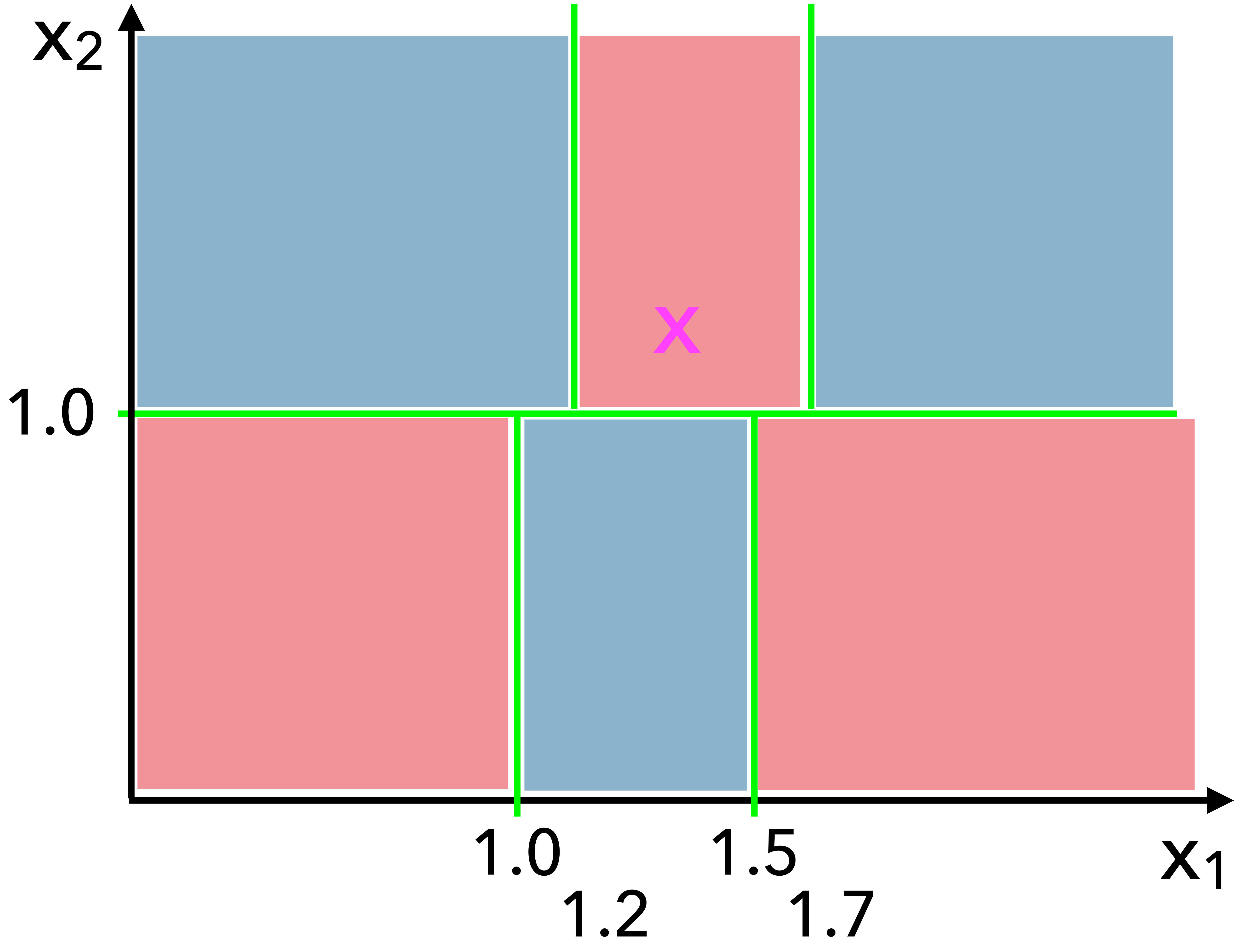
Leaf (node)



DECISION TREES CLASSIFICATION

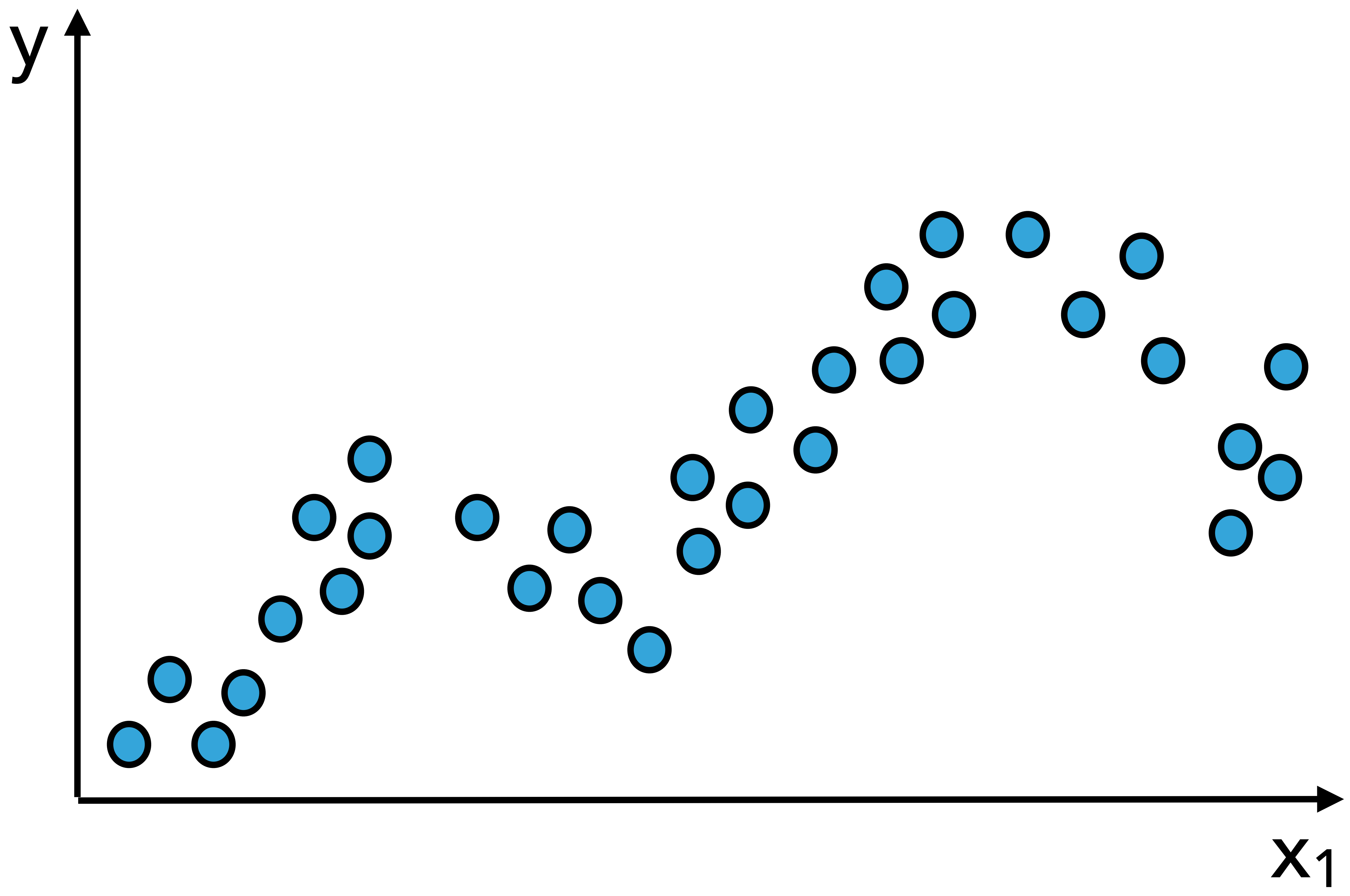
- All that is left is to apply your trained DT to real data!





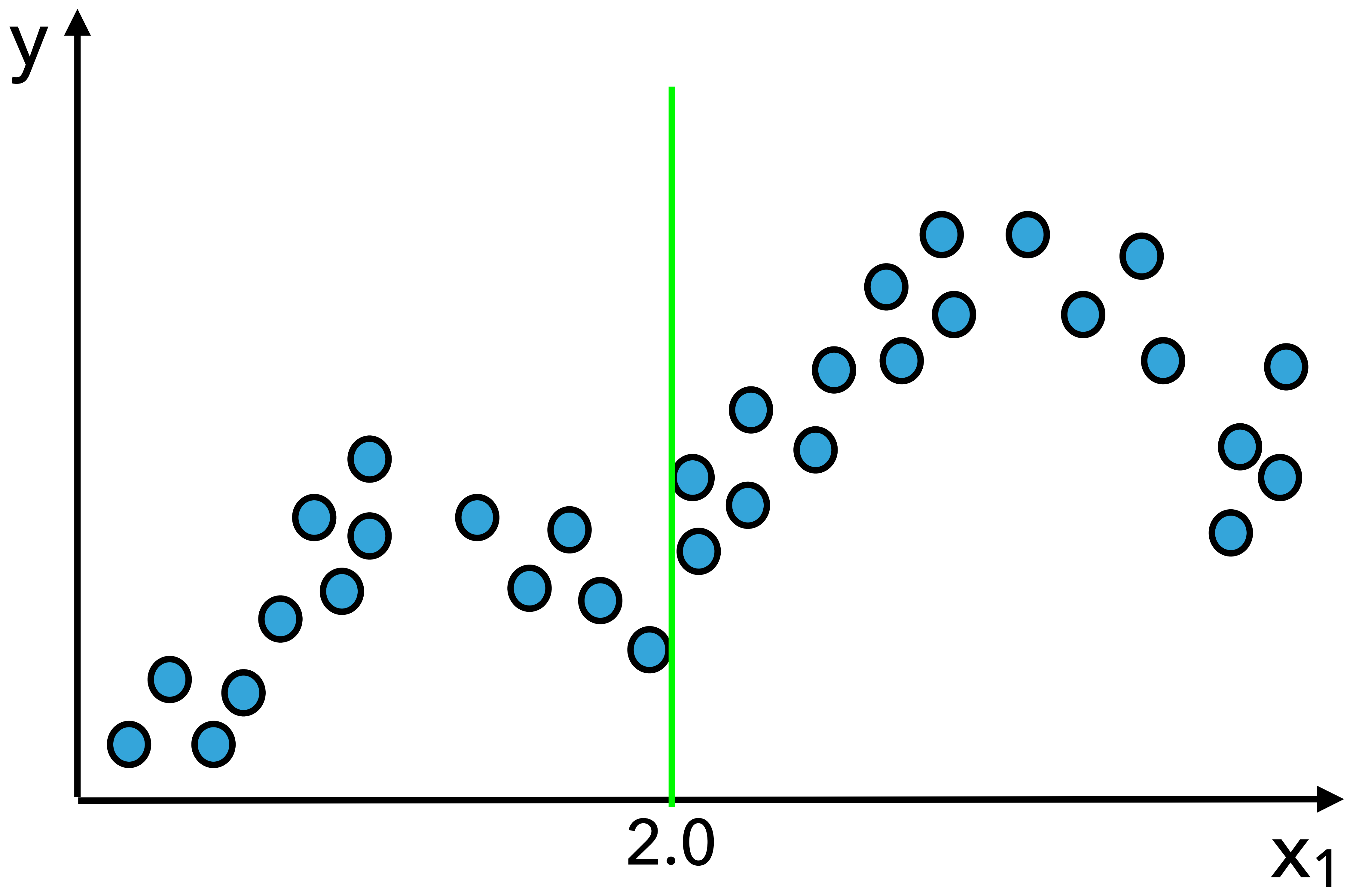
DECISION TREES REGRESSION

- We are given a 2D training dataset (x,y) and our goal is to predict y value for each new data point x
- This time we need to predict a **continuous** value (instead of binary)



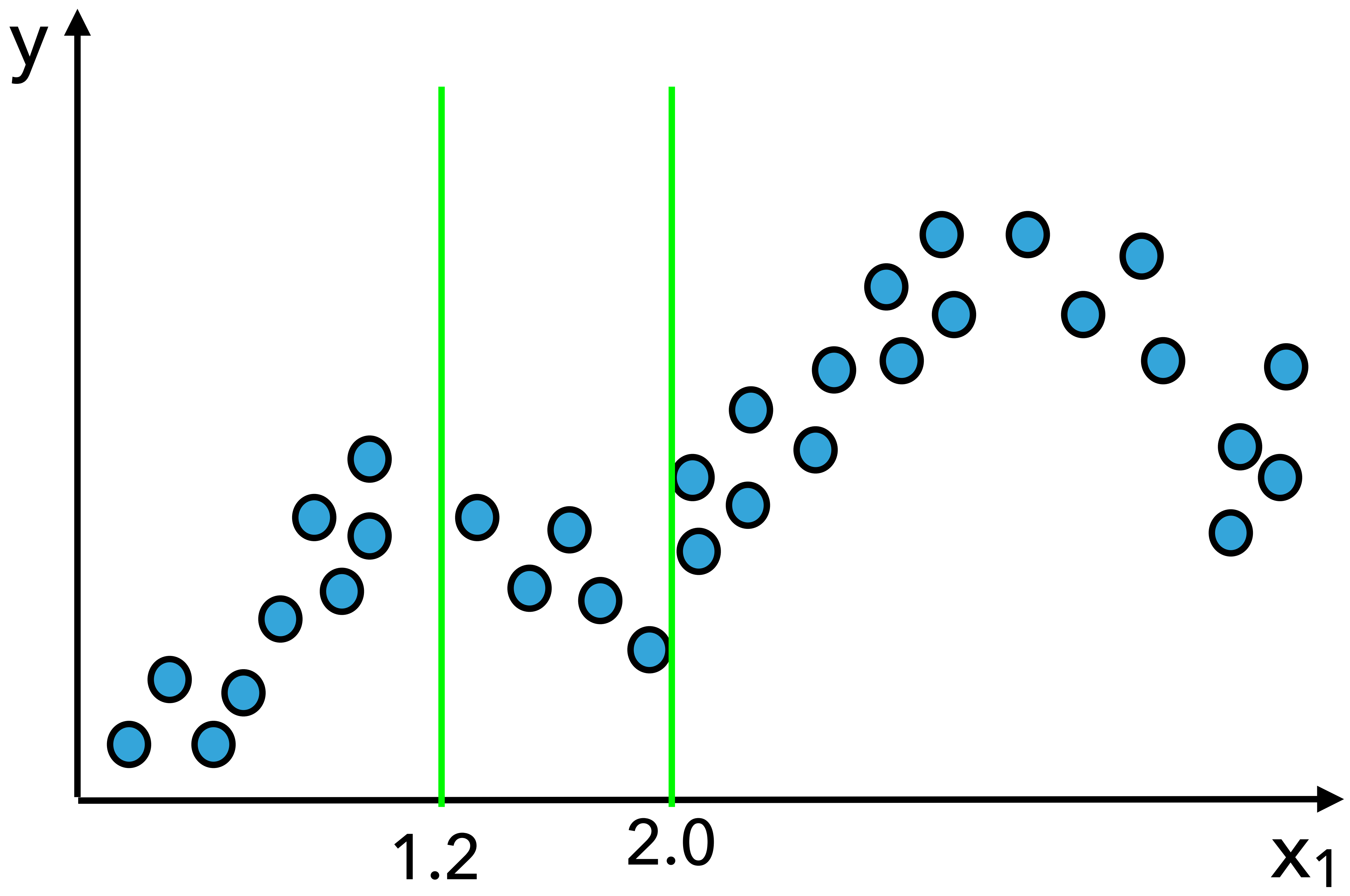
DECISION TREES REGRESSION

- Amazingly, the same logic works!



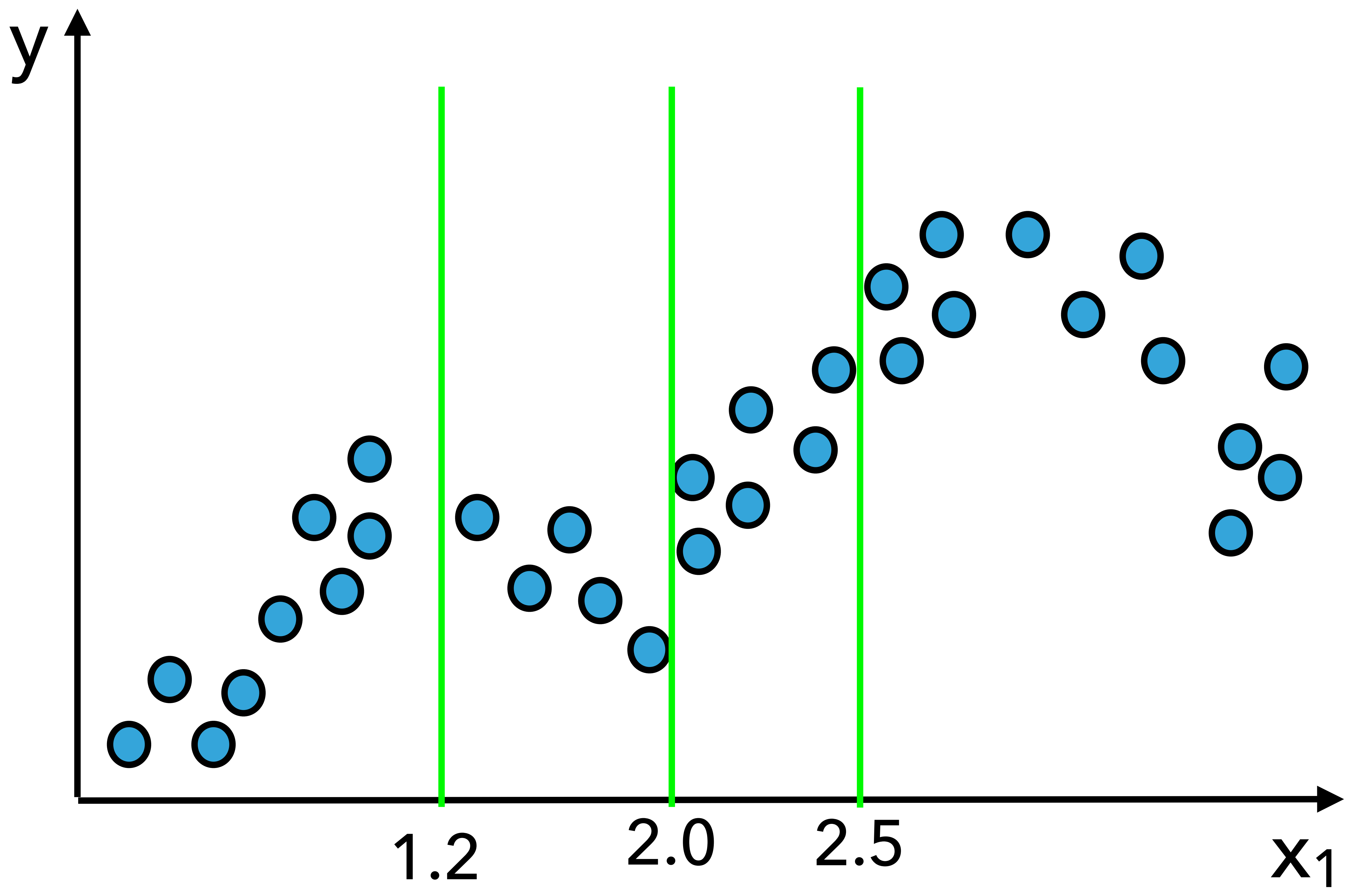
DECISION TREES REGRESSION

- Continue by dividing further new nods with additional simple (binary) cuts that minimise the error



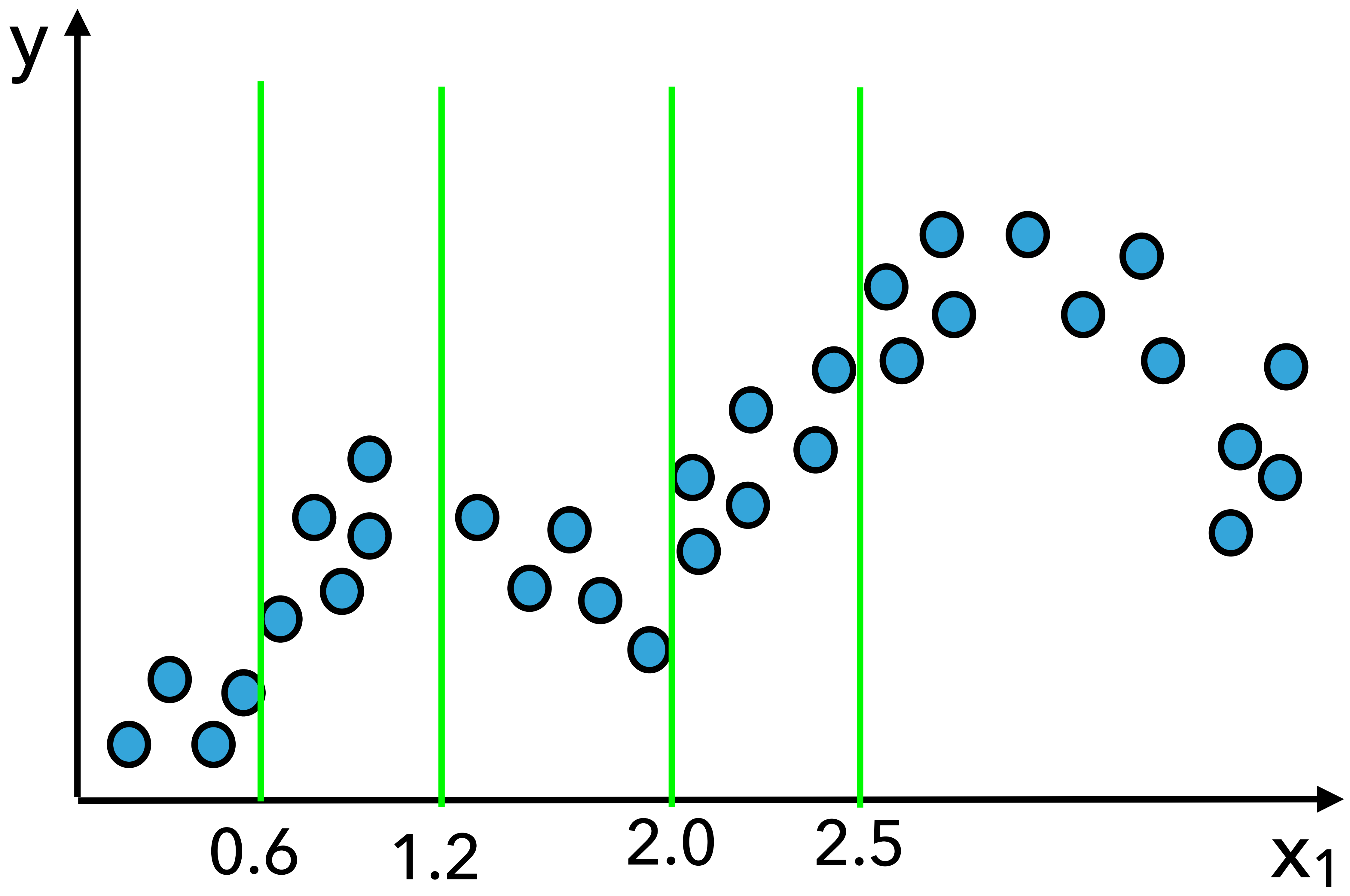
DECISION TREES REGRESSION

- Continue by dividing further new nods with additional simple (binary) cuts that minimise the error



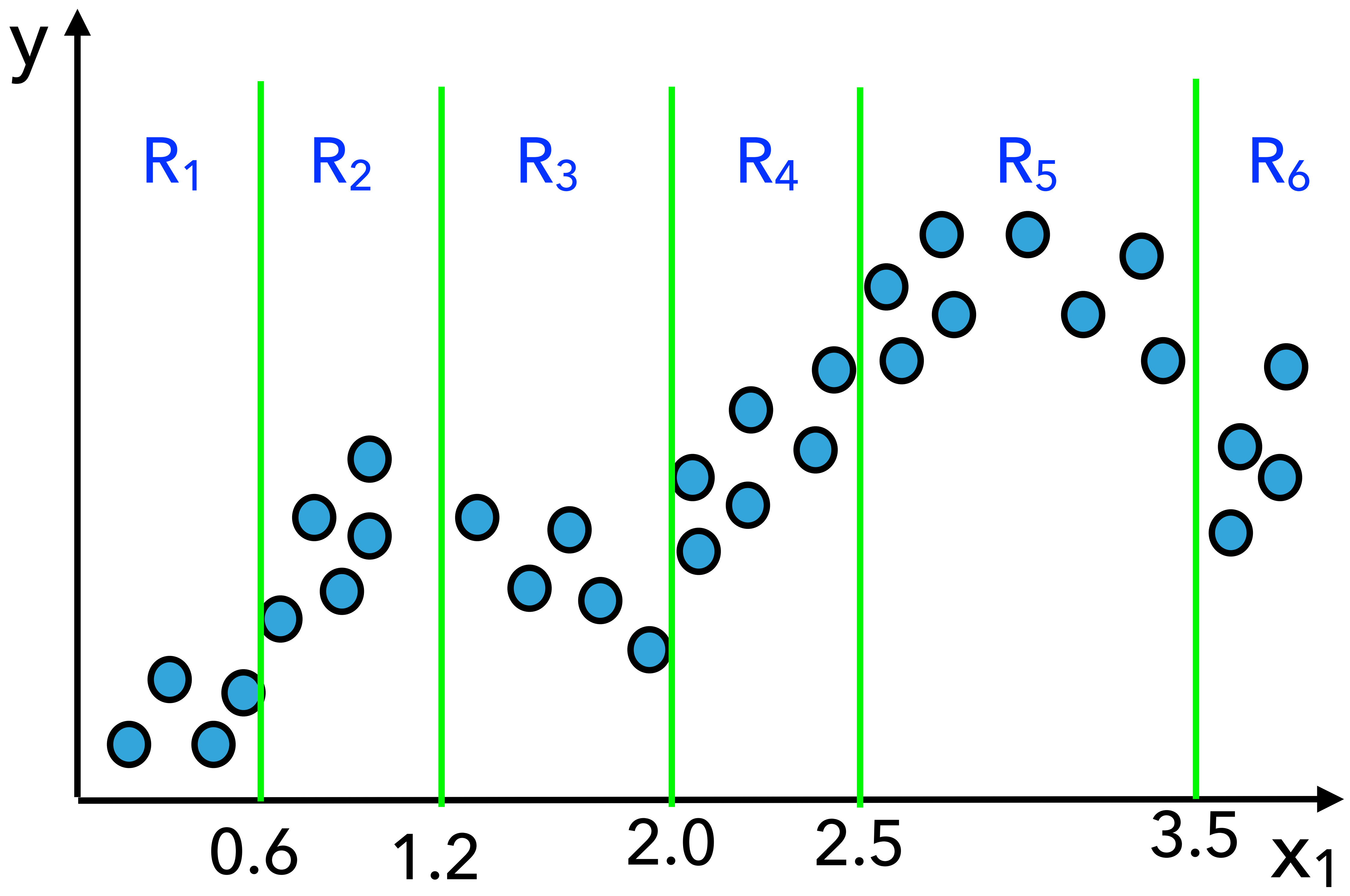
DECISION TREES REGRESSION

- Continue by dividing further new nods with additional simple (binary) cuts that minimise the error



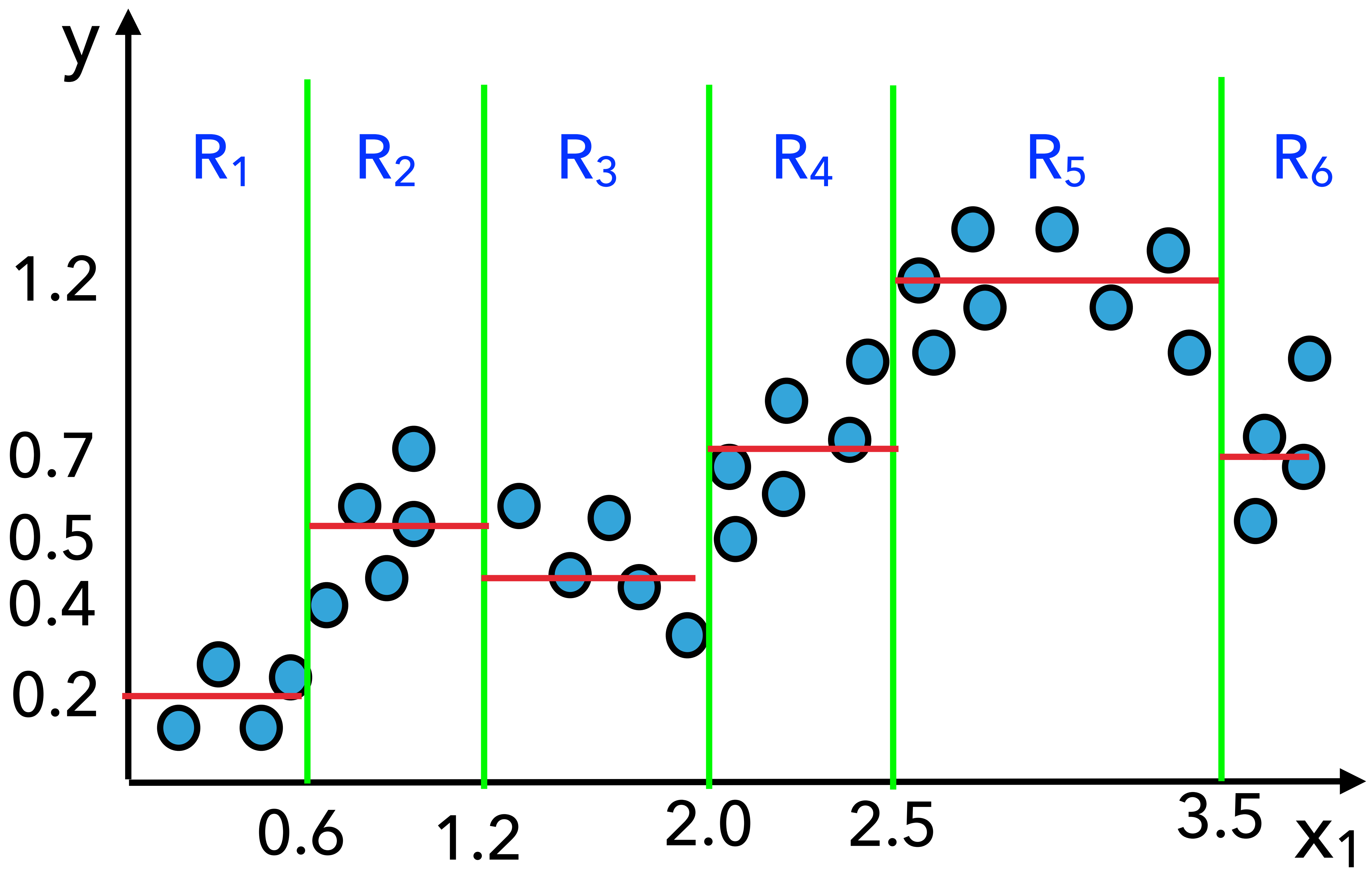
DECISION TREES REGRESSION

- Repeat until every region (leaf) contains a “minimum” number of points that is predefined



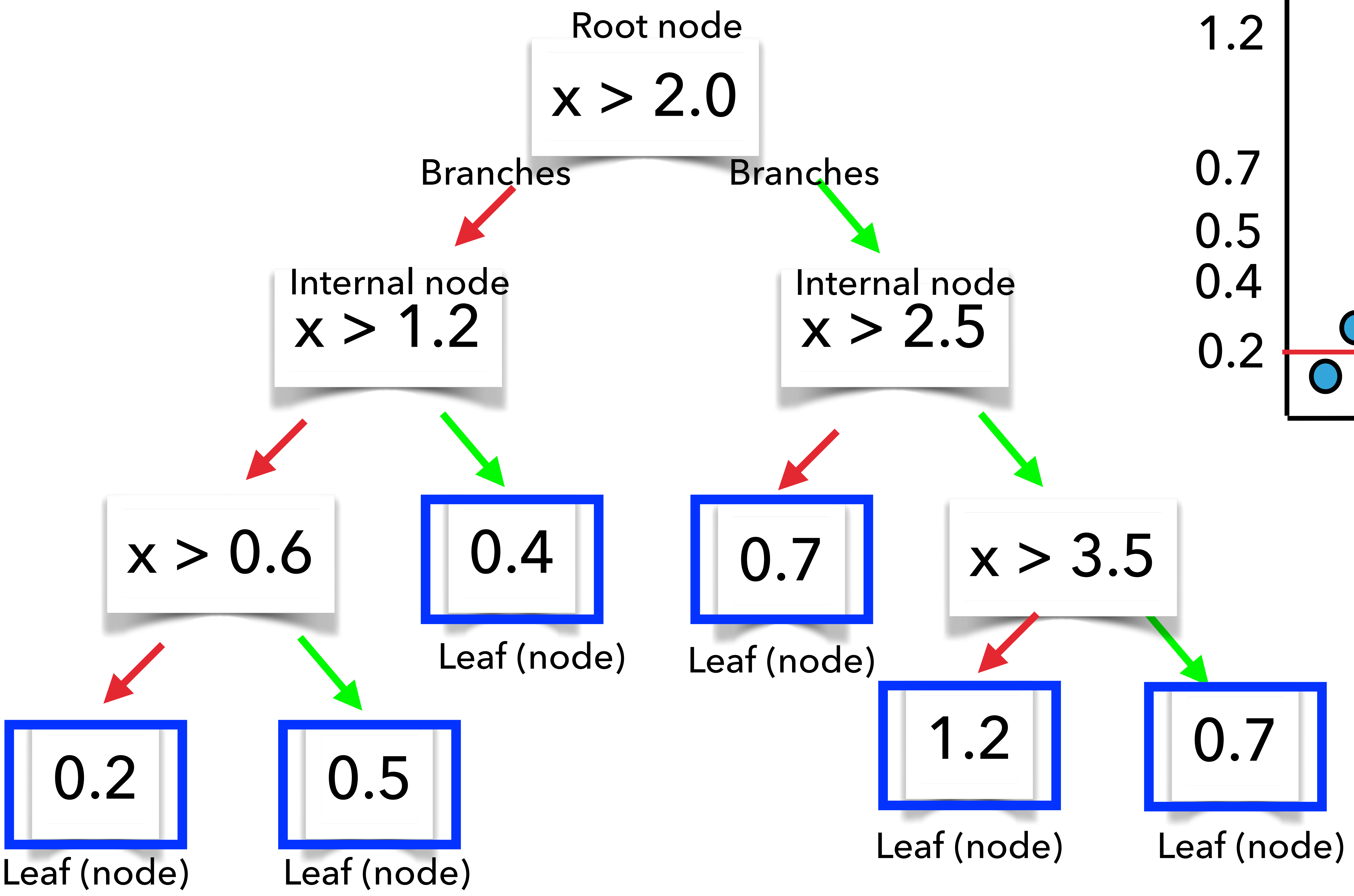
DECISION TREES REGRESSION

- Average of the points in each region gives you the prediction $\hat{y}(x)$



DECISION TREES REGRESSION

- Build a binary tree!



Root node

$x > 2.0$

Branches

Branches

Internal node

$x > 1.2$

Internal node

$x > 2.5$

$x > 0.6$

0.4

Leaf (node)

0.7

Leaf (node)

$x > 3.5$

0.2

Leaf (node)

0.5

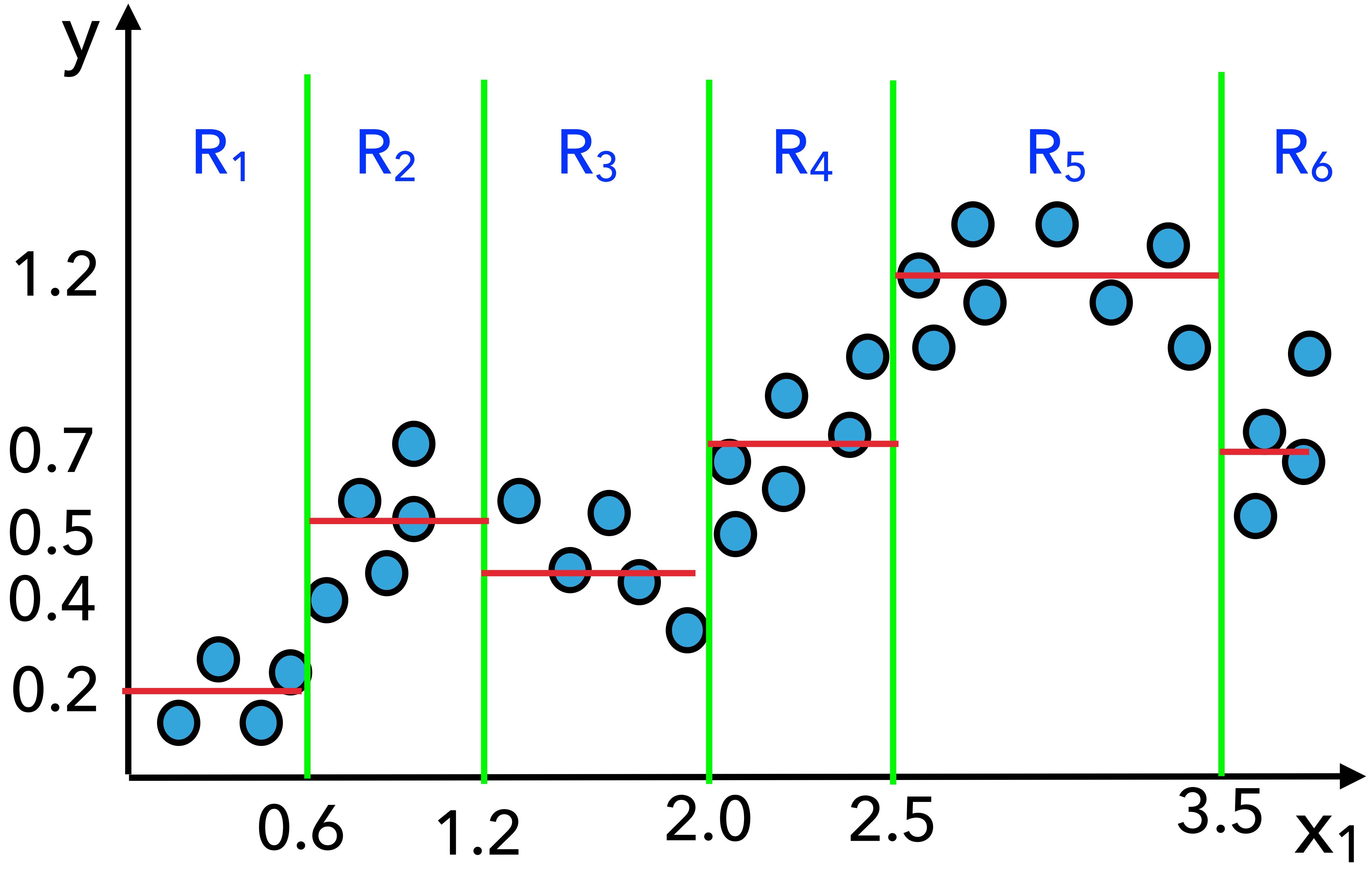
Leaf (node)

1.2

Leaf (node)

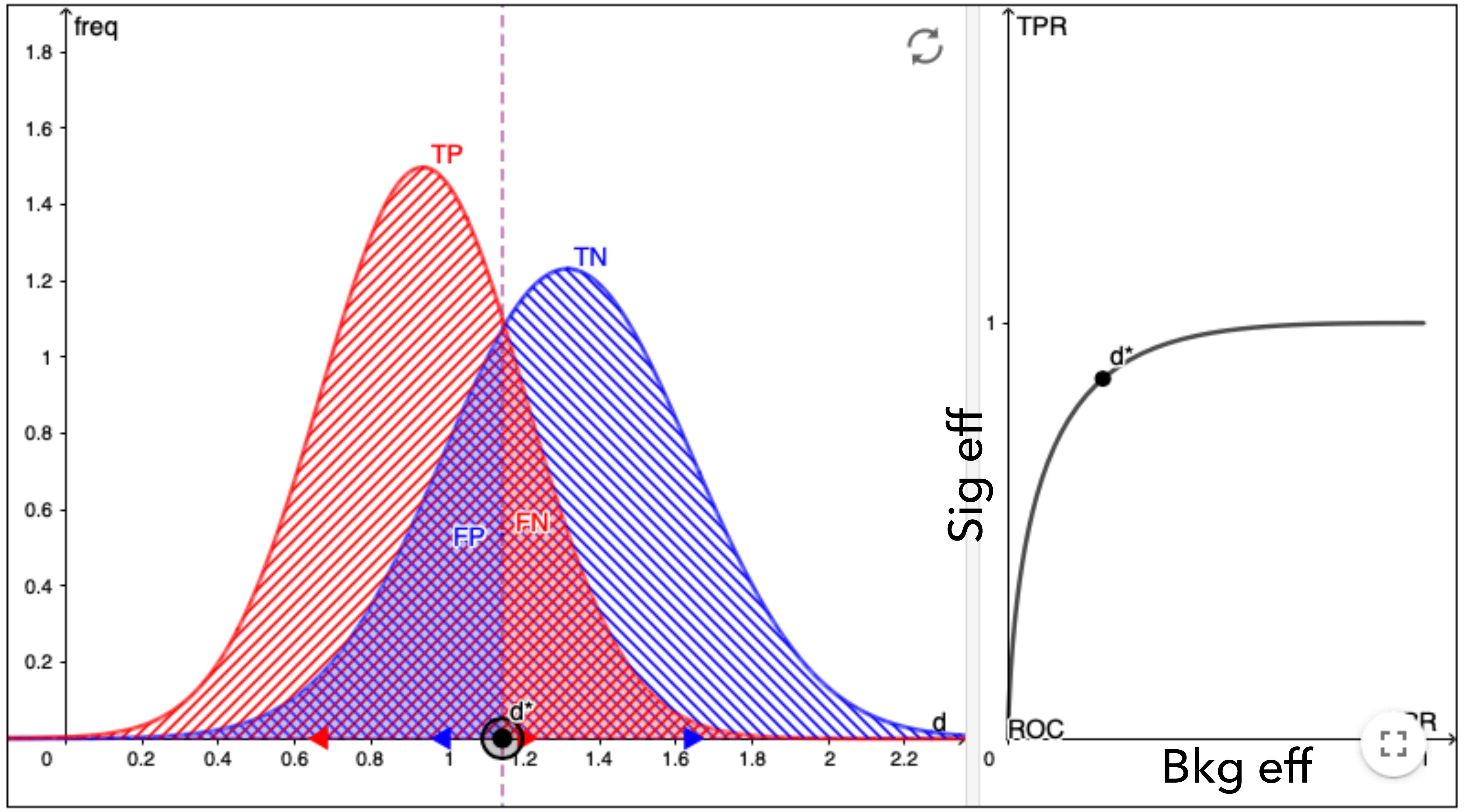
0.7

Leaf (node)



PERFORMANCE MONITORING

- Reminder: Goal is to use ML to approximate LR that is optimal test statistic for hypothesis testing (H_0 vs H_1)
- From outputs of ML draw a PDF of output test statistics for H_0 and H_1
- Draw a Receiver Operators Curve (ROC) and use it to determine cut value on your ML output (if it is continuous rather than binary)

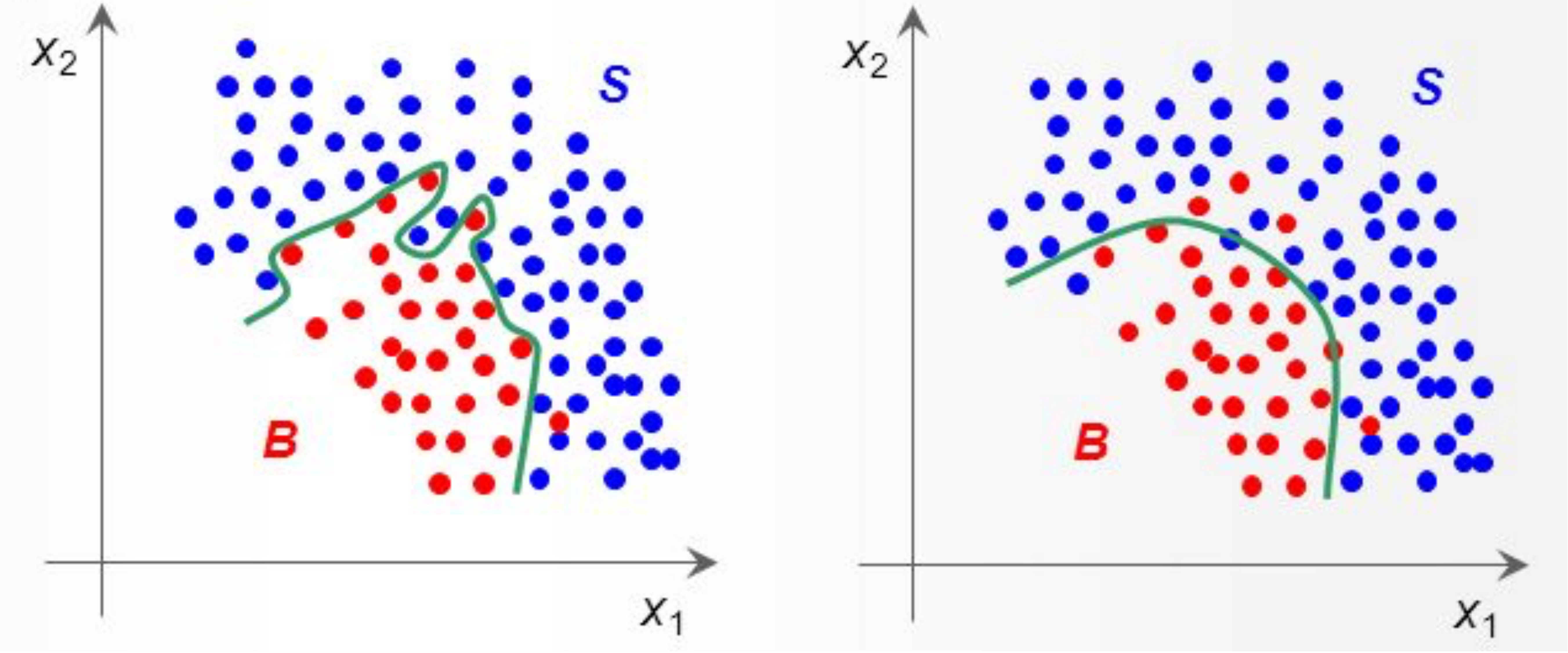


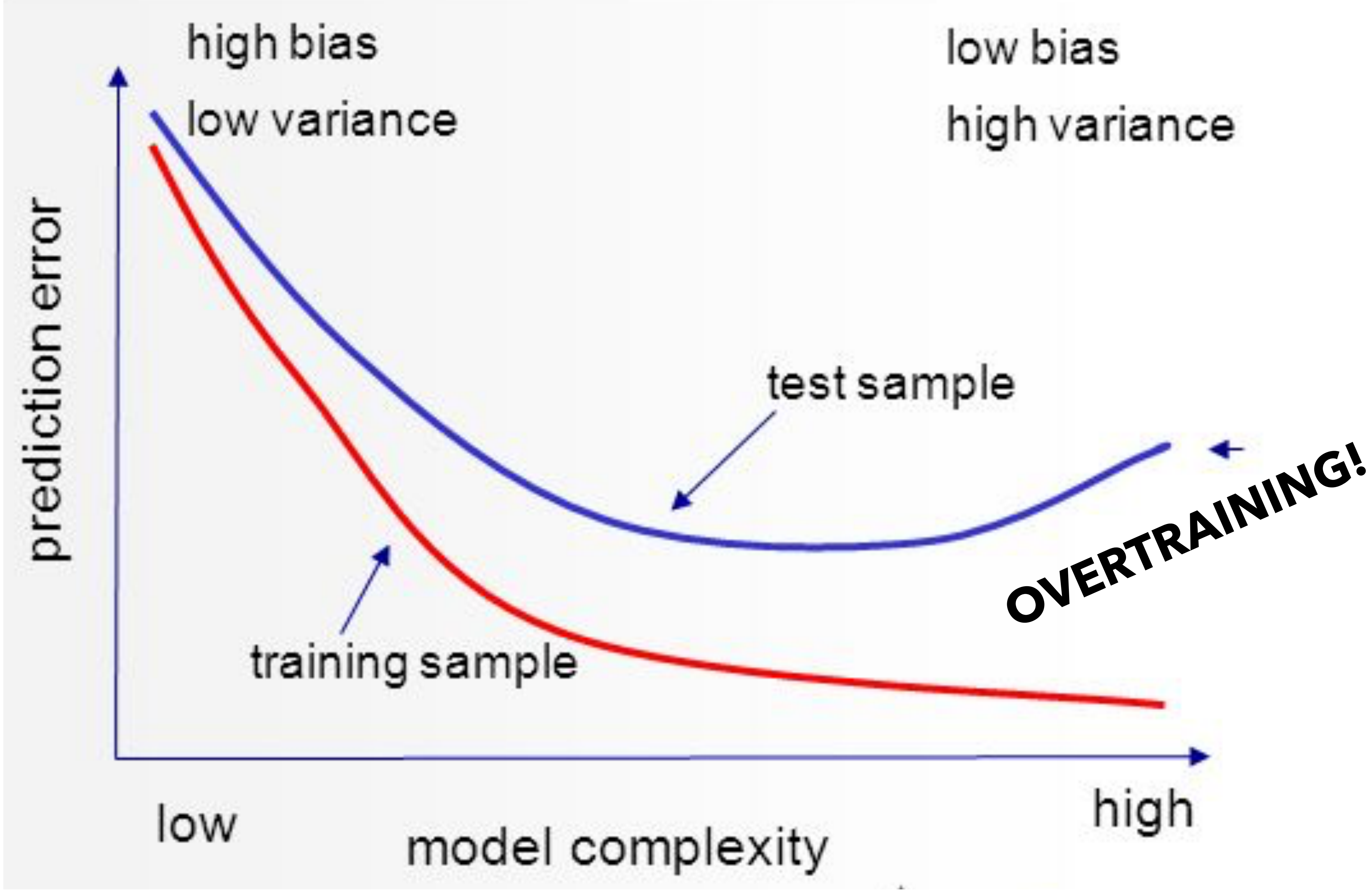
1.) You can set wanted signal efficiency and from it determines what is your background rejection

2.) You can set wanted background rejection and from it determine what is your achieved signal efficiency

OVERTRAINING

- Performance of your DT depends on training
- By increasing the complexity of your DT model (number of internal nodes) you can always improve performance
- Defining a random test sample important to monitor the possible overtraining
  - different from the training sample!!!



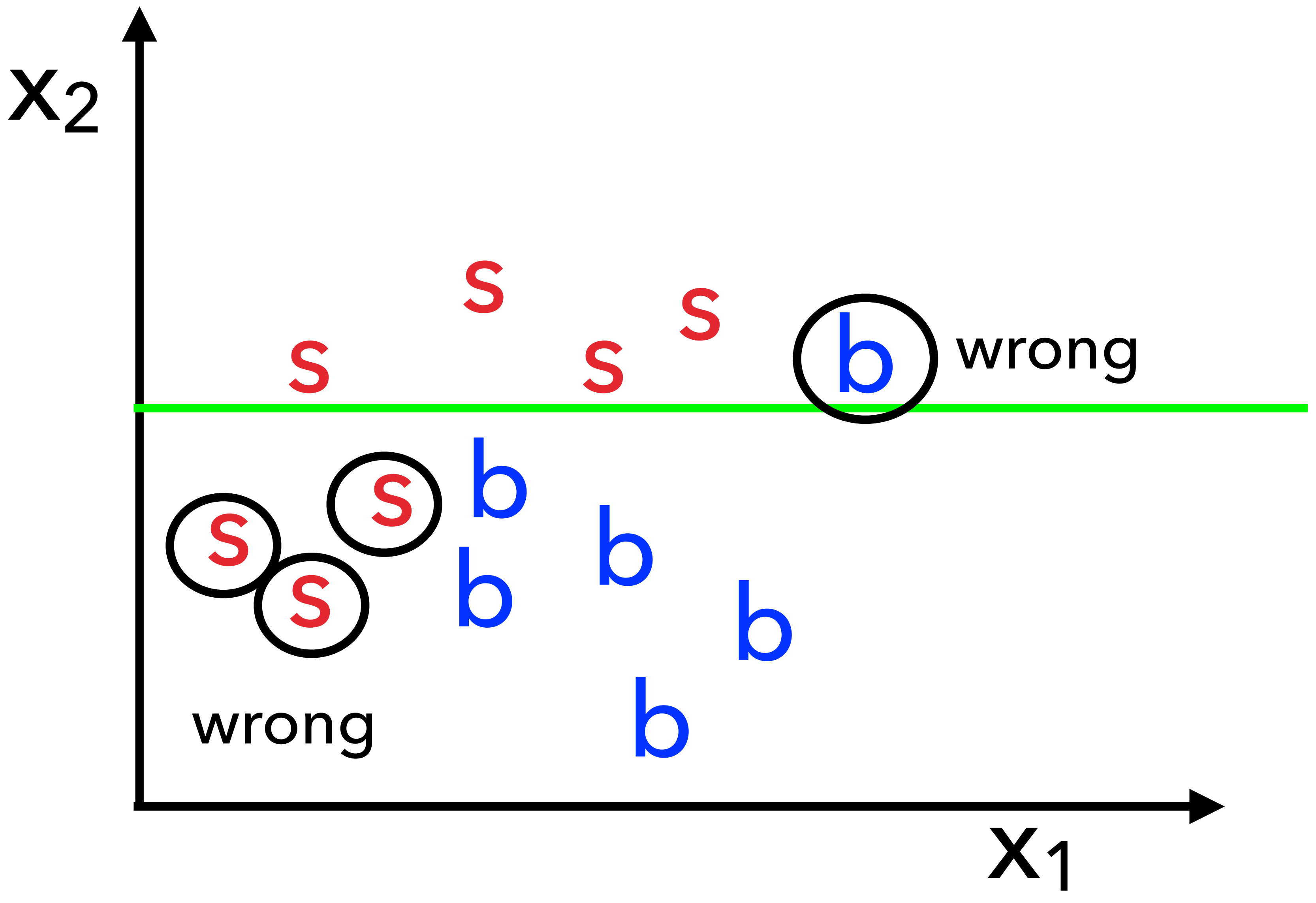


DECISION TREES DON'T WORK…

- .. beacause of three main reasons:
  1. The variables and the order of cuts are chosen on the base of separation of the given data. If you change the training sample you will most likely get a different tree.
  2. Whatever variable is the most discriminating will influence the rest of the tree.
  3. Extremely sensitive to overtraining. They learn the noise of the particular sample used for training and miss the general structure, thus having poor performance when applied on another sample.
- If we can fix these problems we can build a decision tree that works!
  1. **Pruning**: Re-grouping or removing branches to make the tree more stable.
  2. **Bagging**: Take N independent datasets and improve stability by averaging the resulting trees.
  3. **Boosting**: Sequentially train a model learning from the errors of the previous one.
- These techniques can be applied to classification and regression DT (and any other learning algorithm)

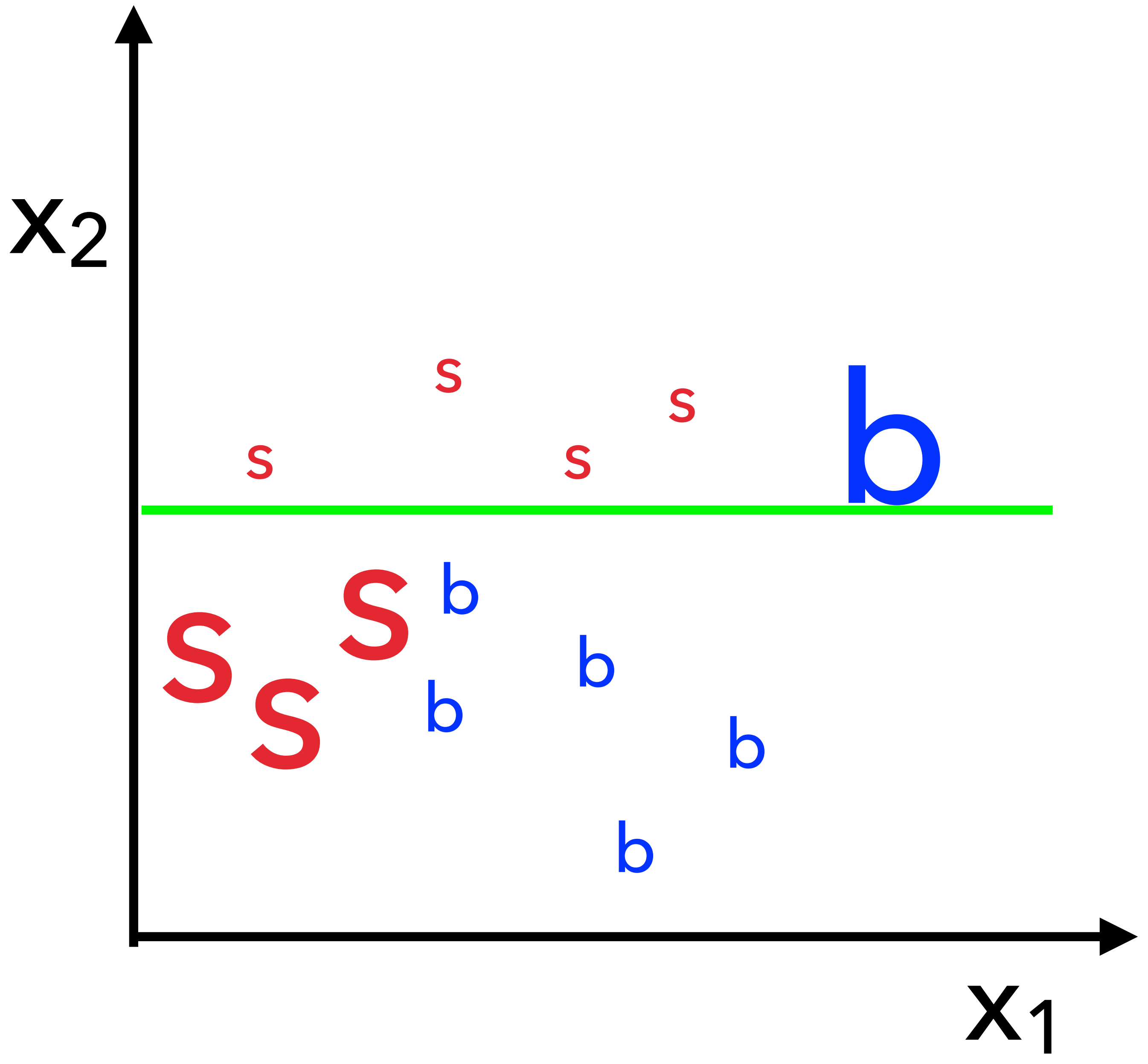
DECISION TREES BOOSTING

- After a first step focus on wrong ones



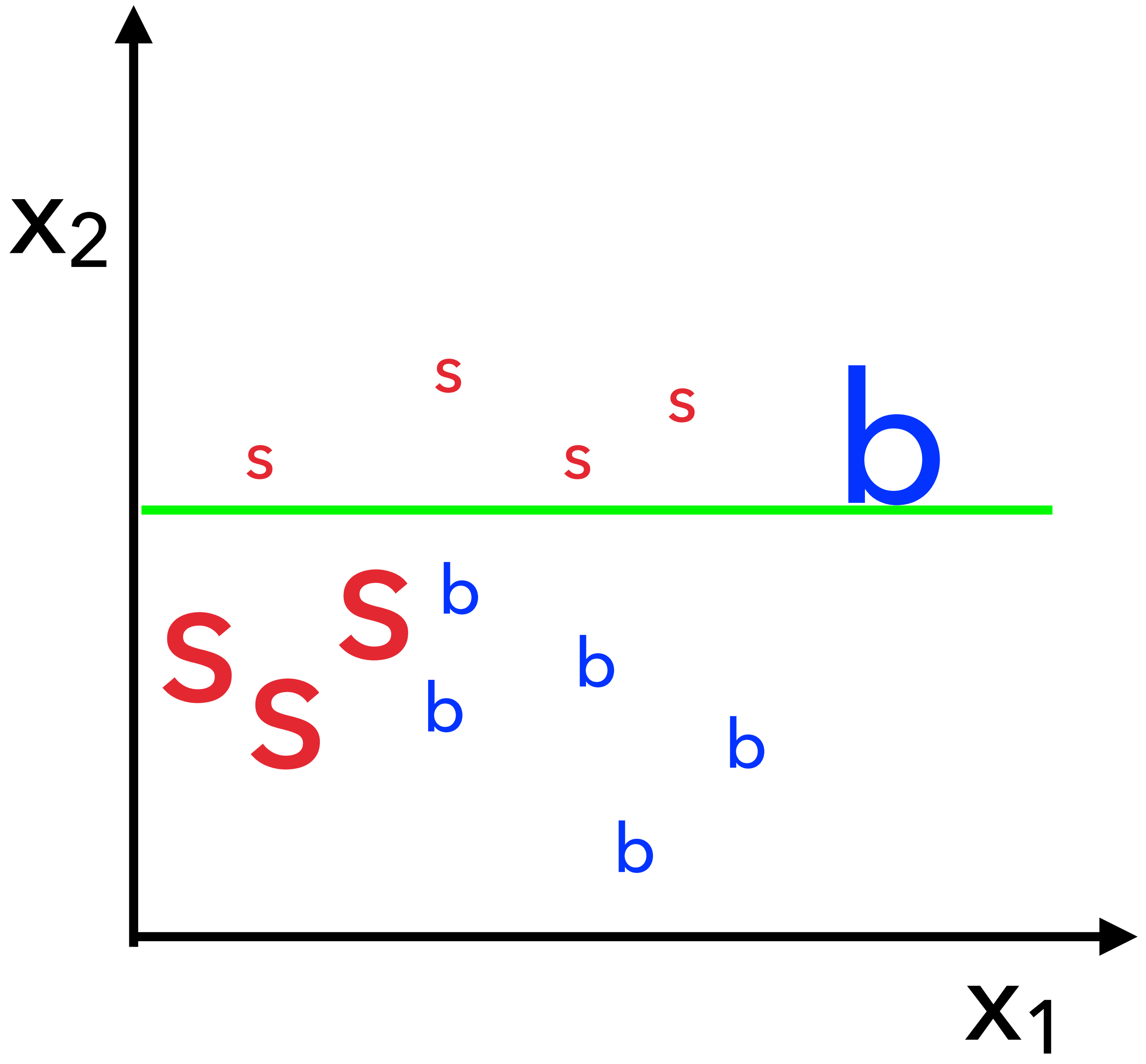
DECISION TREES BOOSTING

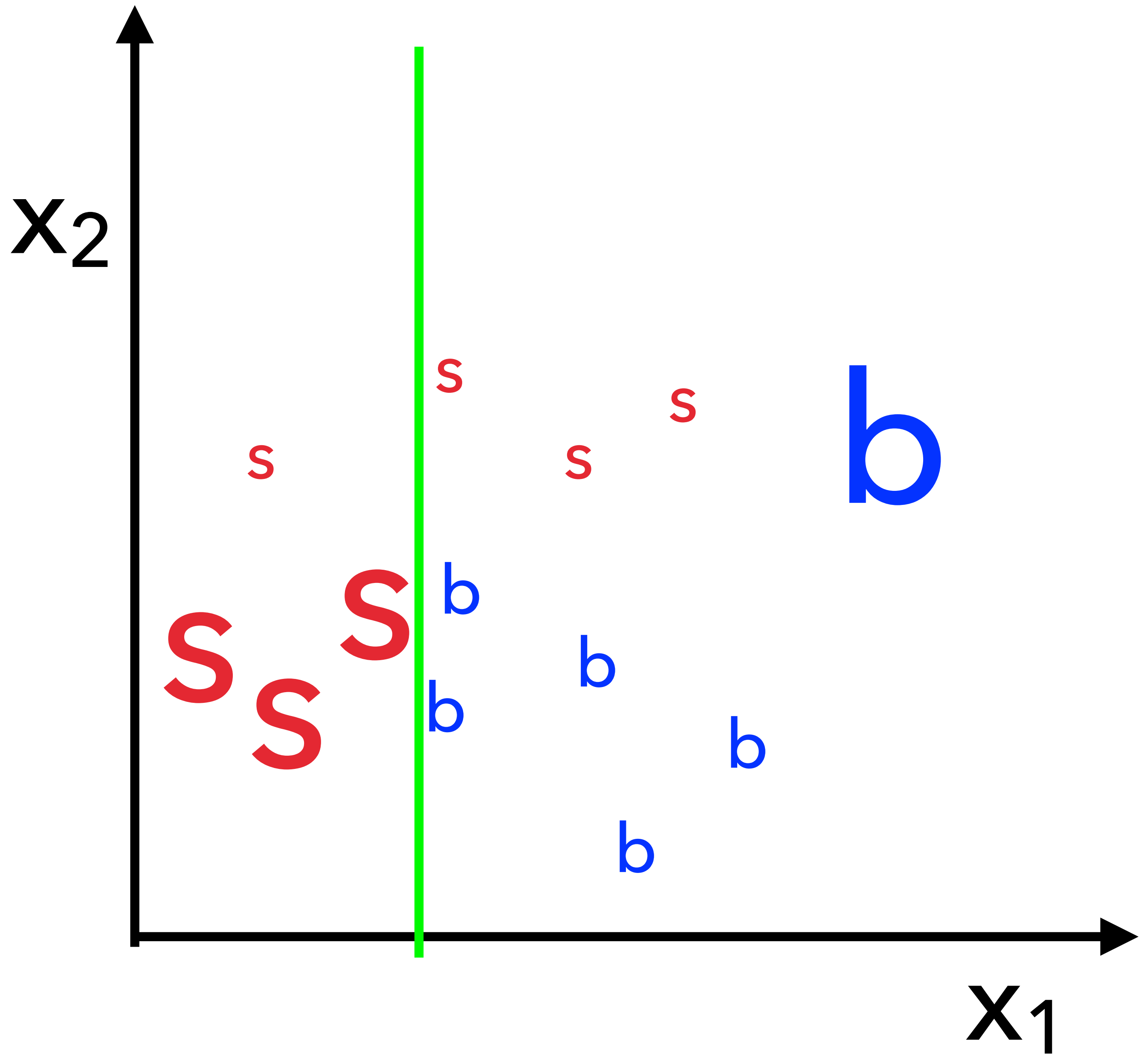
- This can be achieved by increasing weight to wrongly classified events and decreasing weight to correctly classified events



DECISION TREES BOOSTING

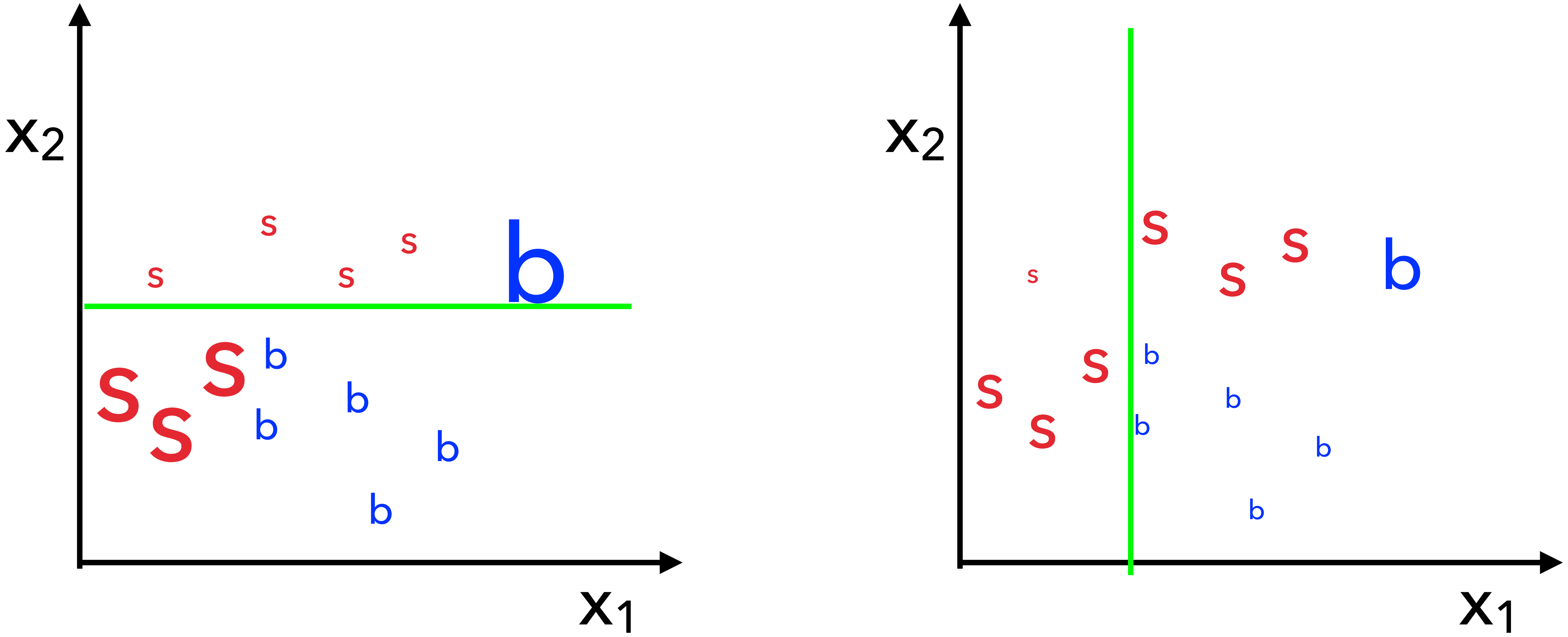
- Build new tree that focuses more on events with more weight





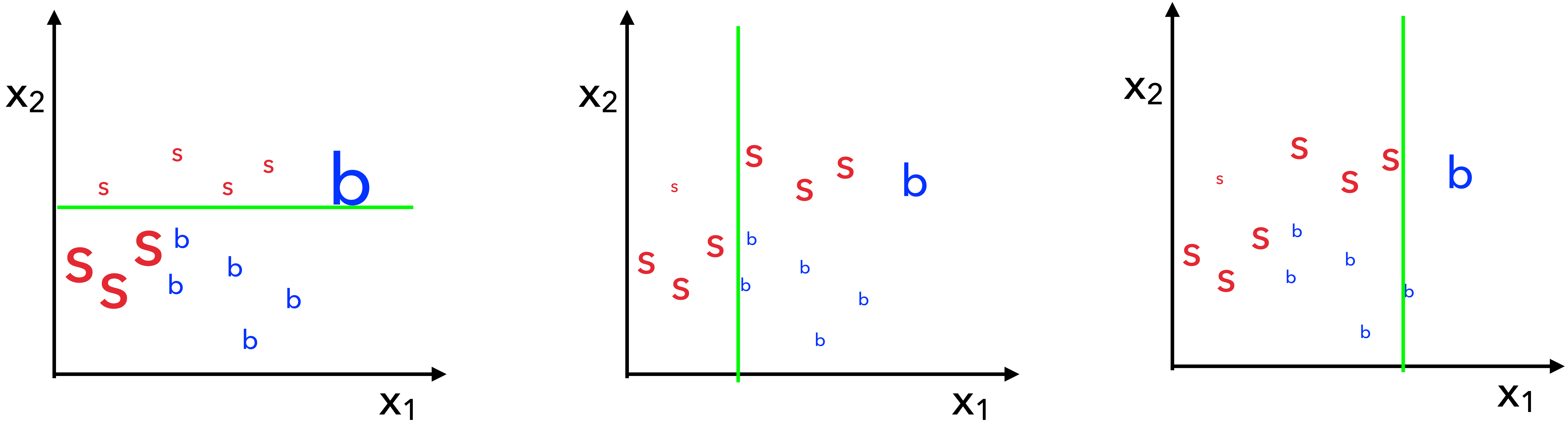
DECISION TREES BOOSTING

- Again increase weight to wrongly classified events and decrease weight to correctly classified events



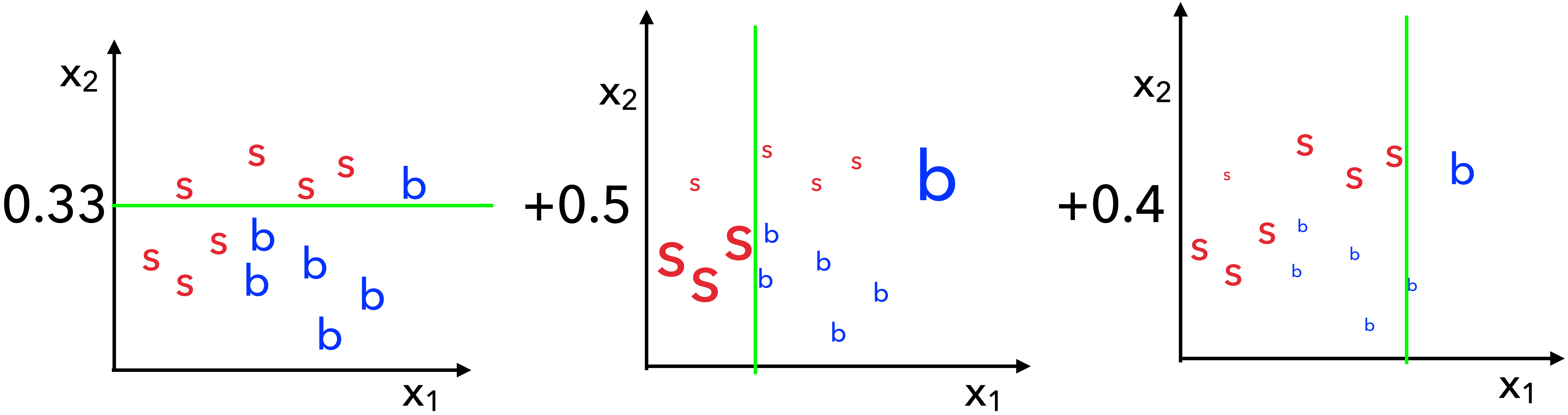
DECISION TREES BOOSTING

- Again increase weight to wrongly classified events and decrease weight to correctly classified events



DECISION TREES BOOSTING

- Finally assign numerical values to two classes (b=-1 s=1) and assign a weight to each of the trees and sum them

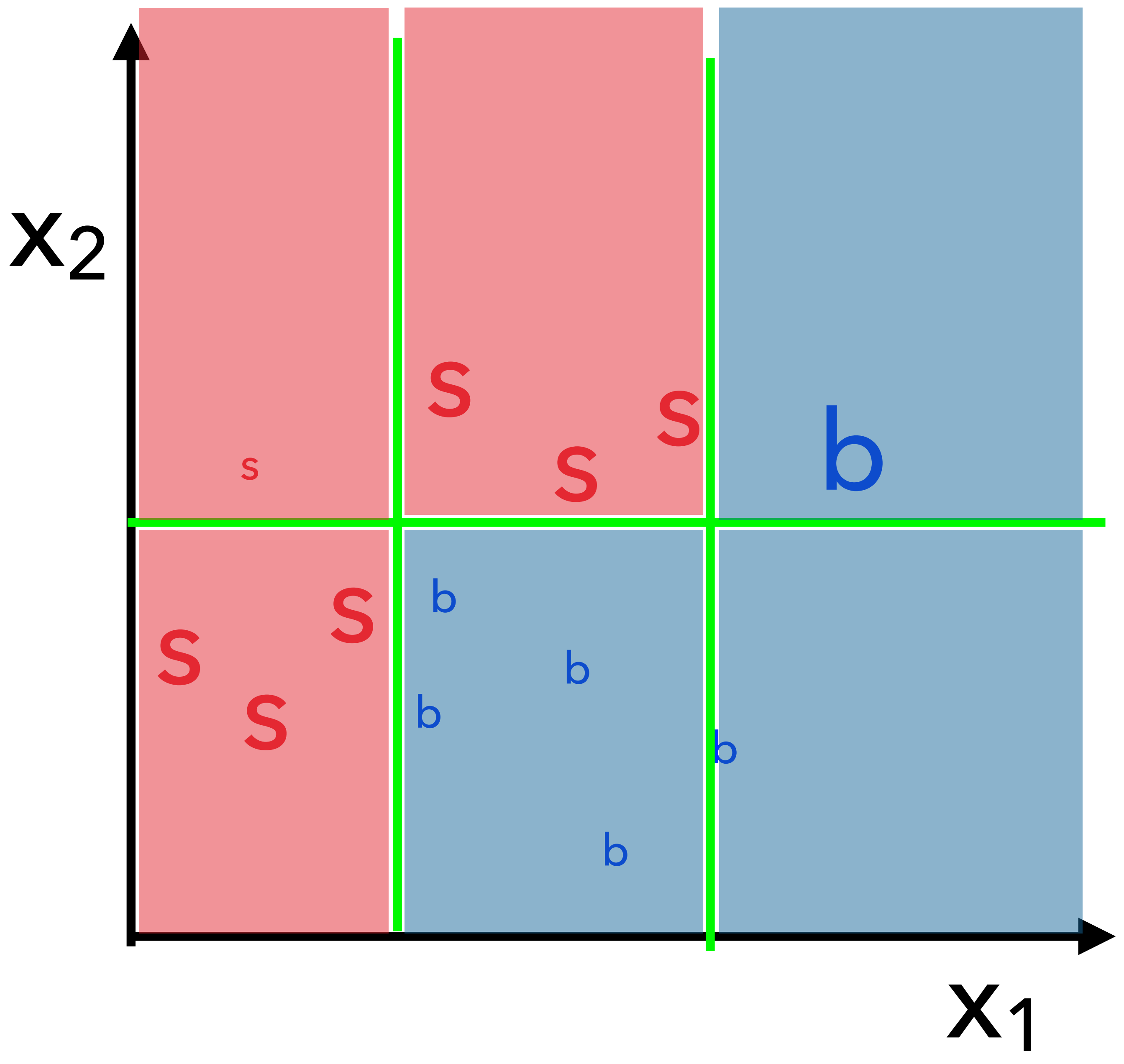


DECISION TREES BOOSTING

- Finally assign numerical values to two classes (b=-1 s=1) and assign a weight to each of the trees and sum them
- Many different algorithms exist to set the weights of different trees

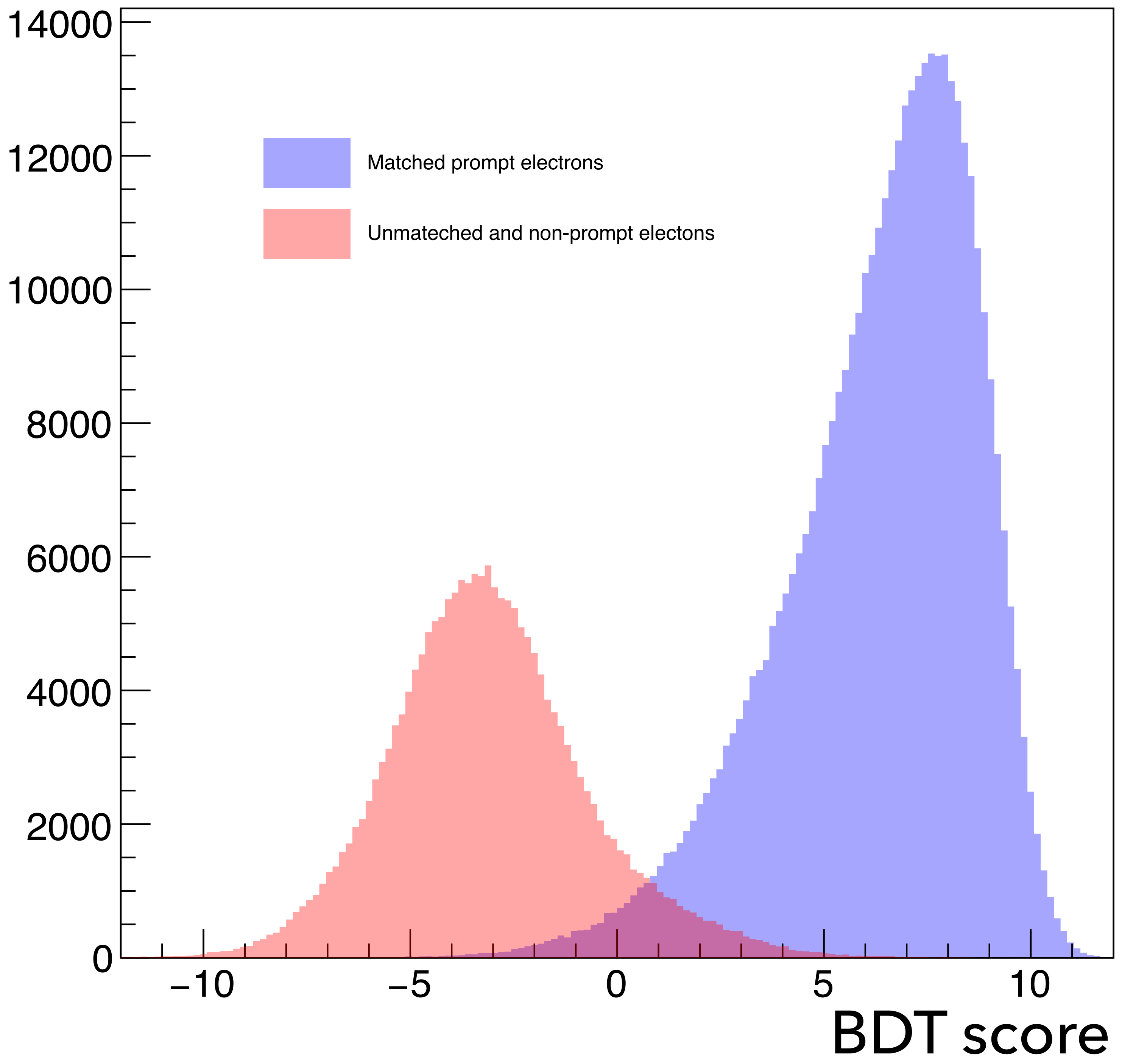
>0=s

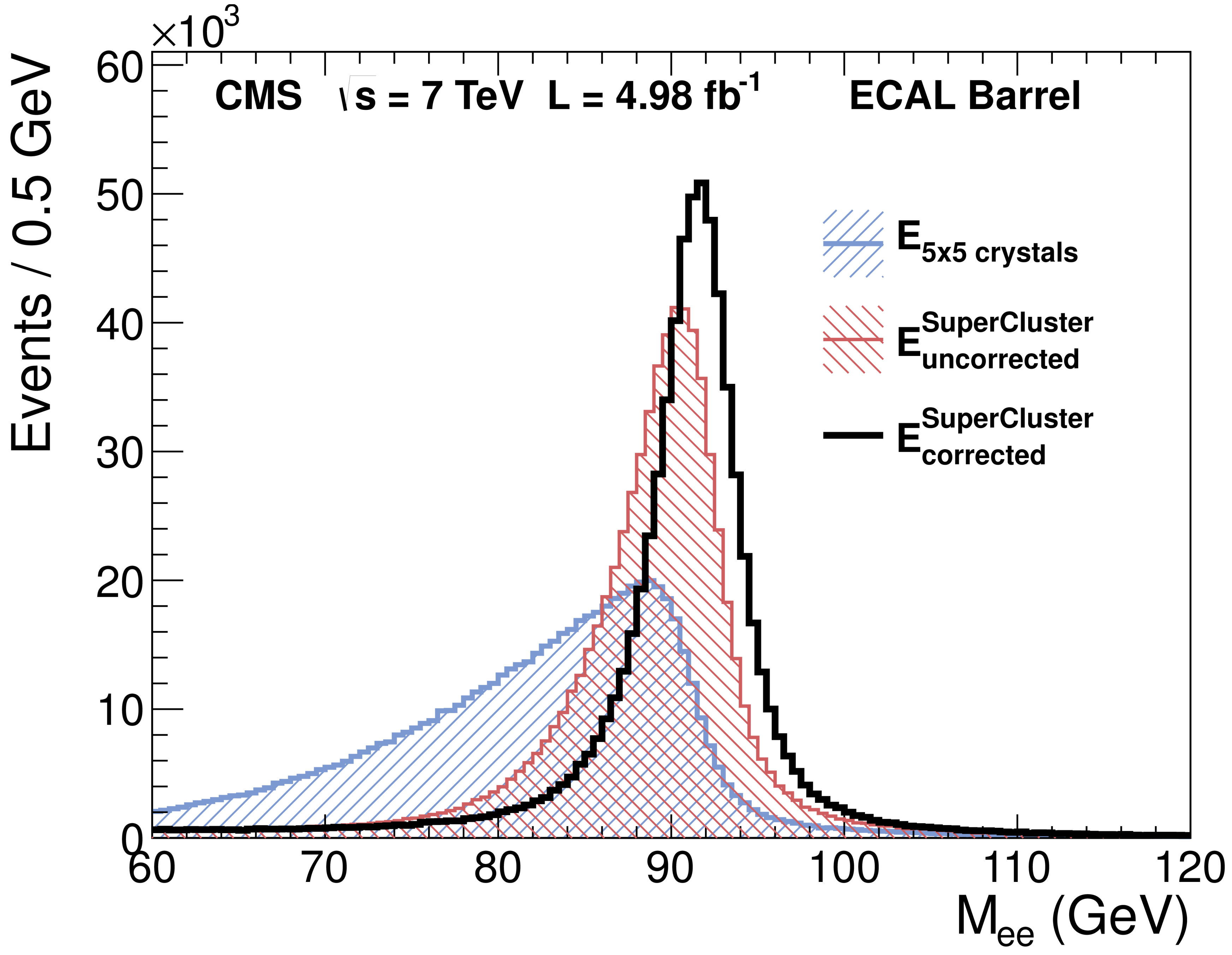
<0=b



BOOSTED DECISION TREES

- Now you know how Boosted Decision Trees (BDTs) work!
- Overtraining is completely in our control by taking special care of BDT parameters (number of trees, max depth of a tree, minimum number of events in leaves, parameters specific to pruning/boosting/bagging, …)
- Adding correlated variables will not degrade the performance of the BDT because the less discriminating one will be de-weighted


BDT score
Matched prompt electrons
Unmateched and non-prompt electons


×10^3
Events / 0.5 GeV
M_ee (GeV)
CMS √s = 7 TeV L = 4.98 fb^-1
ECAL Barrel
E_5x5 crystals
E^SuperCluster_uncorrected
E^SuperCluster_corrected