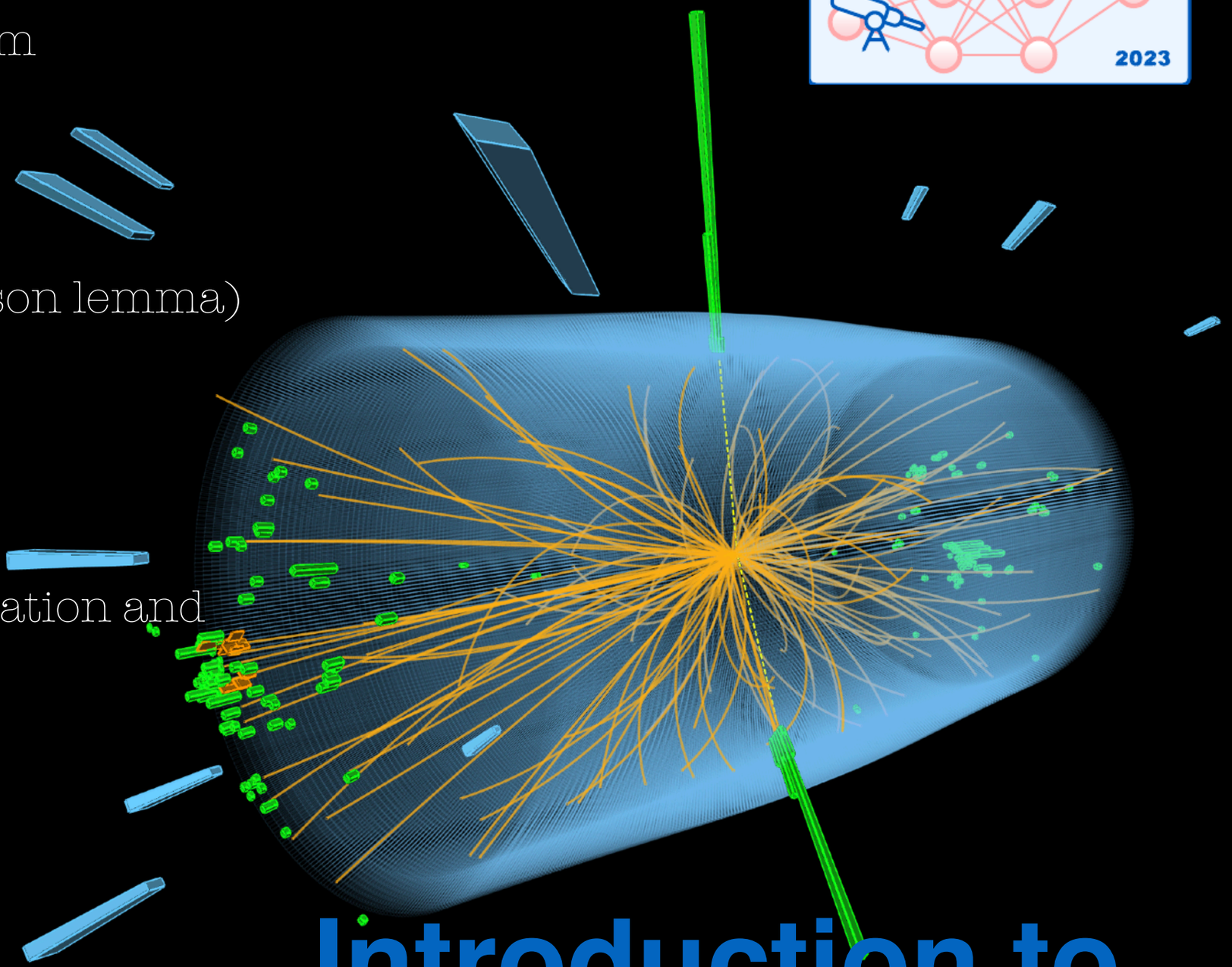
What you will see:

- A new programming paradigm
- Classifiers as test statistics
- Significance / Power
- Type I / II Errors
- ROC Curves
- Best classifier (Neyman Pearson lemma)
- Algorithms:
    - Histograms
    - KDE
    - KNN
    - Curse of dimensionality
    - Decision Trees for classification and regression
        - Boosting
        - Bagging
- HEP examples



Introduction to HEP Data analysis and ML

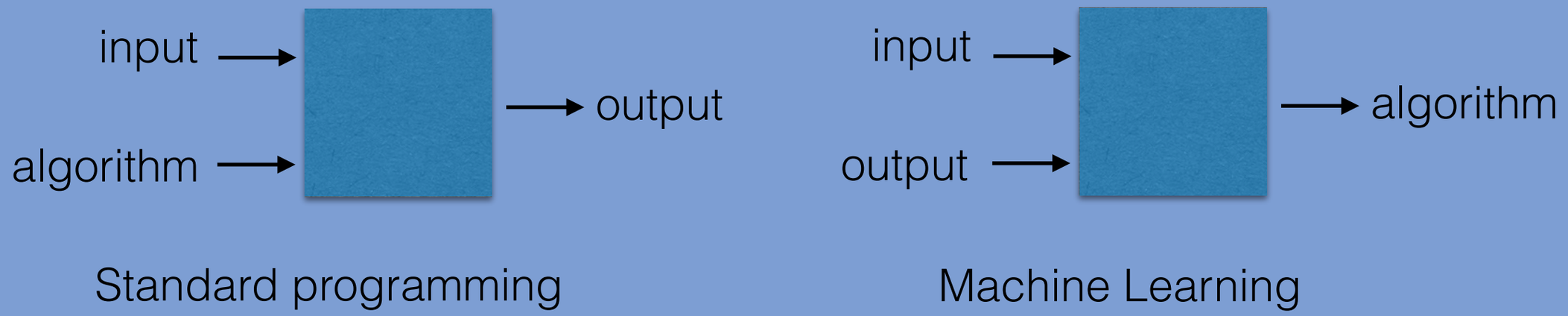
A new programming paradigm

Artificial Intelligence (AI)

The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning, … (see Russel Norvig)

Machine Learning

Machine learning is the technology of getting computers to act without being explicitly programmed.



The big picture

Artificial Intelligence (AI)

The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning, … (see Russel Norvig)

Machine Learning

Machine learning is the technology of getting computers to act without being explicitly programmed.

Supervised Learning

"Classification problems"

Unsupervised Learning

"Clustering"

Supervised: you instruct the algorithm using a sample where you give the correct classification
Unsupervised: you let the algorithm find out what are the characteristics of the data and define the classes

Classification: you assign each member of a data sample to a discrete number of categories
Regression: you assign each member of a data sample to a continuous value

Define the problem

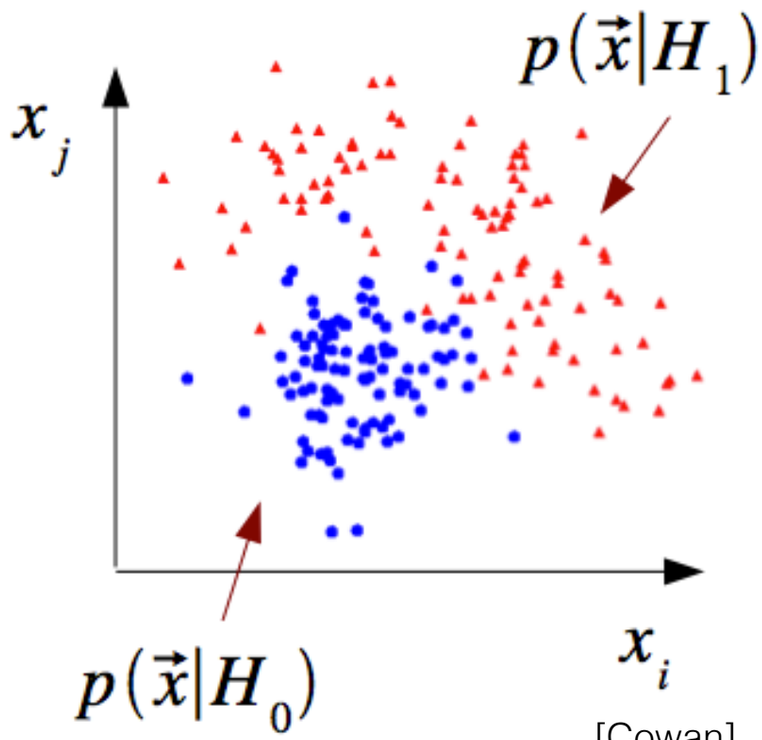
There are many ways to approach ML (CS, statistics, etc..).
I will show some interesting link between ML and Statistics

Suppose we collect events at a collider and we want to separate signal (e.g. Higgs) from background (e.g. QCD). For each event we measure a number of observables:
$\vec{x}$ = (x_1, x_2, x_3,.., x_n) = (#jets, tracks p_T, η, ϕ, MET, tracks dE/dX, #jets, b-tag, …)
Each event $\vec{x}$ lives in a n-dimensional phase space / feature space (where n can be >> 1)

If signal and noise populate different regions of phase space, we can try to separate them.

Translated with some statistics jargon:
$\vec{x}$ follows some n-dimensional joint probability distribution pdf($\vec{x}$|H_i) depending on event hypothesis:
H_0, null-hypothesis, the event is background
H_1, alternative-hypothesis, the event is signal

The goal is to separate the events in the two classes (sig / bkg) taking advantage of the information carried by the observables in $\vec{x}$.
Build a classifier using ML (Multi Variate Analysis)

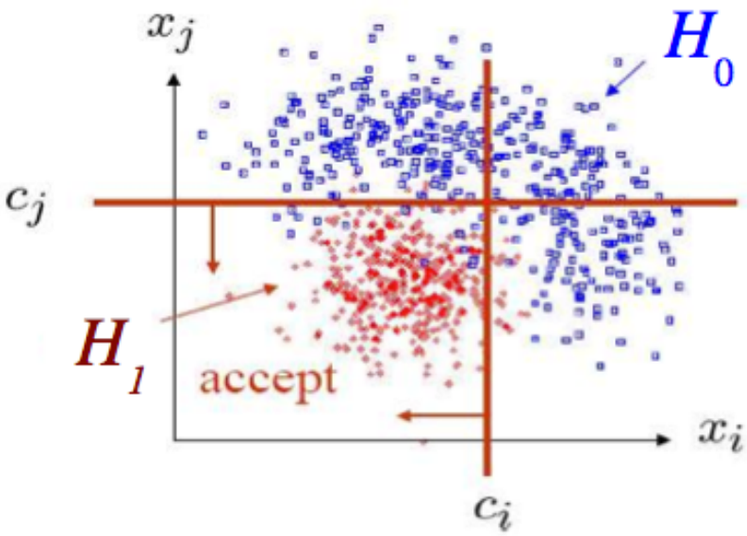


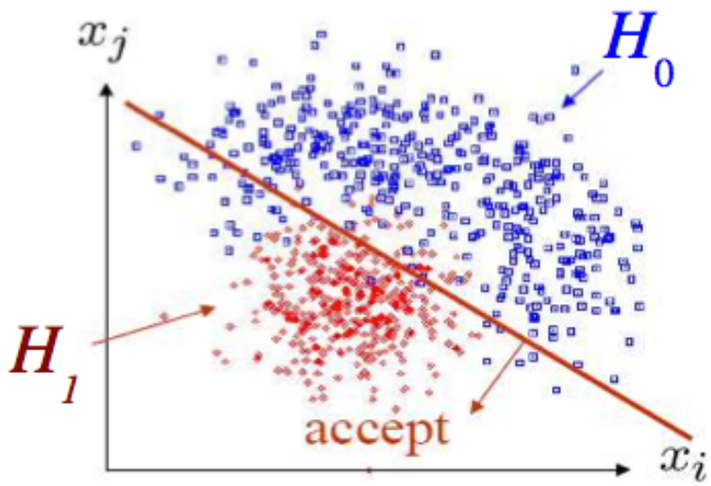
[Cowan]

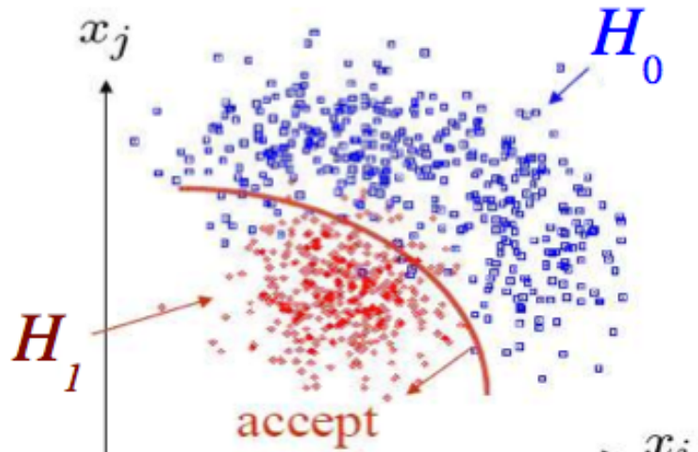
Classifier as a test statistic

You can separate the different classes in several ways by choosing an appropriate boundary

[Cowan]







"Cut based"
(leading to Decision Trees)

Linear discriminant
(Fisher discriminant)

Non-linear discriminant
(Neural Networks, BDTs)

The classifier y is a function of the data "Test Statistic" that allows to reduce the problem dimensionality. Often from n down to 1 !
Typically real number (0<— bkg sig—>1) or even binary (0 = bkg, 1= sig)

The separation between categories is then achieved setting a "decision boundary" as

$y(\vec{x})$ = constant

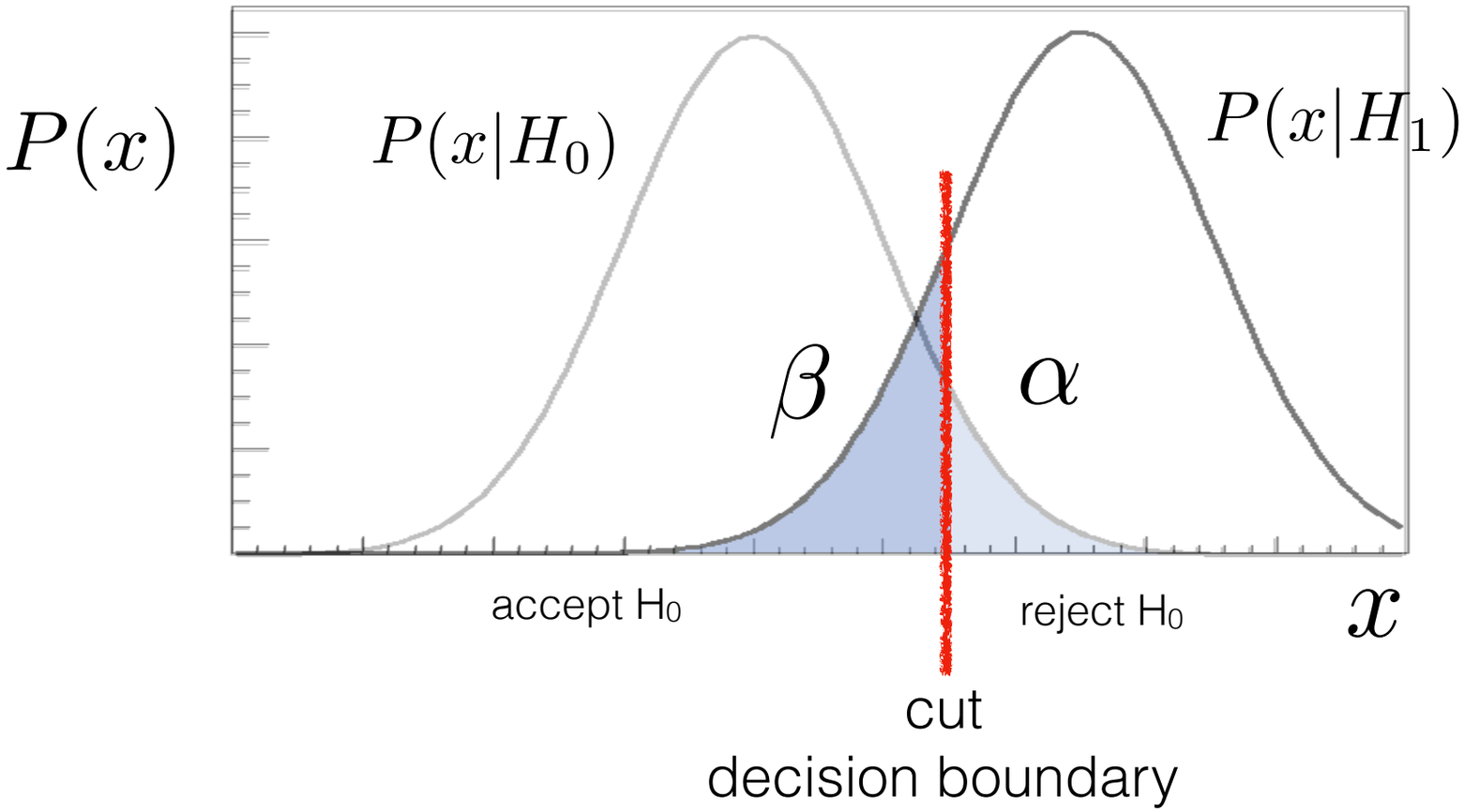
Is there an "optimal way" to separate sig/ bkg ?

Hypothesis testing

H_0 = null hypothesis = background-only
H_1 = alternative hypothesis = signal + background
(…there is no signal-only hypothesis,
the background is always there)

To discover a signal you exclude the background only hypothesis.
We will never be able to say that we have discovered the SM Higgs boson in 2012, we can only exclude that it is NOT the SM Higgs boson.

Significance of a test



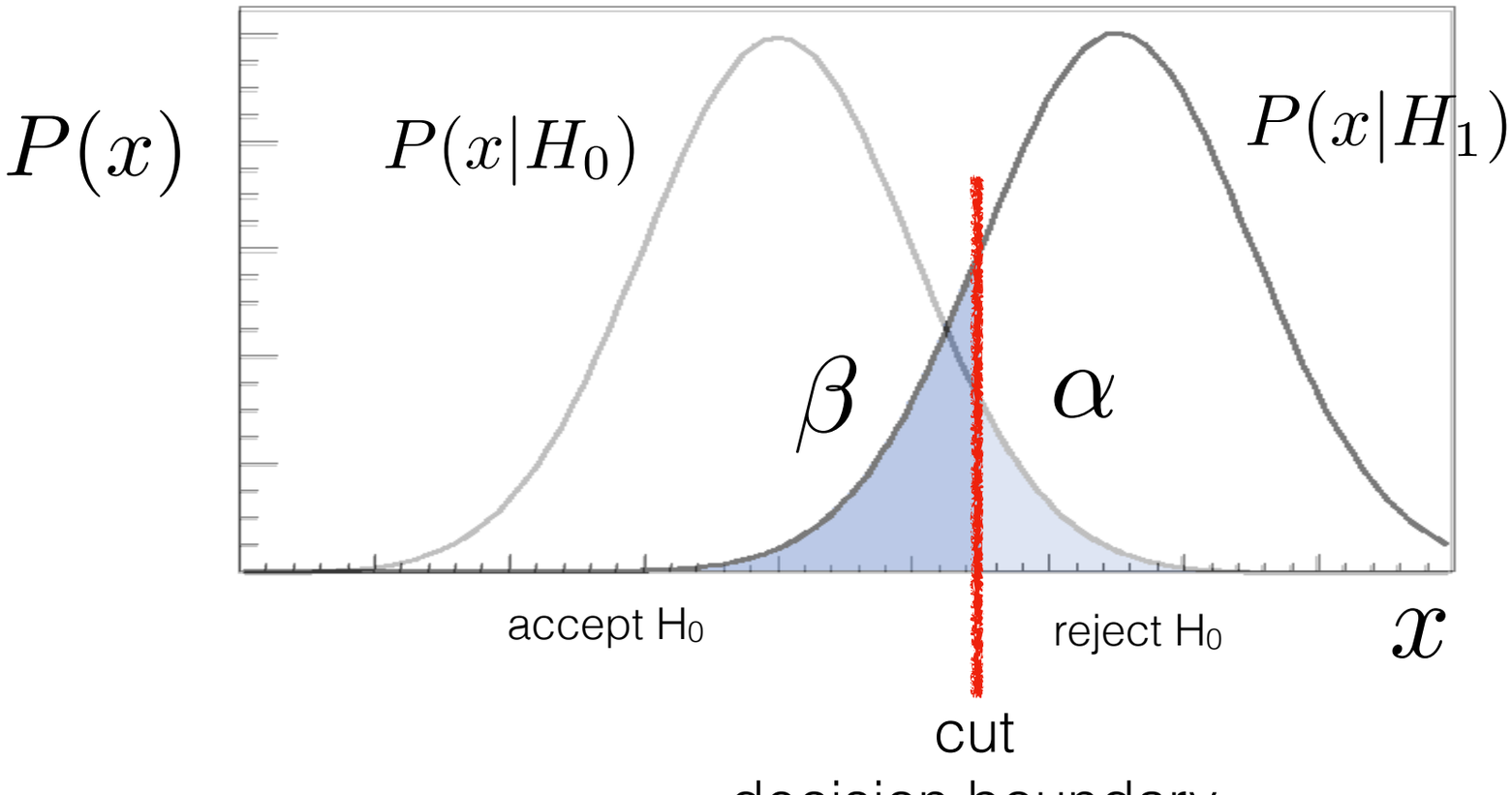
$P(x|\ H_0)$

if x > cut you reject the H_0 (bkg HP)
α = significance of the test : you have a probability α to reject H_0 when true.

Type I Error: reject a true hypothesis (loss, false negative)

NB: You decide the cut, you decide the significance α. Typical values are 5%, 1%, we also say a test has a significant level of 1-α (typically 95%, 99% level)

Power of a test



$P(x|\ H_1)$

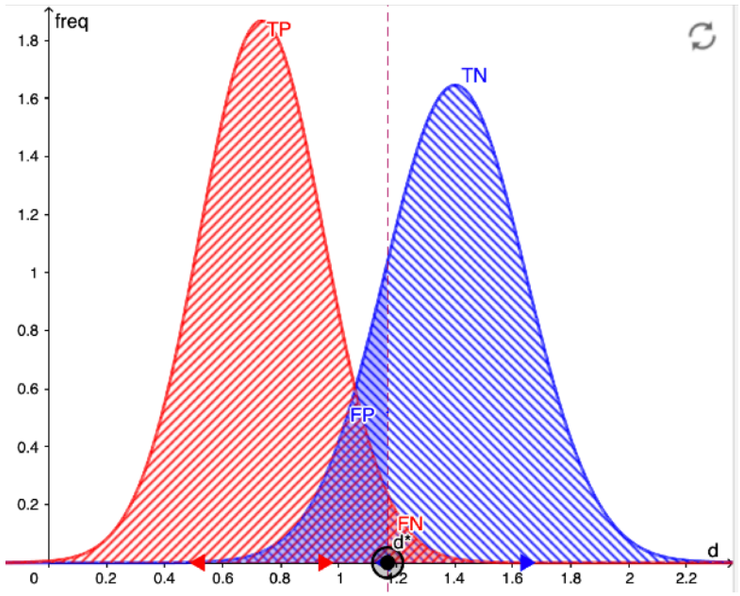
$\beta =$ the probability to accept the null hp H_0 when H_1 is true
1-β = power of the test

Type II Error: accept a false hypothesis (contamination, false positive)

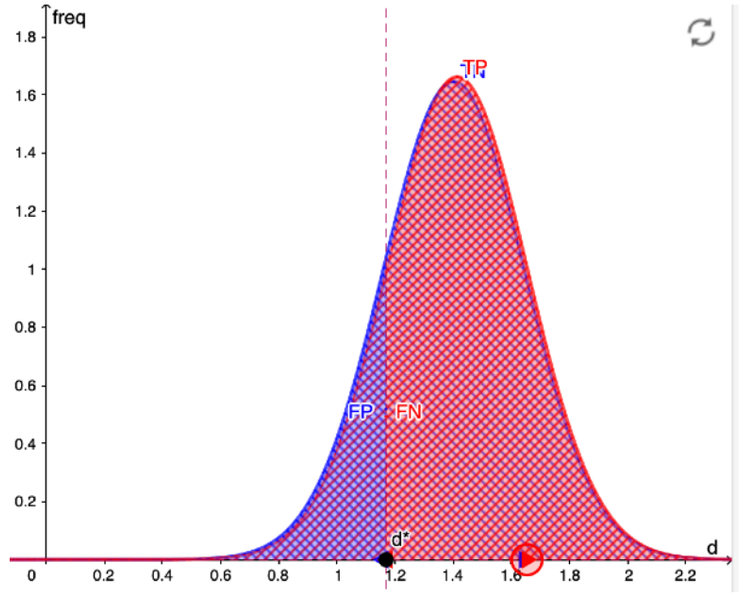
The “separation” between signal and bkg is given by the choice of your classifier. Then by choosing a threshold on the signal efficiency (Type I error) you will get a background rejection (Type II error)

Error types

A good classifier (test statistic) is the one with both α and β small,
i.e. high significance and high power
i.e. H_0 and H_1 very different: “large separation”



high significance
high power



low significance
low power

https://www.geogebra.org/m/tza5dudu

Misinterpreting the data
Type I error: reject a true hypothesis (loss, false negative)
Type II error: accept a false hypothesis (contamination, false positive)

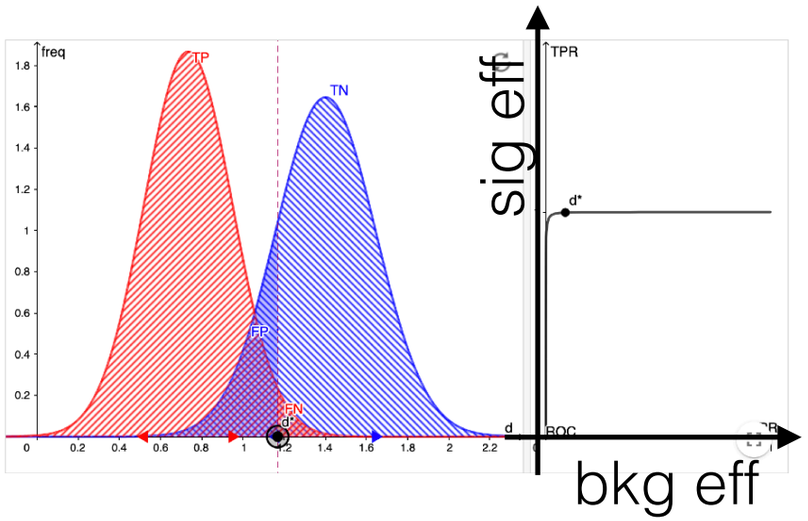
Example: You see a new bump in your data (H_0 = bkg only, H_1 = sig+bkg)
Type I : there’s really no resonance, you reject H_0 and you announce a non existing discovery
Type II: there is a real resonance, you accept H_0 and you miss the Nobel prize

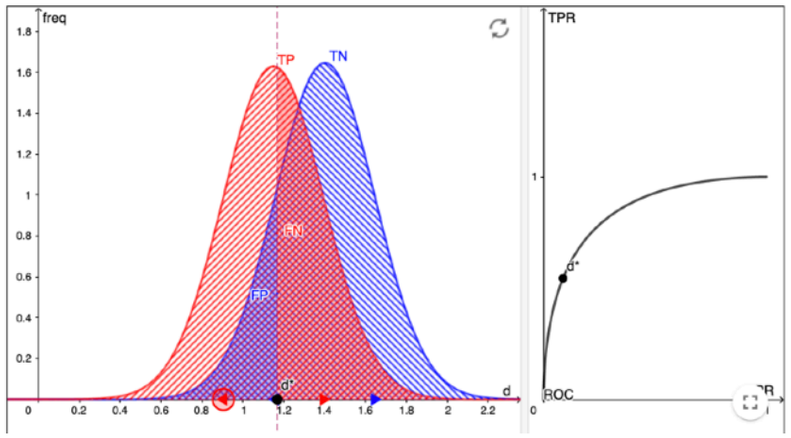
In practice…

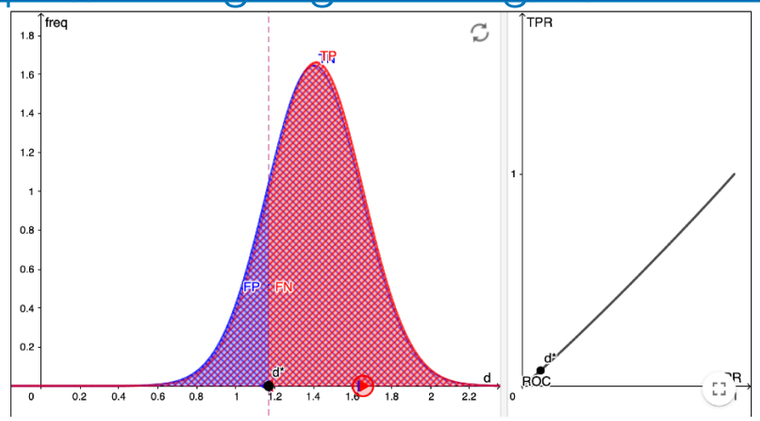
Build the pdfs under $H_0(H_1)$ by throwing toy data distributed as $H_0(H_1)$ and fill an histogram with the values of the classifier. The cut leaves α fraction of events in the tail of pdf(x|H_0) and β fraction of events in the tail of pdf(x|H_1)

Very often we use ROCs (Receiver Operators Curves) to show the performance of a classifier

https://www.geogebra.org/m/tza5dudu







NB: depending on what you plot the curve will be flipped

Background efficiency = 1-α (a test is significant at a level 1-α %)
α = probability of a type I error = background rejection

Signal efficiency = β = probability of a type II error
1-β = power of the test = signal inefficiency

Purity = probability for an event to be signal, once we have accepted it as signal
(#signal evts passing the selection / #total eats passing the selection)

Best classifier ?

The “best” test is the one that makes both α and β as small as possible.
Such a test can be found if-and-only-if the hypothesis and the alternative are *fully specified* (simple hp).

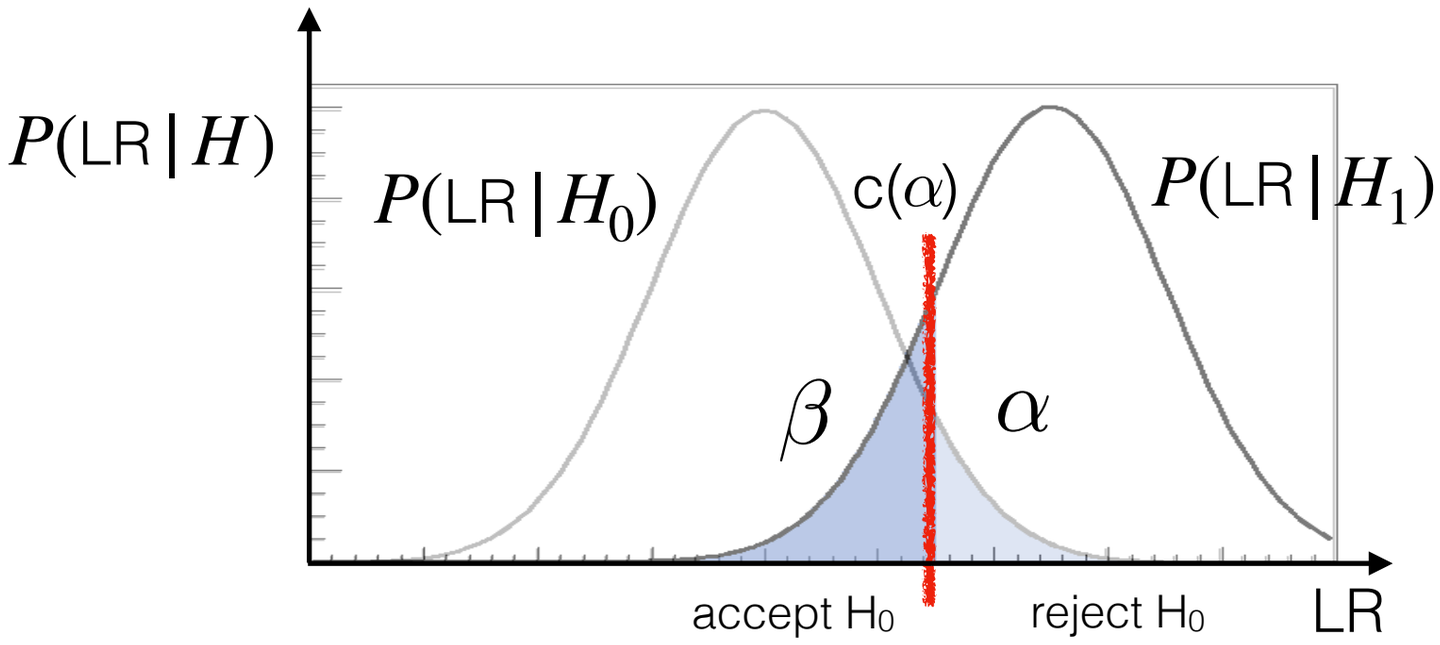
Neyman-Pearson lemma: given α (the significance you decided to have) the test statistic that maximises the power against the alternative hp H_1 is the likelihood ratio

$$\text{LR} = \frac{g(t|H_0)}{g(t|H_1)}$$

Choice of the critical region (reject H_0):
c is determined by the desired significance (α) —> c = c(α)

$$LR > c(\alpha)$$

At this point whatever is the value of β, NP guarantees you that is the smallest you can get, meaning that whatever other test statistic you can build, it will have a larger β.



Machine Learning

NP: the optimal discriminating function is given by the likelihood ratio:

$$\text{LR} = \frac{g(t|H_0)}{g(t|H_1)}$$

Writing explicitly the likelihoods is often impossible and the best we can do is to approximate it.

The algorithms that will see can be used to “learn” the LR using (typically) simulated data.

On using MC to train a classifier: data/mc disagreement.

ML uses almost always MC samples to train algorithms: if the MC does not reproduce correctly in data distribution in the n-dimensional phase space / feature space (i.e. each dimension and all correlations) you may get a biased performance !

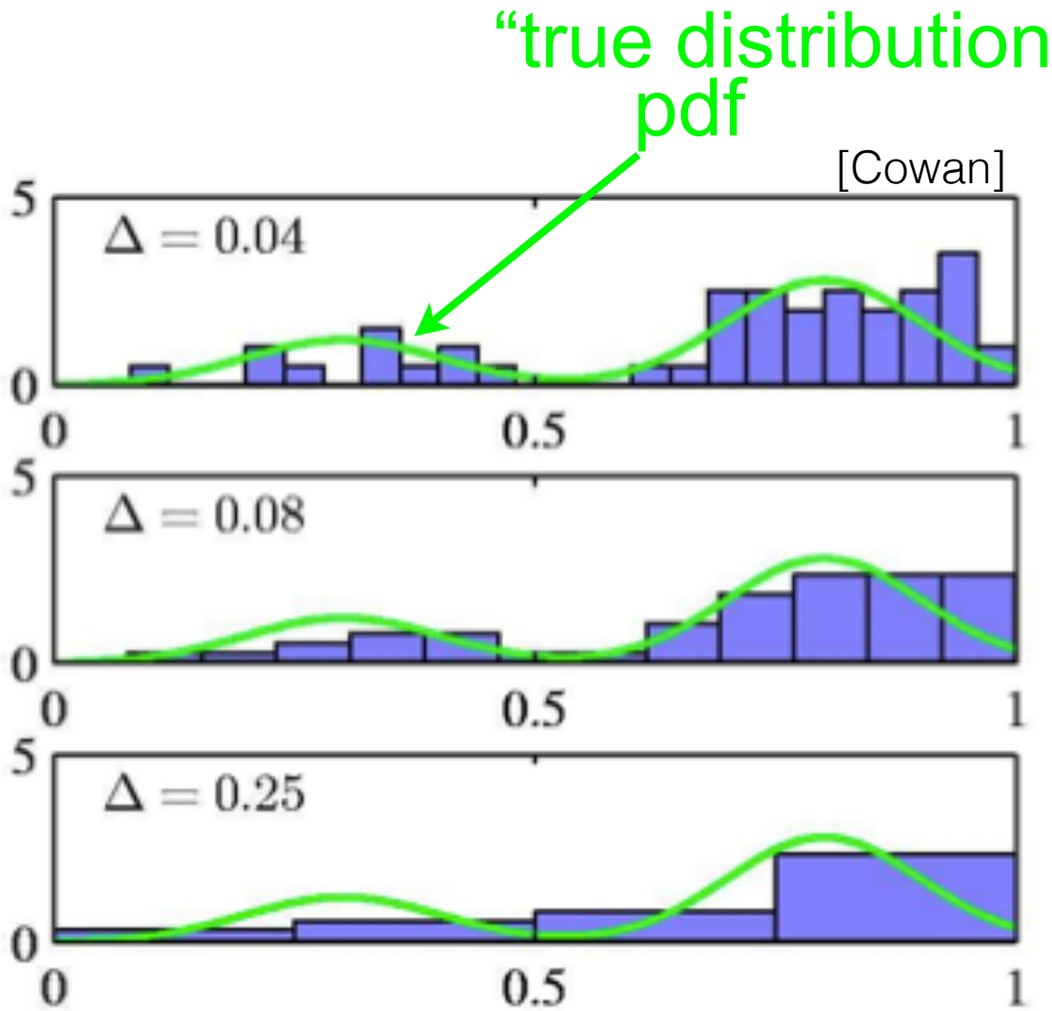
—> see Massimiliano Galli talk on Data/MC matching

Algorithms

Brute force: Histograms

Fill histograms from MC sig/bkg samples to build the pdfs for the Likelihood Ratio y(x)

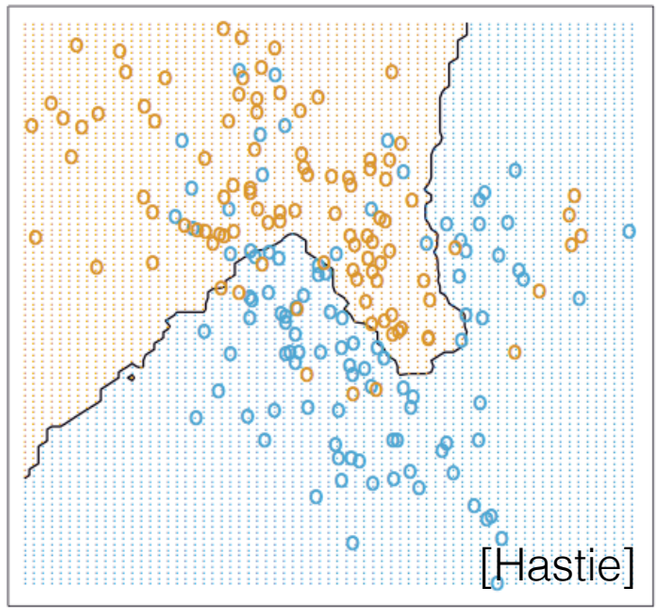
This is the simplest non-parametric model:
“brute force” approach



“Pro” : once the histogram is computed, the data can be discarded.
“Con”: discontinuities at bin edges, bad scaling with dimensionality.

The size of the bin has to be chosen to catch the structure of the pdf.

KDE and K-nearest neighbours



To construct the pdf at $x = (x_1, ..., x_D)$ we can count the number of events in some local neighbourhood of x *(requires definition of "local", i.e., a distance)* and then do a "majority vote"

The distance definition needs some care if you have variables with different ranges and units.

Given any point x you have to classify it as orange or blue (this is how you build the boundary)

Consider a small volume V centred about $x = (x_1, ..., x_D)$.
Suppose from N total events we find K inside the volume V.

Take as estimate for $p(x) = \frac{K}{NV}$

To optimize the classifier performance you can:

Fix K and determine V from data → K-nearest neighbour
Fix V and determine K from data → Kernel Density Estimators

KDE and K-nearest neighbours

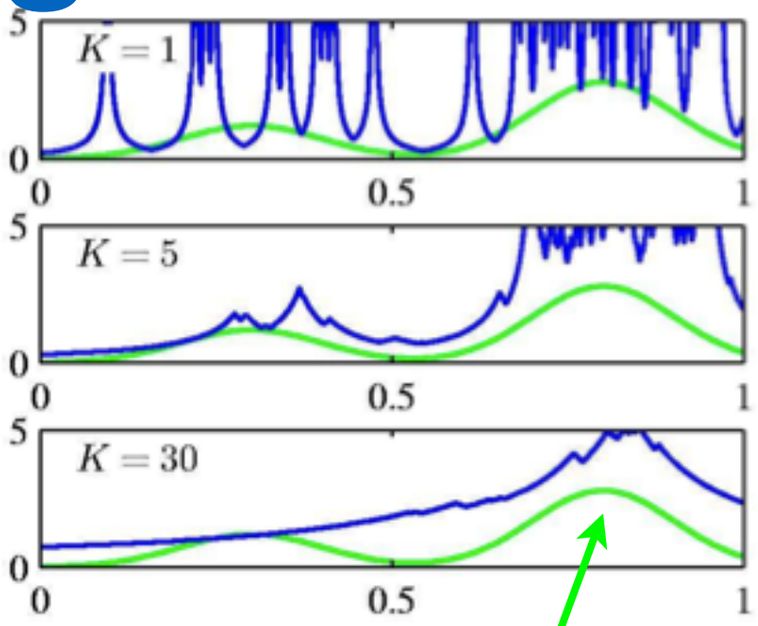
K-nearest neighbors

Fix the number of events K and find the volume V such that V contains K events.

smoothing parameter

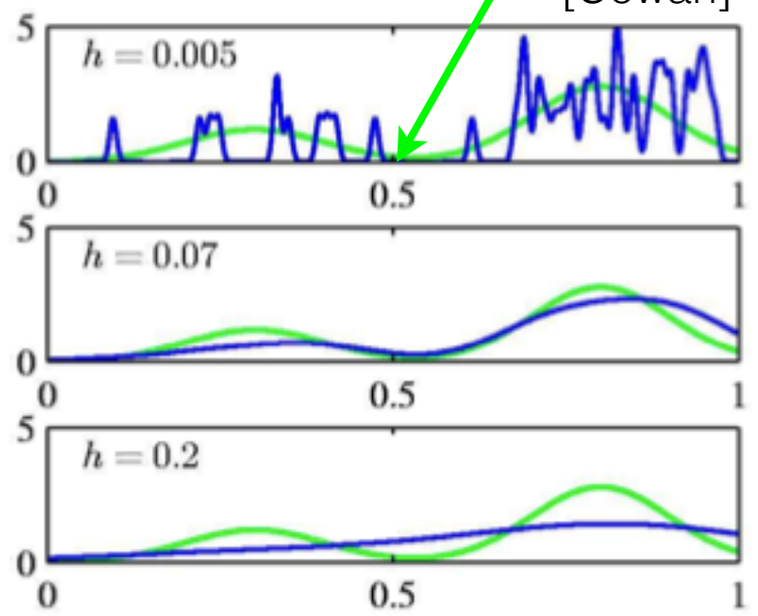
$$\hat{p}(\boldsymbol{x}) = \frac{K}{N V}$$

Large K means small stat error, but large volume, i.e. less local



"true distribution" pdf

[Cowan]



Kernel Density Estimator

example for Gaussian estimator:

Fix the volume V and find the number of events K: what volume ? e.g. gaussian or a cube in n-dimensions

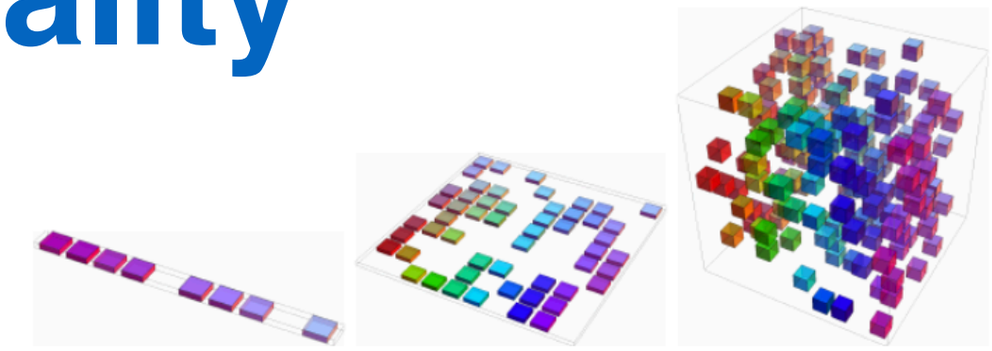
$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi h^2)^{1/2}} \exp\left(\frac{-\|\vec{x} - \vec{x}_i\|^2}{2h^2} \right)$$

e.g: place a Gaussian "kernel" of standard deviation h centred about each data point; At a given x, add up the contribution from all the Gaussians and divide by N.

https://www.deeplearningbook.org/

Curse of dimensionality

The volume grows exponentially, data points become very sparse and flat MC sampling becomes unfeasible



Suppose our data are uniformly distributed in a D-dimensional unit cube.
Sample an interval in 1D with 100 points: distance between points 10^{-2}
To sample a region in 2D with a distance between points of 10^{-2} you need 10^2 x 10^2 pts
In a typical HEP classifier/regression with o(50) dimensions $(10^2)^{50} = 10^{100}$ pts

Flip the argument: if you fix the number of points and you grow the number of dimensions, they become very sparse, i.e. you are not able to properly model the classifier

We need to find a smarter way to sample the phase space

All algorithm suffers one way or another from the curse of dimensionality !

BDTs will cope with it by boxing the phase space in a smart way, for NNs one of the hp is that they learn the manifold over which the data are distributed

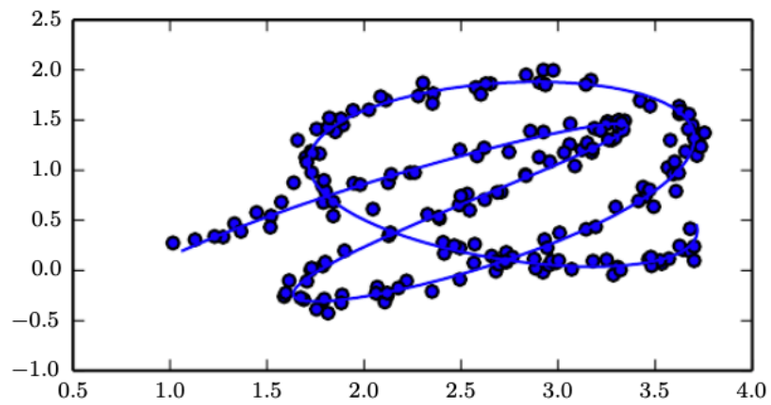
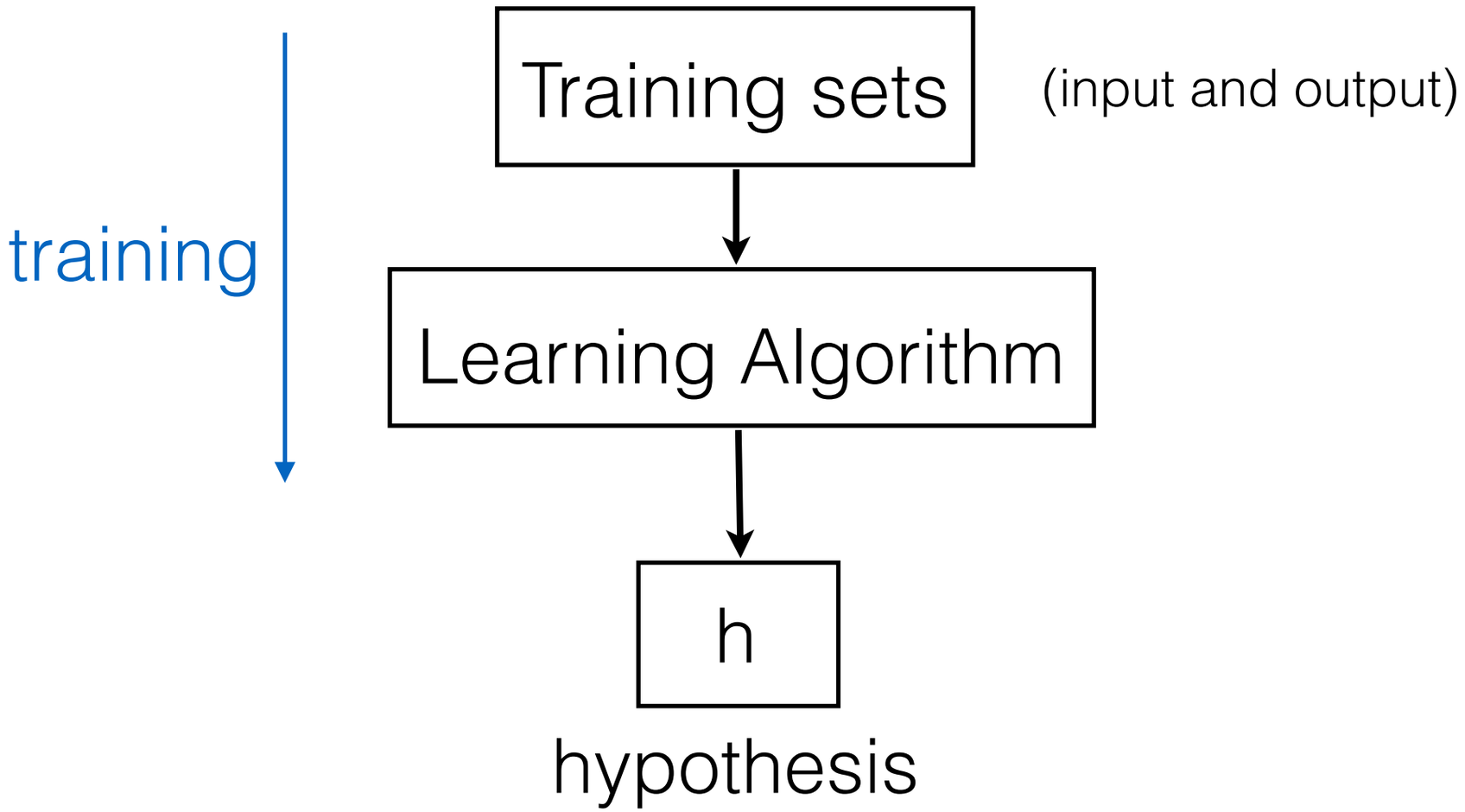
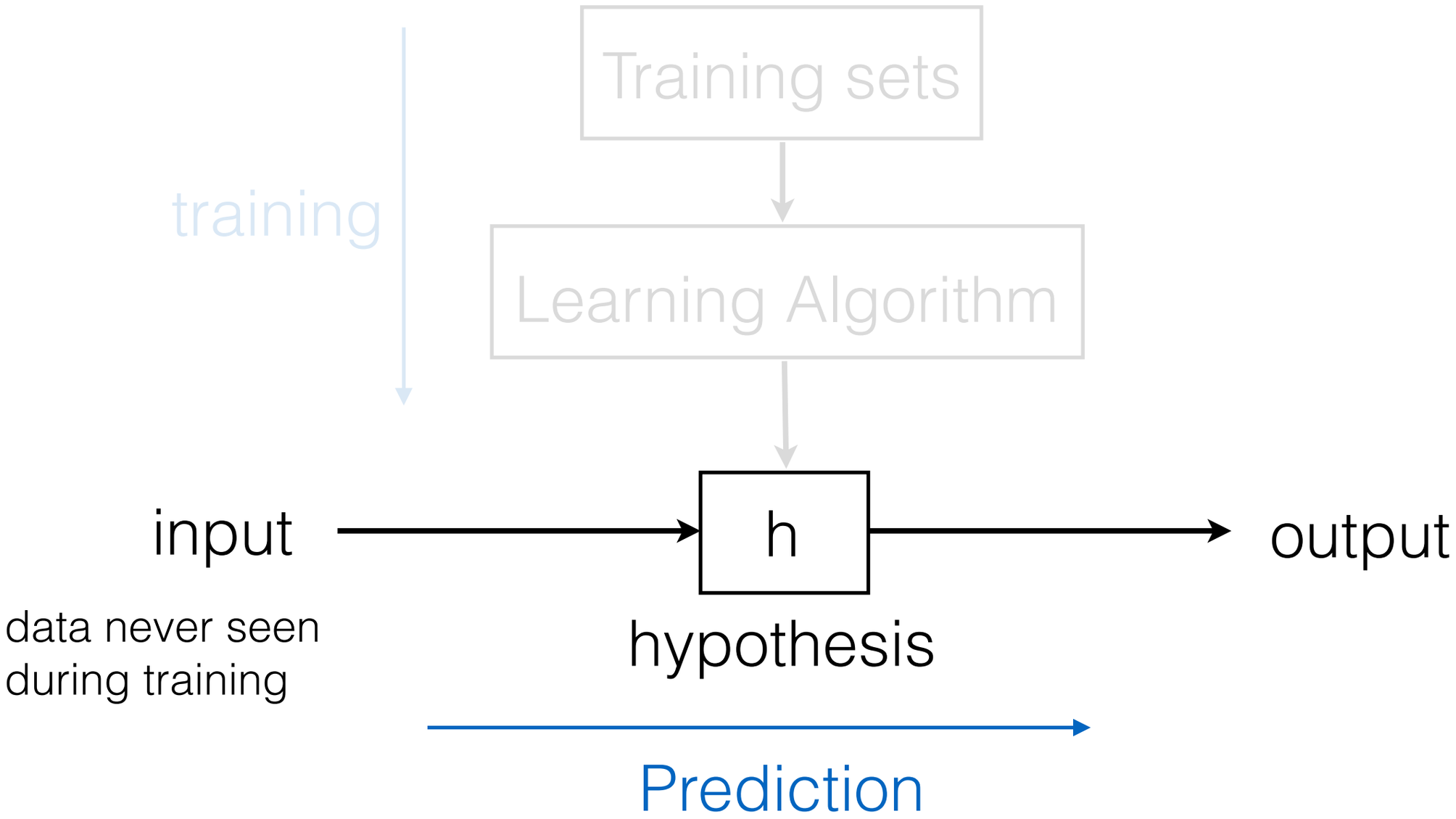


Figure 5.11: Data sampled from a distribution in a two-dimensional space that is actually concentrated near a one-dimensional manifold, like a twisted string. The solid line indicates the underlying manifold that the learner should infer.

Learning algorithm



Learning algorithm



Decision trees: classification

[ML - mm]

Training sample $\in \Re^2$

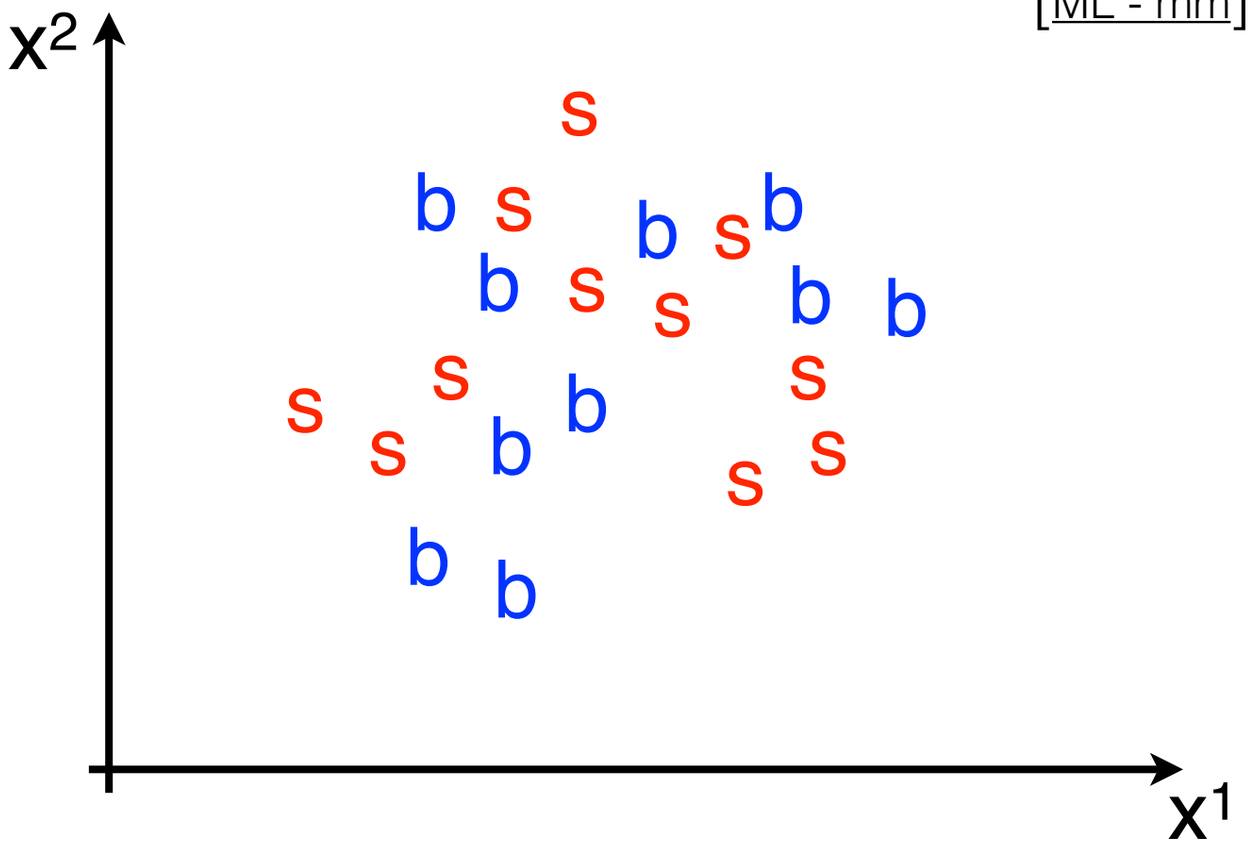
(x^1_1, x^2_1) classes

...

(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s



You might have non-numerical values for the coordinates:

tight / loose identification flag

The idea is to separate the classes using placing simple cuts
(i.e. binary splits of the data $x^i <$ value or $x^i >$ value)

Strategy is to minimize the misclassification at each step

Decision trees: classification

Training sample $\in \Re^2$

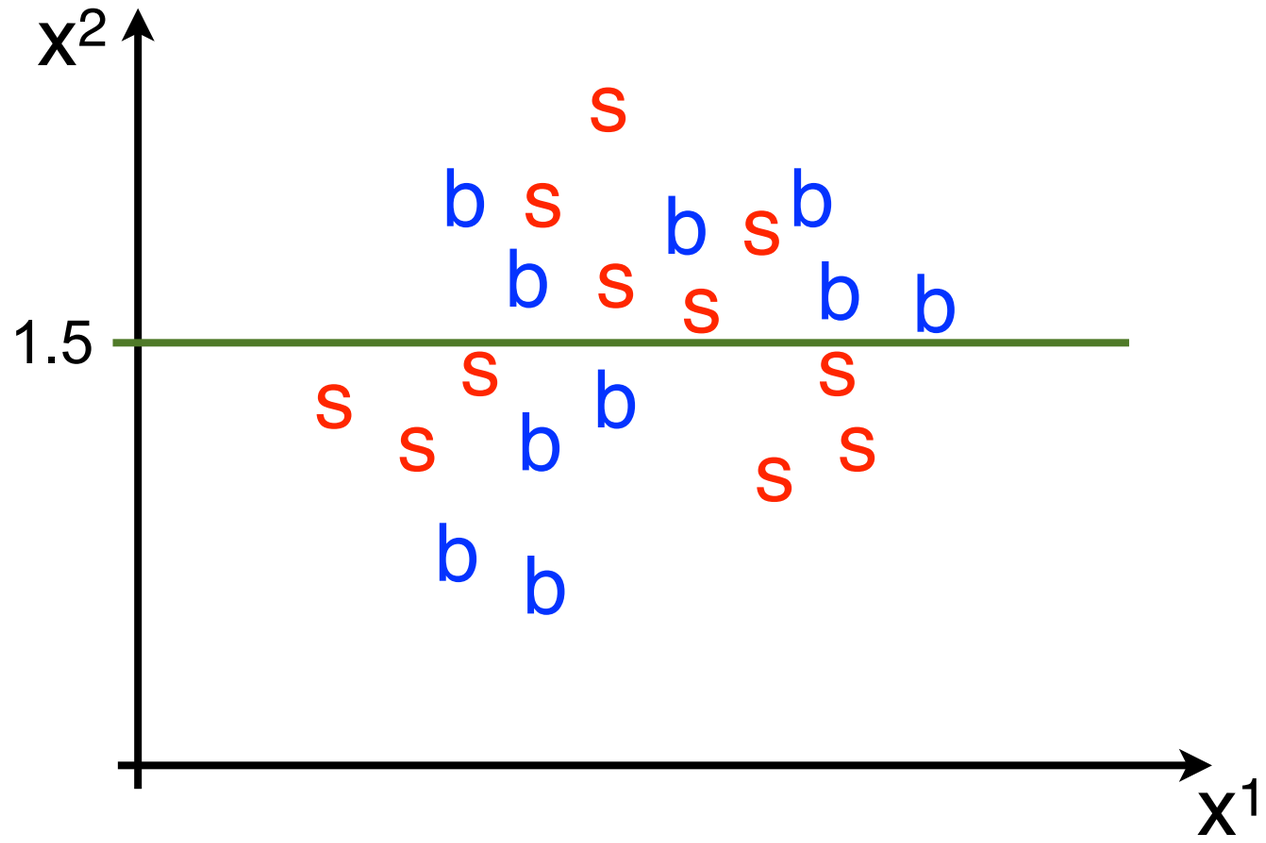
(x^1_1, x^2_1) classes

...

(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s



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" Gini = P(1-P) Where P =purity:

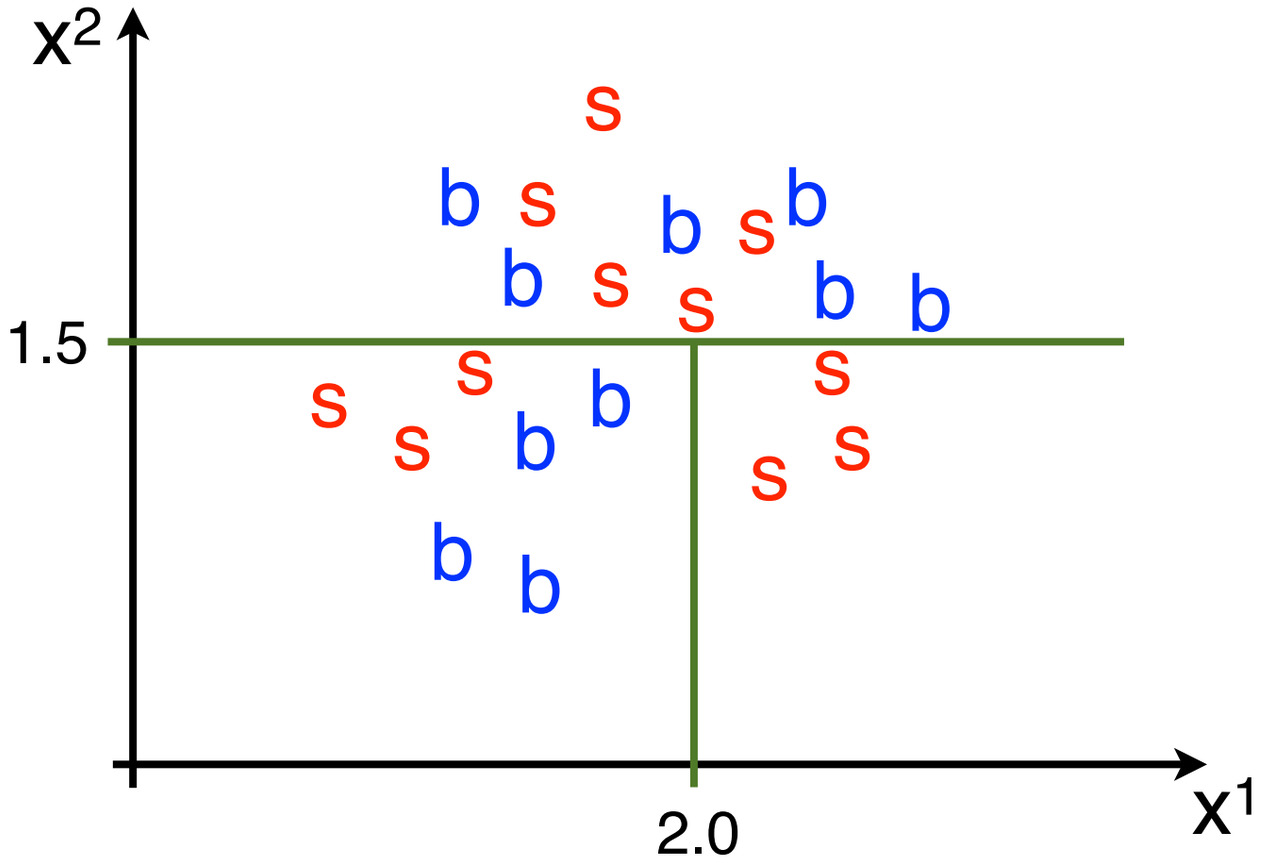
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

This is maximum for P = 0.5 (no separation / random guess) and zero for P = 0 or 1.
(having purity of 0 or 1 is the same, you always have max separation)

Decision trees: classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

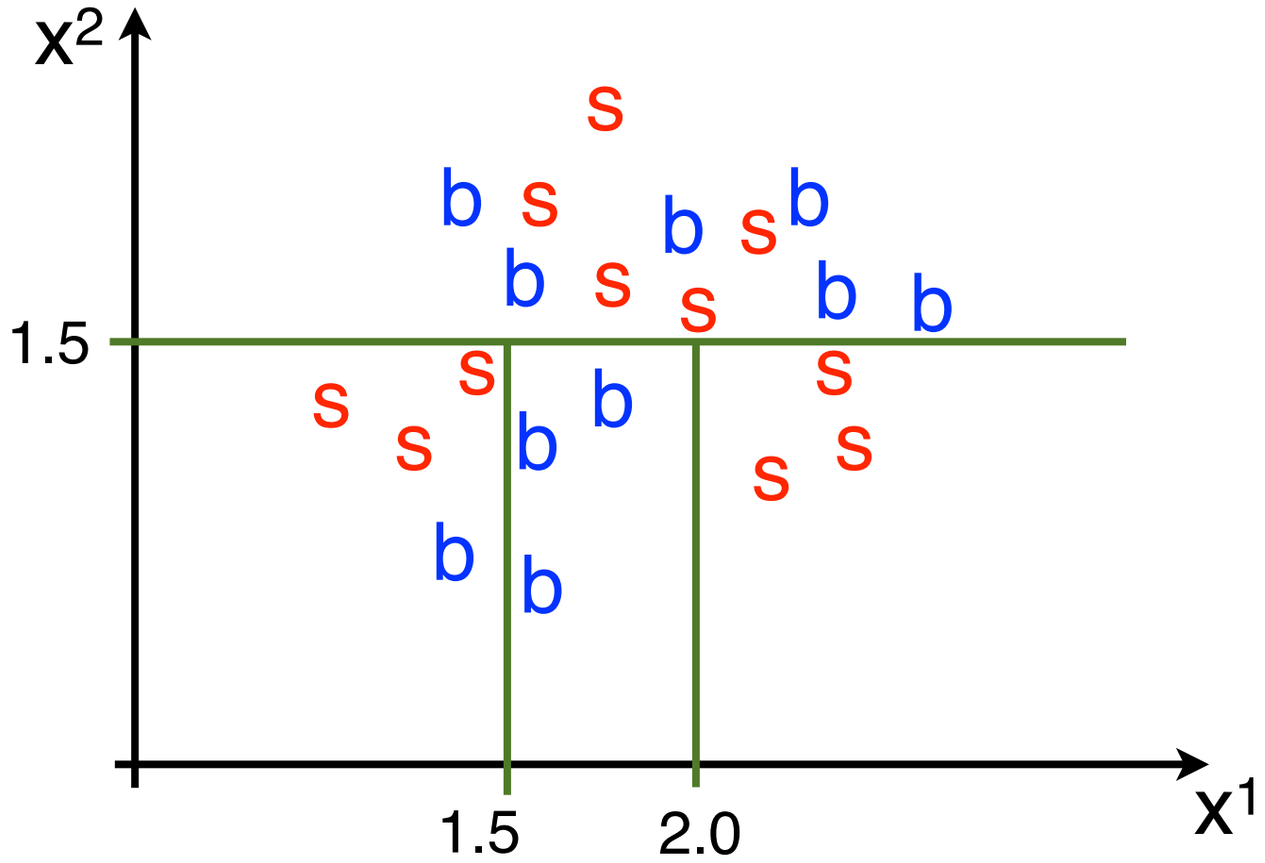


Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \mathfrak{R}^2$

(x^1_1, x^2_1)	classes
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s



Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \mathfrak{R}^2$

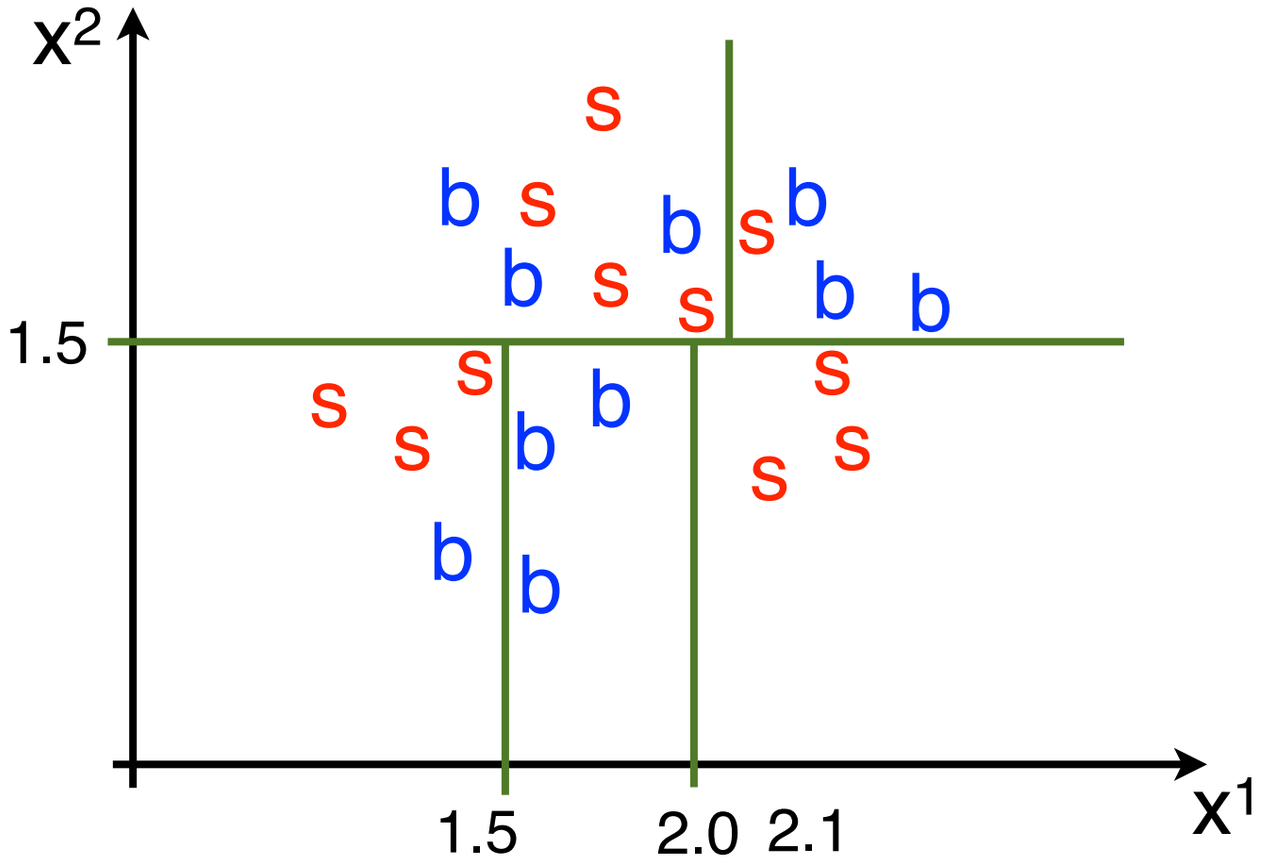
(x^1_1, x^2_1) classes

...

(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s

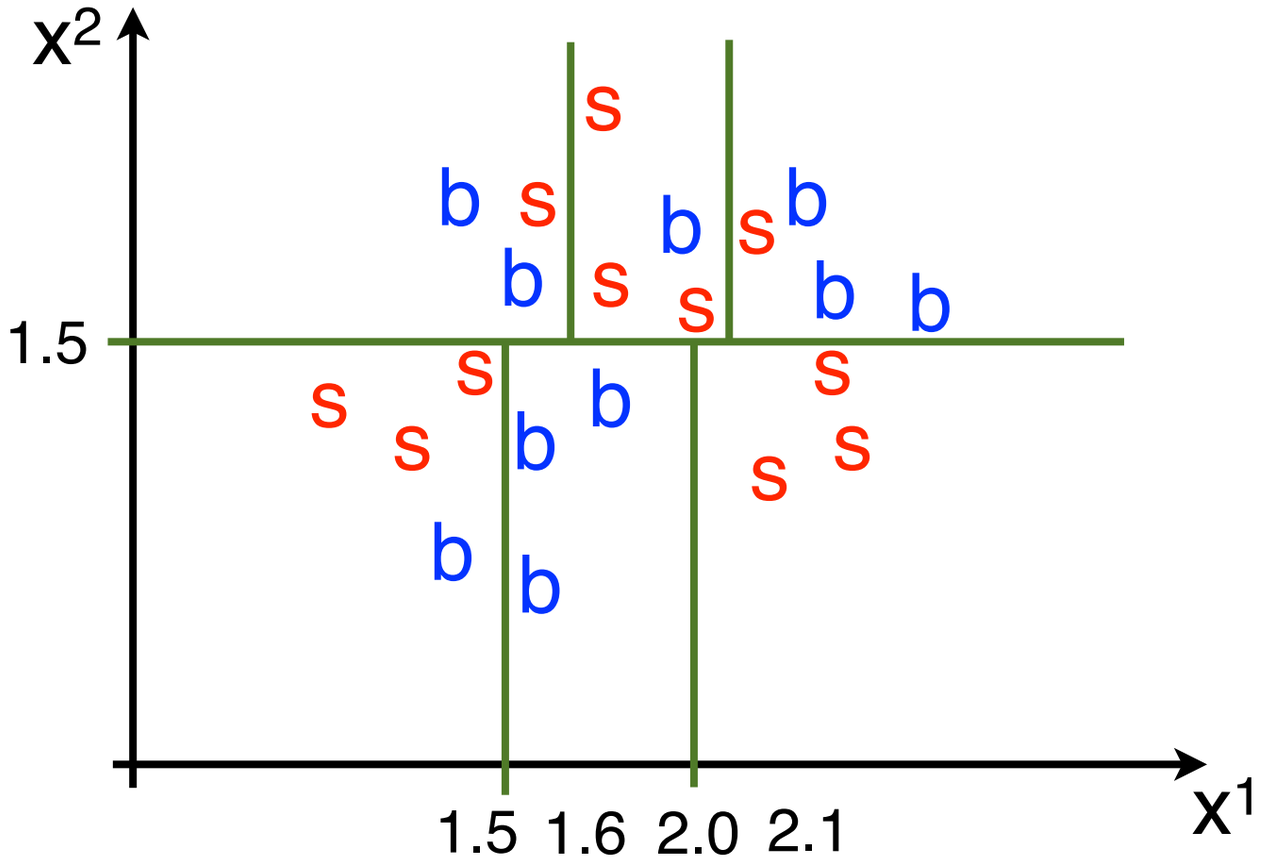


Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \mathfrak{R}^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

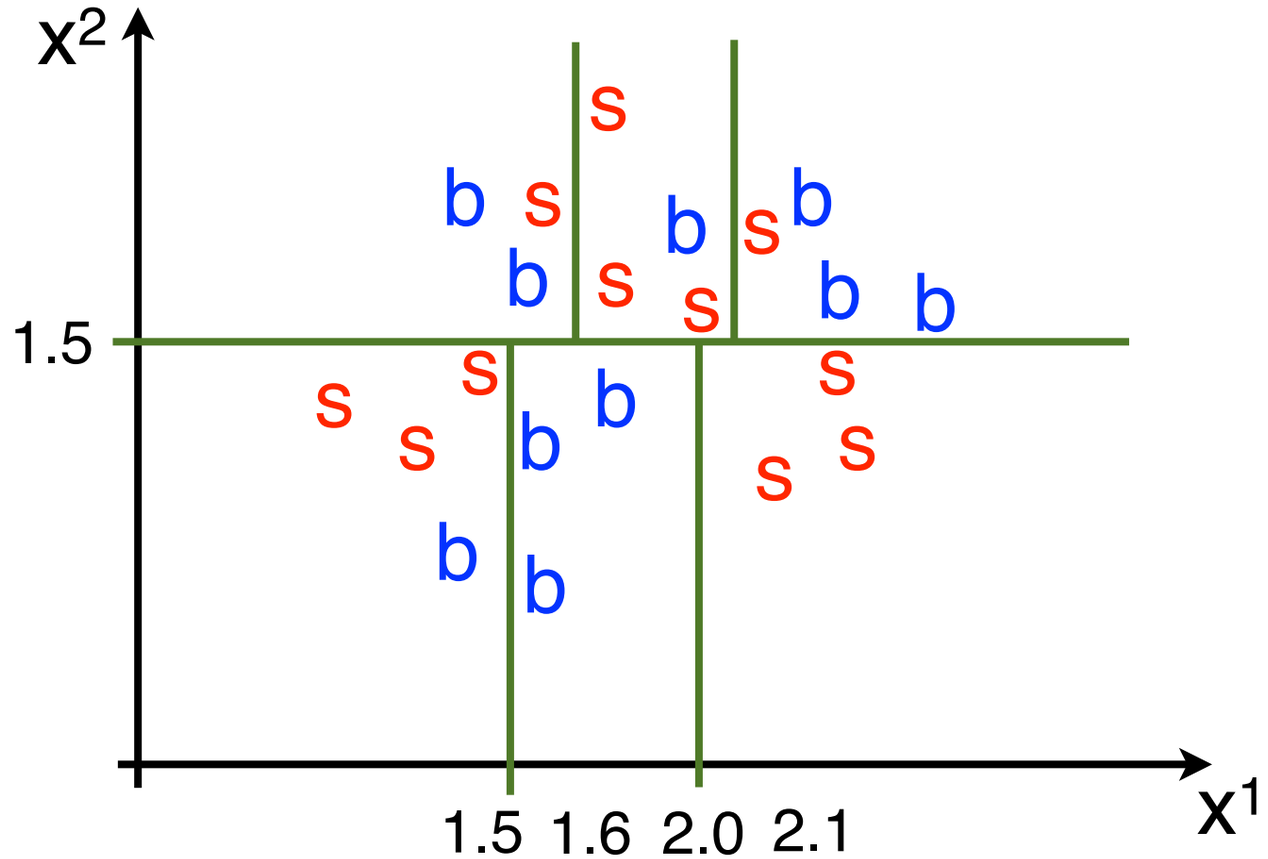


Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s



Repeat until every region (leaf) contains a “minimum” number of points.

Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \Re^2$

(x^1_1, x^2_1) classes

...

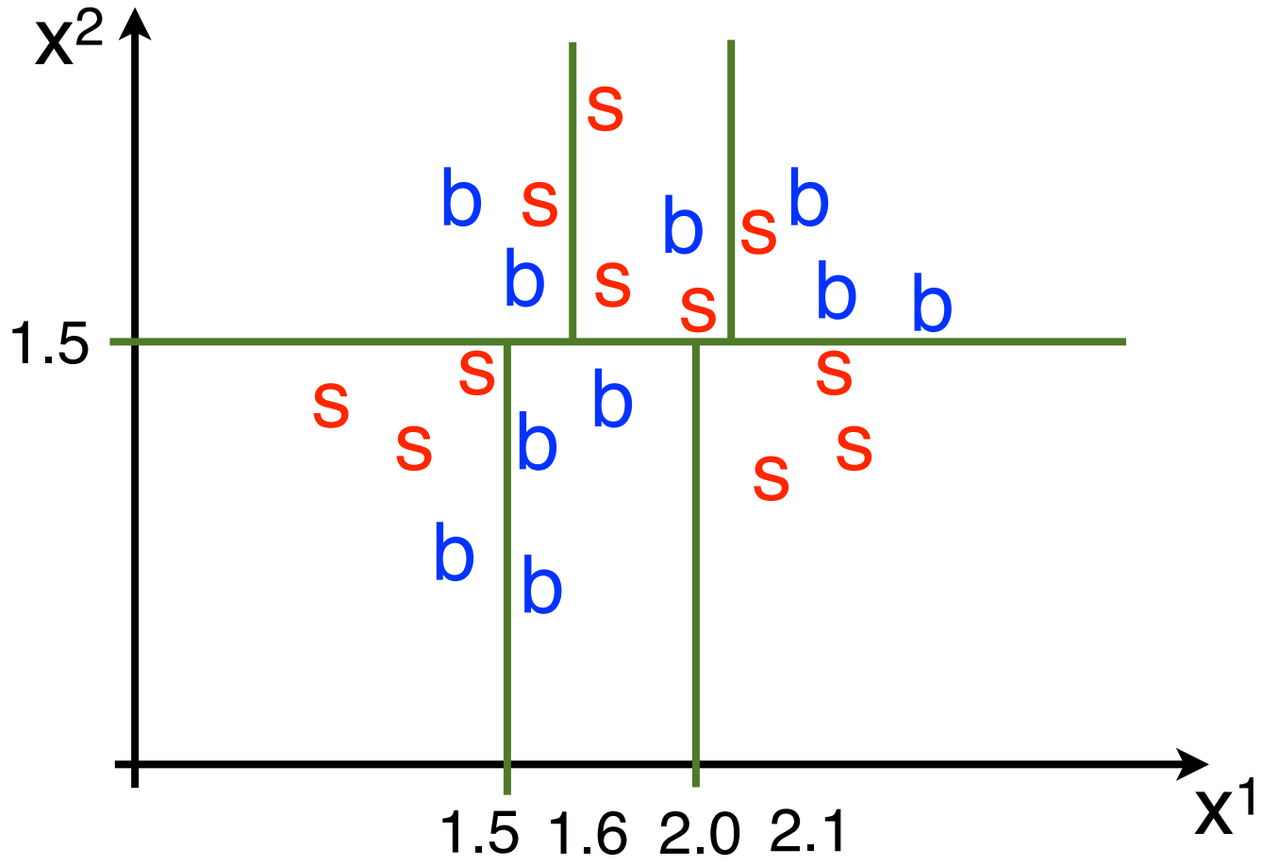
(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s

Build a binary tree:

$x^2 > 1.5$



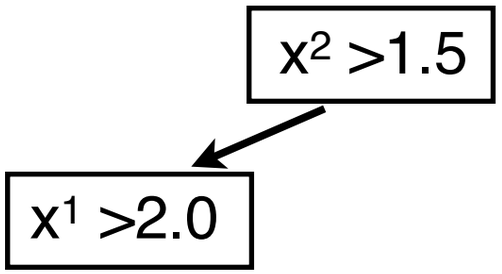
Strategy is to minimize the misclassification at each leaf

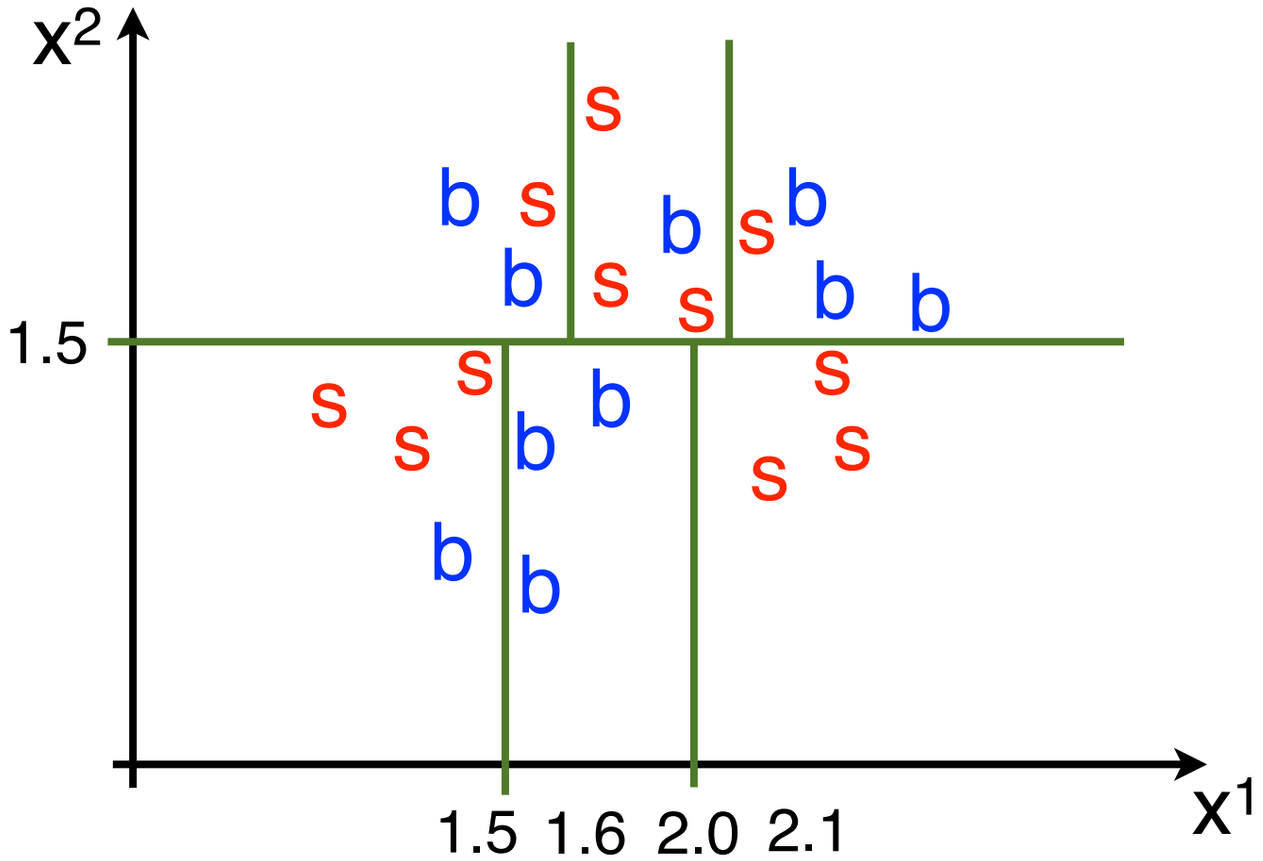
Decision trees: classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

Build a binary tree:





Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \mathfrak{R}^2$

(x^1_1, x^2_1) classes

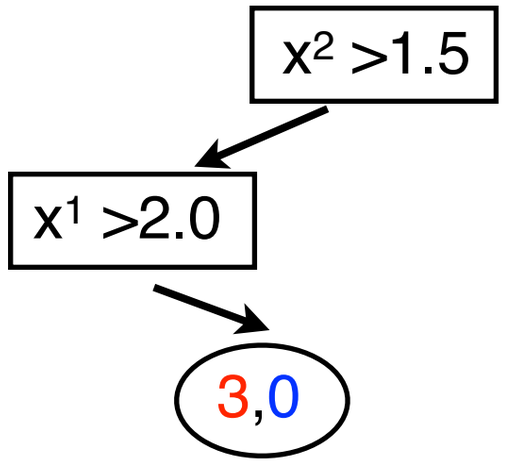
...

(x^1_i, x^2_i) b

...

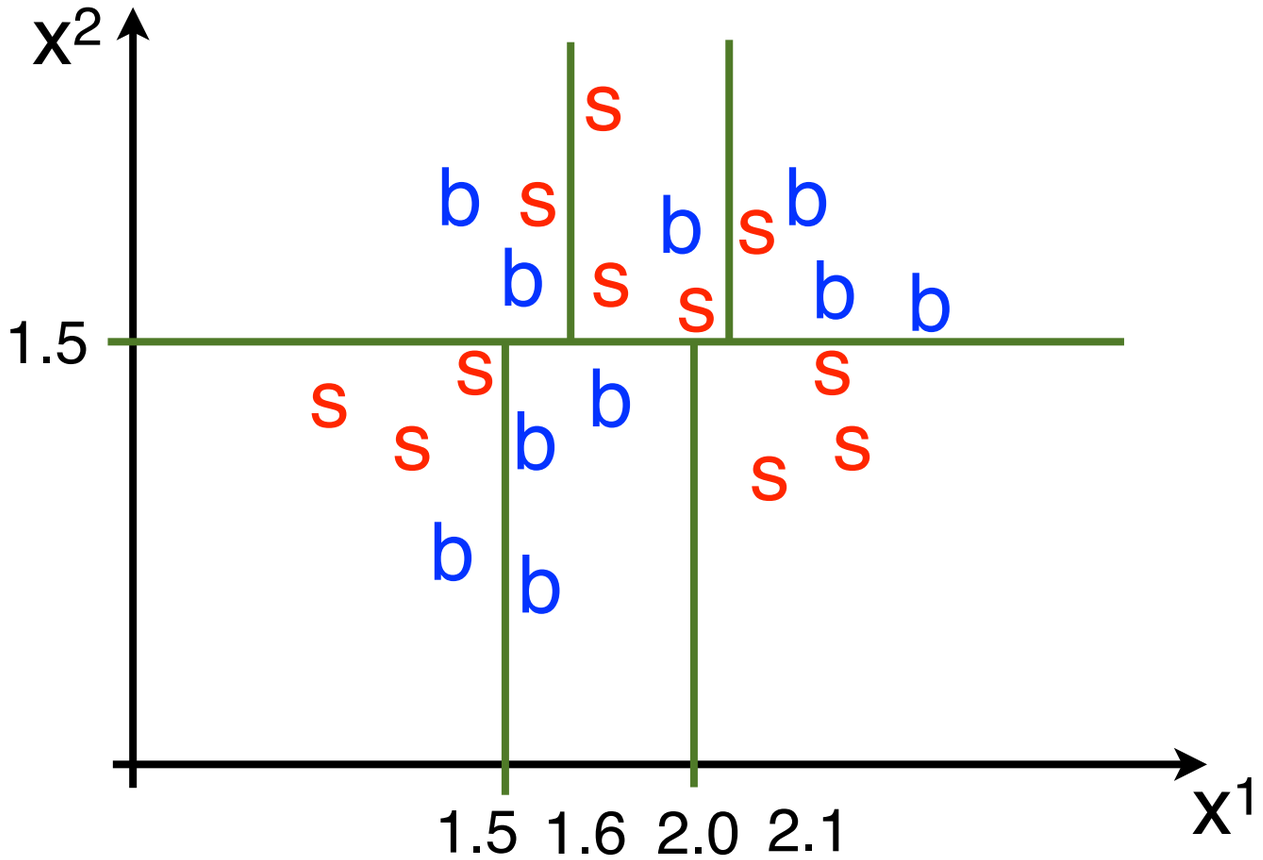
(x^1_n, x^2_n) s

Build a binary tree:



leaf

number of points in each class



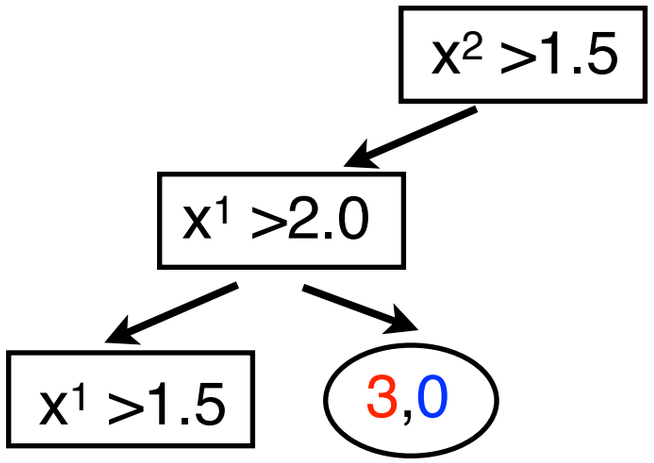
Strategy is to minimize the misclassification at each leaf

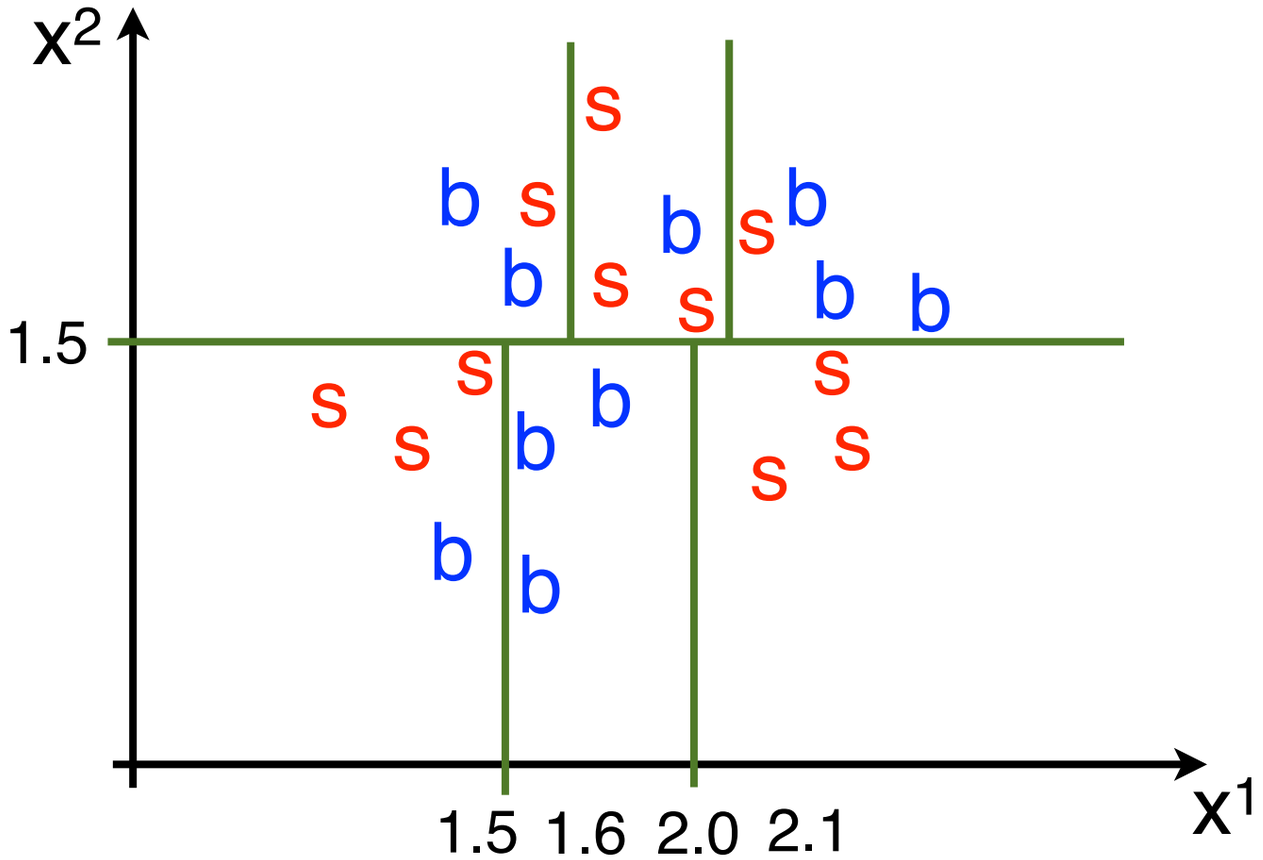
Decision trees: classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

Build a binary tree:





Strategy is to minimize the misclassification at each leaf

Decision trees: classification

Training sample $\in \Re^2$

(x^1_1, x^2_1) classes

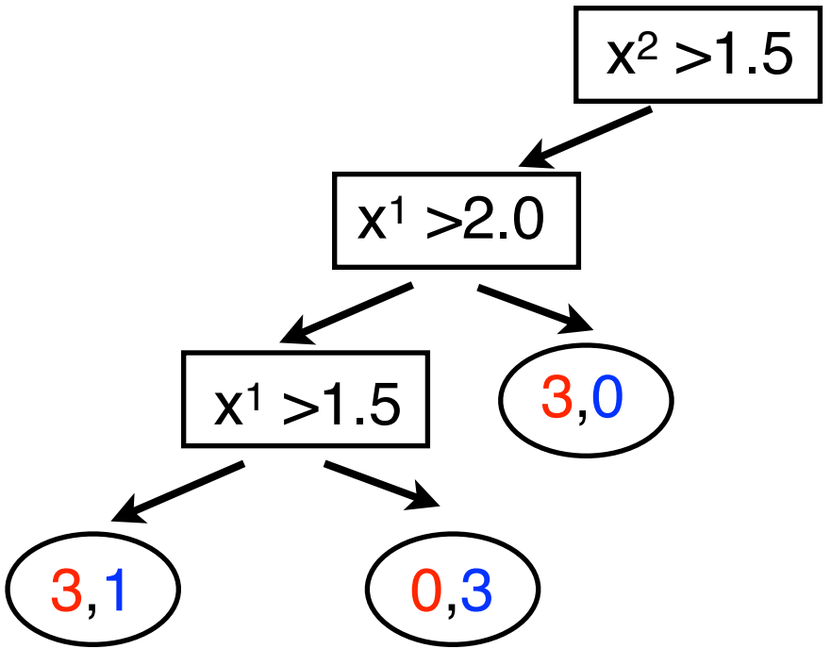
...

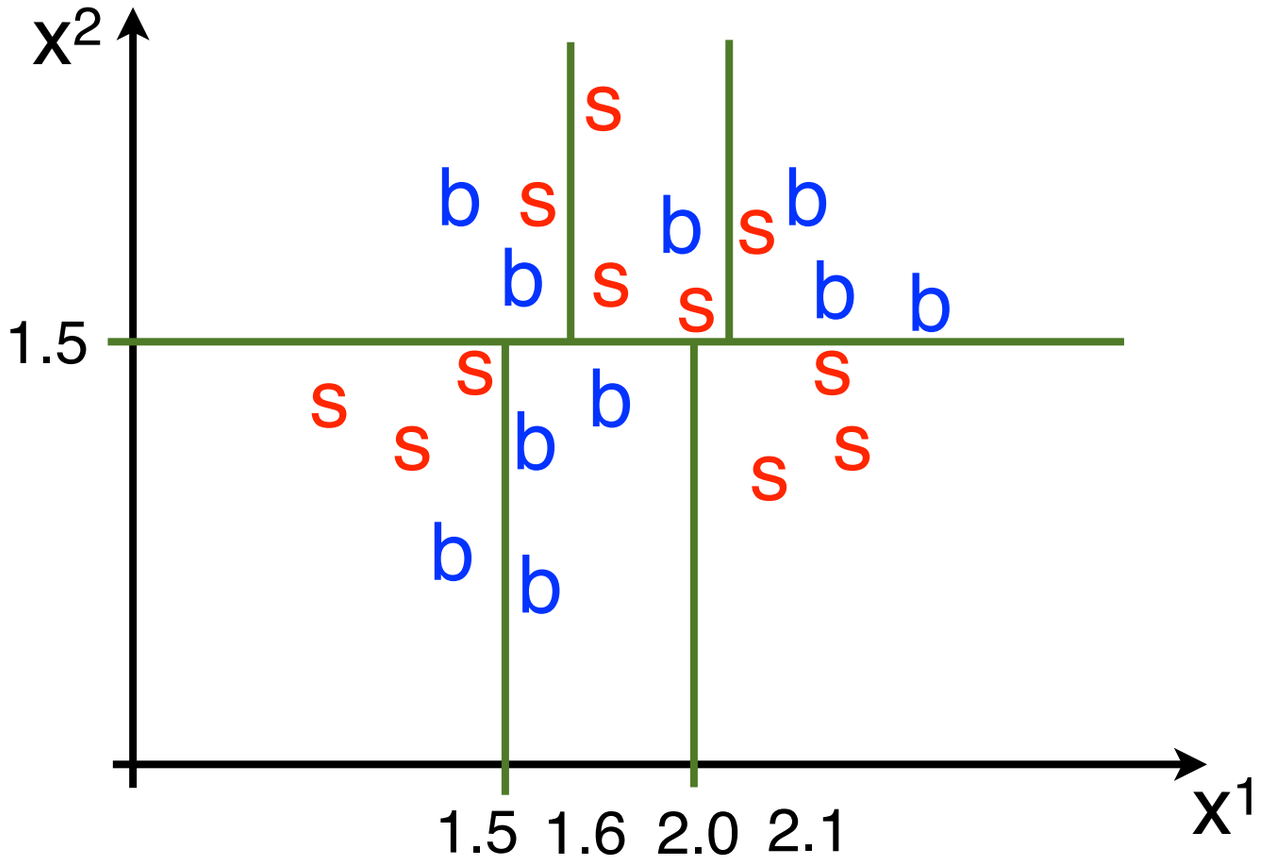
(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s

Build a binary tree:





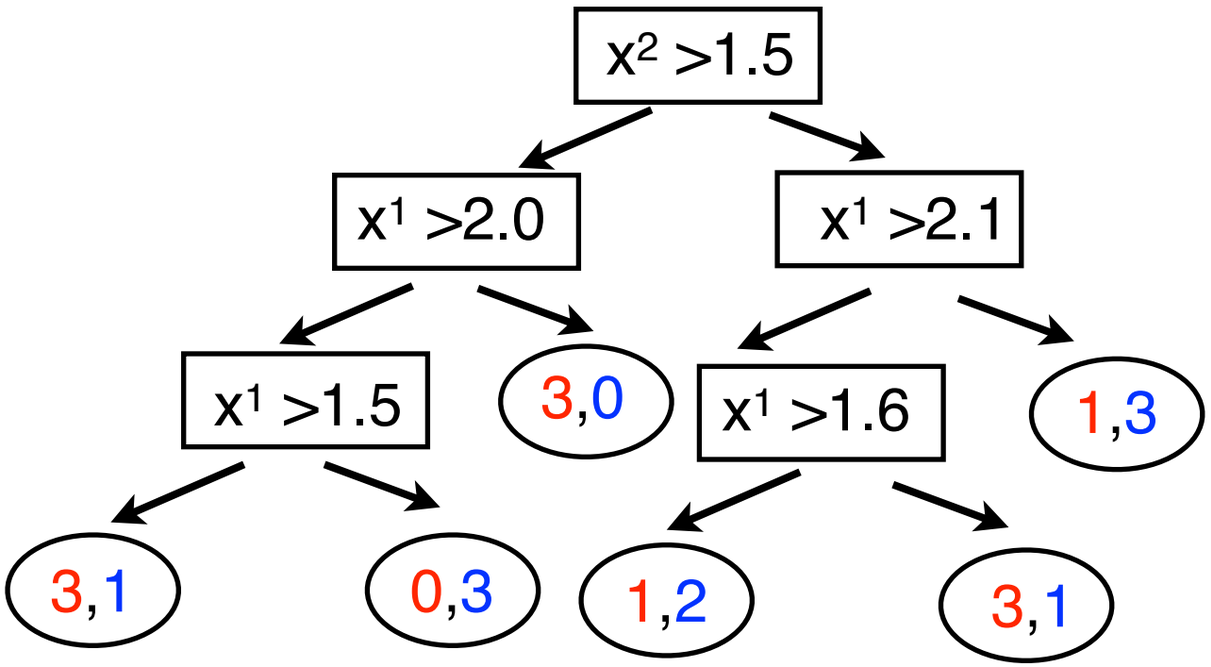
Strategy is to minimize the misclassification at each leaf

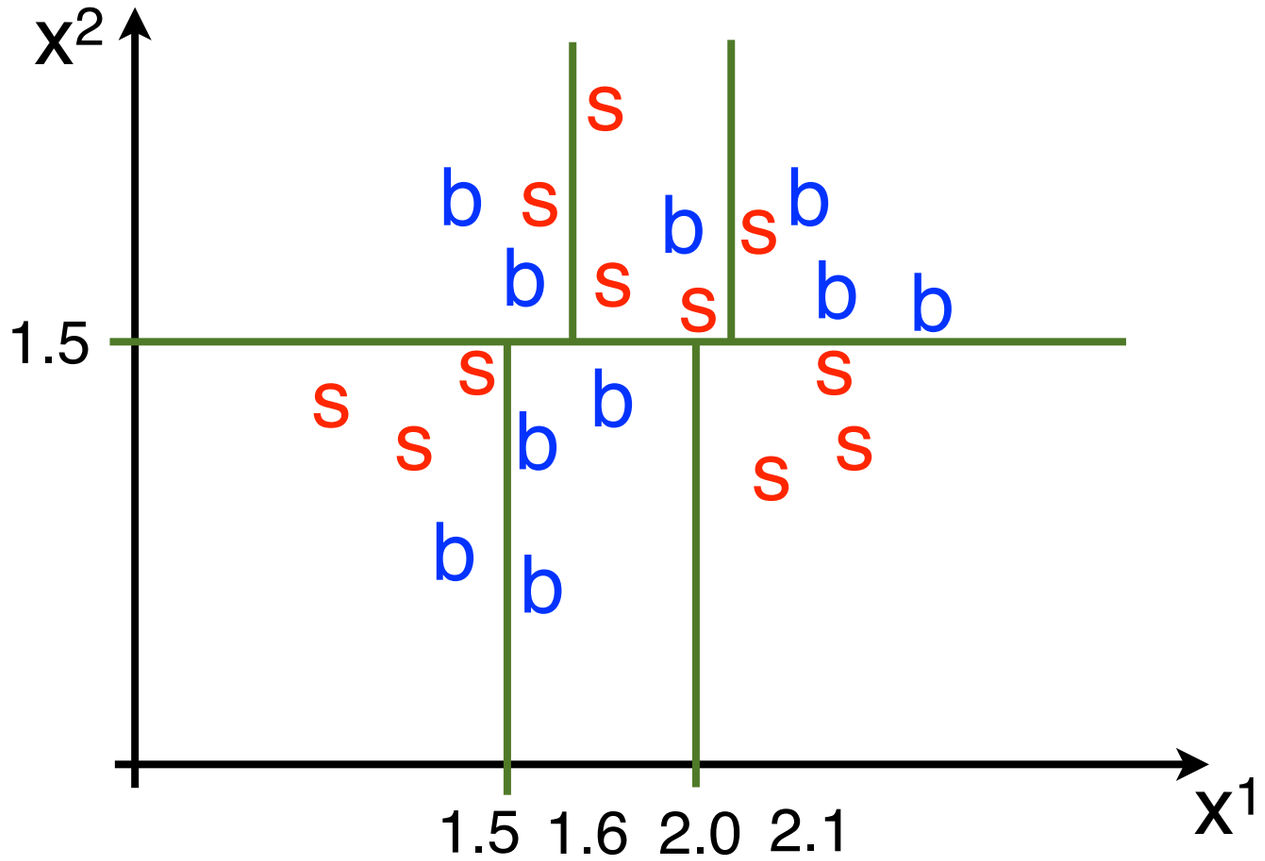
Decision trees: classification

Training sample $\in \Re^2$

(x^1_1, x^2_1)	classes
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

Build a binary tree:





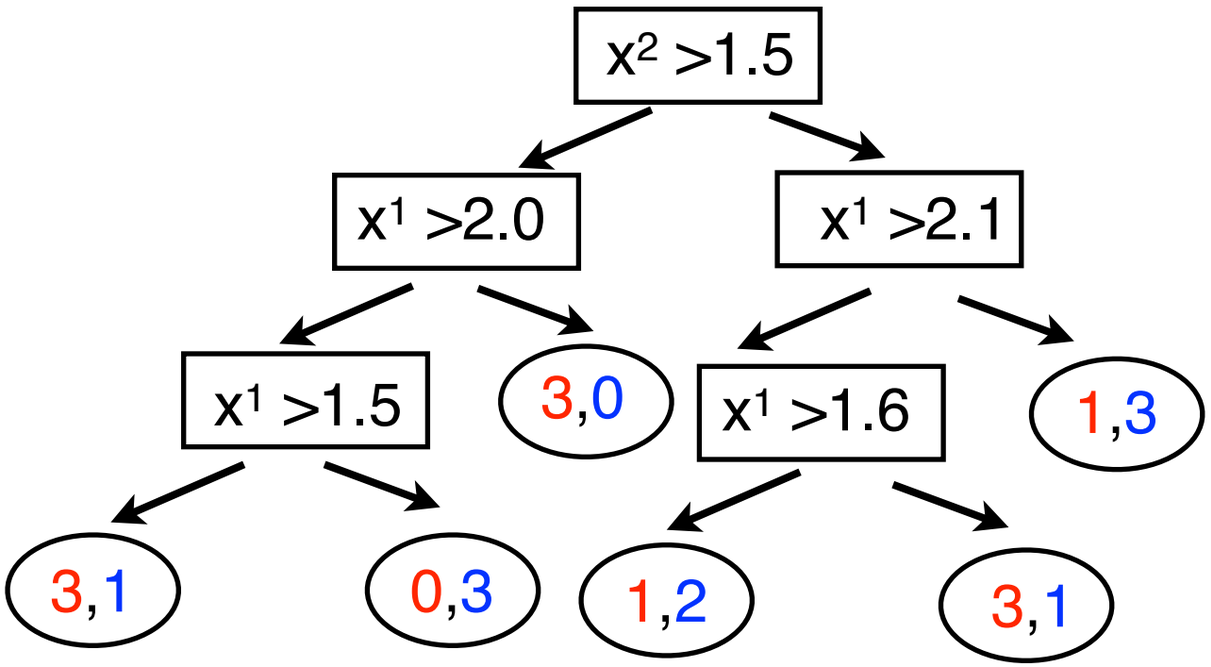
Strategy is to minimize the misclassification at each leaf

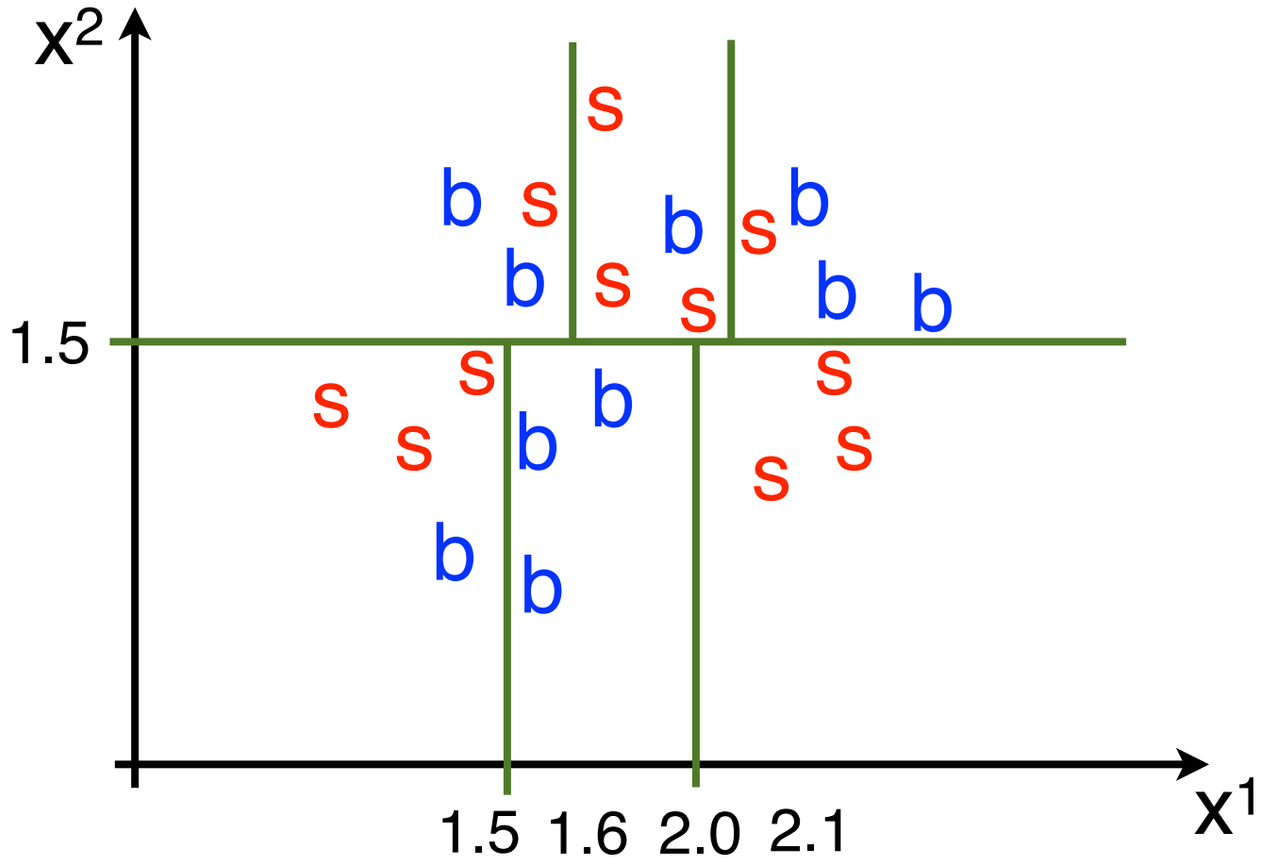
Decision trees: classification

Training sample $\in \Re^2$

	classes
(x^1_1, x^2_1)	
...	
(x^1_i, x^2_i)	b
...	
(x^1_n, x^2_n)	s

Build a binary tree:





Strategy is to minimize the misclassification at each leaf

Now you have to choose how to classify the leaves: Majority vote

Decision trees: classification

Training sample $\in \Re^2$

(x^1_1, x^2_1) classes

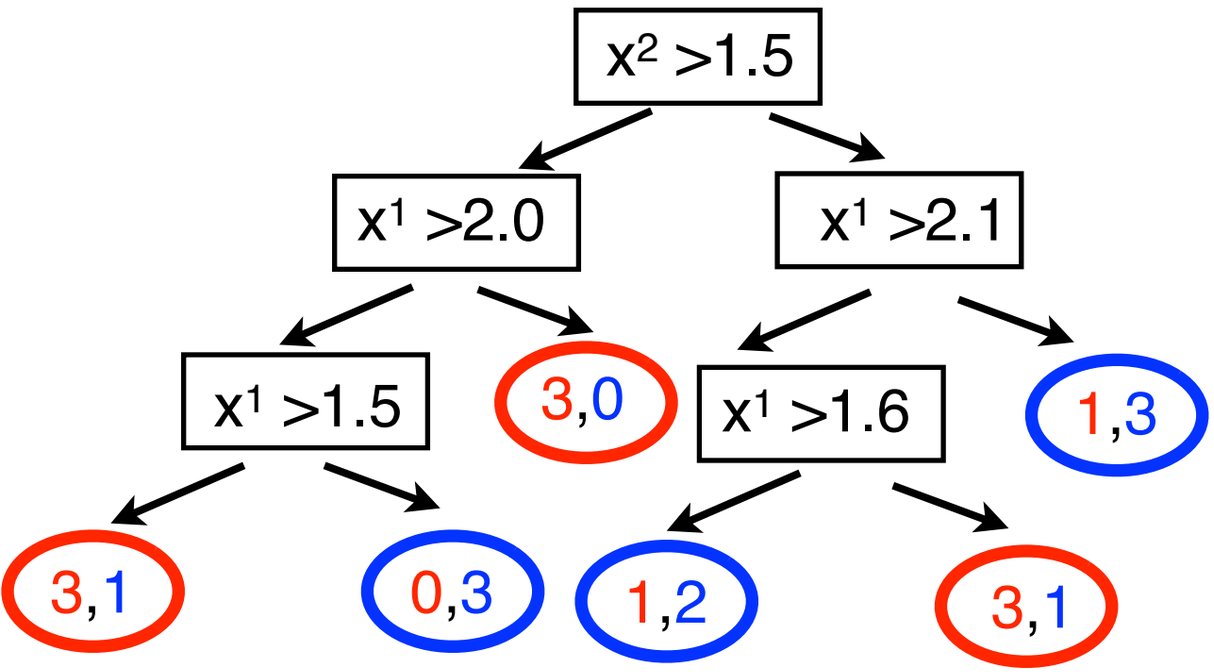
...

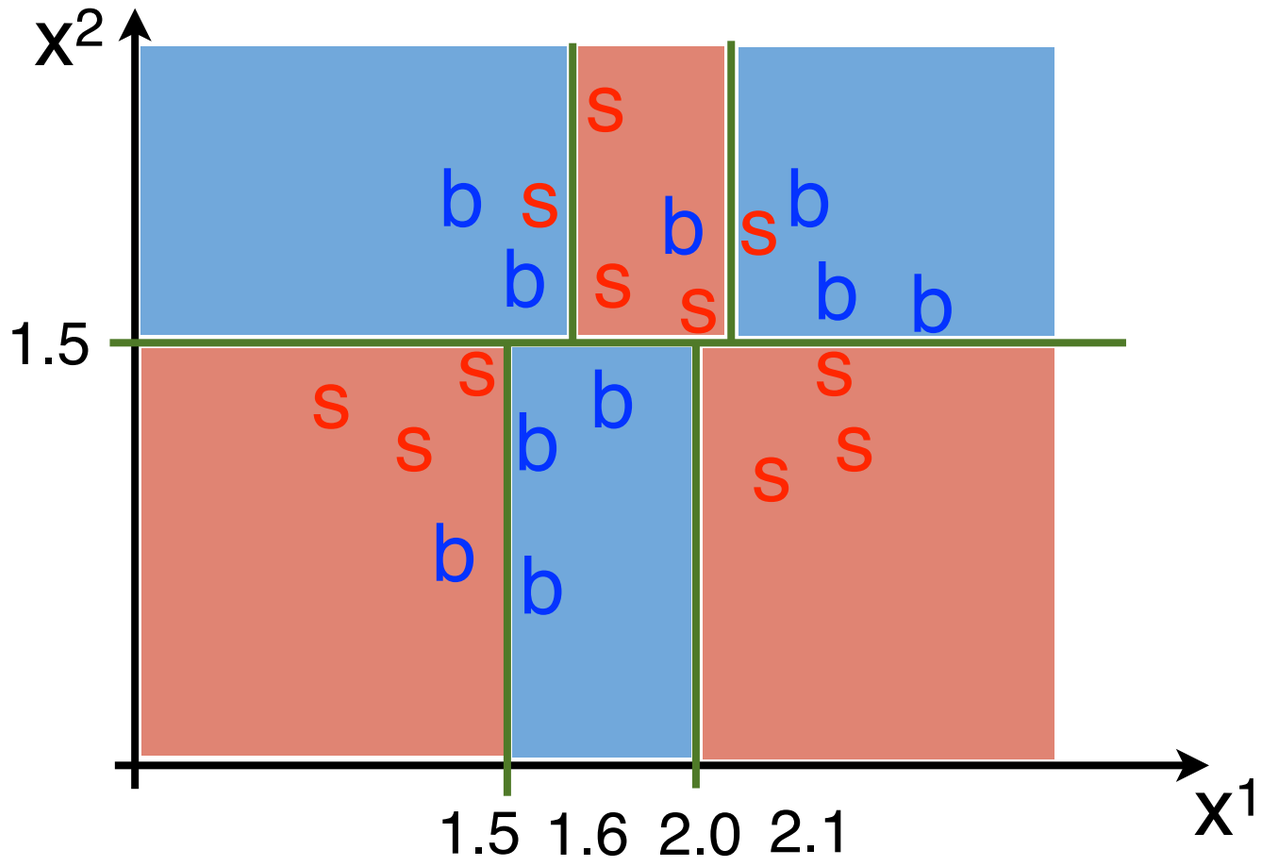
(x^1_i, x^2_i) b

...

(x^1_n, x^2_n) s

Build a binary tree:





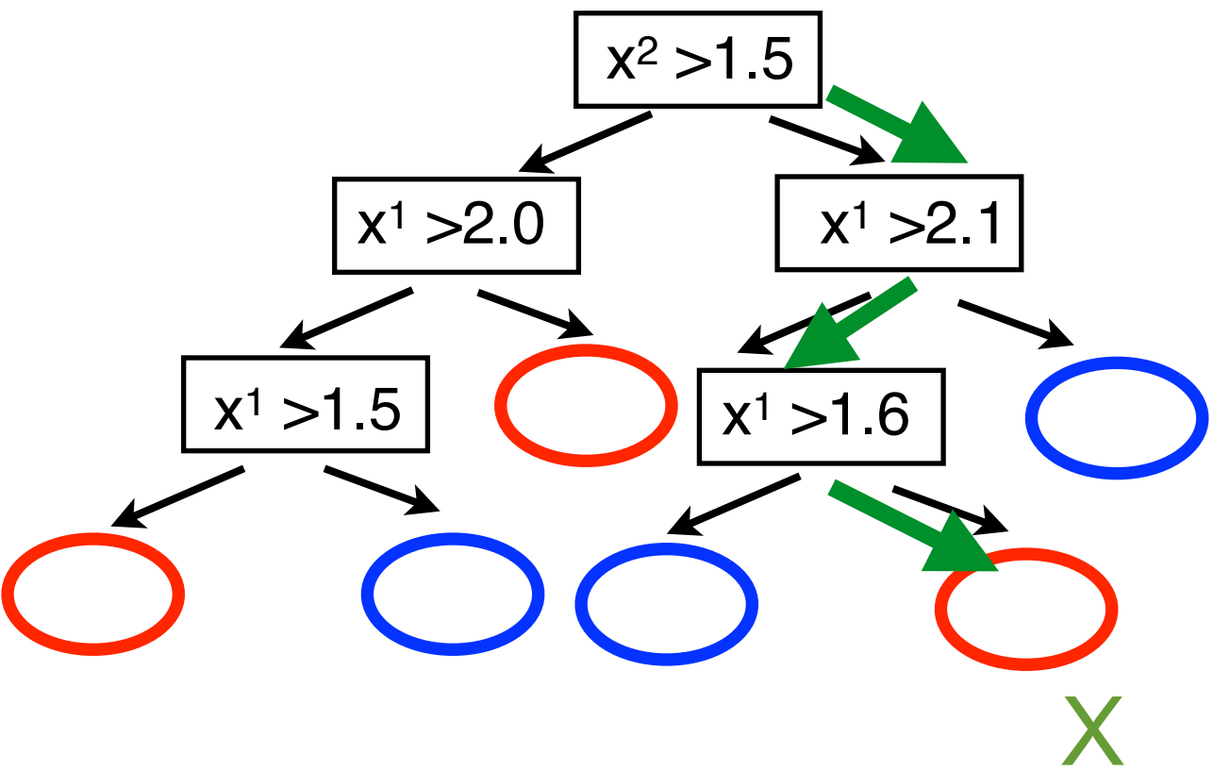
Strategy is to minimize the misclassification at each leaf

Now you have to choose how to classify the leaves: Majority vote

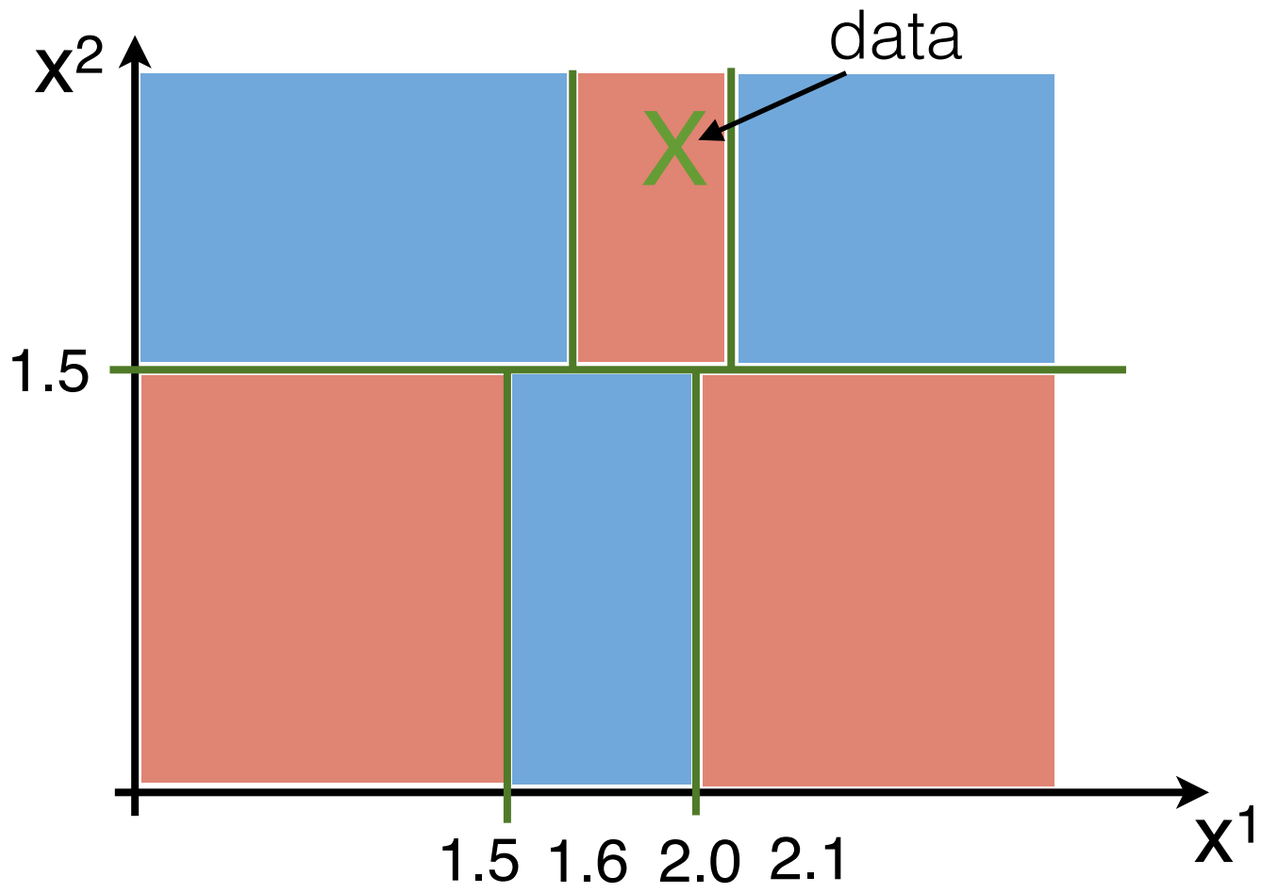
It's like writing a function piece wise constant over the regions

Decision trees: classification

Apply the DT to your data set
(x^1_1, x^2_1)
...
(x^1_i, x^2_i)
...
(x^1_n, x^2_n)



is classified as S



Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

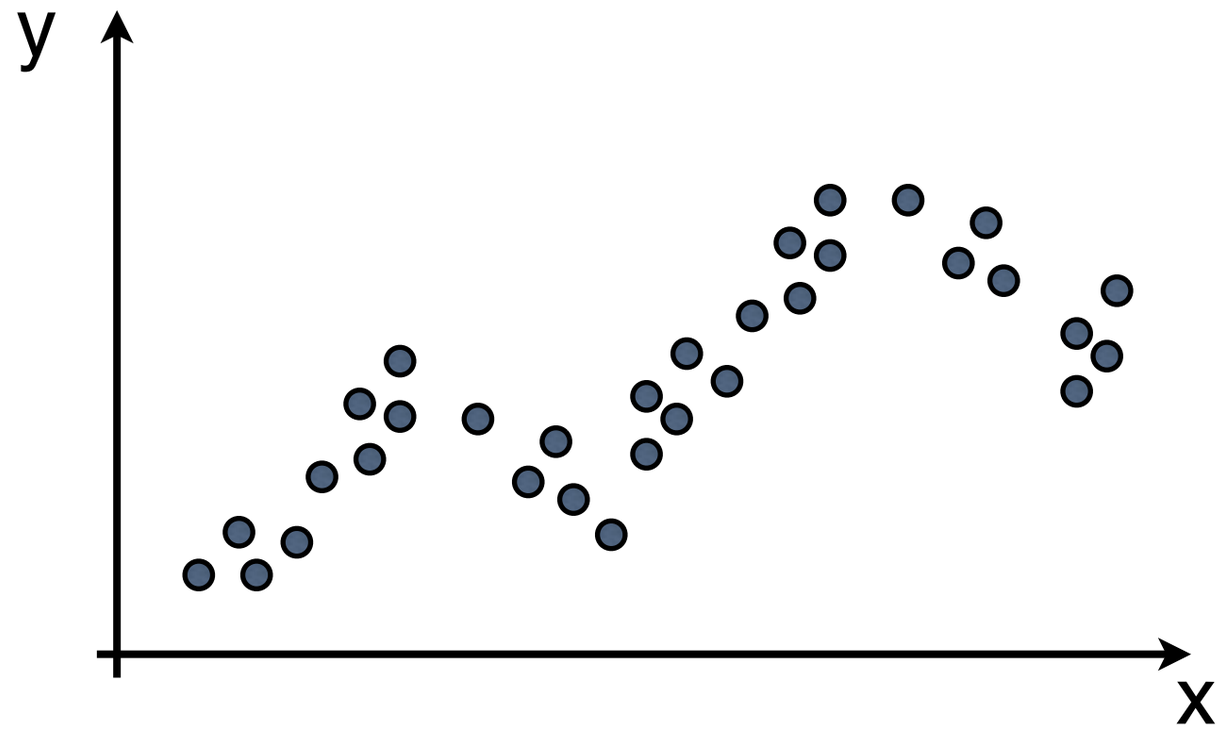
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

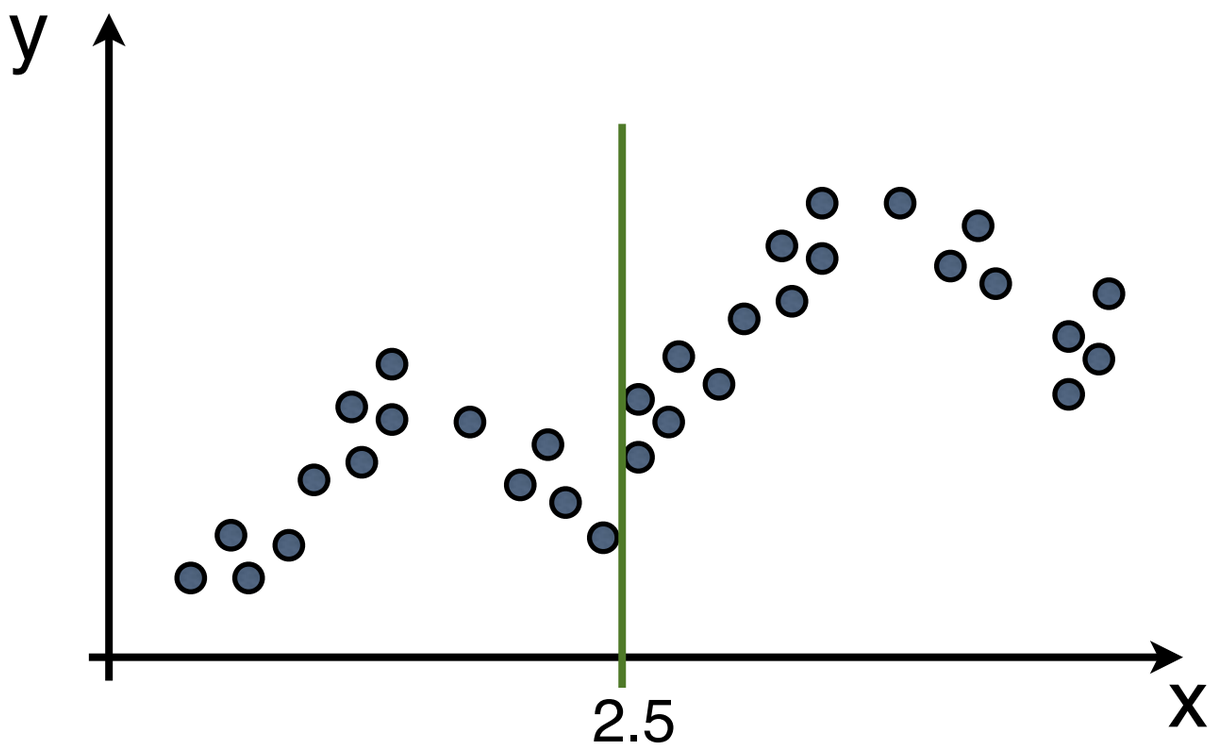
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

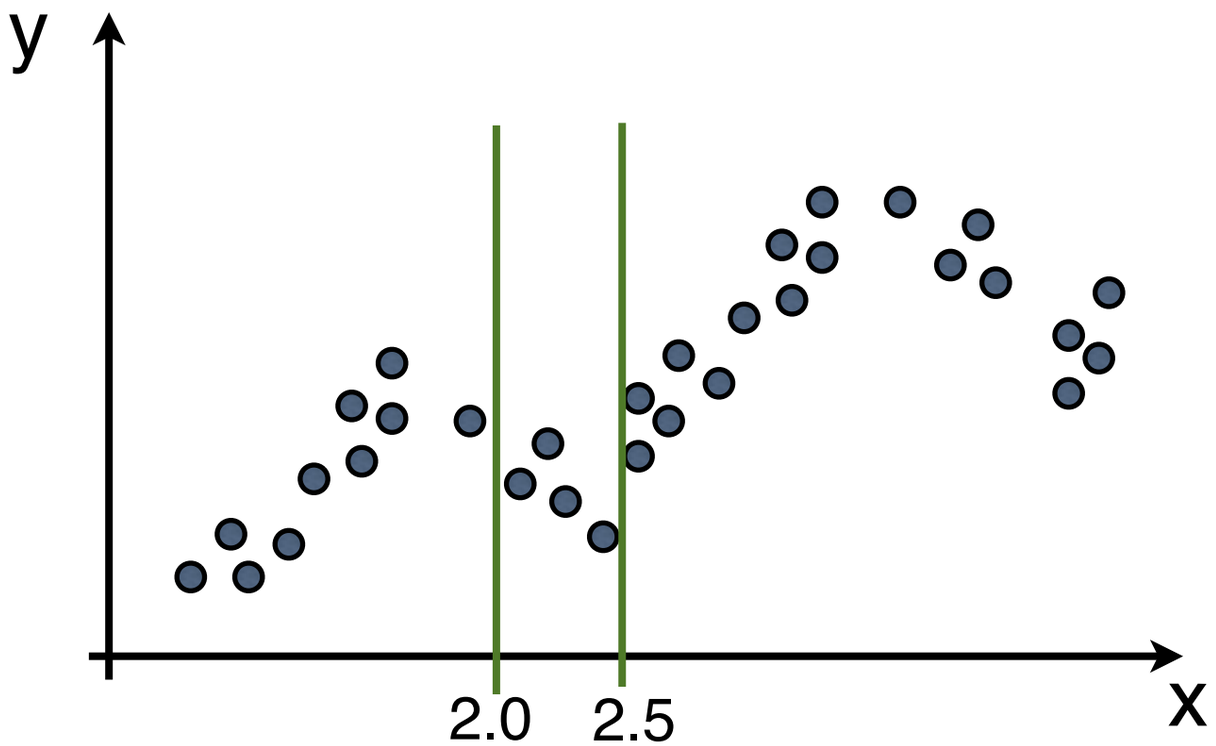
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

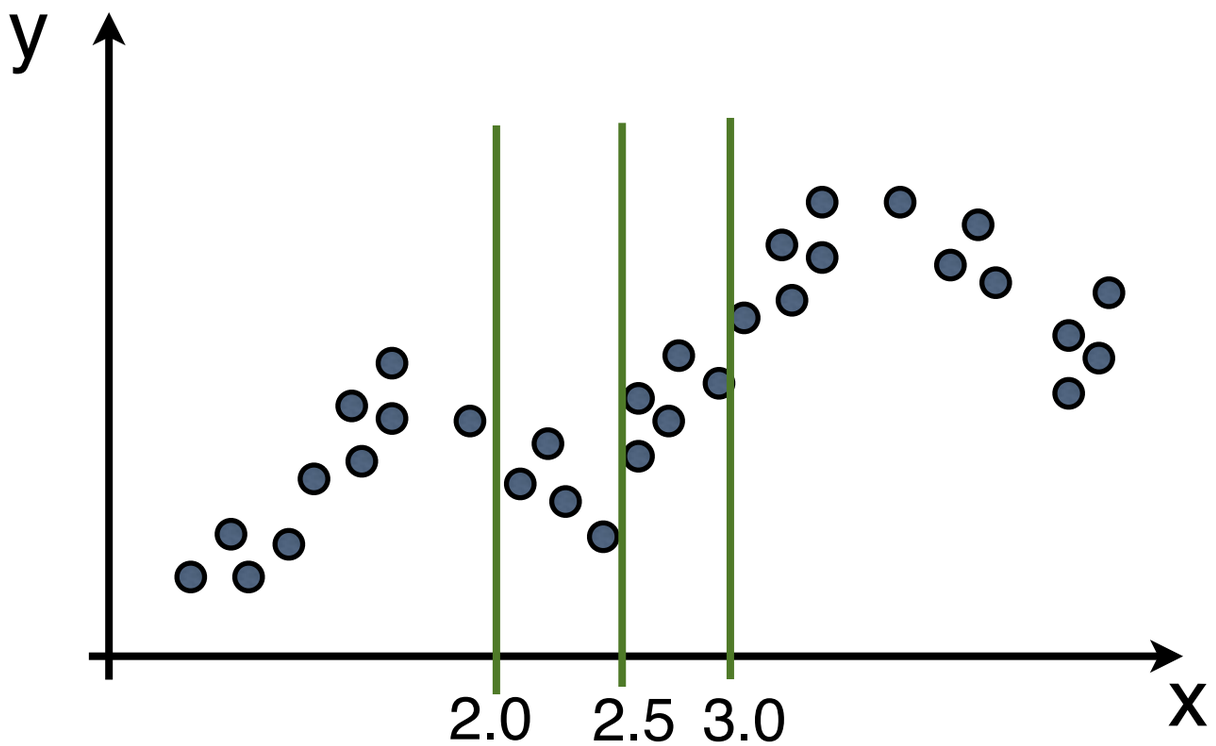
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

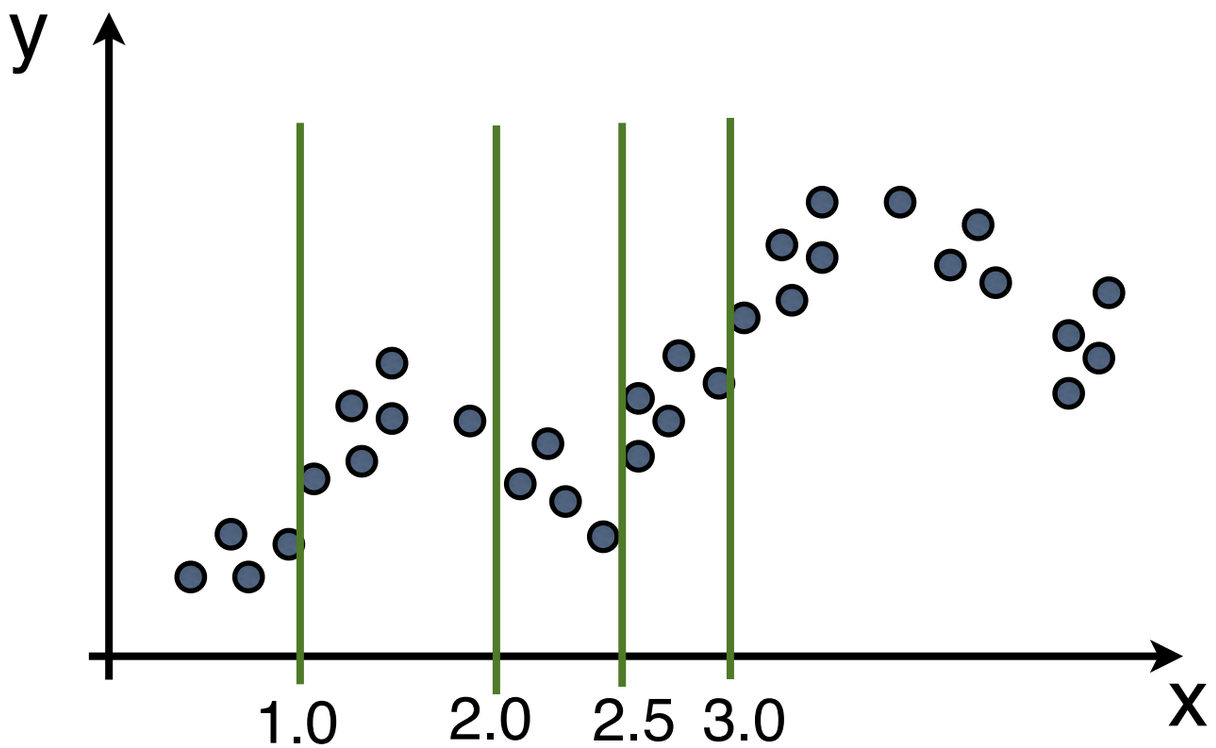
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

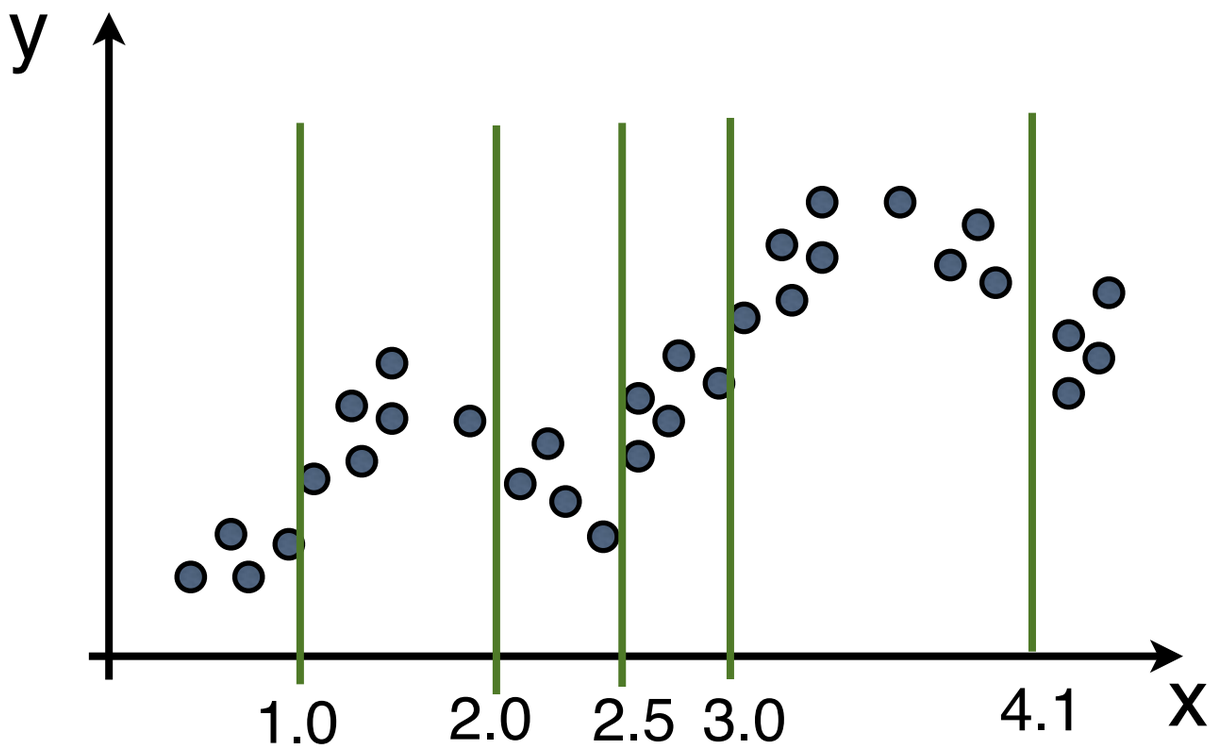
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

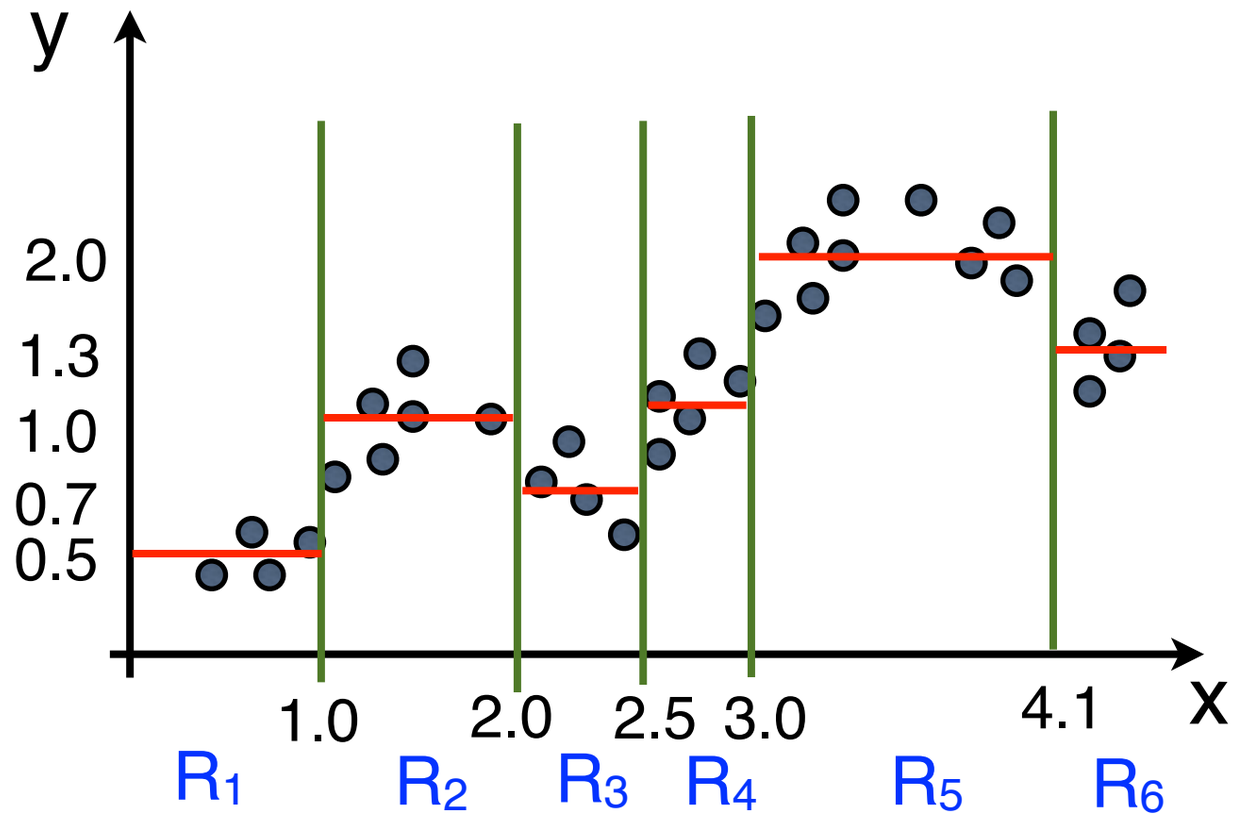
...

(x_i, y_i)

...

(x_n, y_n)

continuous target



Strategy is to minimize the error at each leaf

Repeat until every region (leaf) contains a “minimum” number of points

Average of the points in each region

i.e. given x predict y

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

...

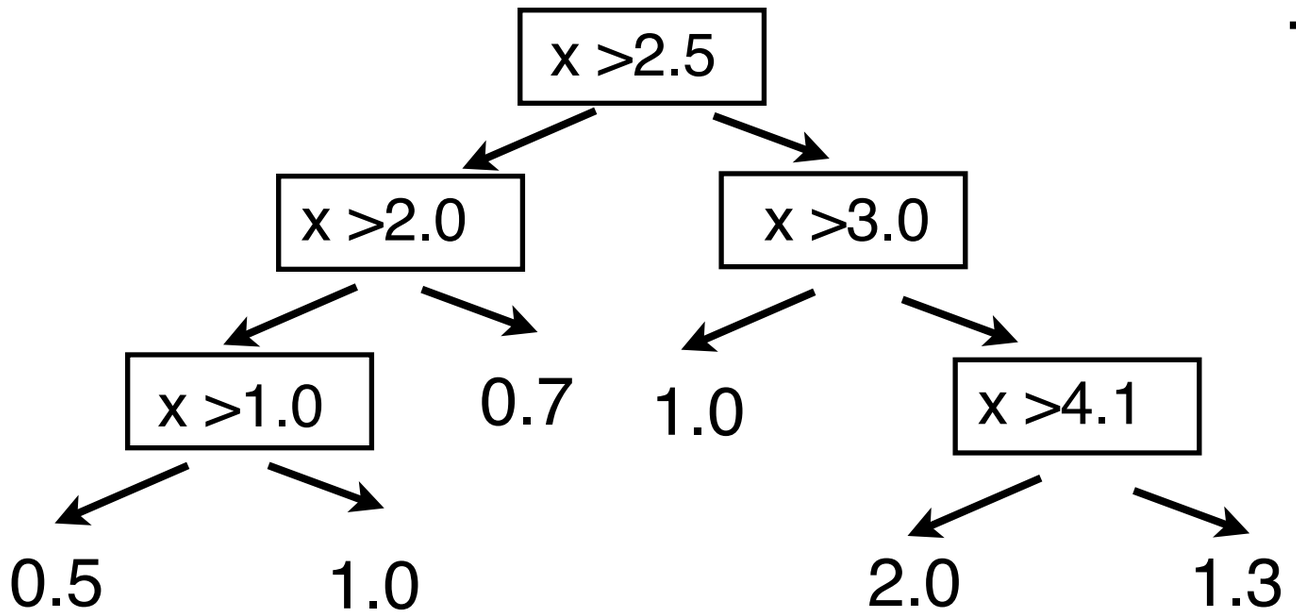
(x_i, y_i)

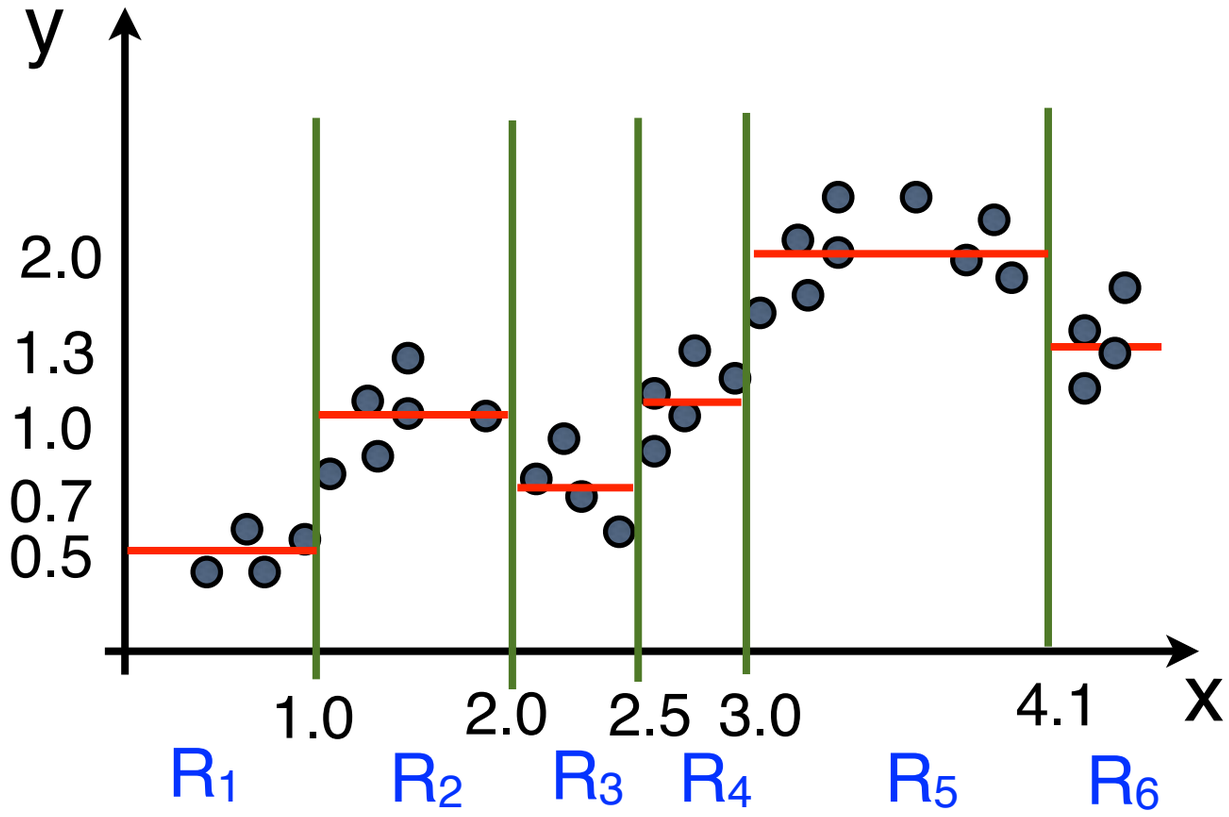
...

(x_n, y_n)

continuous target

Build a binary tree





Strategy is to minimize the error at each leaf

Average of the points in each region

Decision trees: regression

Training sample $\in \mathfrak{R}$

(x_1, y_1)

...

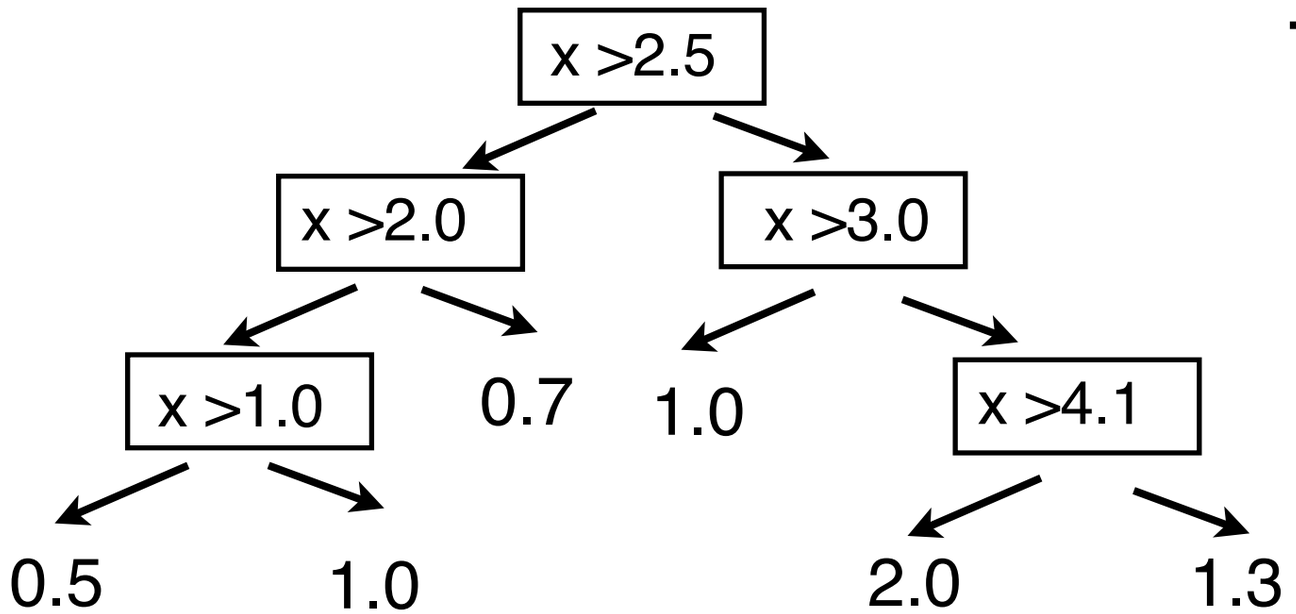
(x_i, y_i)

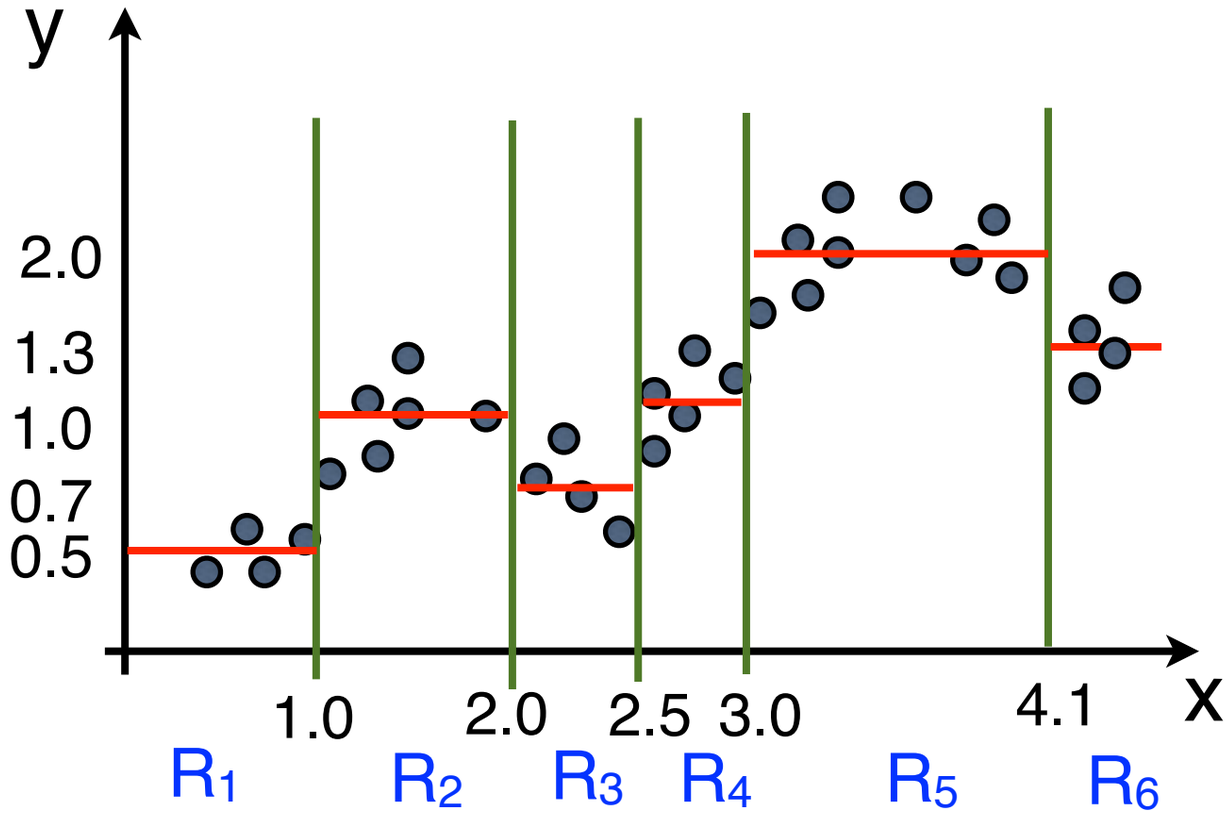
...

(x_n, y_n)

continuous target

Build a binary tree





Strategy is to minimize the error at each leaf

Average of the points in each region

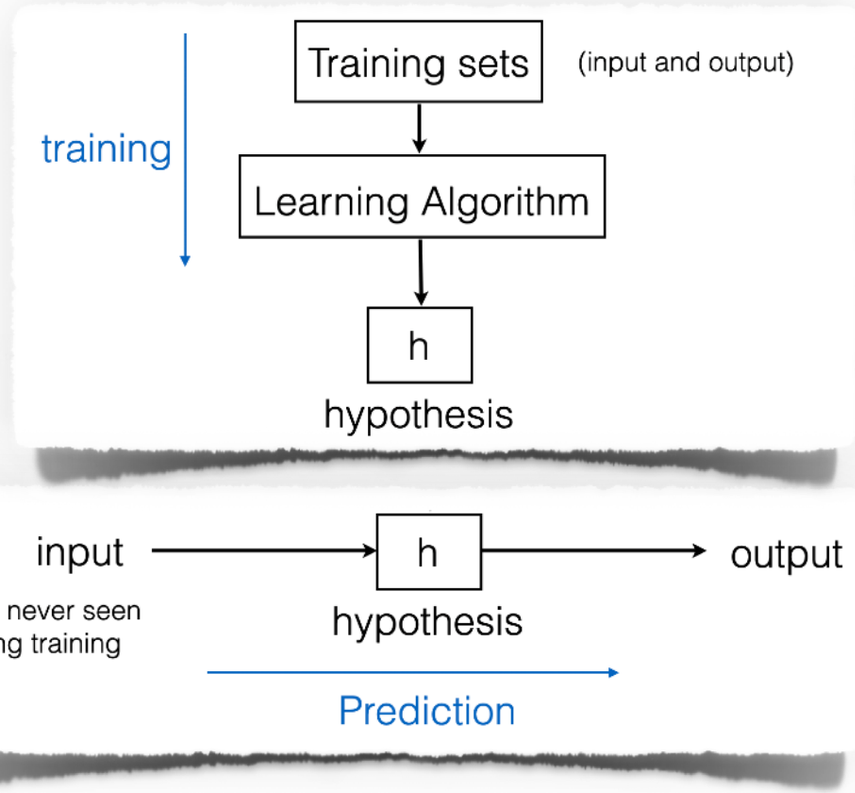
Again it's like writing a real function piece wise constant over the variables' space

Bad news: decision trees are not usable…

Decision Trees tend to be very sensitive to statistical fluctuations of the training sample:

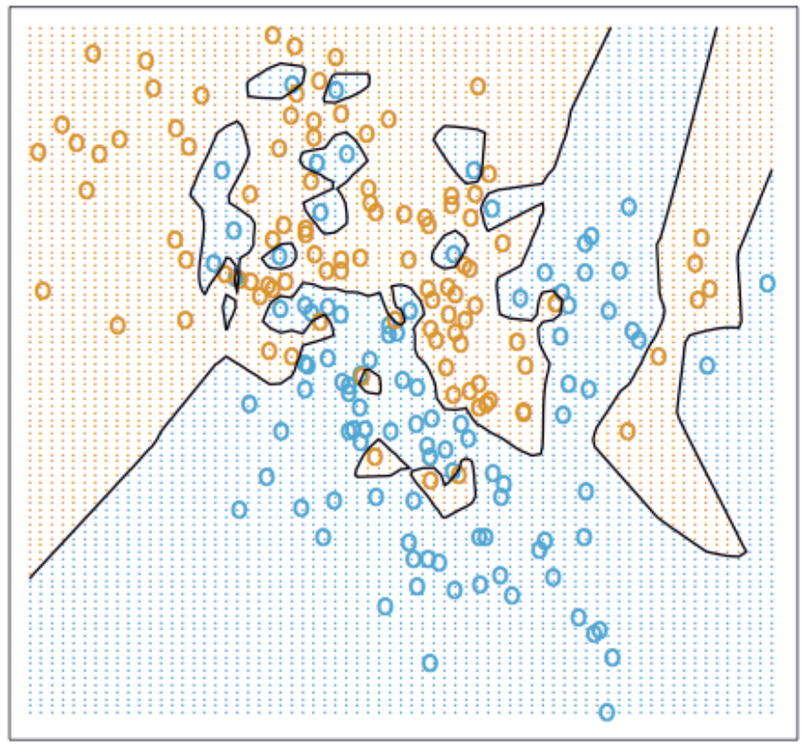
1- The variables and the order are chosen on the base of separation. If you change the training sample you might get different trees.

2- Whatever variable is the most discriminating it will influence the rest of the tree !



3- They are very sensitive to overtraining: learn the noise of the particular sample used for training, but miss the general structure. Poor performance when applied on another sample.

Decision trees are too unstable to be used safely.



Decision trees are not usable…

A simple way to make a decision tree more stable is to "re-group" or remove branches (regions) of your tree: pruning

This is a special case of a general process called "regularisation"

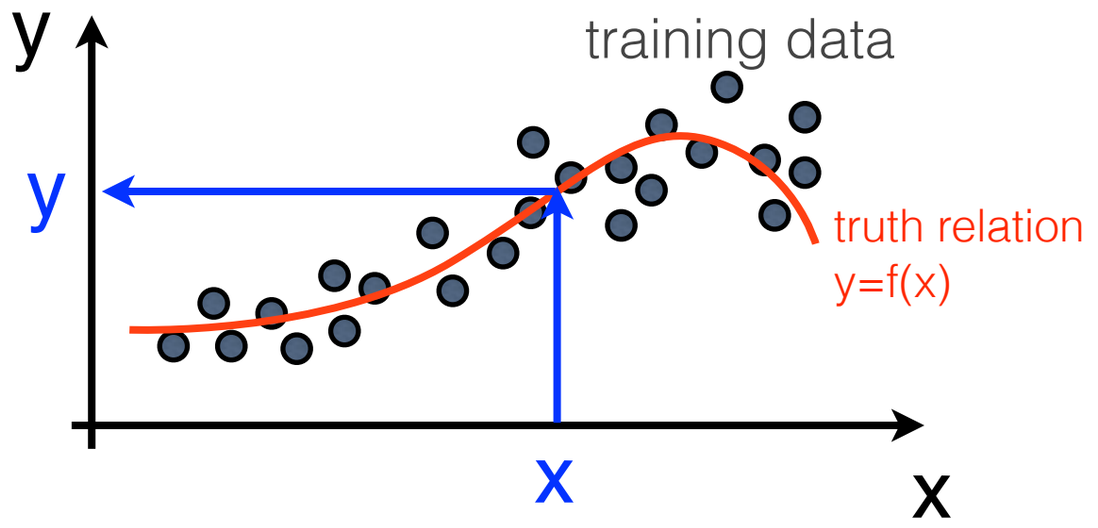
The breakthrough came with aggregation techniques: aggregate copies of the same (or similar) tree

Among the two most used are BAGGING and BOOSTING.

These techniques can be applied to classification and regression (and to any kind of algorithm not only DT).

Bagging: Bootstrapping AGGregation

Consider a regression problem:
(given x predict y)



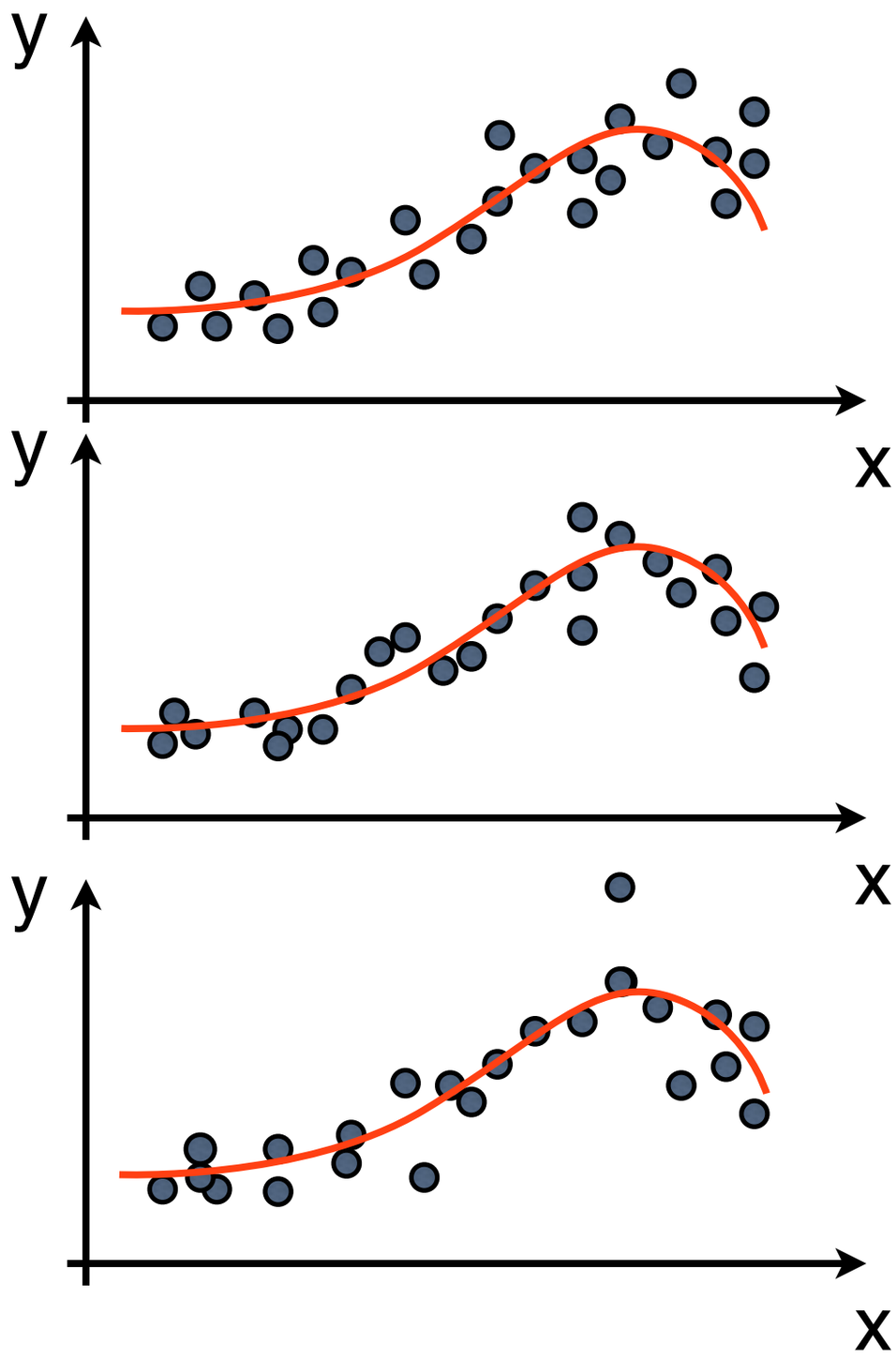
Bagging: Bootstrapping AGGregation

Consider a regression problem:
(given x predict y)

Idea: take N datasets independently drawn from the training sets and improve the prediction by aggregating the trees: averaging

How do we create N datasets ?
resampling

In general the resampling technique used is the bootstrap (hence the name of the technique)

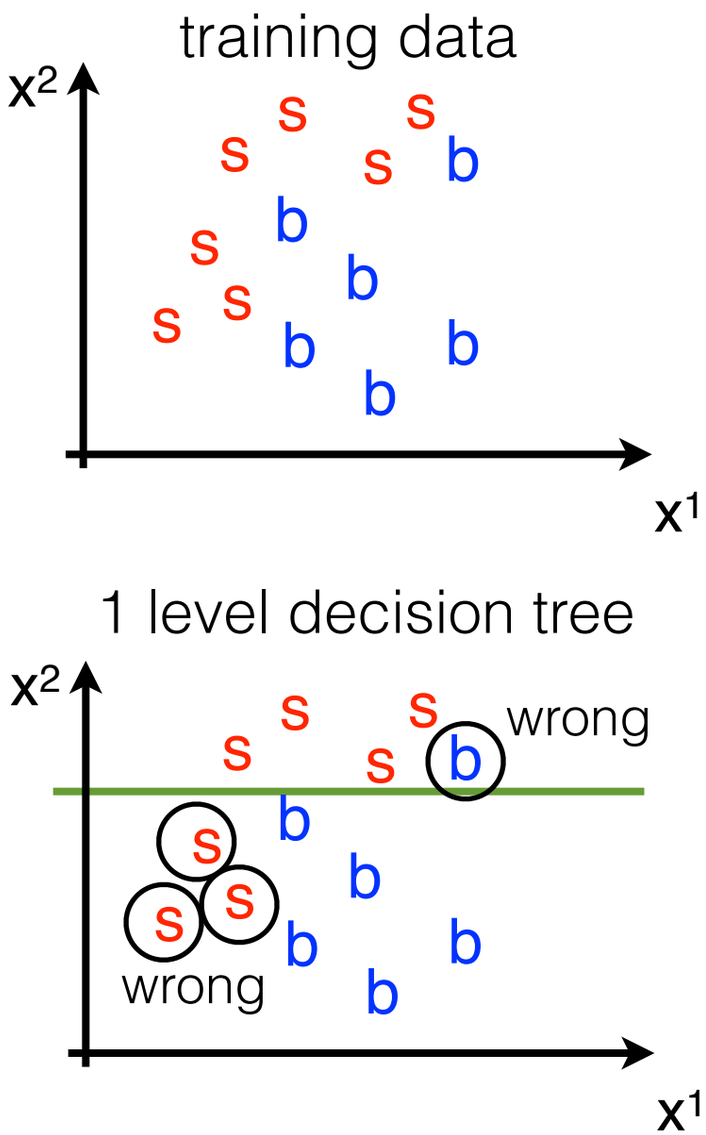


(For a classification you average the prediction of the bootstrapped samples)

[Ihler]

Boosting

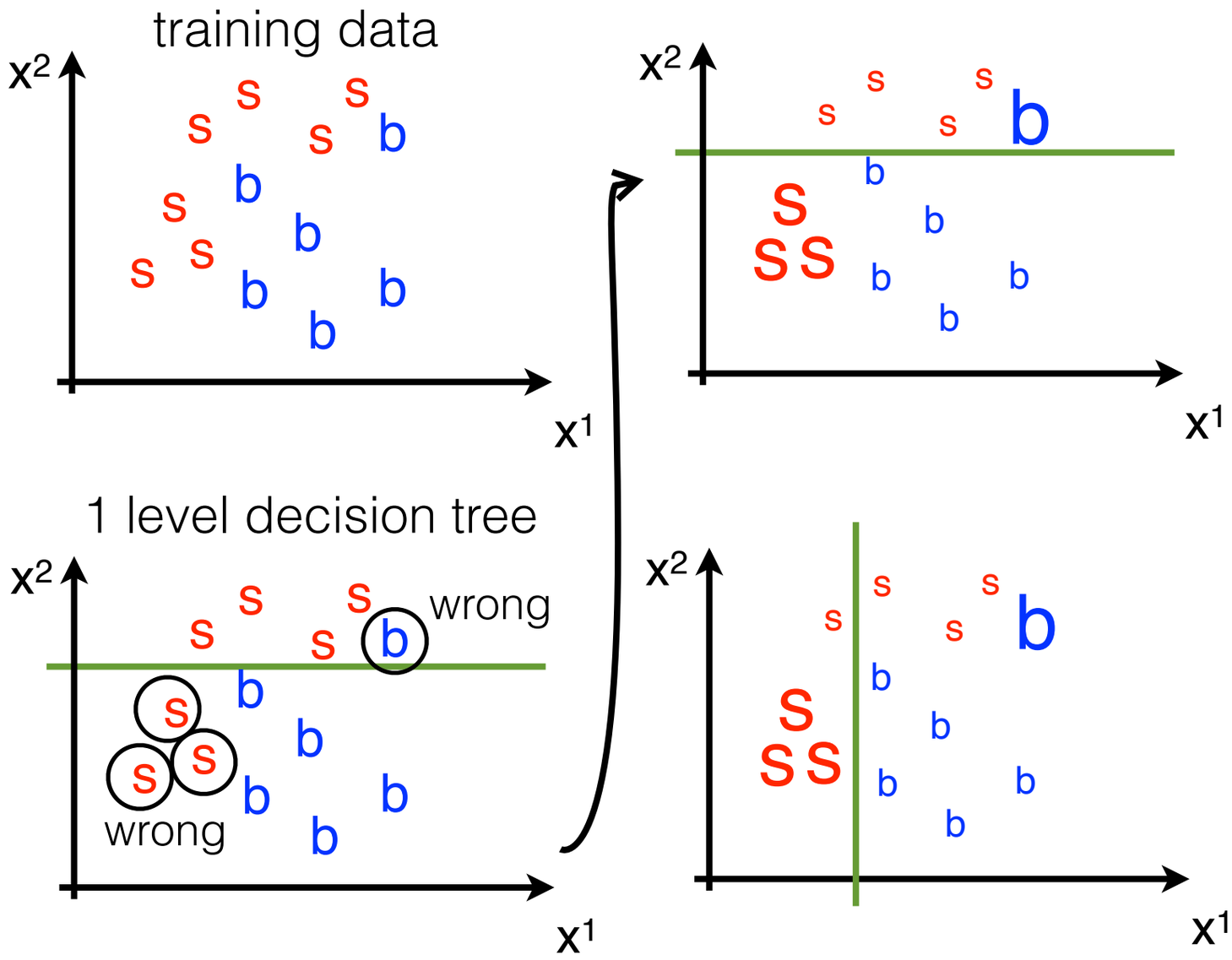
Sequentially training a model learning from the errors of the previous ones. The idea is to create modifications that give smaller error rates than those of the preceding classifiers.



Focus on the 4 wrong ones

Boosting

Sequentially training a model learning from the errors of the previous ones. The idea is to create modifications that give smaller error rates than those of the preceding classifiers.



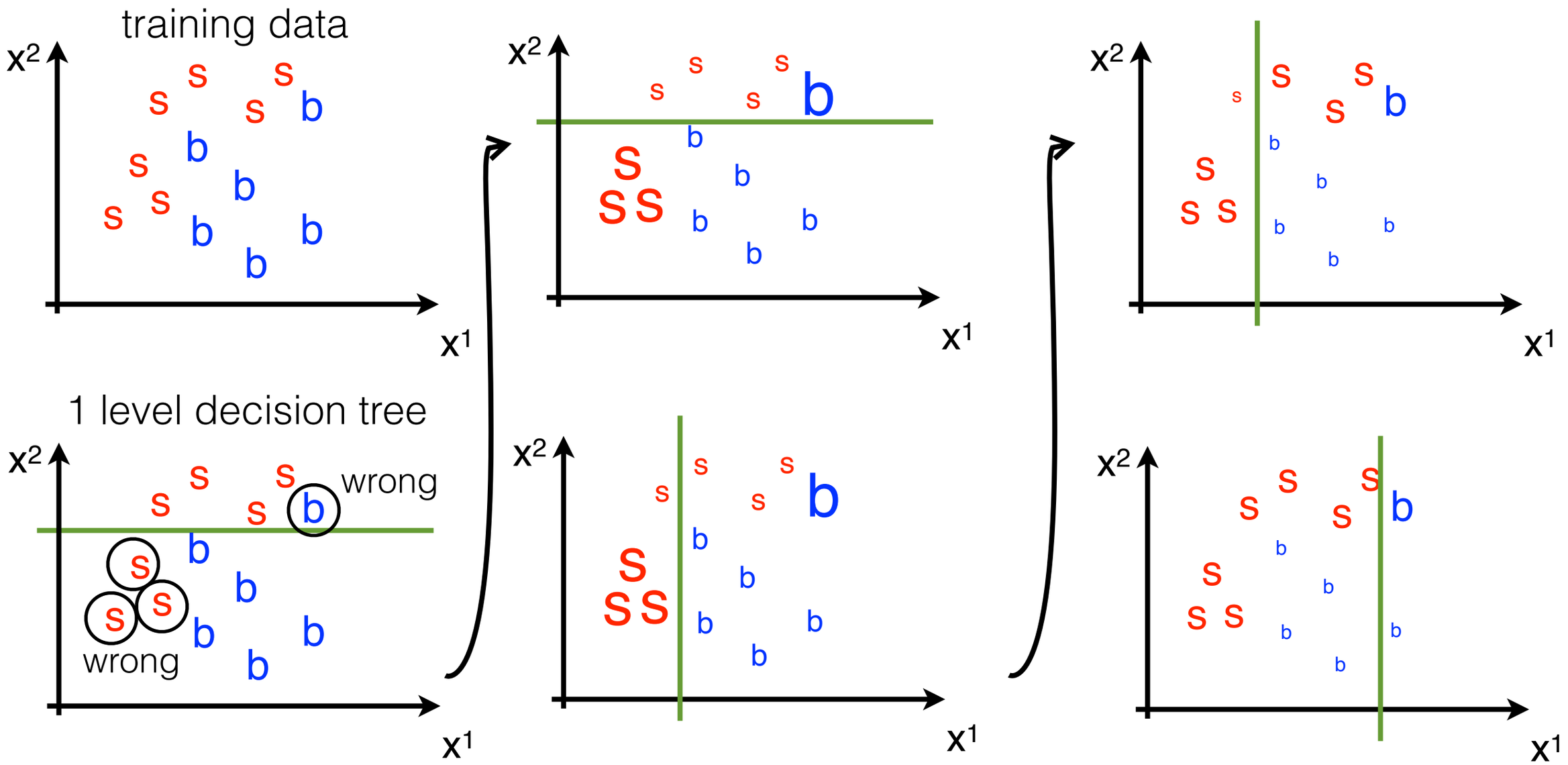
Right classification:
decrease weight

Wrong classification:
increase weight

it is “more important” to make this three “S” right than the other wrong

Boosting

Sequentially training a model learning from the errors of the previous ones. The idea is to create modifications that give smaller error rates than those of the preceding classifiers.



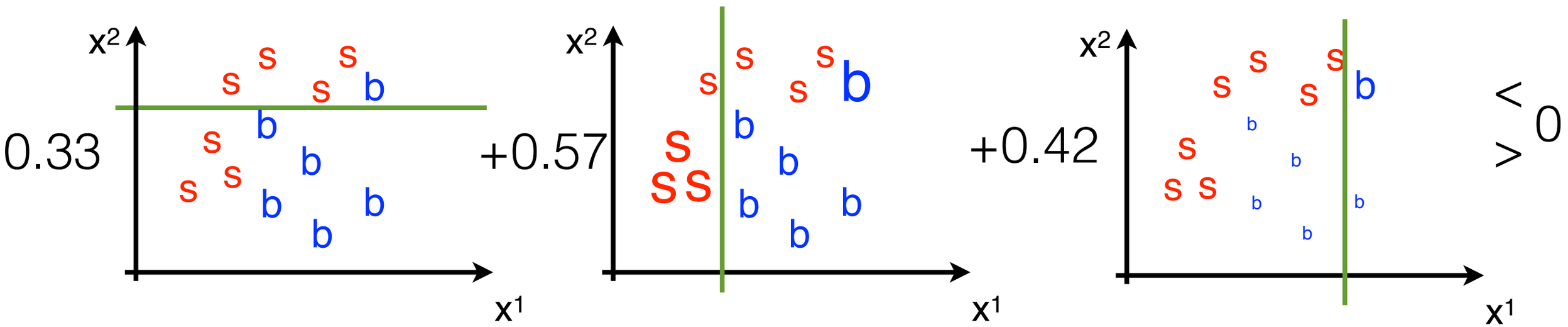
Boosting

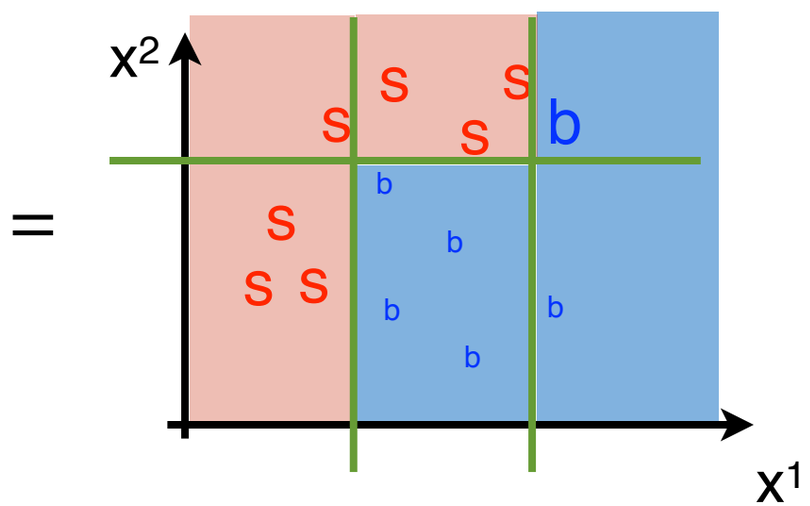
In practice:

assign numerical values to the two classes: b = +1 s =-1

assign a weight to each of the trees and sum them

(see next how to set the weight)





Boosting: assign the weights

Adaptive Boost: Adaboost (one of many algorithms)

For i = 1.. N_{boost}

{

$\vec{w}$ = train ($\vec{x}$,$\vec{y}$)

$\hat{\vec{y}}$ = predict ($\vec{w}$, $\vec{x}$)

Compute the vector of errors

$e = \vec{w} * (\vec{y} == \hat{\vec{y}})$

Set $\alpha_i = \frac{1}{2} \log\left(\frac{1-e}{e}\right)$

Update weights $w_j \rightarrow w_j e^{-\alpha_i (y_j \cdot \hat{y}_j)}$

$\vec{w} = \vec{w} / \sum(\vec{w})$

}

c(i) = classifier/tree (i)
$\vec{x}$ = vector of variables in
$\vec{y}$ = vector of class/target out
$\vec{w}$ = vector of weights

initially set all weights to 1, then evolve them

e = scalar error =
vector of weights * vector of 0s and 1s
correct/wrong

α at the step i

(true,predict)

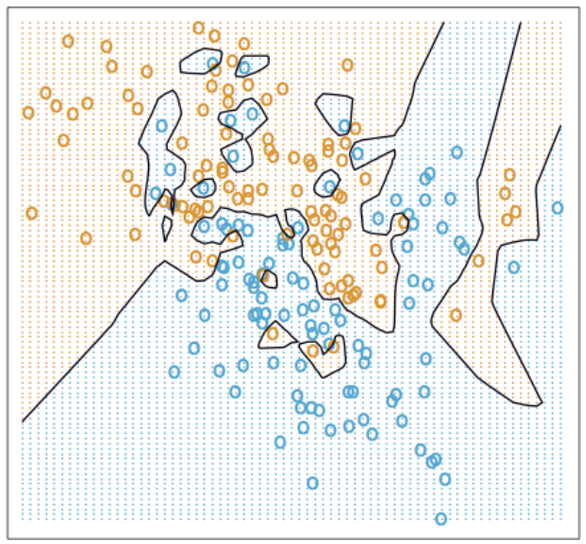
correct (s,s) or (b,b) ⇒ "+ sign"down-weighted

wrong (s,b) or (b,s) ⇒ "- sign" up-weighted

normalize by the sum of all weights

final classifier/tree = $\sum_i \alpha_i \, predict(c(i), x)$

Comments on BDTs:



Overtraining: it is “easy” to control using the parameters of the BDT:

- number of trees
- max depth of the tree
- how many events in the leaves
- (+various parameters of the specific boosting algorithm)

Correlated variables:

- adding correlated variables will not degrade the performance of the BDT because the less discriminating will be automatically de-weighted e.g. Gini index)

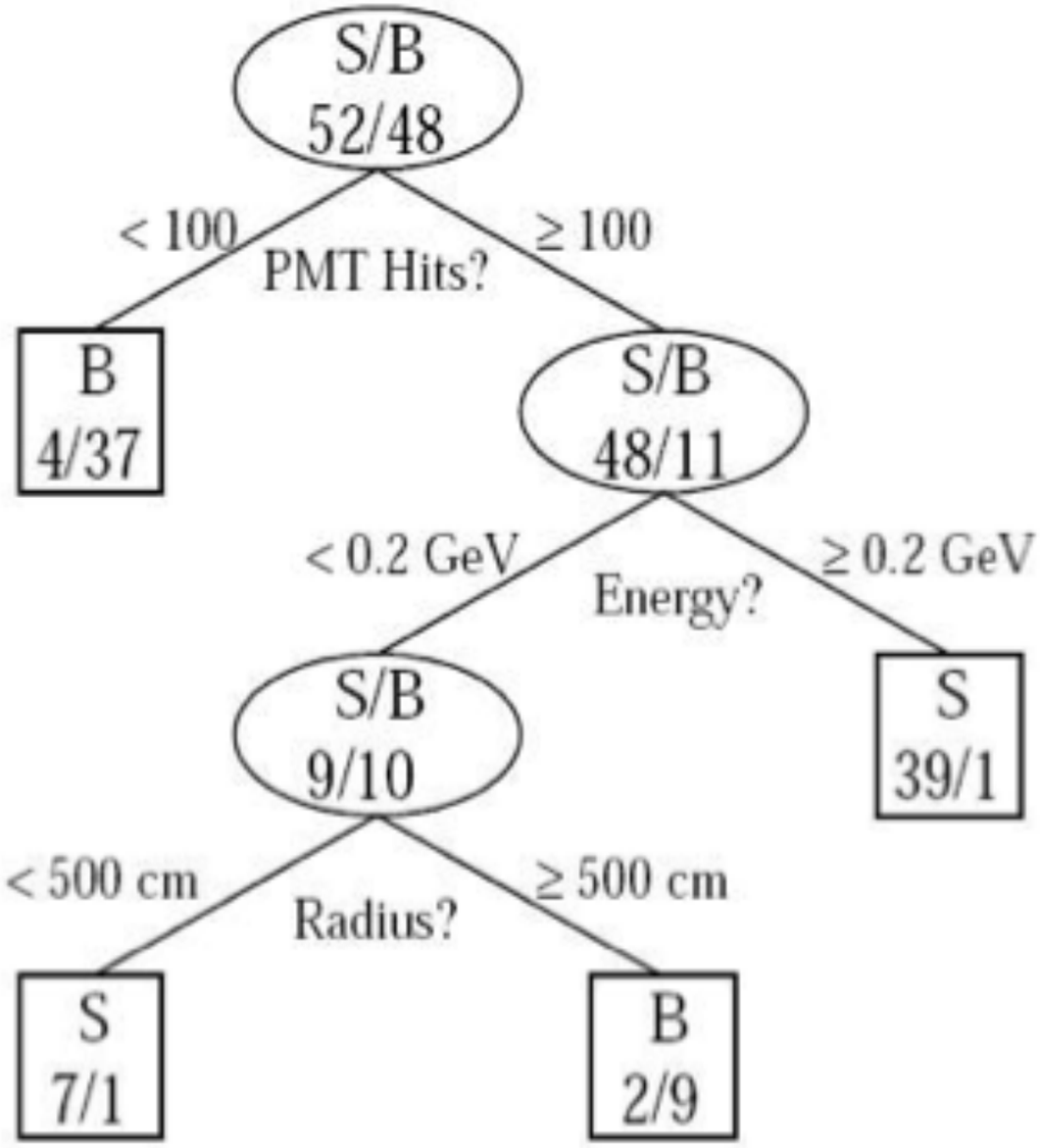
BDT packages:

- (root) TMVA
- scikit learn
- XGBOOST

Examples in HEP:

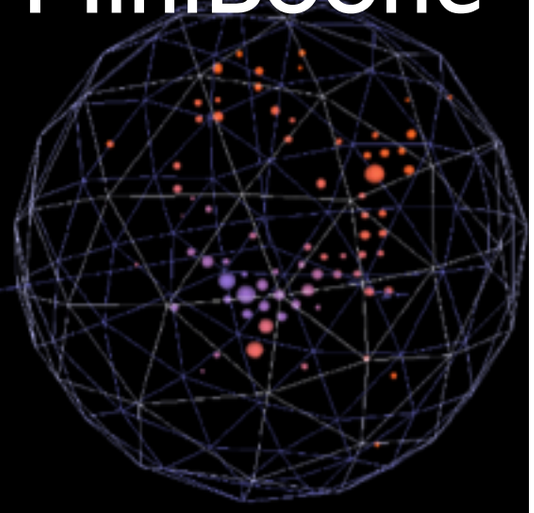
First BDT application in HEP

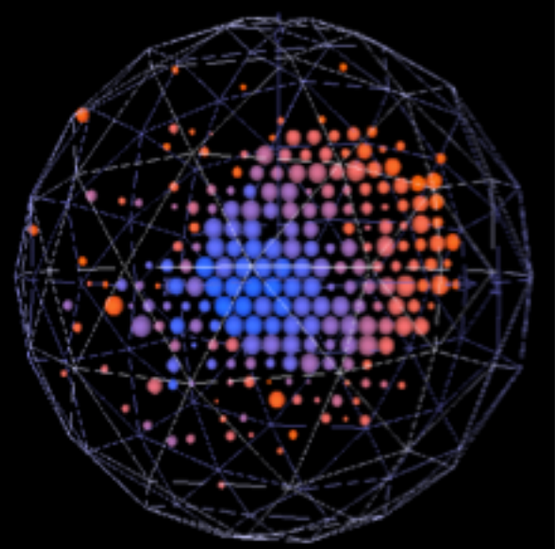
MiniBooNE, B. Roe et al., NIM 543 (2005) 577

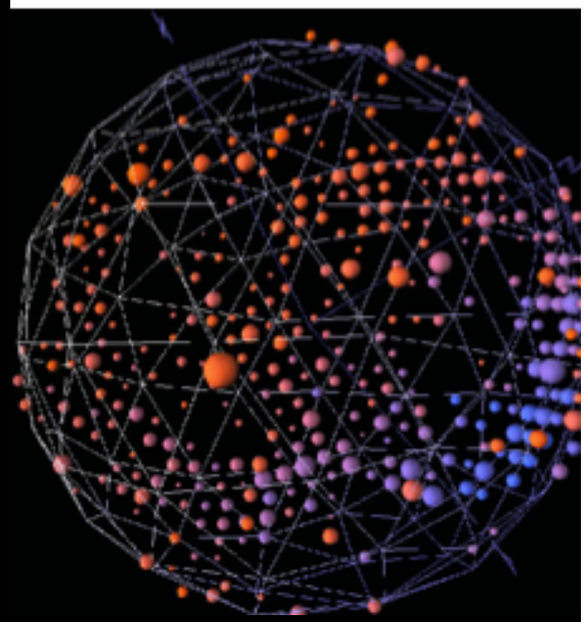


The analysis is set up to distinguish electrons / muon / π^0 events based on PMT hits, reconstructed energy, Radius of the ring

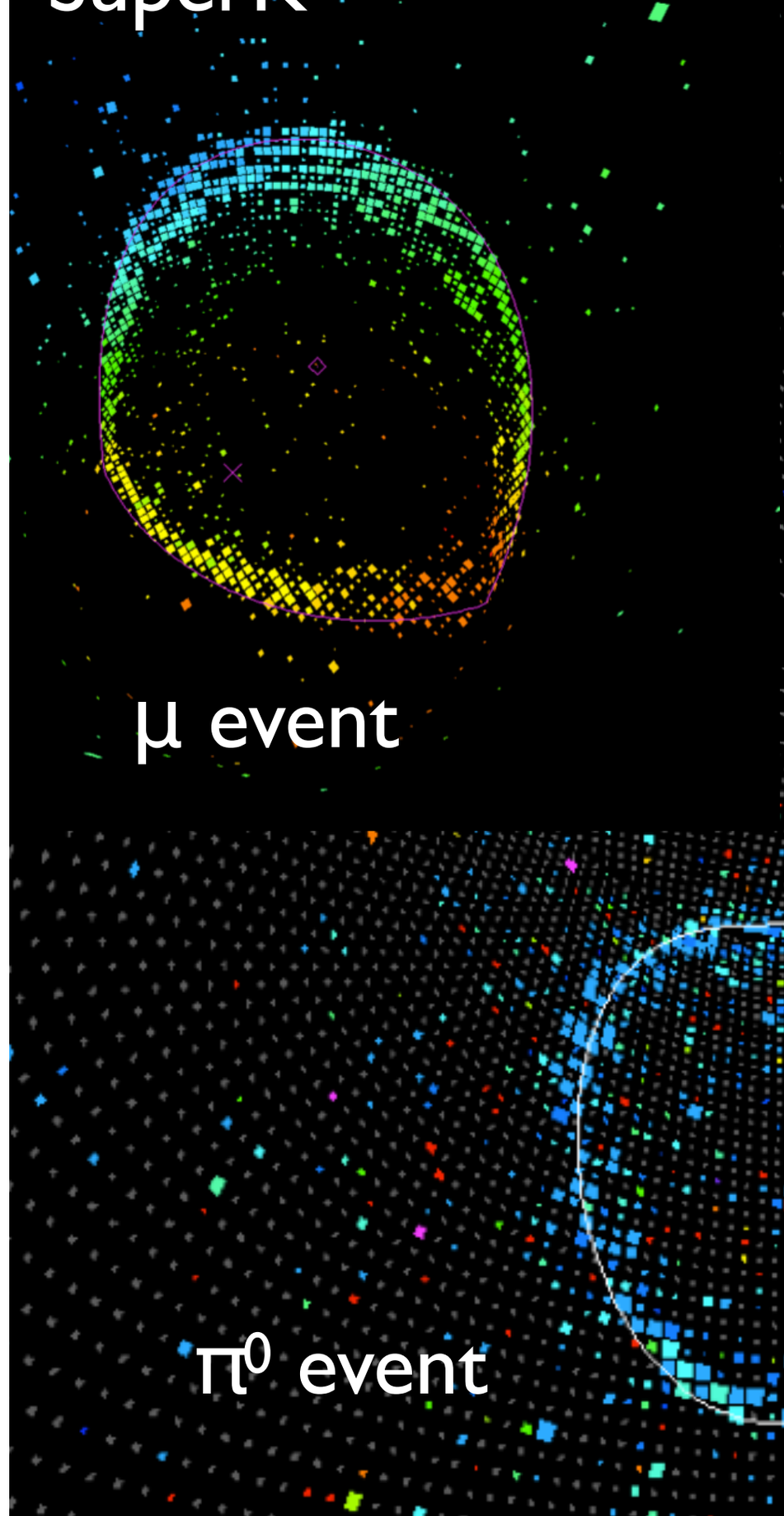
MiniBoone





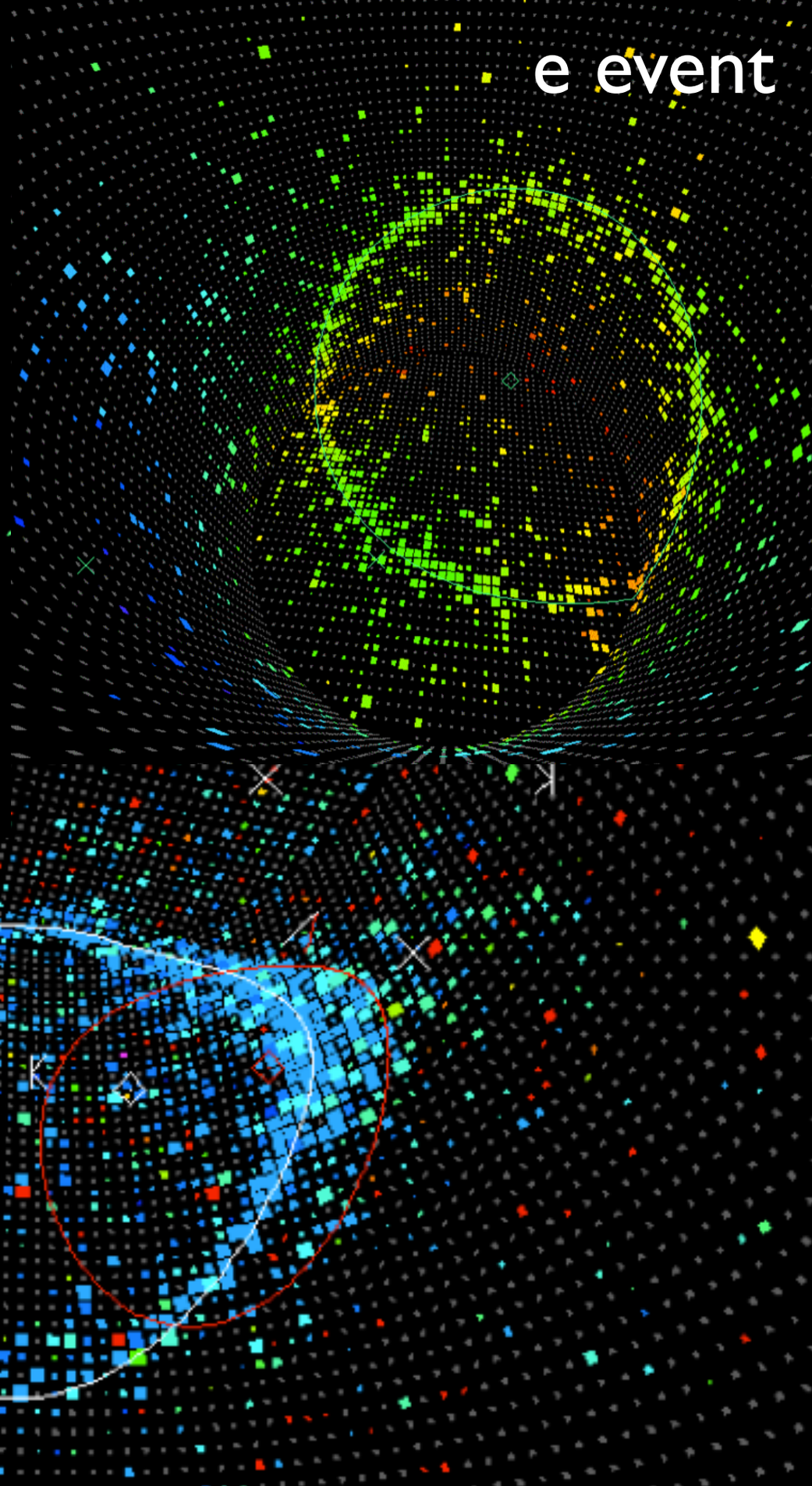


SuperK



μ event

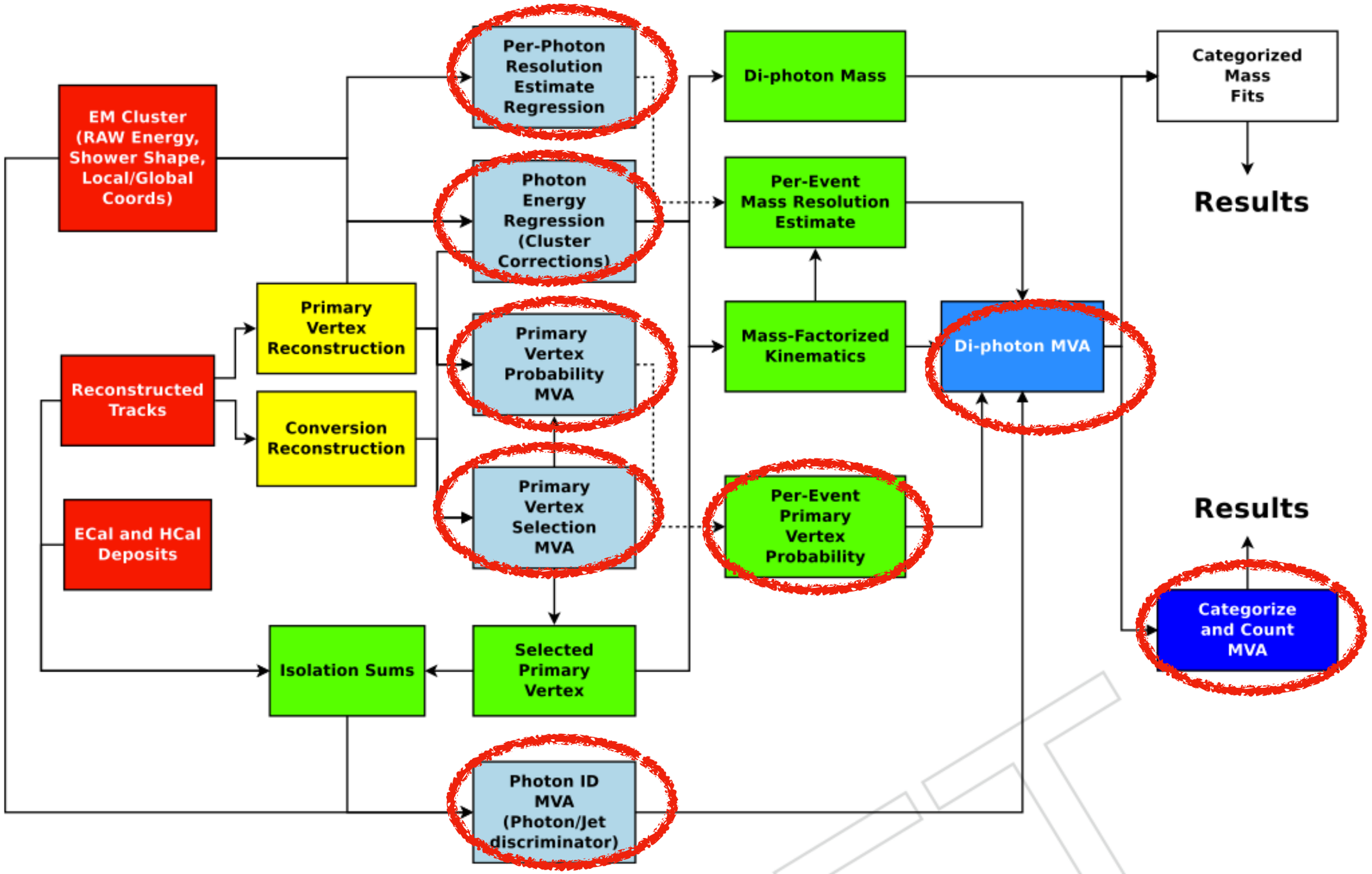
e event



π^0 event

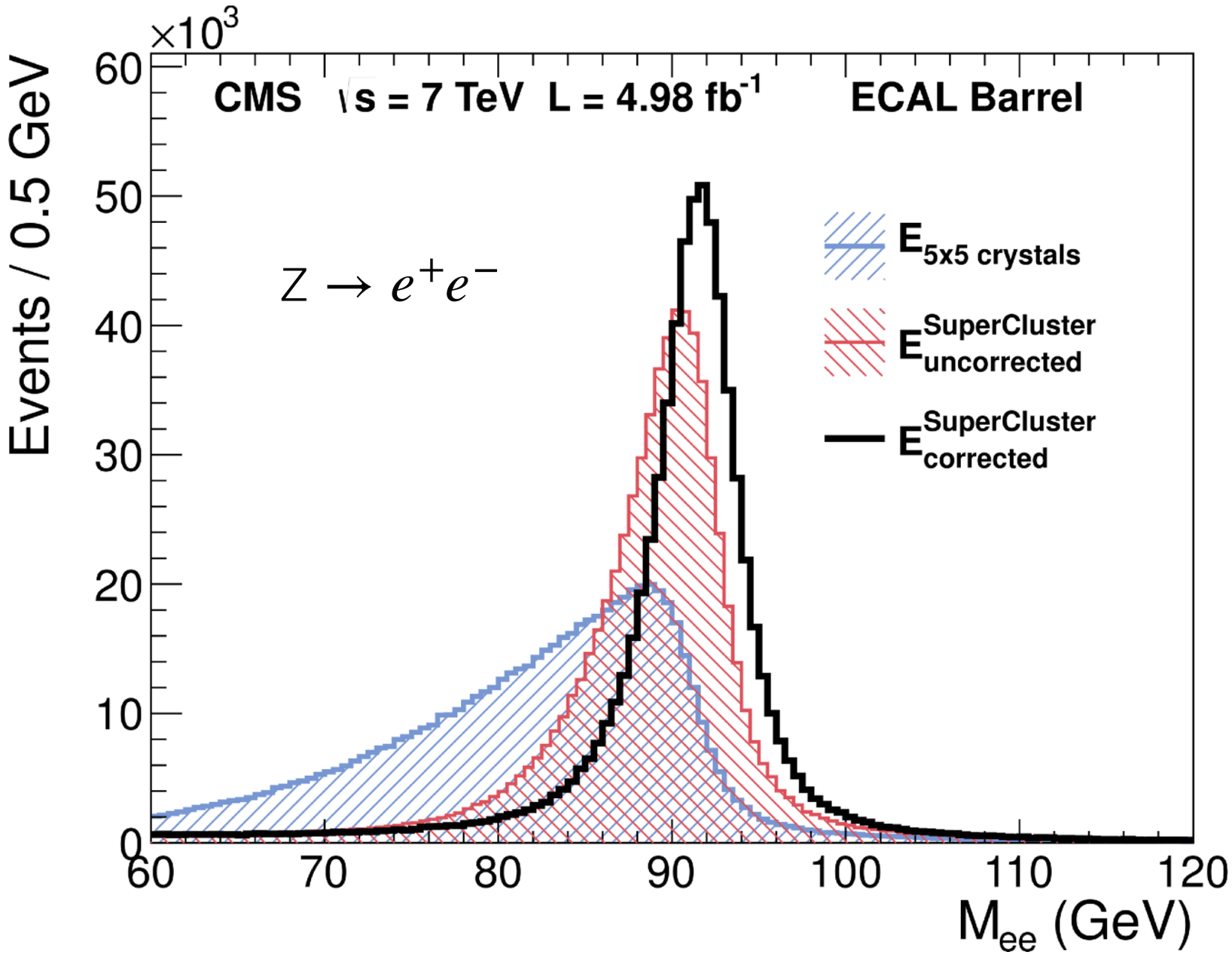
Examples: BDTs in H→γγ search

= BDT



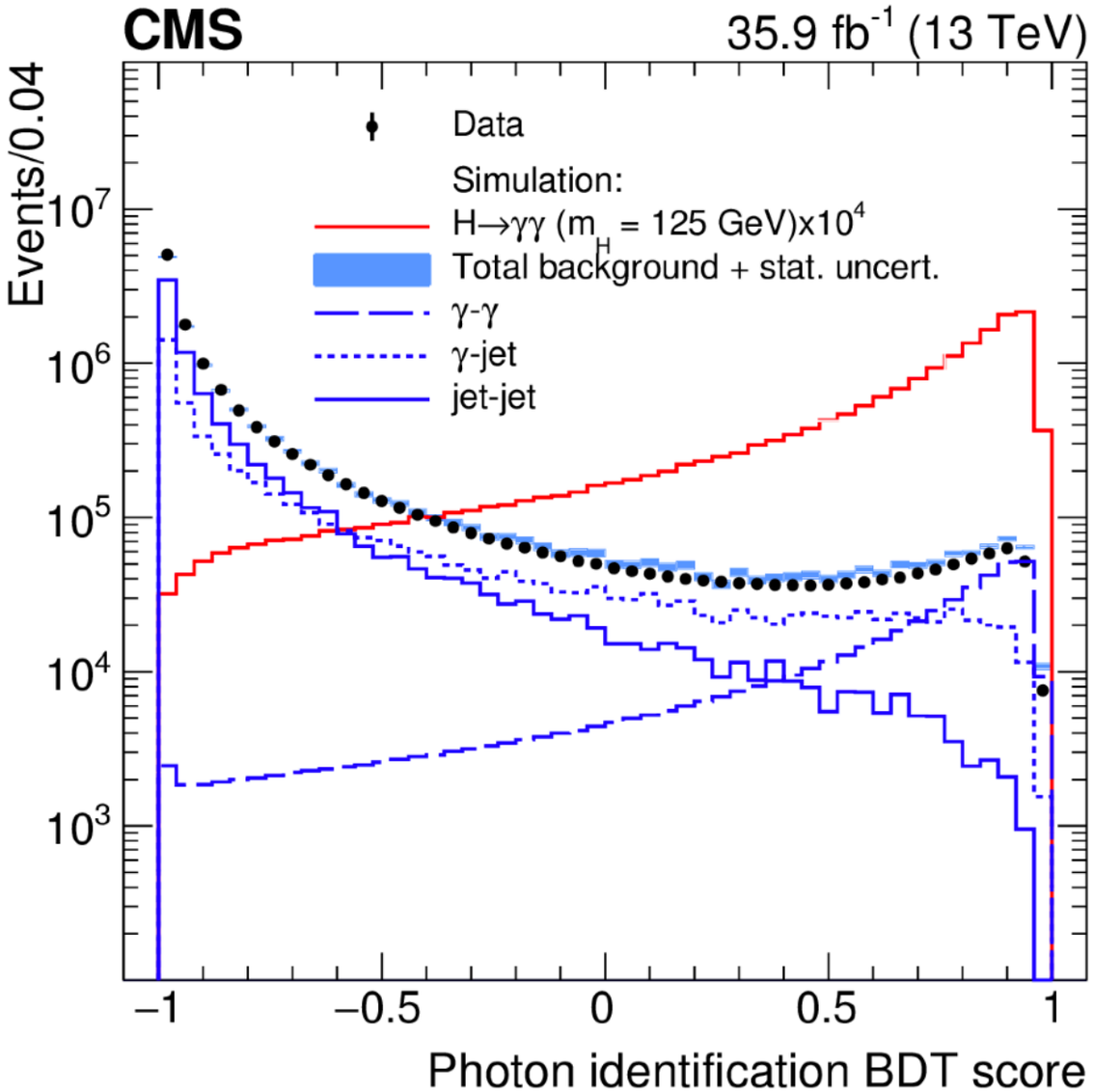
Examples:

Electron energy regression

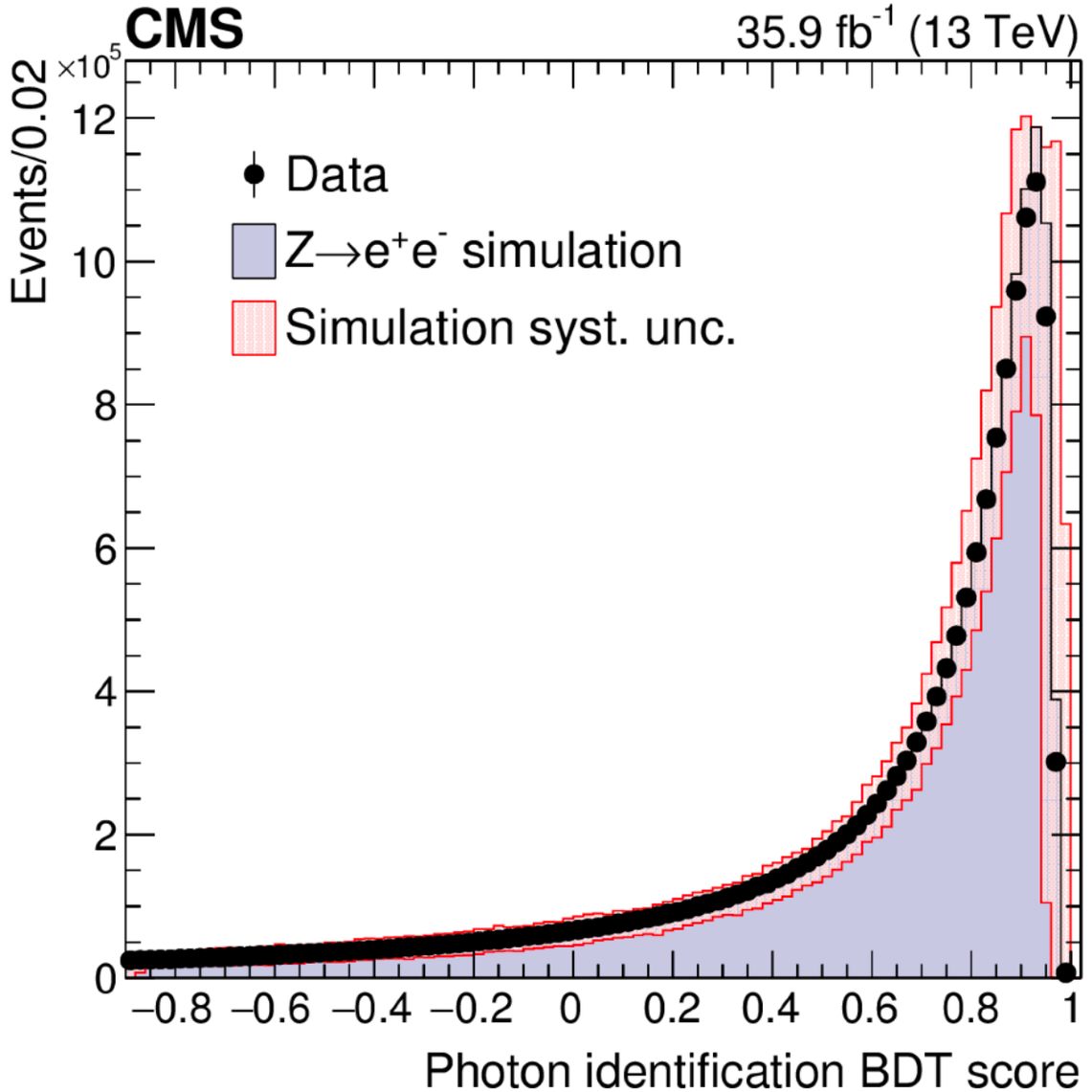


Examples:

Classify photons from jets faking photons



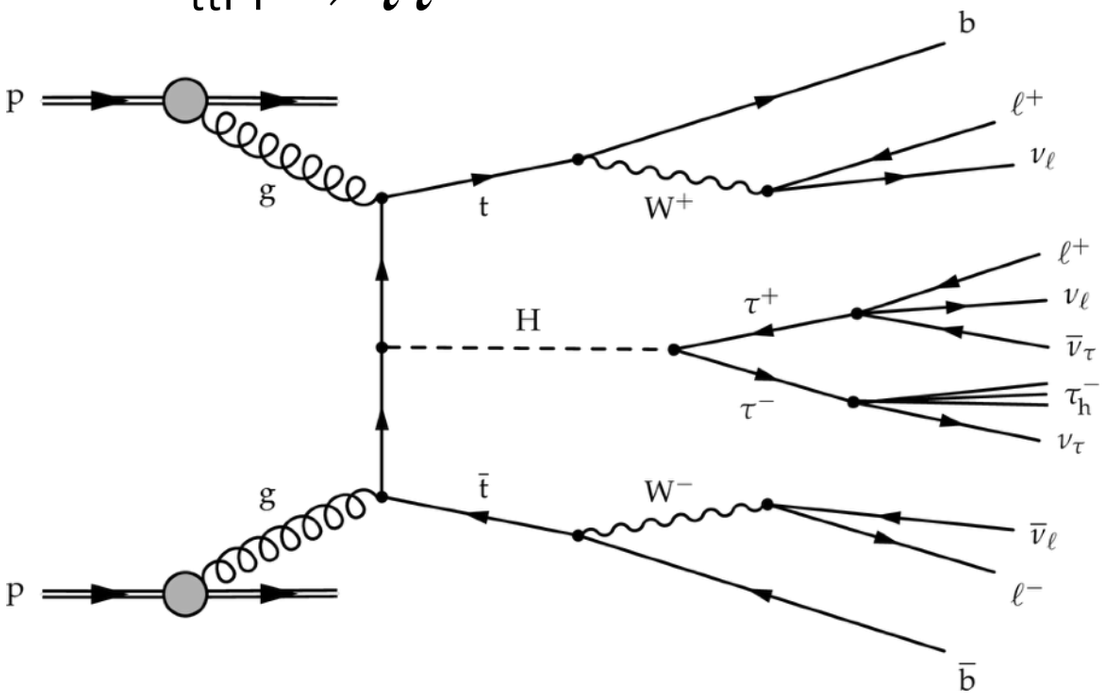
Validation with data Standard candle
(T&P electrons in data and MC)

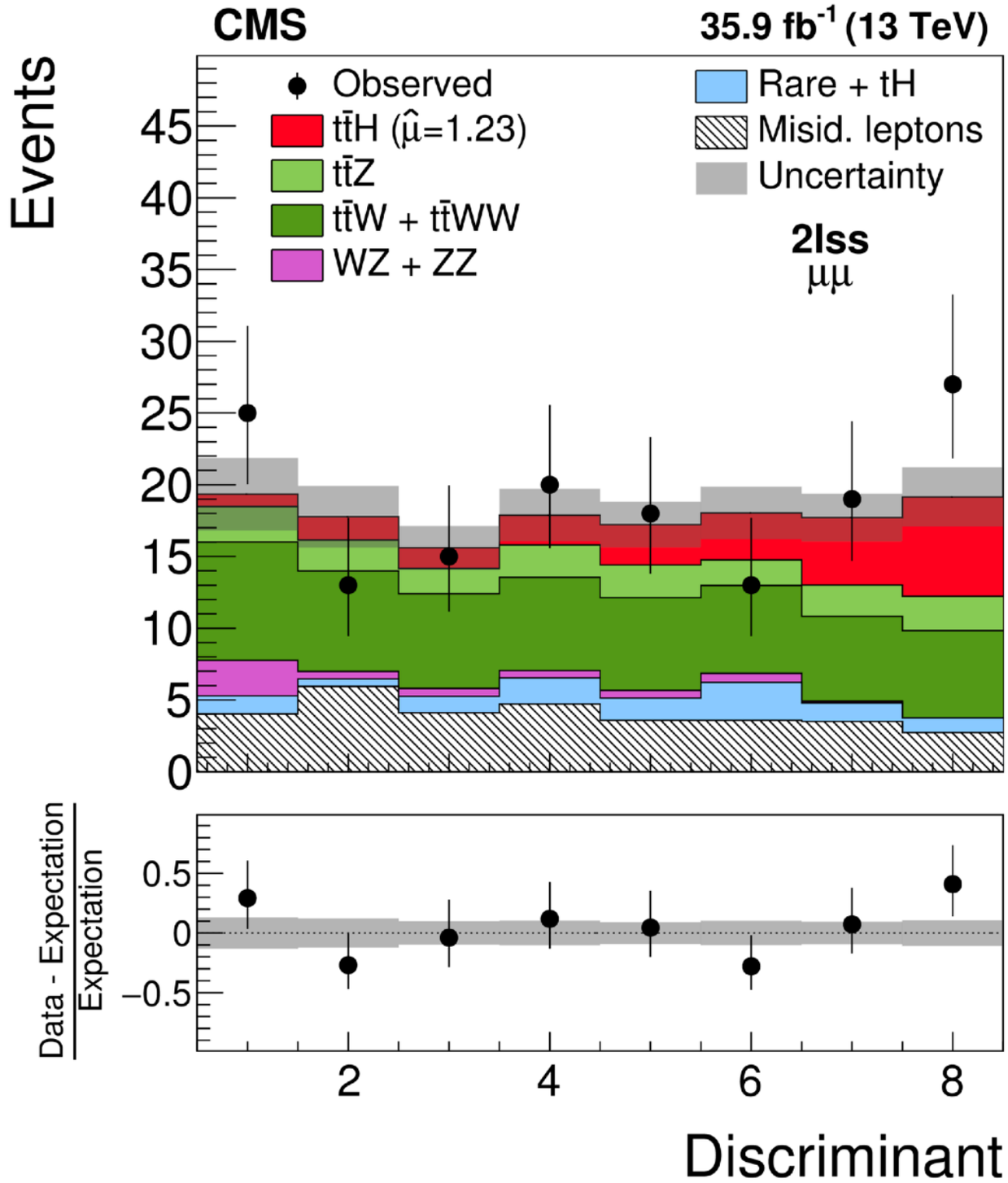


CMS-HIG-17-018

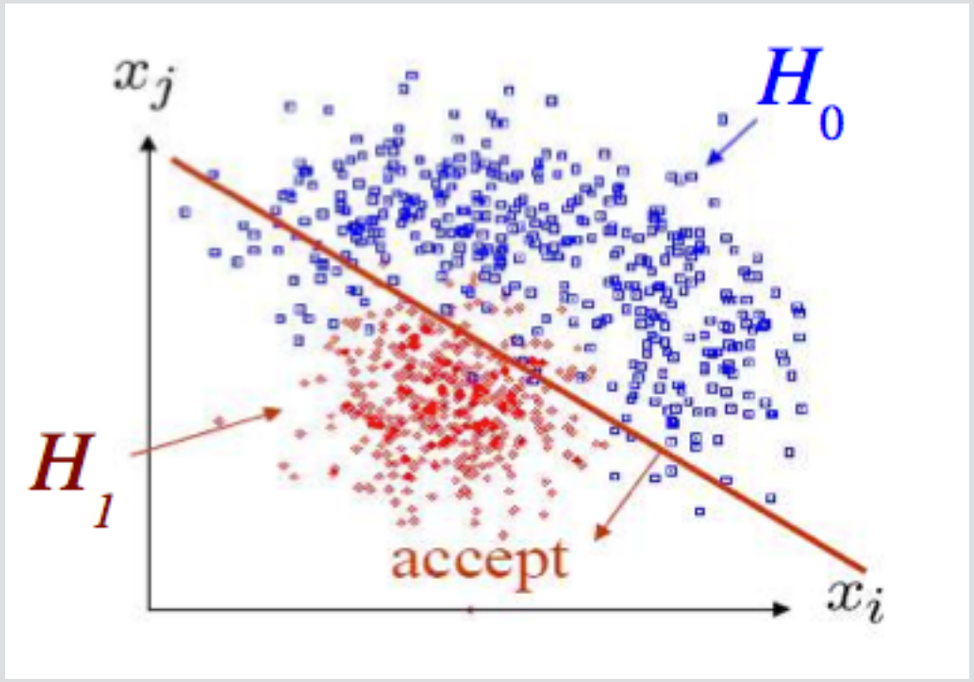
Examples:

ttH → $\tau\tau$





More material



Fisher discriminant

[Cowan]

Linear test statistics (I): Fisher discriminant

Reminder: here we want to approximate the test statistics with a multivariate discriminant

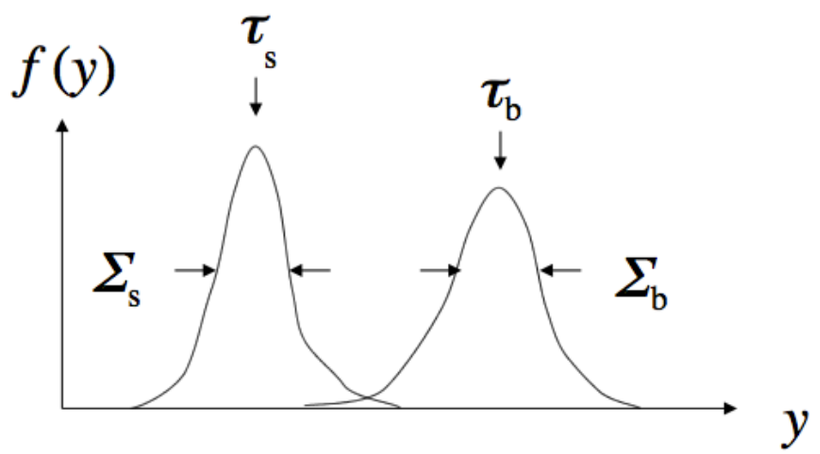
The easiest case is to model the likelihood ratio test statistics with a linear model

$$y(\vec{x}) = \frac{p(\vec{x}|\mathrm{s})}{p(\vec{x}|\mathrm{b})} = \sum_{i=1}^{n} w_i x_i = \vec{w}^T \vec{x}$$

The goal is to choose the weights w_i to maximize the "separation" between the pdf(y|sig), pdf(y|bkg)

Fisher definition of separation:

$$J(\vec{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$



So the goal becomes writing the means τ and the covariances Σ in terms of the weights and then find the weights values that maximize J

$$J = \frac{\text{separation between classes}}{\text{separation(spread) within classes}}$$

Linear test statistics (II)

We have $$(\mu_k)_i = \int x_i \, p(\vec{x}|H_k)\, d\vec{x}$$

k = 0 BKG k = 1 SIG

mean, covariance of $\boldsymbol{x}$

full covariance matrix $$(V_k)_{ij} = \int (x-\mu_k)_i (x-\mu_k)_j \, p(\vec{x}|H_k)\, d\vec{x}$$

where $k = 0,\ 1$ (hypothesis)

and $i, j = 1, \ldots, n$ (component of $\boldsymbol{x}$)

IN THE X SPACE

the input to the discriminant

For the mean and variance of $y(\vec{x})$ we find

$$\tau_k = \int y(\vec{x})\, p(\vec{x}|H_k)\, d\vec{x} = \vec{w}^T \vec{\mu}_k$$

$$\Sigma_k^2 = \int (y(\vec{x})-\tau_k)^2 \, p(\vec{x}|H_k)\, d\vec{x} = \vec{w}^T V_k \vec{w}$$

and in the test statistics space

tau = TARGET

the output of the discriminant

Linear test statistics (III)

The numerator of $J(\boldsymbol{w})$ is

$$(\tau_0-\tau_1)^2=\sum_{i,j=1}^{n} w_i w_j (\mu_0-\mu_1)_i (\mu_0-\mu_1)_j$$

DIFFERENCE BETWEEN THE CLASSES

$$=\sum_{i,j=1}^{n} w_i w_j B_{ij}=\vec{w}^T B \vec{w}$$

'between' classes

means

SPREAD WITHIN THE CLASSES

and the denominator is

$$\Sigma_0^2+\Sigma_1^2=\sum_{i,j=1}^{n} w_i w_j (V_0+V_1)_{ij}=\vec{w}^T W \vec{w}$$

'within' classes

widths

→ maximize w.r.t $\vec{w}$

$$J(\vec{w})=\frac{\vec{w}^T B \vec{w}}{\vec{w}^T W \vec{w}}=\frac{\text{separation between classes}}{\text{separation within classes}}$$

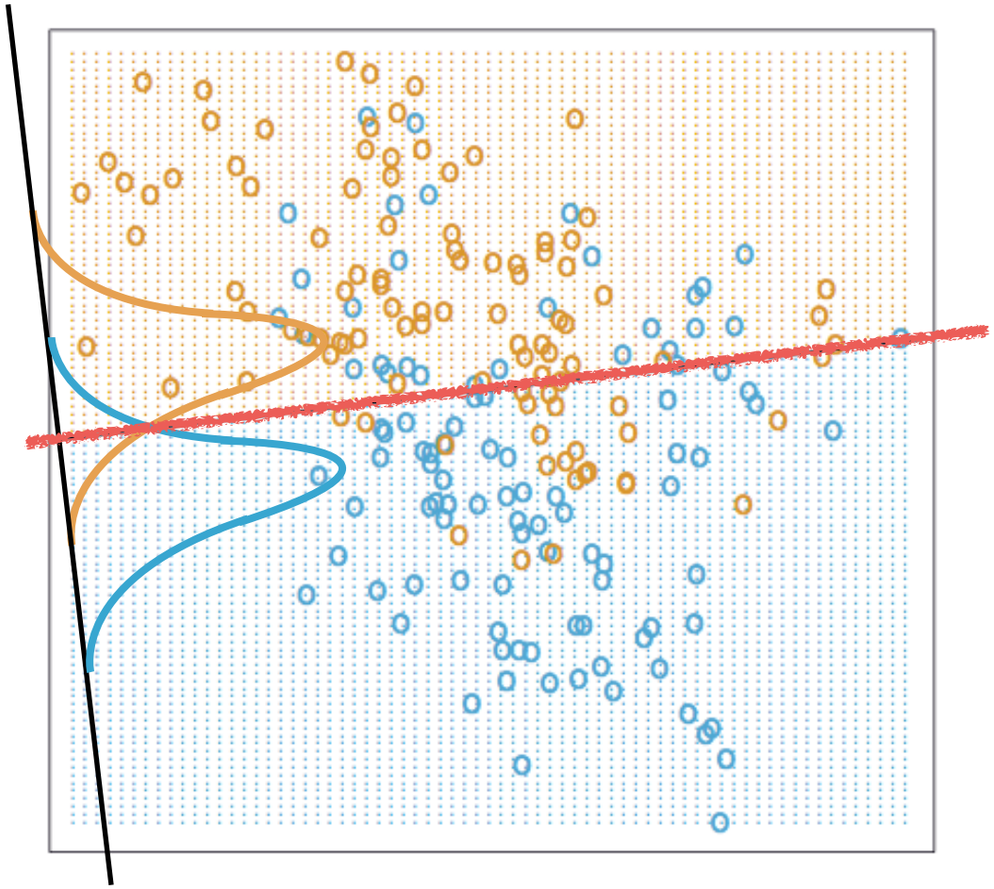
Linear test statistics (IV)

We need to find the coefficients w_i : "training"

The maximum of J is found by solving $\frac{\partial J}{\partial w_i}=0$

giving the Fisher discriminant:

$$y(\vec{x})=\vec{w}^T\vec{x} \qquad \text{with } \vec{w}\propto W^{-1}(\vec{\mu}_0-\vec{\mu}_1)$$



The red straight line represents a n-1 hyperplane in the n-dimensional space. You can see the separation by projecting the points orthogonally to the plane

Linear test statistics (V)

For non linear problems, sometimes you can come up with a non linear map of the input variables and then apply a linear test-statistics

$$\varphi_1 = \tan^{-1}(x_2/x_1)$$

$$\varphi_2 = \sqrt{x_1^2 + x_2^2}$$