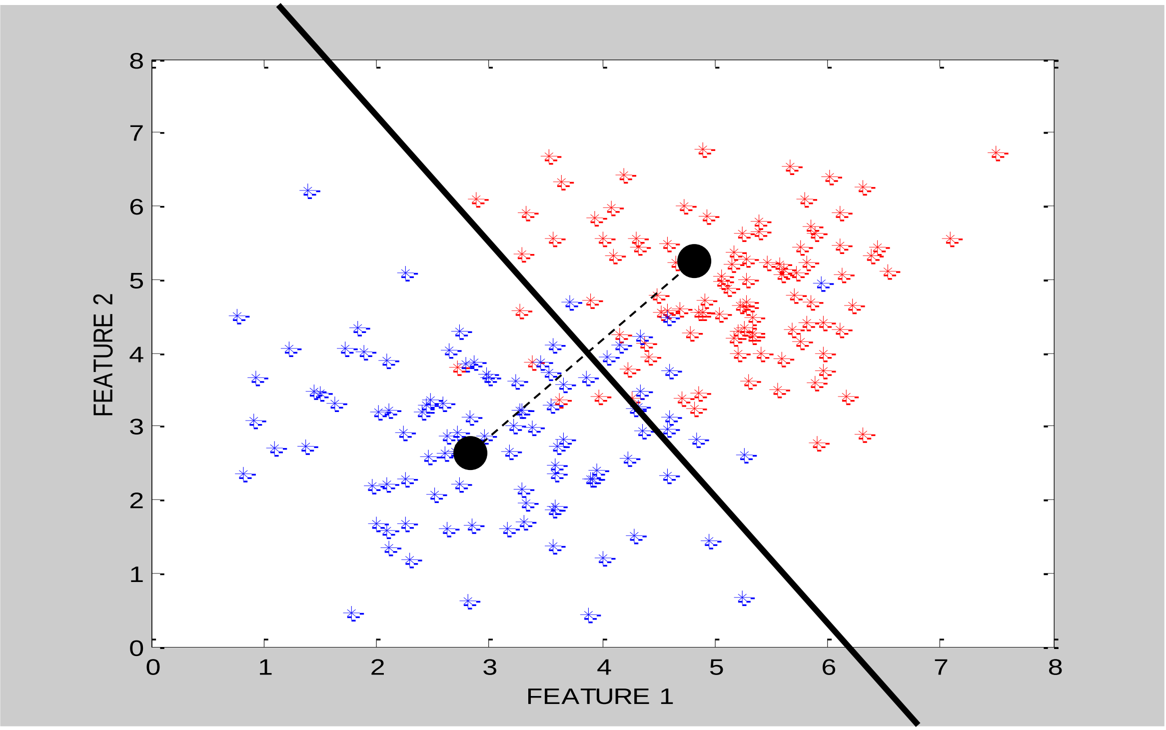
Master in Computer Vision Barcelona

Module: M3. Machine learning for computer vision

Lecture 4: Foundations of Classifier Ensembles

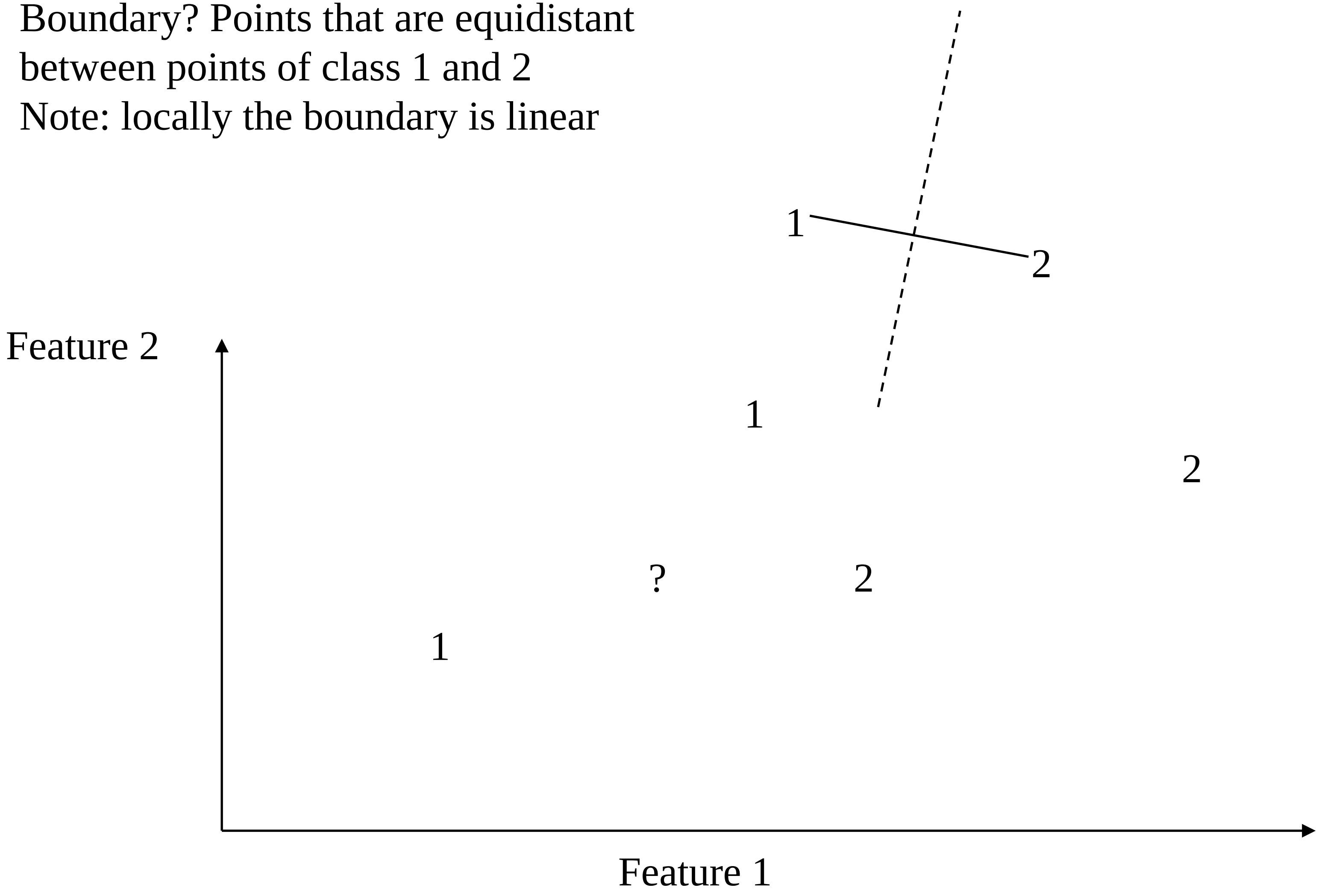
Lecturer: Ramon Baldrich / Fernando Vilariño

Minimum Distance Classifier

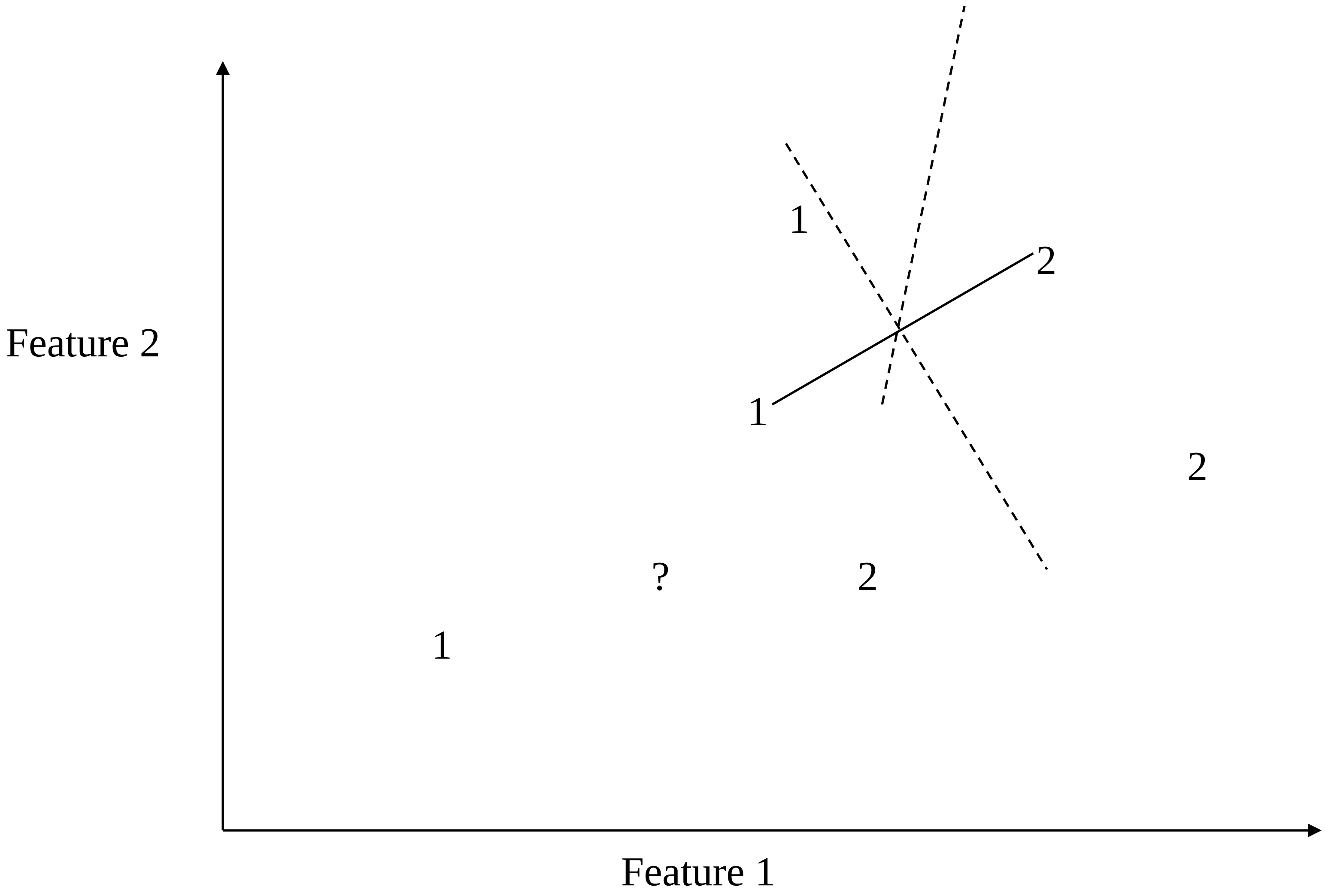


Local Decision Boundaries

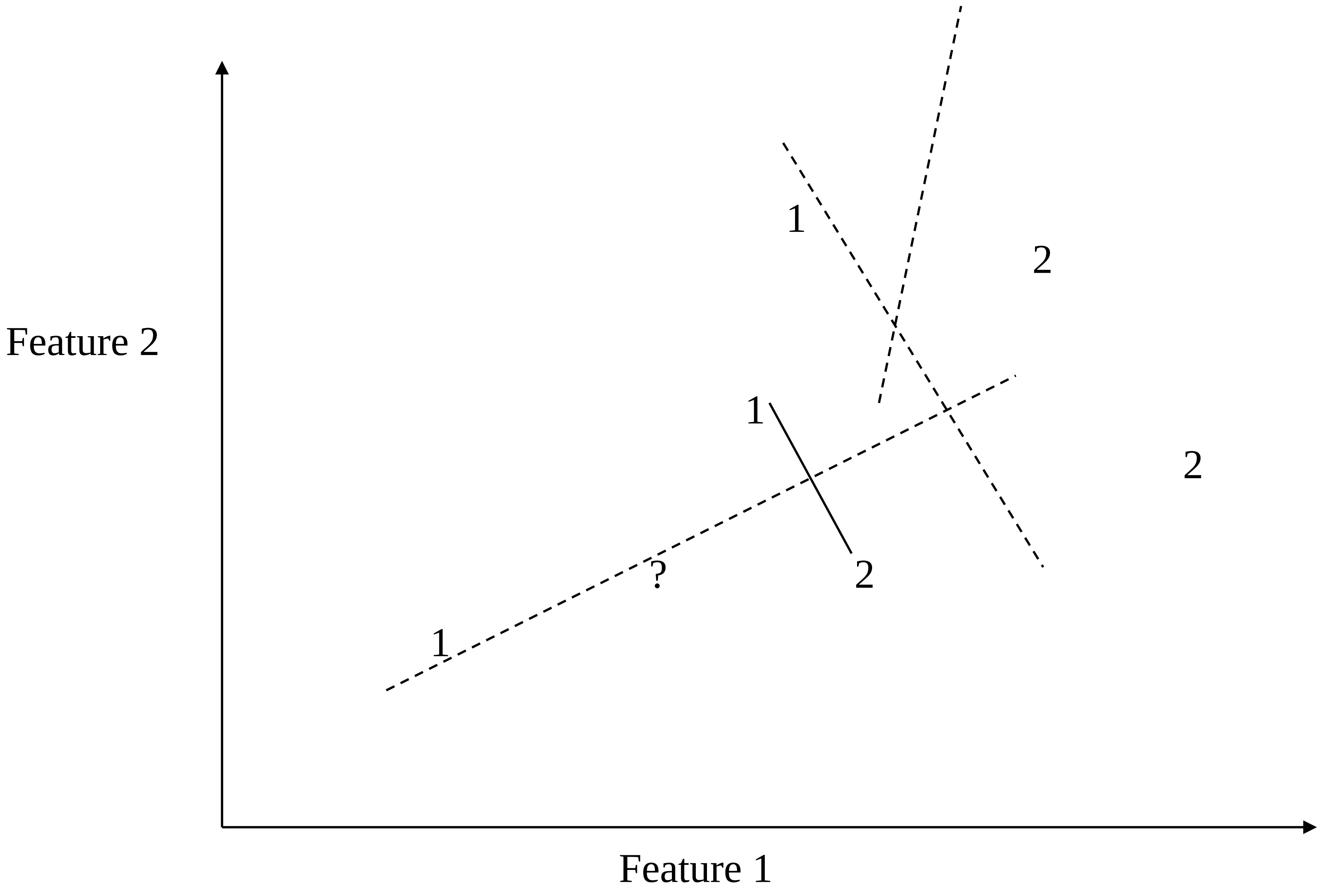
Boundary? Points that are equidistant between points of class 1 and 2
Note: locally the boundary is linear



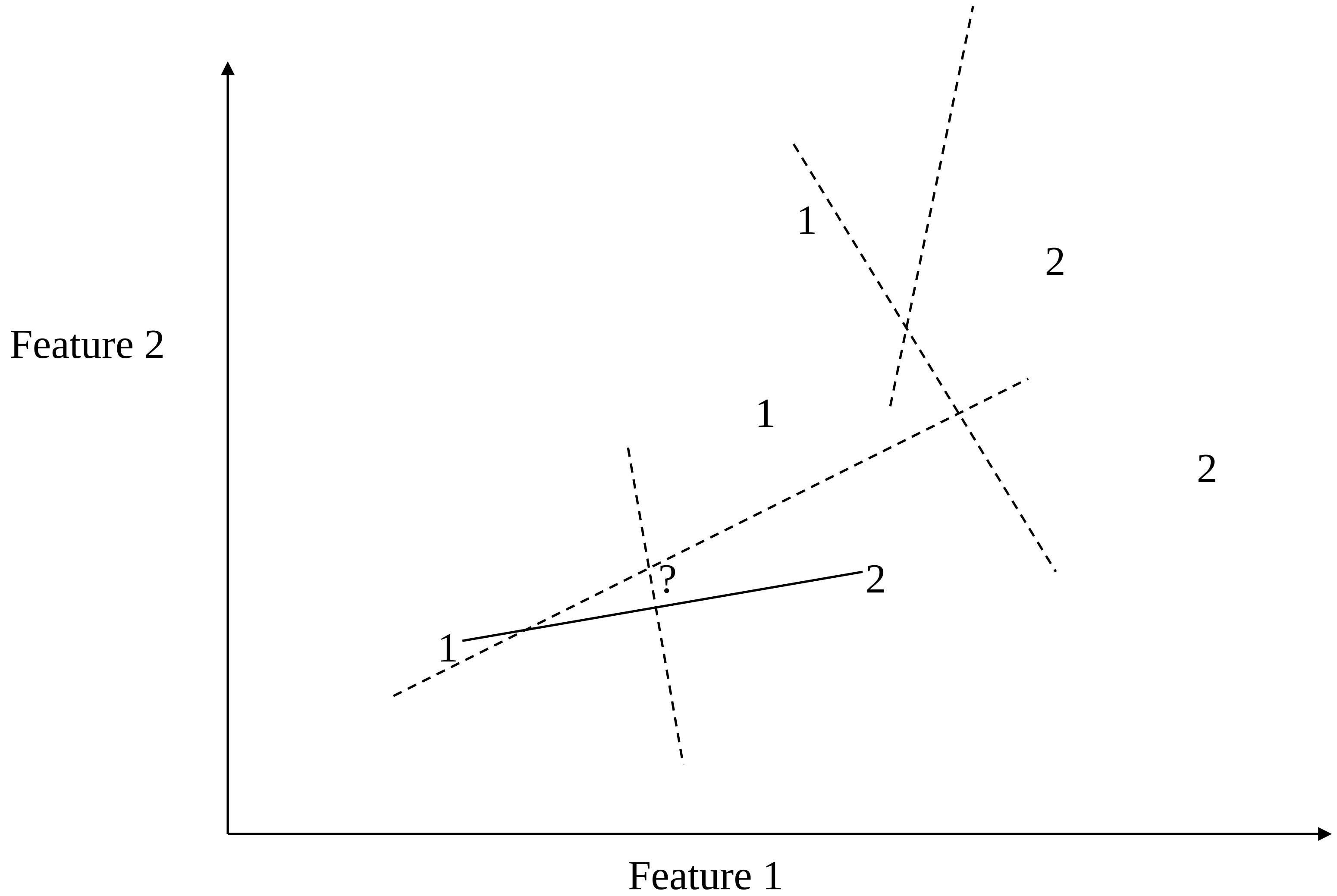
Finding the Decision Boundaries



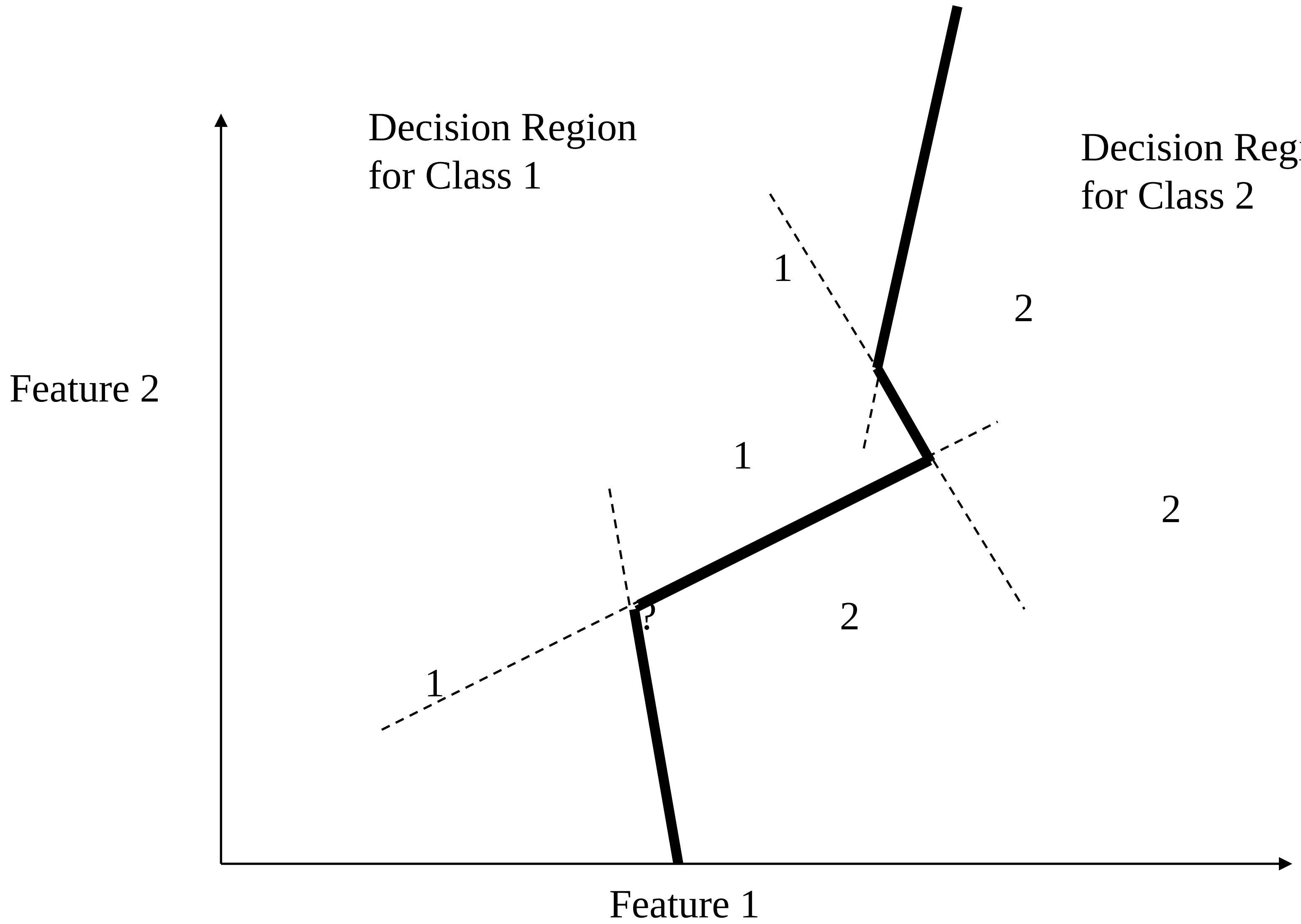
Finding the Decision Boundaries



Finding the Decision Boundaries

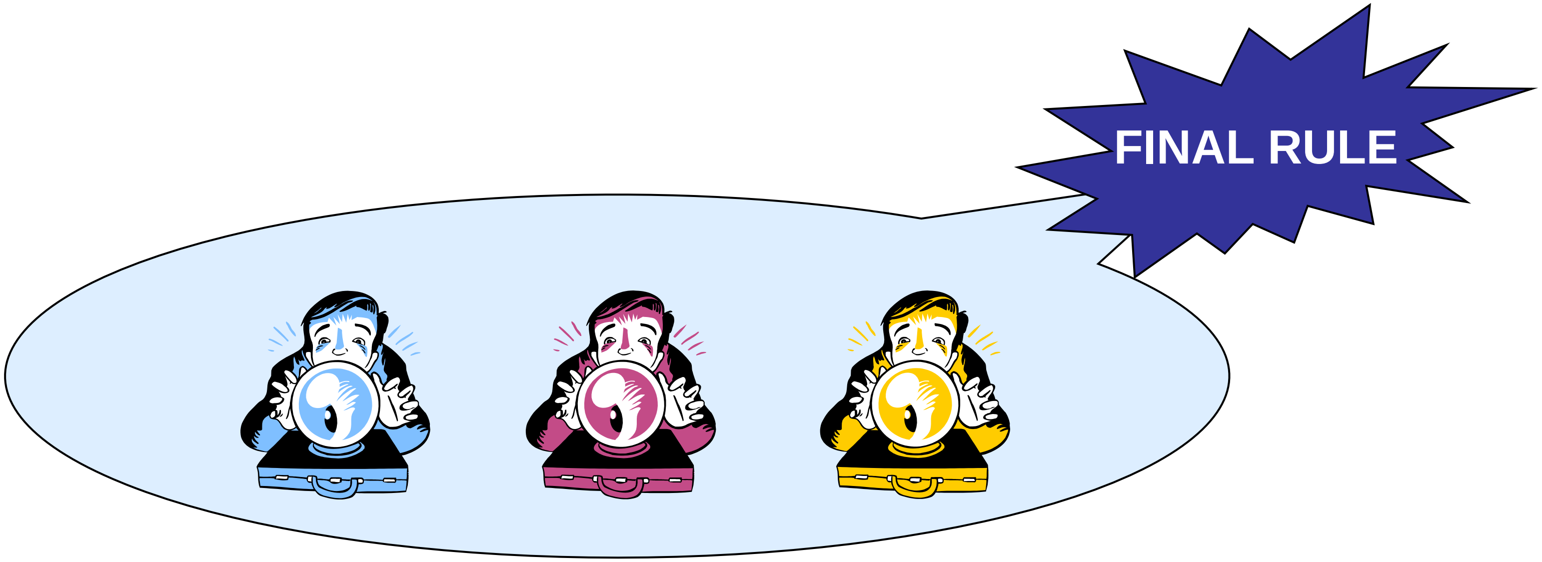


Overall Boundary = Piecewise Linear



- Bagging and Boosting

➔Aggregating Classifiers



Breiman (1996) found gains in accuracy by aggregating predictors built from reweighed versions of the learning set

Bagging and Boosting: Aggregating Classifiers

3 questions:

? How to **reweigh** ?

? How to **aggregate** ?

? Which type of **gain** in accuracy ?

Bagging

- *Bagging* = Bootstrap Aggregating
- Reweighing of the learning sets is done by drawing at random with replacement from the learning sets
- Predictors are aggregated by plurality voting

The Bagging Algorithm

- B bootstrap samples
- From which we derive:
  - ***B* Classifiers** $\in \{-1, 1\}: c^1, c^2, c^3, \ldots, c^B$
  - ***B* Estimated probabilities** $\in [0, 1]: p^1, p^2, p^3, \ldots, p^B$

The aggregate classifier becomes:

$$c_{bag}(x) = sign\left(\frac{1}{B} \sum_{b=1}^{B} c^b(x) \right)$$

or

$$p_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} p^b(x)$$

Bagging Example (Opitz, 1999)

Original	1	2	3	4	5	6	7	8
Training set 1	2	7	8	3	7	6	3	1
Training set 2	7	8	5	6	4	2	7	1
Training set 3	3	6	2	7	5	6	2	2
Training set 4	4	5	1	4	6	4	3	8

Aggregation

Sign

Classifier 1 

+

Classifier 2 

+

Classifier 3 

+

…

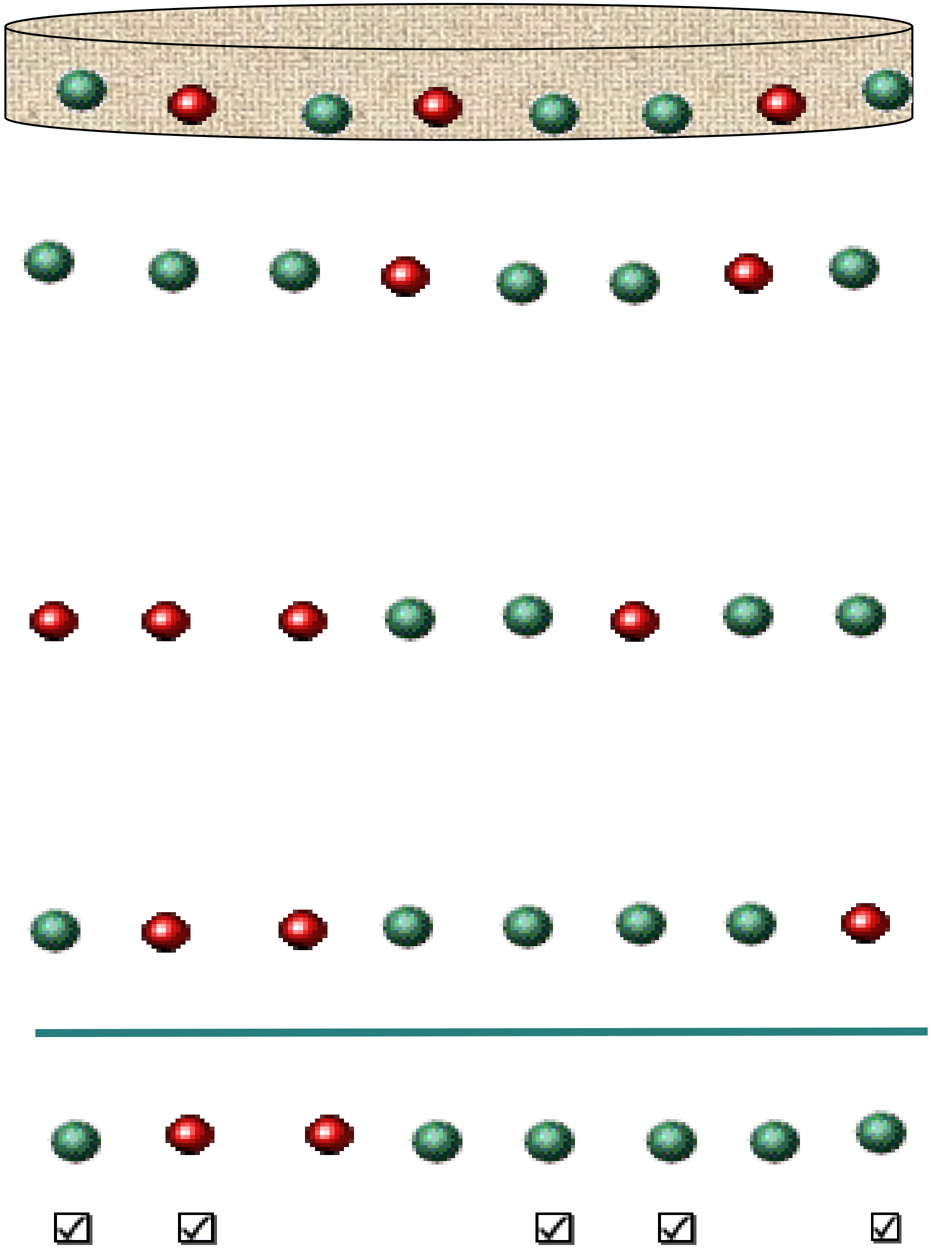
+

Classifier T 

=

Final rule

Initial set



Boosting

- Freund and Schapire (1997), Breiman (1998)
- Data *adaptively resampled*

Previously misclassified observations ➔ weights ⬆

Previously wellclassified observations ➔ weights ⬇

Predictor aggregation done by *weighted voting*

AdaBoost

$$y_i \in \{-1,+1\}$$

- Initialize weights: $w_i^1 = 1/N$
- Fit a classifier with these weights
- Give predicted probabilities to observations according to this classifier

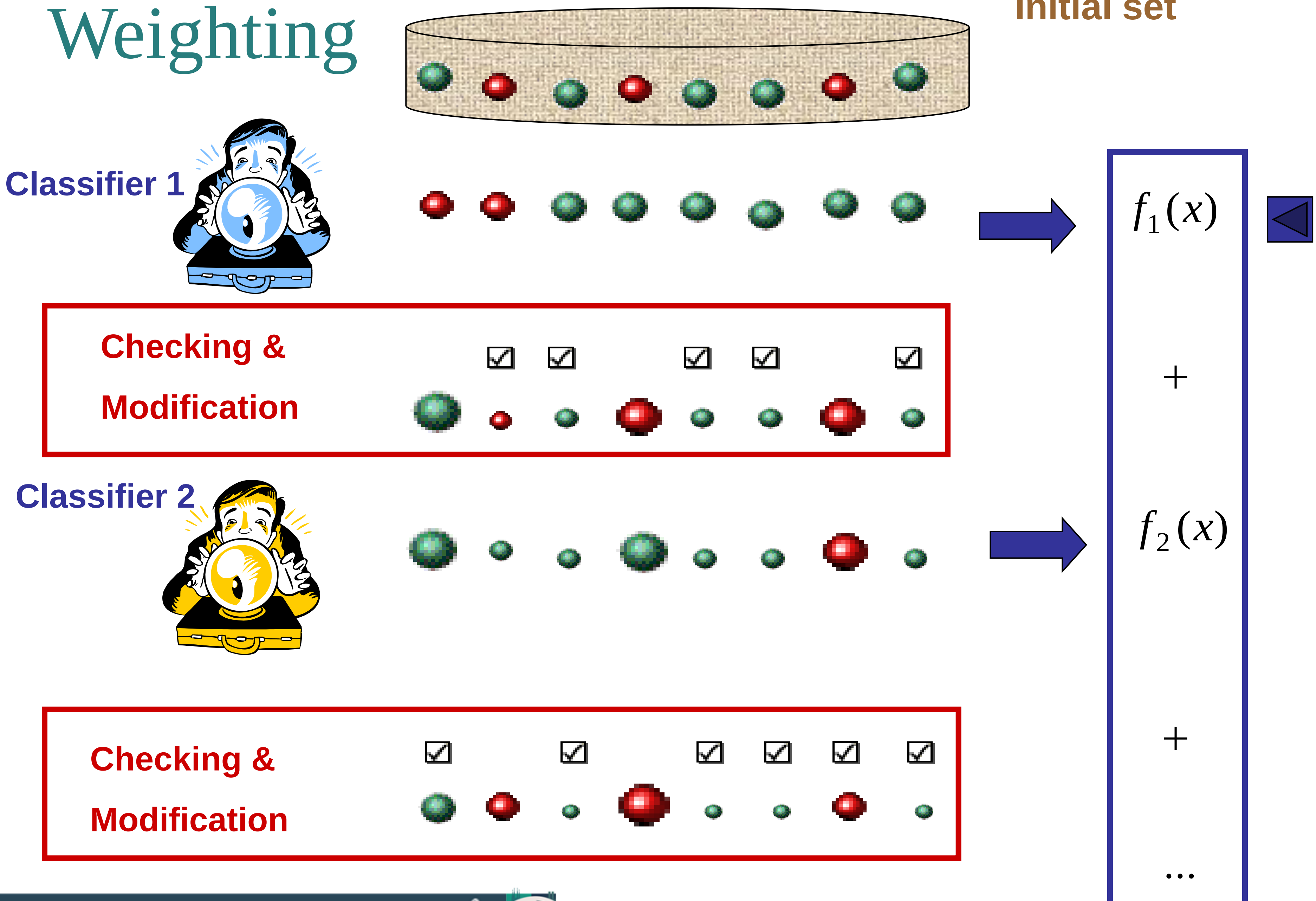
$$p_b(x) = \hat{P}_w(y=1|x) \in [0,1]$$

- Compute "pseudo probabilities": $f_b(x) = \frac{1}{2}\log\left(\frac{p_b(x)}{1-p_b(x)}\right) \in \Re$
- Get new weights: $w_i^{b+1} = w_i^b \exp[-y_i f_b(x_i)]$

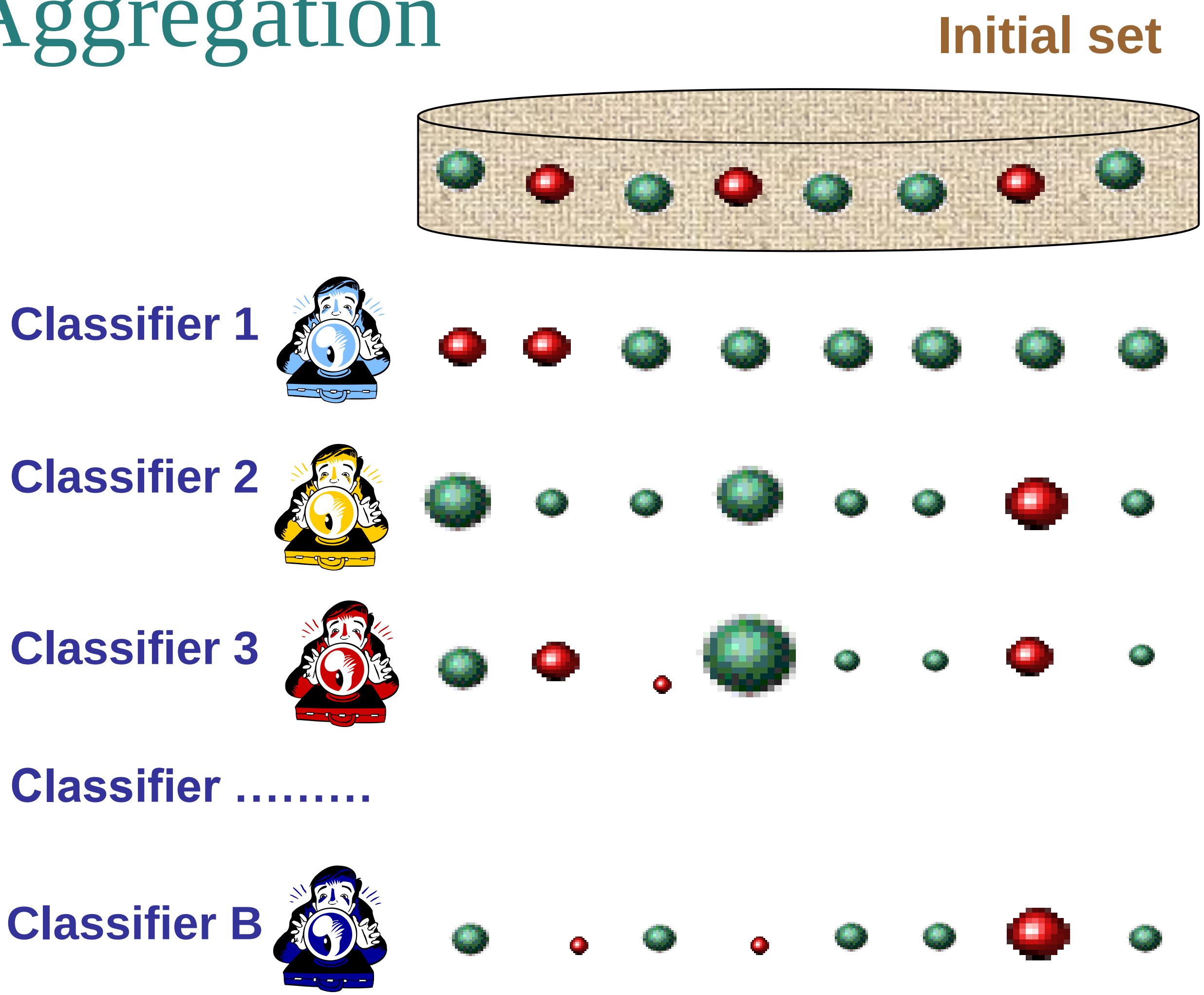
& "Normalize" it (i.e., rescale so that it sums to 1)

- Combine the "pseudo probabilities": $c_{Boost} = sign\left[\sum_{b=1}^{B} f_b(x)\right]$

Weighting


Initial set
Classifier 1
$f_1(x)$
Checking &
Modification
+
Classifier 2
$f_2(x)$
Checking &
Modification
+
...

Aggregation



Sign

$$\underbrace{f_1(x) + f_2(x) + f_3(x) + \dots + f_B(x)}$$

$=$

Final rule

Boosting

- Definition of Boosting:

  Boosting refers to a general method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb.

- Intuition:

  1) No learner is always the best;

  2) Construct a set of base-learners which when combined achieves higher accuracy

Boosting

3) Different learners may:

--- Be trained by different algorithms

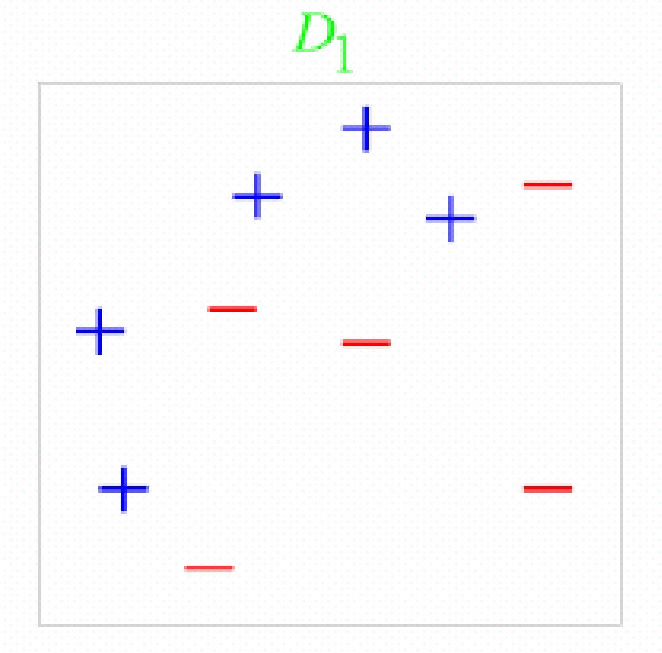
--- Use different modalities(features)

--- Focus on different subproblems

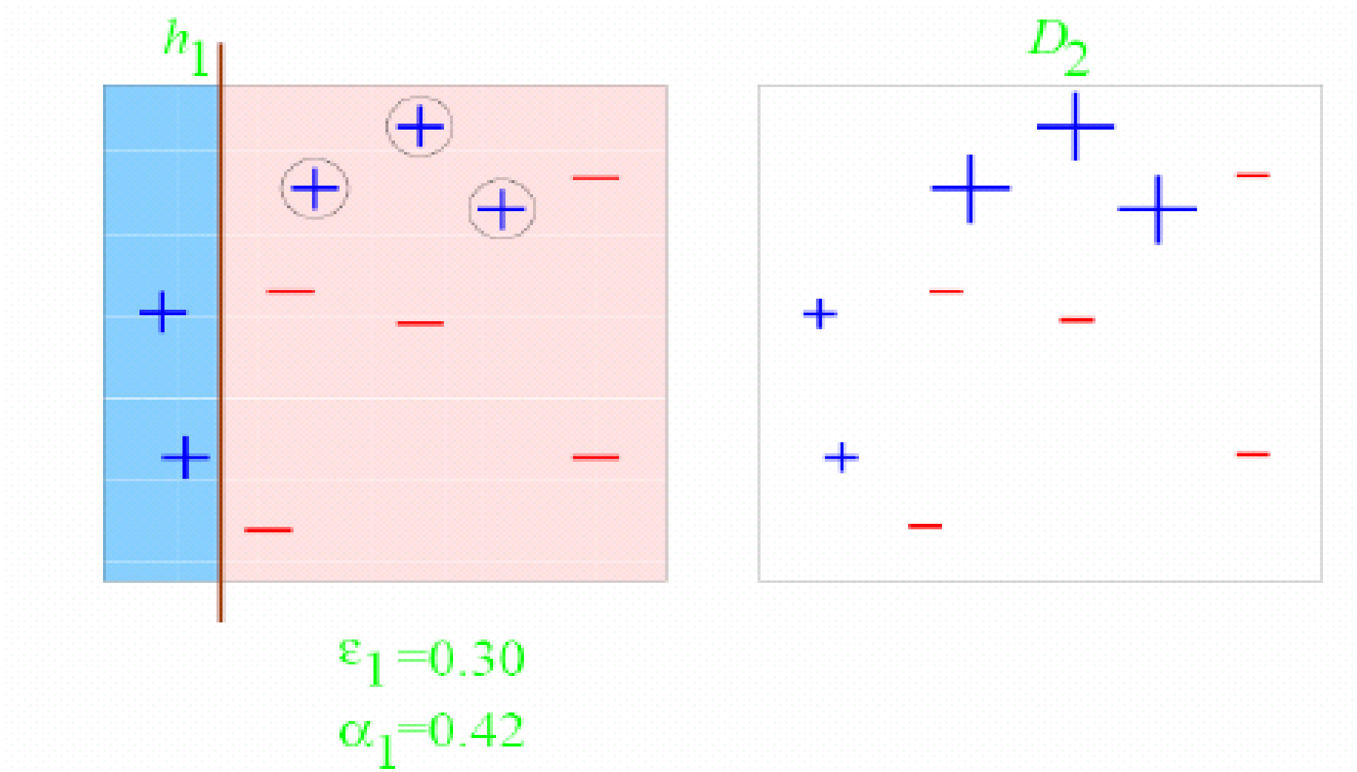
--- ……

4) A week learner is “rough and moderately inaccurate” predictor but one that can predict better than chance.

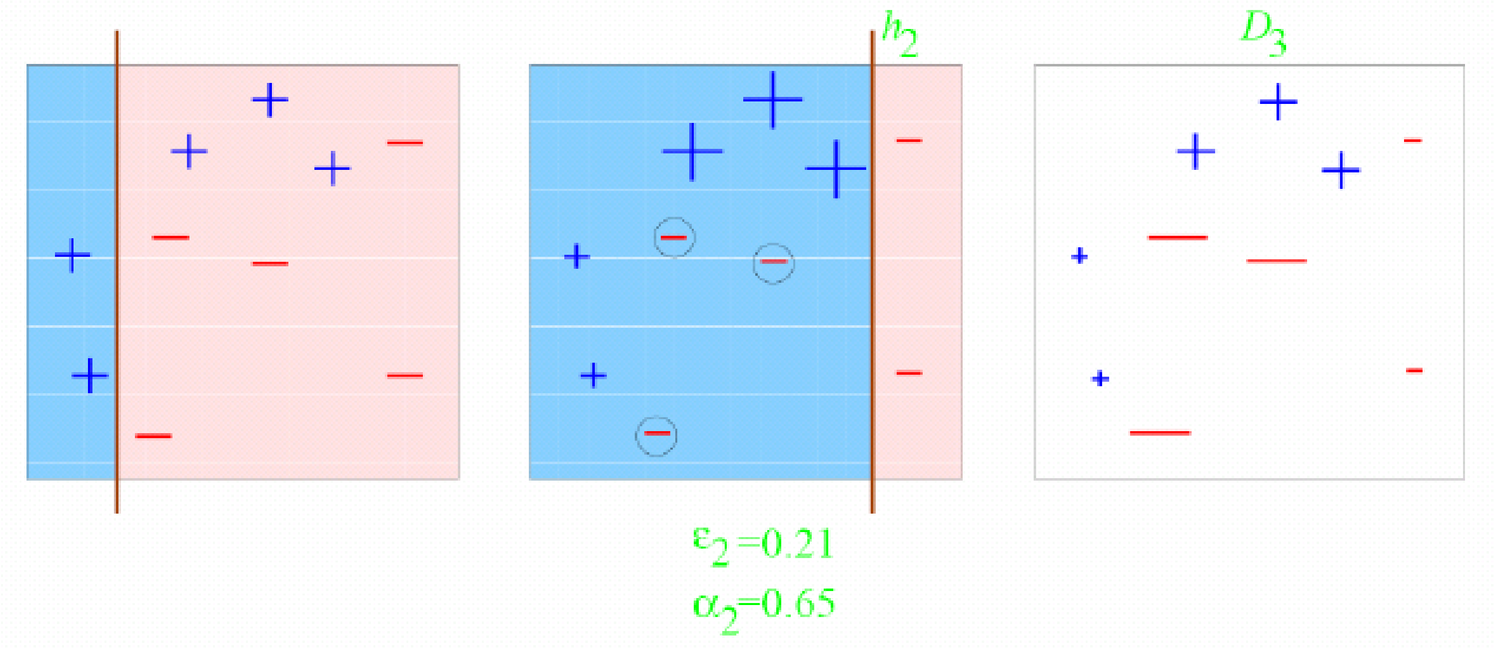
A toy example



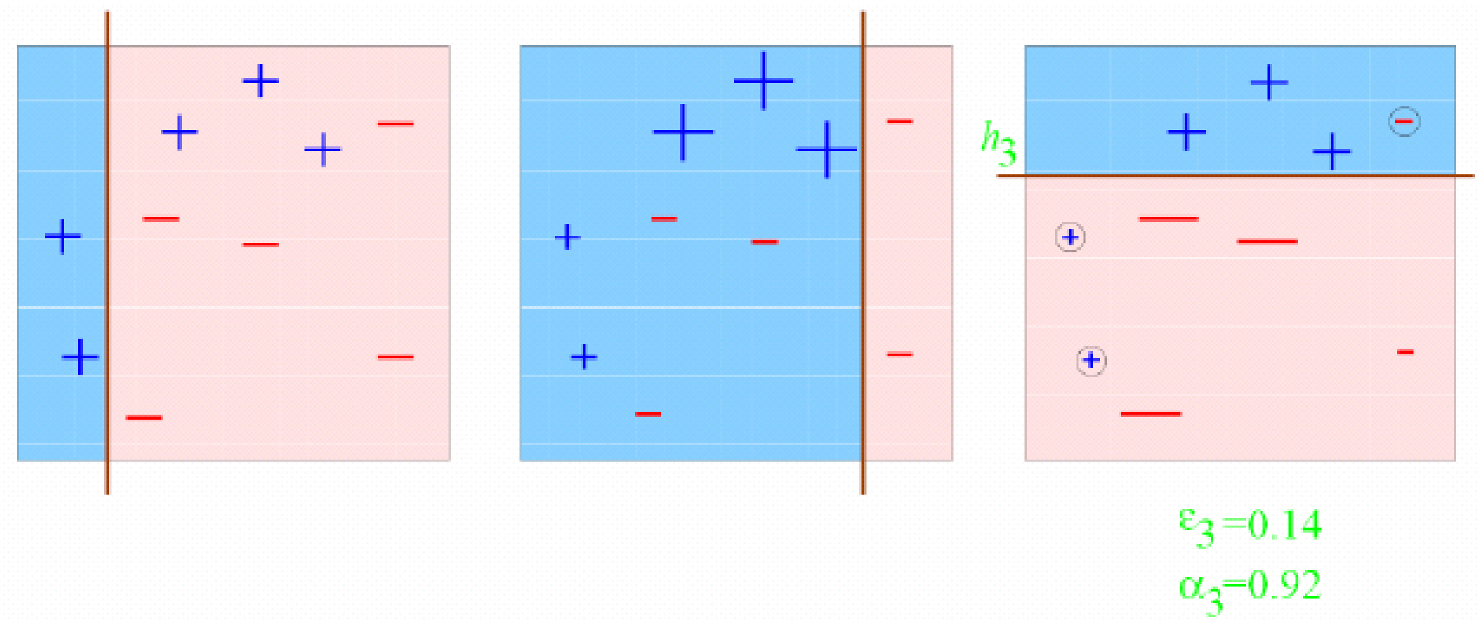
A toy example(cont'd)



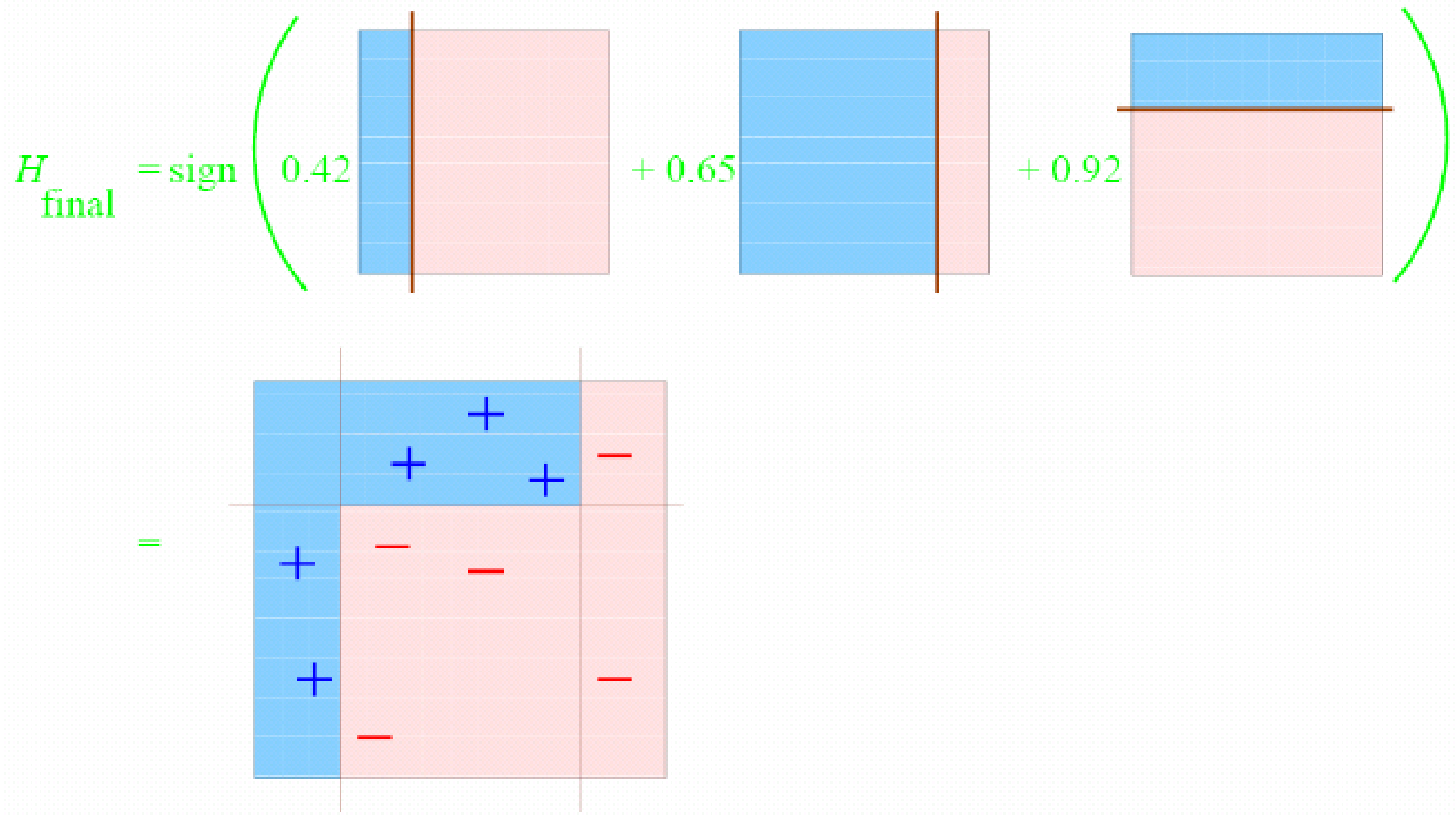
A toy example(cont'd)



A toy example(cont'd)



A toy example(cont'd)



Advantages of Bagging & Boosting

Easy to implement without additional information

- For Bagging : variance reduction, where

$$Var(\hat{c}) = E\left[\left(\hat{c}(x) - E[\hat{c}(x)]\right)^2\right]$$

- For Boosting : variance and bias reduction, where

$$Bias(\hat{c}) = E[\hat{c}(x)] - c(x)$$

- For Boosting : no overfitting

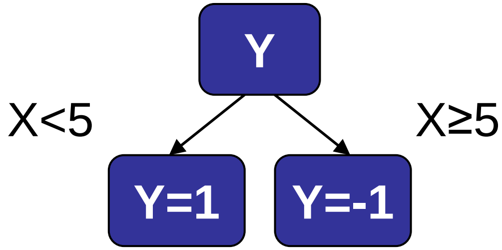
Choose an Unstable Classifier for Bagging

- Changes in the dataset produces big changes in the predictor
  - E.g. : neural networks, classification and regression trees
  - E.g. of stable classifiers: K-nearest neighbours

Reference: BAGGING PREDICTORS
L.BREIMAN, MACHINE LEARNING, 26(2), P.123-140, 1996

Choose a Weak Classifier for Boosting

- A classifier that performs ***slightly better*** than random guessing
- Too weak classifiers do not provide good results
- ***E.g. : classification into 2 classes:***
  - **Random guessing** : error rate of 50%
  - **Weak classifier:** error rate close to 50% (≅45%)
- ***Stumps*** are appropriate weak classifiers (binary trees with 2 terminal nodes)



Low variance

High Bias

Adaboost with trees is ***"the best off-the-shelf classifier in the world"***

(Breiman, 1996)

Reference: ADDITIVE LOGISTIC REGRESSION: A STATISTICAL VIEW OF BOOSTING J.FRIEDMAN, T.HASTIE & R.TIBSHIRANI, THE ANNALS OF STATISTICS, 28(2):337-407, 2000

Learning to Detect Faces

A Large-Scale Application of Machine Learning

(This material is not in the text: for further information see the paper by
P. Viola and M. Jones, International Journal of Computer Vision, 2004

Viola-Jones Face Detection Algorithm

- Overview :
  - Viola Jones technique overview
  - Features
  - Integral Images
  - Feature Extraction
  - Weak Classifiers
  - Boosting and classifier evaluation
  - Cascade of boosted classifiers
  - Example Results

Viola Jones Technique Overview

- Three major contributions/phases of the algorithm :
  - Feature extraction
  - Learning using boosting and decision stumps
  - Multi-scale detection algorithm

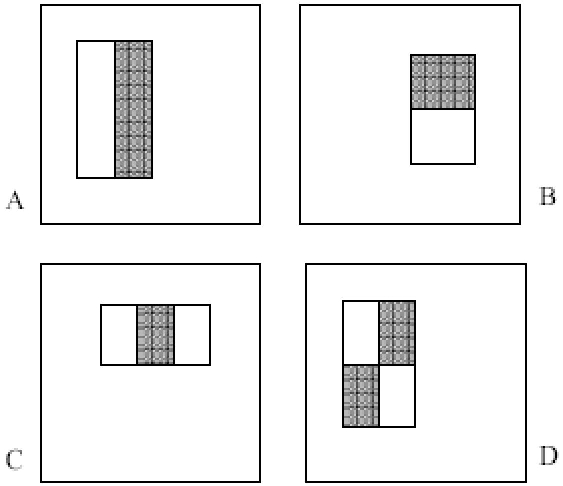
- Feature extraction and feature evaluation.
  - Rectangular features are used, with a new image representation their calculation is very fast.

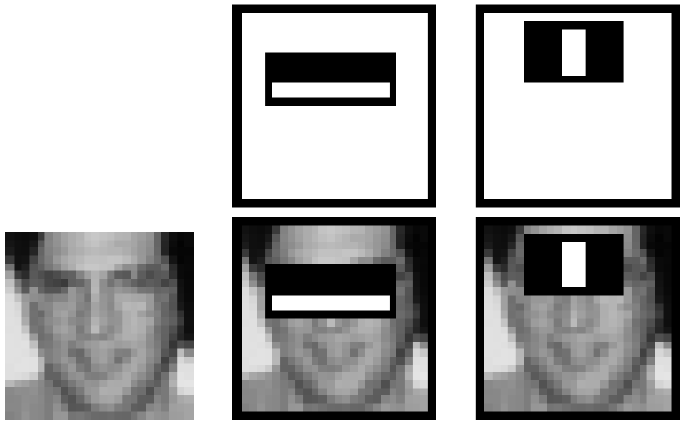
- Classifier learning using a method called boosting

- A combination of simple classifiers is very effective

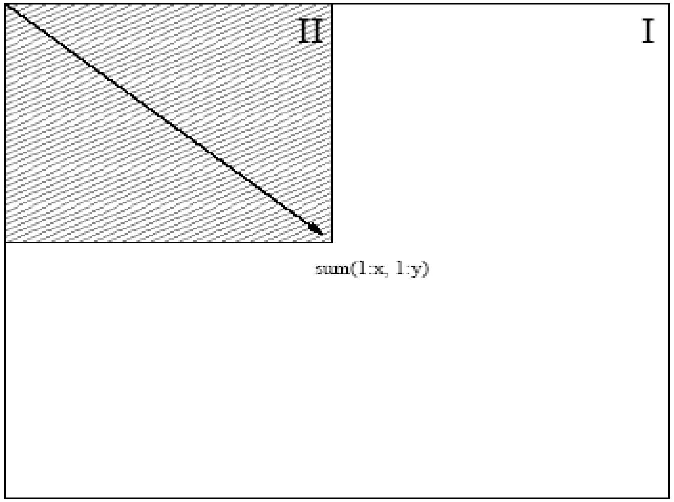
Features

- Four basic types.
  - They are easy to calculate.
  - The white areas are subtracted from the black ones.
  - A special representation of the sample called the **integral image** makes feature extraction faster.





Integral images



- Summed area tables

A representation that means any rectangle's values can be calculated in four accesses of the integral image.

Fast Computation of Pixel Sums

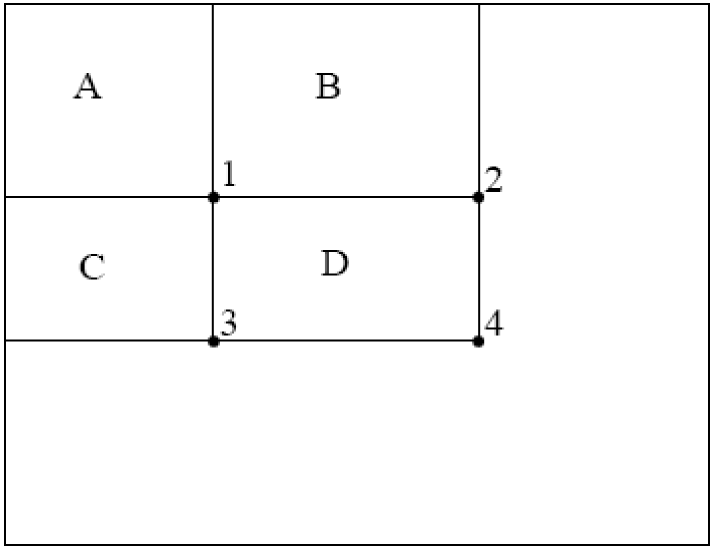
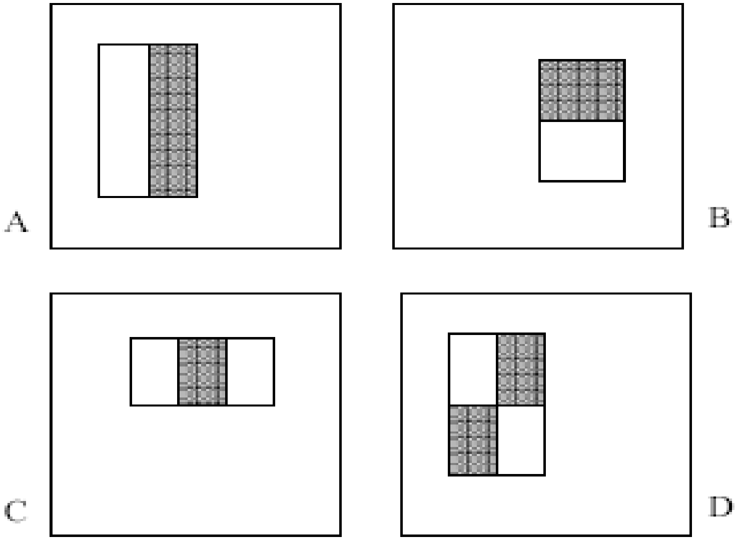


Figure 3: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

Feature Extraction

- Features are extracted from sub windows of a sample image.
  - The base size for a sub window is 24 by 24 pixels.
  - Each of the four feature types are scaled and shifted across all possible combinations
    - In a 24 pixel by 24 pixel sub window there are ~160,000 possible features to be calculated.



Learning with many features

- We have 160,000 features – how can we learn a classifier with only a few hundred training examples without overfitting?

- Idea:
  - Learn a single very simple classifier (a "weak classifier")
  - Classify the data
  - Look at where it makes errors
  - Reweight the data so that the inputs where we made errors get higher weight in the learning process
  - Now learn a 2^{nd} simple classifier on the weighted data
  - Combine the 1^{st} and 2^{nd} classifier and weight the data according to where they make errors
  - Learn a 3^{rd} classifier on the weighted data
  - … and so on until we learn T simple classifiers
  - Final classifier is the combination of all T classifiers

"Decision Stumps"

- Decision stumps = decision tree with only a single root node
  - Certainly a very **weak** learner!
  - Say the attributes are **real-valued**
  - Decision stump algorithm looks at **all possible thresholds** for each attribute
  - Selects the **one with the max information gain**
  - Resulting classifier is a simple threshold on a single feature
    - Outputs a +1 if the attribute is above a certain threshold
    - Outputs a -1 if the attribute is below the threshold
  - Note: can restrict the search for to the n-1 "midpoint" locations between a sorted list of attribute values for each feature. So complexity is n log n per attribute.
  - Note this is exactly equivalent to learning a perceptron with a single intercept term (so we could also learn these stumps via gradient descent and mean squared error)

Boosting with Decision Stumps

- Viola-Jones algorithm
  - With K attributes (e.g., K = 160,000) we have 160,000 different decision stumps to choose from

  - At each stage of boosting
    - given reweighted data from previous stage
    - Train all K (160,000) single-feature perceptrons
    - Select the single best classifier at this stage
    - Combine it with the other previously selected classifiers
    - Reweight the data
    - Learn all K classifiers again, select the best, combine, reweight
    - Repeat until you have T classifiers selected

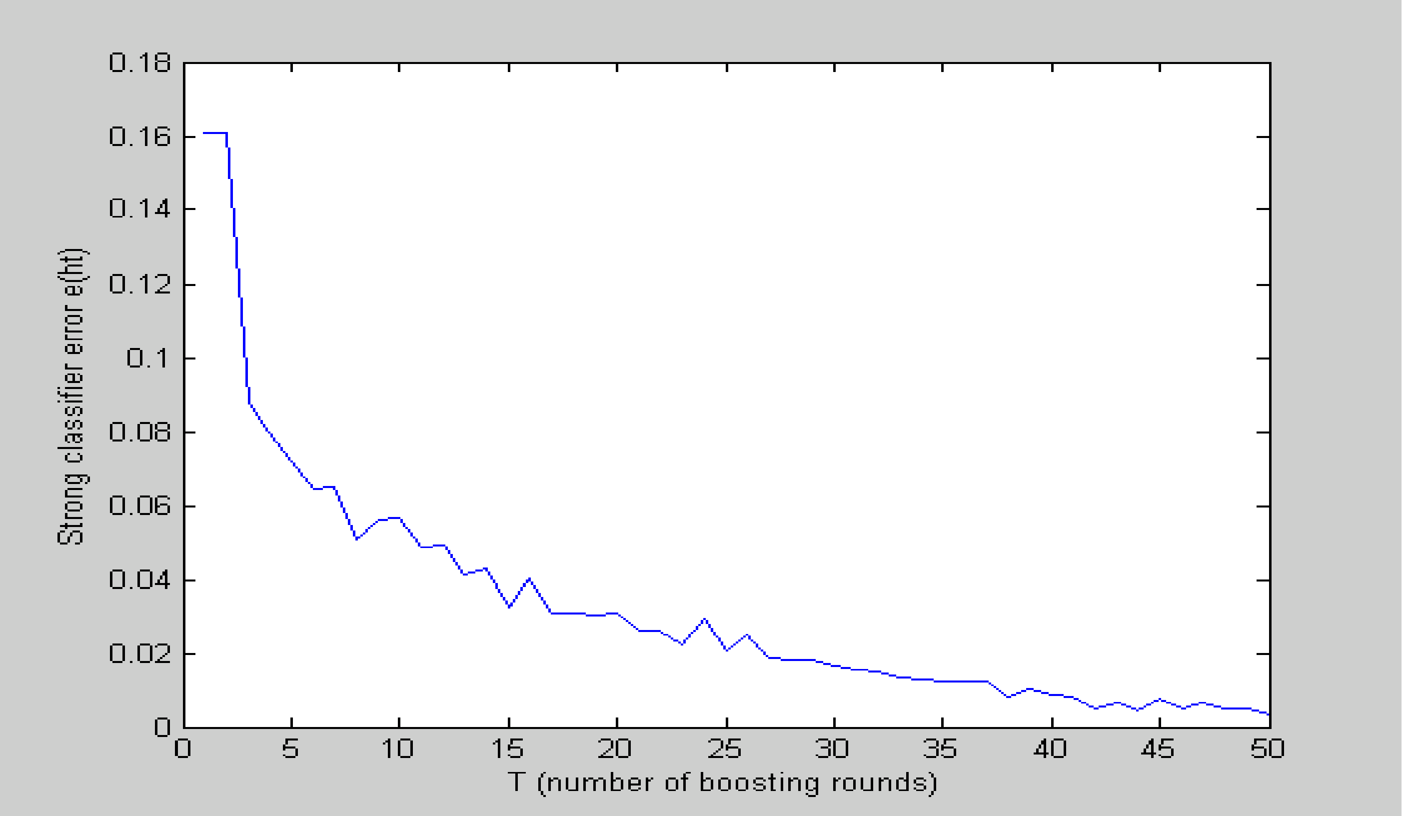
  - Very computationally intensive
    - Learning K decision stumps T times
    - E.g., K = 160,000 and T = 1000

How is classifier combining done?

- At each stage we select the best classifier on the current iteration and combine it with the set of classifiers learned so far

- How are the classifiers combined?
  - Take the weight*feature for each classifier, sum these up, and compare to a threshold (very simple)
  - Boosting algorithm automatically provides the appropriate weight for each classifier and the threshold
  - This version of boosting is known as the AdaBoost algorithm
  - Some nice mathematical theory shows that it is in fact a very powerful machine learning technique

Reduction in Error as Boosting adds Classifiers



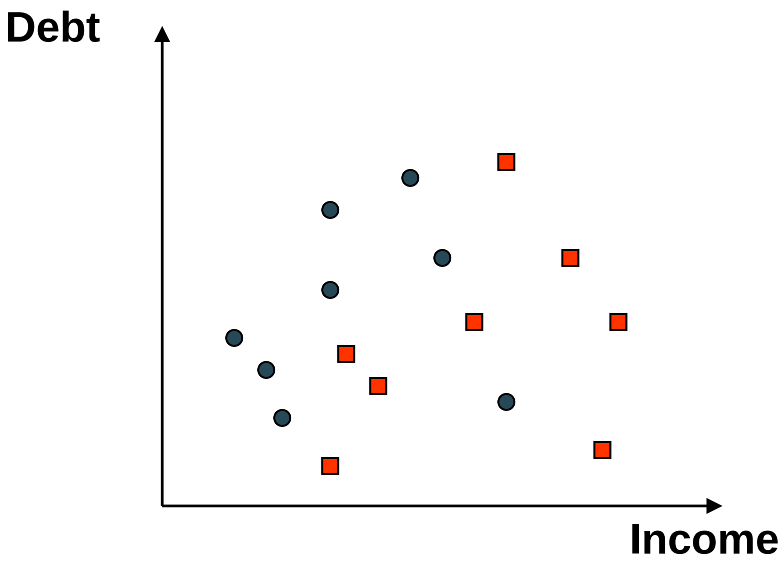
Induction by Decission Trees

Ramon Baldrich

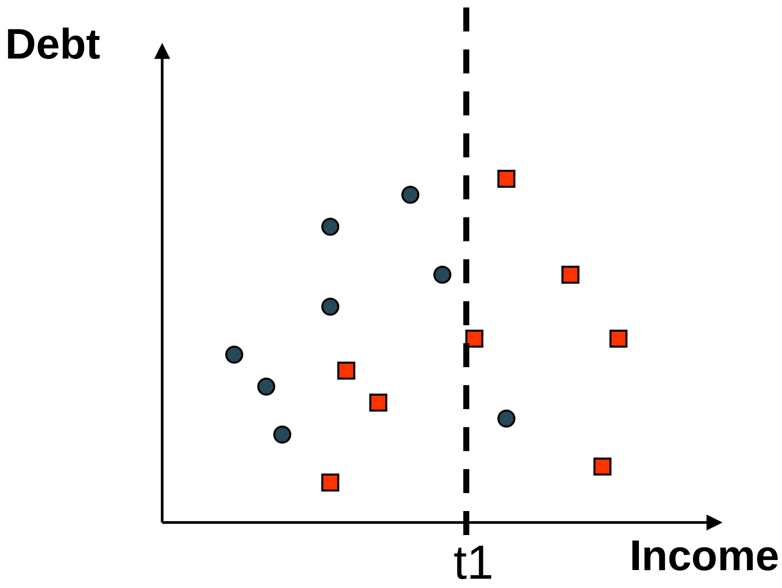
Universitat Autònoma de Barcelona

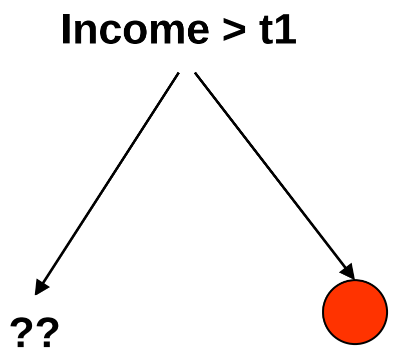
http://www.cs.utexas.edu/~ear/nsc110/ScienceAndSociety/Lectures/AI-long.ppt
http://www.cs.utexas.edu/users/ear/nsc110/Mirrors/DSMirrorsArtificialIntelligence.ppt
http://decisiontrees.net/decision-trees-tutorial/

Decision Tree Example

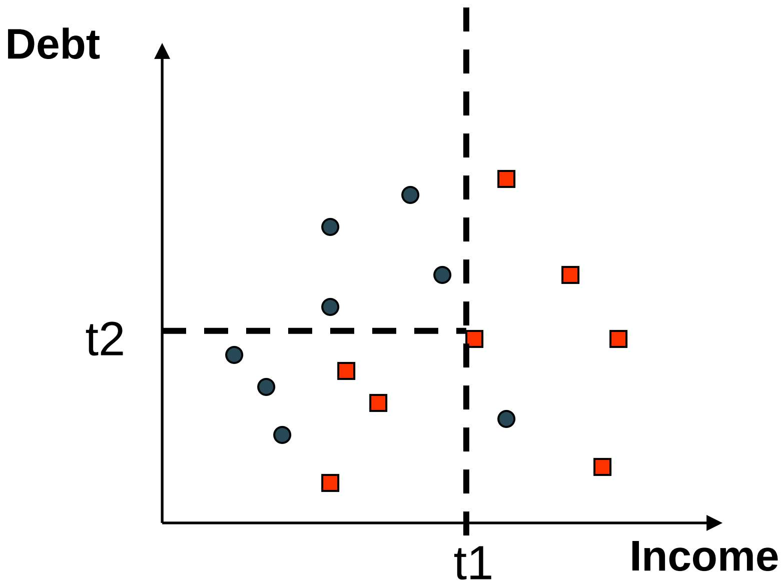


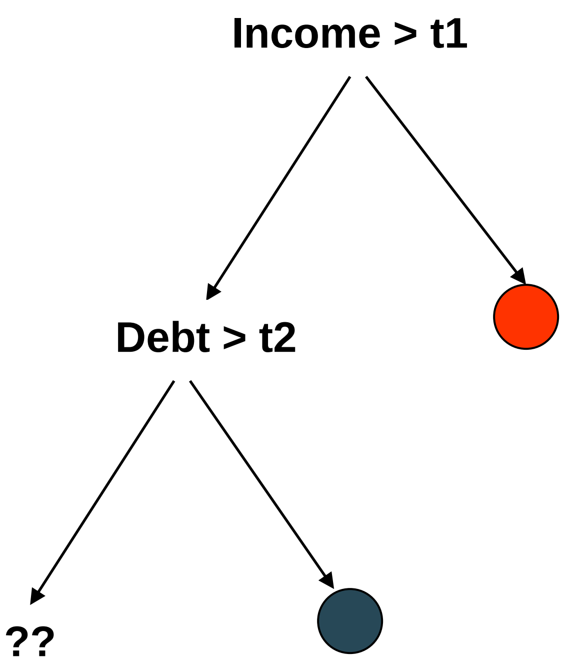
Decision Tree Example



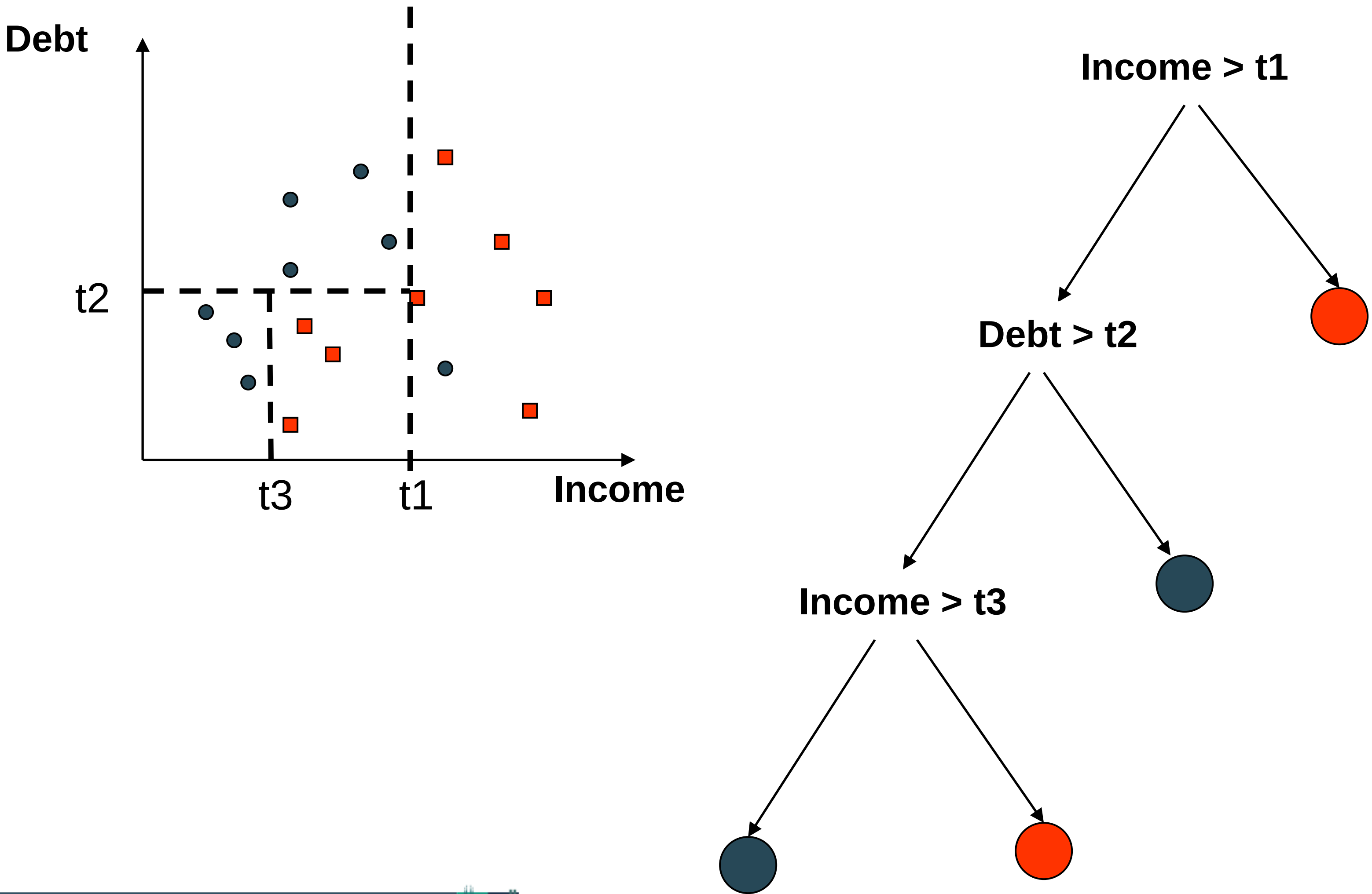


Decision Tree Example

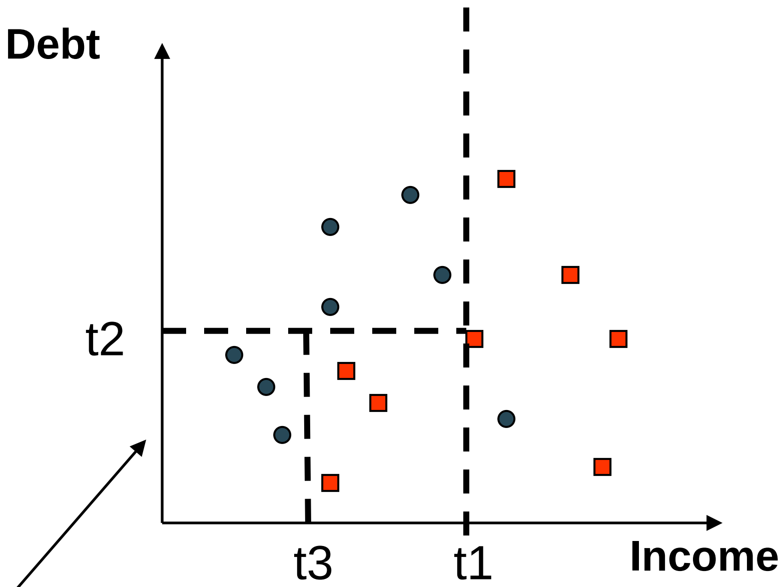


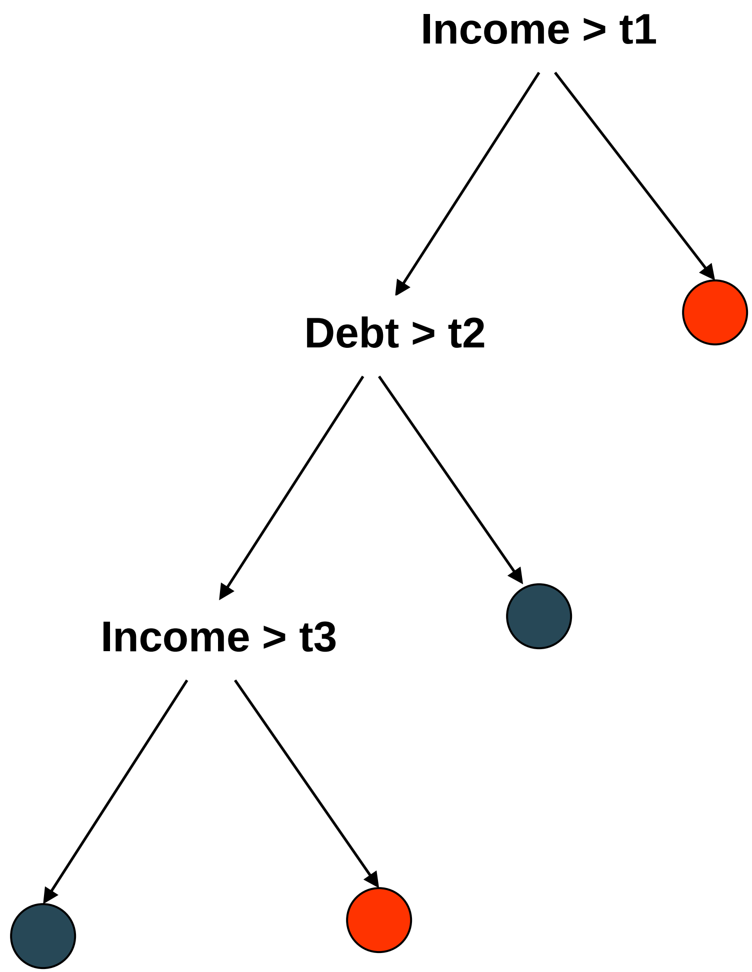


Decision Tree Example



Decision Tree Example





Note: tree boundaries are linear and axis-parallel

Random Forest

http://www.cs.utexas.edu/~ear/nsc110/ScienceAndSociety/Lectures/AI-long.ppt
http://www.cs.utexas.edu/users/ear/nsc110/Mirrors/DSMirrorsArtificialIntelligence.ppt
http://decisiontrees.net/decision-trees-tutorial/

Random Forests

- ‘Ensemble method’ specific for decision tree classifiers
- Random Forests grows lots of DTs
  - Not pruned DT
  - Each base classifier, predicts on all query samples
  - Final result classification: **voting**.

  The forest selects the most voted resulting classification (the most voted ambong all the trees)

Random Forests

Two randmon sources: “Bagging” & “random input vectors”

- Bagging Method: each tree grows using a ‘bootstrap’ sample from training data

- Random input vector: at each node, the best splitting atribute is selected among a random sample of size m of all the data atributes

Which m?

$$m = \frac{M}{3} \quad \text{if regresion}$$
$$m = \lfloor \sqrt{M} \rfloor \text{ if clasification}$$
$$m = \text{"}tunning\ parametre\text{"}$$

Random forest algorithm

- Let N the number of training samples, and M the number of data atributes
- Fix a number *m* of input variables to be used in the decision test in a tree node; *m* should be much lower than *M*.
- We choose a training sample for the tree, choosing n times with reposition among the N available training samples (i.e. bootstrap). Use the rest of samples to estimate the tree error when prdicting sample class.
- For each tree node, randomly choose *m* variables to decide wich is the best test atribute in that node. Get the best division from those *m* variables.
- Each tree grows without limit, no prunning.

Random Forests

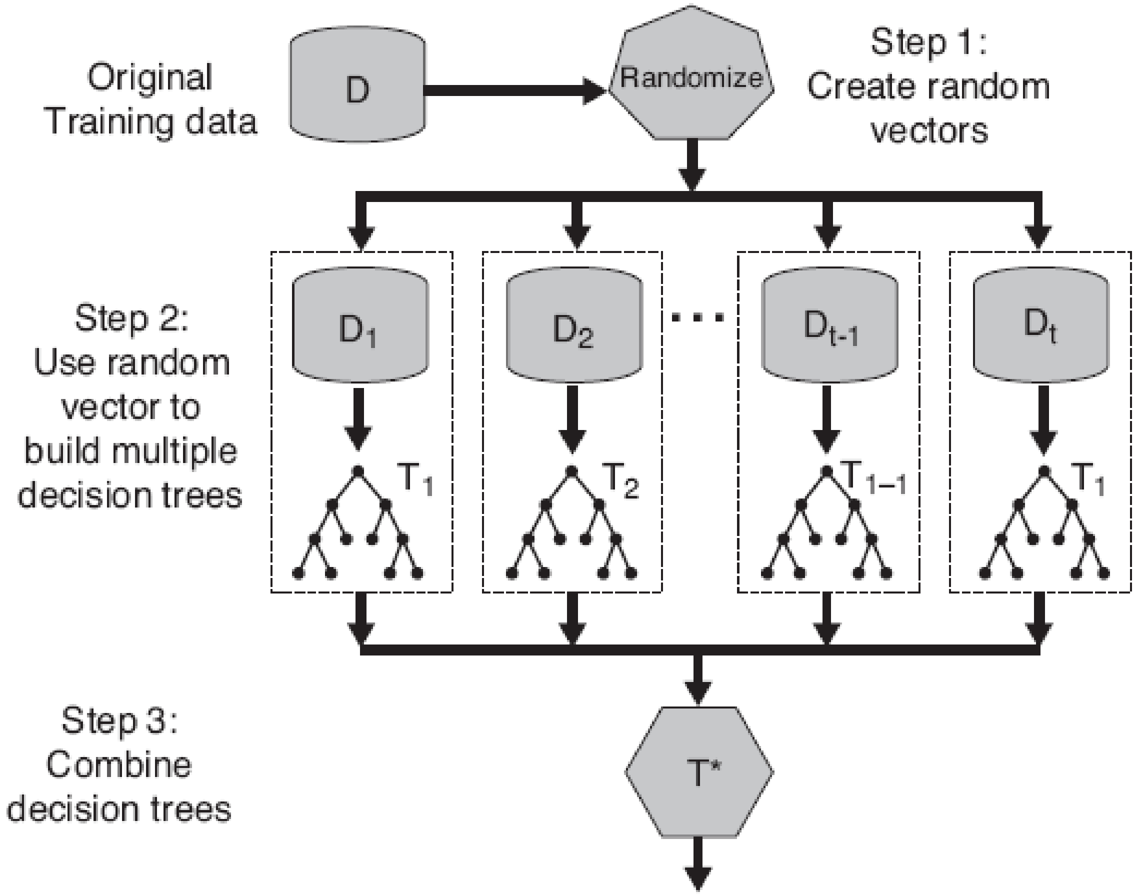
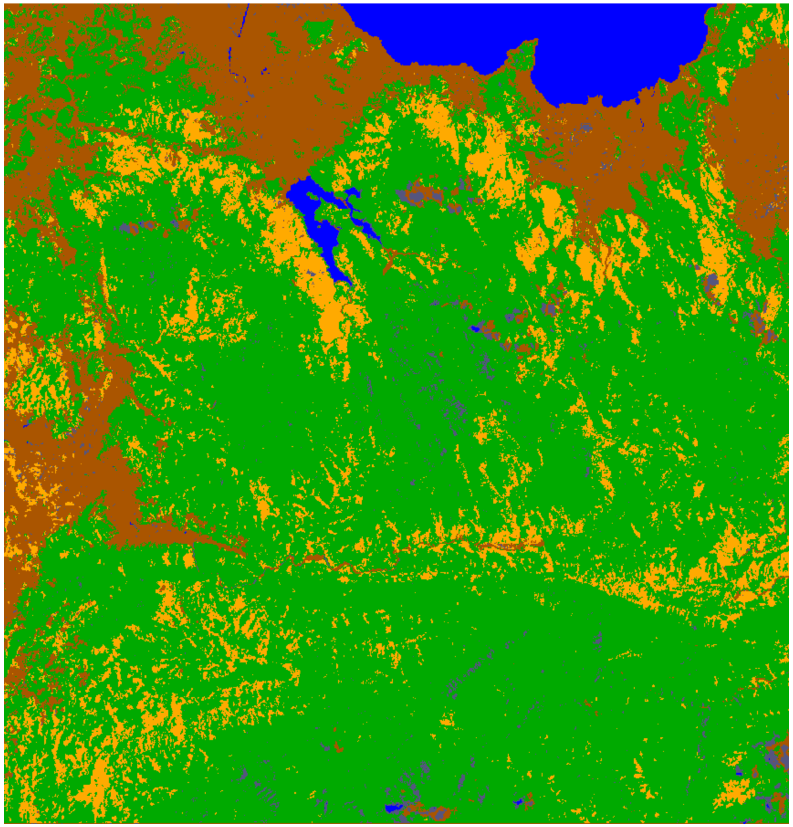


Figure 5.40. Random forests.

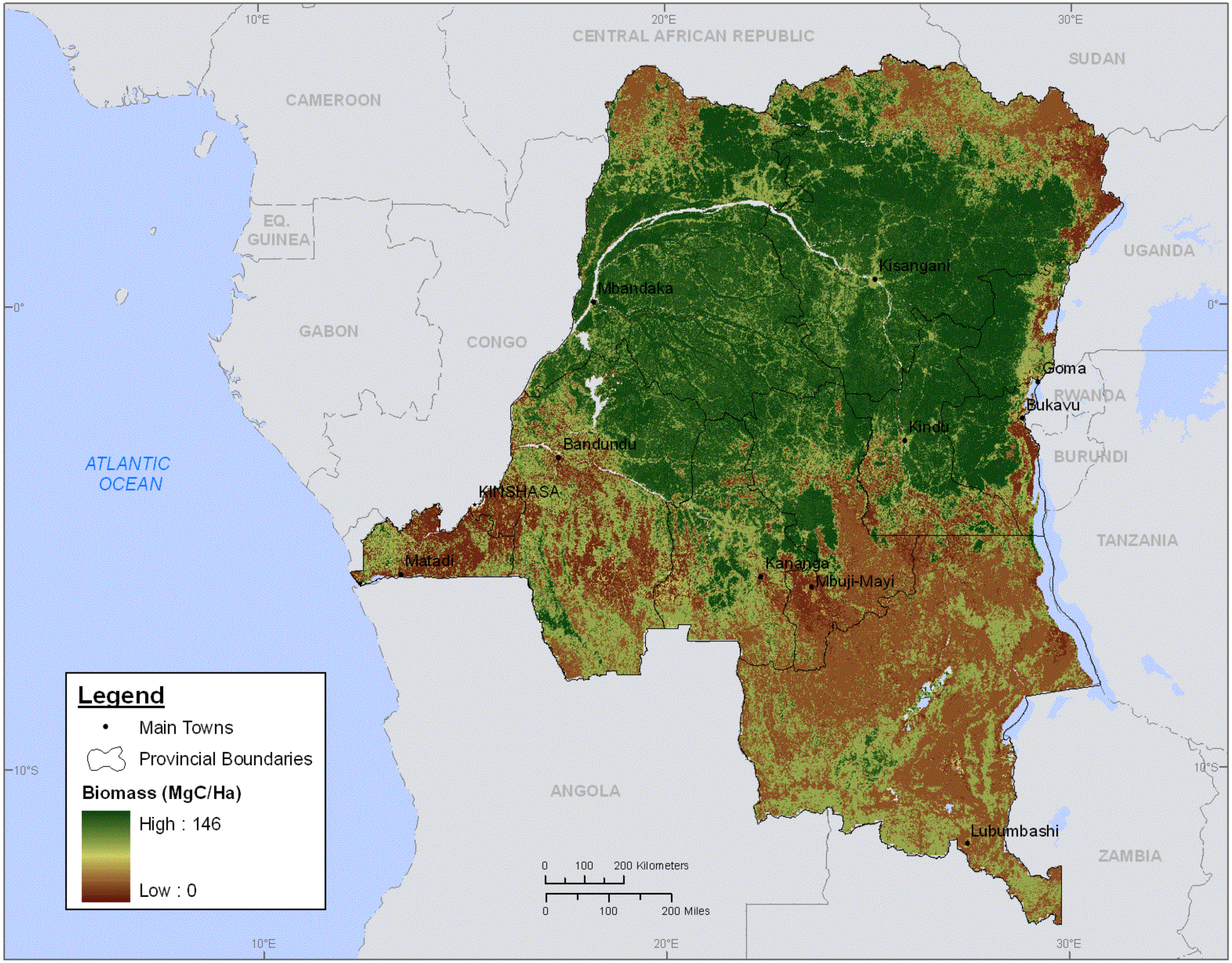
Arbre de decisió



Blue = water
Green = forest
Yellow = shrub
Brown = non-forest
Gray = cloud/shadow

Ned Horning
American Museum of Natural History's Center for Biodiversity and Conservation

Regressió



CENTRAL AFRICAN REPUBLIC
SUDAN
CAMEROON
EQ. GUINEA
GABON
CONGO
UGANDA
RWANDA
BURUNDI
TANZANIA
ZAMBIA
ANGOLA
ATLANTIC OCEAN
Kisangani
Mbandaka
Goma
Bukavu
Kindu
Bandundu
KINSHASA
Matadi
Kananga
Mbuji-Mayi
Lubumbashi
10°E
20°E
30°E
0°
10°S
0 100 200 Kilometers
0 100 200 Miles
Legend
Main Towns
Provincial Boundaries
Biomass (MgC/Ha)
High : 146
Low : 0

Efecte de la població del bosc

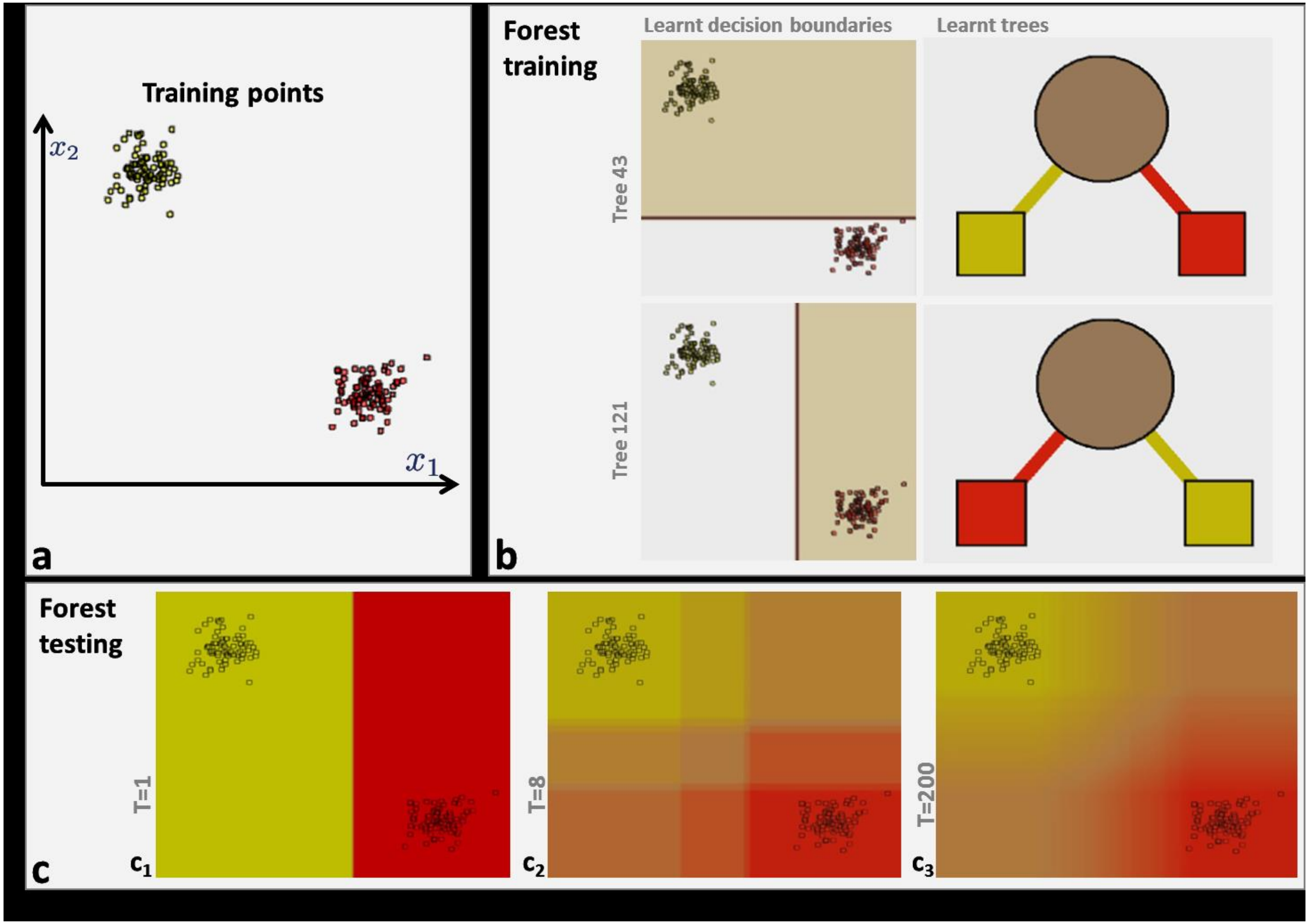


Fig. 3.3: A first classification forest and the effect of forest size T. (a) Training points belonging to two classes. (b) Different training trees produce different partitions and thus different leaf predictors. The colour of tree nodes and edges indicates the class probability of training points going through them. (c) In testing, increasing the forest size T produces smoother class posteriors. All experiments were run with D = 2 and axis-aligned weak learners. See text for details.

Random forest i problemes multiclase

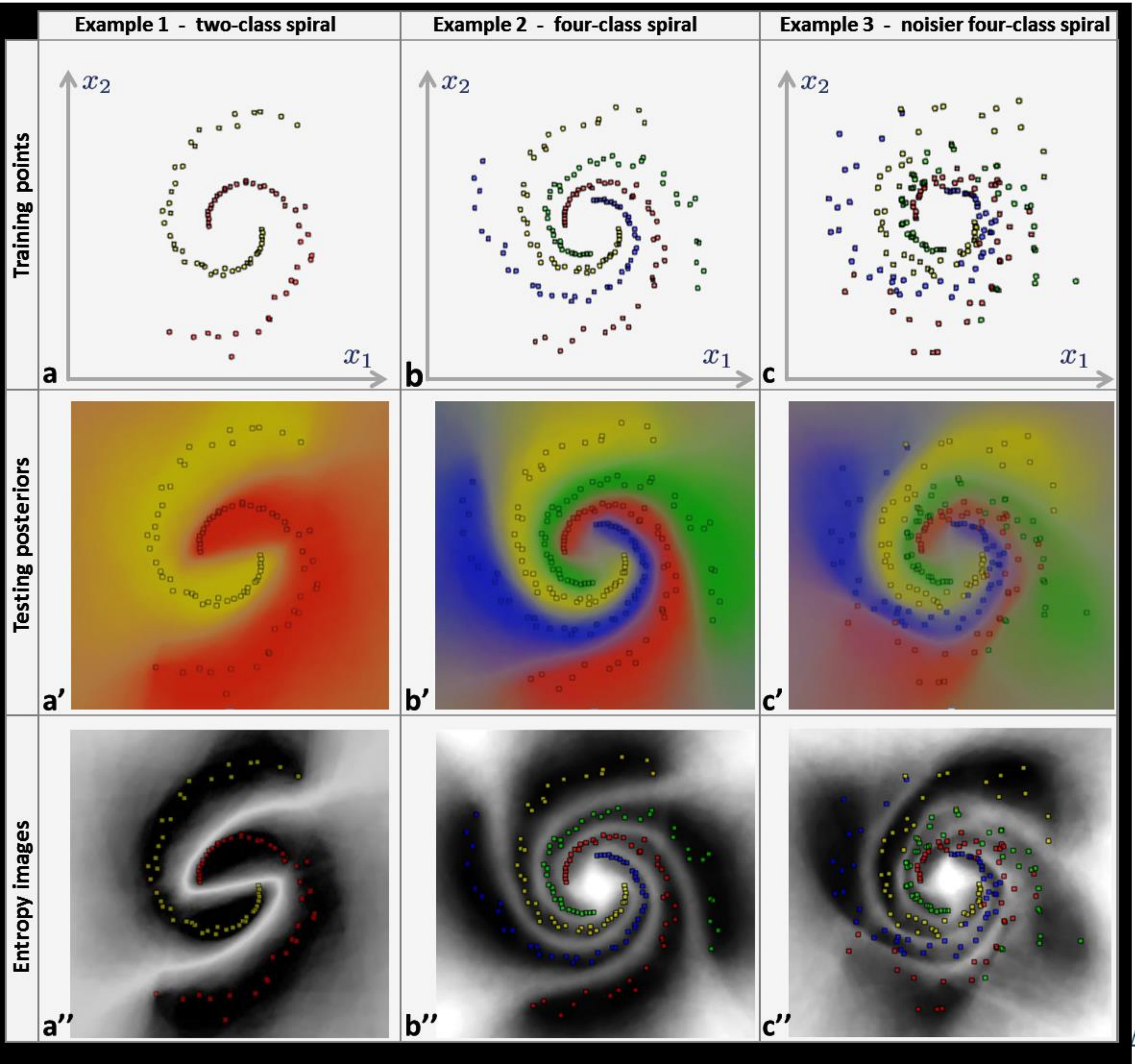


Fig. 3.4: The effect of multiple classes and noise in training data. (a,b,c) Training points for three different experiments: 2-class spiral, 4-class spiral and another 4-class spiral with noisier point positions, respectively. (a',b',c') Corresponding testing posteriors. (a",b",c") Corresponding entropy images (brighter for larger entropy). The classification forest can handle both binary as well as multiclass problems. With larger training noise the classification uncertainty increases (less saturated colours in c' and less sharp entropy in c"). All experiments in this figure were run with T = 200, D = 6, and a conic-section weak-learner model.