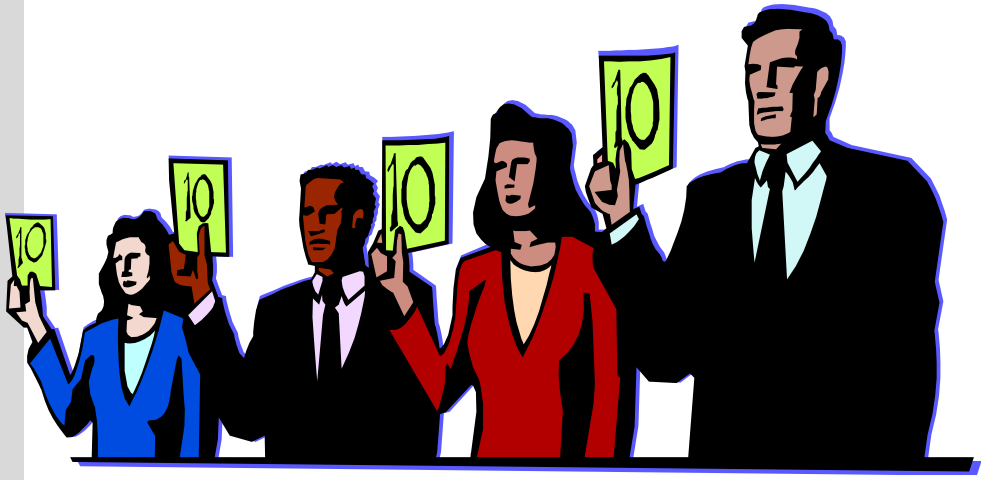
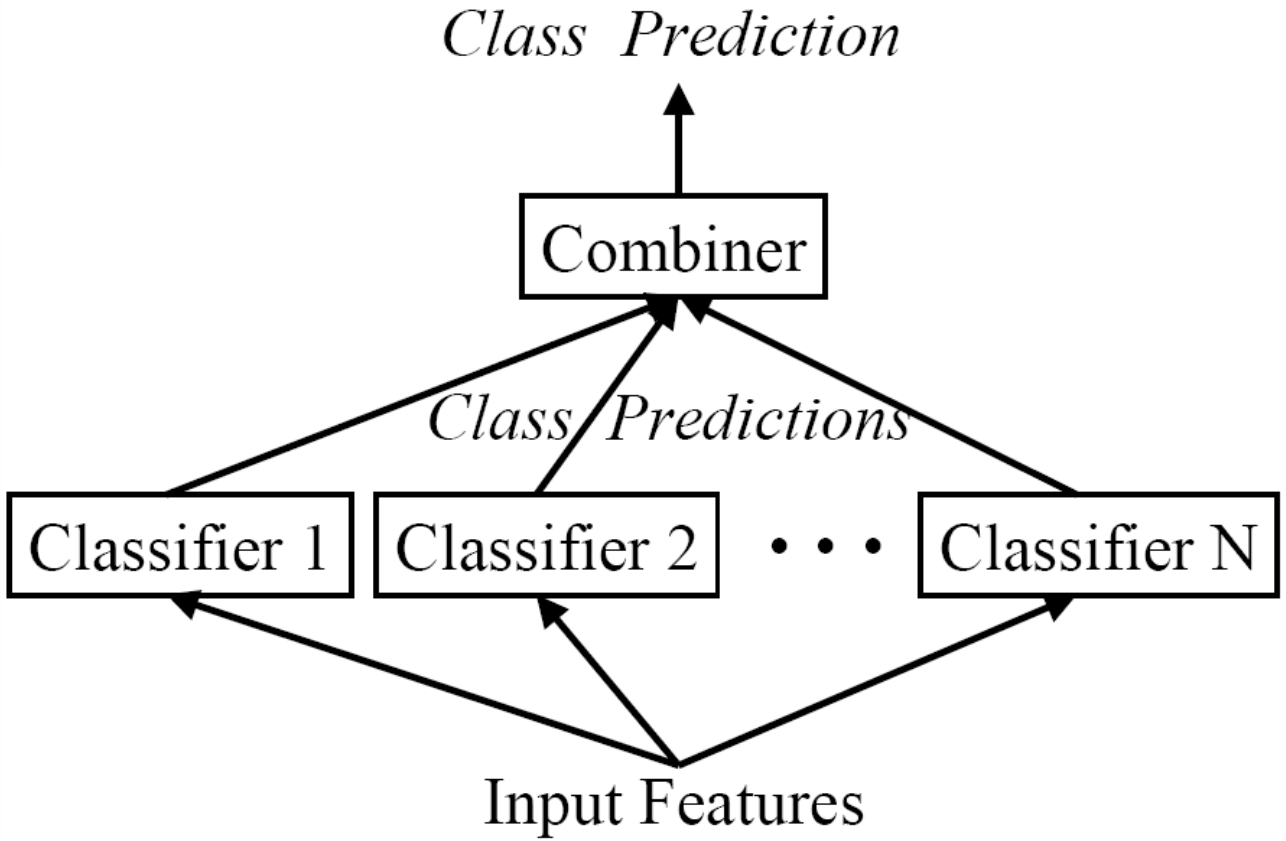
Ensemble Learning

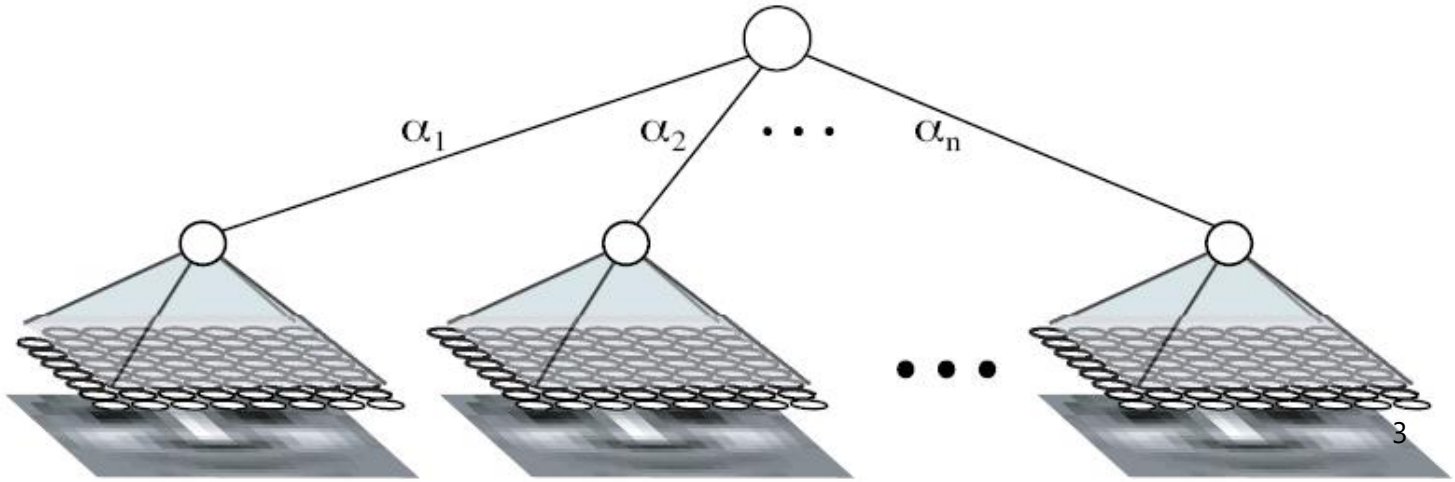
B_3S = Bootstrap, Bagging, Boosting, Stacking

Real World Scenarios

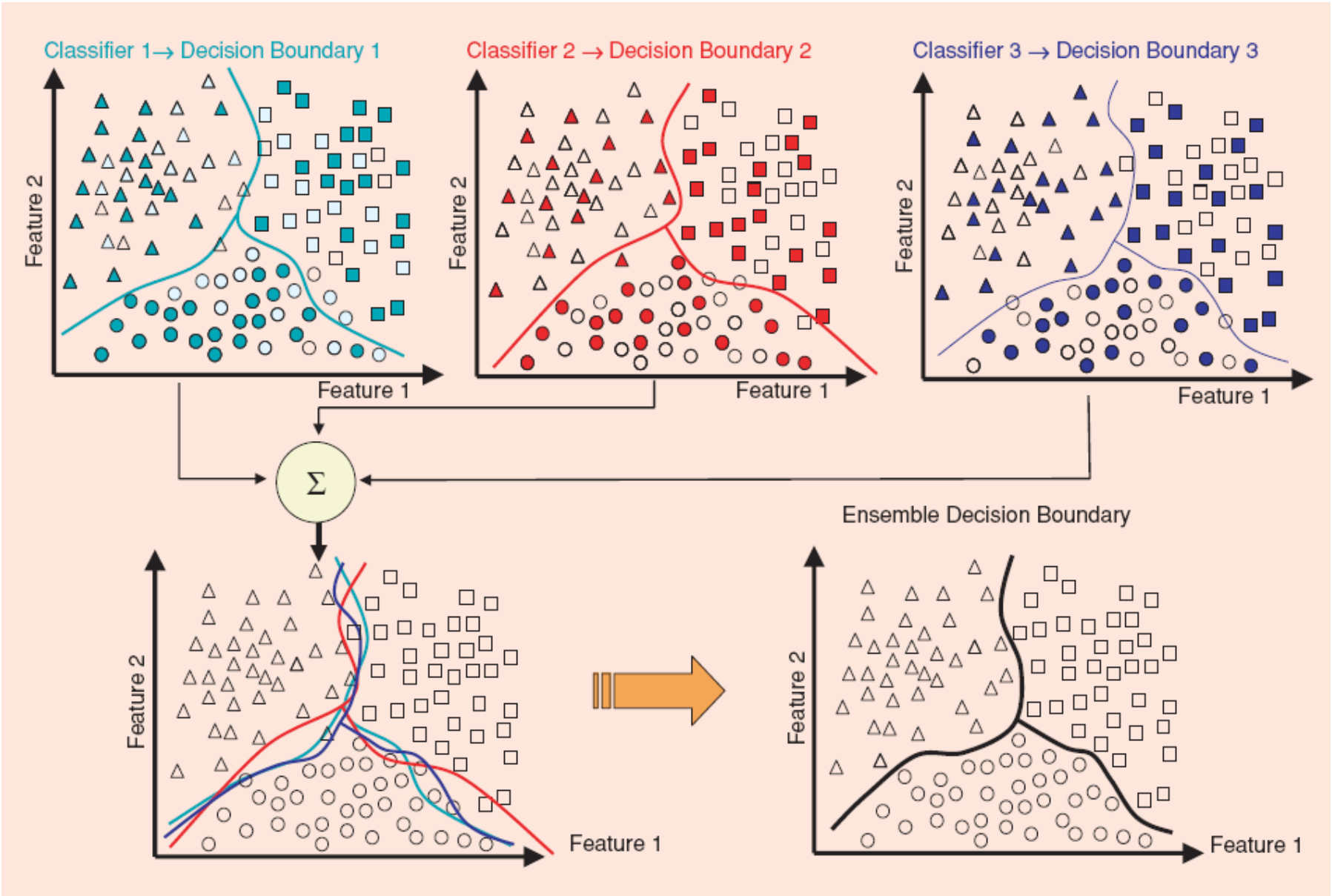


Combination of Classifiers

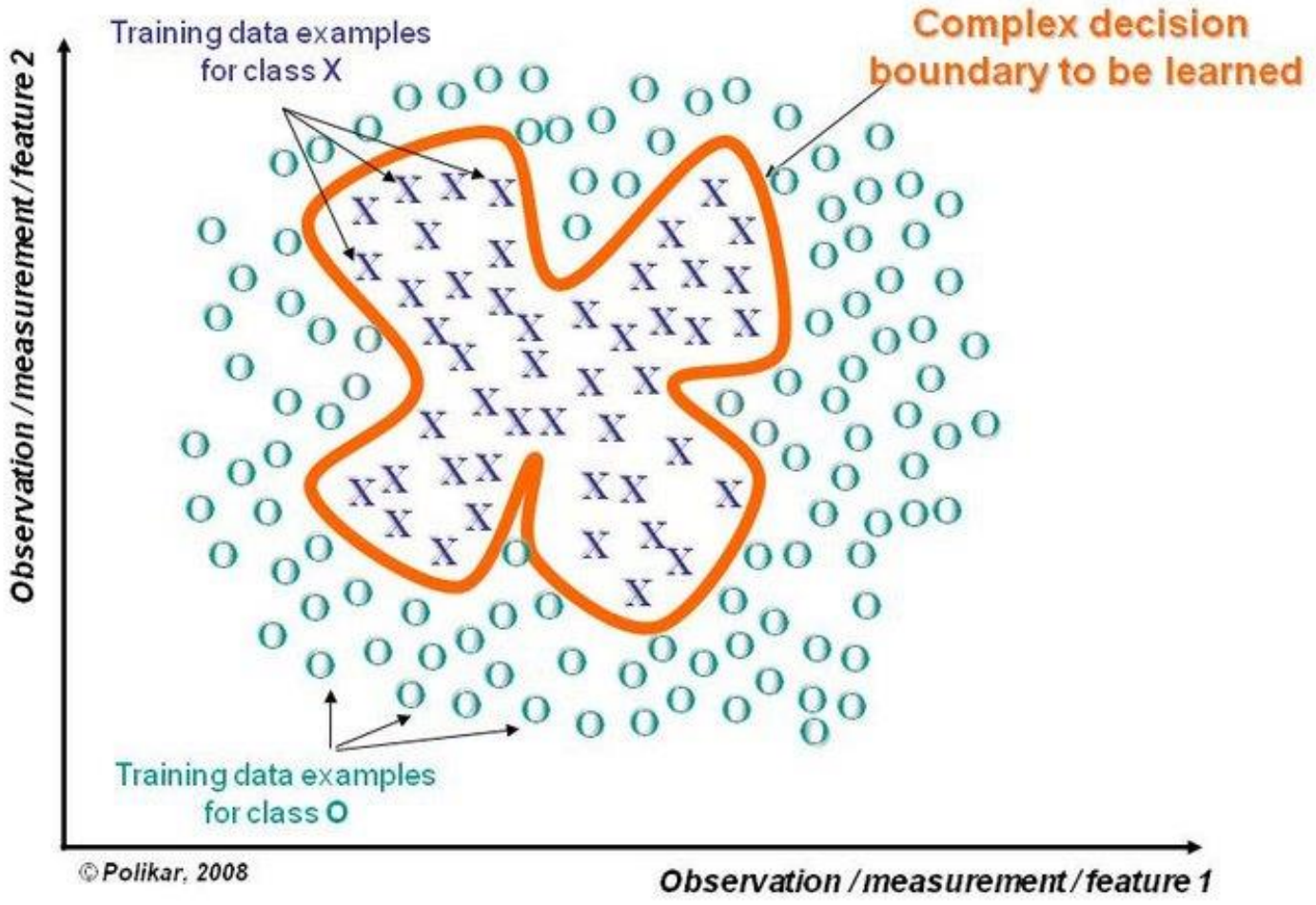




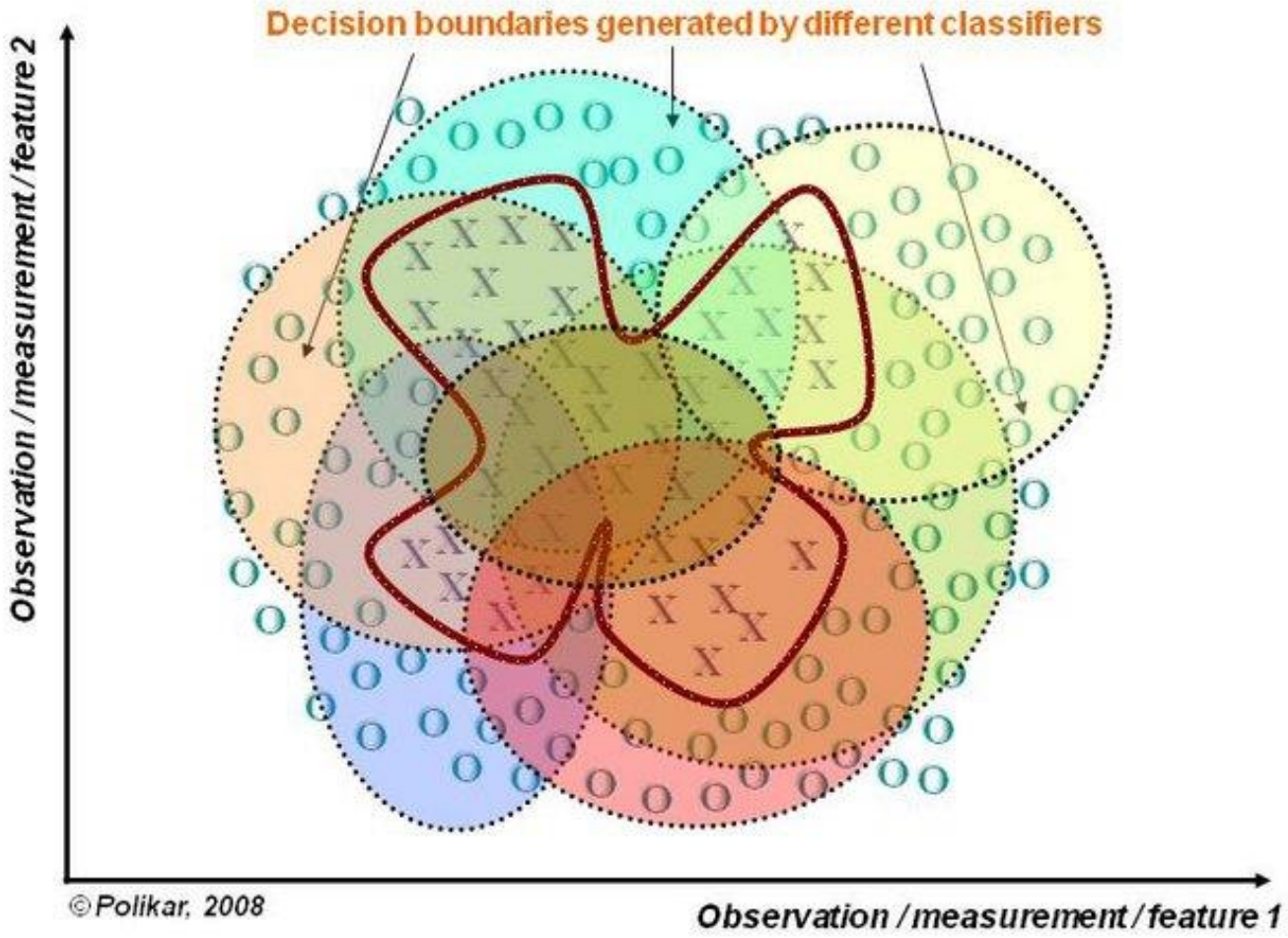
Model Selection



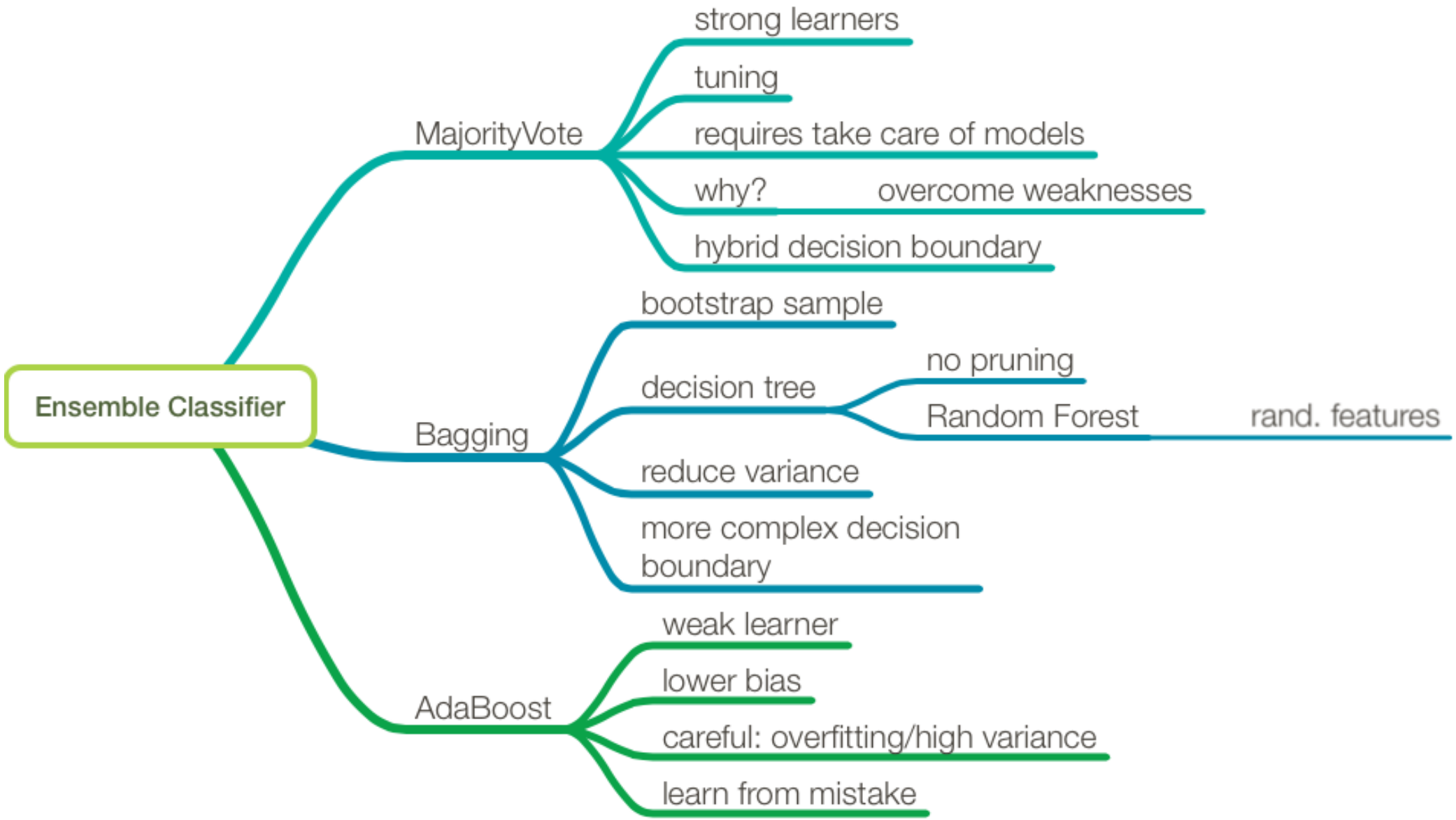
Divide and Conquer



Divide and Conquer



Ensample Classifier - summary



Diversity

The key to the success of ensemble learning

- Need to correct the errors made by other classifiers.
- Does not work if all models are identical.

Different Learning Algorithms

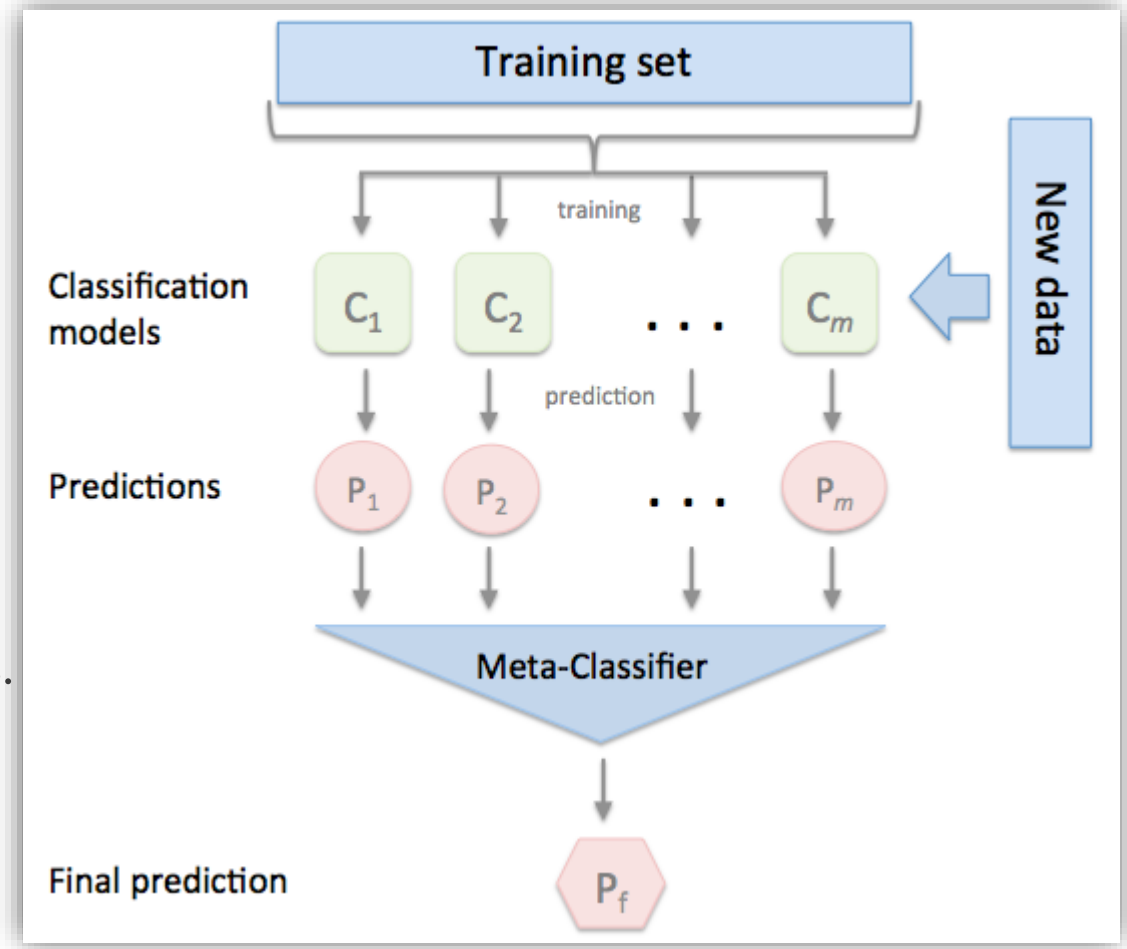
- DT, SVM, NN, KNN …

Different Training Processes

- Different Parameters
- Different Training Sets
- Different Feature Sets

Weak Learners

- Easy to create different decision boundaries.
- Stumps …



Combiners

How to combine the outputs of classifiers.

Averaging

Voting

- Majority Voting: Random Forest
- Weighted Majority Voting: AdaBoost

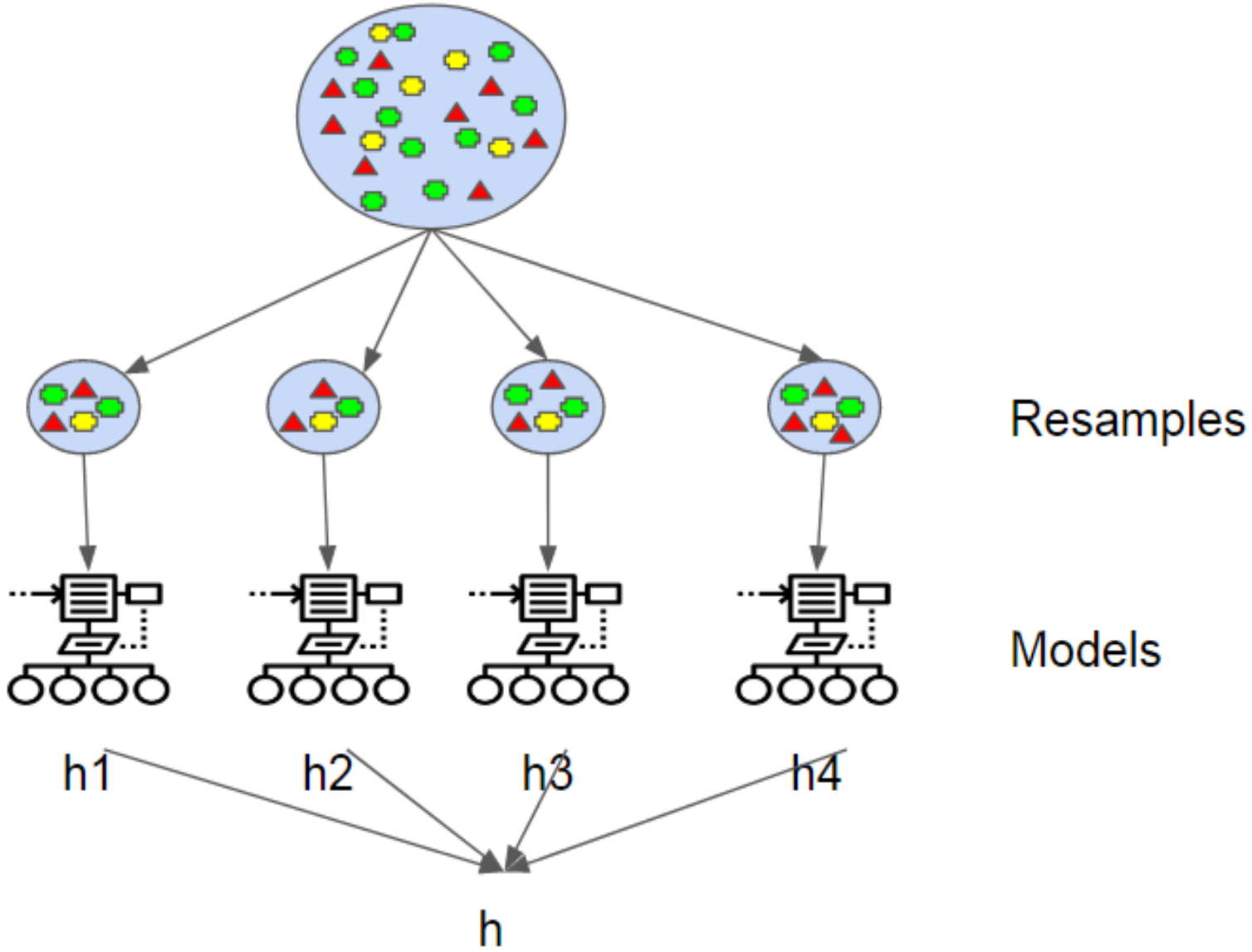
Learning Combiner

- General Combiner: Stacking
- Piecewise Combiner: RegionBoost

No Free Lunch

BAGGING = Bootstrap AGGregatING

The idea of bagging



Bagging Algorithm

Input:

- Training data S with correct labels $\omega_i \Omega = \{\omega_1,...,\omega_C\}$ representing C classes
- Weak learning algorithm **WeakLearn,**
- Integer T specifying number of iterations.
- Percent (or fraction) F to create bootstrapped training data

Do t=1, ..., T

1. Take a bootstrapped replica S_t by randomly drawing F percent of S.
2. Call **WeakLearn** with S_t and receive the hypothesis (classifier) h_t.
3. Add h_t to the ensemble, $\mathcal{E}$.

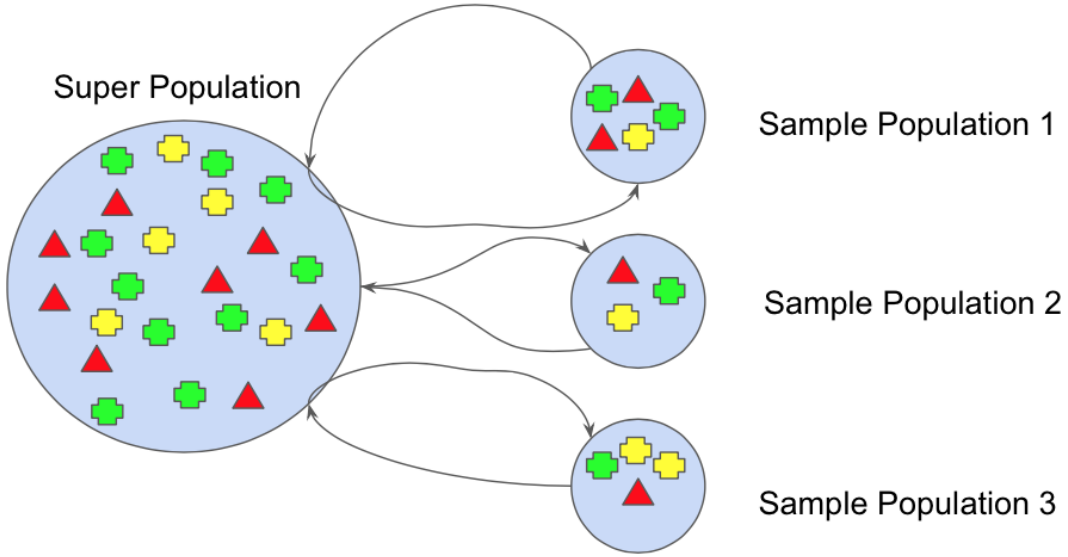
End

Test: Simple Majority Voting – Given unlabeled instance **x**

1. Evaluate the ensemble $\mathcal{E} = \{h_1,...,h_T\}$ on **x**.
2. Let $v_{t,j} = \begin{cases} 1, & \text{if } h_t \text{ picks class } \omega_j \\ 0, & \text{otherwise} \end{cases}$ be the vote given to class ω_j by classifier h_t.
3. Obtain total vote received by each class , $V_j = \sum_{t=1}^{T} v_{t,j}$ $j = 1,...,C$.
4. Choose the class that receives the highest total vote as the final classification.

Enter the Bootstrap

- In the late 70's the statistician Brad Efron made an ingenious suggestion.
- Most (sometimes all) of what we know about the "true" probability distribution comes from the data.
- So let's treat the data as a *proxy* for the true distribution.
- We draw multiple samples from this proxy…
- This is called "resampling".
- And compute the statistic of interest on each of the resulting pseudo-datasets.



Philosophy

- "[Bootstrapping has] requires very little in the way of modeling, assumptions, or analysis, and can be applied in an automatic way to any situation, no matter how complicated".

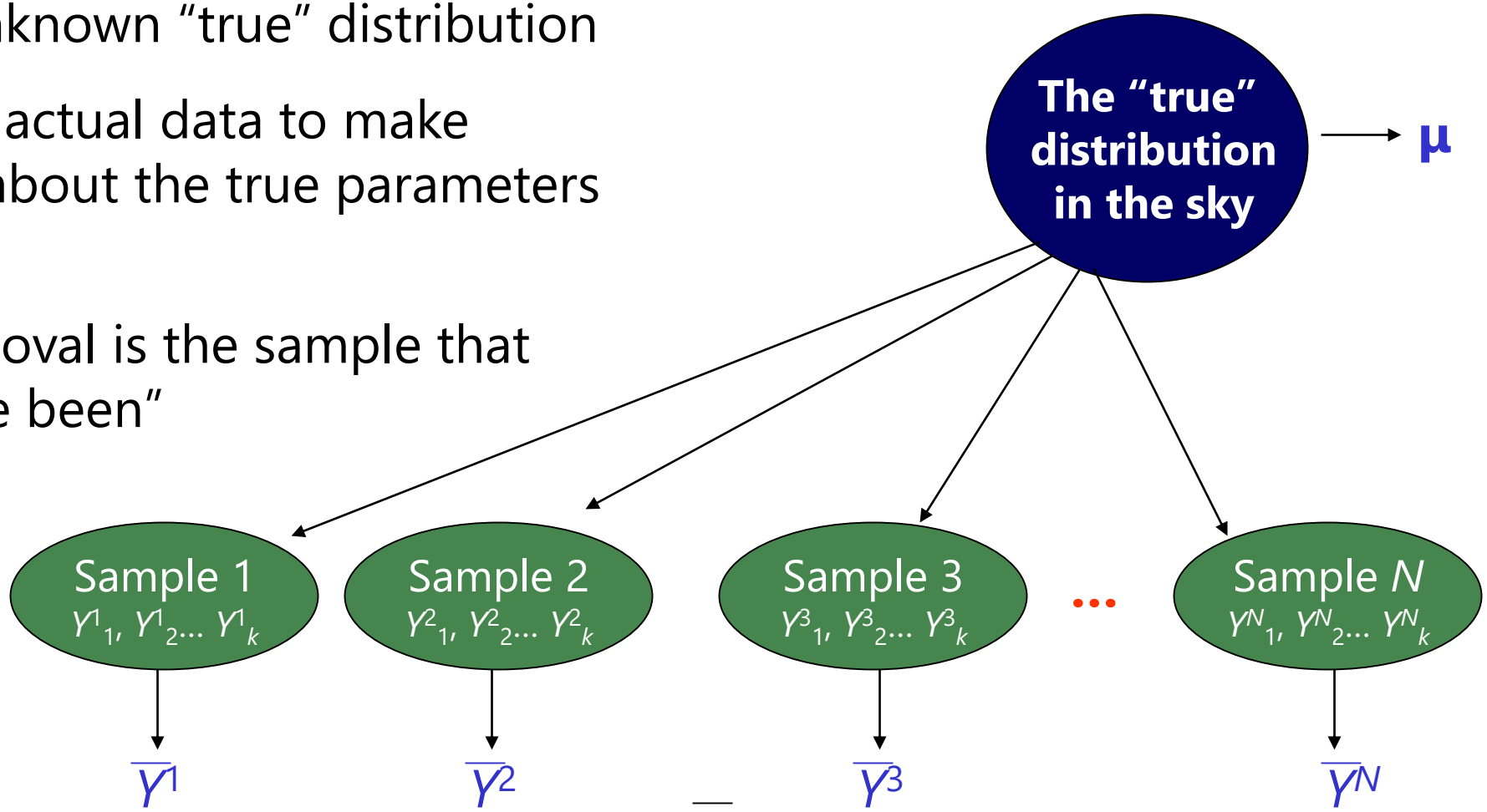
- "An important theme is the substitution of raw computing power for theoretical analysis"
- --Efron and Gong 1983

- Bootstrapping fits very nicely into the "data mining" paradigm.

The Basic Idea

- Any actual sample of data was drawn from the unknown "true" distribution
- We use the actual data to make inferences about the true parameters (μ)
- Each green oval is the sample that "might have been"

Theoretical Picture

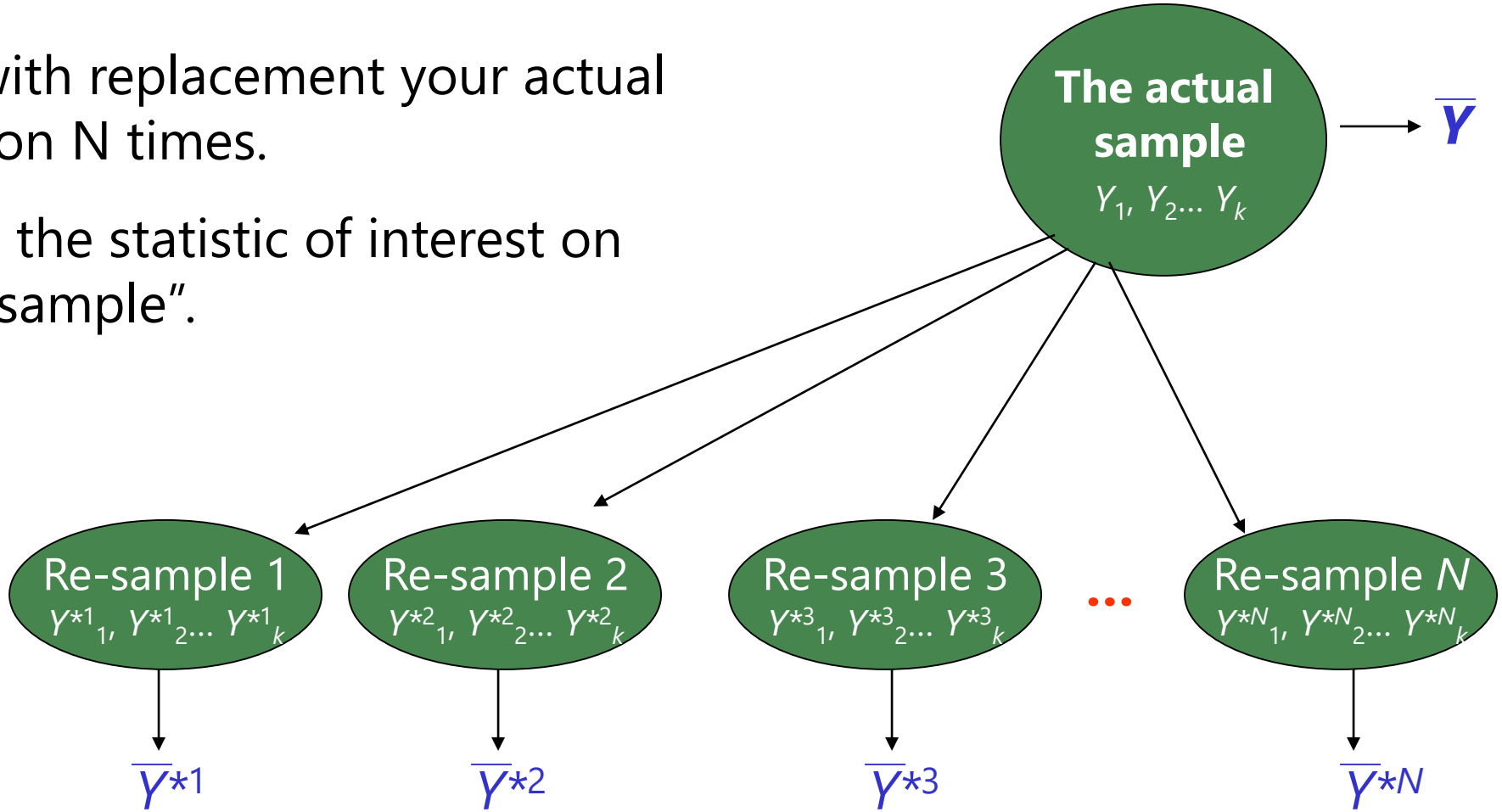


- The distribution of our estimator (Y) depends on both the true distribution *and* the size (k) of our sample

The Basic Idea

- Treat the actual distribution as a proxy for the true distribution.
- Sample with replacement your actual distribution N times.
- Compute the statistic of interest on each "re-sample".

The Bootstrapping Process



→ $\{Y^*\}$ constitutes an estimate of the *distribution* of Y.

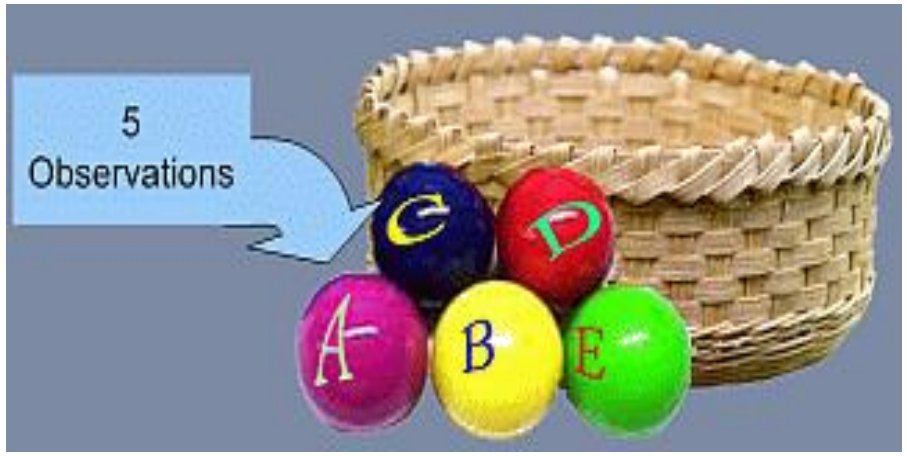
Sampling With Replacement

- In fact, there is a chance of
  - $(1\text{-}1/500)^{500} \approx {}^{1}/_{e} \approx .368$

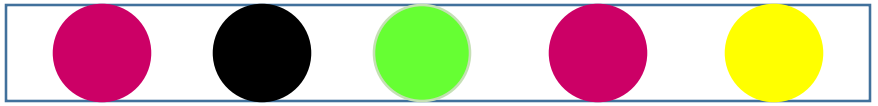
that any one of the original data points won't appear at all if we sample with replacement 500 times.

- ➔ any data point is included with Prob ≈ .632
- Intuitively, we treat the original sample as the "true population in the sky".
- Each *resample* simulates the process of taking a sample from the "true" distribution.

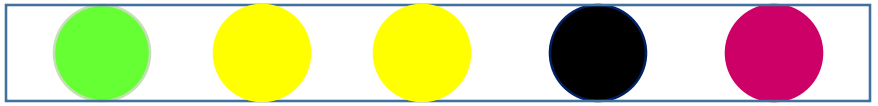
Bootstrap Samples



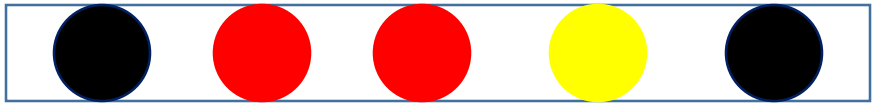
Sample 1



Sample 2

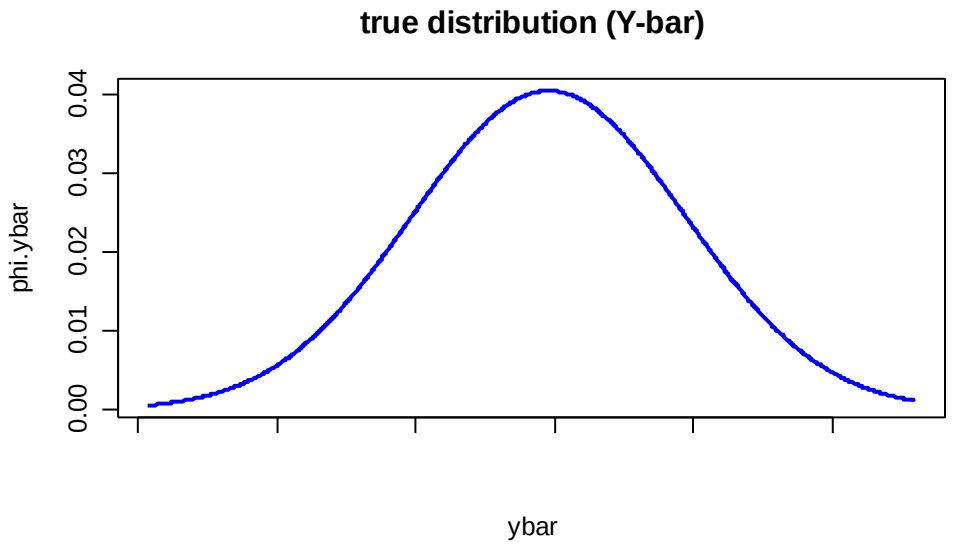


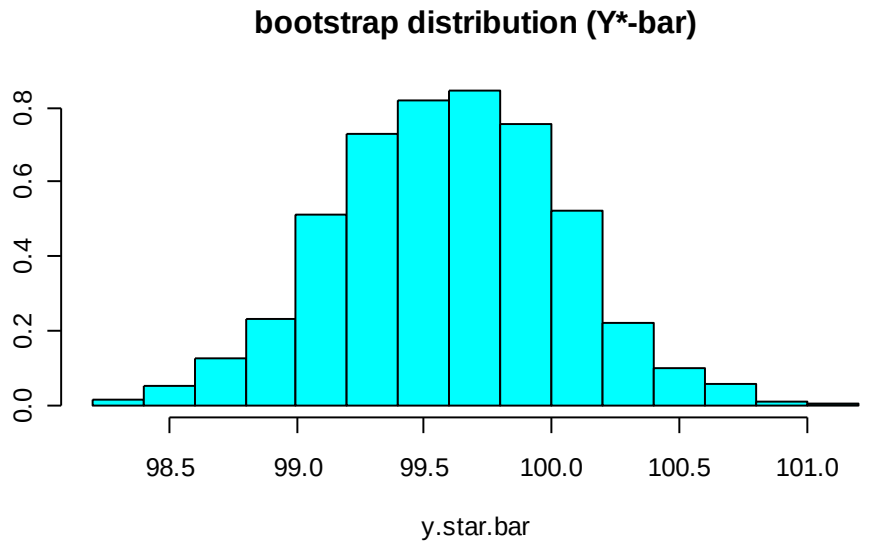
Sample 3



Theoretical vs. Empirical

- Graph on left: Y-bar calculated from an ∞ number of samples from the "true distribution".
- Graph on right: {Y*-bar} calculated in each of 1000 re-samples from the *empirical* distribution.
- Analogy: $\mu : \bar{Y} :: \bar{Y} : \bar{Y}^*$

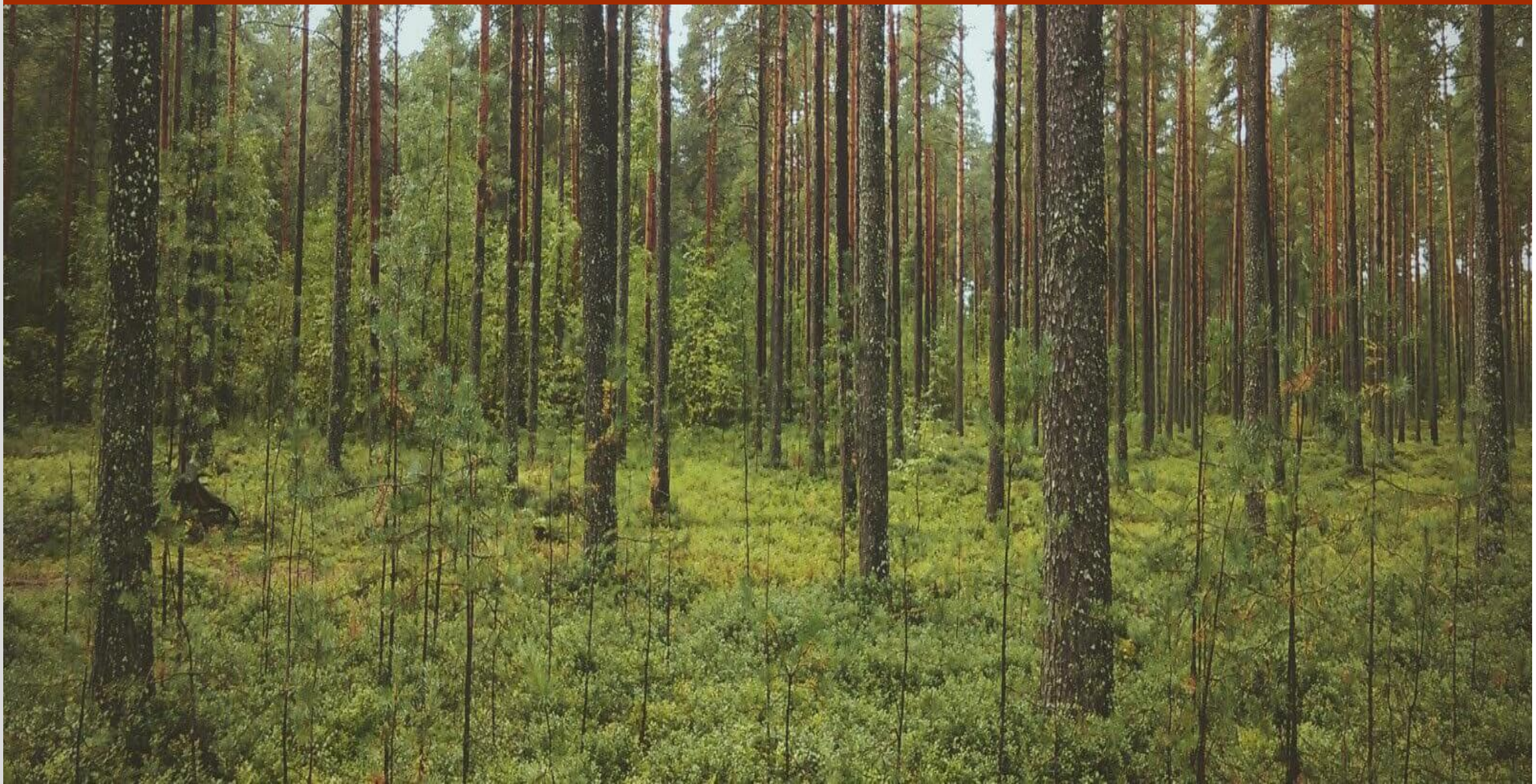




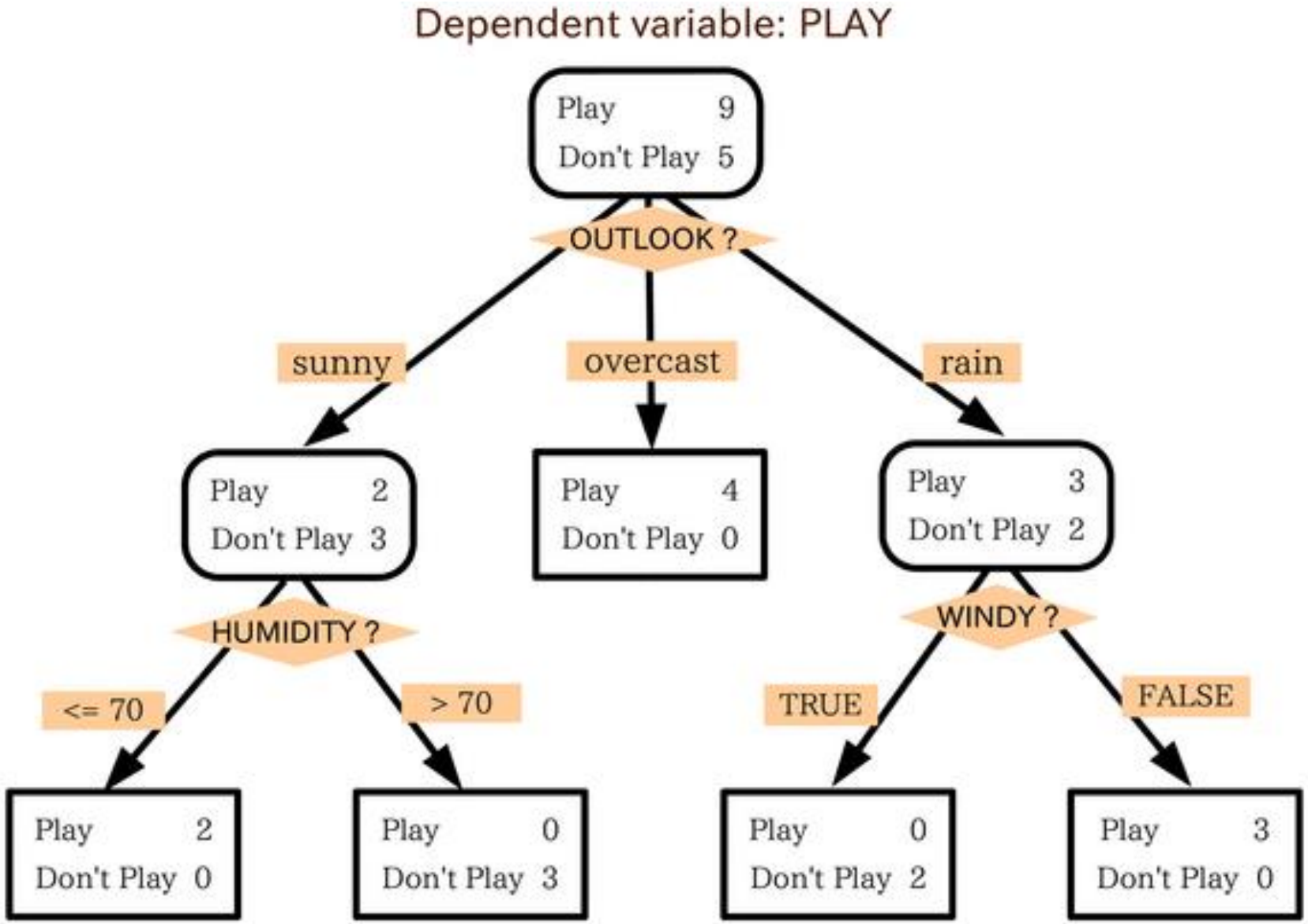
Summary

- The empirical distribution – your data – serves as a proxy to the "true" distribution.
- "Resampling" means (repeatedly) sampling with replacement.
- Resampling the data is analogous to the process of drawing the data from the "true distribution".
- We can resample multiple times
- Compute the statistic of interest *T* on each re-sample
- We get an estimate of the distribution of *T*.

Tree vs. Forest



A Decision Tree



Random Forests

Developed by Prof. Leo Breiman

- Inventor of CART: www.stat.berkeley.edu/users/breiman/ ; http://www.salfordsystems.com/
- Breiman, L.: Random forests. Machine Learning 45(1) (2001) 5–32

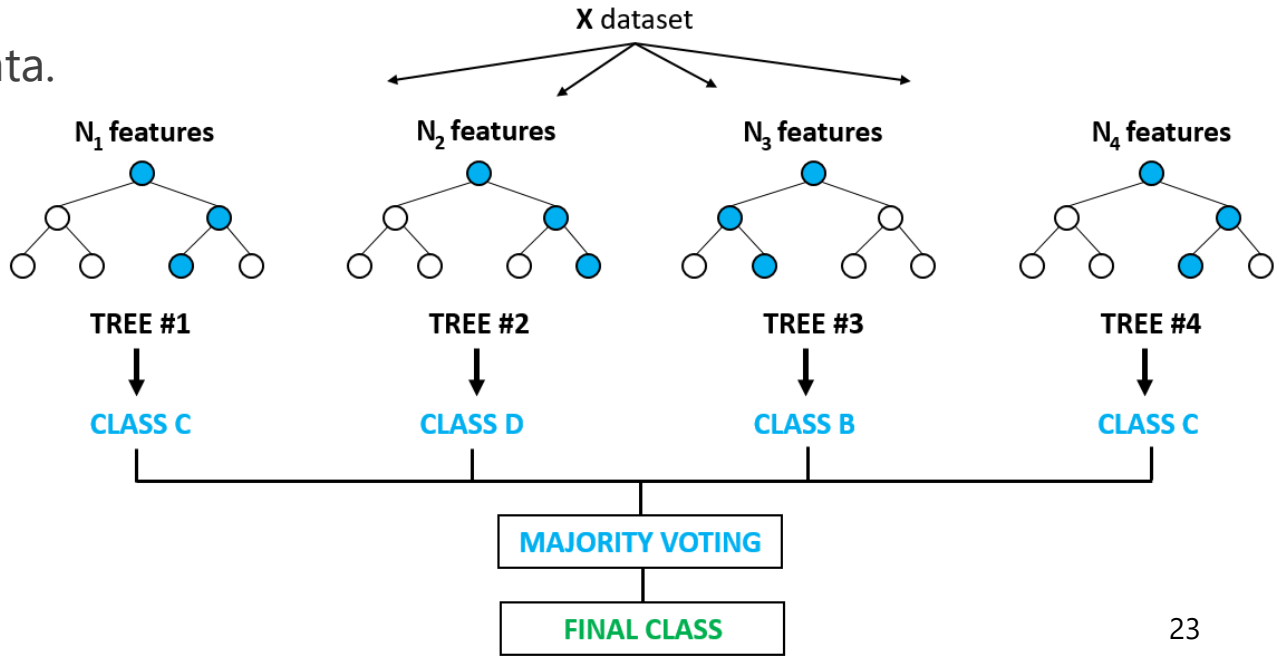
Bootstrap Aggregation (Bagging)

- Resample with Replacement
- Use around two third of the original data.

A Collection of CART-like Trees

- Binary Partition
- No Pruning
- Inherent Randomness

Majority Voting



RF Main Features

Generate substantially different trees:

- Use random bootstrap samples of the training data.
- Use random subsets of variables for each node.

Number of Variables

- Square Root (K)
- K: total number of available variables
- Can dramatically speed up the tree building process.

Number of Trees: 500 or more

Self-Testing

- Around one third of the original data are left out.
- Out of Bag (OOB)
- Similar to Cross-Validation

RF Advantages

All data can be used in the training process.

- No need to leave some data for testing.
- No need to do conventional cross-validation.
- Data in OOB are used to evaluate the current tree.

Performance of the entire RF

- Each data point is tested over a subset of trees.
- Depends on whether it is in the OOB.

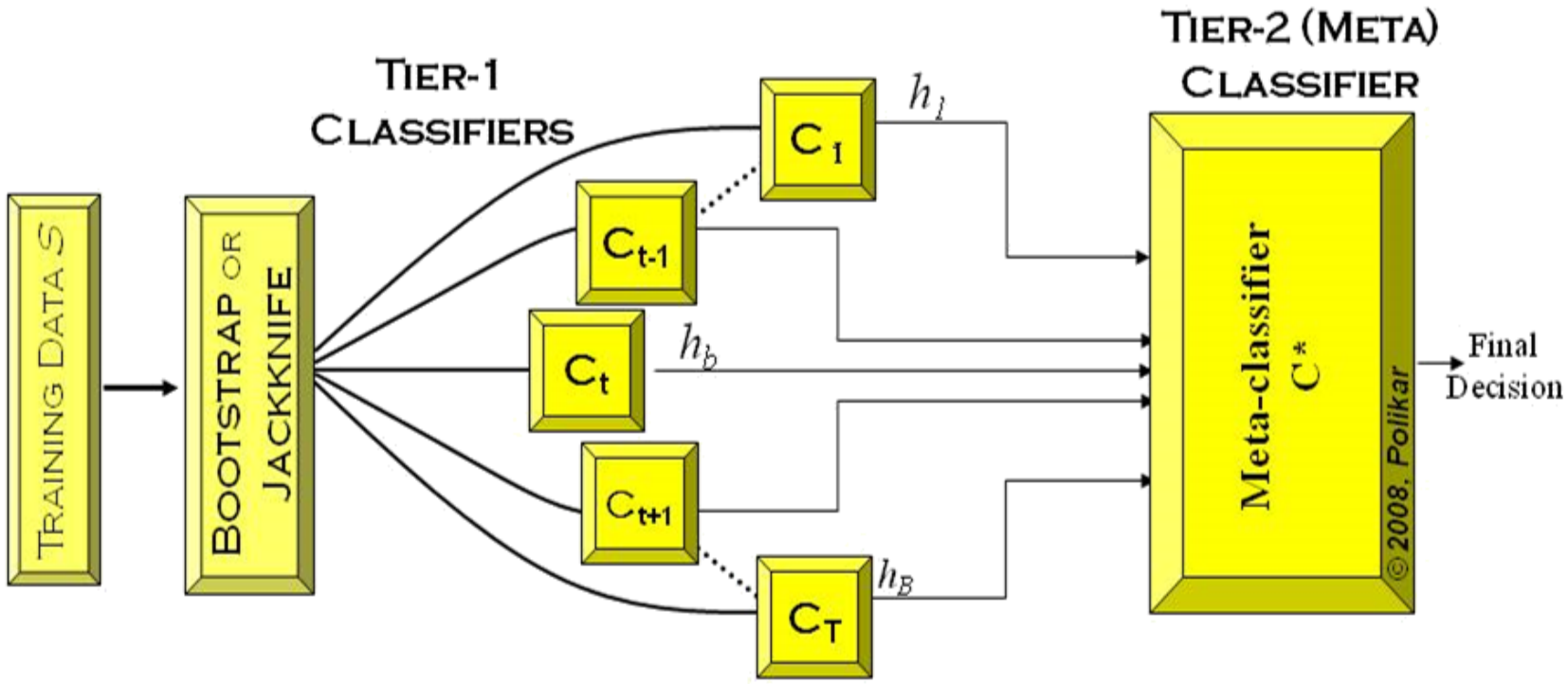
High levels of predictive accuracy

- Only a few parameters to experiment with.
- Suitable for both classification and regression.

Resistant to overtraining (overfitting).

No need for prior feature selection.

Stacking


TRAINING DATA S
BOOTSTRAP OR JACKKNIFE
TIER-1 CLASSIFIERS
C1
Ct-1
Ct
Ct+1
CT
h1
hb
hB
TIER-2 (META) CLASSIFIER
Meta-classifier C*
© 2008, Polikar
Final Decision

Stacking

Input: Data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$;
First-level learning algorithms $\mathcal{L}_1, \cdots, \mathcal{L}_T$;
Second-level learning algorithm $\mathcal{L}$.

Process:

for $t = 1, \cdots, T$:
  $h_t = \mathcal{L}_t(\mathcal{D})$ % Train a first-level individual learner h_t by applying the first-level
end; % learning algorithm $\mathcal{L}_t$ to the original data set $\mathcal{D}$
$\mathcal{D}' = \emptyset$; % Generate a new data set
for $i = 1, \cdots, m$:
  for $t = 1, \cdots, T$:
    $z_{it} = h_t(\boldsymbol{x}_i)$ % Use h_t to classify the training example $\boldsymbol{x}_i$
  end;
  $\mathcal{D}' = \mathcal{D}' \cup \{((z_{i1}, z_{i2}, \cdots, z_{iT}), y_i)\}$
end;
$h' = \mathcal{L}(\mathcal{D}')$. % Train the second-level learner h' by applying the second-level
  % learning algorithm $\mathcal{L}$ to the new data set $\mathcal{D}'$

Output: $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), \cdots, h_T(\boldsymbol{x}))$

Boosting Methods

Summary

1. Overview
2. Boosting – approach, definition, characteristics
3. Early Boosting Algorithms
4. AdaBoost – introduction, definition, main idea, the algorithm
5. AdaBoost – analysis, training error
6. Discrete AdaBoost
7. AdaBoost – pros and contras
8. Boosting Example

Overview

- Introduced in 1990s
- originally designed for classification problems
- extended to regression
- motivation - a procedure that combines the outputs of many “weak” classifiers to produce a powerful “committee”

Boosting Approach

- select small subset of examples
- derive rough rule of thumb
- examine 2nd set of examples
- derive 2nd rule of thumb
- repeat T times

questions:

how to choose subsets of examples to examine on each round?

how to combine all the rules of thumb into single prediction rule?

boosting = general method of converting rough rules of thumb into highly accurate prediction rule

Boosting

Definition

- A machine learning algorithm
- Perform supervised learning
- Increments improvement of learned function
- Forces the weak learner to generate new hypotheses that make less mistakes on “harder” parts.

Characteristics

- iterative
- successive classifiers depends upon its predecessors
- look at errors from previous classifier step to decide how to focus on next iteration over data

History of boosting: Early Boosting Algorithms

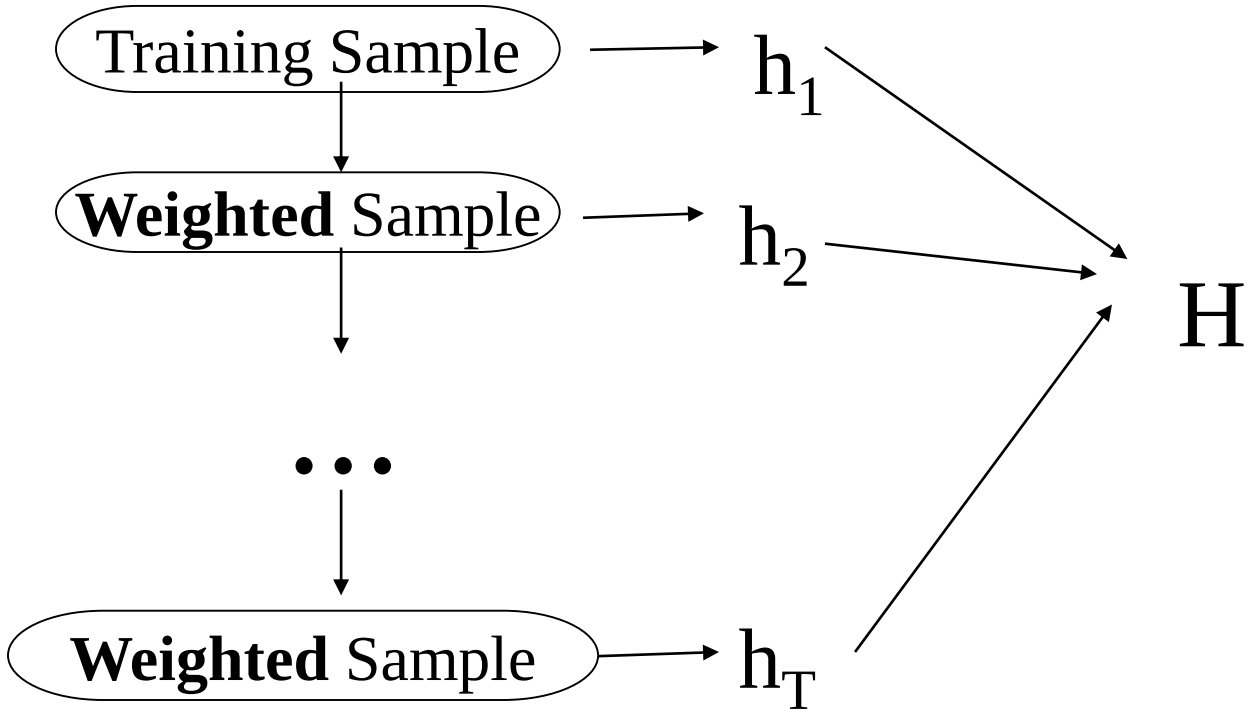
Schapire (1989):

- first provable boosting algorithm
- call weak learner three times on three modified distributions
- get slight boost in accuracy
- apply recursively

Freund (1990)

- "optimal" algorithm that "boosts by majority"
- Drucker, Schapire & Simard (1992):
- first experiments using boosting
- limited by practical drawbacks
- Freund & Schapire (1995) – **AdaBoost**
- strong practical advantages over previous boosting algorithms

Boosting



Boosting

- Train a set of weak hypotheses: h_1, …., h_T.
- The combined hypothesis H is a **weighted** majority vote of the T weak hypotheses.
  - ➔ Each hypothesis h_t has a weight α_t.

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

- During the training, focus on the examples that are misclassified.
  - ➔ At round t, example x_i has the weight $D_t(i)$.

Boosting

- Binary classification problem
- Training data:

$$(x_1, y_1),\ldots,(x_m, y_m), where \;\; x_i \in X, y_i \in Y = \{-1,1\}$$

- $D_t(i)$: the weight of x_i at round t. $D_1(i)=1/m$.
- A learner L that finds a weak hypothesis h_t: X ➔ Y given the training set and D_t
- The error of a weak hypothesis h_t:

$$\varepsilon_t = \Pr_{i\sim D_t}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$$

AdaBoost - Introduction

- Linear classifier with all its desirable properties
- Has good generalization properties
- Is a feature selector with a principled strategy (minimisation of upper bound on empirical error)
- Close to sequential decision making

AdaBoost - Definition

- Is an algorithm for constructing a “strong” classifier as linear combination

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

  of simple “weak” classifiers $h_t(x)$.
- $h_t(x)$ - “weak” or basis classifier, hypothesis, ”feature”
- $H(x) = sign(f(x))$ – “strong” or final classifier/hypothesis

The AdaBoost Algorithm

- Input – a training set: $S = \{(x_1, y_1); \ldots ;(x_m, y_m)\}$

  $x_i \in X$, X instance space

  $y_i \in Y$, Y finite label space

  in binary case Y = {-1,+1}
- Each round, t=1,…,T, AdaBoost calls a given *weak or base learning algorithm* – accepts as input a sequence of training examples (S) and a set of weights over the training example ($Dt(i)$)
- The weak learner computes a weak classifier (h_t), : $h_t : X \rightarrow R$
- Once the weak classifier has been received, AdaBoost chooses a parameter

  ($\alpha_t \in R$) – intuitively measures the importance that it assigns to h_t.

The main idea of AdaBoost

- to use the weak learner to form a highly accurate prediction rule by calling the weak learner repeatedly on different distributions over the training examples.
- initially, all weights are set equally, but each round the weights of incorrectly classified examples are increased so that those observations that the previously classifier poorly predicts receive greater weight on the next iteration.

AdaBoost Algorithm

Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm $\mathcal{L}$;
Number of learning rounds T.

Process:

1. $\mathcal{D}_1(i) = 1/m.$ % Initialize the weight distribution
2. **for** $t = 1, \cdots, T$:
3. $h_t = \mathcal{L}(D, \mathcal{D}_t)$; % Train a learner h_t from D using distribution $\mathcal{D}_t$
4. $\epsilon_t = \Pr_{\boldsymbol{x} \sim \mathcal{D}_t, y} \boldsymbol{I}[h_t(\boldsymbol{x}) \neq y]$; % Measure the error of h_t
5. **if** $\epsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$; % Determine the weight of h_t
7. $$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$$
$$= \frac{\mathcal{D}_t(i) \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$$
% Update the distribution, where
% Z_t is a normalization factor which
% enables $\mathcal{D}_{t+1}$ to be a distribution
8. **end**

Output: $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $D_1(i) = 1/m$ for $i = 1, \ldots, m$. ← Initial Distribution of Data

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$. ← Train model
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right]. \quad \leftarrow \text{Error of model}$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$. ← Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad \leftarrow \text{Update Distribution}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right). \quad \leftarrow \text{Final average}$$

Theorem: training error drops exponentially fast

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

AdaBoost - Analysis

- the weights $D_t(i)$ are updated and normalised on each round. The normalisation factor takes the form

$$Z_t = \sum_{i=1}^{m} D_t(i) \mathrm{e}^{-\alpha_t y_i h_t(x_i)}$$

and it can be verified that Z_t measures exactly the ratio of the new to the old value of the exponential sum

$$\sum_{i=1}^{m} \exp\left(-y_i \sum_{j=1}^{t} \alpha_j h_j(x_i)\right)$$

on each round, so that $\Pi_t Z_t$ is the final value of this sum. We will see below that this product plays a fundamental role in the analysis of AdaBoost.

AdaBoost – Training Error

Theorem:

- run Adaboost
- let $\varepsilon_t = 1/2 - \gamma_t$
- then the training error:

$$H_{final} \leq \prod_t 2\sqrt{\varepsilon_t(1-\varepsilon_t)} = \prod_t \sqrt{1-4\gamma_t^2} \leq \exp(-2\sum_t \gamma_t^2)$$

$$\forall t : \gamma_t \geq \gamma > 0 \Rightarrow H_{final} \leq e^{-2\gamma^2 T}$$

Choosing parameters for Discrete AdaBoost

In Freund and Schapire's original Discrete AdaBoost the algorithm each round selects the weak classifier, h_t, that minimizes the weighted error on the training set

$$\epsilon_t = \sum_i D_t(i)[\![h_t(x_i) \neq y_i]\!] = \sum_i D_t(i)\left(\frac{1 - y_i h_t(x_i)}{2}\right)$$

Minimizing Z_t, we can rewrite:

$$\begin{aligned} Z_t &= \sum_i D_t(i) \mathrm{e}^{-\alpha_t y_i h_t(x_i)} \\ &= \sum_i D_t(i)\left(\frac{1 + y_i h_t(x_i)}{2}\mathrm{e}^{-\alpha_t} + \frac{1 - y_i h_t(x_i)}{2}\mathrm{e}^{\alpha_t}\right) \end{aligned}$$

Choosing parameters for Discrete AdaBoost

- analytically we can choose α_t by minimizing the first (ε_t=…) expression:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

- Plugging this into the second equation (Z_t), we can obtain:

$$Z_t = 2\sqrt{\epsilon_t (1 - \epsilon_t)}.$$

Discrete AdaBoost - Algorithm

Given $(x_1, y_1),\ldots, (x_m, y_m)$ where $x_i \epsilon X$, $y_i \epsilon \{-1, +1\}$

Initialise weights $D_1(i) = 1/m$

Iterate $t = 1,\ldots,T$:

Find $h_t = \arg\min_{h_j} \epsilon_j$ where $\epsilon_j = \sum_{i=1}^{m} D_t(i)[\![h_t(x_i) \neq y_i]\!]$

Set

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Update: $$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Output – the final classifier $$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

AdaBoost – Pros and Contras

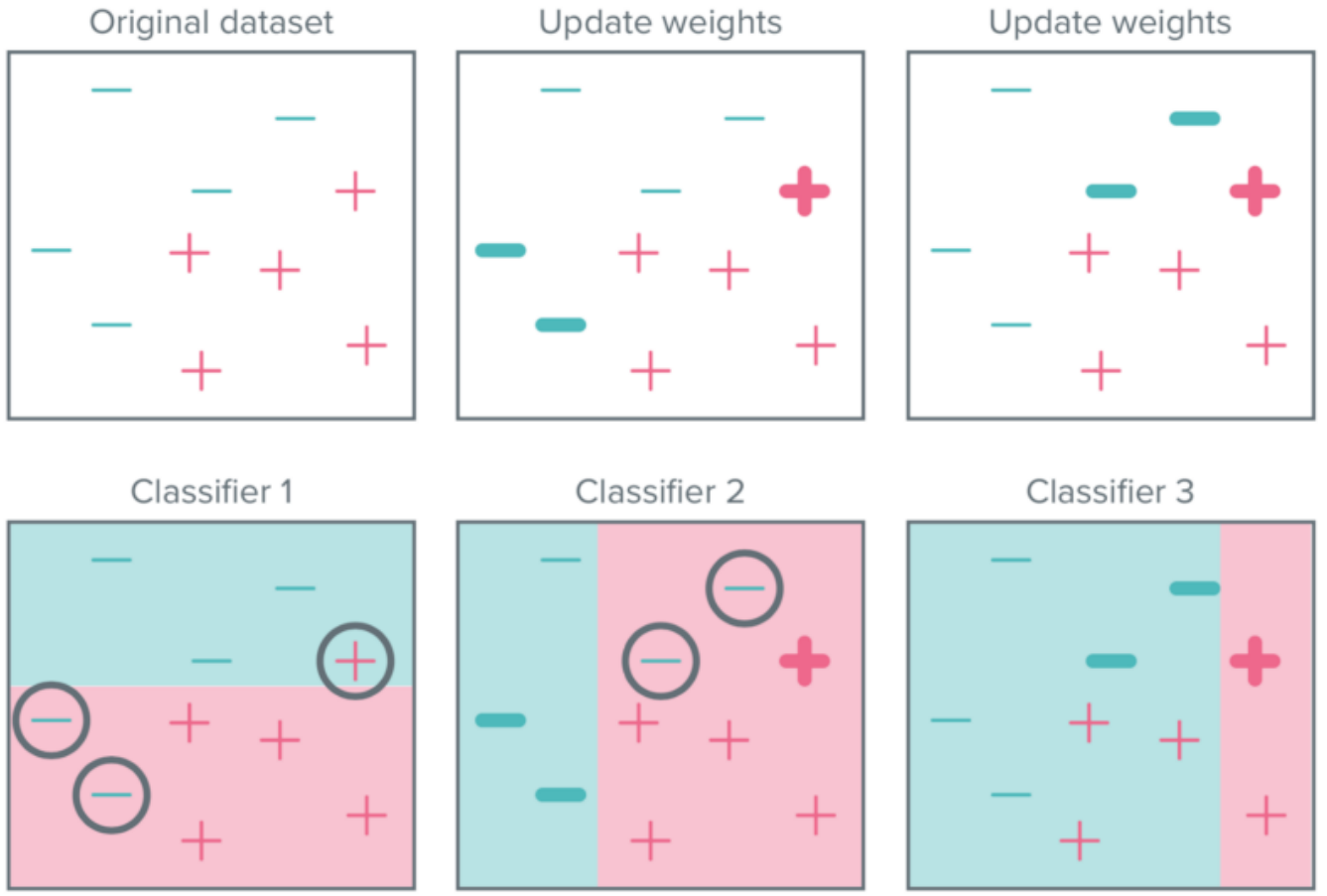
Pros:

- Very simple to implement
- Fairly good generalization
- The prior error need not be known ahead of time

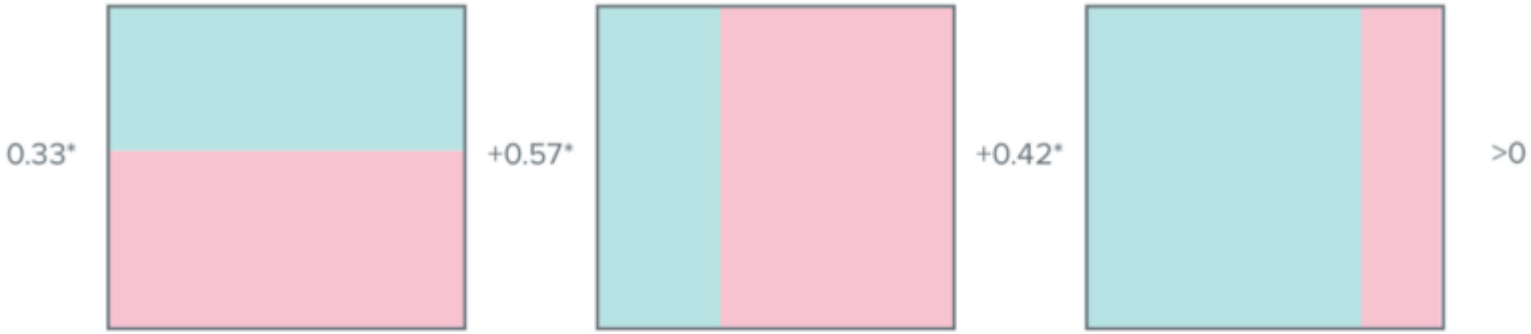
Contras:

- Suboptimal solution
- Can over fit in presence of noise

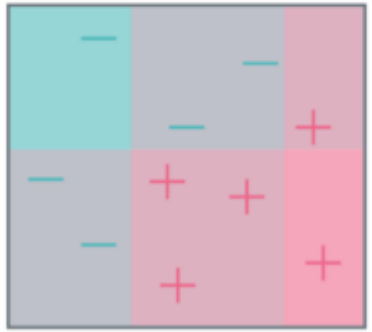
Example


Original dataset
Update weights
Update weights
Classifier 1
Classifier 2
Classifier 3

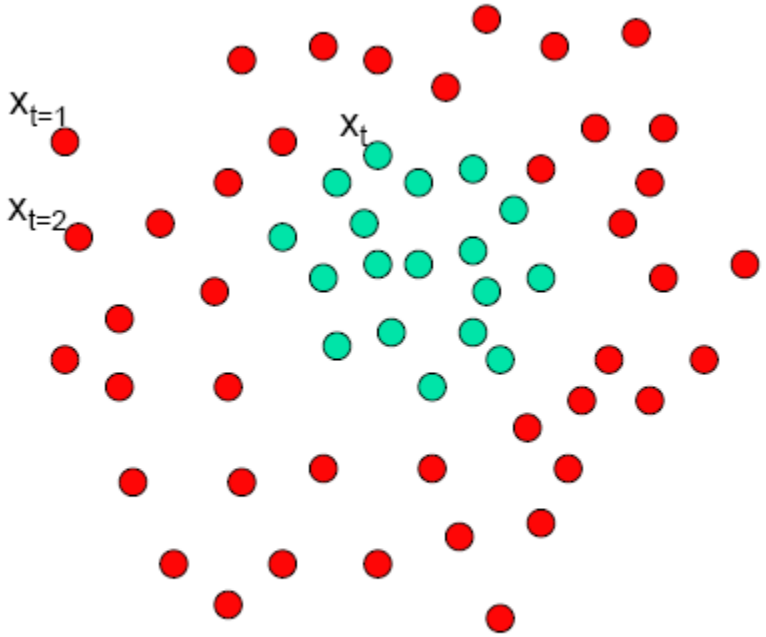
Example



Combined classifier



Boosting - Example



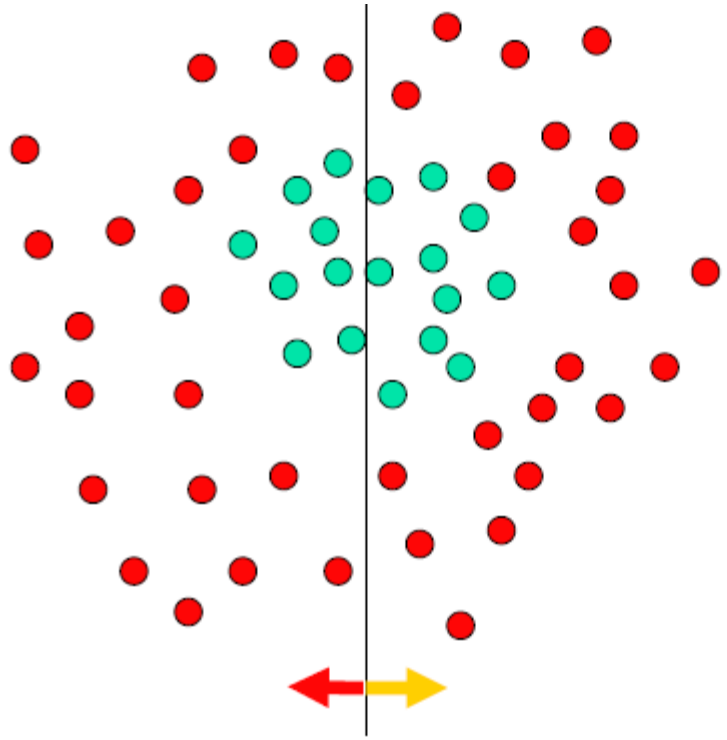
Each data point has a class label:

$$y_t = \begin{cases} +1 \ (\bullet) \\ -1 \ (\bullet) \end{cases}$$

and a weight:

$$w_t = 1$$

Boosting - Example



Each data point has

a class label:

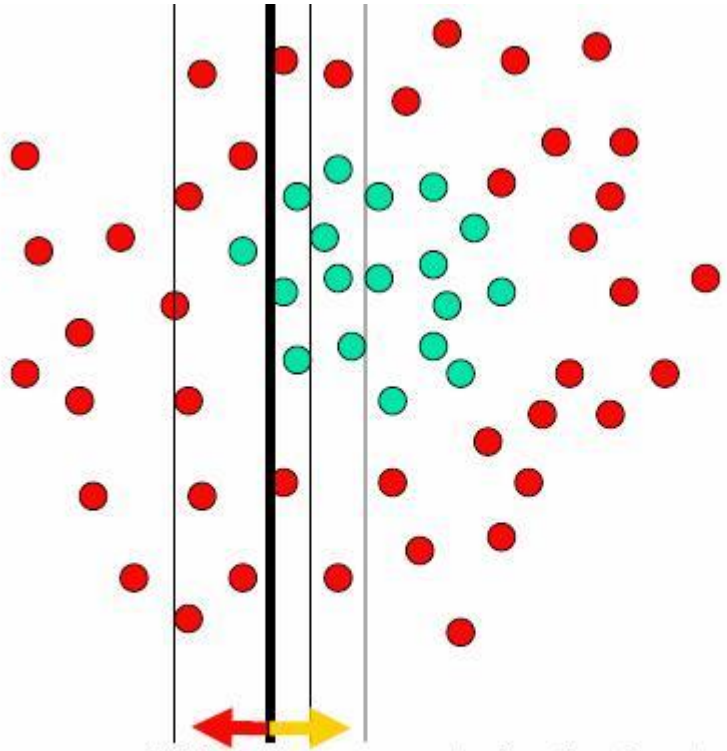
$$y_t = \begin{cases} +1 \ (\bullet) \\ -1 \ (\bullet) \end{cases}$$

and a weight:

$$w_t = 1$$

h => p(error) = 0.5 it is at chance

Boosting - Example



This one seems to be the best

Each data point has

a class label:

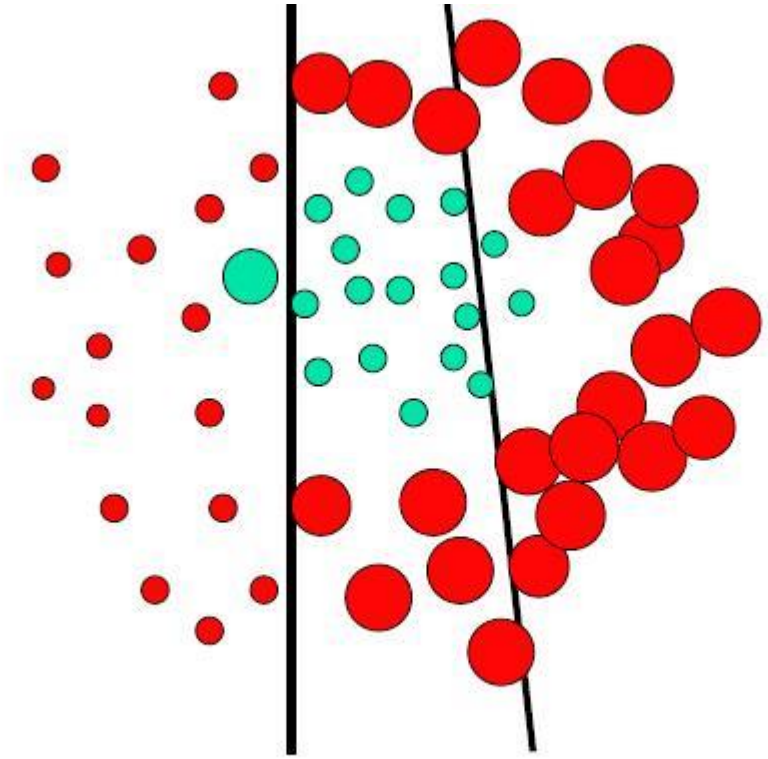
$$y_t = \begin{cases} +1 \text{ (●)} \\ -1 \text{ (○)} \end{cases}$$

and a weight:

$$w_t = 1$$

This is a '**weak classifier**': It performs slightly better than chance.

Boosting - Example



Each data point has

a class label:

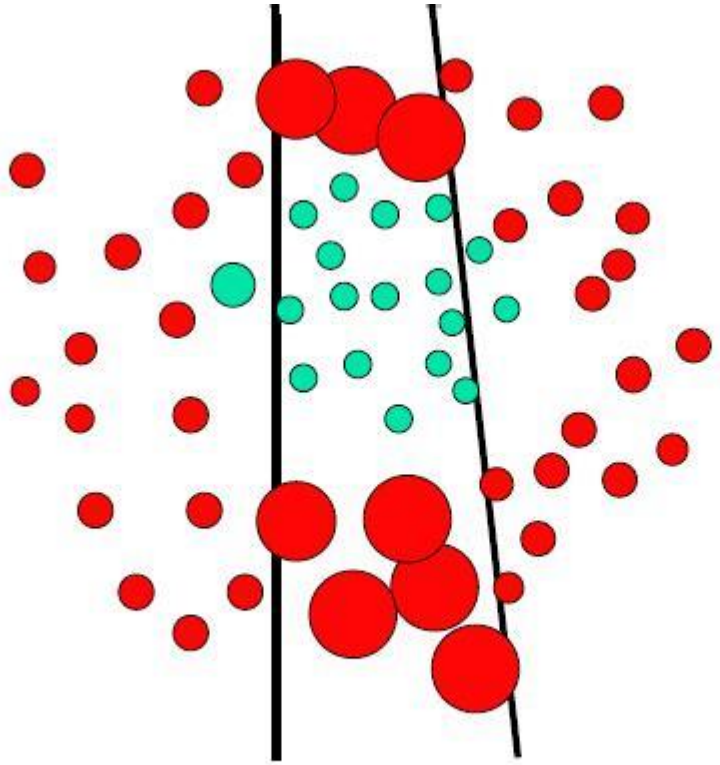
$$y_t = \begin{cases} +1 \ (\bullet) \\ -1 \ (\bullet) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Boosting - Example



Each data point has

a class label:

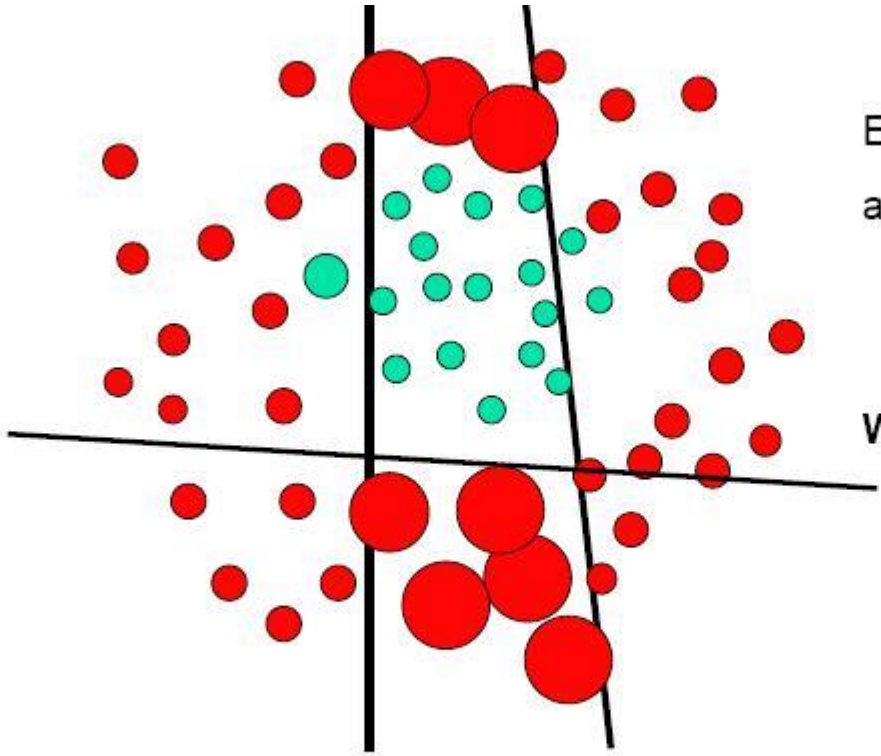
$$y_t = \begin{cases} +1 \ (\bullet) \\ -1 \ (\bullet) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Boosting - Example



Each data point has

a class label:

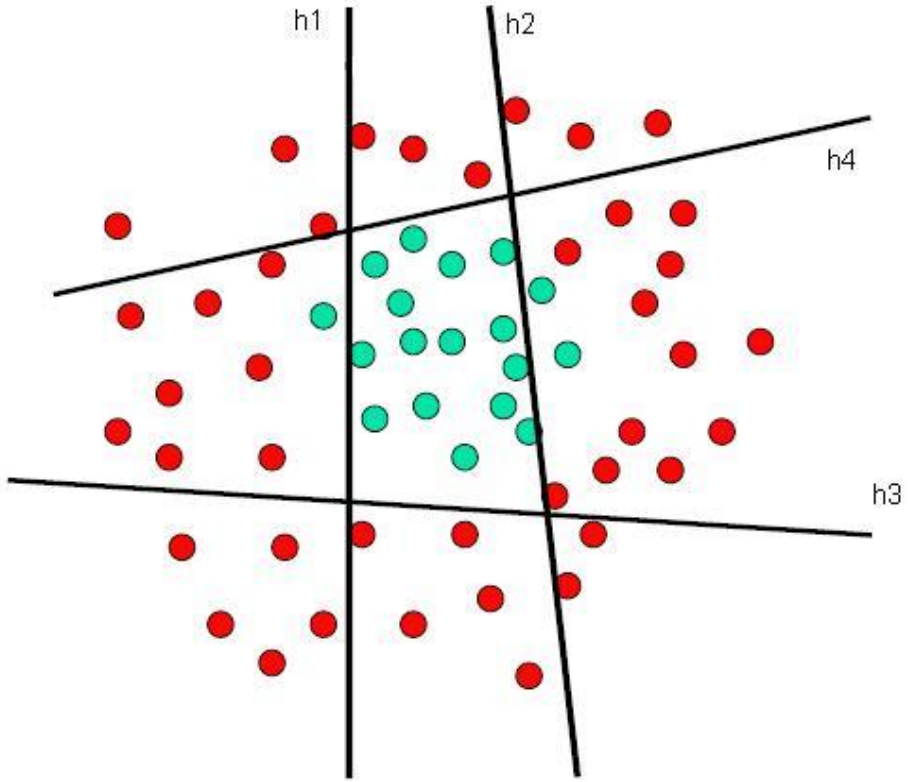
$$y_t = \begin{cases} +1 \text{ (●)} \\ -1 \text{ (●)} \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Boosting - Example



The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

Bibliography

- Friedman, Hastie & Tibshirani: The Elements of Statistical Learning (Ch. 10), 2001
- Y. Freund: Boosting a weak learning algorithm by majority. In *Proceedings of the Workshop on Computational Learning Theory*, 1990.
- Y. Freund and R.E. Schapire: A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, 1995.

Bibliography

- J. Friedman, T. Hastie, and R. Tibshirani: Additive logistic regression: a statistical view of boosting. *Technical Report, Dept. of Statistics, Stanford University*, 1998.
- Thomas G. Dietterich: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 139–158, 2000.