Ensemble Methods

Bias Variance and Noise

- **Error = Variance + Bias^2 + Noise^2**
- Variance: $E[\,(h(x^*) - \underline{h(x^*)})^2\,]$

Describes how much $h(x^*)$ varies from one training set S to another

- Bias: $[\underline{h(x^*)} - f(x^*)]$

Describes the average error of $h(x^*)$.

- Noise: $E[\,(y^* - f(x^*))^2\,] = E[\varepsilon^2] = \sigma^2$

Describes how much y^* varies from $f(x^*)$

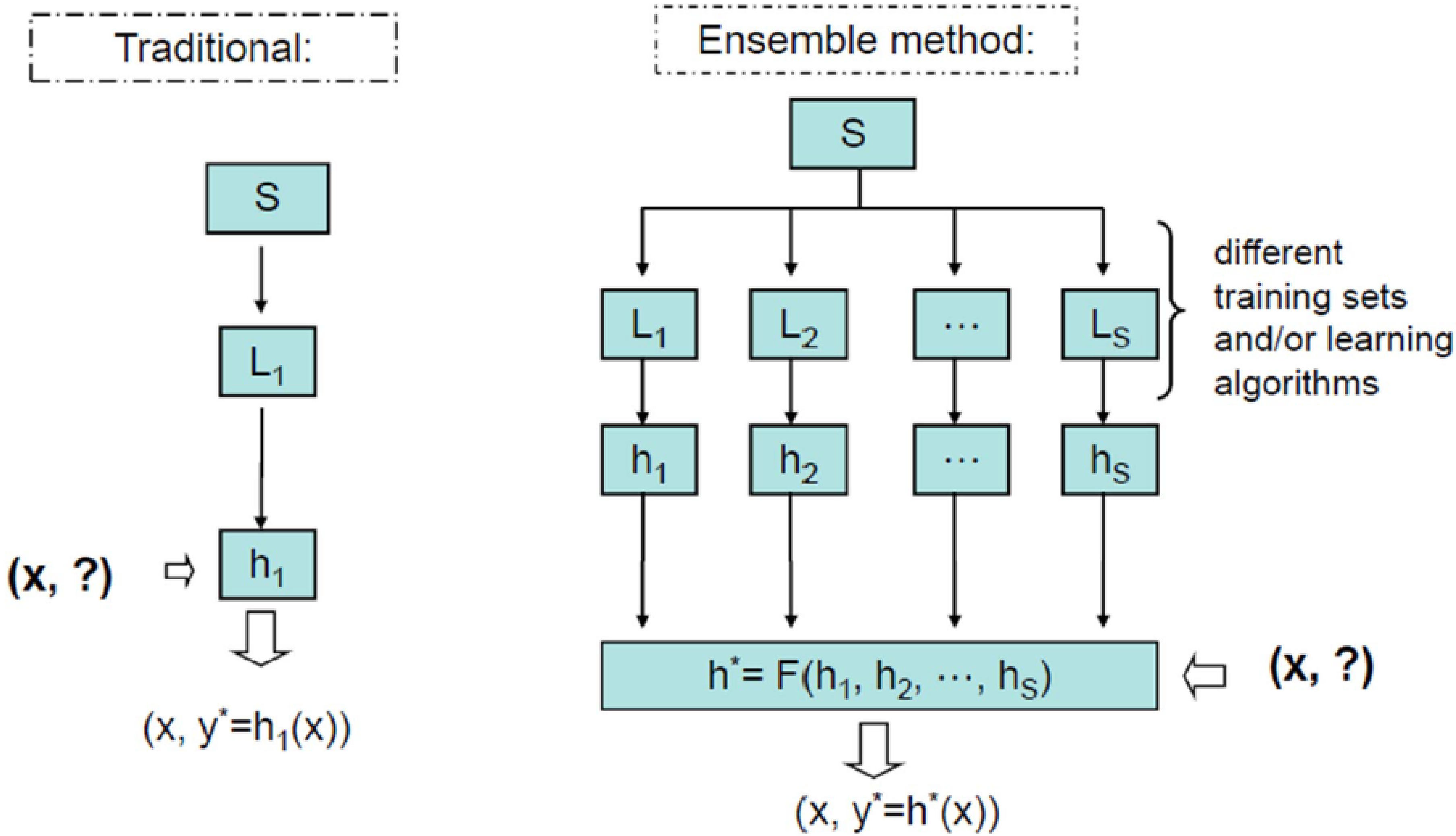
Bias and Variance Measurement Procedure

- Construct B bootstrap replicates of S (e.g., B = 200): S1, …, S_B
- Apply learning algorithm to each replicate S_b to obtain hypothesis h_b
- Let $T_b = S \setminus S_b$ be the data points that do not appear in S_b (out of bag points)
- Compute predicted value $h_b(x)$ for each x in T_b

Estimating B/V/N

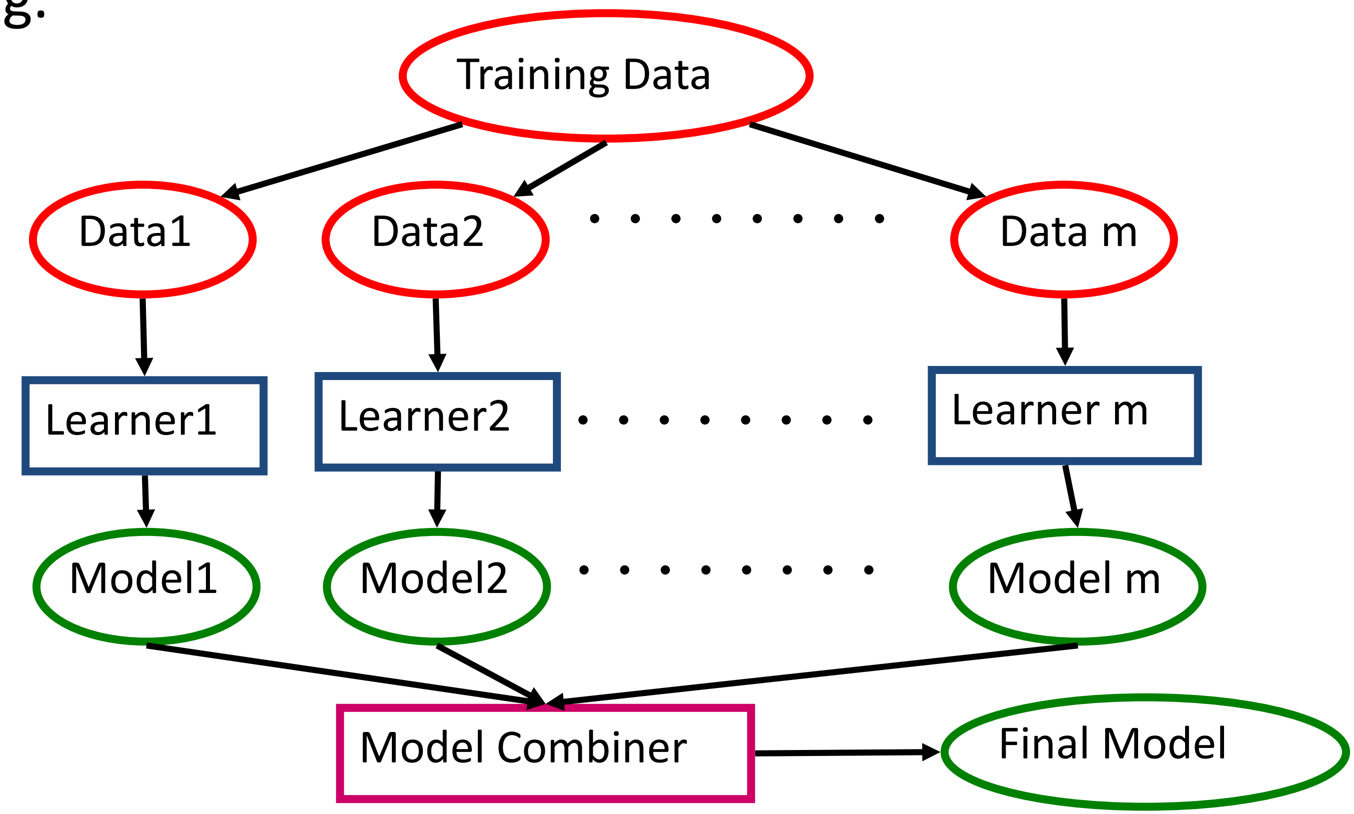
- For each data point x, we will now have the observed corresponding value y and several predictions $y_1, \ldots, y_K$
- Compute the average prediction $\underline{h}$
- Estimate bias as $(\underline{h} - y)$
- Estimate variance as $\Sigma_k\ (y_k - h)2/(K - 1)$
- Assume noise is 0

Ensemble Learning



Learning Ensembles

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms.
- Combine decisions of multiple definitions, e.g. using weighted voting.



How to generate ensembles?

- This is an active research area in machine learning
- We will study two popular methods
  - Bagging
  - Boosting
- Key Feature: They take a **single** learning algorithm and generate multiple variations (ensembles)

Why Ensembles?

- When combining multiple ***independent*** and ***diverse*** decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.
- Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.
  - Who Wants to be a Millionaire: Expert friend vs. audience vote.
- **Theoretically:** They serve to reduce <u>bias</u> and/or <u>variance</u>

"Base" Learning Algorithm

- We can treat the base learning algorithm as a 'black box'.

Protocol to *Learn*:

Input:

S - set of labeled training instances.

Output:

h - a hypothesis from hypothesis space H.

Bagging Algorithm

Given training set S, bagging works as follows:

1. Create T ***bootstrap samples*** $\{S_1, \ldots, S_T\}$of S as follows:
   - For each S_i: Randomly drawing |S| examples from S with replacement
2. For each $i = 1, \ldots, T, h_i = Learn(S_i)$
3. Output $H =< \{h_1, \ldots, h_T\}, majorityVote >$

With large |S|, each S_i will contain $1 - \frac{1}{e} \approx 63.2\%$ unique examples

Bagging

- Create ensembles by repeatedly randomly resampling the training data (Brieman, 1996).
- Given a training set of size *n*, create *m* samples of size *n* by drawing *n* examples from the original data, ***with replacement***.
  - Each ***bootstrap sample*** will on average contain 63.2% of the unique training examples, the rest are replicates.
- Combine the *m* resulting models using simple majority vote.
- Decreases error by decreasing the variance in the results due to ***unstable learners***, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed.

Bias Variance analysis of Bagging

- If we estimate bias and variance using the same *B* bootstrap samples, we will have:
  - Bias = (h – y) [same as before]
  - Variance = Σ_k (h – h)^2/(K – 1) = 0
- Hence, according to this approximate way of estimating variance, bagging removes the variance while leaving bias unchanged.
- In reality, bagging only reduces variance and tends to slightly increase bias

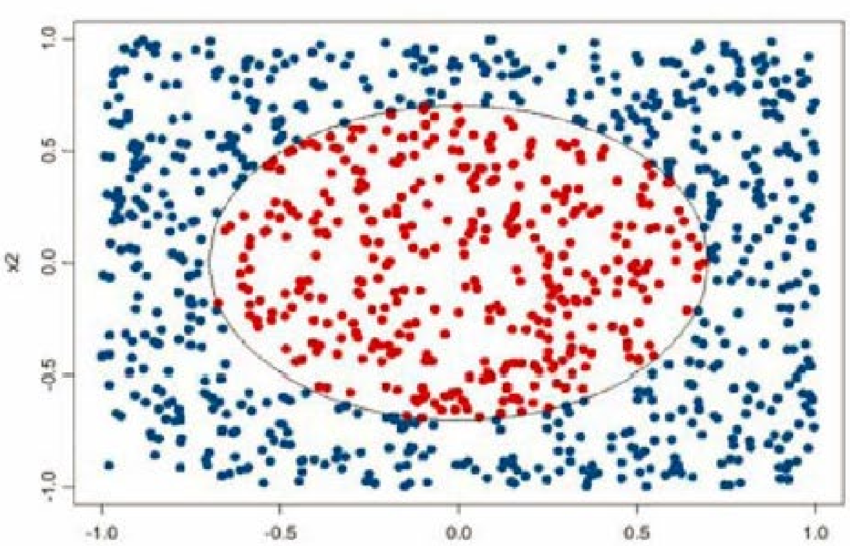
Bias Variance Heuristics

- Models that fit the data poorly have high bias: “inflexible models” such as linear regression, regression stumps
- Models that can fit the data very well have low bias but high variance: “flexible” models such as nearest neighbor regression, regression trees
- This suggests that bagging of a flexible model can reduce the variance while benefiting from the low bias

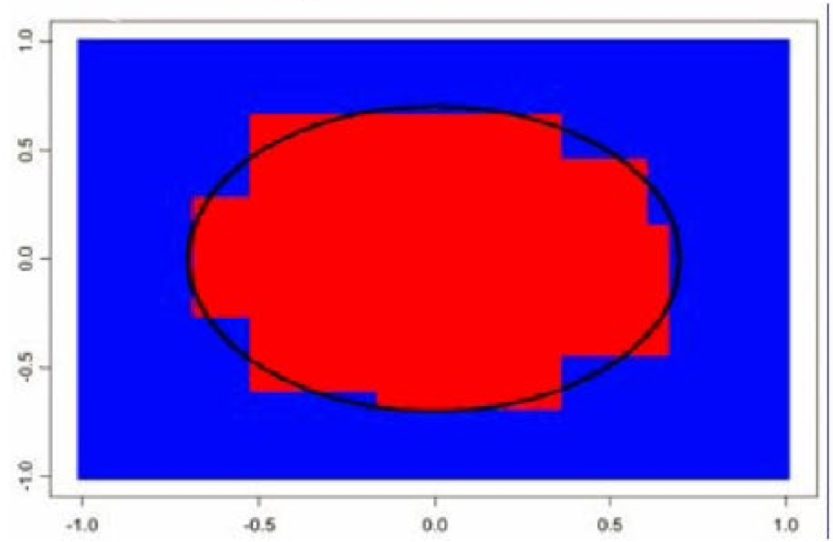
Stability of Learn

- A learning algorithm is **unstable** if small changes in the training data can produce large changes in the output hypothesis (otherwise **stable**).
- Clearly bagging will have little benefit when used with stable base learning algorithms (i.e., most ensemble members will be very similar).
- Bagging generally works best when used with unstable yet relatively accurate base learners

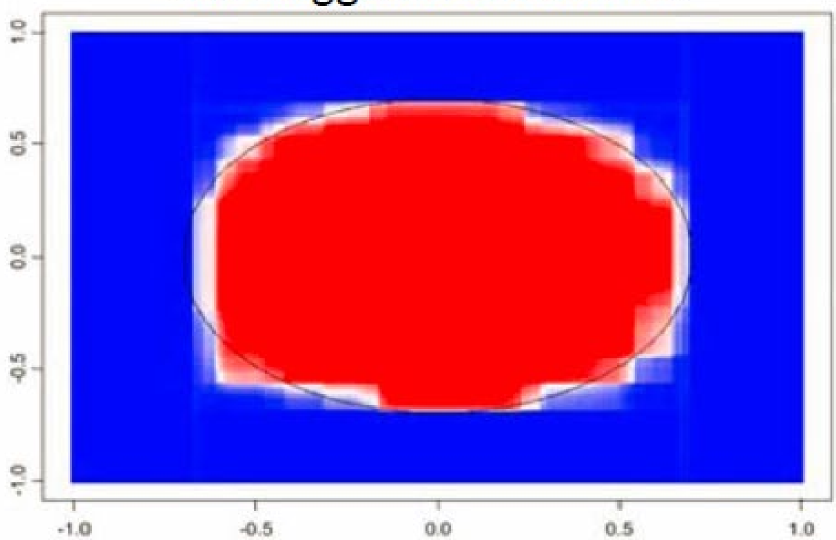
Target concept



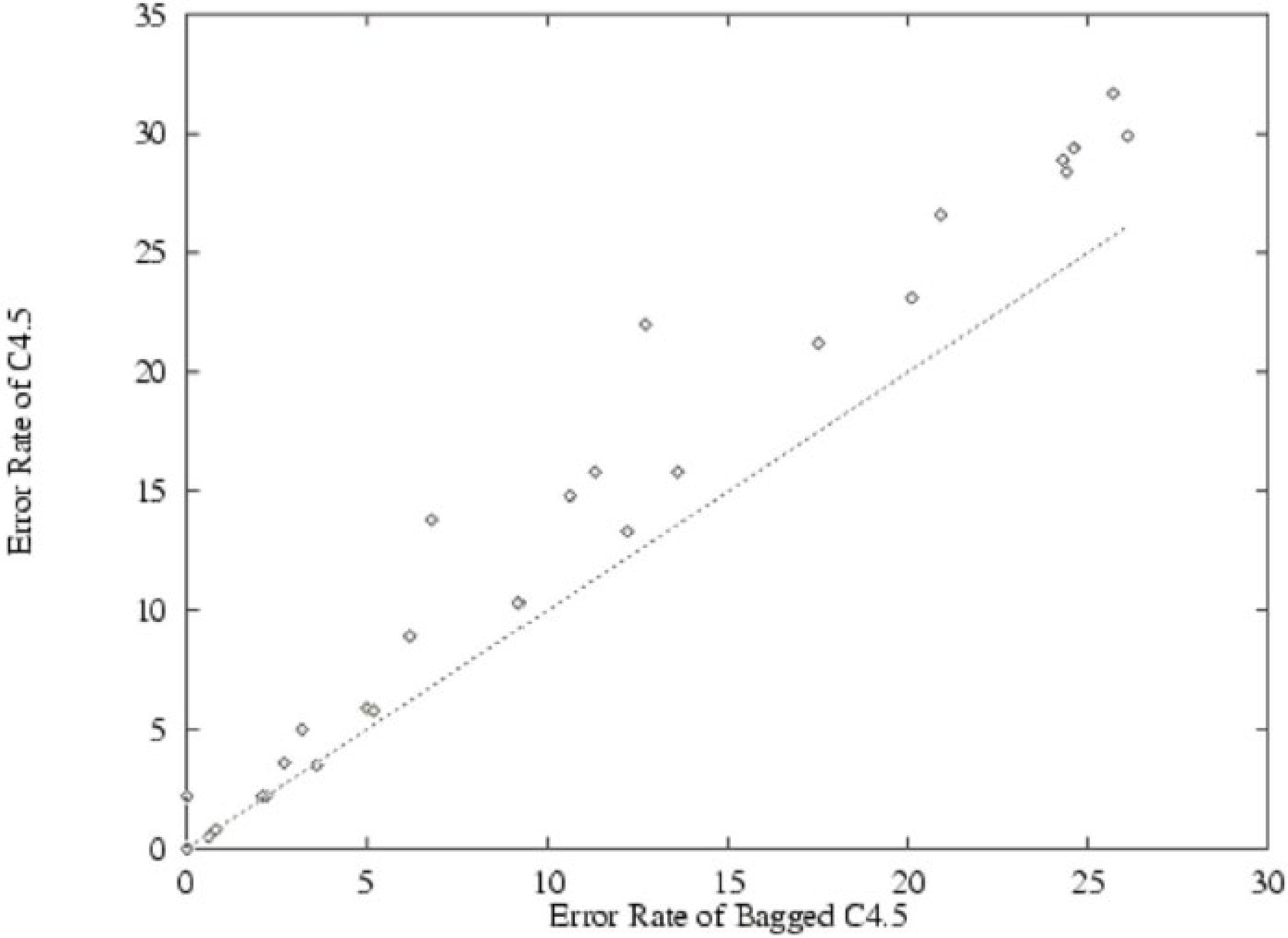
Single decision tree



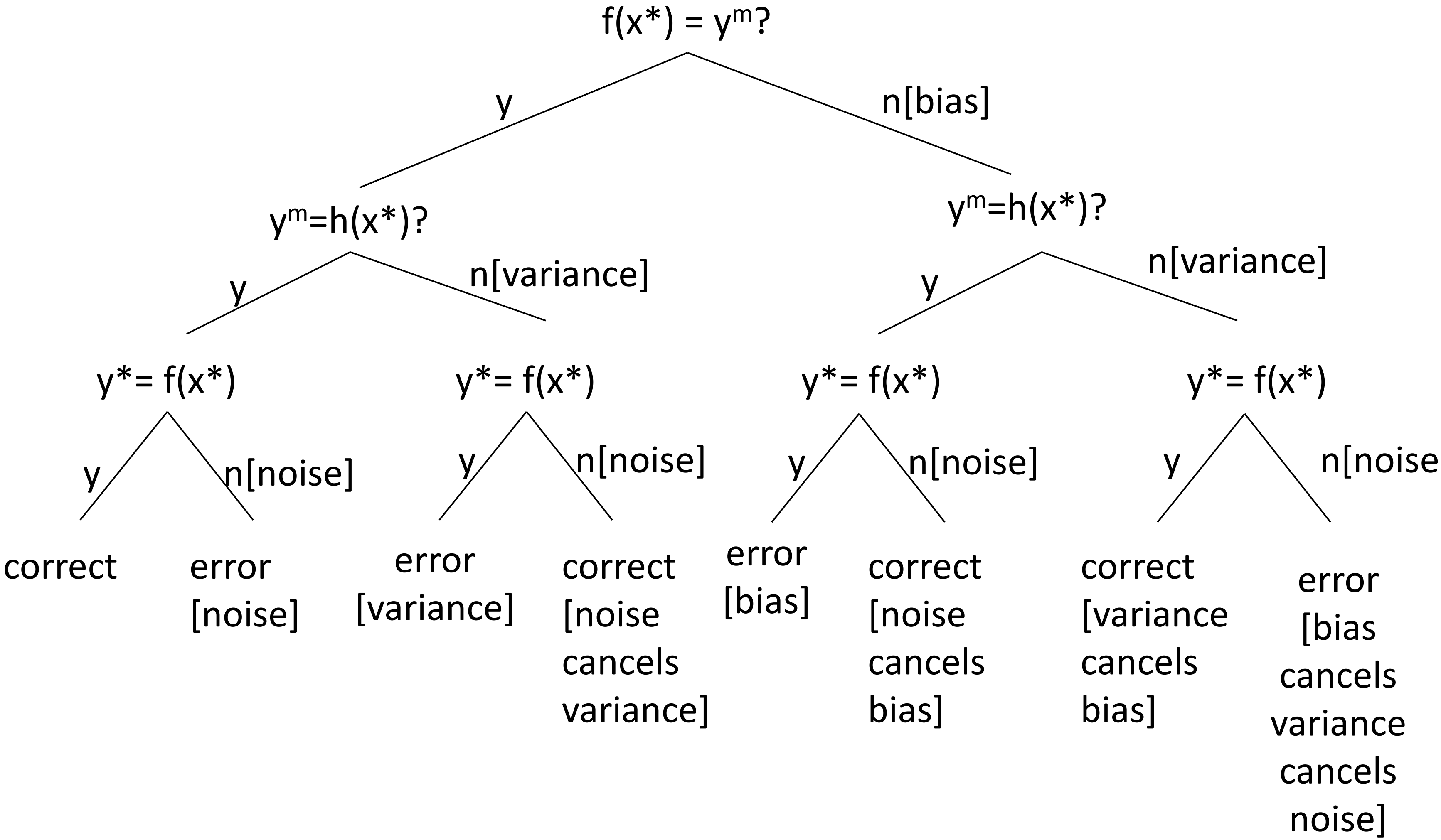
100 bagged decision tree



Bagging Decision Trees
(Freund & Schapire)



Case Analysis of Error



Mean prediction = y^m

Boosting

- Boosting seeks to find a weighted combination of classifiers that fits the data well
- Prediction: Boosting will primarily act to reduce bias

Boosting

Key difference compared to bagging?

- Its iterative.
  - **Bagging :** Individual classifiers were independent.
  - **Boosting:**
    - Look at **errors from previous classifiers** to decide what to **focus** on for the next iteration over data
    - Successive classifiers depends upon its predecessors.
    - Result: more weights on ‘hard’ examples. (the ones on which we committed mistakes in the previous iterations)

Boosting

- Originally developed by computational learning theorists to guarantee performance improvements on fitting training data for a ***weak learner*** that only needs to generate a hypothesis with a training accuracy greater than 0.5 (Schapire, 1990).
- Revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).
- Examples are given weights. At each iteration, a new hypothesis is learned and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.

Boosting – High-level algorithm

- General Loop:

  Set all examples to have equal uniform weights.
  For t from 1 to T do:
  Learn a hypothesis, h_t, from the weighted examples
  Decrease the weights of examples h_t classifies correctly

- Base (weak) learner must focus on correctly classifying the most highly weighted examples while strongly avoiding over-fitting.

- During testing, each of the T hypotheses get a weighted vote proportional to their accuracy on the training data.

AdaBoost

- The boosting algorithm derived from the original proof is impractical
  - requires to many calls to *Learn*, though only polynomially many
- Practically efficient boosting algorithm
  - Adaboost
  - Makes more effective use of each call of *Learn*

Specifying Input Distributions

- AdaBoost works by invoking *Learn* many times on different distributions over the training data set.
- Need to modify base learner protocol to accept a training set distribution as an input.

Protocol to *Learn*:

Input:

S - Set of N labelled training instances.

D - Distribution over S where $D(i)$ is the weight of the *i'th* training instance (interpreted as the probability of observing *i'th* instance). Where $\sum_{i=1}^{N} D(i) = 1$.

Output:

h - a hypothesis from hypothesis space H

D(i) can be viewed as indicating to base learner *Learn* the importance of correctly classifying the *i'th* training instance

AdaBoost (High level steps)

- AdaBoost performs L boosting rounds, the operations in each boosting round l are:

1. Call *Learn* on data set *S* with distribution D_l to produce *l'th* ensemble member h_l, where D_l is the distribution of round l.
2. Compute the $l + 1\ th$ round distribution D_{l+1} by putting more weight on instances that h_l makes mistakes on
3. Compute a voting weight α_l for h_l

The ensemble hypothesis returned is:

H=<$\{h_1, \dots, h_L\}, weightedVote(\alpha_1, \dots, \alpha_L)$>

AdaBoost algorithm:

Input: *Learn* - Base learning algorithm.

S - Set of N labeled training instances.

Output: $H = \langle \{h_1, \ldots, h_L\}, WeightedVote(\alpha_1, \ldots, \alpha_L) \rangle$

Initialize $D_1(i) = 1/N$, for all i from 1 to N. (uniform distribution)

FOR $l = 1, 2, \ldots, L$ **DO**

$h_l = Learn(S, D_l)$

$\varepsilon_l = error(h_l, S, D_l)$

$$\alpha_l = \frac{1}{2}\ln\left(\frac{1-\varepsilon_l}{\varepsilon_l}\right)$$;; if $\varepsilon_l < 0.5$ implies $\alpha_l > 0$

$$D_{l+1}(i) = D_l(i) \times \begin{cases} e^{\alpha_l}, & h_l(x_i) \neq y_i \\ e^{-\alpha_l}, & h_l(x_i) = y_i \end{cases}$$ for i from 1 to N

Normalize D_{l+1} ;; can show that h_l has 0.5 error on D_{l+1}

Note that $\varepsilon_l < 0.5$ implies $\alpha_l > 0$ so weight is decreased for instances h_l predicts correctly and increases for incorrect instances

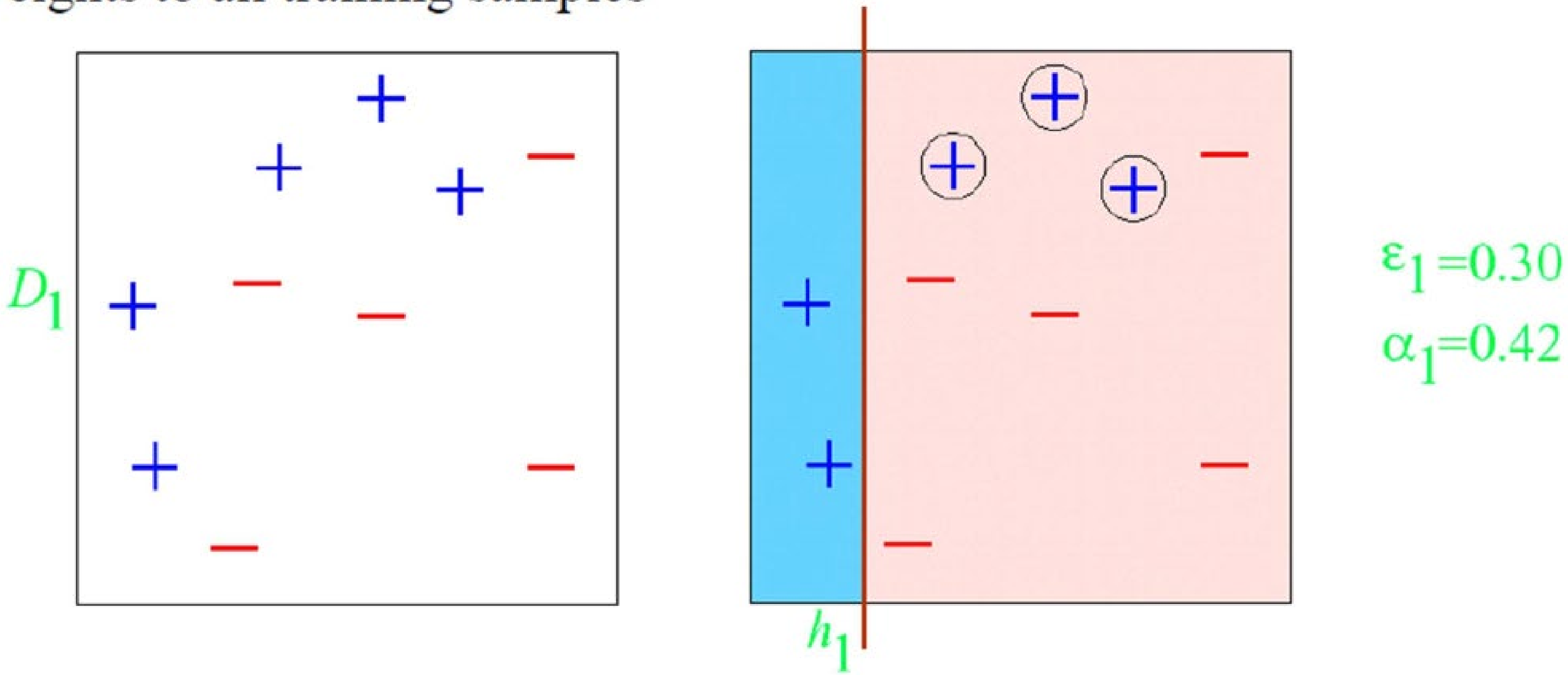
Learning with Weights

- It is often straightforward to convert a base learner to take into account an input distribution D.
  - Decision trees?
  - Neural nets?
  - Logistic regression?
- When it's not straightforward, we can resample the training data according to D

AdaBoost(Example)

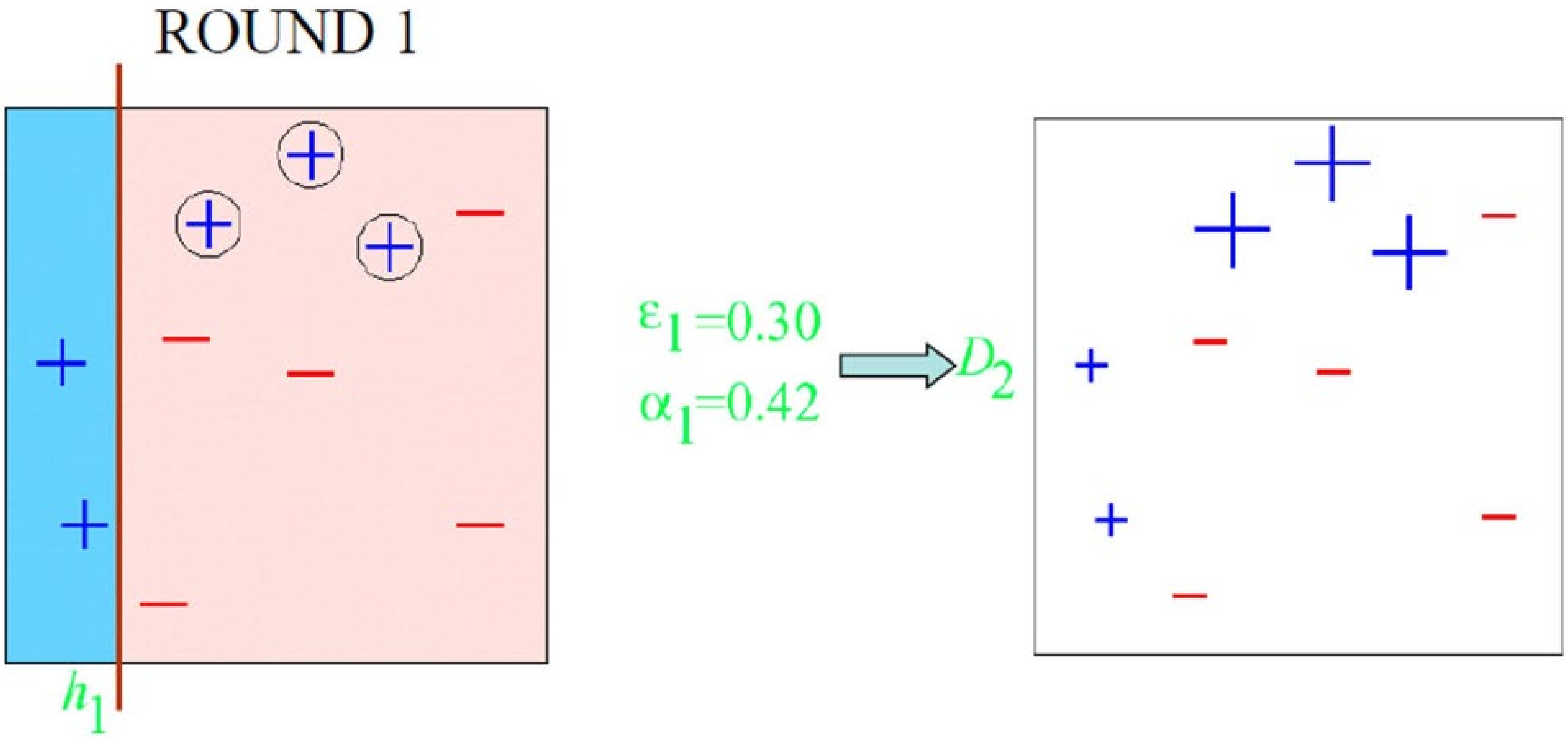
Base Learner: Decision Stump Learner (i.e. single test decision trees)

Original Training set : Equal
Weights to all training samples

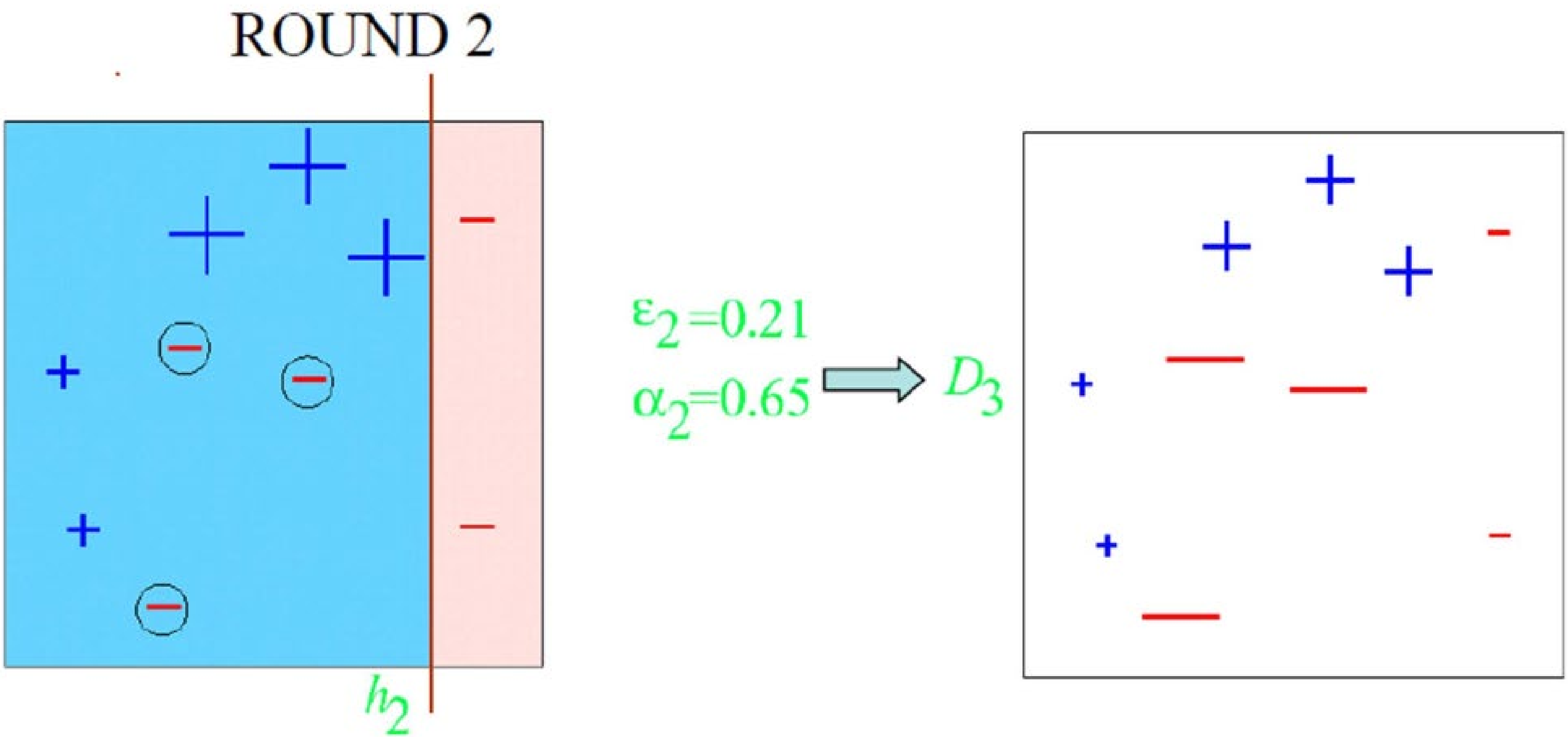


Taken from "**A Tutorial on Boosting**" by Yoav Freund and Rob Schapire

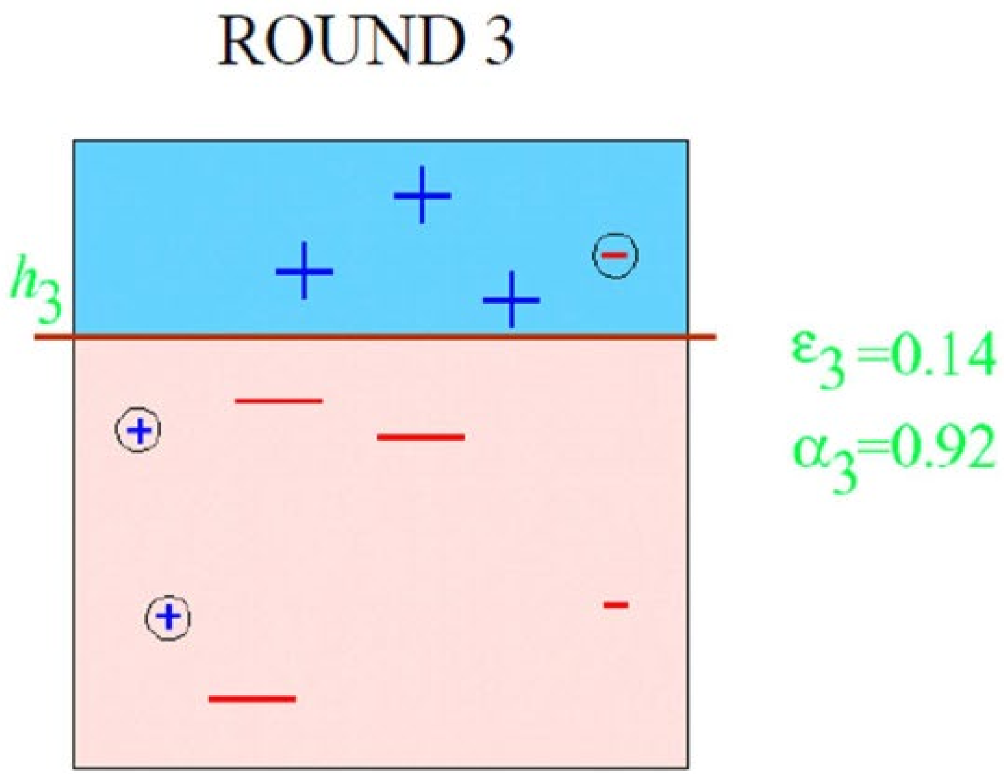
AdaBoost(Example)



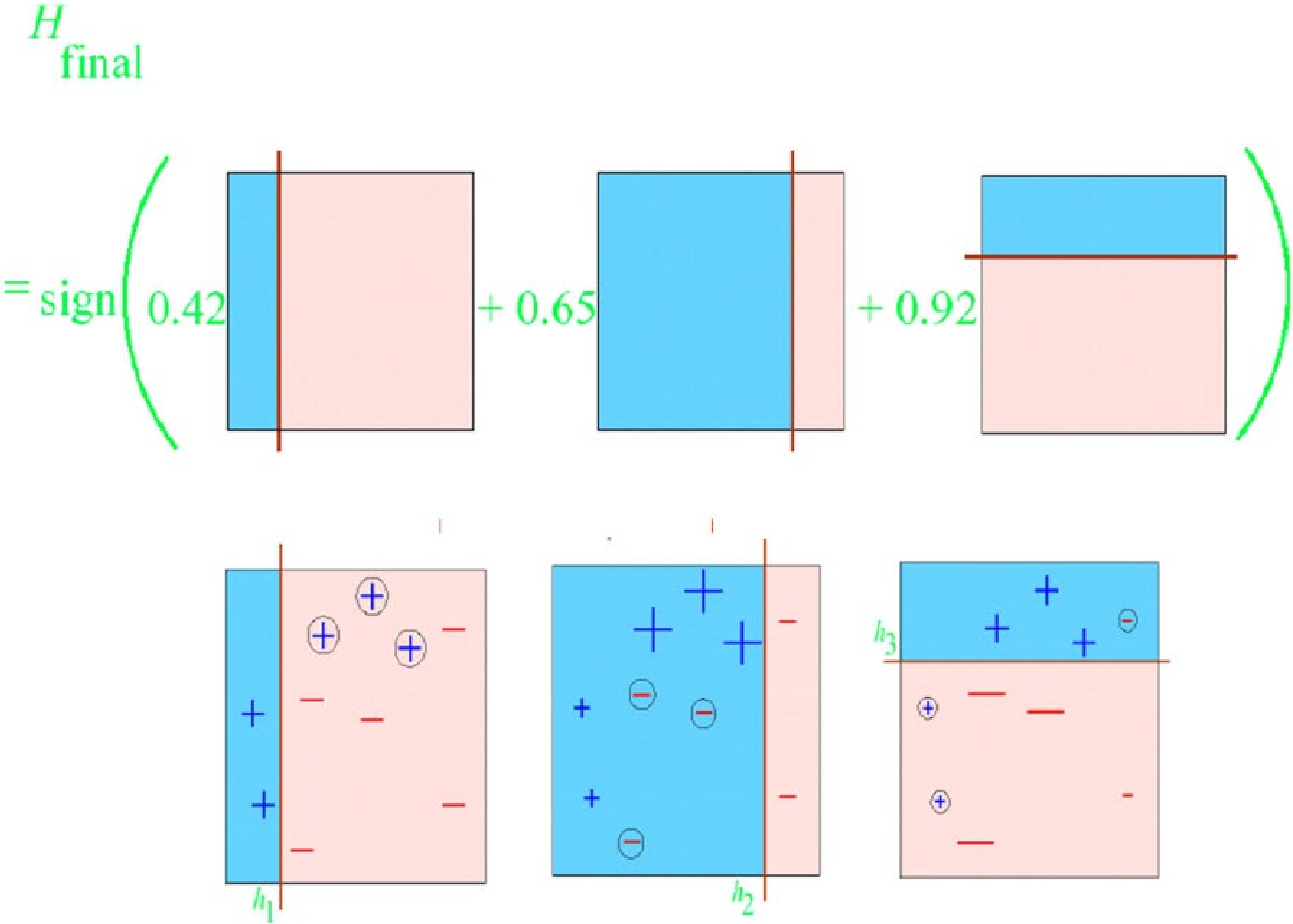
AdaBoost(Example)



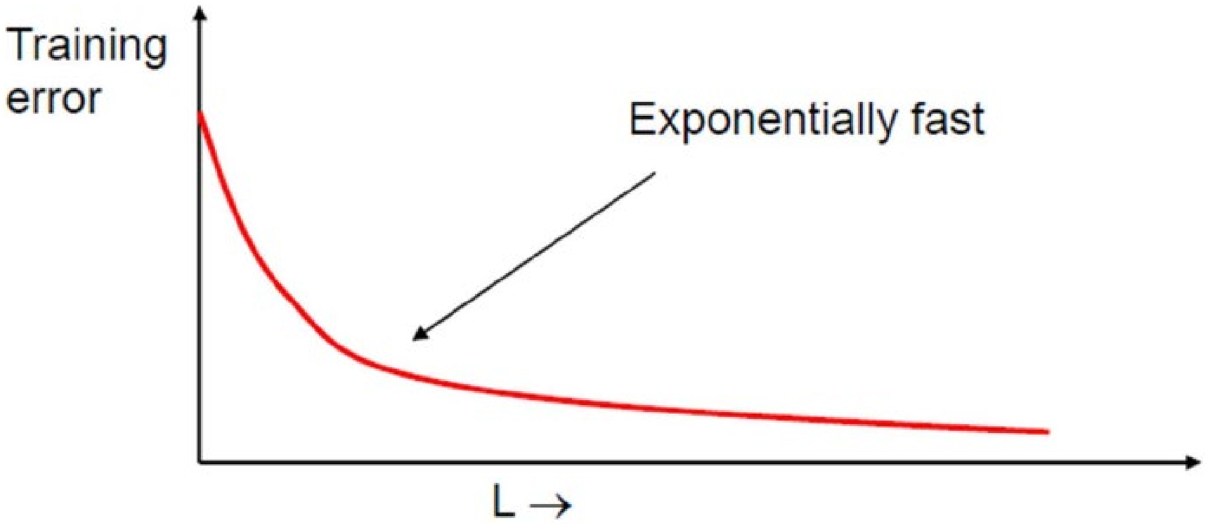
AdaBoost(Example)



AdaBoost(Example)



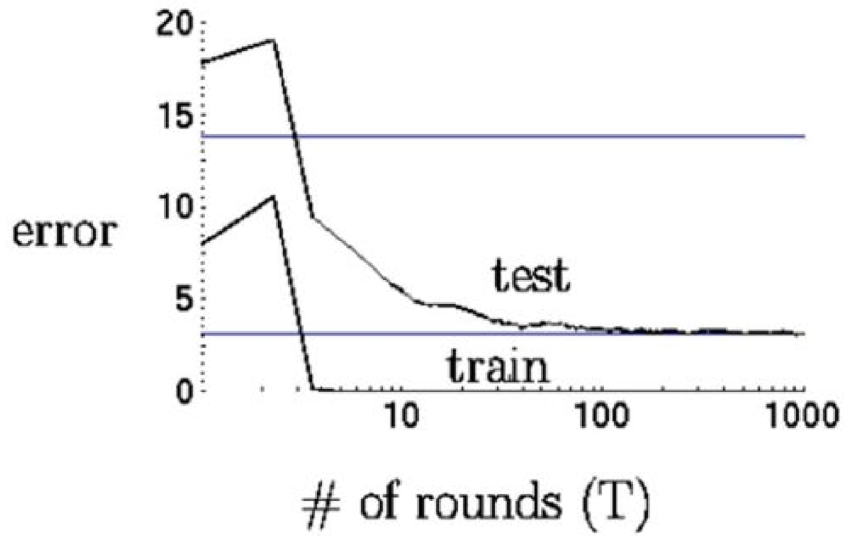
Property of Adaboost



- Suppose L is a weak learner
  - $\varepsilon_l < 0.5$ (slightly better than random guesses)
  - Training error goes to zero exponentially fast

Overfitting?

- Boosting drives training error to zero, will it overfit?
- Curious phenomenon

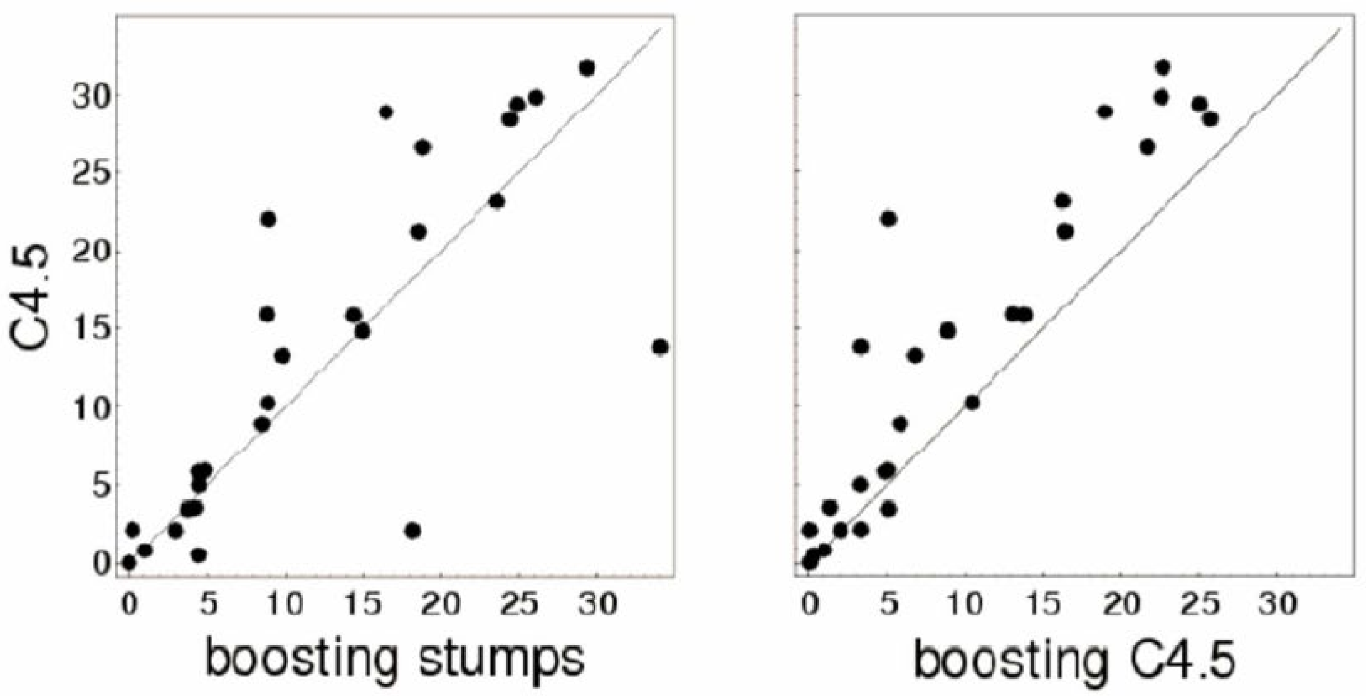


Schapire 1989.
Letter recognition

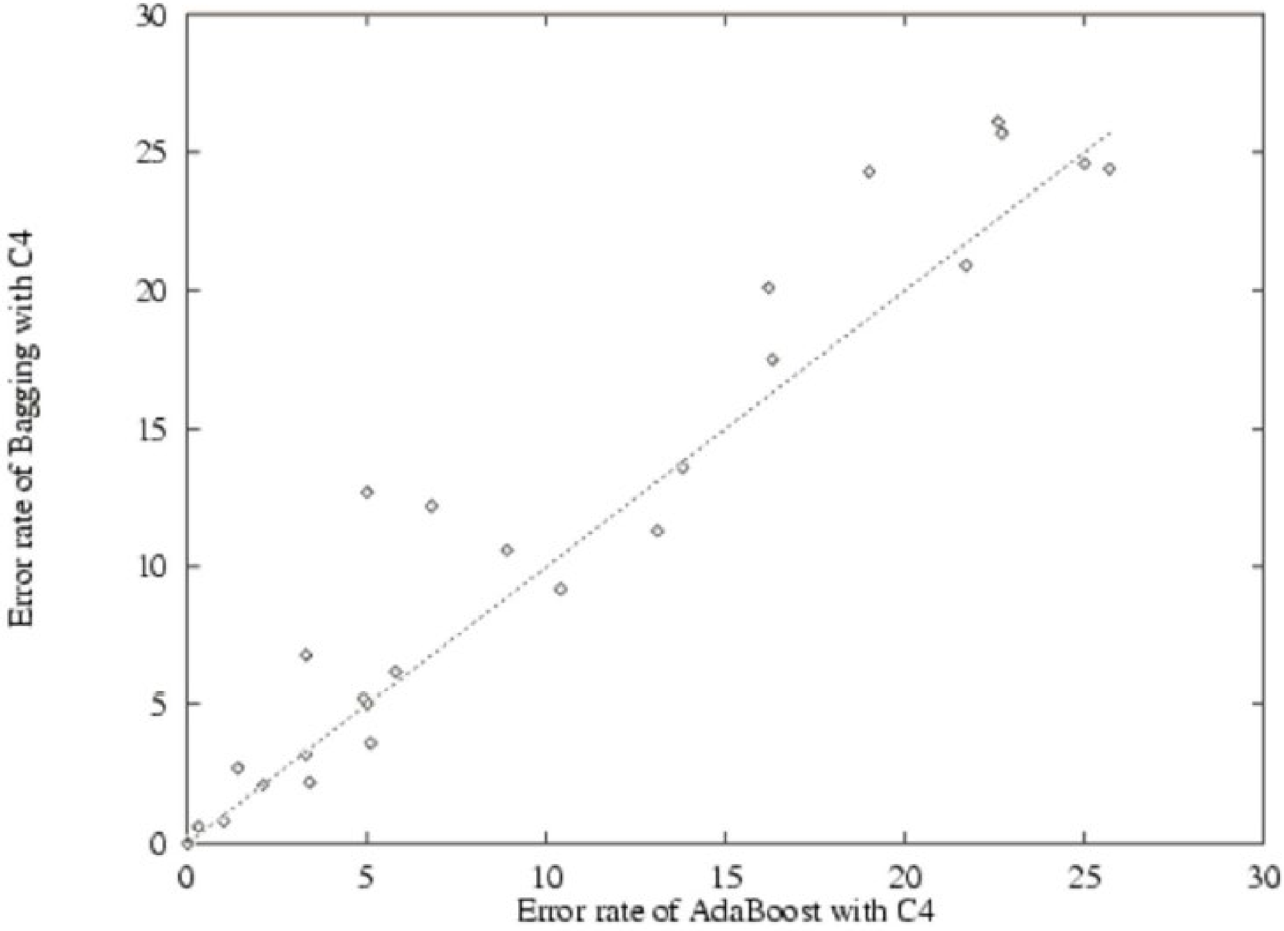
- Boosting is often robust to overfitting (not always)
- Test error continues to decrease even after training error goes to zero

Boosting Performance

- Comparing C4.5, boosting decision stumps, boosting C4.5 using 27 UCI data set
  - C4.5 is a popular decision tree learner



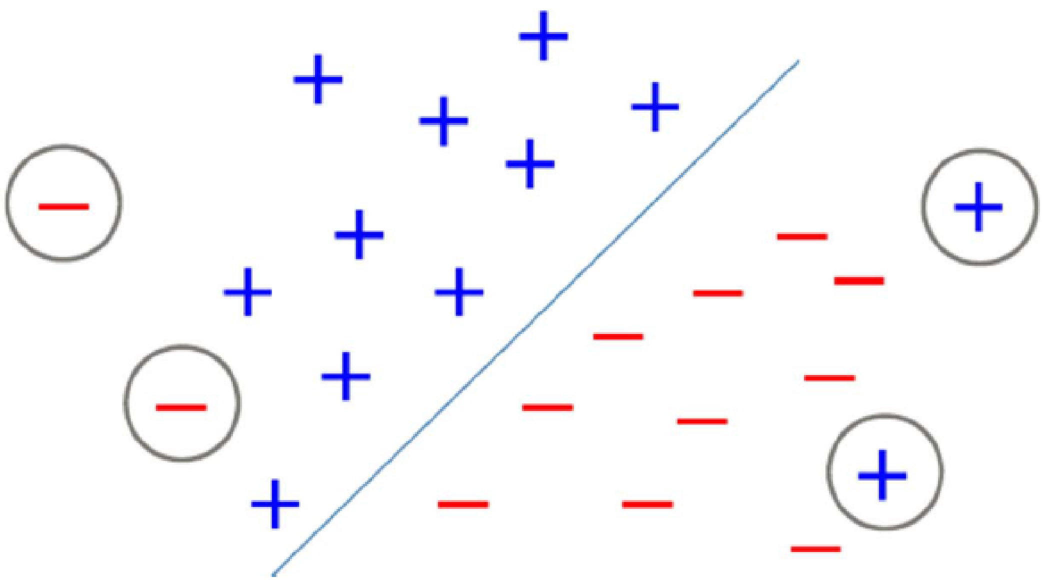
Boosting vs Bagging of Decision Trees



Pitfall of Boosting: sensitive to noise and outliers

Good ☺ : Can identify outliers since focuses on examples that are hard to categorize

Bad ☹ : Too many outliers can degrade classification performance dramatically increase time to convergence



Bias and Variance

- Bias arises when the classifier cannot represent the true function – that is, the classifier underfits the data
- Variance arises when the classifier overfits the data
- There is often a tradeoff between bias and variance

Effect of Boosting

- In the early iterations, boosting is primary a bias-reducing method
- In later iterations, it appears to be primarily a variance-reducing method

Effect of Bagging

- If the bootstrap replicate approximation were correct, then bagging would reduce variance without changing bias
- In practice, bagging can reduce both bias and variance
  - For high-bias classifiers, it can reduce bias (but may increase variance)
  - For high-variance classifiers, it can reduce variance

Summary: Bagging and Boosting

- Bagging
  - Resample data points
  - Weight of each classifier is the same
  - Only variance reduction
  - Robust to noise and outliers
- Boosting
  - Reweight data points (modify data distribution)
  - Weight of classifier vary depending on accuracy
  - Reduces both bias and variance
  - Can hurt performance with noise and outliers