Ensemble Classifiers

- IDEA:
  - do not learn a *single* classifier but learn a *set of classifiers*
  - *combine the predictions* of multiple classifiers
- MOTIVATION:
  - reduce variance: results are less dependent on peculiarities of a single training set
  - reduce bias: a combination of multiple classifiers may learn a more expressive concept class than a single classifier
- KEY STEP:
  - formation of an ensemble of *diverse* classifiers from a single training set

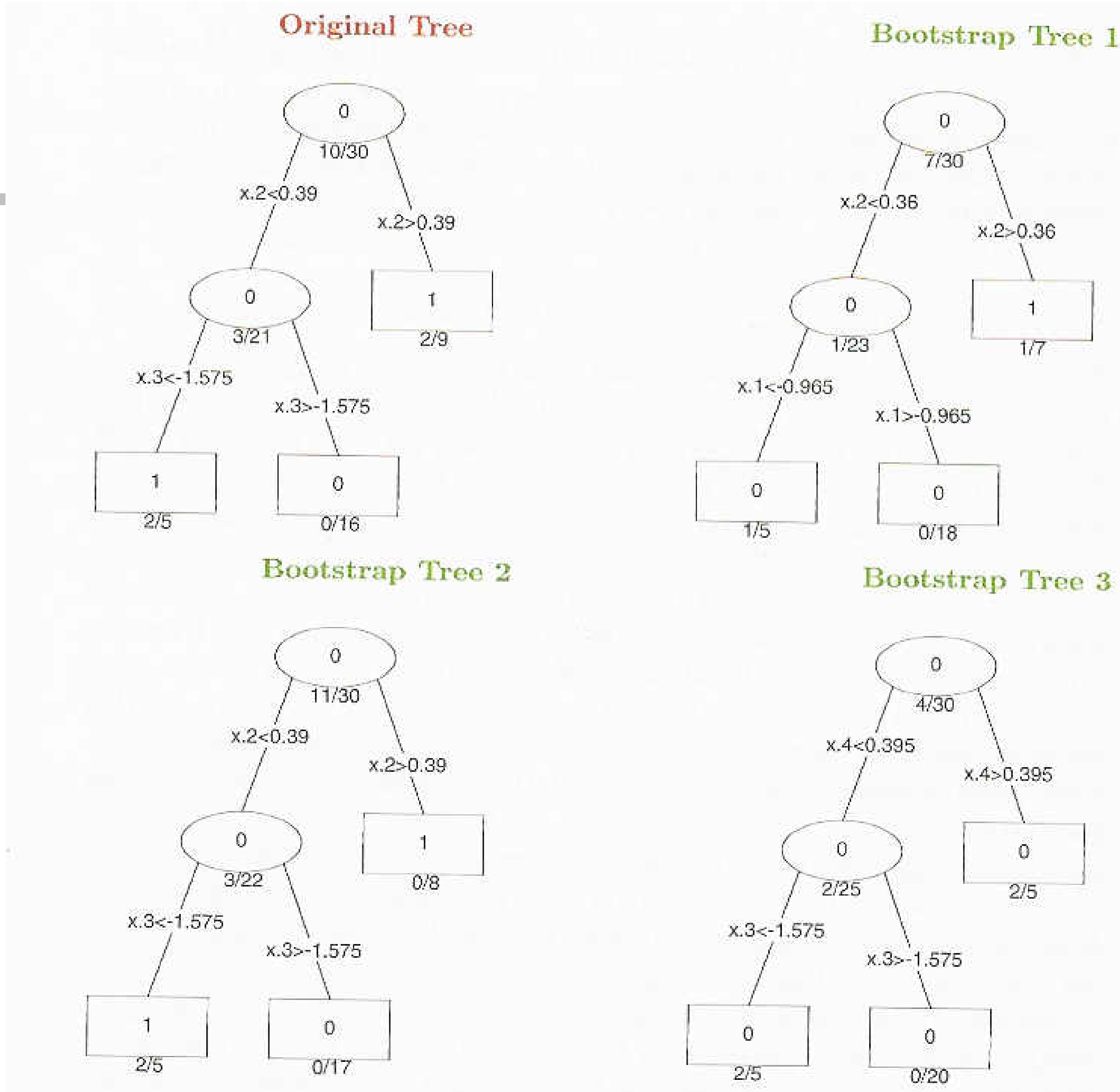
Forming an Ensemble

- Modifying the data
  - Subsampling
    - bagging
    - boosting
  - feature subsets
    - randomly feature samples
- Modifying the learning task
  - pairwise classification / round robin learning
  - error-correcting output codes
- Exploiting the algorithm characterisitics
  - algorithms with random components
    - neural networks
  - randomizing algorithms
    - randomized decision trees
  - use multiple algorithms with different characteristics
- Exploiting problem characteristics
  - e.g., hyperlink ensembles

Bagging

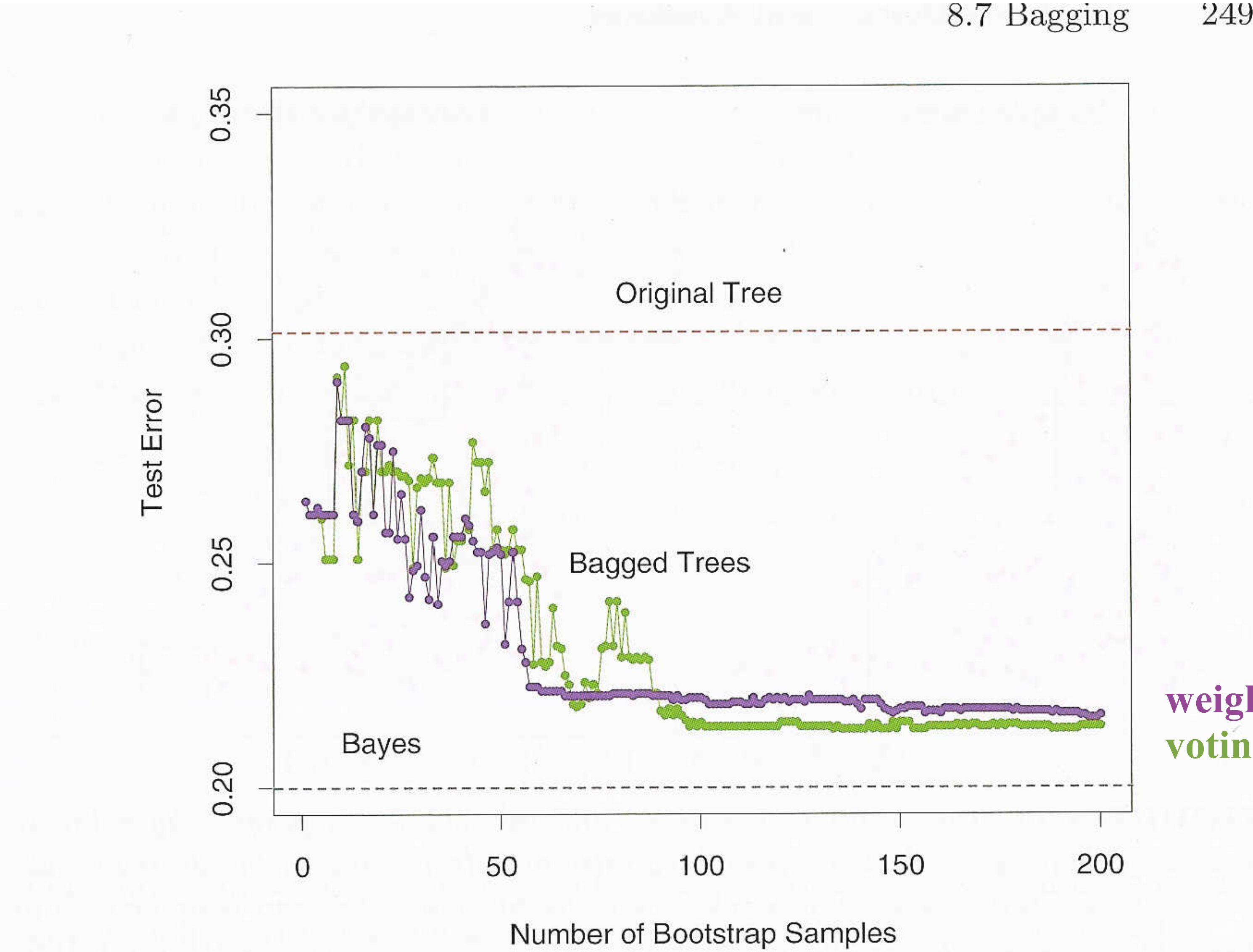
1. for $m = 1$ to M // M ... number of iterations
   a) draw (with replacement) a bootstrap sample S_m of the data
   b) learn a classifier C_m from S_m
2. for each test example
   a) try all classifiers C_m
   b) predict the class that receives the highest number of votes

- variations are possible
  - e.g., size of subset, sampling w/o replacement, etc.
- many related variants
  - sampling of features, not instances
  - learn a set of classifiers with different algorithms



from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001

Bagged Trees



weighted voting
voting

from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001

Boosting

- Basic Idea:
  - later classifiers focus on examples that were misclassified by earlier classifiers
  - weight the predictions of the classifiers with their error
- Realization
  - perform multiple iterations
    - each time using different example weights
  - weight update between iterations
    - increase the weight of incorrectly classified examples
    - this ensures that they will become more important in the next iterations
      (misclassification errors for these examples count more heavily)
  - combine results of all iterations
    - weighted by their respective error measures

Dealing with Weighted Examples

Two possibilities (→ cost-sensitive learning)

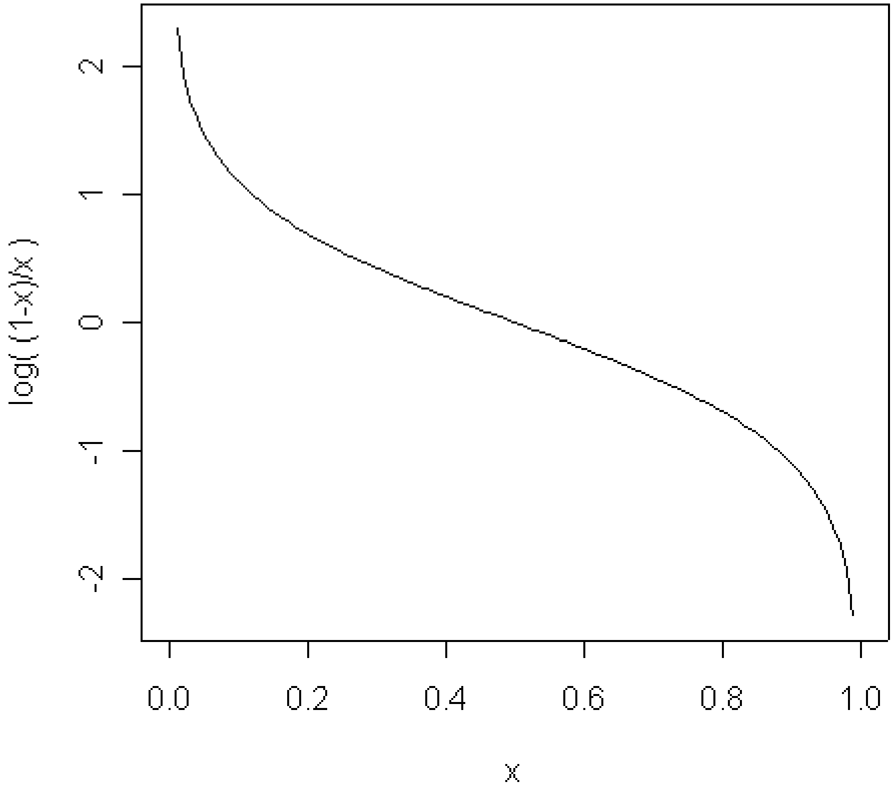
- directly
  - example e_i has weight w_i
  - number of examples $n \Rightarrow$ total example weight $\sum_{i=1}^{n} w_i$
- via sampling
  - interpret the weights as probabilities
  - examples with larger weights are more likely to be sampled
  - assumptions
    - sampling with replacement
    - weights are well distributed in [0,1]
    - learning algorithm sensible to varying numbers of identical examples in training data

Boosting – Algorithm AdaBoost

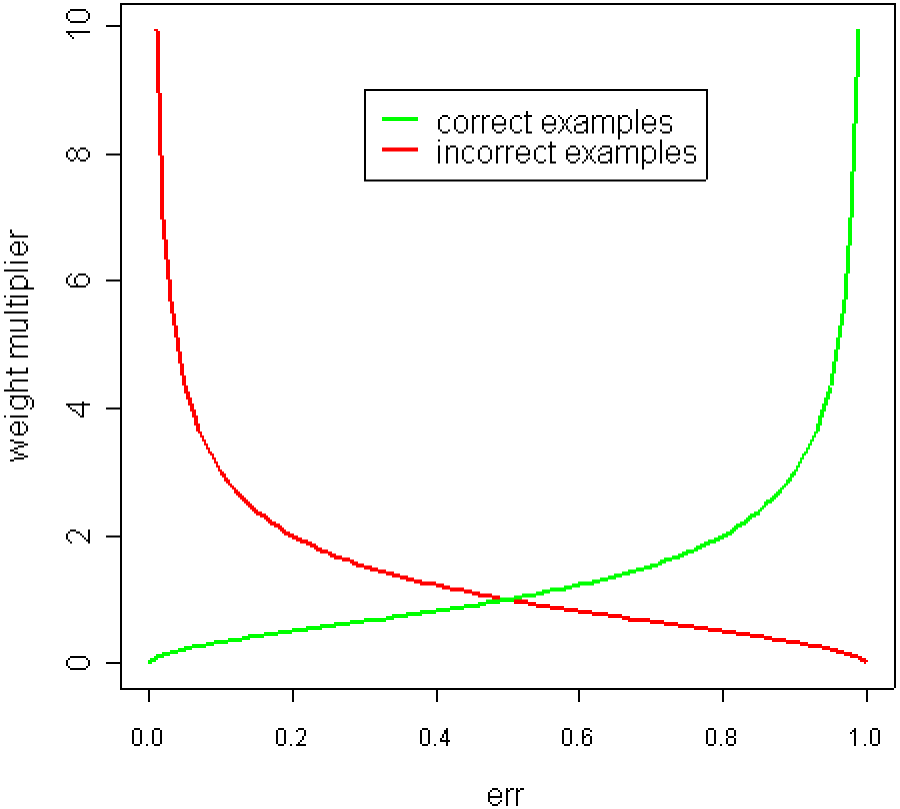
1. initialize example weights $w_i = 1/N \quad (i = 1..N)$
2. for $m = 1$ to M // M ... number of iterations
   a) learn a classifier C_m using the current example weights
   b) compute a weighted error estimate $err_m = \frac{\sum w_i \text{ of all incorrectly classified } e_i}{\sum_{i=1}^{N} w_i}$
      ($\sum_{i=1}^{N} w_i = 1$ because weights are normalized)
   c) compute a classifier weight $\alpha_m = \frac{1}{2} \log\left(\frac{1-err_m}{err_m}\right)$
   d) for all correctly classified examples e_i: $w_i \leftarrow w_i e^{-\alpha_m}$
   e) for all incorrectly classified examples e_i: $w_i \leftarrow w_i e^{\alpha_m}$
      (update weights so that sum of correctly classified examples equals sum of incorrectly classified examples)
   f) normalize the weights w_i so that they sum to 1
3. for each test example
   a) try all classifiers C_m
   b) predict the class that receives the highest sum of weights α_m

Illustration of the Weights

- Classifier Weights α_m
  - differences near 0 or 1 are emphasized

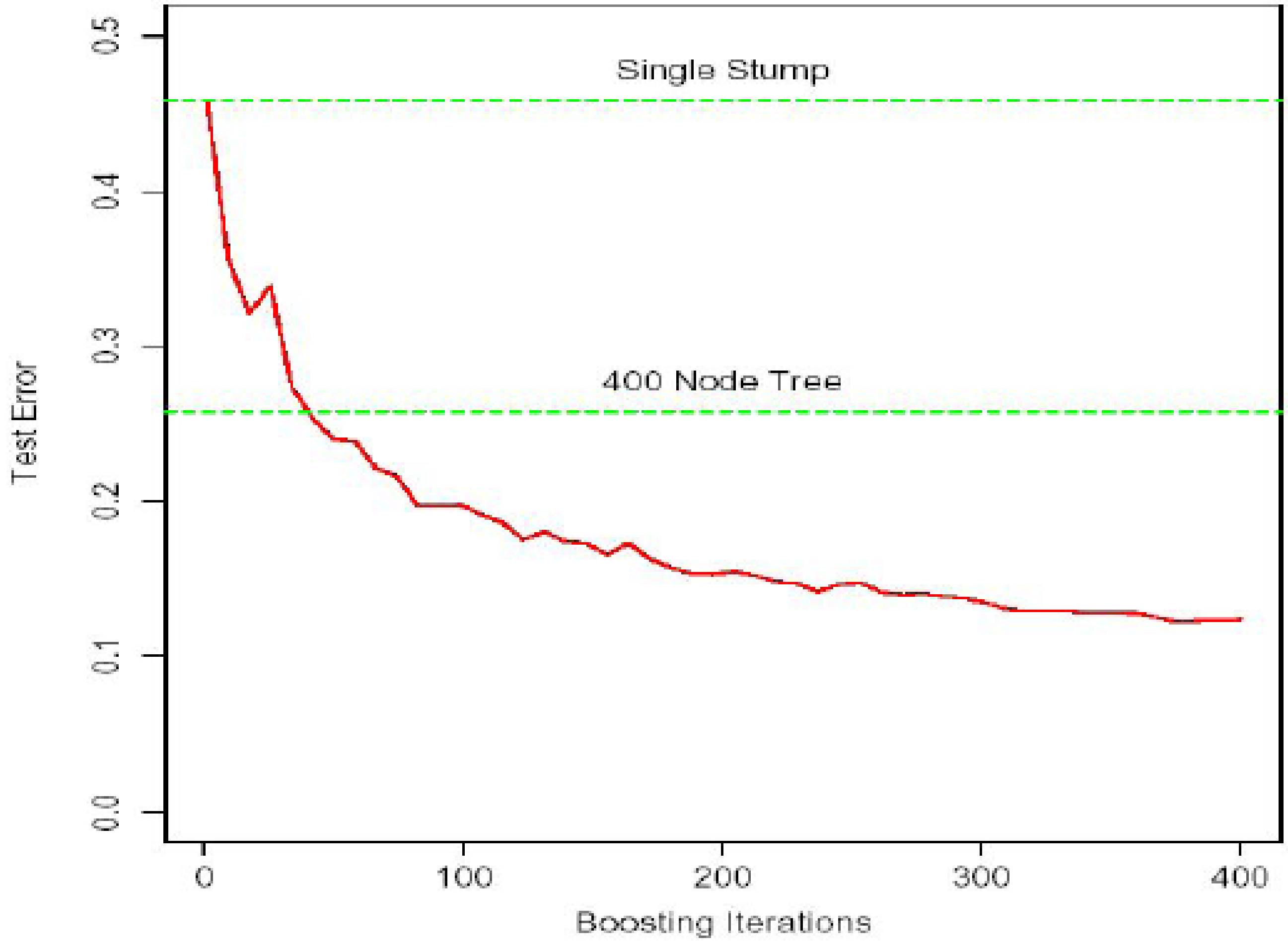


- Example Weights
  - multiplier for correct and incorrect examples, depending on error



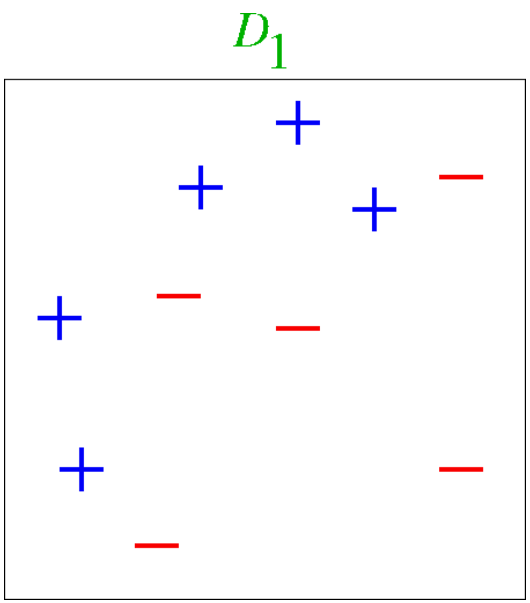
Boosting – Error rate example

- boosting of decision stumps on simulated data



from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001

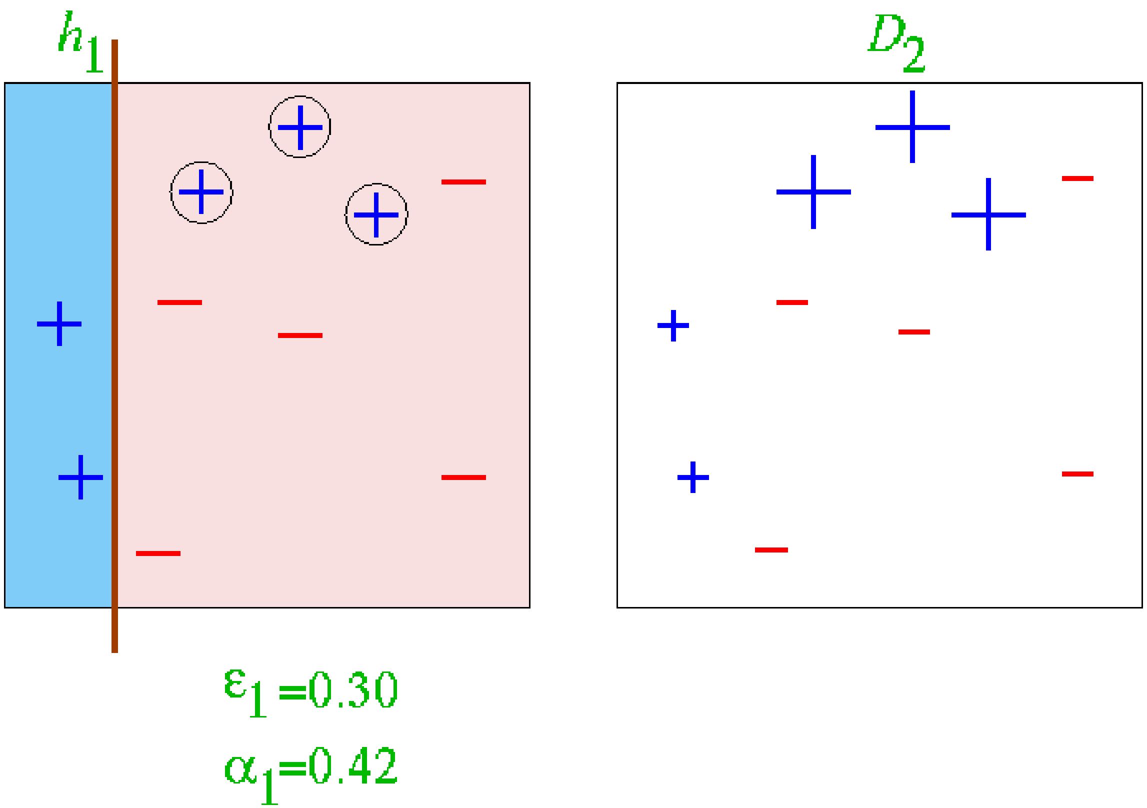
Toy Example



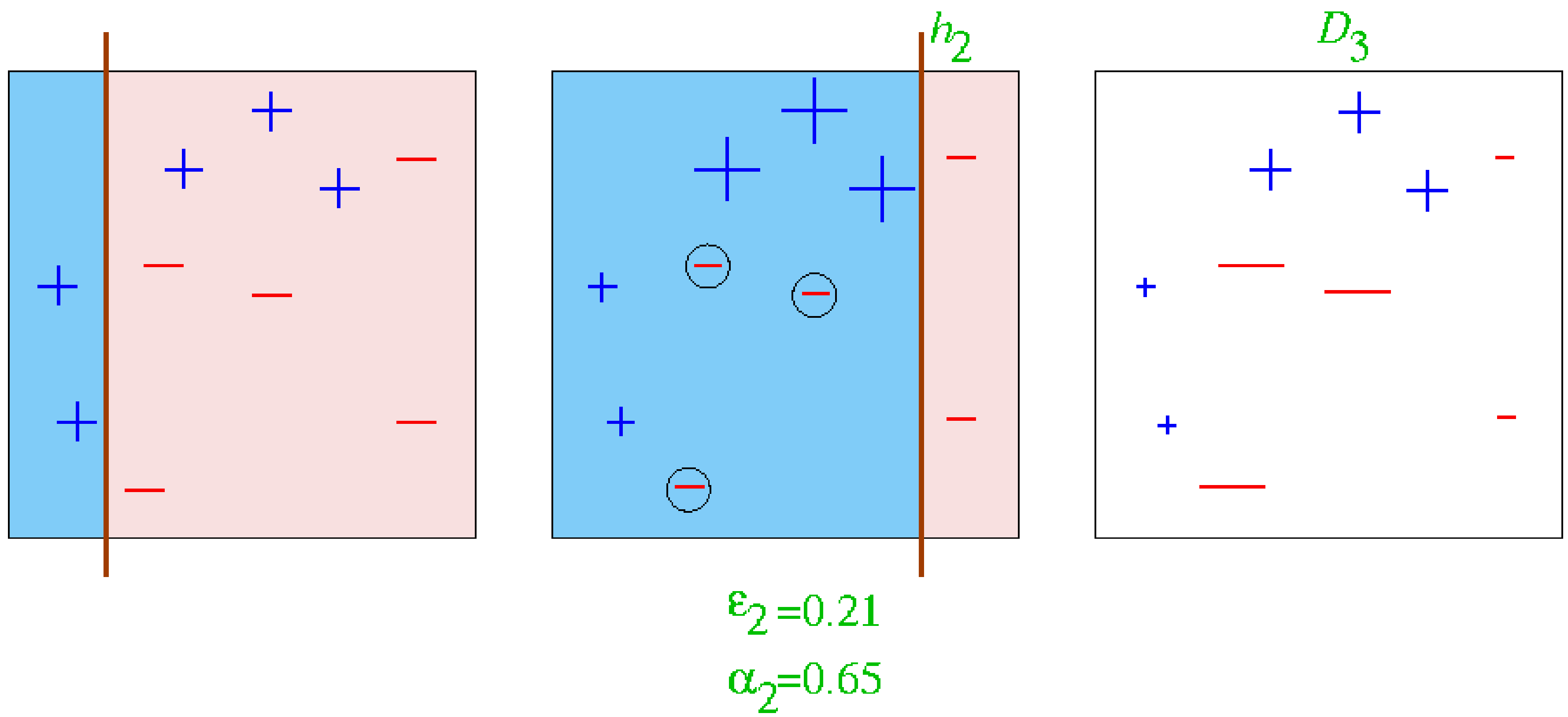
(taken from Verma & Thrun, Slides to CALD Course CMU 15-781, Machine Learning, Fall 2000)

- An Applet demonstrating AdaBoost
  - http://www.cse.ucsd.edu/~yfreund/adaboost/

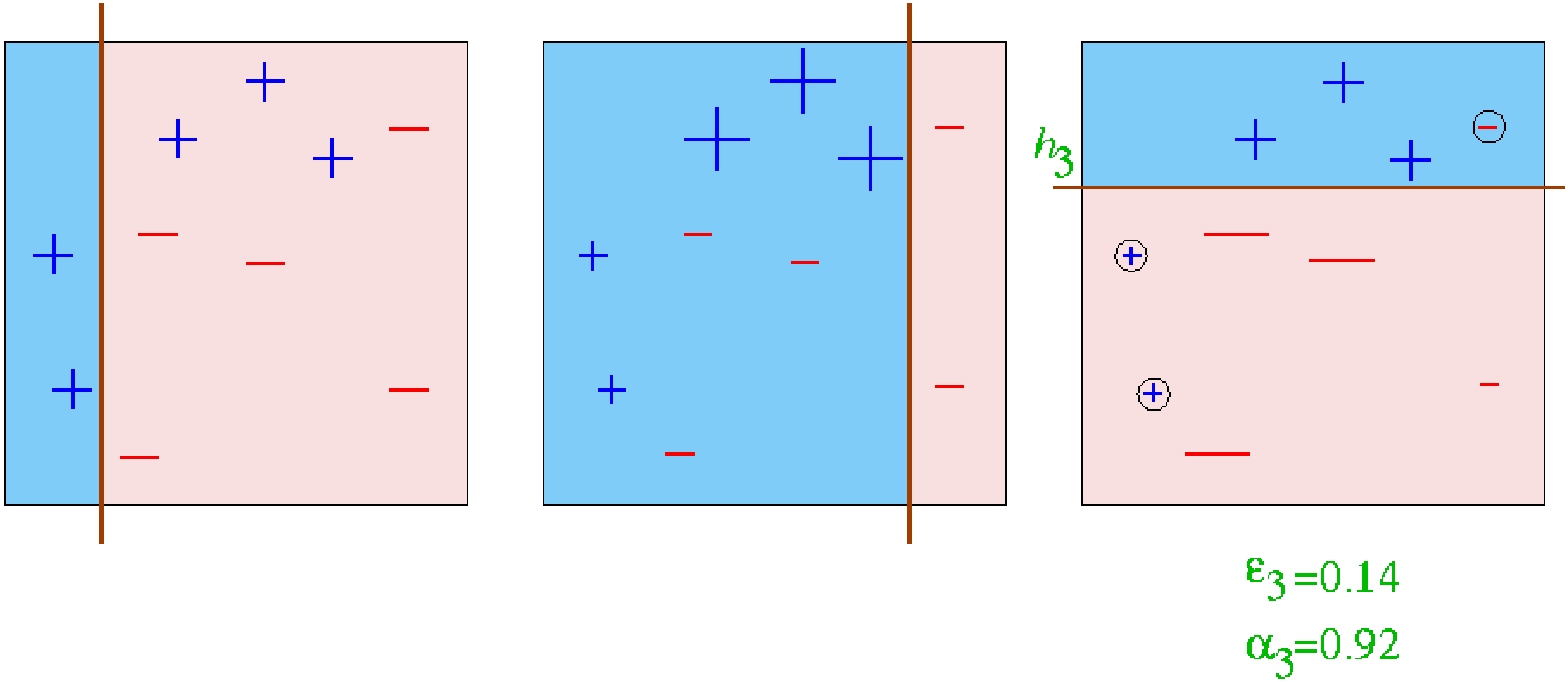
Round 1



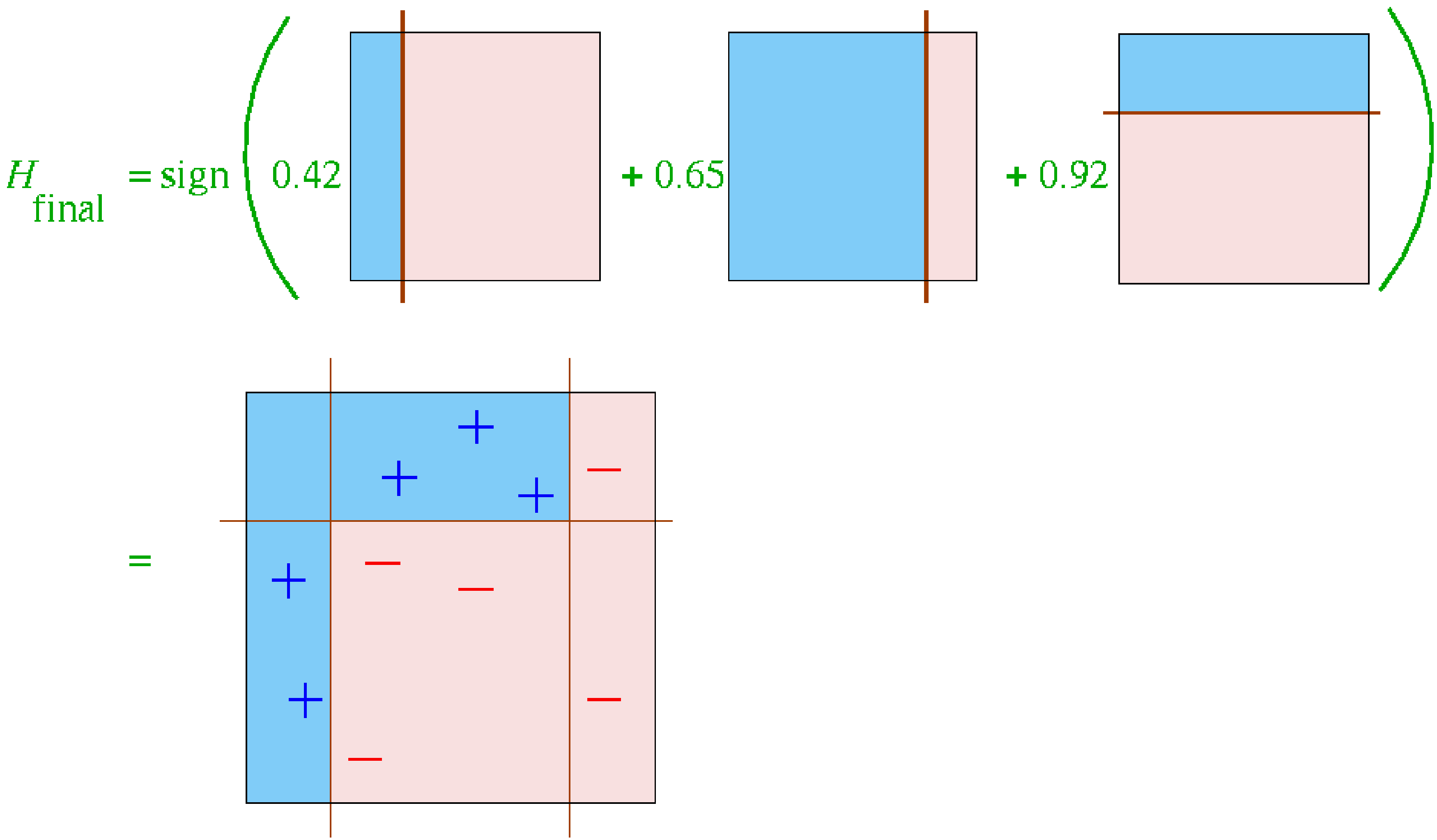
Round 2



Round 3



Final Hypothesis



Example

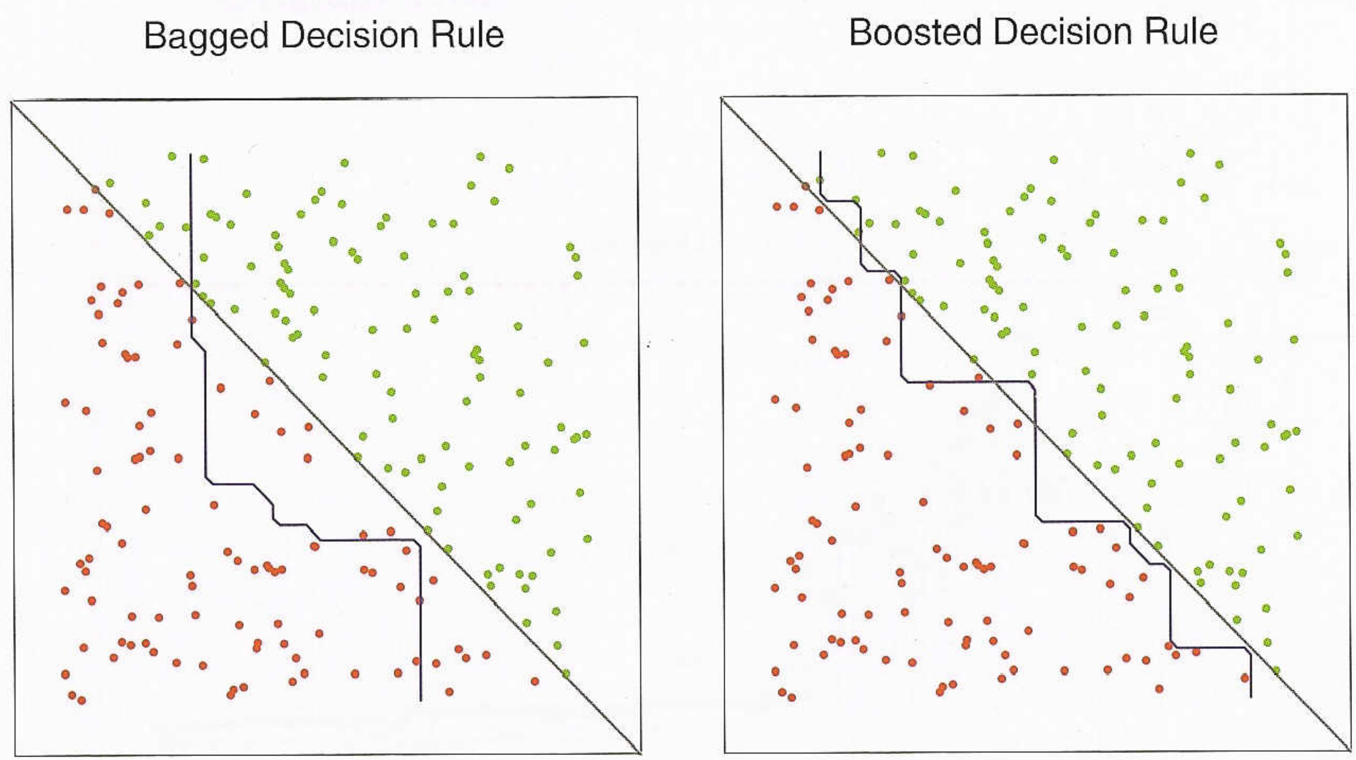


FIGURE 8.11. *Data with two features and two classes, separated by a linear boundary. Left panel: decision boundary estimated from bagging the decision rule from a single split, axis-oriented classifier. Right panel: decision boundary from boosting the decision rule of the same classifier. The test error rates are* 0.166, *and* 0.065 *respectively. Boosting is described in Chapter 10.*

from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001

Comparison Bagging/Boosting

- Bagging
  - noise-tolerant
  - produces better class probability estimates
  - not so accurate
  - statistical basis
  - related to random sampling
- Boosting
  - very susceptible to noise in the data
  - produces rather bad class probability estimates
  - if it works, it works really well
  - based on learning theory (statistical interpretations are possible)
  - related to windowing

Combining Predictions

- voting
  - each ensemble member votes for one of the classes
  - predict the class with the highest number of vote (e.g., bagging)
- weighted voting
  - make a *weighted* sum of the votes of the ensemble members
  - weights typically depend
    - on the classifiers confidence in its prediction (e.g., the estimated probability of the predicted class)
    - on error estimates of the classifier (e.g., boosting)
- stacking
  - Why not use a classifier for making the final decision?
  - training material are the class labels of the training data and the (cross-validated) predictions of the ensemble members

Stacking

- Basic Idea:
  - learn a function that combines the predictions of the individual classifiers
- Algorithm:
  - train n different classifiers $C_1 \ldots C_n$ (the *base classifiers*)
  - obtain predictions of the classifiers for the training examples
    - better do this with a cross-validation!
  - form a new data set (the *meta data*)
    - **classes**
      - the same as the original dataset
    - **attributes**
      - one attribute for each base classifier
      - value is the prediction of this classifier on the example
  - train a separate classifier M (the *meta classifier*)

Stacking (2)

- Example:

Attributes			Class
x_{11}	$\dots$	x_{1n_a}	t
x_{21}	$\dots$	x_{2n_a}	f
$\dots$	$\dots$	$\dots$	$\dots$
x_{n_e1}	$\dots$	$x_{n_en_a}$	t

training set

C_1	C_2	$\dots$	C_{n_c}
t	t	$\dots$	f
f	t	$\dots$	t
$\dots$	$\dots$	$\dots$	$\dots$
f	f	$\dots$	t

predictions of the classfiers

C_1	C_2	$\dots$	C_{n_c}	Class
t	t	$\dots$	f	t
f	t	$\dots$	t	f
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
f	f	$\dots$	t	t

training set for stacking

- Using a stacked classifier:
  - try each of the classifiers $C_1 \dots C_n$
  - form a feature vector consisting of their predictions
  - submit this feature vectors to the meta classifier M