

Ensemble Learning

CS534

Key Concepts

Bagging, Bootstrapping – sampling with replacement

Bias variance decomposition and tradeoff

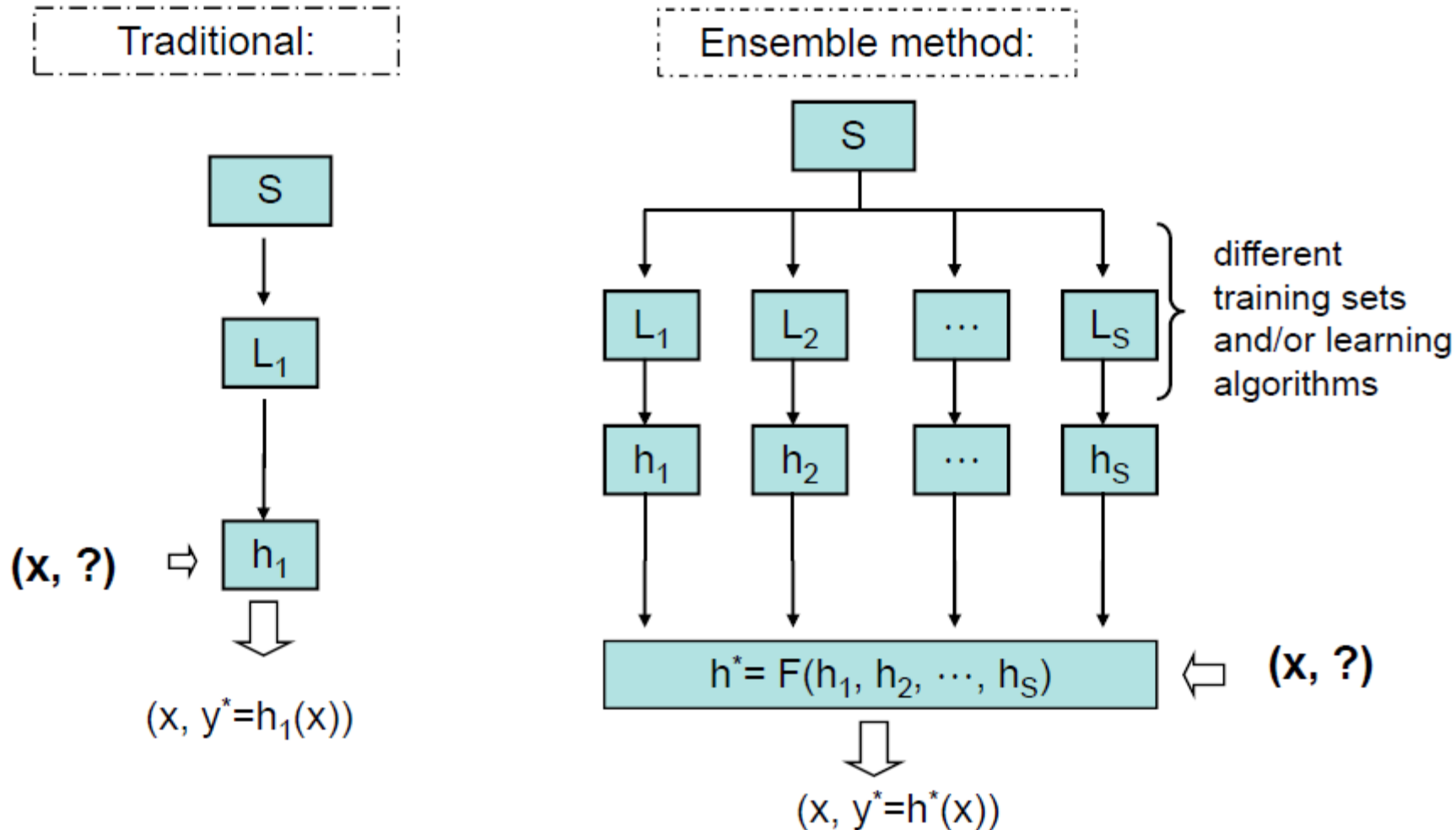
Bagging's impact on bias and variance

Random Forest

Boosting

Functional Gradient descent for learning additive models

Ensemble Learning



Instead of trying to find the best single classifier, learn many of them and then aggregate and let them vote.

How to generate ensembles?

- There have been a wide range of methods developed
 - Main idea is that we need to make them diverse
- We will study some popular approaches
 - Bagging (and Random Forest, a variant that builds de-correlated trees)
 - Boosting
- Both methods take a single (base) learning algorithm and generate ensembles

Base Learning Algorithm

- We are given a '*black box*' learning algorithm *Learn* referred to as the base learner.

Protocol to *Learn*:

Input:

S - set of labeled training instances.

Output:

h - a hypothesis from hypothesis space H .

Bootstrap Aggregating (Bagging)

- Leo Breiman, “Bagging Predictors”, *Machine Learning*, 24, 123-140 (1996)
- Create many different training sets by sampling the original training set and learn a hypothesis for each dataset.
- Resulting hypotheses will vary due to using different training sets
- Combine these hypotheses using majority vote

Bagging Algorithm

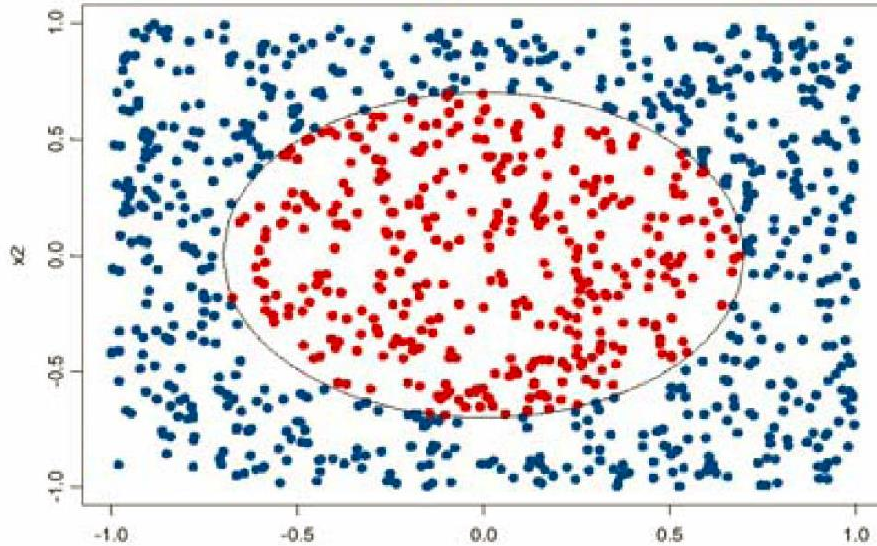
Given training set S , bagging works as follows:

1. Create T ***bootstrap samples*** $\{S_1, \dots, S_T\}$ of S as follows:
 - Each S_i is generated by randomly drawing $|S|$ examples from S **with replacement**
2. For each $i = 1, \dots, T$, $h_i = \text{Learn}(S_i)$
3. Output $H = \langle \{h_1, \dots, h_T\}, \text{majorityVote} \rangle$

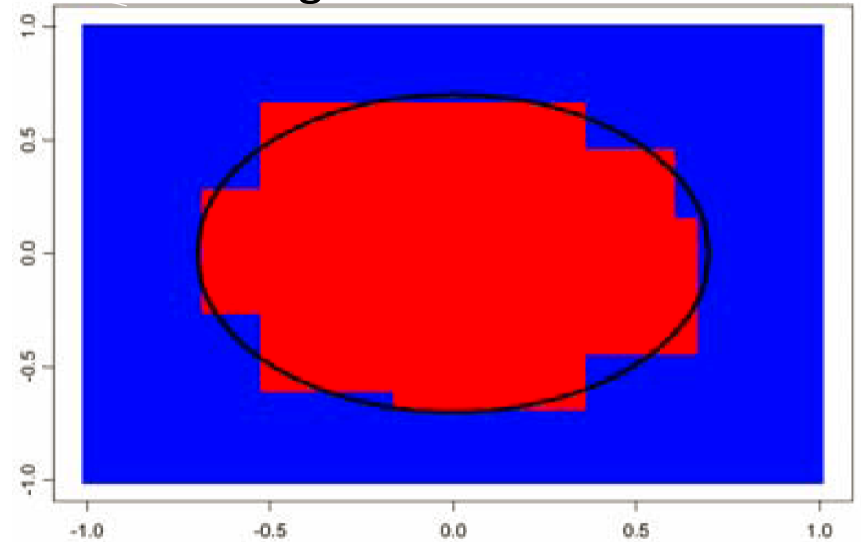
With large $|S|$, each S_i will contain $1 - \frac{1}{e} \approx 63.2\%$ unique examples

A visual example

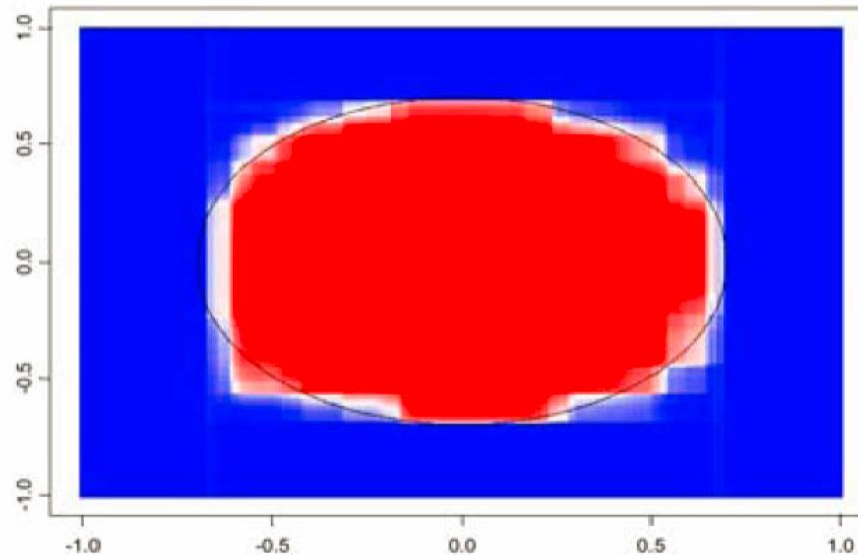
Target concept



Single decision tree

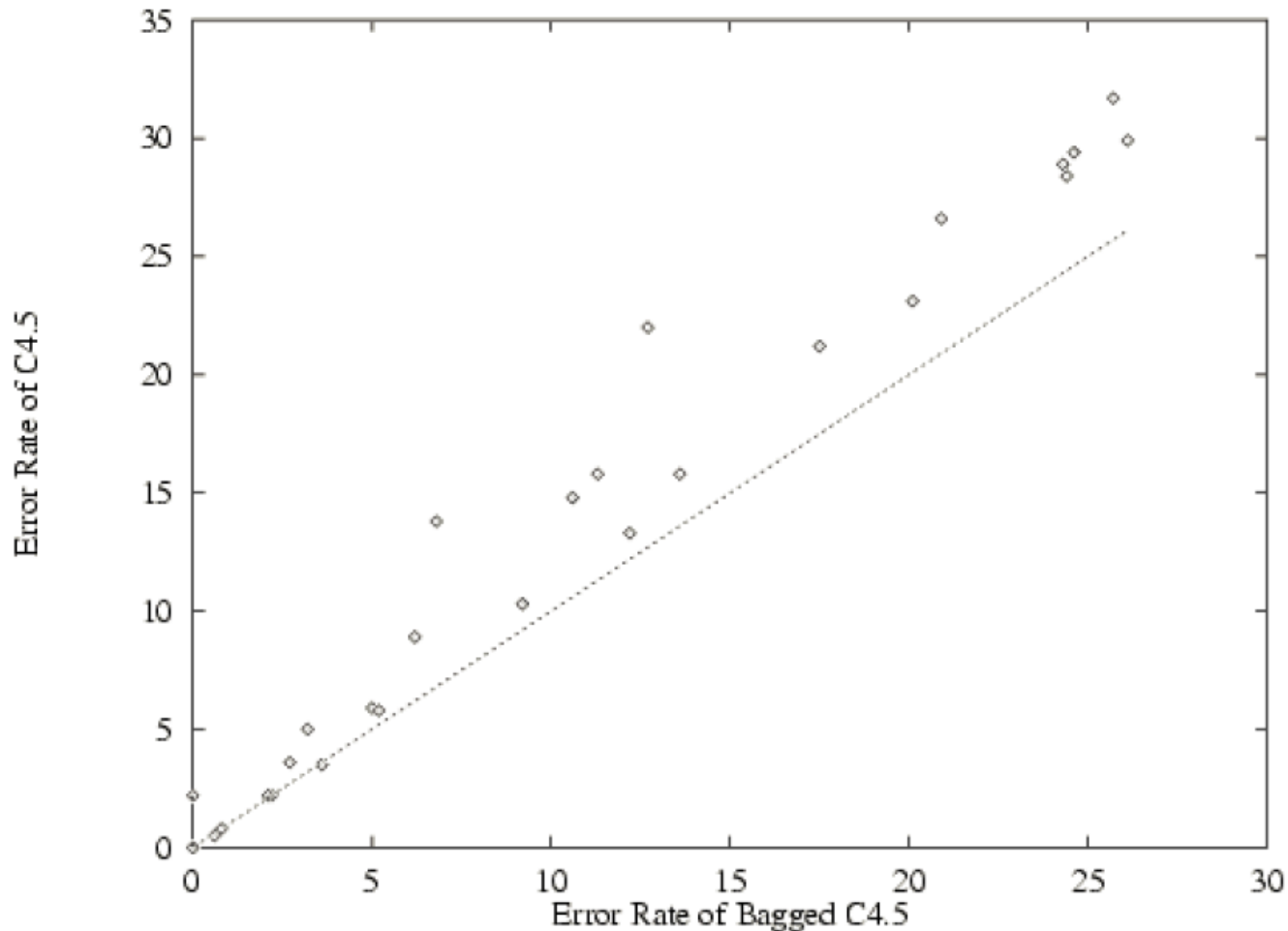


100 bagged decision tree



Bagging Decision Trees

(Freund & Schapire)



Comparison of the error of regular decision tree and bagged decision tree on a collection of UCI benchmark datasets, each data set is denoted by a single marker.

The Bias-Variance Decomposition

Suppose our training data is in the form of (\mathbf{x}, t) , where the target t is generated by some underlying function $h(\mathbf{x})$ plus some *noise*:

$$t = h(\mathbf{x}) + \text{noise}$$

Learning from a randomly sampled training set \mathcal{D} , let $y(\mathbf{x}; \mathcal{D})$ denote the learned model, what can we say about its expected loss?

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x}$$

$$\text{variance} = \int \mathbb{E}_{\mathcal{D}} [\{y(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}; \mathcal{D})]\}^2] p(\mathbf{x}) d\mathbf{x}$$

$$\text{noise} = \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt$$

Bias and variance trade-off

- Simple basic classifiers/models
 - High bias because of limited capacity
 - Low variance because they tend to be stable
- Complex classifiers/models
 - Low bias because they are flexible enough to capture the target concept
 - High variance (unstable) – slight change of training data leads to large change on the learned model

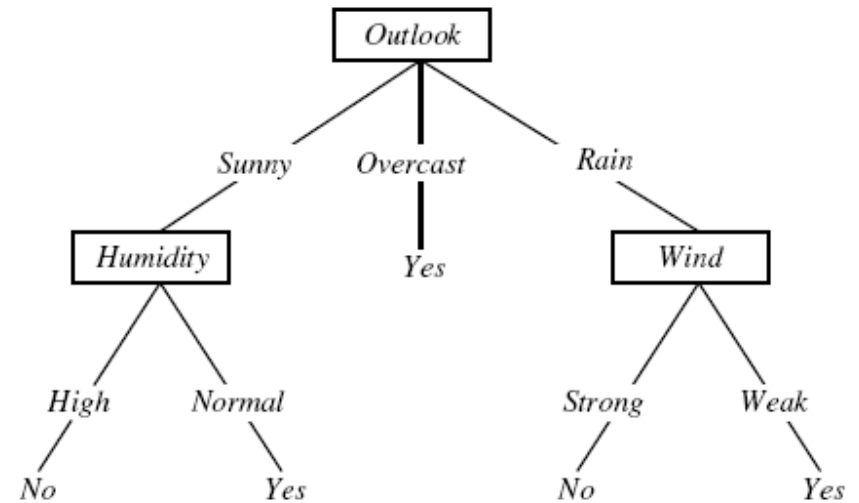
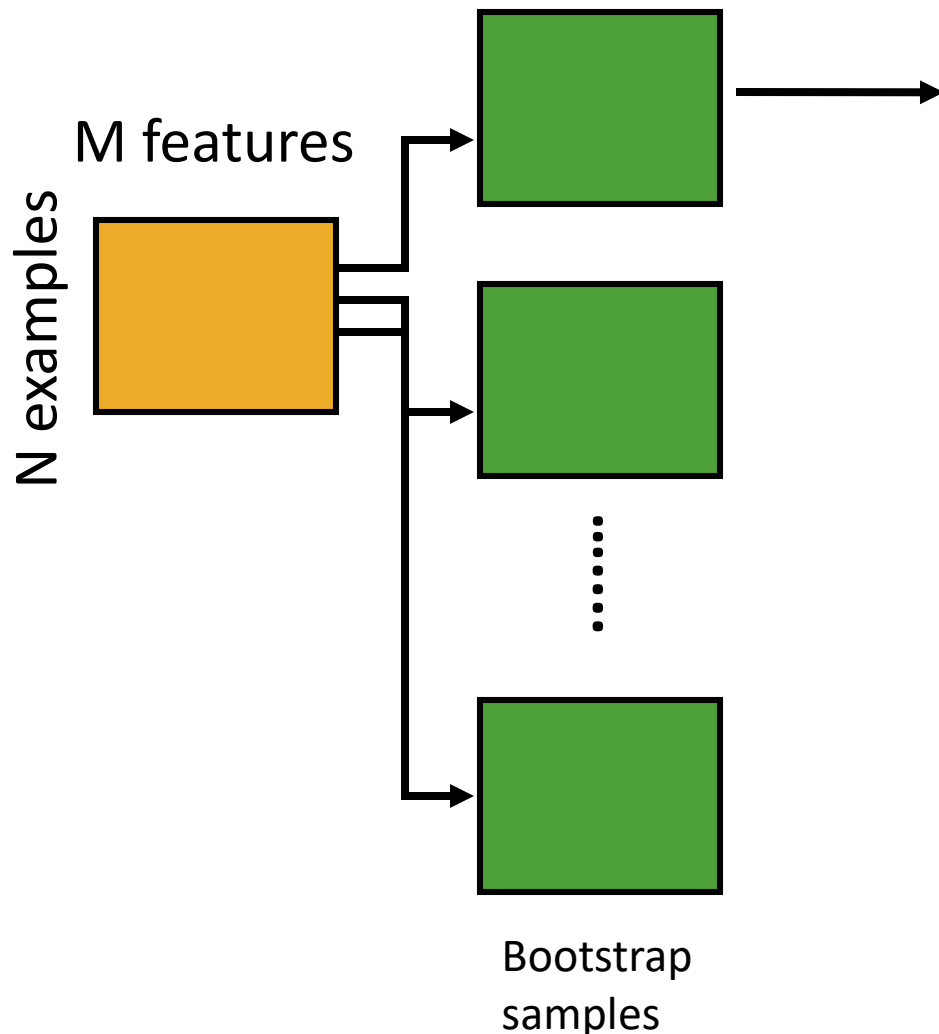
Question: how does bagging influence different parts of this loss decomposition?

Bias? , Variance?, Noise?

Random Forest

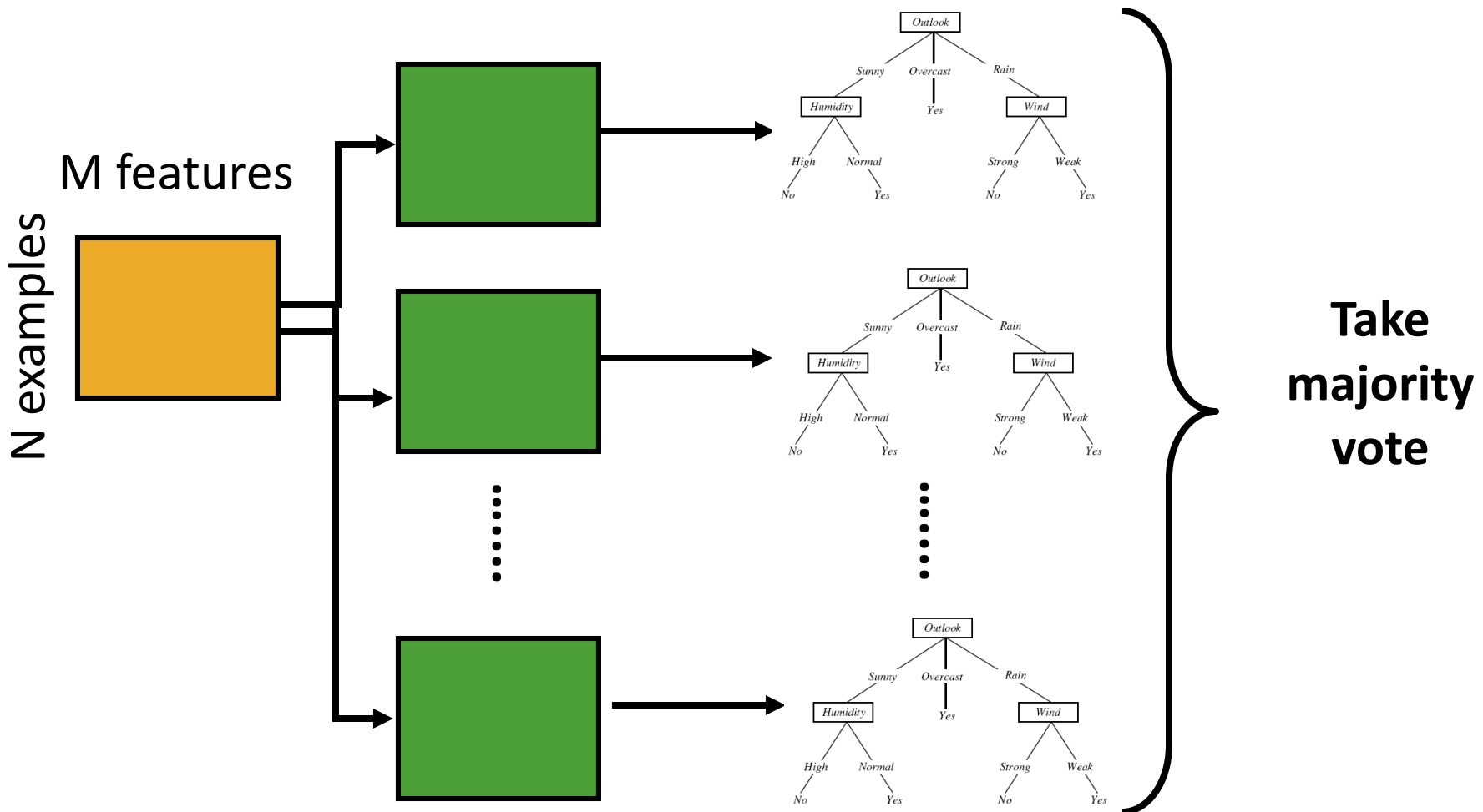
- An extension to bagging
- Builds an ensemble of **de-correlated** decision trees – this will further increase the variance reduce capability

Random Forest Classifier



- Each bootstrapped sample is used to build a tree
- When building the tree, each node only **choose from $m < M$ randomly sampled features**
- Gini index is used to select the test

Random Forest Classifier



Random forest learns trees that makes de-correlated errors

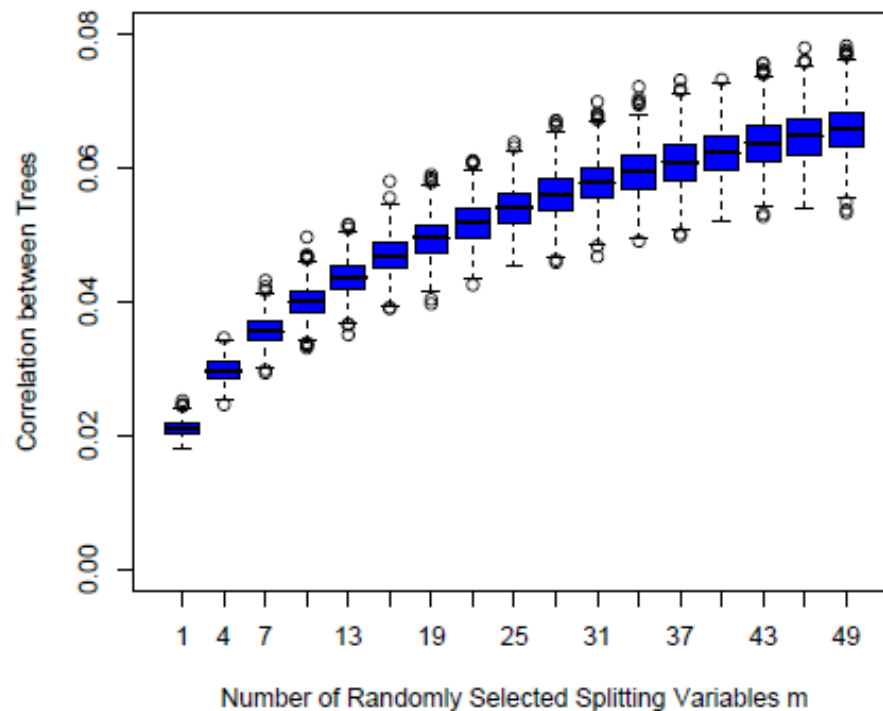


FIGURE 15.9. *Correlations between pairs of trees drawn by a random-forest regression algorithm, as a function of m . The boxplots represent the correlations at 600 randomly chosen prediction points x .*

Random forest

- Available package:
- http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- To read more:
- <http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf>

Boosting

- Its iterative.
 - **Bagging** : Individual classifiers were independent.
 - **Boosting**:
 - Try to force different classifiers to focus on different input space, and hence make different errors
 - In each iteration, focus more on **errors from previous classifiers**, and less on examples that are already correct
 - Each classifier is measured in terms of its accuracy, more accurate classifier gets more weight in final voting

Some Boosting History

(Optional Reading)

- The idea of boosting began with a learning theory question first asked in the late 80's.
- The question was answered in 1989 by Robert Schapire resulting in the first theoretical boosting algorithm
- Schapire and Freund later developed a practical boosting algorithm called Adaboost
- Many empirical studies show that Adaboost is highly effective (very often they outperform ensembles produced by bagging)

History: Strong vs weak learning (Optional Reading)

Strong Learning	Weak Learning
\exists algorithm A	\exists algorithm A
$\forall c \in \mathcal{C}$	$\exists \gamma > 0$
$\forall D$	$\forall c \in \mathcal{C}$
$\forall \epsilon > 0$	$\forall D$
$\forall \delta > 0$	$\forall \epsilon \geq \frac{1}{2} - \gamma$ say, $\epsilon=0.49$
A produces $h \in \mathcal{H}$:	A produces $h \in \mathcal{H}$:
$Pr[err(h) > \epsilon] \leq \delta$	$Pr[err(h) > \epsilon] \leq \delta$

Strong = weak?

Strong = Weak PAC learning (Optional reading)

The key idea is that we can learn a little on every distribution

Produce 3 hypothesis as follows

h_1 is the result of applying *Learn* to all training data.

h_2 is the result of applying *Learn* to filtered data distribution such that h_1 has only 50% accuracy on the data.

(e.g., to generate an example flip a coin, if head then draw examples until h_1 makes an error, and give it to *Learn*; if tail then wait until h_1 is correct, and give it to *Learn*)

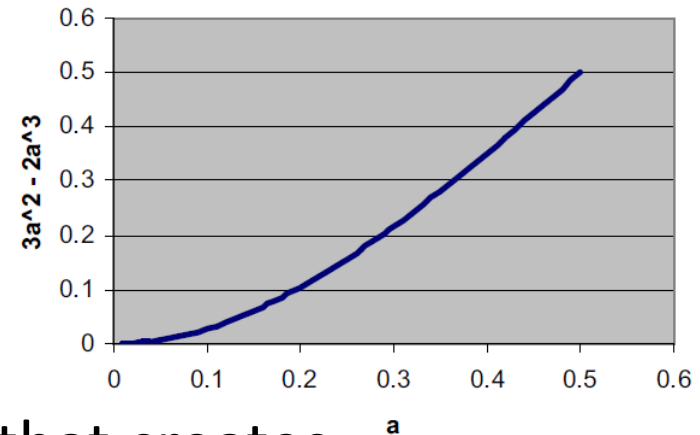
h_3 is the result of applying *Learn* to training data on which h_1 and h_2 disagree.

- We can then let them vote, the resulting error rate will be improved. We can repeat this until reaching the target error rate

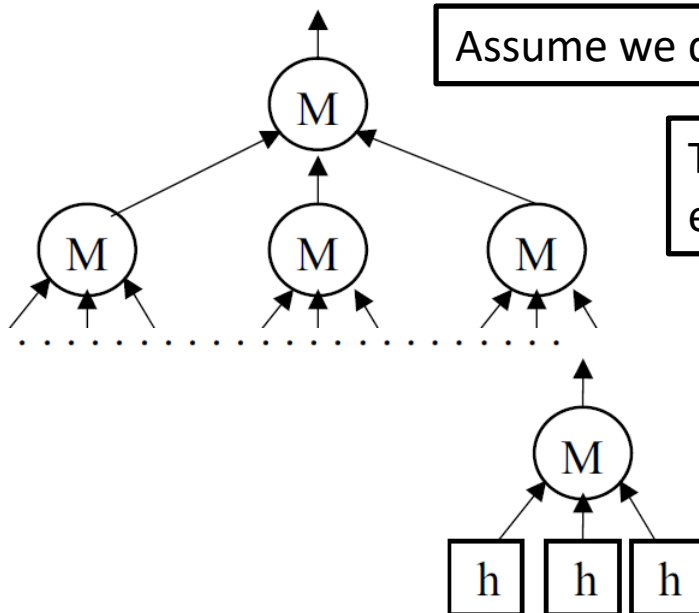
(Optional reading)

Consider $E = \langle \{h_1, h_2, h_3\}, \text{majorityVote} \rangle$.

- If h_1, h_2, h_3 have error rates less than a , it can be shown that the error rate of E is upper-bounded by $g(a)$: $3a^2 - 2a^3 < a$



This fact leads to a recursive algorithm that creates a hypothesis of arbitrary accuracy from weak hypotheses.



Assume we desire an error rate less than e .

These need only achieve an error rate less than $g^{-1}(e)$

As we move down the tree, the error we need to achieve increases according to g^{-1}

Eventually the error rate needed will be attainable by the weak learner

AdaBoost

(Optional reading)

- The boosting algorithm derived from the original proof is impractical
 - requires too 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.
- The base learner protocol needs 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

$D(i)$ indicate to base learner *Learn* the importance of the i -th training instance

AdaBoost algorithm:

Assume that everything is binary (binary example)

Input: *Learn* - Base learning algorithm.
 S - Set of N labeled training instances.

Output: $H = \langle \{h_1, \dots, h_L\}, \text{WeightedVote}(\alpha_1, \dots, \alpha_L) \rangle$
 L is another hyper parameter

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

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

h1 returns output1
h2 returns output2 $h_l = \text{Learn}(S, D_l)$ Apply base learning to S with D_l distribution

$\varepsilon_l = \text{error}(h_l, S, D_l)$ Measure **weighted** error of h_l

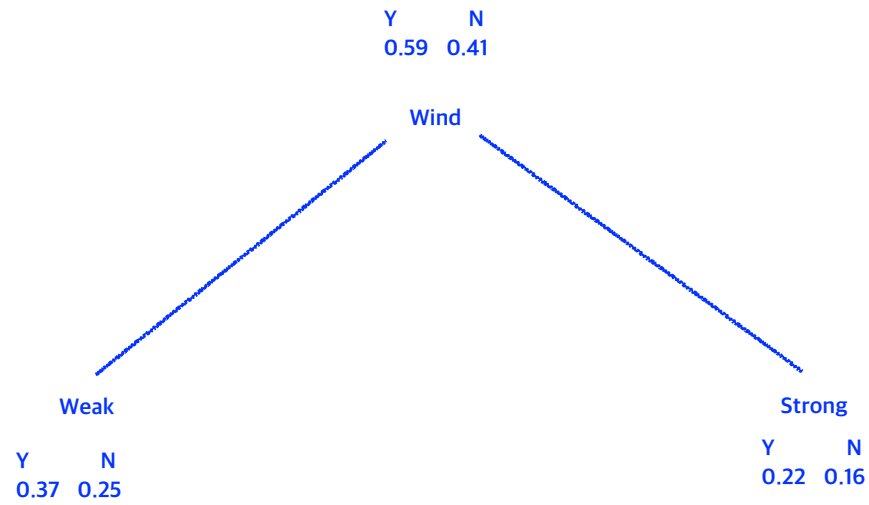
$\alpha_l = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_l}{\varepsilon_l} \right)$ **;; if $\varepsilon_l < 0.5$ implies $\alpha_l > 0$**
because, when epsilon < 0.5, (1-e)/e is positive > 1
If epsilon is not < 0.5, flip the decision (terminate or abort)

for all l , (여기도 loop 하나 더 있음다)
 $D_{l+1}(i) = D_l(i) \times \begin{cases} e^{\alpha_l}, & h_l(x_i) \neq y_i \\ e^{-\alpha_l}, & h_l(x_i) = y_i \end{cases}$ if this is true, it means that you made a mistake previously
D is a vector of weight to assign to training example
 $D_l(i) * e^{\alpha_l}$ 하면, $D_{l+1}(i)$ 는 increase 왜냐 $e^{\alpha_l} > 1$
for i from 1 to N
 $D_l(i) * e^{-\alpha_l}$ 하면, $D_{l+1}(i)$ 는 decrease 왜냐 $e^{-\alpha_l} < 1$

Normalize D_{l+1} **;; can show that h_l has 0.5 error on D_{l+1}**
if this is true, it means that our decision is correct

when to stop? decided by L

α_l is used to update and generate the distribution D_{l+1} based on D_l , it is also used as the weight for classifier h_l in final weighted voting.



$$P(T = Y \mid W = \text{weak}) = .37 / (.37 + .25)$$

Learning with Weighted Data

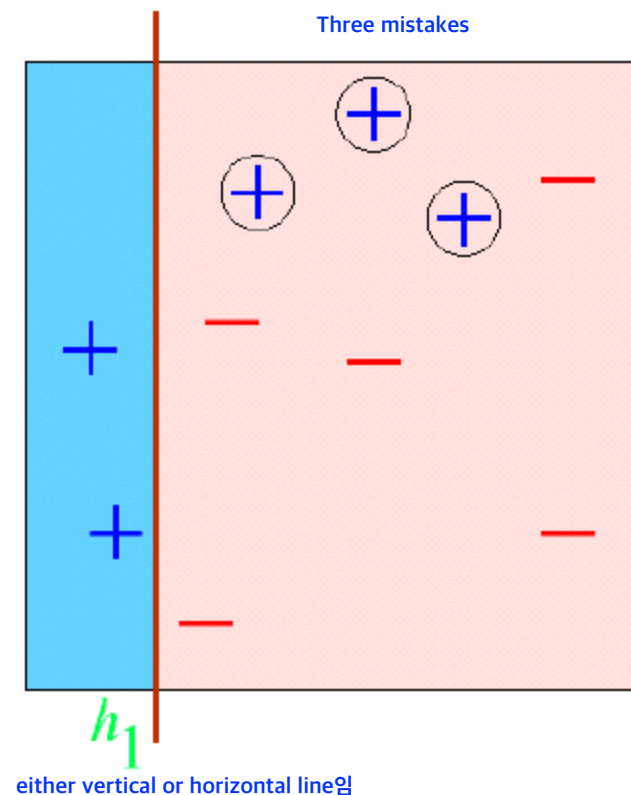
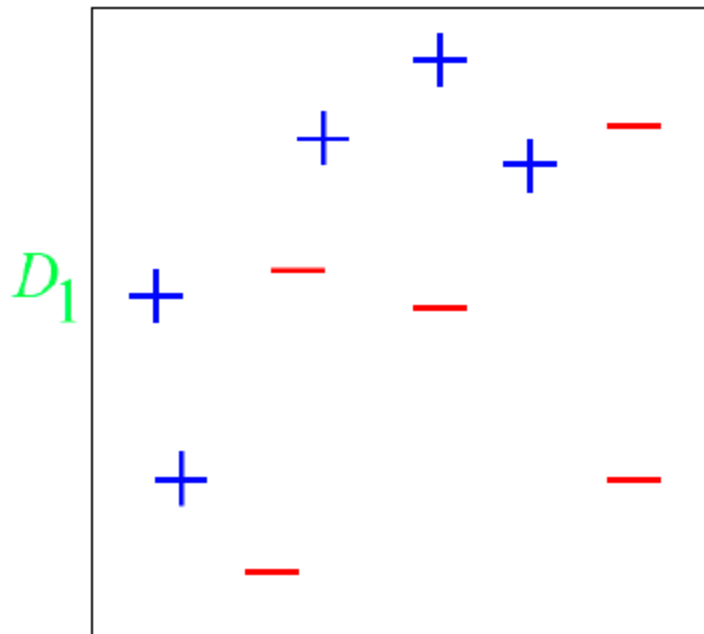
- Not all examples are equal
 - Some are more important than the others
- Using weights in learning can be achieved by:
 - if your learning algorithm works by optimizing a objective, such as logistic regression, perceptron, neural networks, you can weight the contribution of each example to the objective by its weight
 - If you also simply treat the i -th example as $D(i)$ examples
 - E.g., in MLE or in DT

$$\sum_i I(y_i = 1) \Rightarrow \sum_i D(i) I(y_i = 1)$$

AdaBoost(Example)

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

Original Training set : Equal
Weights to all training samples



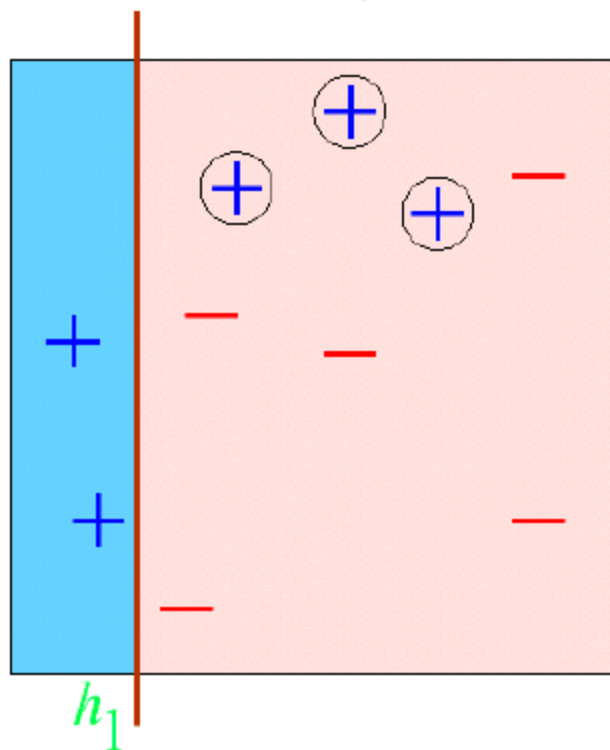
$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$

그니까 increase할그양 D_2 는

AdaBoost(Example)

ROUND 1



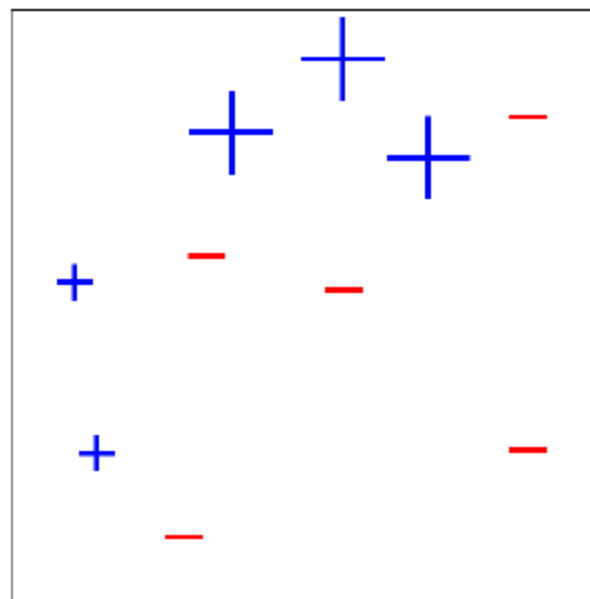
$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$



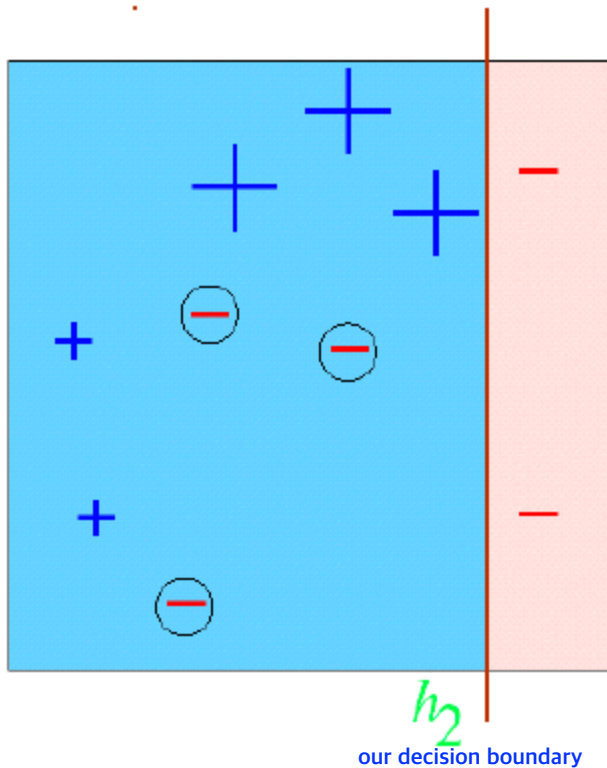
D_2

이렇게 mistake에 weighted가 돼서 Given



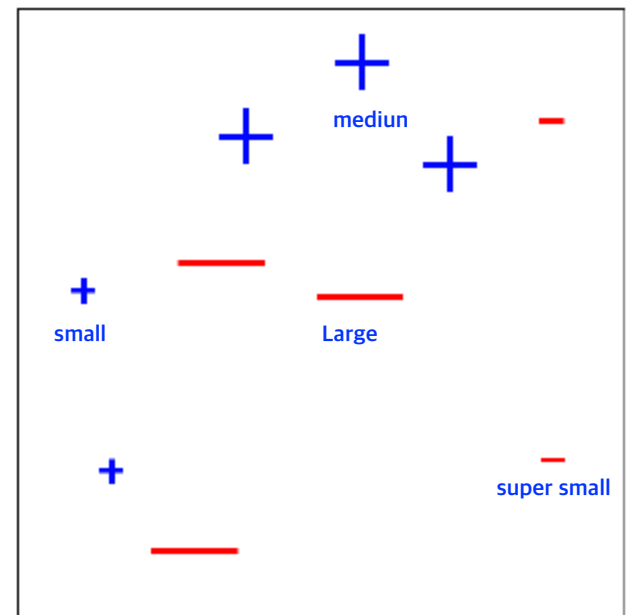
AdaBoost(Example)

ROUND 2



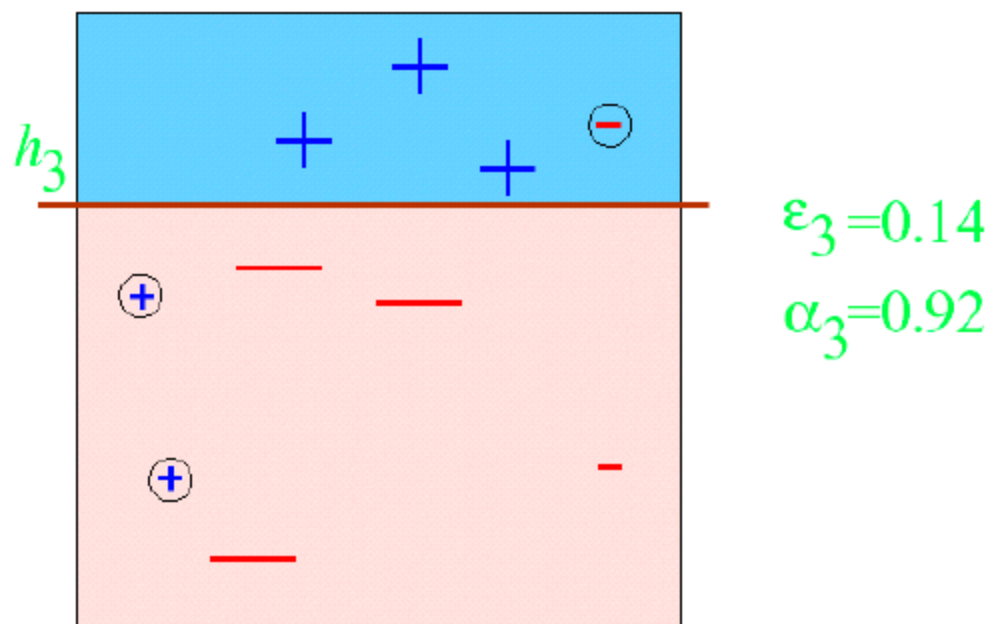
still has 3 mistakes, but the weight is small,
so epsilon2 is smaller

$$\epsilon_2 = 0.21$$
$$\alpha_2 = 0.65 \rightarrow D_3$$



AdaBoost(Example)

ROUND 3

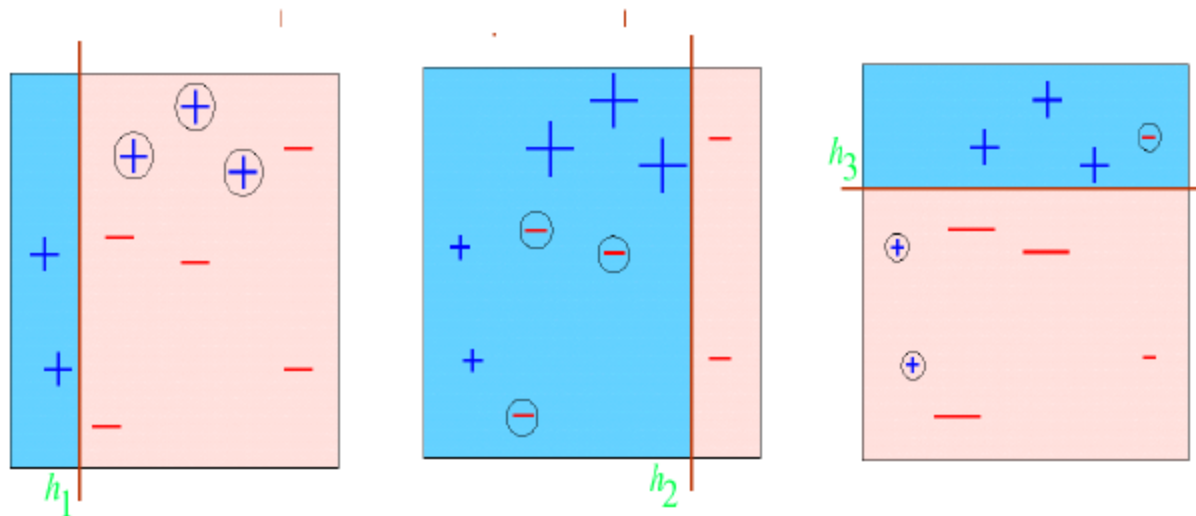


This is how go through different classifiers

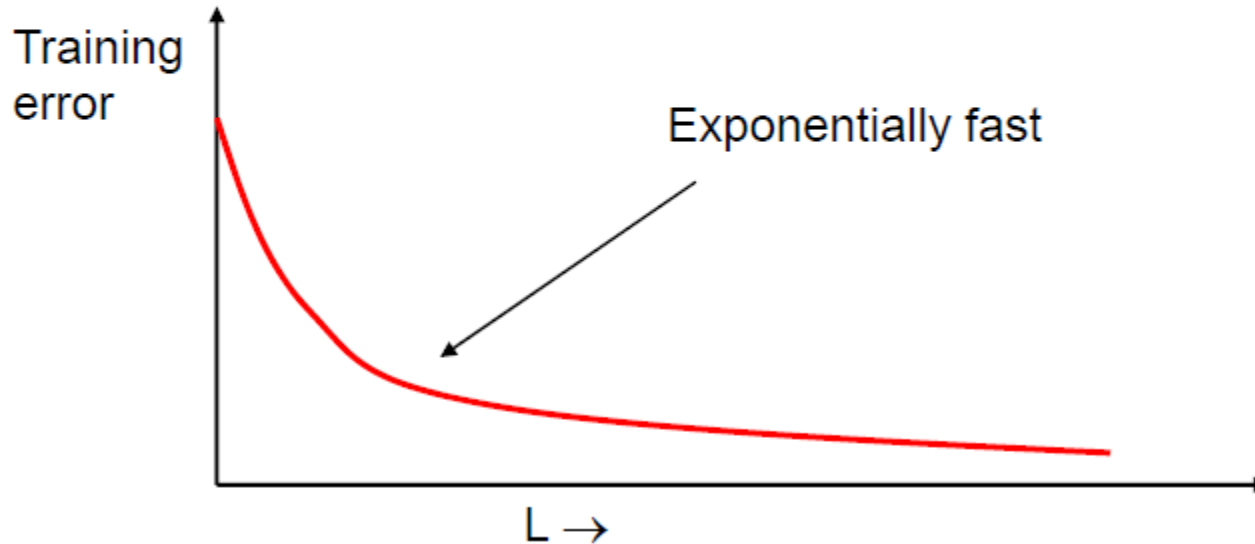
AdaBoost(Example)

H_{final}

$$= \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} \right)$$



Property of Adaboost

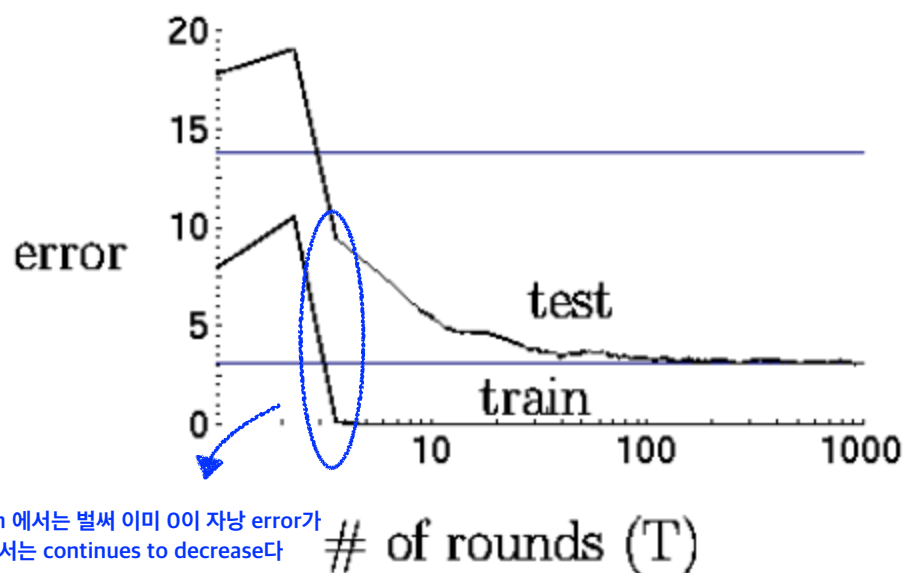


The training error goes to **zero** exponentially fast as we increase the ensemble size if your base learner is better than randomly guessing

What about **test error**?

Overfitting?

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

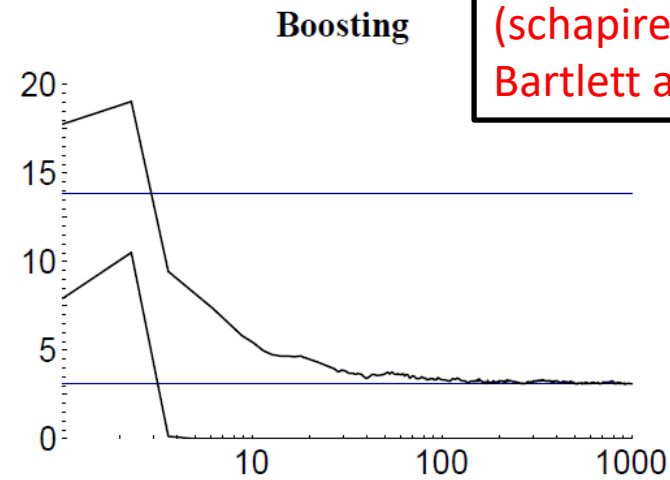
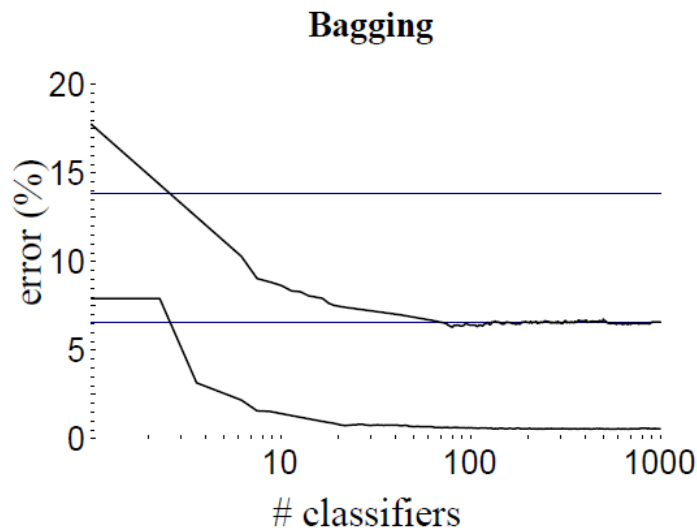


Schapire 1989.
Letter recognition

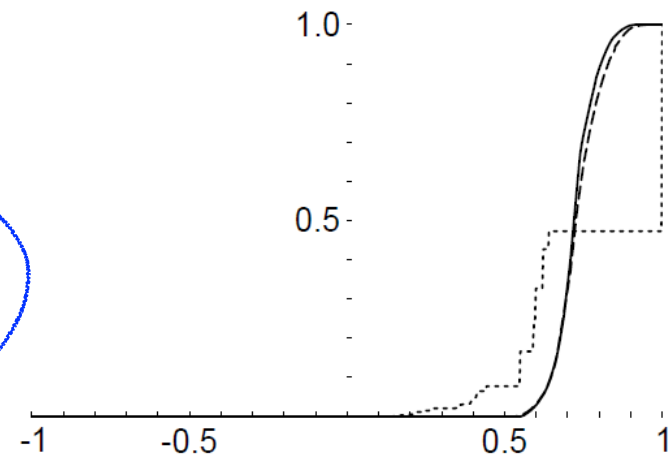
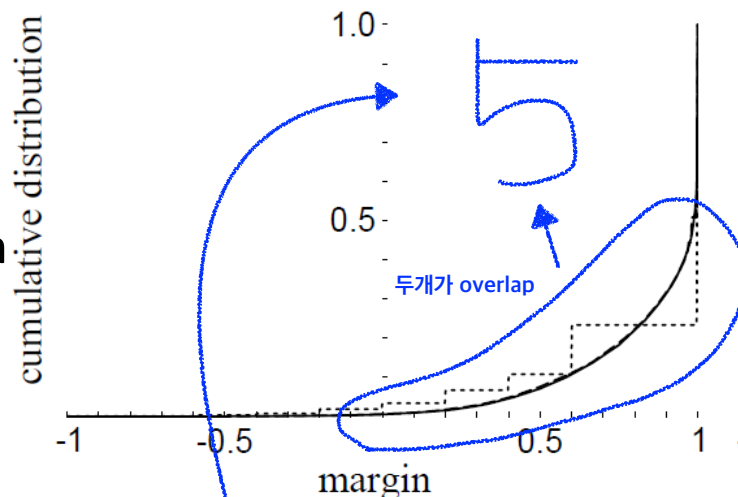
- Boosting is often robust to overfitting (not always)
- Test error continues to decrease even after training error goes to zero

(schapire, Freund, Bartlett and Lee 1998)

Training
/testing
error



Margin
Distribution



$y \cdot f(x)$ where y is true value between -1 to 1, and $f(x)$ is our output 그래서 Functional margin이라 애는

Figure 1: Error curves and margin distribution graphs for bagging and boosting C4.5 on the letter dataset. Learning curves are shown directly above corresponding margin distribution graphs. Each learning-curve figure shows the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of classifiers combined. Horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. The margin distribution graphs show the cumulative distribution of margins of the training instances after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

almost same, but gives more smooth한 그래프

Margin Based Error bound

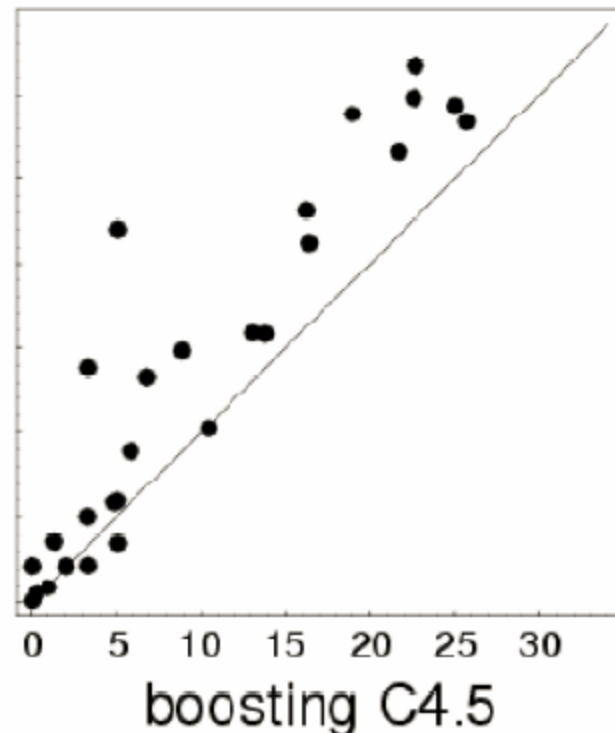
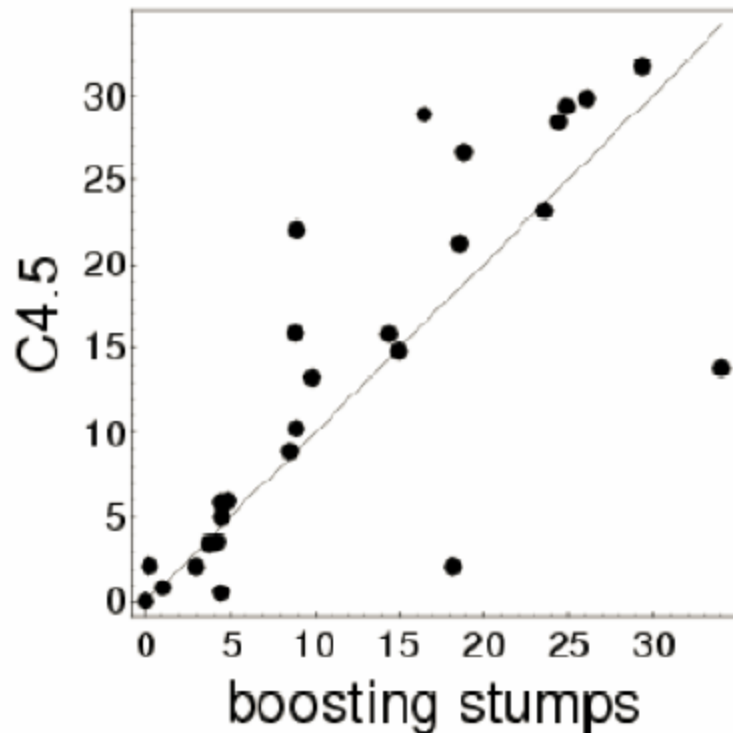
(Schapire, Freund, Bartlett and Lee 1998)

$$\mathbf{P}_{\mathcal{D}} [yf(x) \leq 0] \leq \mathbf{P}_S [yf(x) \leq \theta] + O \left(\frac{1}{\sqrt{m}} \left(\frac{d \log^2(m/d)}{\theta^2} + \log(1/\delta) \right)^{1/2} \right).$$

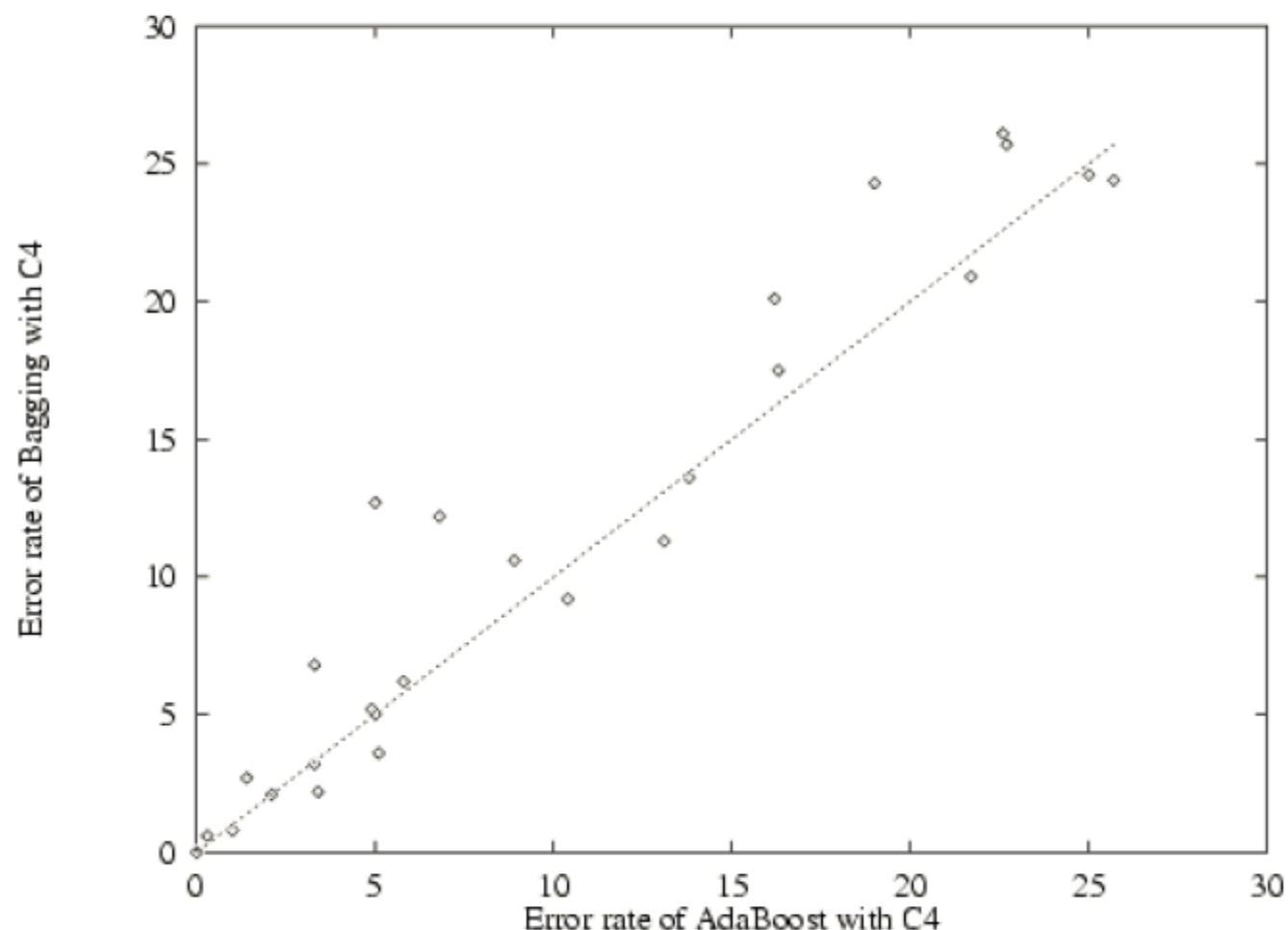
- Boosting increases the margin very aggressively since it concentrates on the hardest examples.
- If margin is large, more weak learners agree and hence more rounds does not necessarily imply that final classifier is getting more complex.
- Bound is independent of number of rounds T !
- Boosting can still overfit if margin is too small , weak learners are too complex or perform arbitrarily close to random guessing

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



AdaBoost as Functional Gradient Descent

- We will now derive AdaBoost in a way that can be adapted in various ways
- This recipe will let you derive “boosting-style” algorithms for particular learning settings of interest
 - E.g., general mis-prediction cost, semi-supervised learning, ranking
- these boosting-style algorithms will not generally be boosting algorithms in the theoretical sense but they often work quite well

Iterative Learning of Additive Models via Gradient Descent

accumulate하게 구해서 더하고,
그담 그걸로 다시 구해서 또 더하고 그래서
additive model이라고 불리는거

- Consider the final hypothesis: it takes the sign of an ***additive expansion of a set of base classifiers***

$$H(\mathbf{x}) = \text{sign} [f(\mathbf{x})] = \text{sign} \left[\sum_{l=1}^L \alpha_l h_l(\mathbf{x}) \right]$$

- AdaBoost iteratively finds at each iteration an $h(\cdot)$ to add to $f(\cdot)$

$$f_l(\mathbf{x}) = f_{l-1}(\mathbf{x}) + \alpha_l h_l(\mathbf{x})$$

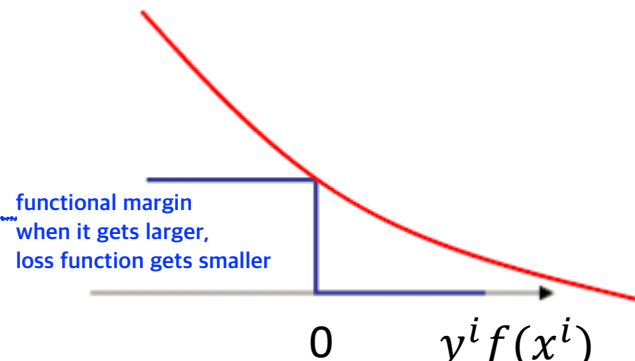
- The goal is to minimize a loss function on the training example:

$$\sum_{i=1}^N L \left(y^i, \sum_{l=1}^L \alpha_l h_l(\mathbf{x}^i) \right)$$

- We would like to minimize the error:

$$L(y^i, f(x^i)) = [y^i \neq \text{sgn}(f(x^i))] \quad \text{Note } y \in \{-1, 1\}$$

- Instead, Adaboost can be viewed as minimizing an exponential loss function, which is a smooth upper bound on 0/1 error:

$$L(y^i, f(x^i)) = e^{-y^i f(x^i)}$$


functional margin
when it gets larger,
loss function gets smaller

$$\arg \min_f \sum_{i=1}^N L(y^i, f(x^i))$$

$$= \arg \min_{\alpha, h_l} \sum_{i=1}^N e^{-y^i \cdot [f_{l-1}(x^i) + \alpha \cdot h_l(x^i)]}$$

at iteration l , look
for h_l and α

$$= \arg \min_{\alpha, h_l} \sum_{i=1}^N e^{-y^i \cdot f_{l-1}(x^i)} \cdot e^{-y^i \cdot \alpha \cdot h_l(x^i)}$$

Fix α and optimize h_l

$$\arg \min_{h_l} \sum_{i=1}^N \underline{e^{-y^i \cdot f_{l-1}(x^i)} \cdot e^{-y^i \cdot \alpha \cdot h_l(x^i)}}$$

$$= \arg \min_{h_l} \sum_{i=1}^N \underline{w_l^i} \cdot e^{-y^i \cdot \alpha \cdot h_l(x^i)}$$

$$= \arg \min_{h_l} \sum_{y^i = h_l(x^i)}^N w_l^i \cdot e^{-\alpha} + \sum_{y^i \neq h_l(x^i)}^N w_l^i \cdot e^{\alpha}$$

correct incorrect

$$= \arg \min_{h_l} \sum_{i=1}^N w_l^i \cdot e^{-\alpha} - \sum_{y^i \neq h_l(x^i)}^N w_l^i \cdot e^{-\alpha} + \sum_{y^i \neq h_l(x^i)}^N w_l^i \cdot e^{\alpha}$$

$$= \arg \min_{h_l} e^{-\alpha} \sum_{i=1}^N w_l^i + (e^{\alpha} - e^{-\alpha}) \cdot \sum_{y^i \neq h_l(x^i)}^N w_l^i$$

$$= \arg \min_{h_l} e^{-\alpha} + (e^{\alpha} - e^{-\alpha}) \cdot \frac{\sum_{y^i \neq h_l(x^i)}^N w_l^i}{\sum_{i=1}^N w_l^i}$$

normalize the whole by sum of $w_{l,i}$

instead of minimizing the whole things,
just minimize this term

$$= \arg \min_{h_l} e^{-\alpha} + (e^{\alpha} - e^{-\alpha}) \cdot \sum_{i=1}^N \frac{w_l^i}{\sum_{i=1}^N w_l^i} \cdot [y^i \neq h_l(x^i)]$$

애가 우리가 쓰는 D가 됩니당

Fix $h_l(\cdot)$ and find α

$$\begin{aligned} & \arg \min_{\alpha} e^{-\alpha} + (e^{\alpha} - e^{-\alpha}) \cdot \sum_{i=1}^N \frac{w_l^i}{\sum_{i=1}^N w_l^i} \cdot I[y^i \neq h_l(x^i)] \\ = & \arg \min_{\alpha} \underbrace{e^{-\alpha} + (e^{\alpha} - e^{-\alpha}) \cdot \epsilon_l}_{J(\alpha)} \end{aligned}$$

$$\begin{aligned} \frac{\partial J(\alpha)}{\partial \alpha} &= -e^{-\alpha} + \epsilon_l \cdot e^{\alpha} + \epsilon_l \cdot e^{-\alpha} \\ &= e^{-\alpha} \cdot (\epsilon_l - 1) + e^{\alpha} \cdot \epsilon_l = 0 \end{aligned}$$

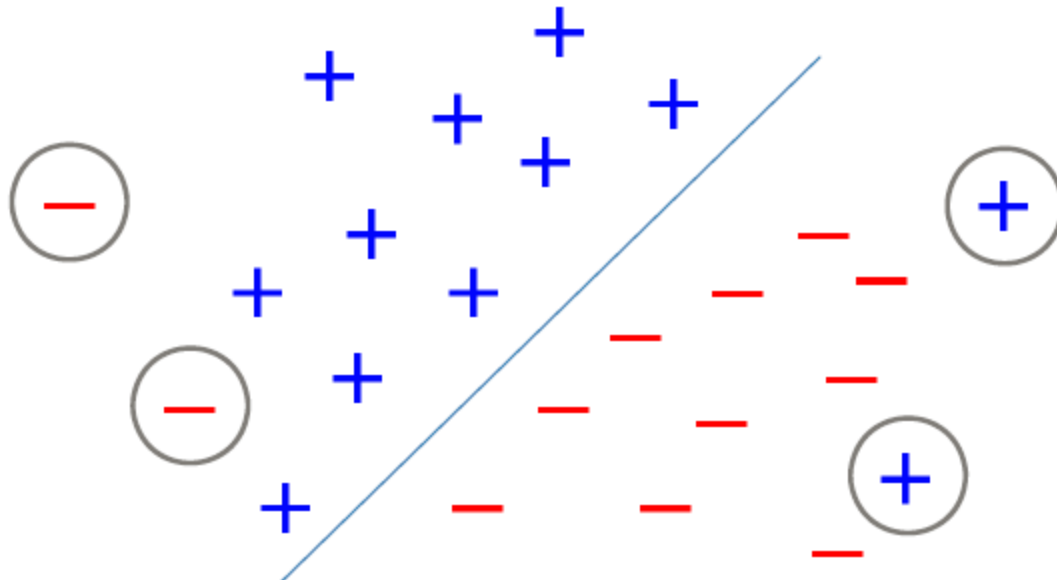
$$\Rightarrow e^{\alpha} \cdot \epsilon_l = e^{-\alpha} \cdot (1 - \epsilon_l)$$

$$\Rightarrow e^{2\alpha} = \frac{1 - \epsilon_l}{\epsilon_l} \Rightarrow \alpha = \frac{1}{2} \ln \frac{1 - \epsilon_l}{\epsilon_l}$$

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



Summary

- Bagging
 - Resample data points
 - Weight of each classifier is the same
 - Only variance reduction
 - Robust to noise and outliers
- Random forest
 - (Instance) Bagging + feature bagging
 - Increased diversity among the learned classifiers
 - Robust to noise and outliers
 - Efficient for large dataset
 - Can be used to estimate variable importance
- Boosting
 - Reweight data points (modify data distribution)
 - Weight of classifier vary depending on accuracy
 - Reduces both bias and variance (in early iterations, primarily bias reduction, later primarily variance reduction)
 - Can hurt performance with noise and outliers
 - The gradient boosting view leads to new algorithms