Ensemble Learning

Jihoon Yang

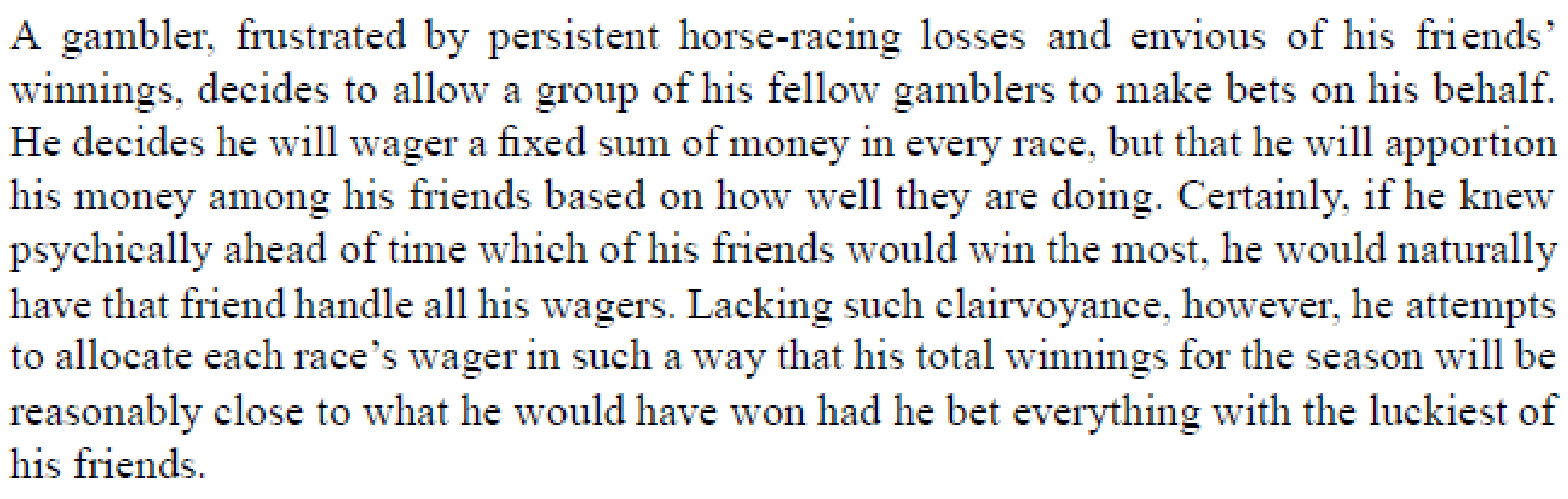
Machine Learning Research Laboratory
Department of Computer Science & Engineering
Sogang University
Email: yangjh@sogang.ac.kr
URL: mllab.sogang.ac.kr

Accuracy Boosting Using Ensemble Classifiers

- Ensemble methods
- Bagging
- Boosting
- Error-correcting coding
- Why does ensemble learning work?

What is Ensemble Learning?

- **Ensemble learning refers to a collection of methods that learn a target function by training a number of individual learners and combining their predictions**



A gambler, frustrated by persistent horse-racing losses and envious of his friends' winnings, decides to allow a group of his fellow gamblers to make bets on his behalf. He decides he will wager a fixed sum of money in every race, but that he will apportion his money among his friends based on how well they are doing. Certainly, if he knew psychically ahead of time which of his friends would win the most, he would naturally have that friend handle all his wagers. Lacking such clairvoyance, however, he attempts to allocate each race's wager in such a way that his total winnings for the season will be reasonably close to what he would have won had he bet everything with the luckiest of his friends.

[Freund & Schapire, 1995]

What is Ensemble Learning?

- An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples

- Aka *committees*

- Dietterich, Ensemble methods in machine learning (2000).

Ensemble Learning

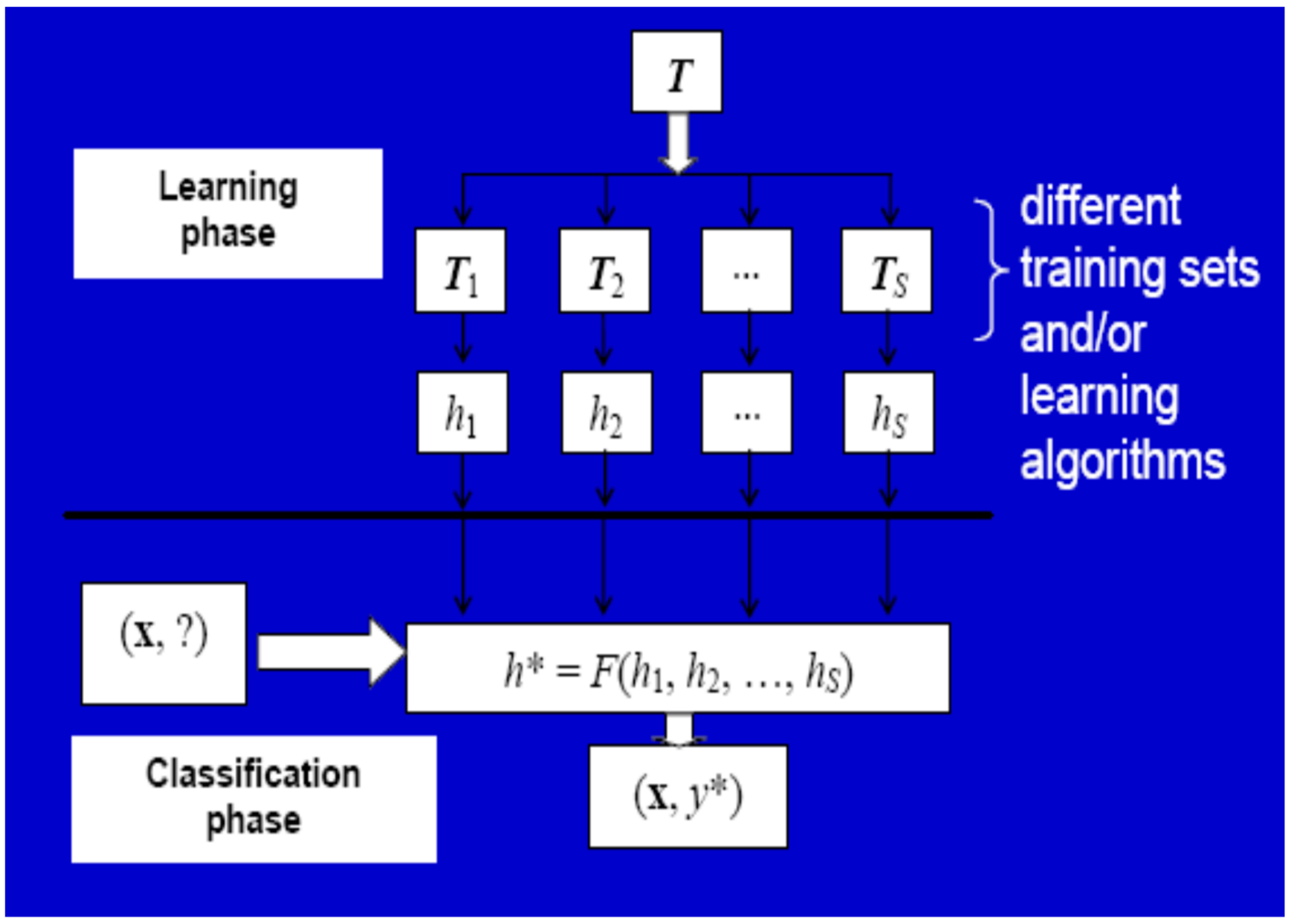
- *Intuition*: Combining predictions of an ensemble is more accurate than a single classifier

- *Justification*:
  - ✓ Easy to find quite correct "rules of thumb" however hard to find single highly accurate prediction rule
  - ✓ If the training examples are few and the hypothesis space is large then there are several equally accurate classifiers
  - ✓ Hypothesis space does not contain the true function, but it has several good approximations
  - ✓ Exhaustive global search in the hypothesis space is expensive so we can combine the predictions of several locally accurate classifiers

Ensemble Learning

- **The classifiers should be accurate and diverse**
- **An accurate classifier is one that has an error rate of better than random guessing (known as *weak learners*)**
- **Two classifiers are diverse if they make different errors on new data points**

Ensemble Learning



How to make an effective ensemble?

- **Two basic questions in designing ensembles:**

  - **How to generate the base classifiers?**

    $h_1, h_2, \cdots$

  - **How to combine them?**

    $F(h_1(x), h_2(x), \cdots)$

How to Combine Classifiers

- **Usually take a weighted vote:**

$$ensemble(x) = sign\left(\sum_i w_i h_i(x)\right)$$

  - **–*wi* is the *weight* of hypothesis *hi***
  - **–*wi* > *wj* means *hi* is more reliable than *hj***
  - **–Typically *wi* > *0* (though could have *wi* < *0* meaning *hi* is more often wrong than right)**

- **Bayesian averaging is an example**

$$p(f(x) = y \mid D, x) = \sum_{h \in H} \underbrace{h(x)}_{\text{값}}\, \underbrace{p(h \mid D)}_{\text{개연값}} = \sum_{h \in H} h(x)\, p(D \mid h)\, p(h)$$

How to Generate Base Classifiers

- A variety of approaches
  - Bagging (Bootstrap aggregation)
  - Boosting (Specifically, Adaboost – Adaptive Boosting algorithm))
  - …

BAGGing = Bootstrap AGGregation
(Breiman, Bagging Predictors, Machine Learning, 1996)

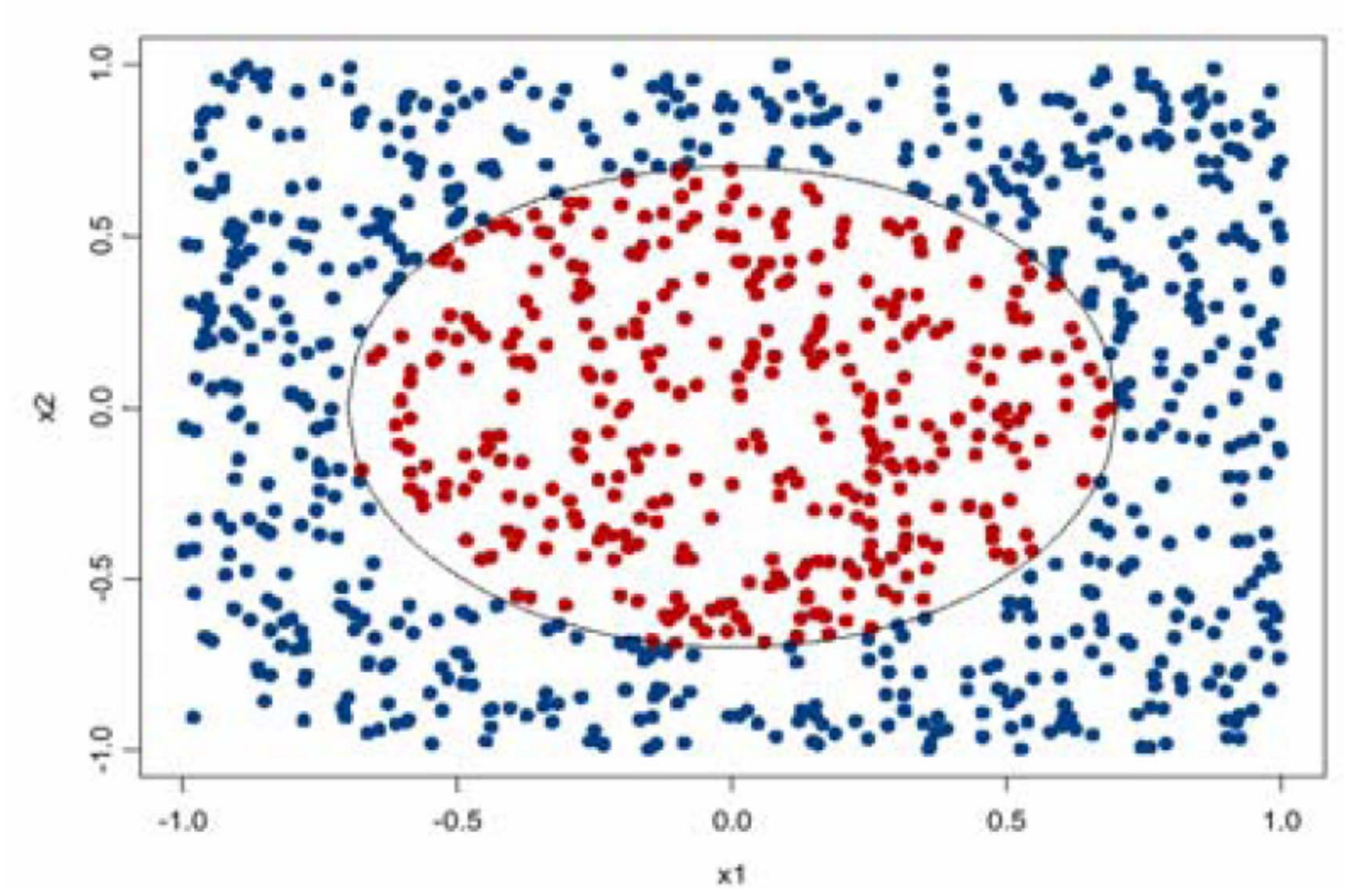
- Generate a *random sample* from *training set* by selecting elements *with replacement* (known as *bootstrap sample*)
- Repeat this sampling procedure, getting a sequence of *k independent training sets*
- A corresponding sequence of classifiers $C_1, C_2, \cdots, C_k$ is constructed for each of these training sets, by using the same classification algorithm
- To classify an unknown sample *X*, let each classifier predict
- The Bagged Classifier *C** then combines the predictions of the individual classifiers to generate the final outcome. (Sometimes combination is simple voting.)

Bagging

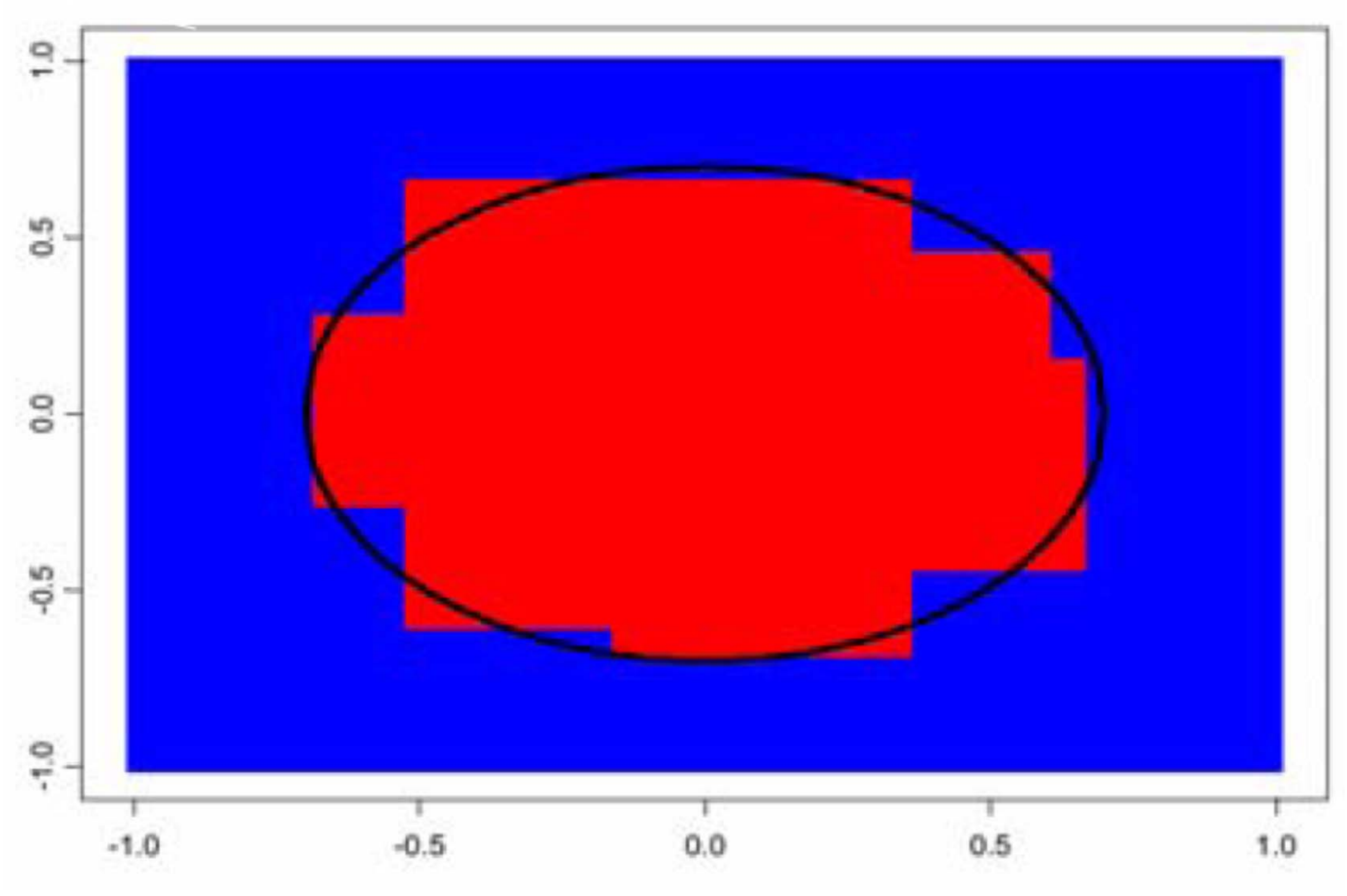
```
for $i = 1, 2, \cdots, k$:
    $T_i \leftarrow$ randomly select $M$ training instances
                    with replacement
    $h_i \leftarrow learn(T_i)$
```

- Combine the T_i using uniform voting ($w_i = 1/k$ for all i)

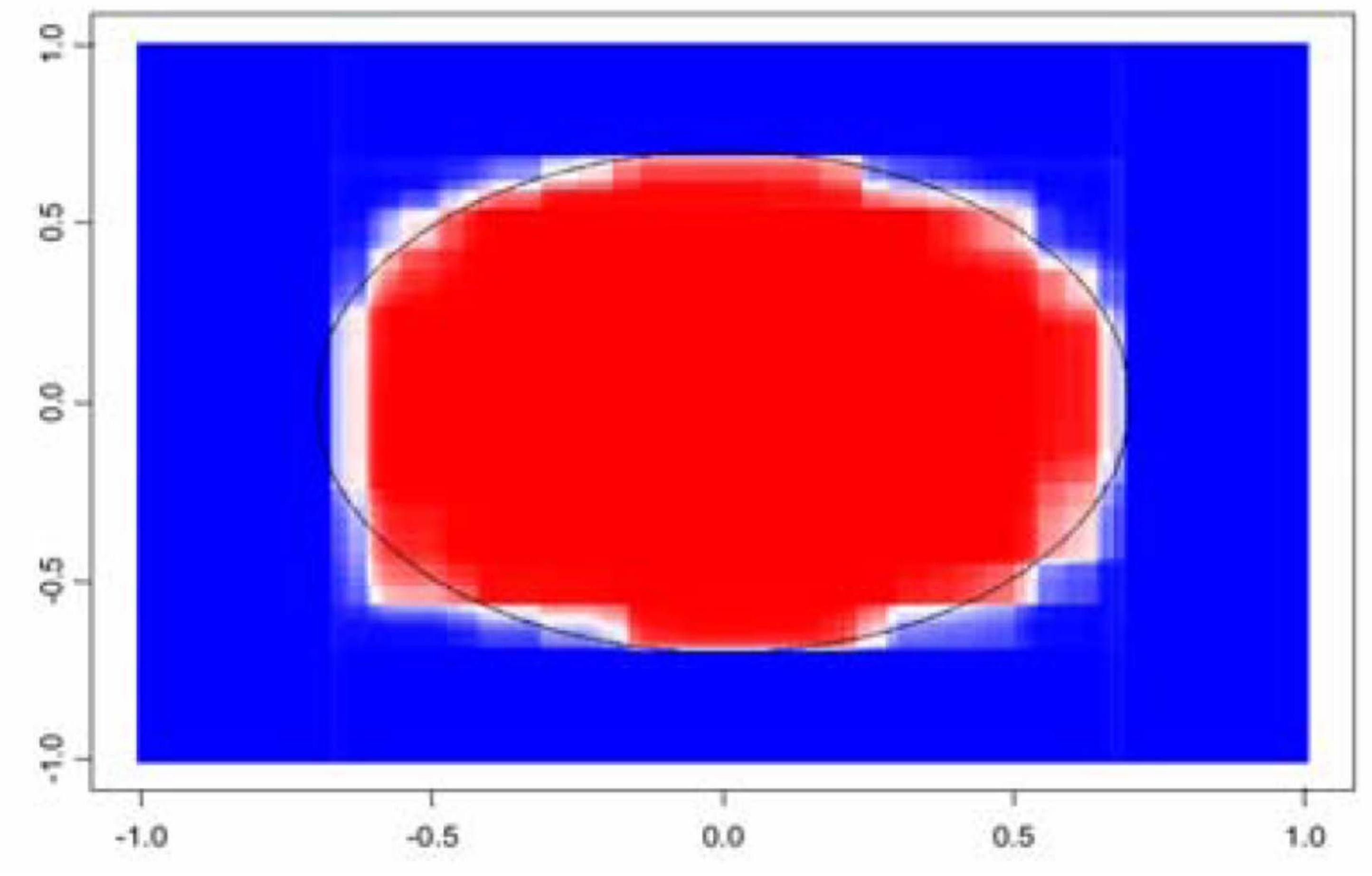
Bagging Example



CART Decision Boundary



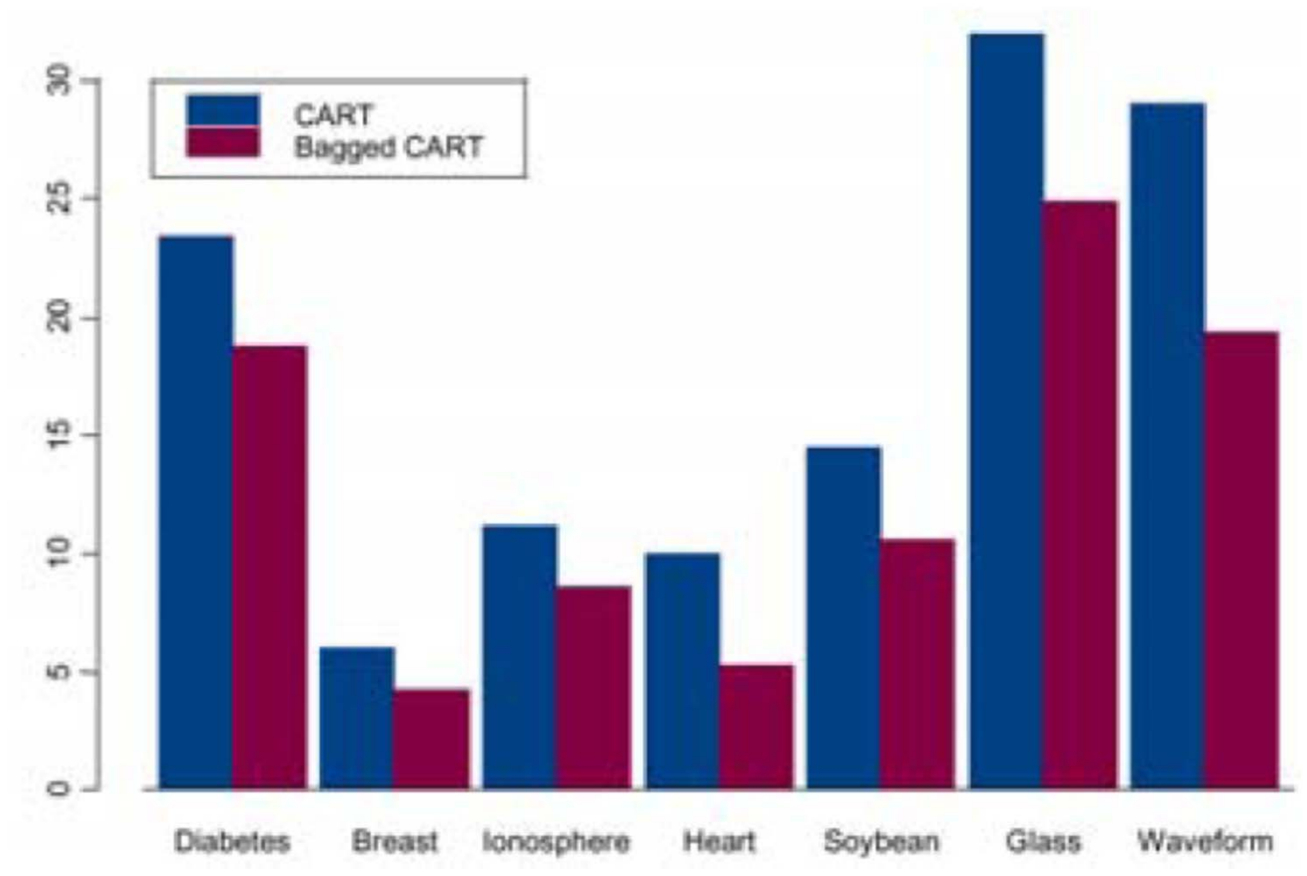
100 Bagged Trees



Shades of blue/red indicate strength of vote for particular classification

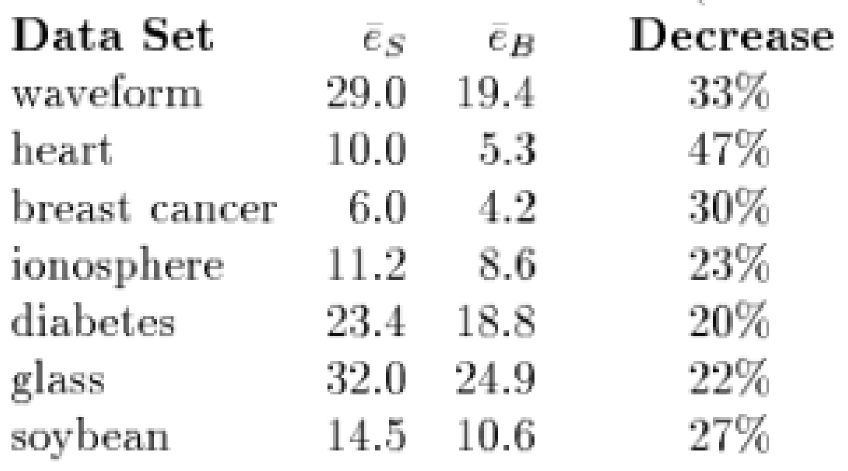
Bagging CART

Misclassification Rates



Bagging CART

Misclassification rates



Data Set	$\bar{e}_S$	$\bar{e}_B$	Decrease
waveform	29.0	19.4	33%
heart	10.0	5.3	47%
breast cancer	6.0	4.2	30%
ionosphere	11.2	8.6	23%
diabetes	23.4	18.8	20%
glass	32.0	24.9	22%
soybean	14.5	10.6	27%

-50 bootstrap samples

Breiman, L. (1996), Bagging Predictors, Machine learning

Bagging

들어가는 알고리즘의 안정성이 중요

- A critical factor in whether bagging will improve accuracy is the stability of the learning algorithm
- A learning algorithm is unstable if a small change in the training data can result in large change in the output classifier
- Unstable: neural networks, decision trees;
  Stable: linear regression, *k*-nearest neighbor
- Bagging works well for unstable procedures
- It is a relatively easy way to improve an existing method

Boosting

- **Boosting, like bagging, is an ensemble method**
- **The prediction generated by the classifier is a combination of the prediction of several predictors**
- **What is different?**
  - **It is iterative**
  - **Boosting: *Successive classifier depends upon its predecessors***
  - **Previous methods (e.g. bagging): *Individual classifiers were independent***
  - **Training examples may have *unequal weights***
  - **Look at errors from previous classifier step to decide how to focus on next iteration over data**
  - **Set weights to *focus more on "hard" examples* (the ones on which we committed mistakes in the previous iterations)**

Boosting

- **Basic idea:**
  - **Assign a weight to every training set instance**
  - **Initially, all instances have the same weight**
  - **As boosting proceeds, it adjusts weights based on how well we have predicted data points so far**
    - **Data points correctly predicted → low weight**
    - **Data points mispredicted → high weight**

- **Results: As learning proceeds, the learner is forced to focus on portions of data space not previously well predicted**

Boosting Algorithm

- *W(x)* is the distribution of weights over the *N* training points

$$\sum W(x_i) = 1$$

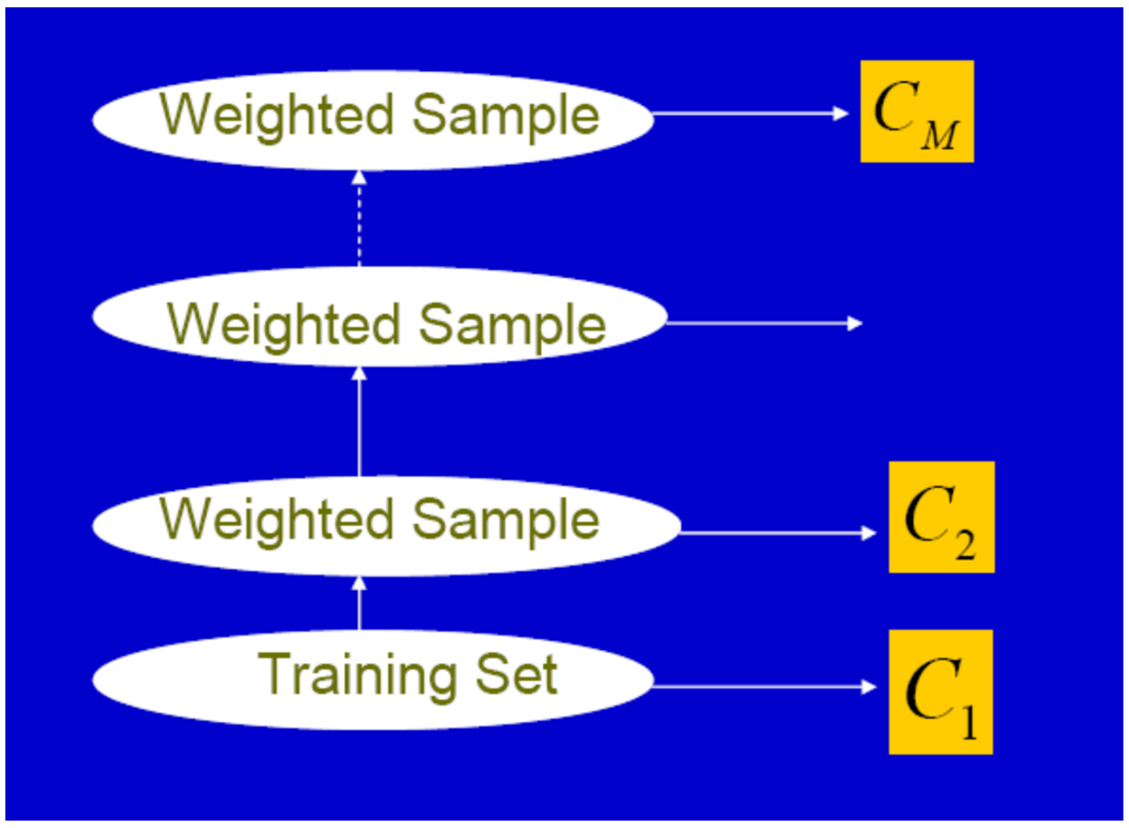
- Initially assign uniform weights $W_0(x) = 1/N$ for all *x*, step $k=0$

- At iteration *k*:
  - Find best weak classifier $C_k(x)$ using weights $W_k(x)$ with
    - Error rate ε_k
    - α_k is the weight of the classifier C_k
    - For each x_i, update weights based on ε_k to get $W_{k+1}(x_i)$

$$C_{FINAL}(x) = sign\left(\sum_i \alpha_i C_i(x)\right)$$

Boosting Algorithm

$$C(x) = \sum_{j=1}^{M} \alpha_j C_j(x)$$



AdaBoost Algorithm

- ***W(x)*** **is the distribution of weights over the *N* training points**

$$\sum W(x_i) = 1$$

- **Initially assign uniform weights *W₀(x) = 1/N* for all *x***
- **At iteration *k*:**
  - **Find best weak classifier *Cₖ(x)* using weights *Wₖ(x)***
  - **Compute the error rate *εₖ* as**

$$\varepsilon_k = \left[\sum W_k(x_i) \cdot I(y_i \neq C_k(x_i))\right] / \left[\sum W_k(x_i)\right]$$

  - **Weight the classifier *Cₖ* by *αₖ***

$$\alpha_k = \frac{1}{2} \ln\left((1-\varepsilon_k)/\varepsilon_k\right)$$

  - **For each *xᵢ*,**

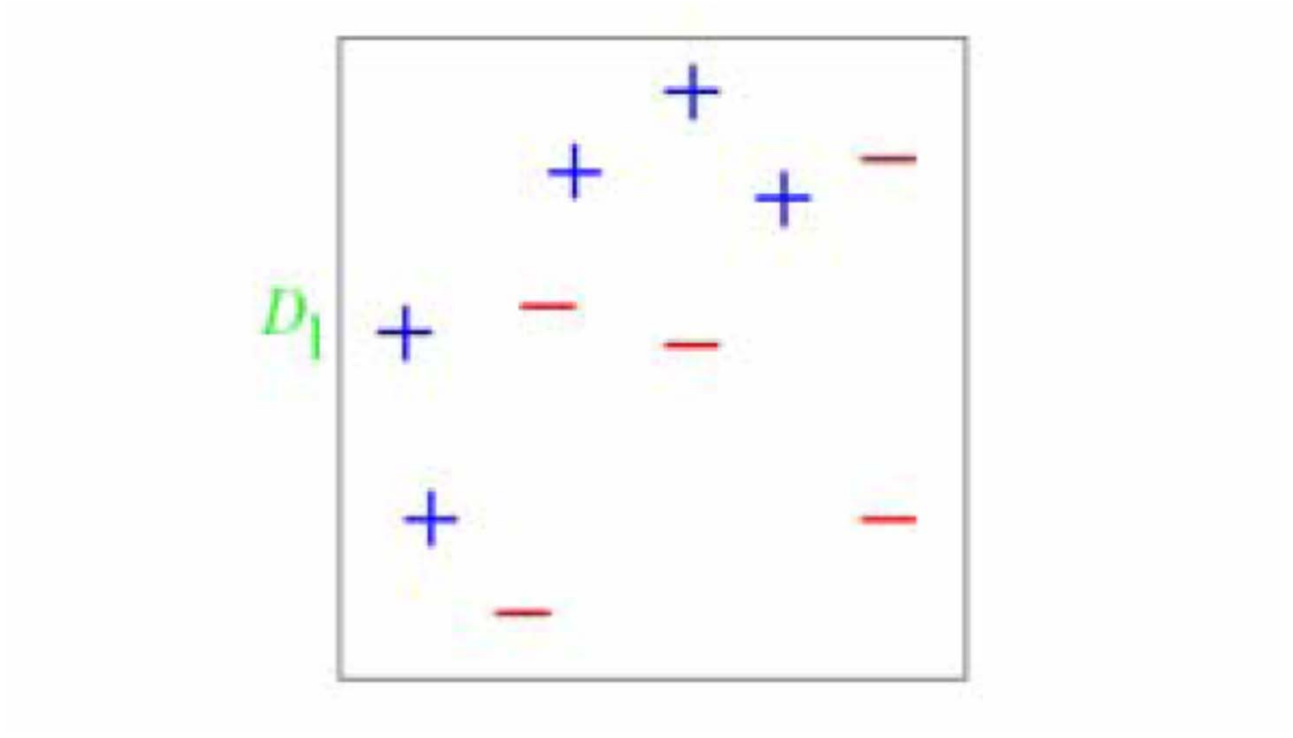
$$W_{k+1}(x_i) = W_k(x_i) \cdot \exp\left[\alpha_k \cdot I(y_i \neq C_k(x_i))\right]$$

  then normalize weights

- **Final classifier:**

$$C_{FINAL}(x) = sign\left(\sum_i \alpha_i C_i(x)\right)$$

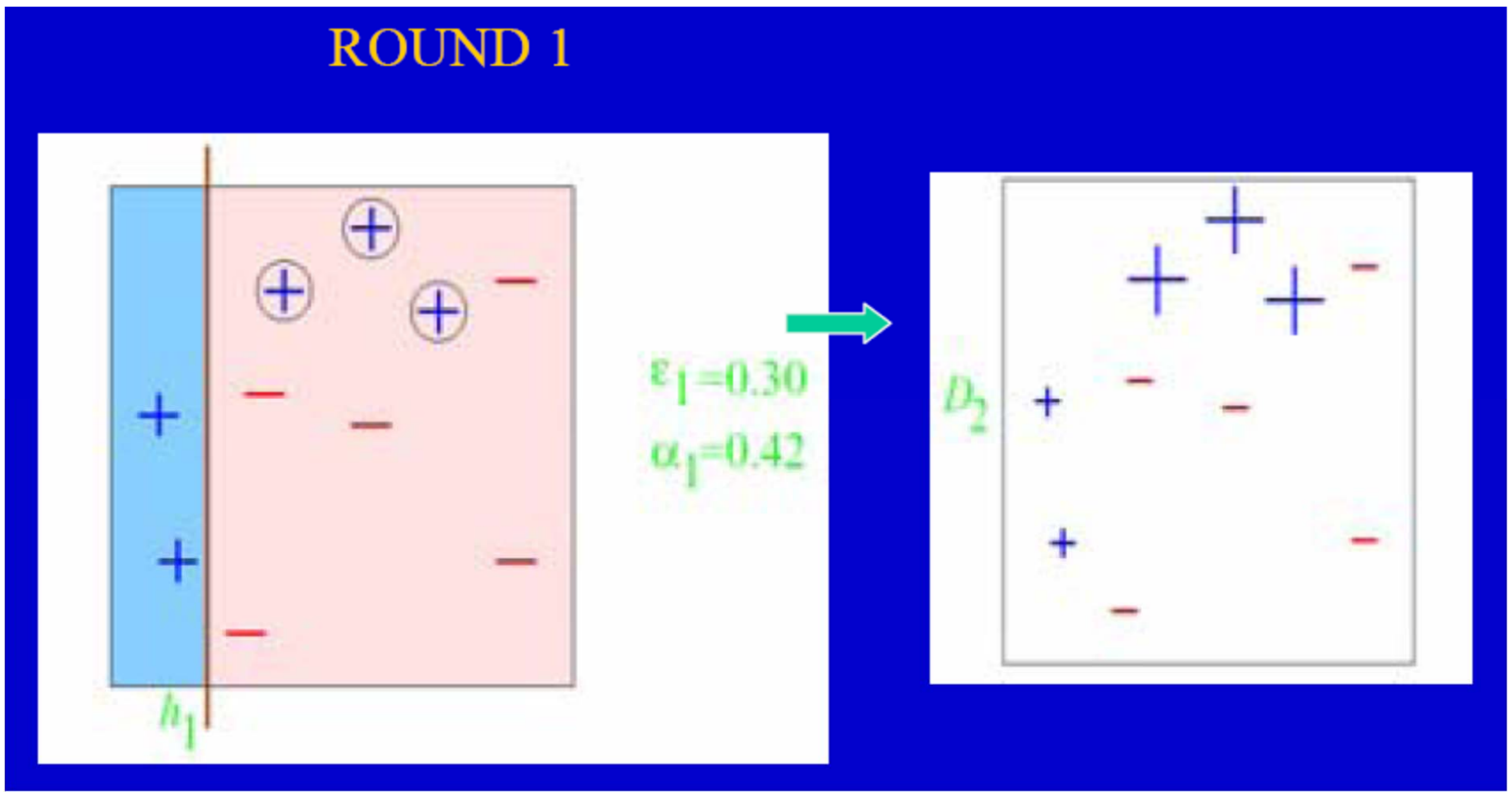
AdaBoost Example



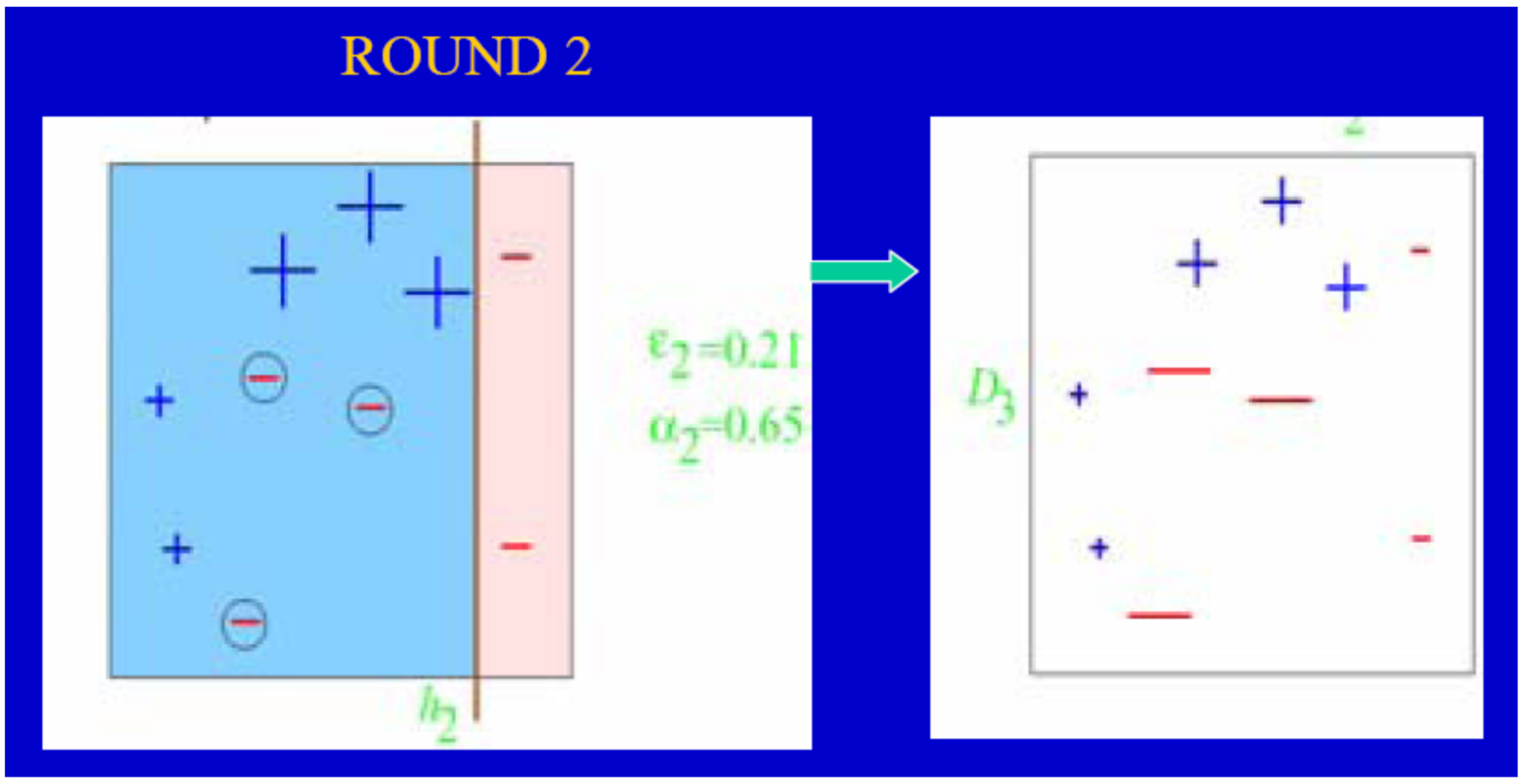
Original Training Set: equal weights to all training samples

SOGANG UNIVERSITY

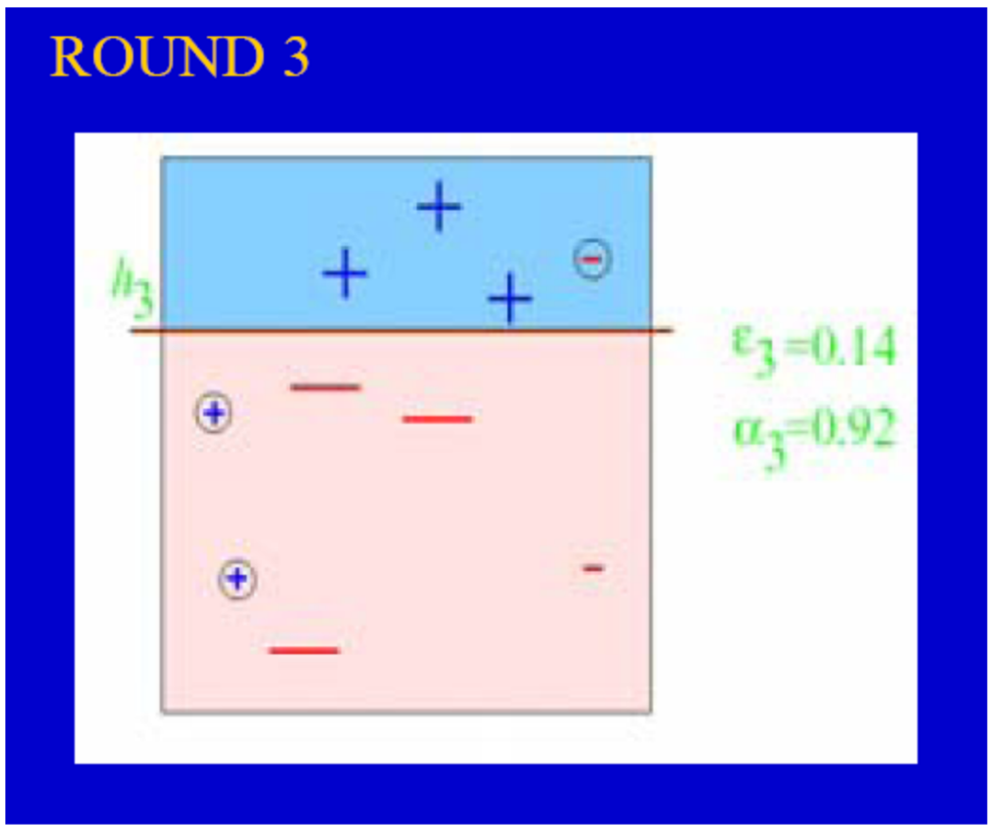
AdaBoost Example



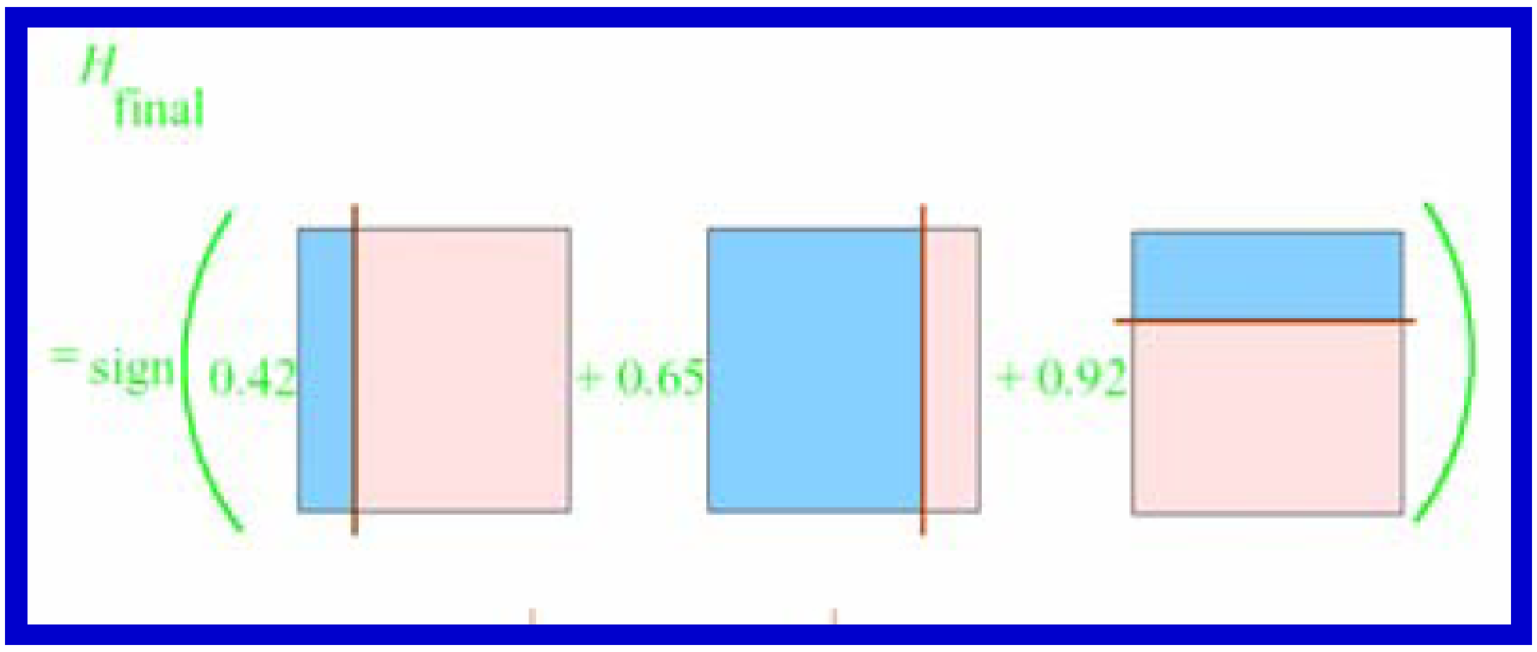
AdaBoost Example



AdaBoost Example



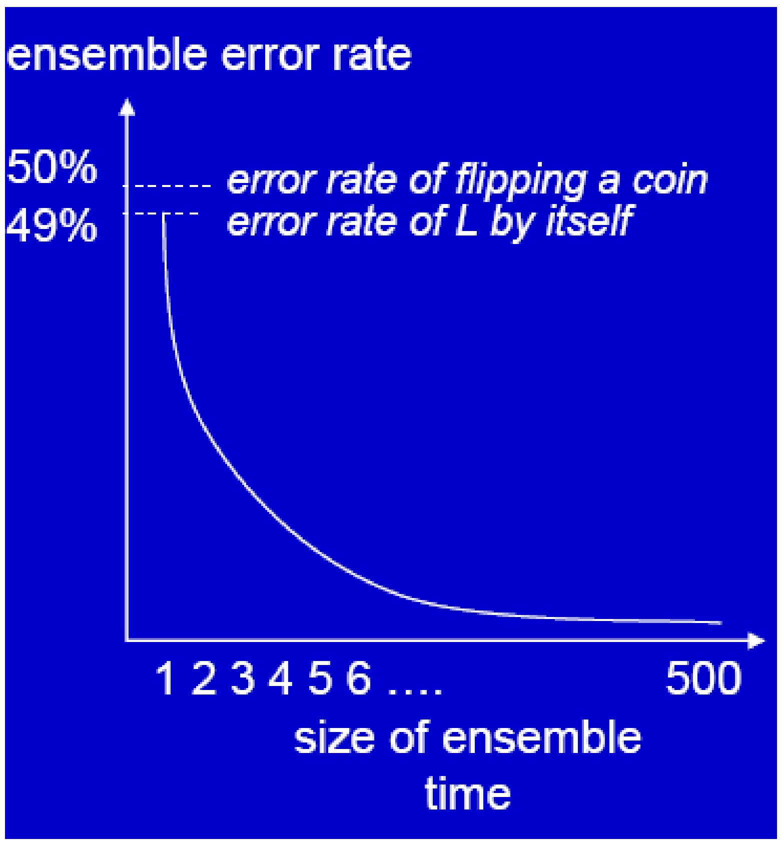
AdaBoost Example


H_{final}
$= \text{sign}\left(0.42 \quad + 0.65 \quad + 0.92 \right)$

Boosting

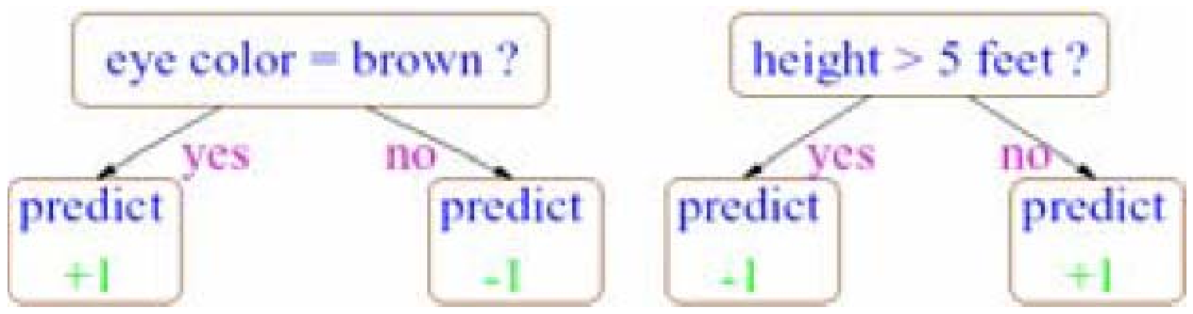
- Suppose *L* is a *weak* learner – one that can learn a hypothesis that is better than rolling a dice – but perhaps only a tiny bit better

- Theorem: Boosting *L* yields an ensemble with arbitrarily low error on the training data!

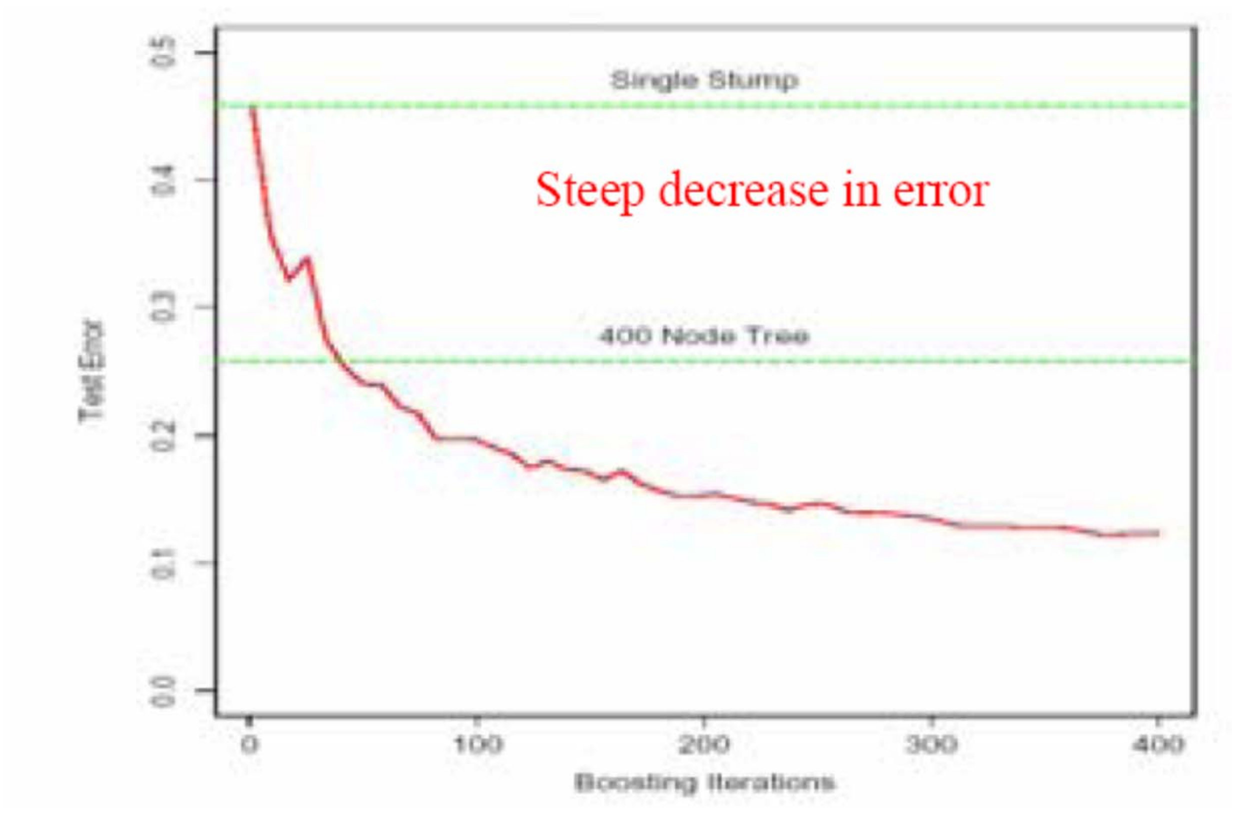


Boosting Performance

- **Decision stumps are very simple classifiers that test condition on a single attribute**
- **Suppose we use decision stumps as individual classifiers whose predictions were combined to generate the final prediction**
- **Suppose we plot the misclassification rate of the Boosting algorithm against the number of iterations performed**



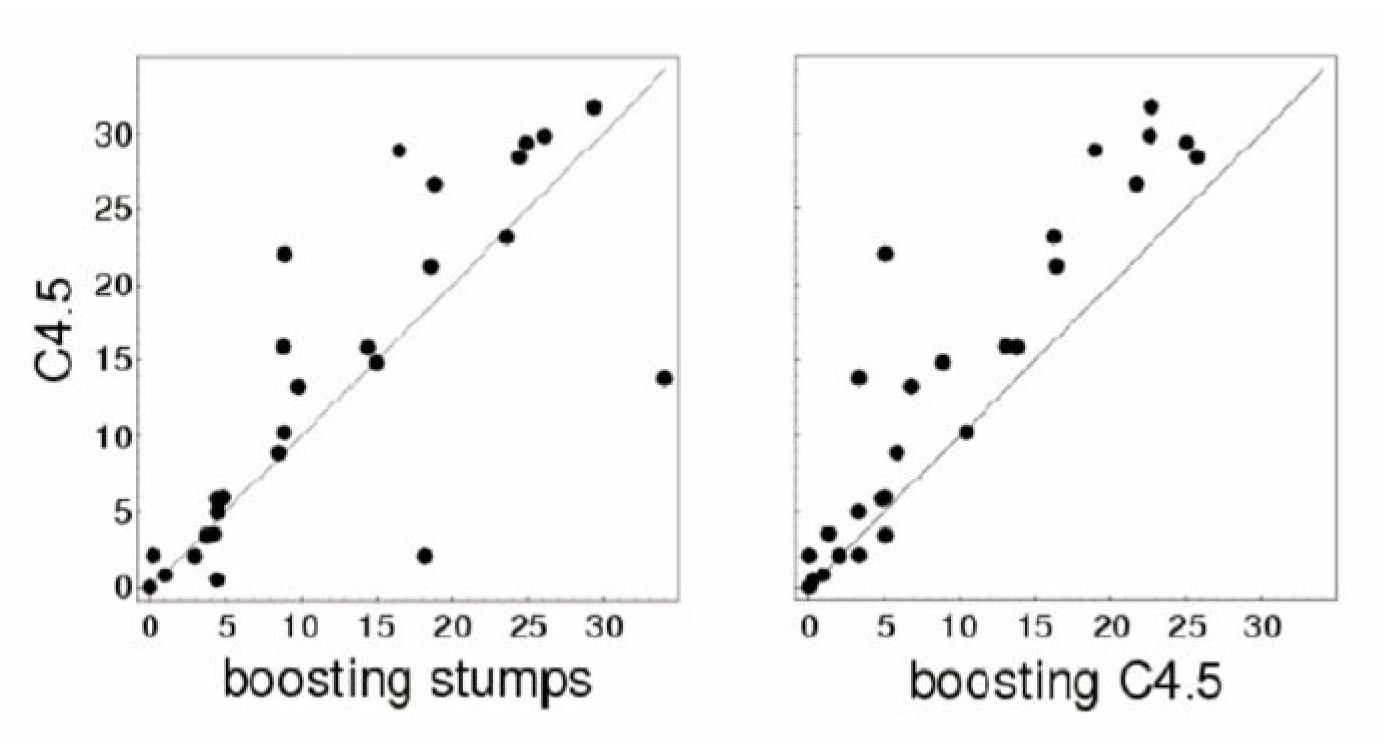
Boosting Performance



SOGANG UNIVERSITY

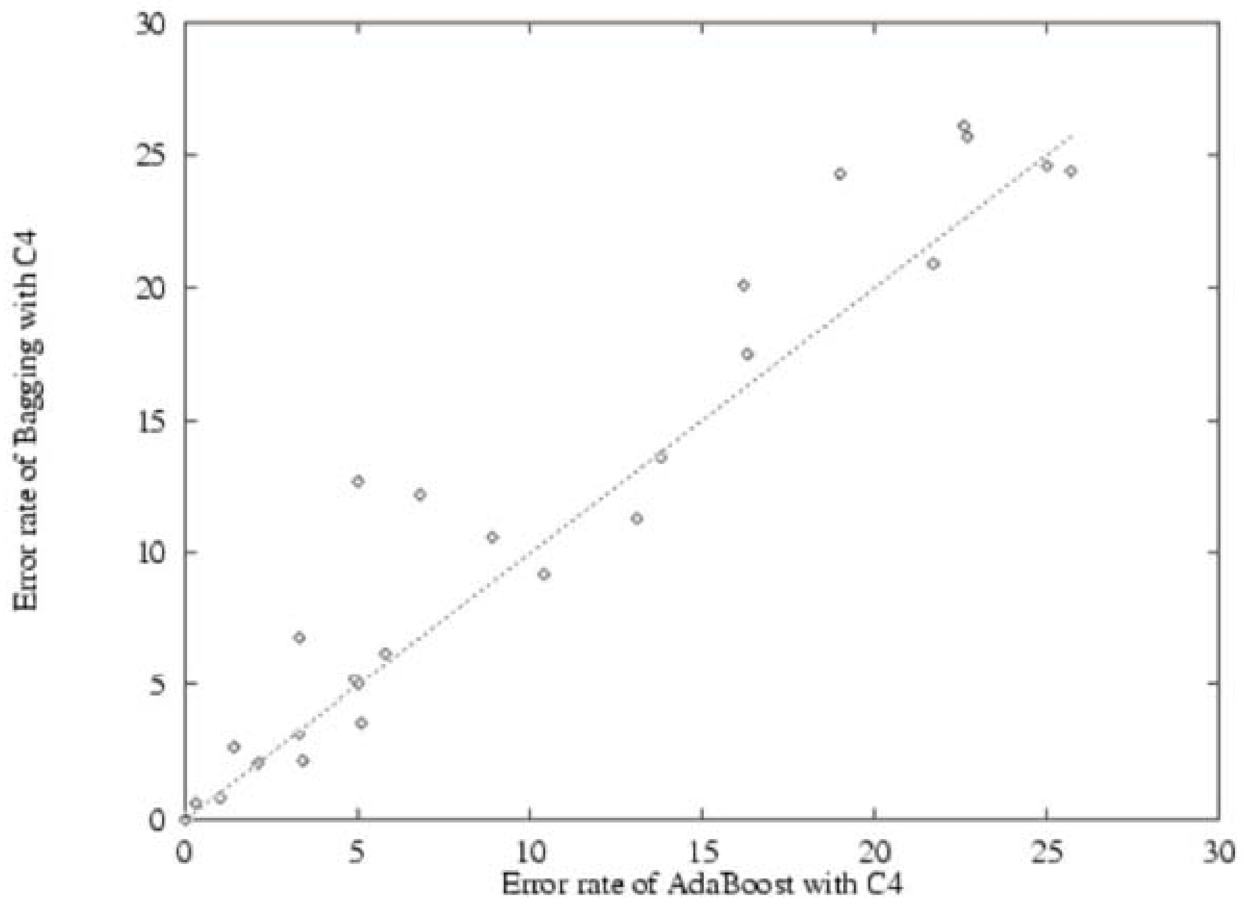
Boosting Performance

- **Comparing C4.5, boosting decision stumps, boosting C4.5 using 27 UCI datasets**



SOGANG
UNIVERSITY

Boosting vs. Bagging of Decision Trees



Boosting Performance

- Observations
  - First few (about 50) iterations increase the accuracy substantially – seen by the steep decrease in misclassification rate
  - As iterations increase training error decreases
  - As iterations increase, generalization error decreases?
    → very loose theoretical upper bound on generalization error

Can Boosting do well if?

- Individual classifiers are not very accurate and have high variance (e.g. decision stumps)?
  - It can if the individual classifiers have considerable mutual disagreement

- Individual classifier is very accurate and has low variance (e.g. SVM with a good kernel function)?
  - No..

Boosting as Additive Model

- The final prediction in boosting *f(x)* can be expressed as an *additive expansion* of individual classifiers

$$f(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m)$$

- The process is *iterative* and can be expressed as follows

$$f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$$

- Typically we could try to *minimize a loss function* on the training examples

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m) \right)$$

Boosting as Additive Model

- **Simple case: squared-error loss**

$$L(y, f(x)) = \frac{1}{2}(y - f(x))^2$$

- **Forward stage-wise modeling amounts to just fitting the residuals from previous iteration**

$$\begin{aligned} & L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma_m)) \\ &= (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma_m))^2 \\ &= (r_{im} - \beta b(x_i;\gamma_m))^2 \end{aligned}$$

Boosting as Additive Model

- **AdaBoost for classification uses the exponential loss function:**

L(y, f(x)) = exp(−y·f(x))

$$\arg\min_{f} \sum_{i=1}^{N} L\left(y_i, f(x_i)\right)$$

$$= \arg\min_{\beta, G_m} \sum_{i=1}^{N} \exp\left(-y_i \cdot \left[f_{m-1}(x_i) + \beta \cdot G_m(x_i)\right]\right)$$

$$= \arg\min_{\beta, G_m} \sum_{i=1}^{N} \exp\left(-y_i \cdot f_{m-1}(x_i)\right) \exp\left(-y_i \cdot \beta \cdot G_m(x_i)\right)$$

SOGANG UNIVERSITY

Boosting as Additive Model

- **First assume that *β* is constant, and minimize w.r.t. *G*:**

$$\arg\min_{\beta, G_m} \sum_{i=1}^{N} \exp\left(-y_i \cdot f_{m-1}(x_i)\right)\exp\left(-y_i \cdot \beta \cdot G_m(x_i)\right)$$

$$= \arg\min_{\beta, G_m} \sum_{i=1}^{N} w_i^{(m)} \cdot \exp\left(-y_i \cdot \beta \cdot G_m(x_i)\right), \text{ where } w_i^{(m)} = \exp\left(-y_i \cdot f_{m-1}(x_i)\right)$$

$$= \arg\min_{G} \sum_{y_i = G(x_i)}^{N} w_i^{(m)} \cdot e^{-\beta} + \sum_{y_i \neq G(x_i)}^{N} w_i^{(m)} \cdot e^{\beta}$$

$$= \arg\min_{G} \left(e^{\beta} - e^{-\beta}\right) \sum_{i=1}^{N} \left[w_i^{(m)} \cdot I\left(y_i \neq G(x_i)\right)\right] + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}$$

$$= \arg\min_{G} \left(e^{\beta} - e^{-\beta}\right) \frac{\sum_{i=1}^{N} \left[w_i^{(m)} \cdot I\left(y_i \neq G(x_i)\right)\right]}{\sum_{i=1}^{N} w_i^{(m)}} + e^{-\beta}$$

Boosting as Additive Model

$$\arg\min_{G}\left(e^{\beta}-e^{-\beta}\right)\boxed{\frac{\sum_{i=1}^{N}\left[w_i^{(m)}\cdot I\left(y_i\neq G(x_i)\right)\right]}{\sum_{i=1}^{N}w_i^{(m)}}}+e^{-\beta}$$

$$=\arg\min_{G}\left(e^{\beta}-e^{-\beta}\right)\cdot\boxed{err_m}+e^{-\beta}=H(\beta)$$

- ***Errm***: **training error on the weighted samples**

- ***On each iteration we must find a classifier that minimizes the training error on the weighted samples!***

Boosting as Additive Model

- **Note that we have found *G*, we minimize w.r.t. *β*:**

$$H(\beta) = err_m \cdot \left(e^{\beta} - e^{-\beta}\right) + e^{-\beta}$$

$$\frac{\partial H}{\partial \beta} = err_m \cdot \left(e^{\beta} + e^{-\beta}\right) - e^{-\beta} = 0$$

$$1 - e^{\beta} \cdot err_m \left(e^{\beta} + e^{-\beta}\right) = 0$$

$$1 - e^{2\beta} \cdot err_m - err_m = 0$$

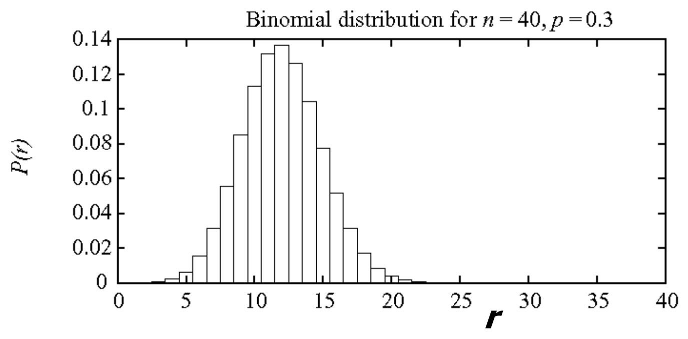
$$\frac{1 - err_m}{err_m} = e^{2\beta}$$

$$\beta = \frac{1}{2} \ln\left(\frac{1 - err_m}{err_m}\right)$$

SOGANG UNIVERSITY

Why do ensembles work?

- Because *uncorrelated* errors of individual classifiers can be eliminated by averaging
- Assume: 40 base classifiers, majority voting, each error rate 0.3
- Probability of getting *r* incorrect votes from 40 classifiers



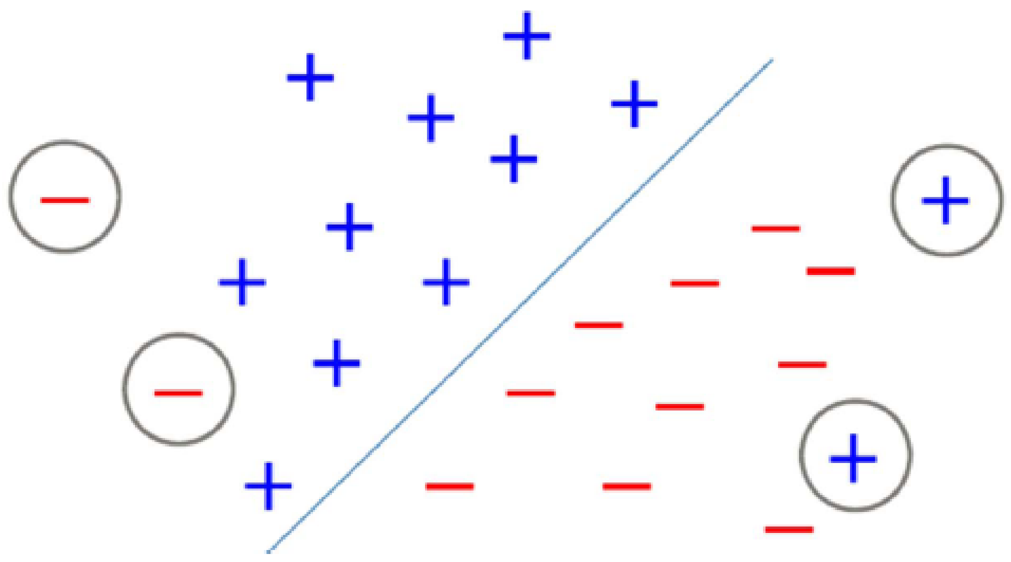
$$P(r) = \frac{n!}{r!(n-r)!} error_D(h)^r (1 - error_D(h))^{n-r}$$

- *p*(ensemble is wrong) = *p*(>20 incorrect votes) = 0.01

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 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

Error Correcting Output Codes

- So far, we've been building the ensemble by tweaking the set of training instances

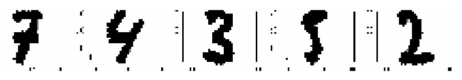
출력변조

- ECOC involves tweaking the output (class) to be learned

- Solving multiclass learning problems
  - using an ensemble of binary classifiers

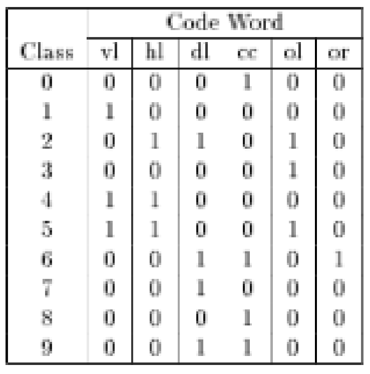
Thomas G. Dietterich and G. Bakiri (1995), Solving Multiclass Learning Problems via Error-Correcting Output Codes, *JAIR*.

Example: Handwritten number recognition

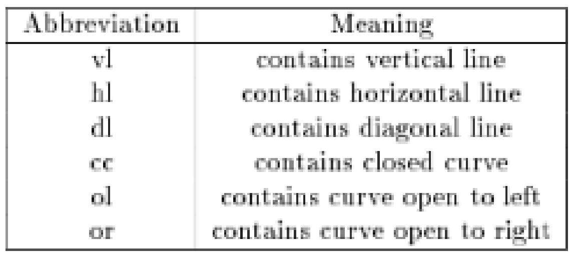
 $\rightarrow$ 7, 4, 3, 5, 2

- “obvious” approach: learn function: Scribble $\rightarrow$ {0,1,2,⋯,9}
  $\rightarrow$ doesn’t work very well (too hard!)

- What if we “decompose” the learning task into six “subproblems”?



Class	Code Word					
	vl	hl	dl	cc	ol	or
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	0	1	0	0
9	0	0	1	1	0	0

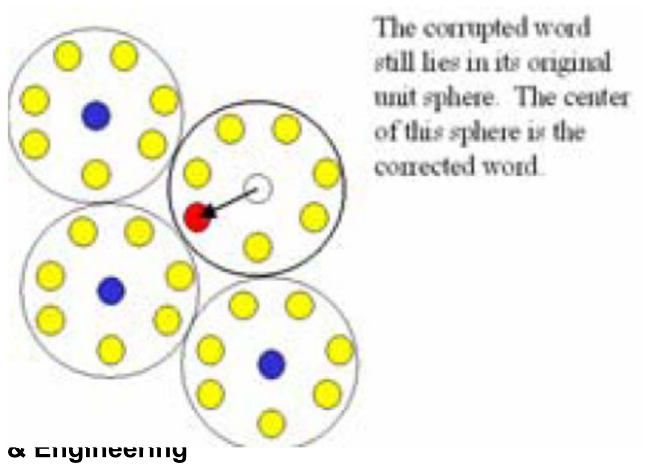


Abbreviation	Meaning
vl	contains vertical line
hl	contains horizontal line
dl	contains diagonal line
cc	contains closed curve
ol	contains curve open to left
or	contains curve open to right

1. Learn an ensemble of classifiers, one specialized to each of the 6 “sub-problems”
2. To classify a new scribble, invoke each ensemble member, then predict the class whose code-word is closest (Hamming distance) to the predicted code

Error Correcting Codes

- Suppose we want to send *n*-bit messages through a noisy channel
- To ensure robustness to noise, we can map each *n*-bit message into an *m*-bit code word (*m* > *n*) – note |*codes*| >> |*messages*|
- When we receive a code (*m*-bit string), translate it to message corresponding to the "nearest" (Hamming distance) code
- Key to robustness: Assign the codes so that each *n*-bit "clean" message is surrounded by a "buffer zone" of similar *m*-bit codes to which no other *n*-bit message is mapped


The corrupted word still lies in its original unit sphere. The center of this sphere is the corrected word.

blue = message (n bits)
yellow = code (m bits)

white = intended message
red = received code

Error Correcting Codes

- A measure of the quality of an error-correcting code is the minimum Hamming distance between any pair of code words
- If the minimum Hamming distance is *d*, then the code can correct at least (*d*-1)/2 single bit errors

ISBN

- The International Standard Book Number (ISBN) system identifies every book with a ten-digit number, such as 0-226-53420-0 (13 digits since January 1, 2007)

- The first nine digits are the actual number but the tenth is added according to a mathematical formula based on the first nine

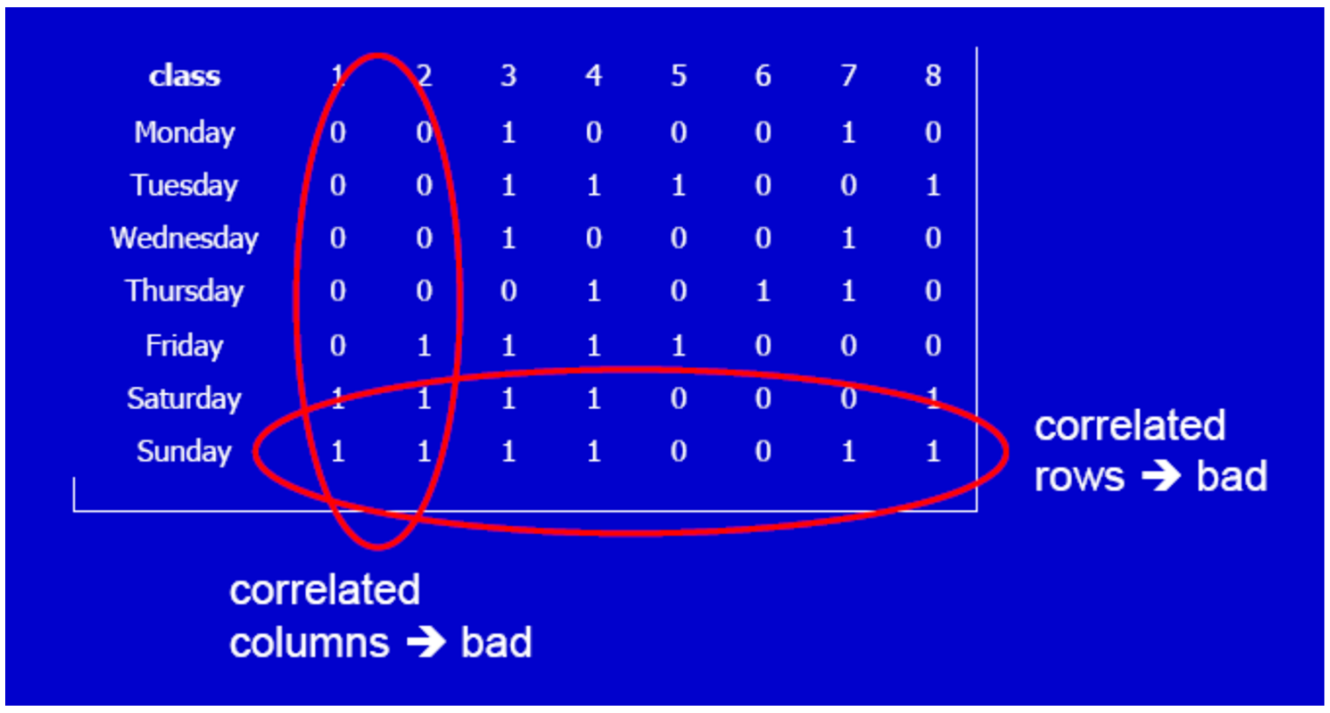
- If a single one of the digits is changed, as in a misprint when ordering a book, a simple check verifies that something is wrong

Designing Code-words for ECOC Learning

- Coding: *k* labels → *m* bit codewords
- Good coding:
  - Row separation: want “assigned” codes to be well-separated by lots of “unassigned” codes
  - Column separation: each bit *i* of the codes should be uncorrelated with all other bits *j*
- Selecting good codes is hard!

class	1	2	3	4	5	6	7	8
Monday	0	0	1	0	0	0	1	0
Tuesday	0	0	1	1	1	0	0	1
Wednesday	0	0	1	0	0	0	1	0
Thursday	0	0	0	1	0	1	1	0
Friday	0	1	1	1	1	0	0	0
Saturday	1	1	1	1	0	0	0	1
Sunday	1	1	1	1	0	0	1	1

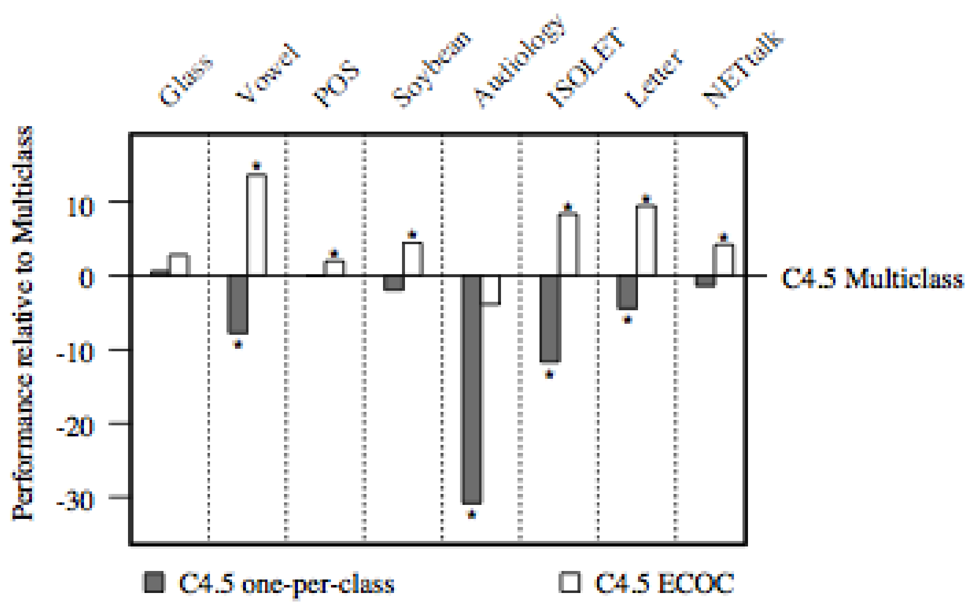
Bad Codes



class	1	2	3	4	5	6	7	8
Monday	0	0	1	0	0	0	1	0
Tuesday	0	0	1	1	1	0	0	1
Wednesday	0	0	1	0	0	0	1	0
Thursday	0	0	0	1	0	1	1	0
Friday	0	1	1	1	1	0	0	0
Saturday	1	1	1	1	0	0	0	1
Sunday	1	1	1	1	0	0	1	1

correlated rows ➔ bad
correlated columns ➔ bad

Performance of ECOC



An asterisk indicates that the difference is statistically significant at the 0.05 level

SOGANG UNIVERSITY

SUMMARY

- Ensembles: basic motivation – creating a committee of experts is typically more effective than trying to derive a single super-genius

- Key issues:
  - Generation of base models
  - Integration of base models

- Popular ensemble techniques
  - Manipulate training data: *bagging* and *boosting* (ensemble of "experts", each specializing on different portions of the instance space)
  - Manipulate input feature space: train classifiers using different subset of features; work when the input features are highly redundant
  - Manipulate output values: *error-correcting output coding* (ensemble of "experts", each predicting 1 bit of the multibit full class label)