

HTF: 8.7, 10, 16
B: 14 – 14.3



Ensemble Methods

R Greiner
Cmput 466 / 551

Some material from Tom Dietterich, E Roberto, M Botta, R Schapire

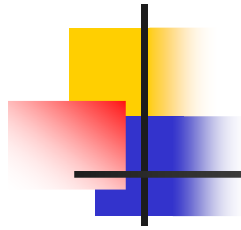


Motivation

- If **1** learner is good
 - produces **1** effective classifiermaybe **many** would be better
- Eg, why not learn $\{ h_1, h_2, h_3 \}$, then
 - $h^*(x) = \text{majority}\{ h_1(x), h_2(x), h_3(x) \}$
- If h_i 's make INDEPENDENT mistakes, h^* is more accurate!
 - Eg: If $\text{err}(h_i) = \epsilon$, then $\text{err}(h^*) = 3\epsilon^2$
(0.01 \rightarrow 0.0003)
 - If use majority of $2k-1$ hyp, then

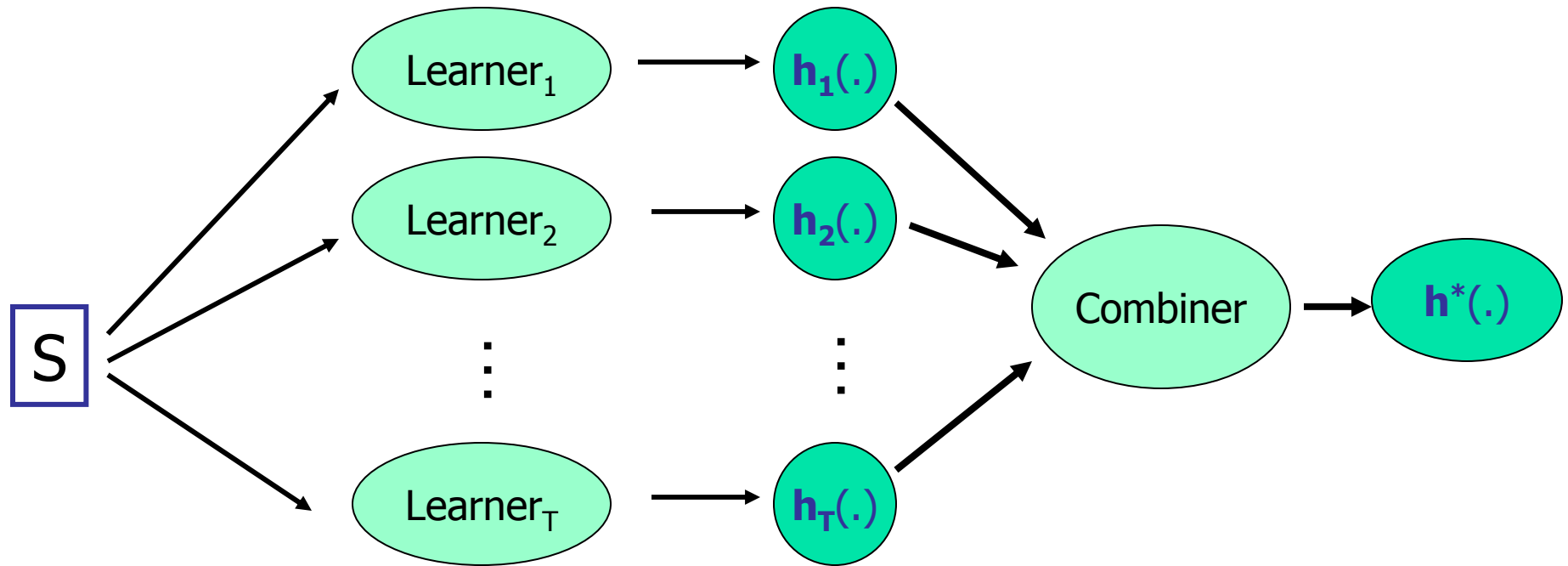
$$\text{err}(h) \approx \binom{2k-1}{k} \epsilon^k$$

Learn then Combine Many Classifiers



Original Data

Classifier





Challenges

1. How to generate the base classifiers?

- h_1, h_2, \dots
- Different learners, Bootstrap samples, ...

2. How to integrate/combine them?

- $h^*(x) = F(h_1(x), h_2(x), \dots)$
- Average, Weighted Average, Instance-specific decisions, ...



Types of Ensemble Methods



1. Subsample Training Sample

- Bagging
- Boosting



2. Manipulate Input Features



3. Manipulate Output Targets

- ECOC

4. Injecting Randomness

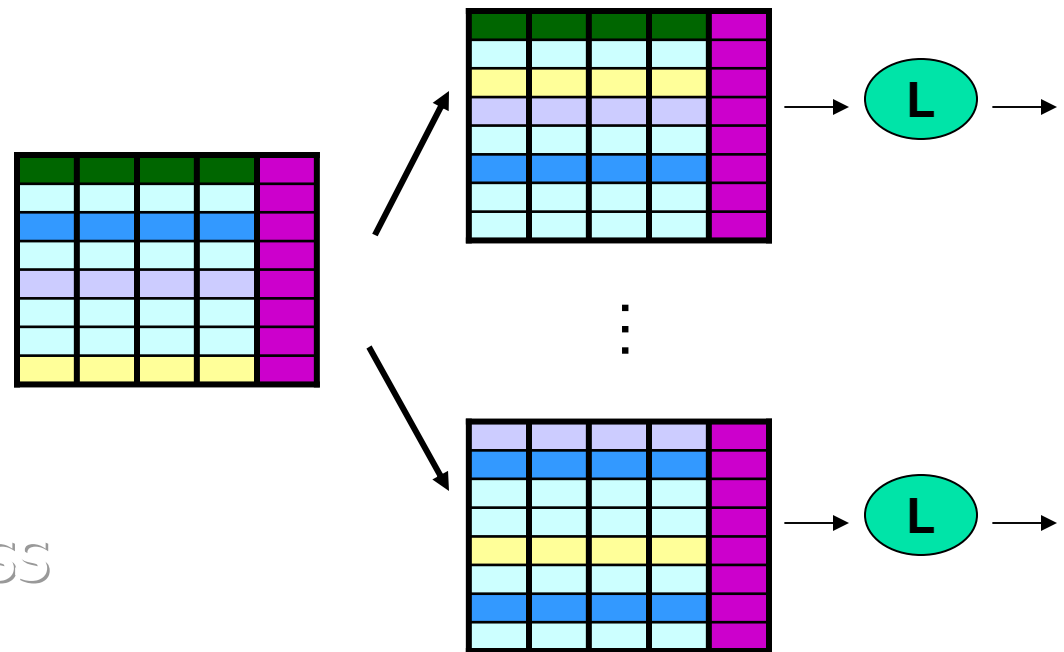
- Data
- Algorithm

5. Algorithm Specific methods

- Other combinations
- Why do Ensembles work?

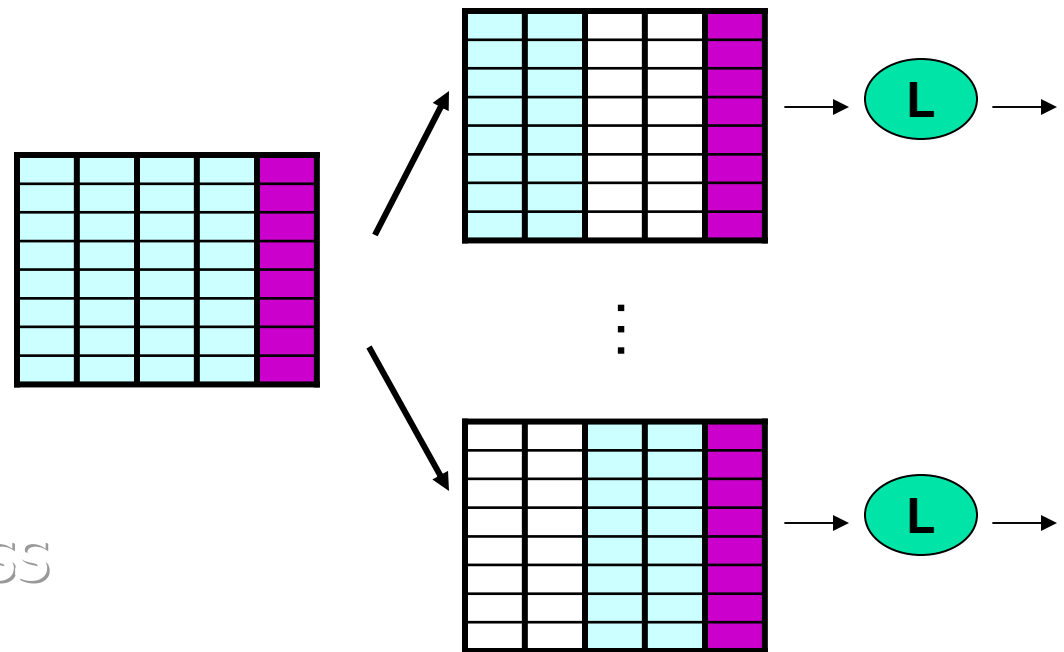
Types of Ensemble Methods

1. **Subsample Training Sample**
2. Manipulate Input Features
3. Manipulate Output Targets
4. Injecting Randomness
5. Algorithm Specific methods



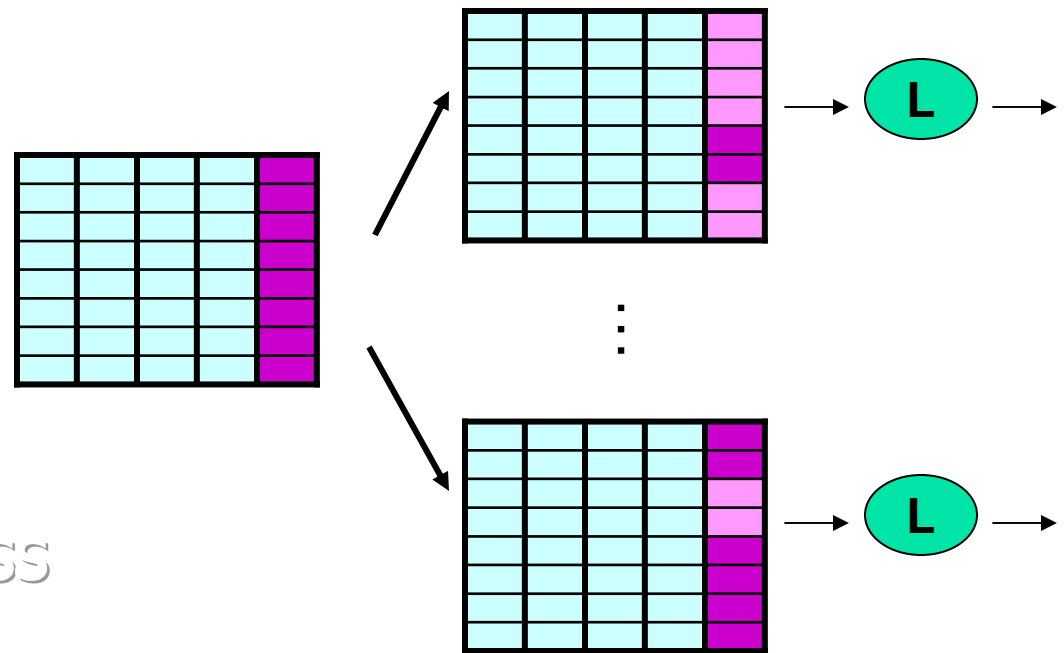
Types of Ensemble Methods

1. Subsample Training Sample
- 2. Manipulate Input Features**
3. Manipulate Output Targets
4. Injecting Randomness
5. Algorithm Specific methods



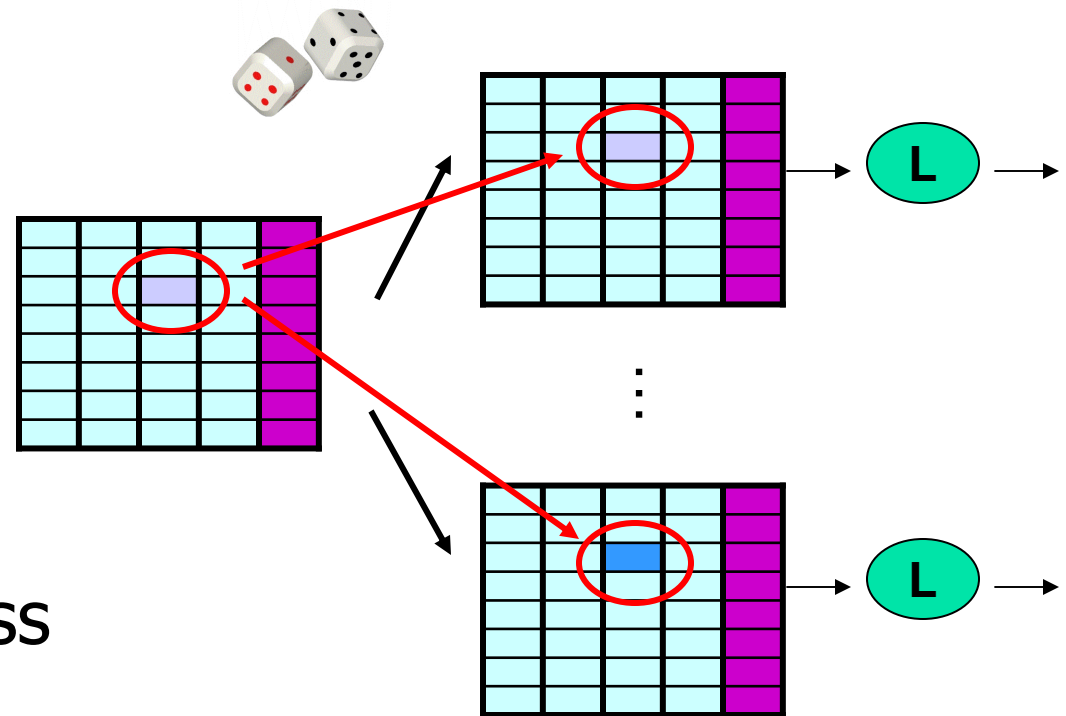
Types of Ensemble Methods

1. Subsample Training Sample
2. Manipulate Input Features
- 3. Manipulate Output Targets**
4. Injecting Randomness
5. Algorithm Specific methods



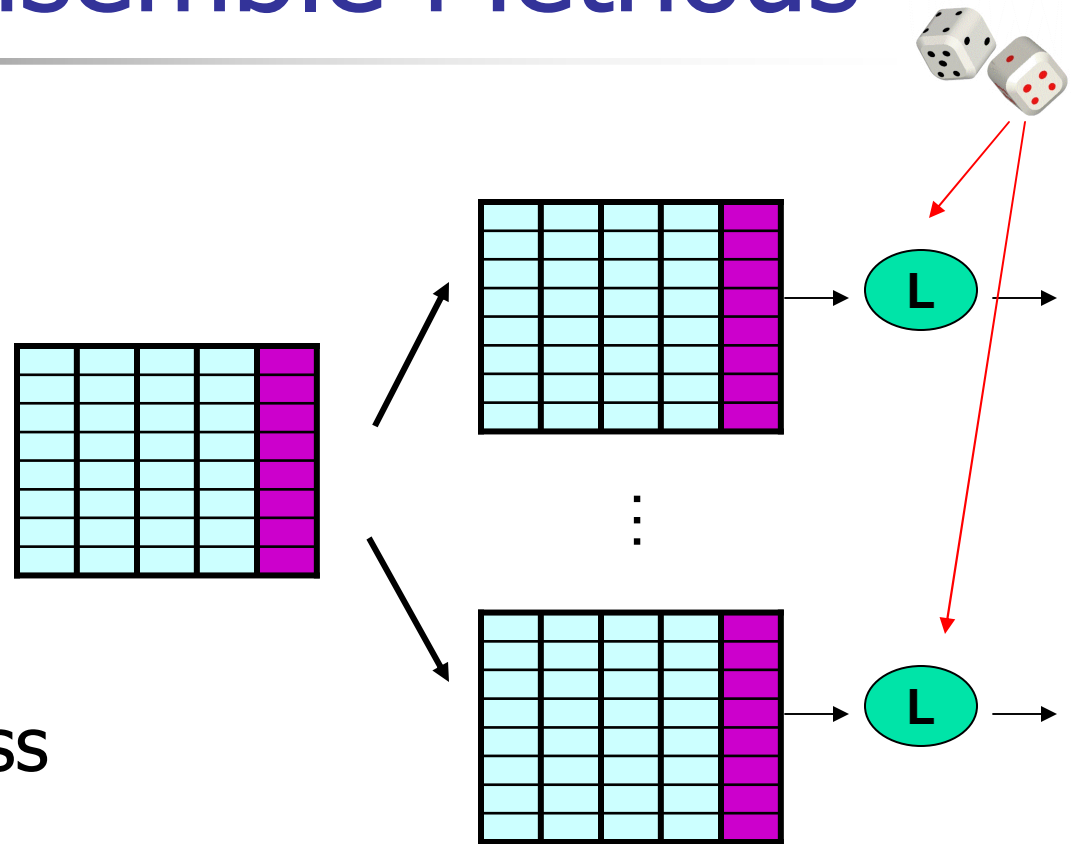
Types of Ensemble Methods

1. Subsample Training Sample
2. Manipulate Input Features
3. Manipulate Output Targets
- 4. Injecting Randomness**
5. Algorithm Specific methods



Types of Ensemble Methods

1. Subsample Training Sample
2. Manipulate Input Features
3. Manipulate Output Targets
- 4. Injecting Randomness**
5. Algorithm Specific methods





1. Subsample Training Sample

Defn: Learner is **UNSTABLE** if its output classifier undergoes **major** changes in response to **small** changes in training data

- Unstable: Decision-tree, neural network, rule learning alg's
- Stable: Linear regression, nearest neighbor, linear threshold algorithms, ...
- Subsampling is best for unstable learners
- Techniques:
 - (Cross-Validated Committees)
 - Bagging
 - Boosting

1a: Bagging: Bootstrap Aggregating

- For $b = 1, \dots, T$ do
 - S_b = bootstrap replicate of S
 - Apply learning algorithm to S_b to learn h_b
- To classify new point \mathbf{x} ,
using unweighted vote:

$$\hat{h}(x) = \text{sign} \left(\frac{1}{T} \sum_{i=1}^T h_i(x) \right)$$

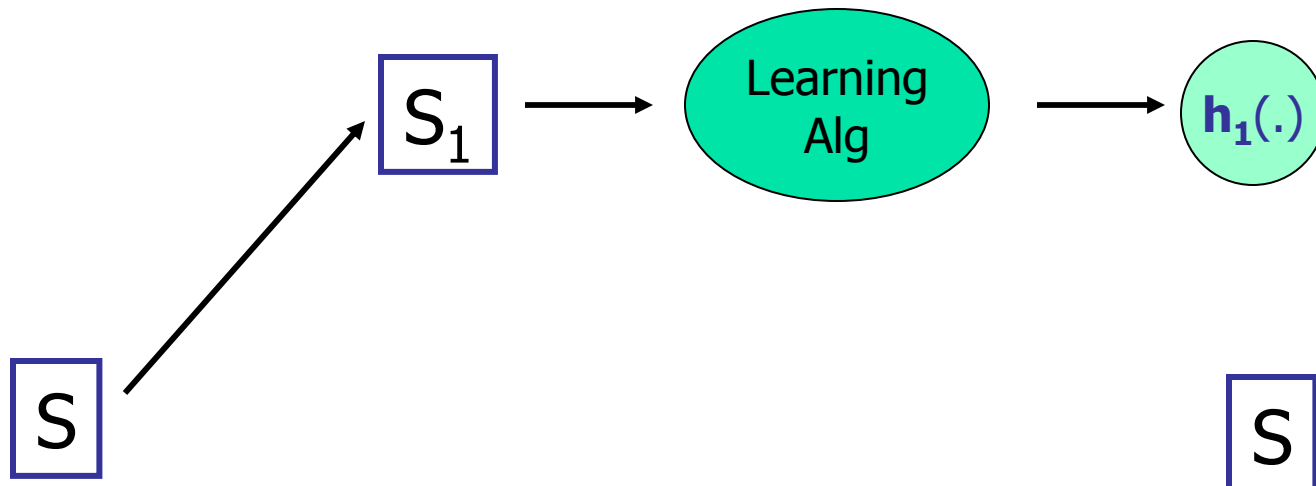
$$\hat{h}(x) = \operatorname{argmax}_r \{ |h_i(x) = r| \}$$

Bootstrap Replicates

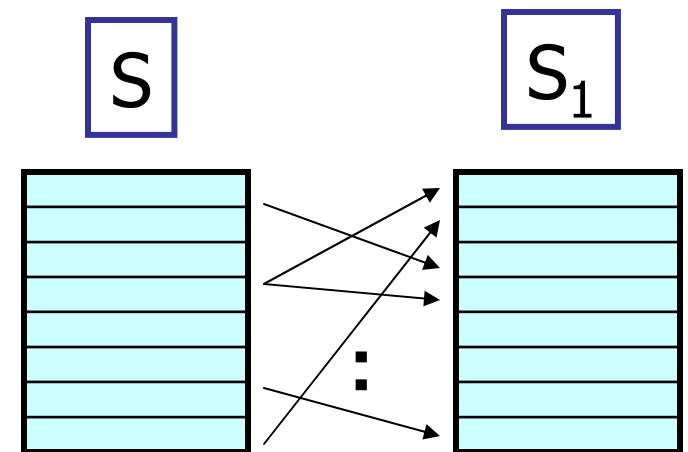
Original Data

Bootstrap Replicate

Classifier



Form S_1 by drawing $|S|$ instances from S , with replacement

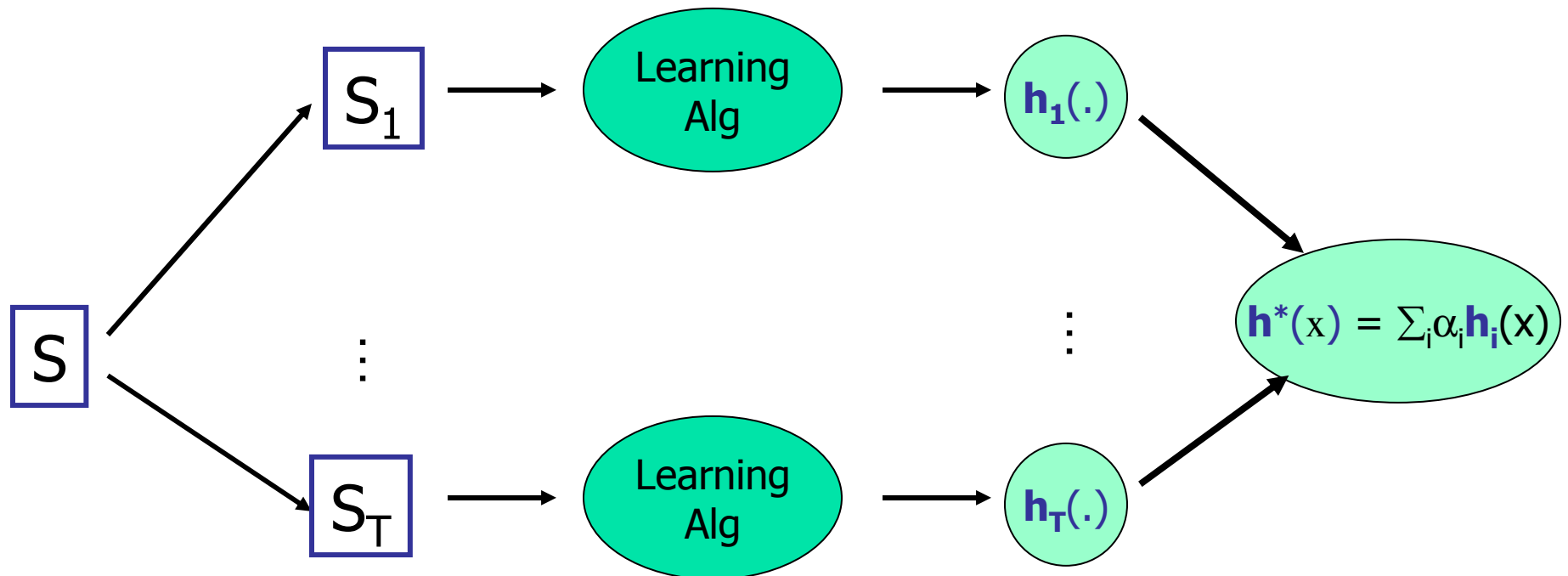


Bootstrap Replicates

Original Data

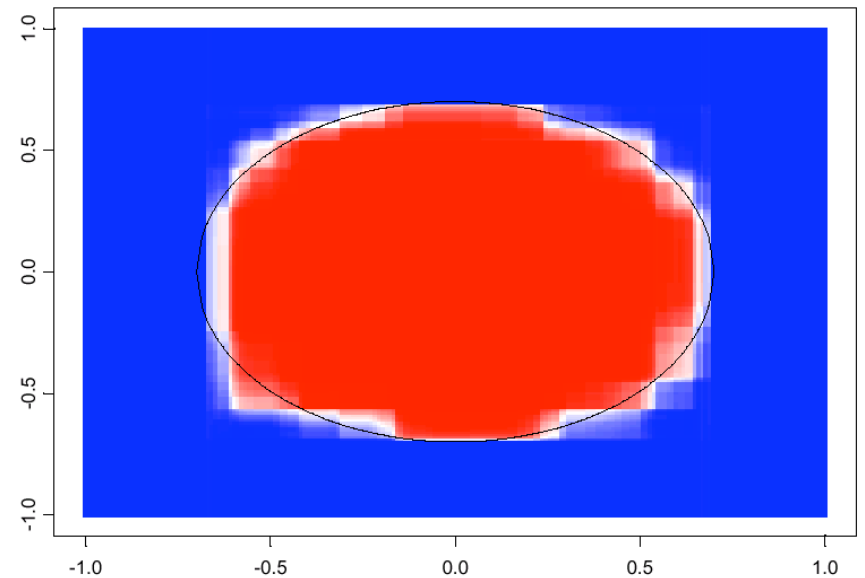
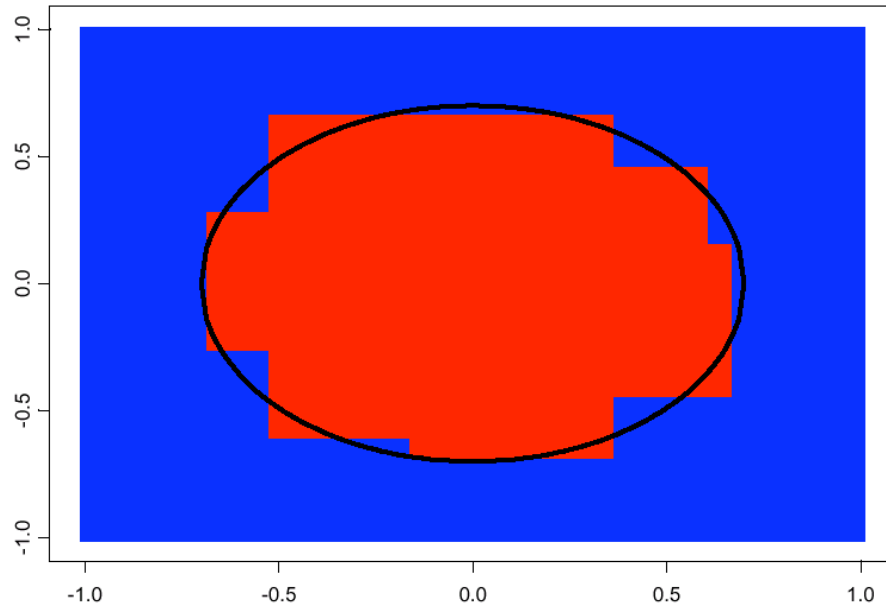
Bootstrap Replicate

Classifier



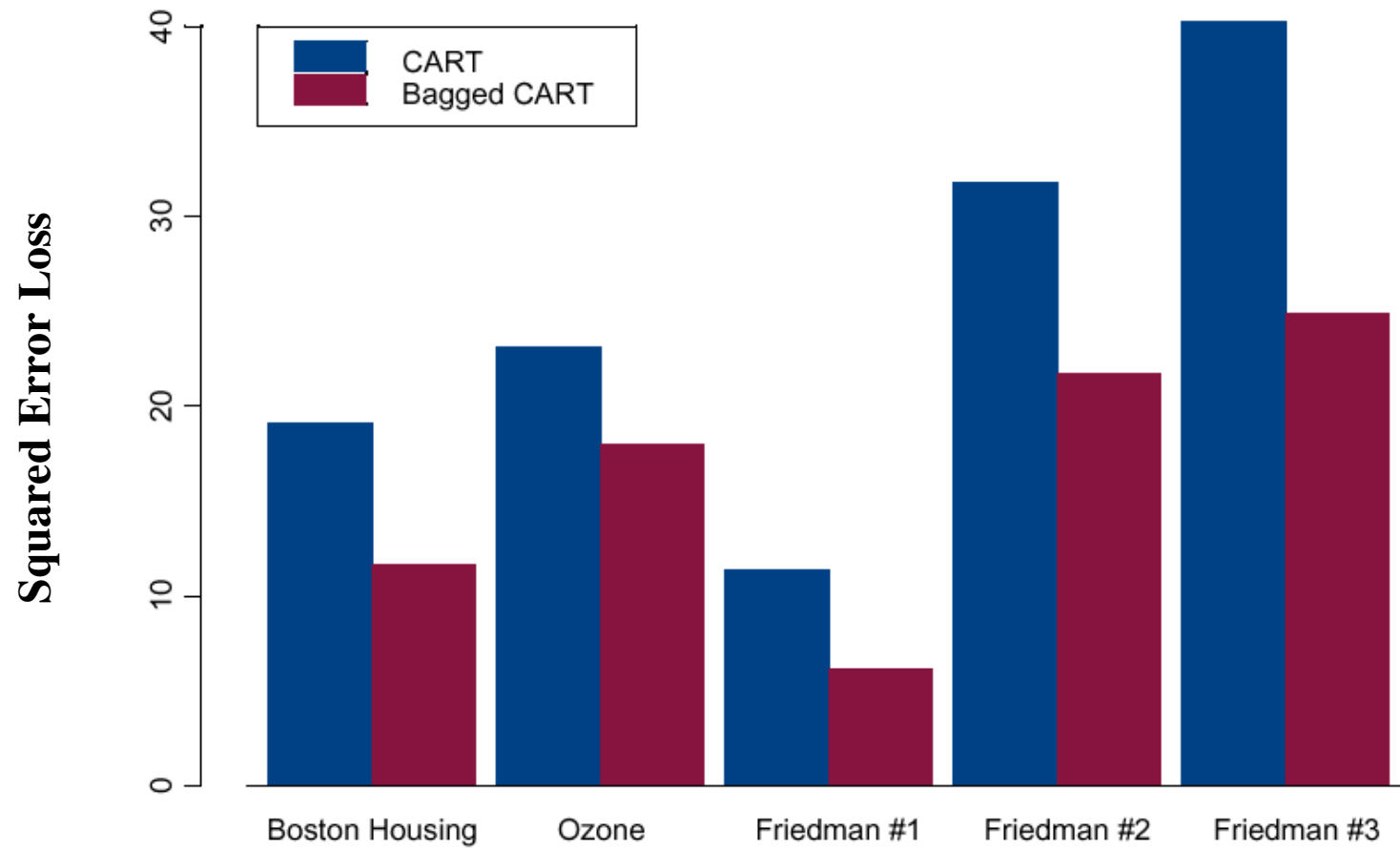
- Each S_i is bootstrap replicate
- h_i = classifier based on S_i
- $\alpha_i = 1/T$

CART vs Boosted-CART

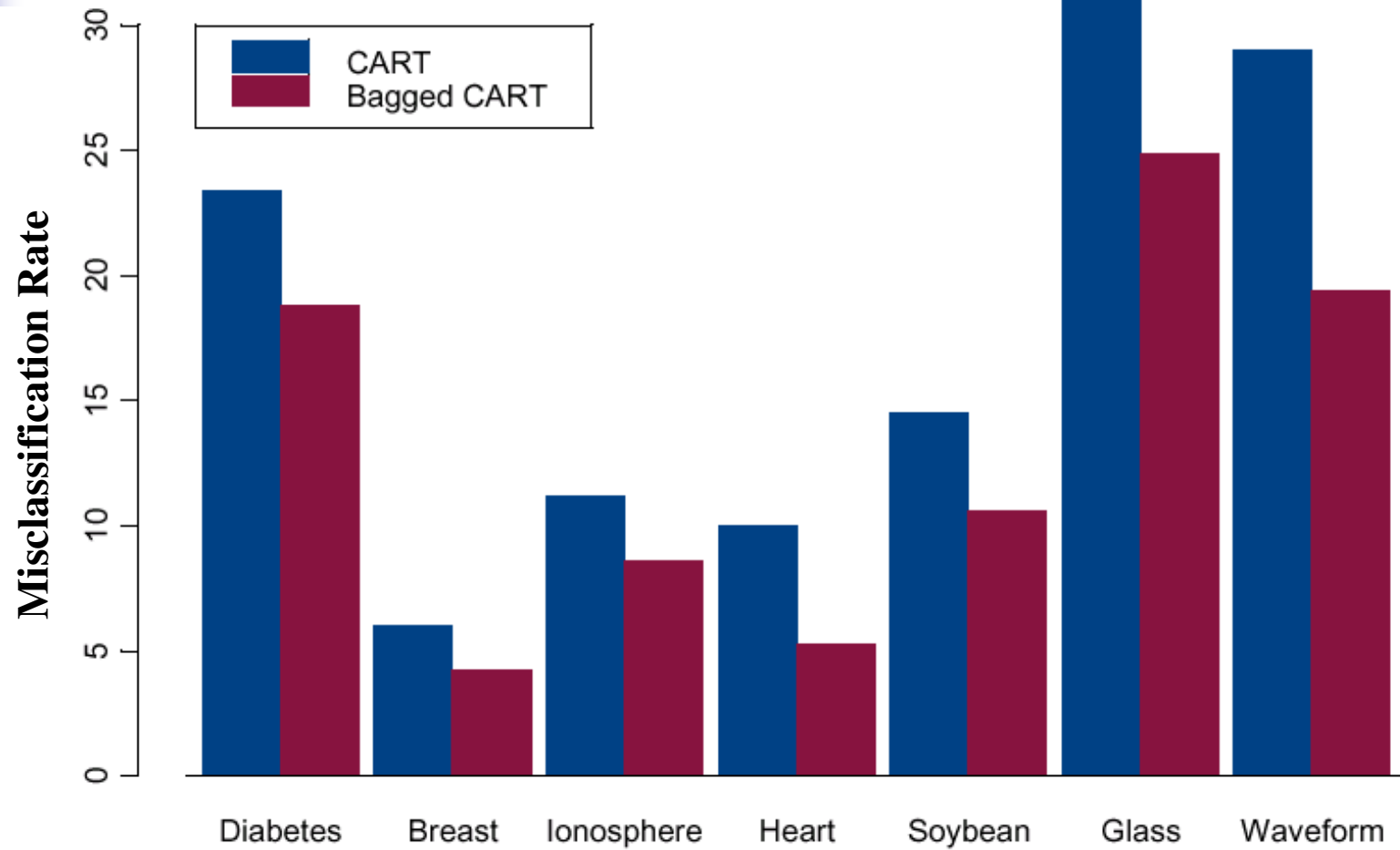


- 100 bagged trees
- shades of blue/red indicate strength of vote for particular classification

Regression Results



Classification Results



Expected Error

$$\hat{h}(x) = \frac{1}{T} \sum_{i=1}^T h_i(x)$$

- Assume $h_i(\mathbf{x}) = y(\mathbf{x}) + \varepsilon_i(\mathbf{x})$
- $E_{\mathbf{x}}[(h_i(\mathbf{x}) - y(\mathbf{x}))^2] = E_{\mathbf{x}}[\varepsilon_i(\mathbf{x})^2]$
- ... so average MSError (over T regressors) is

$$E_{AV} = 1/T \sum_i E_{\mathbf{x}}[\varepsilon_i(\mathbf{x})^2]$$

- What is average error of about $\hat{h}(\mathbf{x})$?
 - Assume $\varepsilon_i(\mathbf{x})$ each 0-mean and uncorrelated

$$E_{\mathbf{x}}[\varepsilon_i(\mathbf{x})] = 0$$

$$E_{\mathbf{x}}[\varepsilon_i(\mathbf{x}) \varepsilon_j(\mathbf{x})] = 0 \quad \text{for } i \neq j$$

$$\begin{aligned} E_{\hat{h}} &= E_{\mathbf{x}}[y(\mathbf{x}) - 1/T \sum_i h_i(\mathbf{x})]^2 \\ &= E_{\mathbf{x}}[1/T \sum_i \varepsilon_i(\mathbf{x})]^2 = 1/T E_{AV} !! \end{aligned}$$

- In general: $E_{\hat{h}} \leq E_{AV}$



Estimated Bias, Variance of Bagging

- Estimating bias and variance using same L bootstrap samples,
 - Bias = $E_{\mathbf{x}}[\underline{h}(\mathbf{x}) - y]$ (same as before)
 - Variance = $\sum_i (h_i(\mathbf{x}) - \hat{h}(\mathbf{x}))^2 / (T-1) \dots = 0 !$
- ⇒ (Given this estimate of variance) Bagging ...
 - removes variance
 - leaves bias unchanged
- Actually...
 - bagging only *reduces* variance
 - tends to slightly increase bias



Bias/Variance Heuristics

- Models that fit data *poorly* have high bias:
 - “inflexible models”,
 - eg, linear regression, regression stumps
 - Models that can fit data *very well* have low bias but often high variance:
 - “flexible” models
 - eg, nearest neighbor regression, regression trees
- ⇒ bagging of flexible models can reduce variance while benefiting from low bias



Types of Ensemble Methods

1. Subsample Training Sample
 - Bagging
 - Boosting
2. Manipulate Input Features
3. Manipulate Output Targets
 - ECOC
4. Injecting Randomness
5. Algorithm Specific methods
 - Other combinations
 - Why do Ensembles work?



1b: Boosting

- **Boosting** = general method of using...
 - “weak” learning algorithm $L(\dots)$
 - can reliably produce classifiers (at least) slightly better than random,
 - say, accuracy $\geq 55\%$ (in two-class setting)

to produce highly accurate predictor

- single classifier with very high accuracy,
 - say, 99%

... given sufficient data...



Strong vs Weak Learnability

- boosting's roots are in "PAC" learning model (Valiant)
- Given random examples from unknown, arbitrary distribution...
- *strong* PAC learning algorithm:
 - for any distribution, with high probability, given poly # of examples, polynomial time,
 - can always find classifier with *arbitrarily small generalization error*
- *weak* PAC learning algorithm
 - same... but generalization error only needs to be *slightly better than random guessing* ($1/2 - \gamma$)
- [Kearns & Valiant '88]:
 - does weak learnability imply strong learnability?



Early Boosting Algorithms

- [Kearns & Valiant '88]:
 - does weak learnability imply strong learnability?
- YES! [Schapire '89]:
 - provable boosting algorithm
- [Freund '90]:
 - “optimal” algorithm that “boosts by majority”
- [Drucker, Schapire & Simard '92]:
 - first experiments using boosting
 - limited by practical drawbacks

AdaBoost

- [Freund & Schapire '95]:
 - introduced “AdaBoost” algorithm
 - strong practical advantages over previous boosting algorithms
- experiments and applications using AdaBoost:

[Drucker & Cortes '96]

[Jackson & Craven '96]

[Freund & Schapire '96]

[Quinlan '96]

[Breiman '96]

[Maclin & Opitz '97]

[Bauer & Kohavi '97]

[Schwenk & Bengio '98]

[Schapire, Singer & Singhal '98]

[Abney, Schapire & Singer '99]

[Haruno, Shirai & Ooyama '99]

[Cohen & Singer '99]

[Dietterich '00]

[Schapire & Singer '00]

[Collins '00]

[Escudero, Márquez & Rigau '00]

[Iyer, Lewis, Schapire et al. '00]

[Onoda, Rätsch & Müller '00]

[Tieu & Viola '00]

[Walker, Rambow & Rogati '01]

[Rochery, Schapire, Rahim & Gupta '01]

[Merler, Furlanello, Larcher & Sboner '01]

[Di Fabrizio, Dutton, Gupta et al. '02]

[Qu, Adam, Yasui et al. '02]

[Tur, Schapire & Hakkani-Tür '03]

[Viola & Jones '04]

[Middendorf, Kundaje, Wiggins et al. '04]

⋮

- continuing development of theory and algorithms:

[Breiman '98, '99]

[Schapire, Freund, Bartlett & Lee '98]

[Grove & Schuurmans '98]

[Mason, Bartlett & Baxter '98]

[Schapire & Singer '99]

[Cohen & Singer '99]

[Freund & Mason '99]

[Domingo & Watanabe '99]

[Mason, Baxter, Bartlett & Frean '99]

[Duffy & Helmbold '99, '02]

[Freund & Mason '99]

[Ridgeway, Madigan & Richardson '99]

[Kivinen & Warmuth '99]

[Friedman, Hastie & Tibshirani '00]

[Rätsch, Onoda & Müller '00]

[Rätsch, Warmuth, Mika et al. '00]

[Allwein, Schapire & Singer '00]

[Friedman '01]

[Koltchinskii, Panchenko & Lozano '01]

[Collins, Schapire & Singer '02]

[Demiriz, Bennett & Shawe-Taylor '02]

[Lebanon & Lafferty '02]

[Wyner '02]

[Rudin, Daubechies & Schapire '03]

[Jiang '04]

[Lugosi & Vayatis '04]

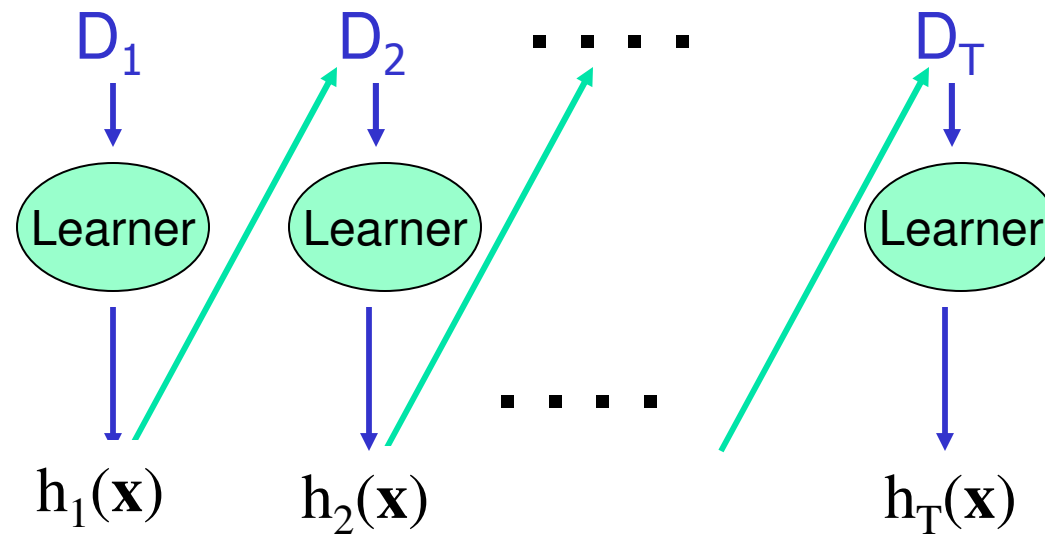
[Zhang '04]

⋮

Boosting Overview

Distribution:

$$\sum_i D_t(i) = 1$$



$$h^*(\mathbf{x}) = \text{sign}\left(\sum_t \alpha_t h_t(\mathbf{x})\right)$$



A Formal Description of Boosting

- Training set $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$
 - $y_i \in \{-1, +1\}$ correct label of instance $\mathbf{x}_i \in X$

- for $t = 1, \dots, T$:

- construct distribution D_t on $\{1, \dots, m\}$
- find weak classifier $h_t : X \rightarrow \{-1, +1\}$

with small error ε_t on D_t :

$$\varepsilon_t = \Pr_{i \in D_t} [h_t(\mathbf{x}_i) \neq y_i] = \sum_{i: y_i \neq h_t(\mathbf{x}_i)} D_t(i)$$

- output final classifier h^* based on $\{h_t(\mathbf{x})\}$



AdaBoost

- constructing D_t :

- $D_1(i) = 1/m$

- given D_t and h_t :
$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i))$$

where

$$\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} D_t(i)$$

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

Z_t = normalization constant

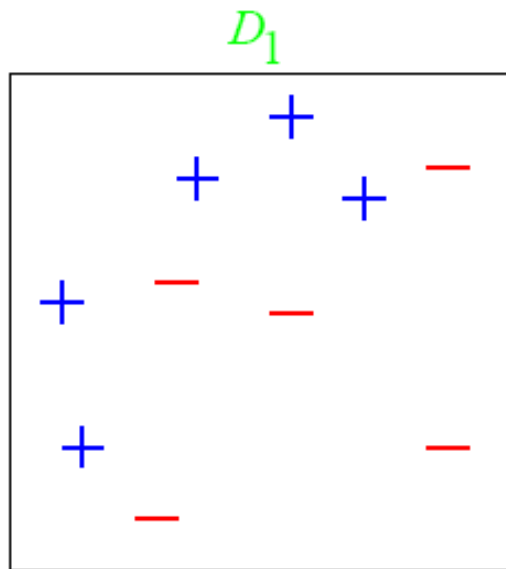
$\epsilon < 1/2$, so $\alpha > 0$, so $e^{-\alpha} < 1$, so ...

if correct, $D_{t+1}(i) < D_t(i)$... if wrong, $D_{t+1}(i) > D_t(i)$

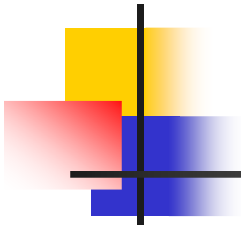
- final classifier:
$$h^*(\mathbf{x}) = \text{sign} \left(\sum_t \alpha_t h_t(\mathbf{x}) \right)$$



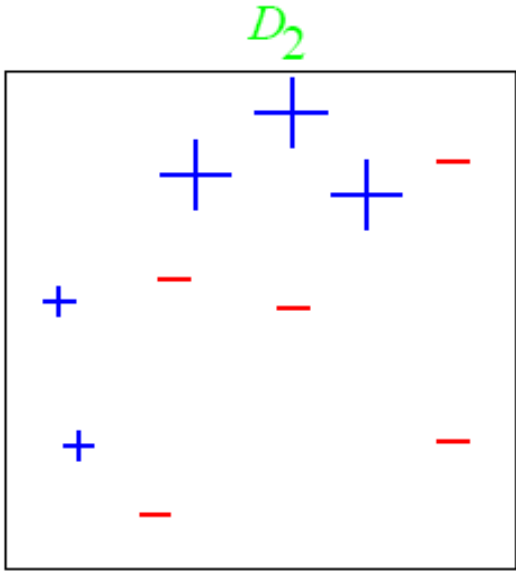
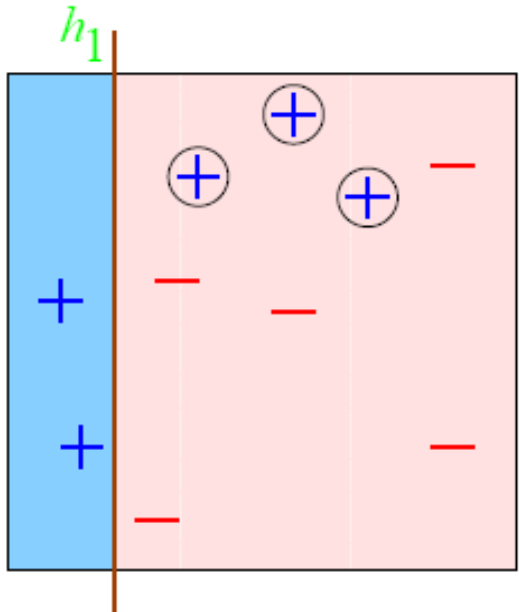
Toy Example



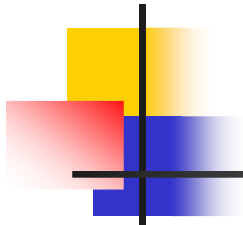
here: each weak classifiers = a vertical or horizontal half-planes



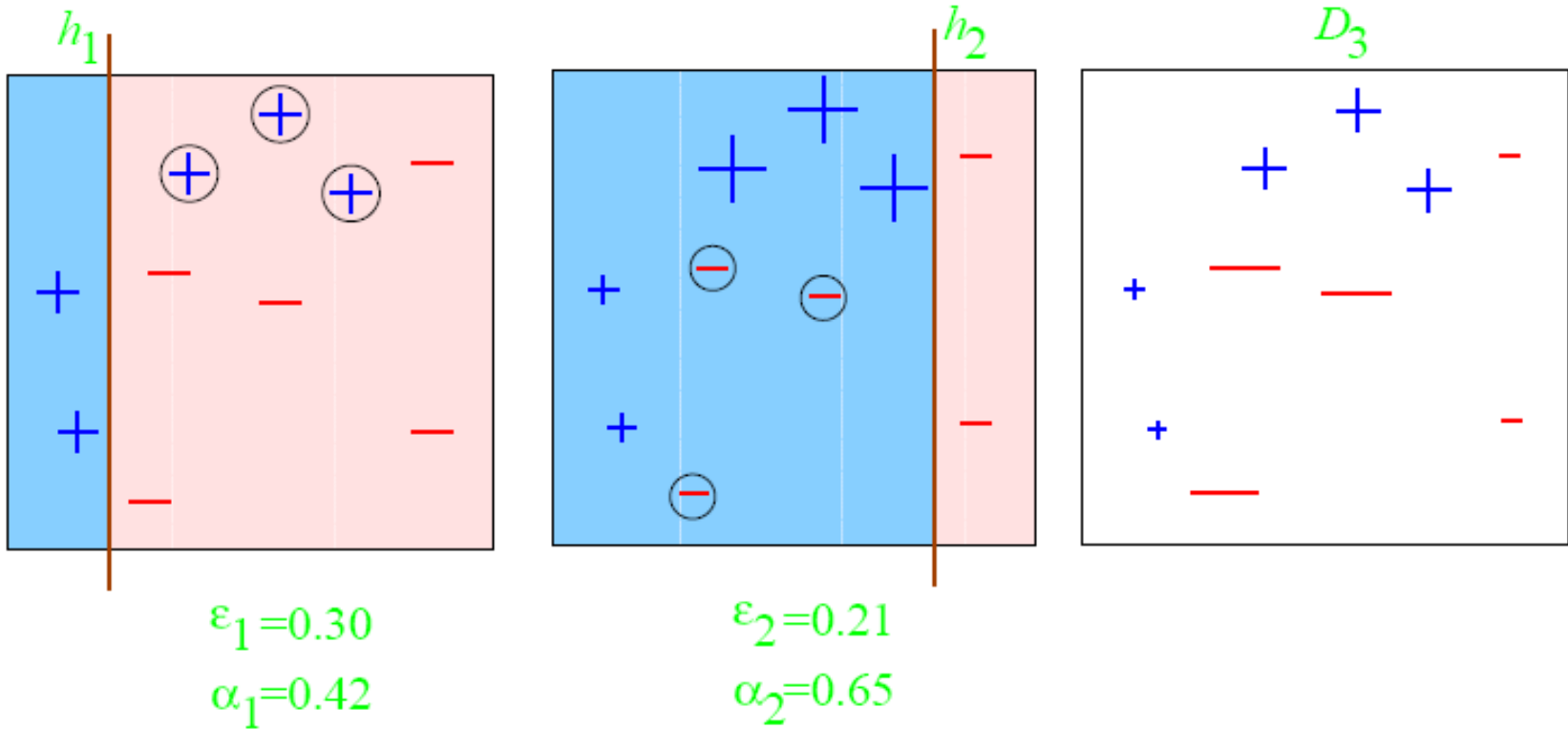
Round 1

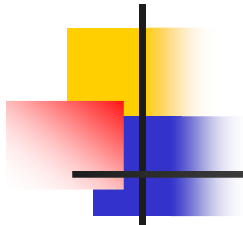


$\epsilon_1=0.30$
 $\alpha_1=0.42$

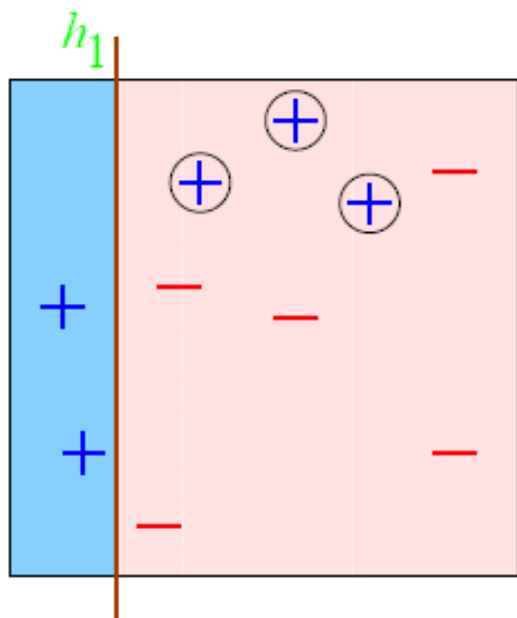


Round 2

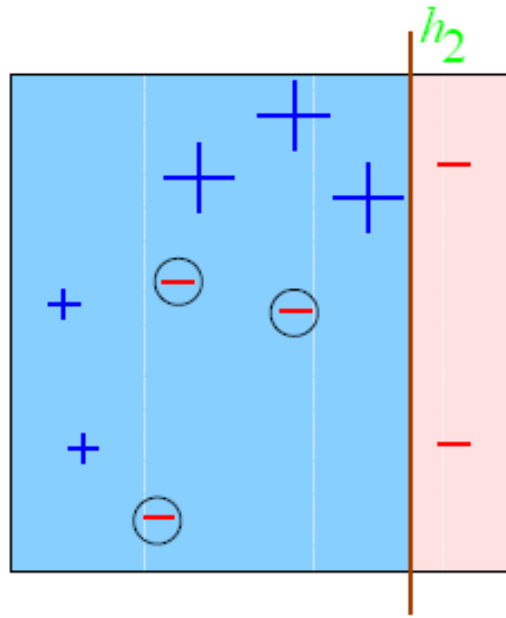




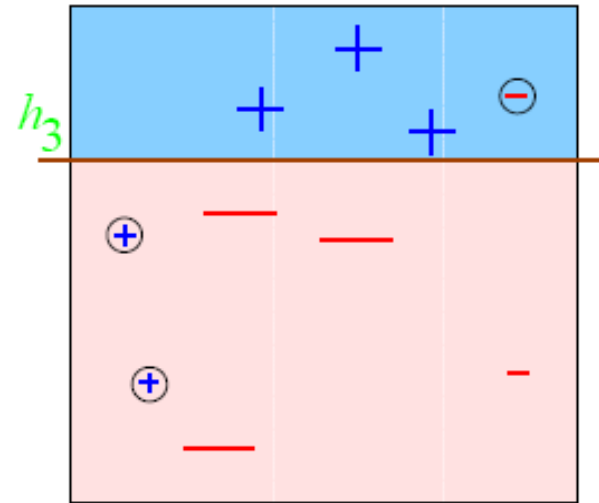
Round 3



$$\begin{aligned}\epsilon_1 &= 0.30 \\ \alpha_1 &= 0.42\end{aligned}$$

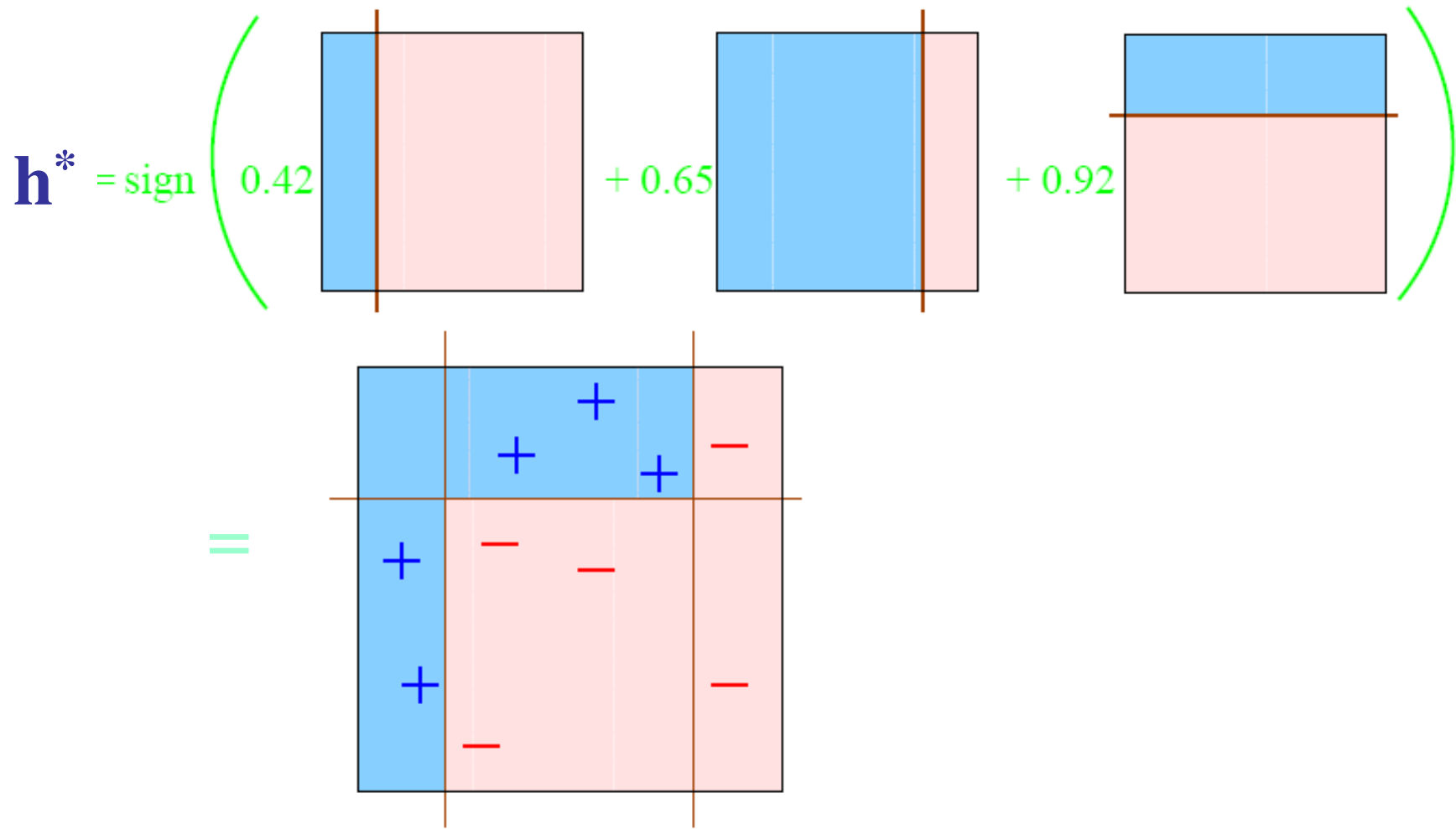


$$\begin{aligned}\epsilon_2 &= 0.21 \\ \alpha_2 &= 0.65\end{aligned}$$



$$\begin{aligned}\epsilon_3 &= 0.14 \\ \alpha_3 &= 0.92\end{aligned}$$

Final Classifier





Learn from weighted instances?

- How can a learning alg use distribution D ?

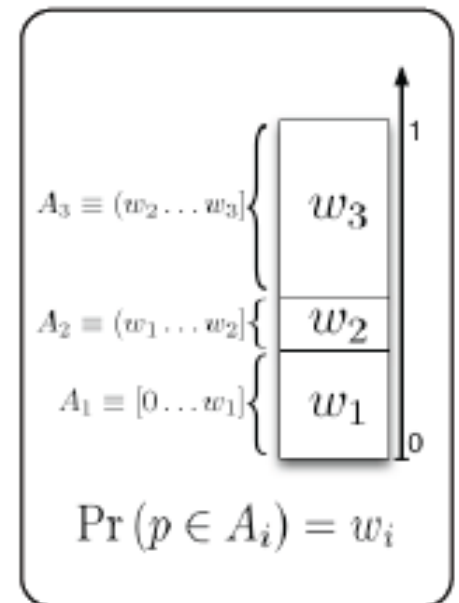
1. Reweighting

- Can modify many learning algorithms to deal with weighted instances:
 - ID3:
 - entropy, information-gain equations use COUNTs $\#(X=3, C=+)$
... assumes all weights=1
 - Modify to use *weight* of each instance
 - Naïve Bayes: ditto
 - k-NN: multiple vote from an instance by its weight

Learn from weighted instances?

Resampling

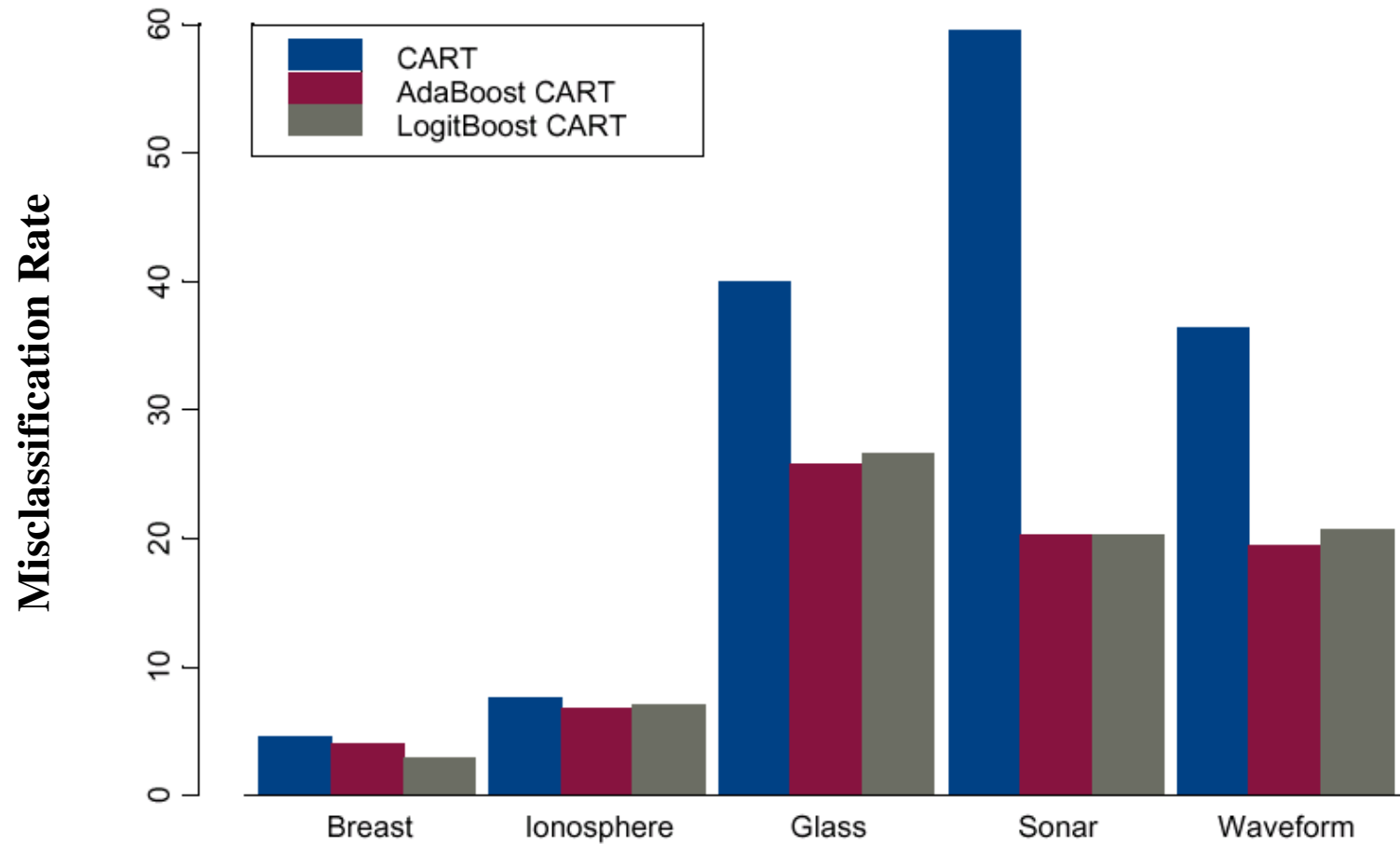
- Given dataset S and distribution D , produce new dataset S' that embodies D
 - Stochastically
 - Using weight ratio ...
- How many?
 - More is good...
 - Typically $|S'| = |S|$
- If possible, use Re-weighting
 - Re-sampling is only an approximation



Stochastic Resampling...

- Let S' be the empty set
- Let $D = (w_1, \dots, w_n)$ be the weights of examples in S
 - $w_i = D(i)$ corresponds to example x_i
- While not-enough-samples
 - Draw $n \in [0..1]$ according to $U(0,1)$
 - $S' \leftarrow S' \cup \{x_k\}$ where k is such that $\sum_{i=1}^{k-1} w_i < n \leq \sum_{i=1}^k w_i$
- return S'

Comparison





Analyzing the Training Error

Theorem:

- Let $\gamma_t = 1/2 - \varepsilon_t$
- $\text{training_error}(h^*) \leq \exp(-2 \sum_t \gamma_t^2)$

- If $\forall t : \gamma_t \geq \gamma > 0$
then $\text{training_error}(h^*) \leq \exp(-2\gamma^2 T)$
- *AdaBoost* is adaptive:
 - does not need to know γ or T a priori
 - can exploit $\gamma_t \gg \gamma$



Proof

- $f(x) = \sum_t \alpha_t h_t(x) \Rightarrow h^*(x) = \text{sign}(f(x))$

- Step 1: unwrapping recurrence:

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

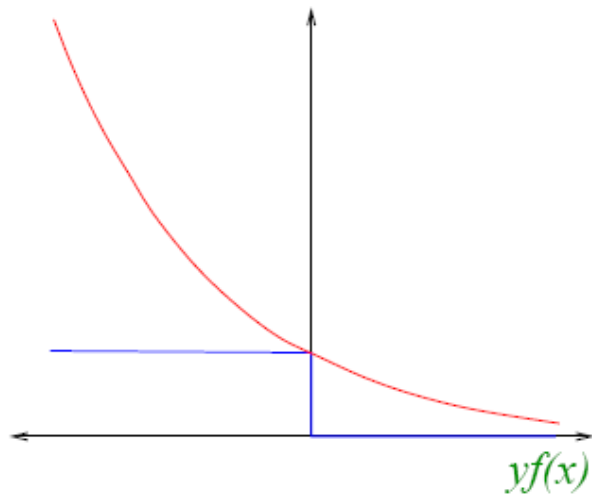
$$D_{\text{final}}(i) = \underbrace{\frac{1}{m}}_{D_1} \frac{\exp\left(-y_i \sum_t \alpha_t h_t(x_i)\right)}{\prod_t Z_t} = \frac{1}{m} \frac{\exp(-y_i f(x_i))}{\prod_t Z_t}$$

Proof (II)

$$D_{\text{final}}(i) = \frac{1}{m} \frac{\exp(-y_i f(x_i))}{\prod_t Z_t}$$

- Step 2: $\text{training_error}(h^*) \leq \prod_t Z_t$

- Proof: $\text{training_error}(h^*) = \frac{1}{m} \sum_i \begin{cases} 1 & \text{if } y_i \neq h^*(x_i) \\ 0 & \text{else} \end{cases}$



$$= \frac{1}{m} \sum_i \begin{cases} 1 & \text{if } y_i f(x_i) \leq 0 \\ 0 & \text{else} \end{cases}$$

$$\leq \frac{1}{m} \sum_i \exp(-y_i f(x_i))$$

$$= \sum_i D_{\text{final}}(i) \prod_t Z_t$$

$$= \prod_t Z_t$$



Proof (III)

- *Step 3:* $Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$

- Proof:
$$\begin{aligned} Z_t &= \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i)) \\ &= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\ &= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t} \\ &= 2\sqrt{\epsilon_t(1 - \epsilon_t)} \end{aligned}$$

$$\epsilon_t = \sum_{i:y_i \neq h_t(x_i)} D_t(i)$$

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



Proof (IV)

- Step 4: $2\sqrt{\varepsilon(1-\varepsilon)} \leq \exp(-2\gamma^2)$

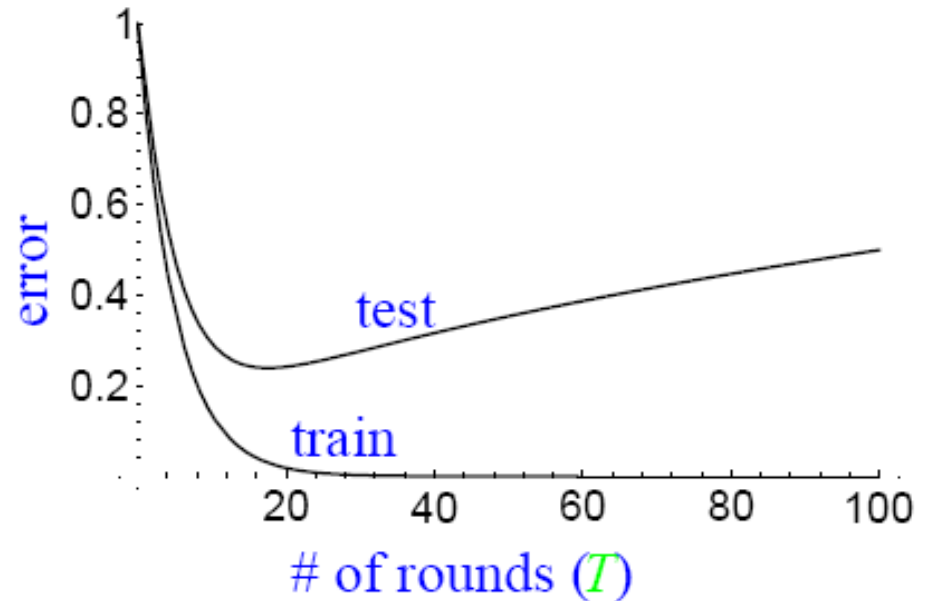
- $2\sqrt{\varepsilon(1-\varepsilon)} = \sqrt{4\left(\frac{1}{2}-\gamma\right)\left(1-\left(\frac{1}{2}-\gamma\right)\right)} = \sqrt{1-4\gamma^2}$

- Suffices to show $\forall 0 \leq a \leq \frac{1}{4} \quad \sqrt{1-4a} \leq e^{-2a}$

- True if, for all $a \in [0, \frac{1}{4}]$
 $g(a) = (1 - 4a) - e^{-4a} \leq 0$

- $g(0) = 1 - 0 - e^0 = 0$
 $g'(a) = -4 - (-4)e^{-4a} = 4(e^{-4a} - 1) \leq 0$

How Will Test Error Behave? (A First Guess)

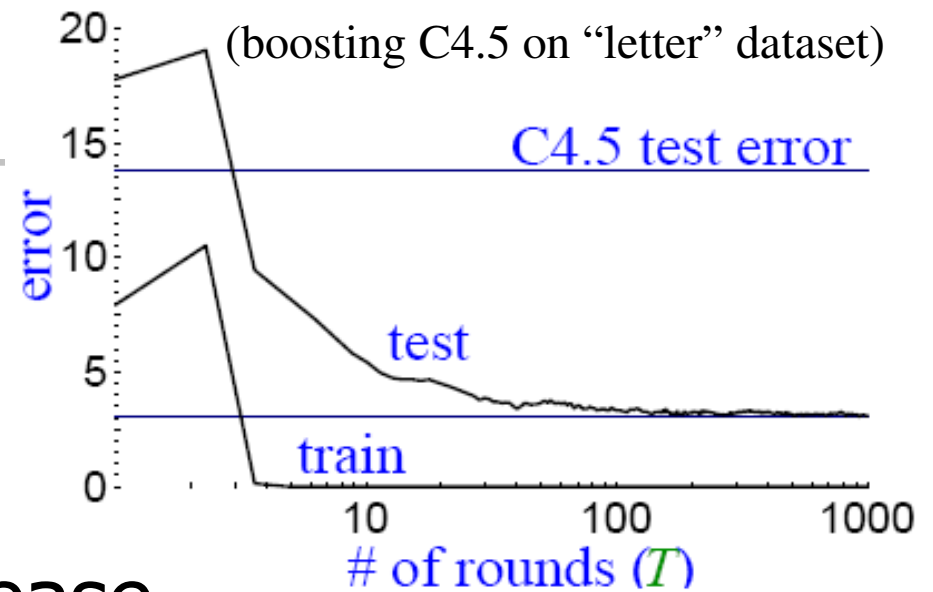


Expect...

- training error to continue to drop (or reach 0)
- test error to increase when h^* becomes "too complex"
 - "Occam's razor"
 - overfitting
 - hard to know when to stop training

Actual Typical Run

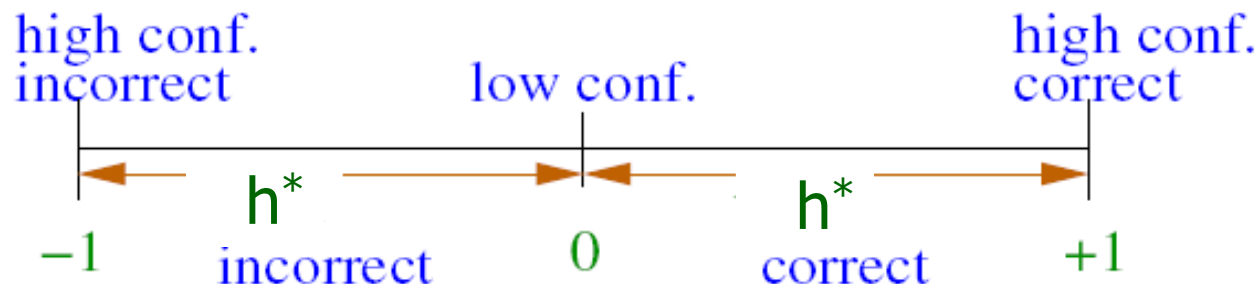
	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1



- test error does not increase, even after 1000 rounds
 - (total size $> 2,000,000$ nodes)
- test error continues to drop, even after training error is 0!
- Occam's razor: "simpler rule is better"... appears to not apply!

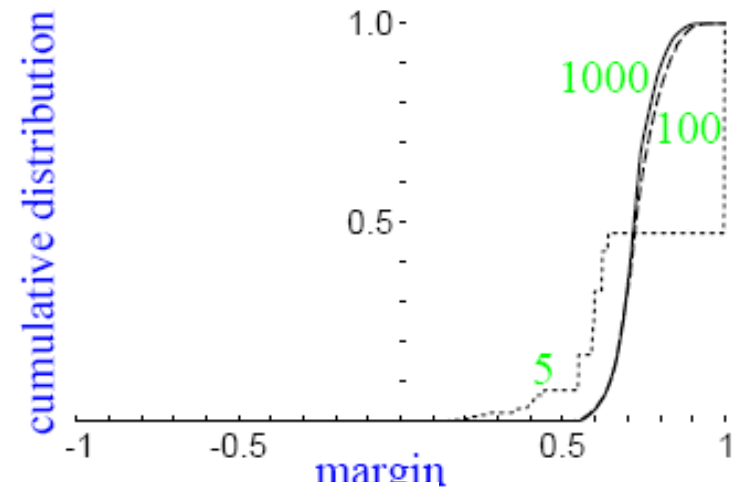
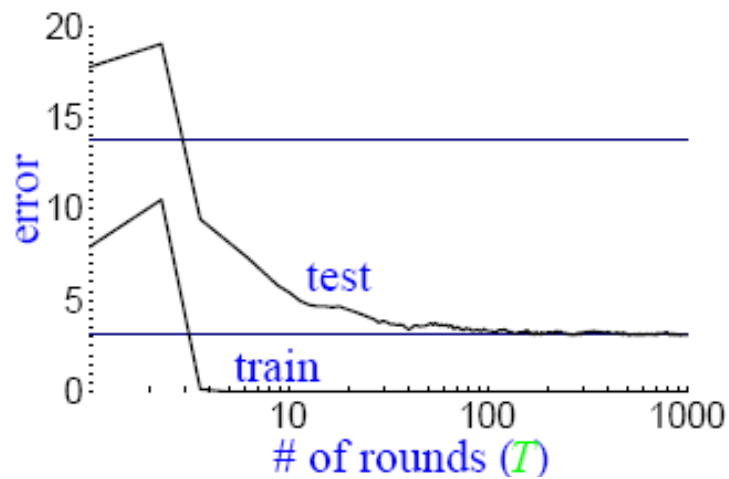
A Better Story: ... using Margins

- key idea:
 - training error only measures whether classifications are **right** or **wrong**
 - should also consider **confidence** of classifications
- h^* is weighted majority vote of weak classifiers
- measure confidence by **margin**
 - = strength of the vote
 - = (weighted fraction voting correctly)
 - (weighted fraction voting incorrectly)



Empirical: Margin Distribution

- margin distribution
= cumulative distribution of margins of training ex's



	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem:

Large margins \Rightarrow better bound on generalization error

- (independent of # of rounds \approx complexity of h^*)
- proof idea: if all margins are large, then can approximate final classifier h^* by a much smaller classifier (just as polls can predict not-too-close election)

- Theorem:

Boosting tends to increase margins of training examples

- (given weak learning assumption)
- proof idea: similar to training error proof

- SO:

although final classifier h^* is getting larger, margins are likely to be increasing,

so final classifier h^* actually getting close to a simpler classifier, driving down the test error



More Technically...

- with high probability, $\forall \theta > 0$:

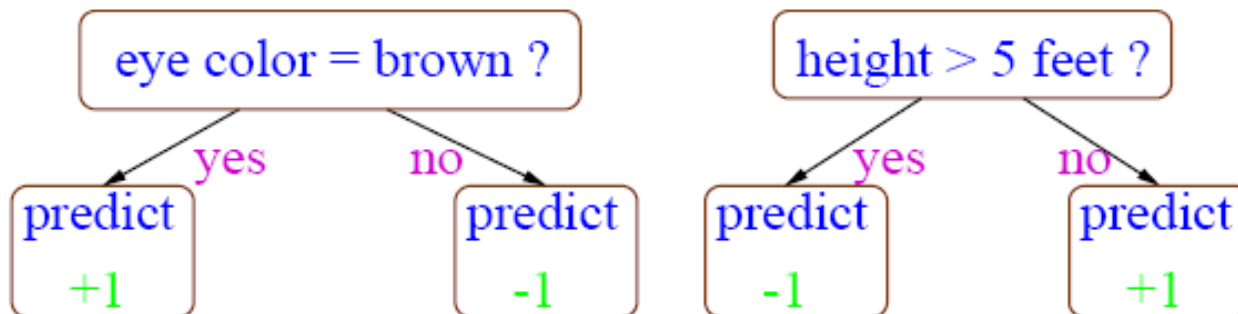
$$\text{generalization error} \leq \hat{\text{Pr}}[\text{margin} \leq \theta] + \tilde{O}\left(\frac{\sqrt{d/m}}{\theta}\right)$$

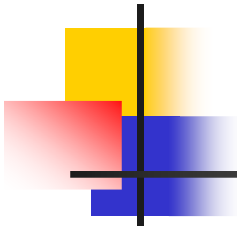
($\text{Pr}[\]$ = empirical probability)

- bound depends on
 - m = # training examples
 - d = “complexity” of weak classifiers
 - entire distribution of margins of training examples
- $\text{Pr}[\text{margin} \leq \theta] \rightarrow 0$ exponentially fast (in T)
if (error of h_t on \mathbf{D}_t) $< 1/2 - \theta$ ($\forall t$)
 - so: if weak learning assumption holds,
then all examples will quickly have “large” margins

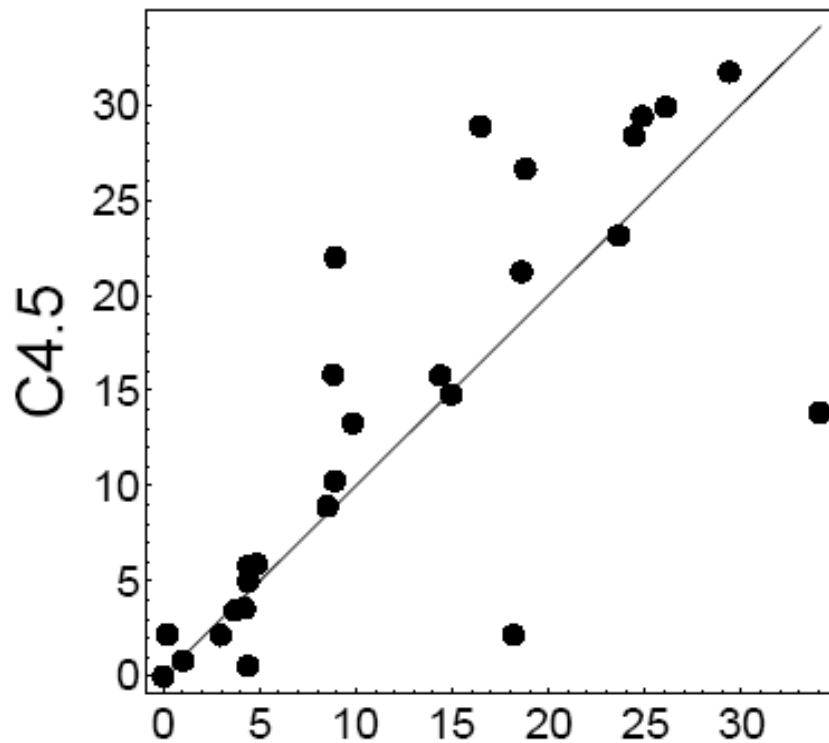
UCI Experiments

- tested AdaBoost on UCI benchmarks
- used:
 - C4.5 (Quinlan's decision tree algorithm)
 - "decision stumps": very simple rules of thumb that test on single attributes

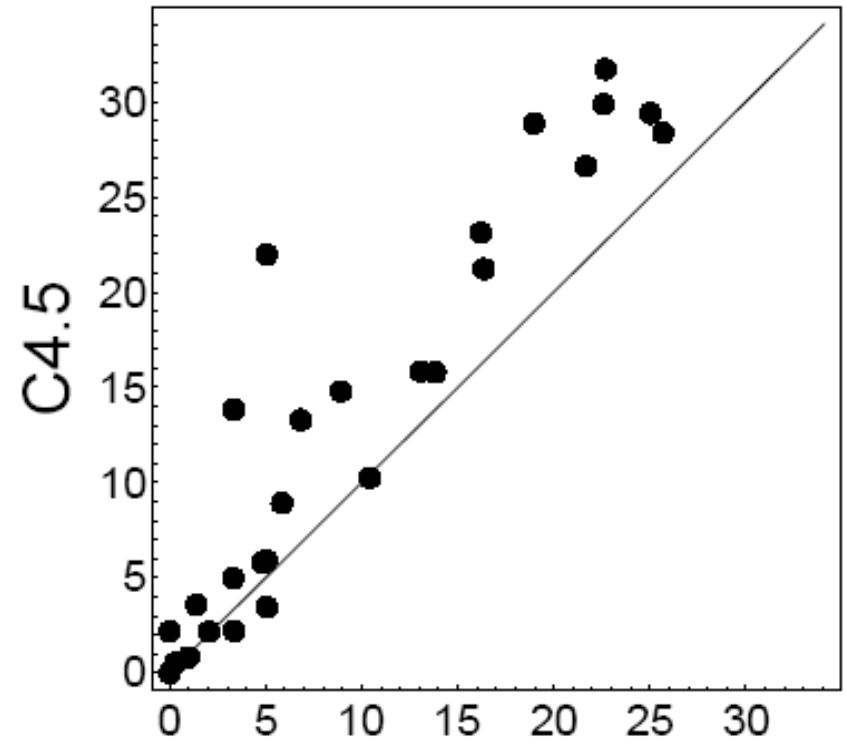




UCI Results



boosting Stumps



boosting C4.5



Multiclass Problems

- $y \in Y = \{1, \dots, k\}$ $h_t : X \rightarrow Y$

direct approach
(AdaBoost.M1):

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$h^*(x) = \arg \max_{y \in Y} \sum_{t: h_t(x)=y} \alpha_t$$

- can prove same bound on error if $\forall t : \epsilon_t \leq 1/2$
 - in practice, not usually a problem for “strong” weak learners (e.g., C4.5)
 - significant problem for “weak” weak learners (e.g., decision stumps)
- instead, reduce to binary...

Reducing Multiclass to Binary

- If labels = $\{a, b, c, d, e\}$
- replace each training example by five $\{-1, +1\}$ -labeled examples:

$$x, c \rightarrow \begin{cases} (x, a), & -1 \\ (x, b), & -1 \\ (x, c), & +1 \\ (x, d), & -1 \\ (x, e), & -1 \end{cases}$$

- predict with label receiving most (weighted) votes



AdaBoost.MH

- can prove:

$$\text{training error}(\mathbf{h}^*) \leq \frac{k}{2} \cdot \prod Z_t$$

- reflects fact that small number of errors in binary predictors can cause overall prediction to be incorrect
- extends immediately to multi-label case
 - (more than one correct label per example)



Other Uses of Boosting

- Output code
 - [Schapire, Allwein & Singer] [Dietterich & Bakiri]
- Ranking problems
 - [Schapire, Freund, Iyer & Singer]
- Confidence-rated predictions
 - [Schapire & Singer]
- Face Detection
 - [Viola & Jones]
- Active Learning
 - [Lewis & Gale] [Abe & Mamitsuka]
- Applications:
 - Text Categorization [Schapire & Singer]
 - Human-computer Spoken Dialogue [Schapire, Rahim, Di Fabrizio, Dutton, Gupta, Hollister & Riccardi]

Application: Detecting Faces

- **problem**: find faces in photograph or movie
- **weak classifiers**: detect light/dark rectangles in image



- many clever tricks to make extremely fast and accurate



Practical Advantages of *AdaBoost*

- fast
- simple and easy to program
- no parameters to tune (except T , sometimes)
- flexible — can combine with any learning algorithm
- no prior knowledge needed about weak learner
- provably effective, given weak classifier
 - → shift in mind set — goal now is merely to find classifiers barely better than random guessing
- versatile
 - can use with data that is textual, numeric, discrete, etc.
 - has been extended to learning problems well beyond binary classification



Caveats

- performance of *AdaBoost* depends on **data** and **weak learner**
- consistent with theory, AdaBoost can **fail** if...
 - weak classifiers too complex
→ overfitting
 - weak classifiers too weak ($\gamma_t \rightarrow 0$ too quickly)
→ underfitting
→ low margins → overfitting
- empirically, *AdaBoost* seems especially susceptible to uniform noise



Conclusions wrt Boosting

- **boosting is a practical tool** for classification and other learning problems
 - grounded in rich theory
 - performs well experimentally
 - often (but not always!) resistant to overfitting
 - many applications and extensions
- **many ways** to think about boosting
 - none is entirely satisfactory by itself, but each useful in its own way
 - considerable room for further theoretical and experimental work



Conclusions wrt Boosting

Boosting is a practical tool for classification and other learning problems

- grounded in rich theory
- performs well experimentally
- often (but not always!) resistant to overfitting
- many applications and extensions



Effect of Boosting

- In the early iterations, boosting primarily *reduces bias*
- In later iterations, boosting primarily *reduces variance* (apparently)

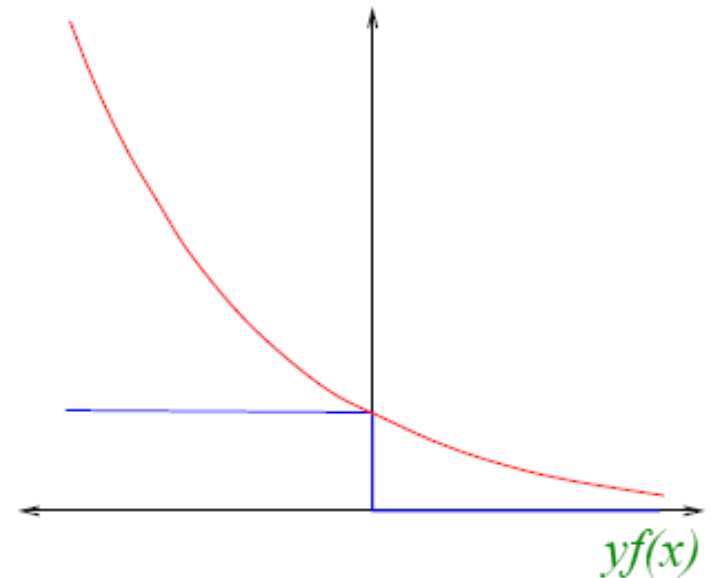
Exponential Loss

- *AdaBoost* minimizes “exponential loss”...

$$\prod_t Z_t = \frac{1}{m} \sum_i \exp(-y_i f(x_i))$$

where $f(\mathbf{x}) = \sum_t \alpha_t h_t(\mathbf{x})$

- exponential loss is an upper bound on 0-1 (classification) loss
- On each round, *AdaBoost* greedily chooses α_t and h_t to minimize loss





Coordinate Descent

- $\{g_1, \dots, g_N\}$ = all weak classifiers
- want to find $\lambda_1, \dots, \lambda_N$ to minimize

$$L(\lambda_1, \dots, \lambda_N) = \sum_i \exp \left(-y_i \sum_j \lambda_j g_j(x_i) \right)$$

- AdaBoost uses coordinate descent:
 - initialize all $\lambda_j = 0$
 - each round:
 - choose one coordinate λ_t (corresponding to h_t) and
 - update (increment by α_t)
 - choose update causing biggest decrease in loss
- (powerful technique for minimizing over huge space of functions)



Types of Ensemble Methods

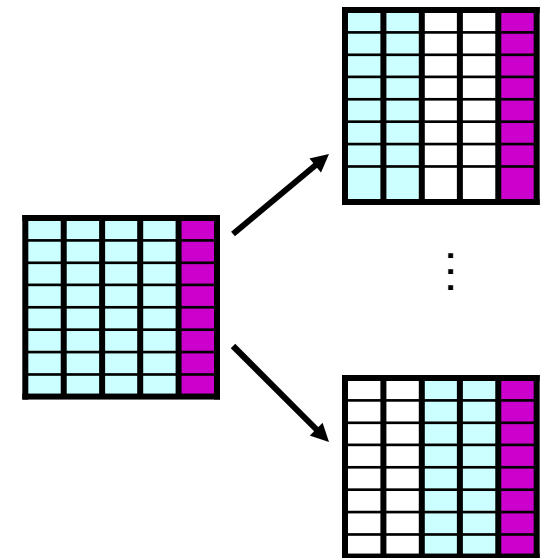
1. Subsample Training Sample
 2. Manipulate Input Features
 3. Manipulate Output Targets
 4. Injecting Randomness
 5. Algorithm Specific methods
- Other combinations
 - Why do Ensembles work?

2: Manipulate INPUT FEATURES

- Different learners see different subsets of features (of each of the training instances)
- Eg: 119 features for classing volcanoes on Venus
 - Divide into 8 disjoint subsets (by hand)...
 - and use 4 networks for each
 - ... 32 NN classifiers

Did VERY well [Cherkauer'96]

- Tried w/sonar dataset – 25 input features
Did NOT work [Tumer/Ghost'96]
- Technique works best when input features highly redundant





3: Manipulate OUTPUT Targets

Spse K outputs $Y = \{ y_1, \dots, y_K \}$

a. Could learn 1 classifier, into Y ($|Y|$ values)

b. Or could learn K binary classifiers:

- y_1 vs $Y - y_1$
- y_2 vs $Y - y_2$
- ...

then vote.

c. Build $\ln K$ binary classifiers

- h_i specifies i^{th} bit of index $\in \{1, 2, \dots, K\}$
- Each h_i sub-classifier splits output-values into 2 subsets
 - $h_0(x)$ is 1 if " y_1, \dots, y_8 "; else 0
 - $h_1(x)$ is 1 if " $y_1 - y_4; y_9 - y_{12}$ "; else 0
 - $h_2(x)$ is 1 if " $y_1, y_2; y_5, y_6; y_9, y_{10}; y_{13}, y_{14}$ "; else 0
 - ...



Error Correcting Output Code

- Why not $> \ln K$ binary classifiers . . .
 - “Error-Correcting Codes” (some redundancy)
 - [Dietterich/Bakiri'95]
- View $[h_1(x), \dots, h_m(x)]$ as code-word;
return label y_i with nearest codeword
- Better: can combine with AdaBoost
 - [Schapire'97]

Recognizing Handwritten Number

7 4 3 5 2 → 7, 4, 3, 5, 2

- “Obvious” approach: learn $F: \text{Scribble} \rightarrow \{0,1,2,\dots,9\}$
 - ...doesn't work very well (too hard!)
- Or... “decompose” the learning task into 6 “subproblems”

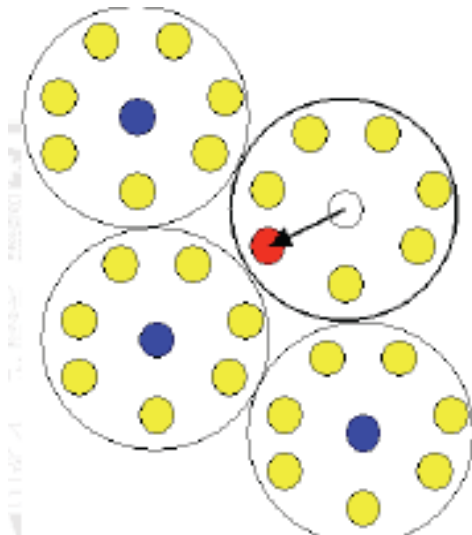
Class	Code Word					
	vl	hl	dl	cc	ol	or
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	0	1	0	0
9	0	0	1	1	0	0

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 6 classifiers, one for each “sub-problem”
2. to classify a new scribble:
 - Run each classifier
 - 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, map each n -bit message into an m -bit code ($m > n$)
 - note $|\text{codes}| \gg |\text{messages}|$
- To “decode” a msg, translate it to message corresponding to the “nearest” code
 - (Hamming distance)
- 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

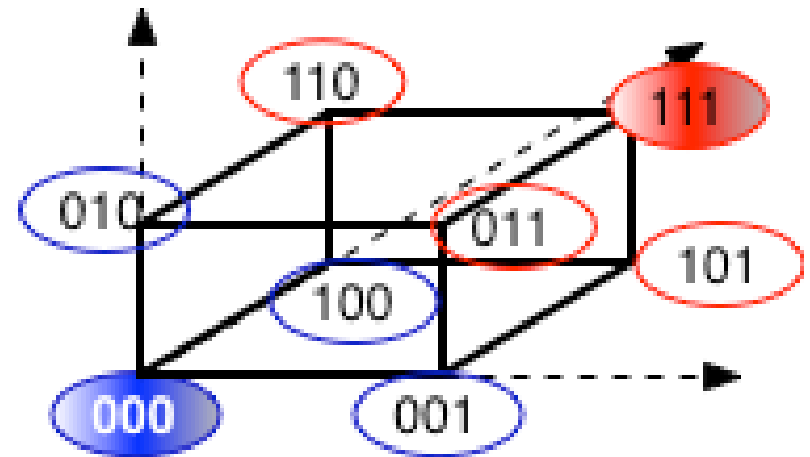


blue = message (n bits)
yellow = code (m bits)

white = intended message
red = received code

Error Correcting Code

- Use **3** bits to encode **2** possible messages
- Codewords {**000**, **111**}
- As differ in >2 places, can detect and correct any "single digit" error!



Code-words for ECOC learning

- Coding: k labels \rightarrow m bit codewords
- Good coding:
 - 1. row separation: want “assigned” codes separated by lots of “unassigned” codes
 - 2. column separation: each bit i of the codes should be uncorrelated with all other bits j

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

rows correlated: BAD!

columns correlated: BAD!

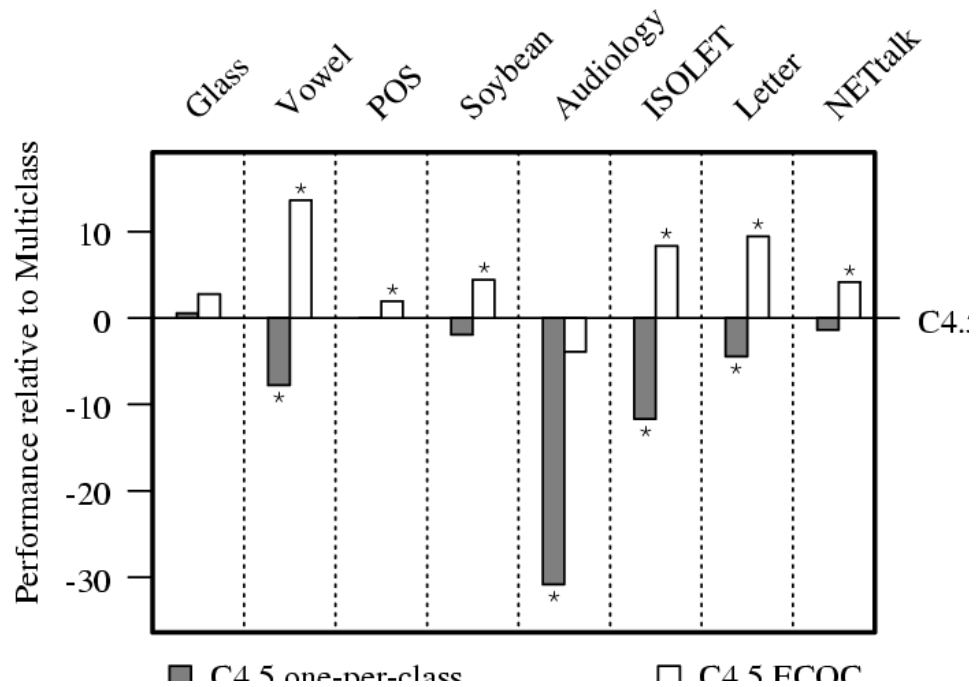


Finding Good Codes

- Lots of tricks...
- Simple approach:
select the codewords *at random*.
- if $2^m \gg k$, then obtain a “good” code with high probability
 - such codes work well in practice

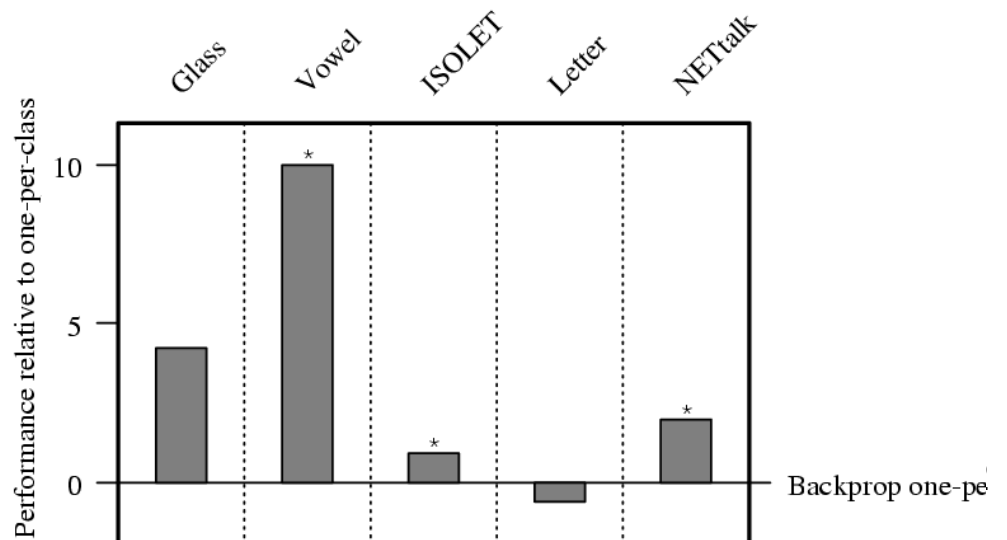


Results



% decrease in error of ECOC over an ID3-like learning algorithm

(% decrease in error of ECOC over a neural network learner





Types of Ensemble Methods

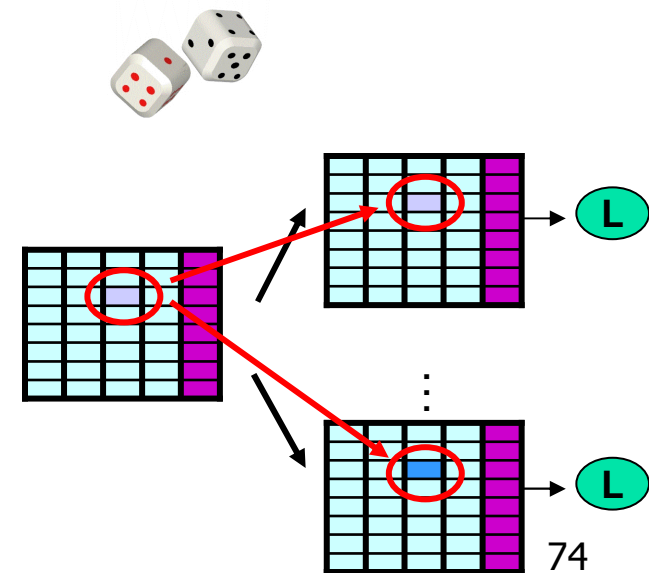
1. Subsample Training Sample
2. Manipulate Input Features
3. Manipulate Output Targets
4. Injecting Randomness
 - Data
 - Learner
5. Algorithm Specific methods
 - Other combinations
 - Why do Ensembles work?

4a: Injecting Randomness to Data

Add **0-mean Gaussian noise** to input features

Draw w/replacement from original data,
but add noise

- For Neural Nets:
 - Large improvement on
 - + synthetic benchmark;
 - + medical Dx
 - [Raviv/Intrator'96]



4b: Injecting Randomness to Learner

- For Neural Nets:

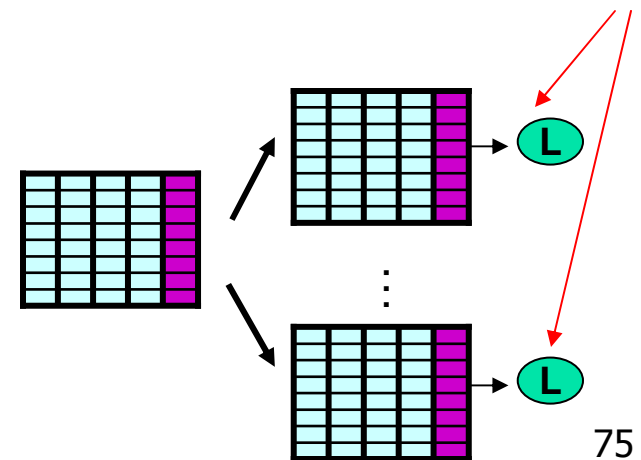
... Different **random initial values** of weights

But really independent?

Empirical test: [Pamanto, Munro, Doyle 1996]

Cross-validated committees BEST,

then Bagging, then Random initial weights





Randomness – w/ C4.5

- **C4.5** uses Info Gain to decide which attribute to split on
 - Why not consider top 20 attributes; choose one at random?
 - ⇒ Produce 200 classifiers (same data)
 - To classify new instance: Vote
 - Empirical test: [Dietterich/Kong 1995]
Random better than bagging, than single C4.5
- **FOIL** (for learning Prolog-like rules)
 - Chose any test whose info gain within 80% of top
 - Ensemble of 11
STATISTICALLY BETTER
than 1 run of FOIL [Ali/Pazzani'96]



5: Algorithm Specific (NNs)

Seek “diverse” population of NNs

- Simultaneously train several NN's with **penalty for correlations**.

Backprop minimizes error function =

sum of MSE and correlations [Rosen'96]

- Use operators to build new structures; keep R “best”

- **DIVERSITY + ACCURACY**

(like GA [Opitz/Shavlik'96])

- Give different NNs **different auxiliary tasks**

- (eg, predict one input feature)

in addition to primary task

Backprop use BOTH in error, so produces different nets

[Abu-Mostafa'90; Caruana'96]

- For each $[x_i, y_i]$, re-train NN_j with

- $[x_i, [y_i, 1]]$ if $NN_j(x_i)$ closest to y_i

- $[x_i, [y_i, 0]]$ otherwise

(So diff NNs get **different training values**, to help NN learn where it performs best) [Munro/Parmanto'97]



Algorithm Specific (NN #2)

- Person identifies which region of input space
 - (Highway, 2lane-road, dirt-road, ...)
- Train NN_i for region _{i} ... eg, to steer, . . .
- Each NN_i also learns to reconstruct image
 - Same intermediate layer!
- When “running”, each NN_i
 - proposes steering direction,
 - reconstructs of image
- Take direction from NN_i with best reconstruction [Pomerleau]
- Also: train on “bad” situation, by distorting image, and defining correct label



Algorithm Specific (DTs, ...)

- “Option tree”:
 - Decision Tree whose internal nodes have > 1 splits, each producing own sub-decision-tree
 - (Eval: go down each, then vote) [Buntine'90]
- Empirical: accuracy \approx bagged C4.5 trees but MUCH more understandable
- Can try different modalities, but not clear how DIVERSE they will be
 - Use cross-validation to check for both accuracy and diversity



Combining Classifier: Linear

Linear Combination

- Unweighted: Bagging, ErrorCorrecting, Boosted (weighted)

■ Bayesian Model

If each h_t produces class prob. estimates

$$P(f(x) = y \mid h_t)$$

should use:

$$P(f(x) = y) = \sum_t P(f(x) = y \mid h_t) P(h_t)$$

- Forecasting lit. suggests this is very robust [Clemen'89]

■ Variance-based

- Use least squares regression to find weights that max accuracy on training data
- Uncorrelated $\Rightarrow h_t$'s weight $\propto 1/\text{Var}(h_t)$
Can also deal w/ less correlated subset



Combining Classifiers: Linear, II

Linear Combination (con't)

- **Gating** [Jordan/Jacobs'94]
 - Learn classifier's $\{ h_1, \dots, h_T \}$
 - $\text{output}(\mathbf{x}) = \sum_t w_t h_t(\mathbf{x})$
 - $w_t(\mathbf{x}) = \exp(\mathbf{v}_t \mathbf{x}) / \sum_u \exp(\mathbf{v}_u \mathbf{x})$
 - Problem: lot of parameters to learn: $\{\mathbf{v}_u\}$, as well as params for all h_t 's
- **Cross-Validation** [Ali/Pazzani'96; Buntine'90]
 - Obtain weights from performance on hold-out set

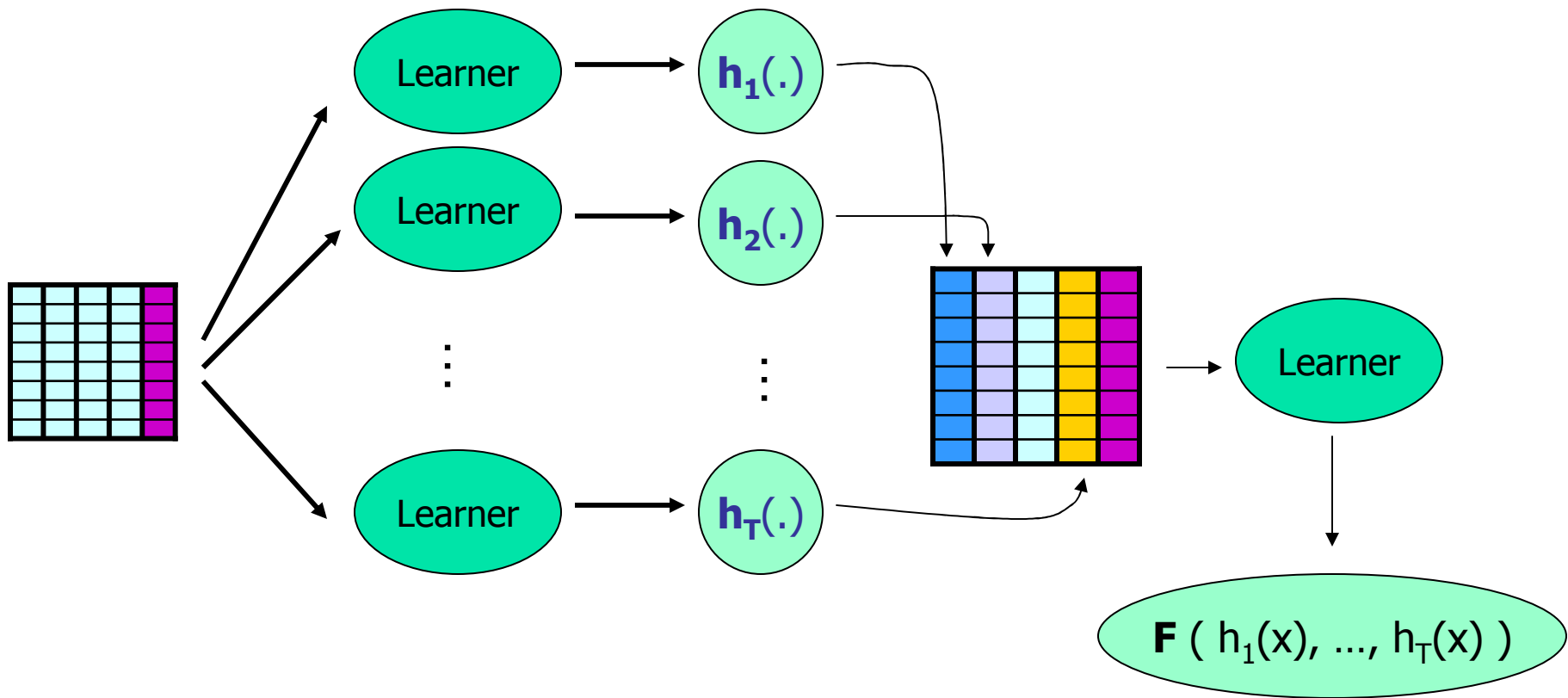


Combining Classifiers: NonLinear

Stacking [Wolper'92; Breiman'96]

- Given learners $\{ L_i(\cdot) \}$, obtain $h_i = L_i(S)$
- Want classifier $h^*(x) = F(h_1(x), \dots, h_T(x))$
- Let $h_t^{(-i)} = L_t(S - \mathbf{x}_i)$ be classifier learned using L_t , on all but instance \mathbf{x}_i
 - ... so $T \times |S|$ classifiers
- Let $\hat{y}_i^{(t)} = h_t^{(-i)}(\mathbf{x}_i)$
- Now learn $F(\dots)$ from $\{ [[\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \dots, \hat{y}_i^{(T)}], y_i] \}_i$

Stacking





Why do ensemble work?

Many reasons justify ensemble approach:


- Bias/Variance decomposition
- A(nother) statistical motivation
- Representational issues
- Computational issues



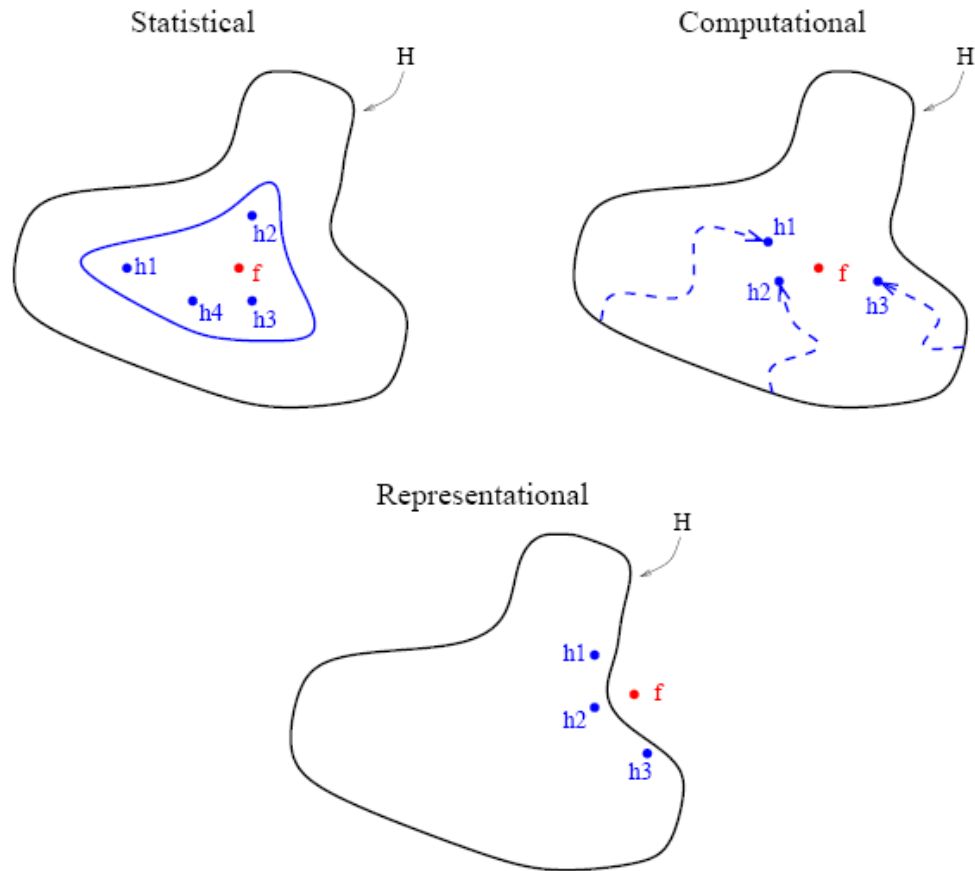
Why do ensembles work? (AdaBoost)

- **Empirical** evidence suggests that *AdaBoost* reduces both **bias** and **variance** part of the error
 - bias is mostly reduced in early iterations
 - while variance in later ones

Lesson learned?

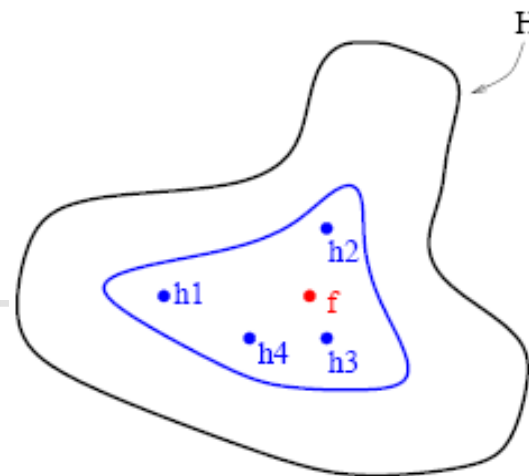
- **Use Bagging with low bias and high variance classifiers**
 - e.g., decision trees, 1-nn, ...
- **Always try AdaBoost** 
 - Typically produces excellent results.
 - Works especially well with very simple learners
 - eg, decision stumps

Other explanations?



[T. G. Dietterich. *Ensemble methods in machine learning*.
Lecture Notes in Computer Science, 1857:1–15, 2000.]

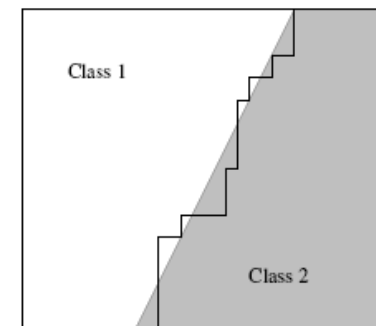
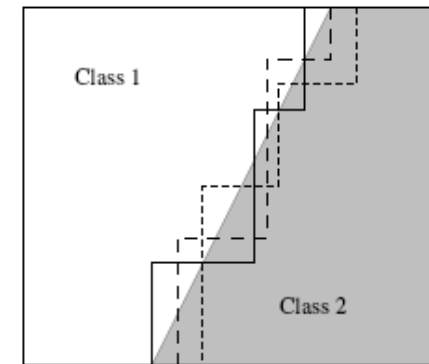
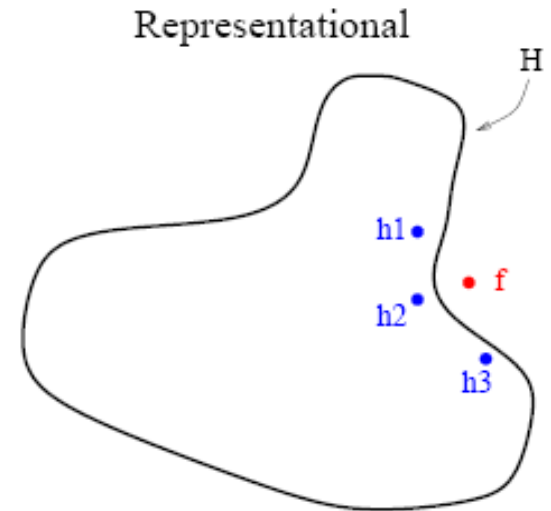
1. Statistical



- Given a finite amount of data, many hypothesis are typically equally good. How can the learning algorithm select among them?
- **Optimal Bayes classifier recipe:**
 - take a *weighted* majority vote of *all* hypotheses,
 - weighted by their posterior probability
 - ...**provably** the best possible classifier
- Ensemble learning \approx approximation of the Optimal Bayes rule

2. Representational

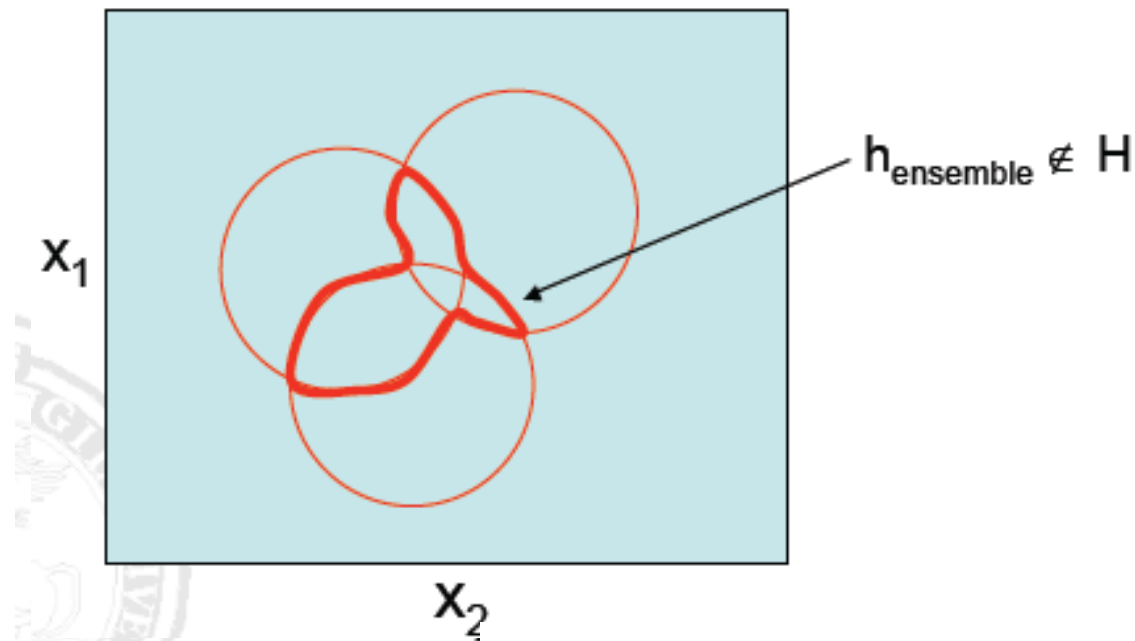
- Optimal target function may not be ANY individual classifier, but may be (approximated by) ensemble averaging
- Eg... a decision trees
 - boundaries are axis-parallel hyperplanes
 - By averaging a large number of such “staircases”, can approximate diagonal decision boundary with arbitrarily good accuracy



Representational (example 2)

- Space: $[0,1] \times [0,1]$
- Hypothesis space H of “discs”

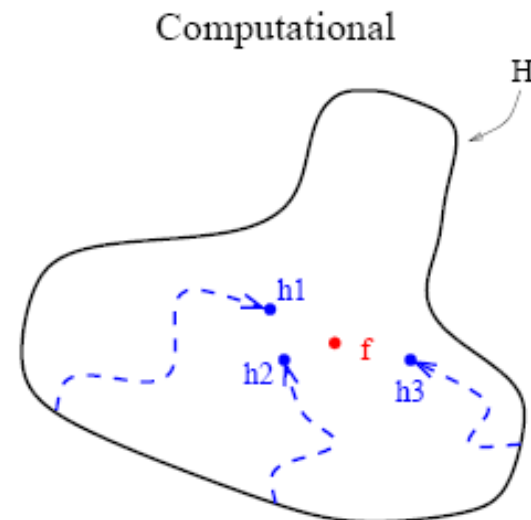
$h_1, h_2, h_3 \in H$



h_{ensemble} cannot be returned by “base” learner,
but h_{ensemble} can be returned by ensemble

3. Computational

- Essentially all learning alg's search through some space of hypotheses to find one that is "good enough" for the given training data
- As many interesting hypothesis spaces are huge/infinite, heuristic search is essential
 - (eg ID3 greedily search in space of decision trees)
- Learner might get stuck in a local minimum
- One strategy for avoiding local minima: repeat the search many times with random restarts
→ bagging





Effect of Bagging

- If bootstrap replicate approx'n is 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 V_u)
 - For high-variance classifiers, it can reduce variance



Summary of Ensembles

- Ensembles: basic motivation
creating a **committee of experts** is typically more effective than creating a single **supergenius**
- Key issues:
 - Generating base models
 - Integrating responses from base models
- Popular ensemble techniques
 - manipulate training data: bagging and boosting
(ensemble of "experts", each specializing on different portions of the instance space)
 - manipulate output values: error-correcting output coding
(ensemble of "experts", each predicting 1 bit of the {multibit} full class label)
- Why does ensemble learning work?

Comparison of Ensemble Methods

Pairwise comparison of 4 ensemble methods (W-L-T)

	C4.5	ADABOOST C4.5	Bagged C4.5
Random C4.5	14 - 0 - 19	1 - 7 - 25	6 - 3 - 24
Bagged C4.5	11 - 0 - 22	1 - 8 - 24	
ADABOOST C4.5	17 - 0 - 16		

	C4.5	ADABOOST C4.5	Bagged C4.5
Random C4.5	5 - 2 - 2	5 - 0 - 4	0 - 2 - 7
Bagged C4.5	7 - 0 - 2	6 - 0 - 3	
ADABOOST C4.5	3 - 6 - 0		

... added 20% synthetic class label noise

[T. G. Dietterich. *Ensemble methods in machine learning*.
Lecture Notes in Computer Science, 1857:1–15, 2000.]



Specific Problems

- AdaBoost is good way to construct ensemble of DTs
 - But if data noisy: AdaBoost places high weight on incorrectly-labeled data
- \Rightarrow constructs bad classifier
- ErrorCorrected Output does not work well with local algs (like nearest neighbor)
- ? Combination of Ensemble methods

**Ensemble
Methods**

×

**Learning
Algorithms**

- General Problem:
 - lots of memory to store ensemble 200 DTs: 59M !
 - how to interpret
 - (one DT easy to understand; but 200 of them?)
 - CPU time