

10701

Machine Learning

Boosting

Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
 - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
 - Low variance, don't usually overfit
- **Simple (a.k.a. weak) learners are bad**
 - High bias, can't solve hard learning problems
- Can we make all weak learners always good???
 - **No!!!**
 - **But often yes...**

Simplest approach: A “bucket of models”

- Input:
 - your top T favorite learners (or tunings)
 - L_1, \dots, L_T
 - A dataset D
- Learning algorithm:
 - Use 10-CV to estimate the error of L_1, \dots, L_T
 - Pick the best (lowest 10-CV error) learner L^*
 - Train L^* on D and return its hypothesis h^*

Pros and cons of a “bucket of models”

- Pros:
 - Simple
 - Will give results not much worse than the best of the “**base learners**”
- Cons:
 - What if there’s not a single best learner?
- Other approaches:
 - Vote the hypotheses (how would you weight them?)
 - Combine them some other way?
 - How about *learning* to combine the hypotheses?

Stacked learners: first attempt

- Input:
 - your top T favorite learners (or tunings)
 - L_1, \dots, L_T
 - A dataset D containing (\mathbf{x}, y) ,
- Learning algorithm:
 - Train L_1, \dots, L_T on D to get h_1, \dots, h_T
 - Create a new dataset D' containing $(\mathbf{x}', y'), \dots$
 - \mathbf{x}' is a vector of the T predictions $h_1(\mathbf{x}), \dots, h_T(\mathbf{x})$
 - y is the label y for \mathbf{x}
 - Train new classifier on D' to get h' --- which combines the predictions!
- To predict on a new \mathbf{x} :
 - Construct \mathbf{x}' as before and predict $h'(\mathbf{x}')$

Pros and cons of stacking

- Pros:
 - Fairly simple
 - Slow, but easy to parallelize
- Cons:
 - What if there's not a single best *combination scheme*?
 - E.g.: for movie recommendation sometimes L1 is best for users with many ratings and L2 is best for users with few ratings.

Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are **good at different parts of the input space**
- **Output class:** (Weighted) vote of each classifier
 - Classifiers that are most “sure” will vote with more conviction
 - Classifiers will be most “sure” about a particular part of the space
 - On average, do better than single classifier!
- **But how do you ???**
 - force classifiers to learn about different parts of the input space?
 - weigh the votes of different classifiers?

Comments

- Ensembles based on blending/stacking were key approaches used in the netflix competition
 - Winning entries blended many types of classifiers
- Ensembles based on stacking are the main architecture used in Watson
 - Not all of the base classifiers/rankers are learned, however; some are hand-programmed.

Boosting [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let the learned classifiers vote
- On each iteration t :
 - weight each training example by how incorrectly it was classified
 - Learn a hypothesis – h_t
 - A strength for this hypothesis – α_t
- Final classifier:
 - A linear combination of the votes of the different classifiers weighted by their strength
- **Practically useful**
- **Theoretically interesting**

Learning from weighted data

- **Sometimes not all data points are equal**
 - Some data points are more equal than others
- **Consider a weighted dataset**
 - $D(i)$ – weight of i th training example (\mathbf{x}^i, y^i)
 - Interpretations:
 - i th training example counts as $D(i)$ examples
 - If I were to “resample” data, I would get more samples of “heavier” data points
- **Now, in all calculations, whenever used, i th training example counts as $D(i)$ “examples”**
 - e.g., MLE for Naïve Bayes, redefine $Count(Y=y)$ to be weighted count

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak classifier $h_t : X \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

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

where Z_t is a normalization factor

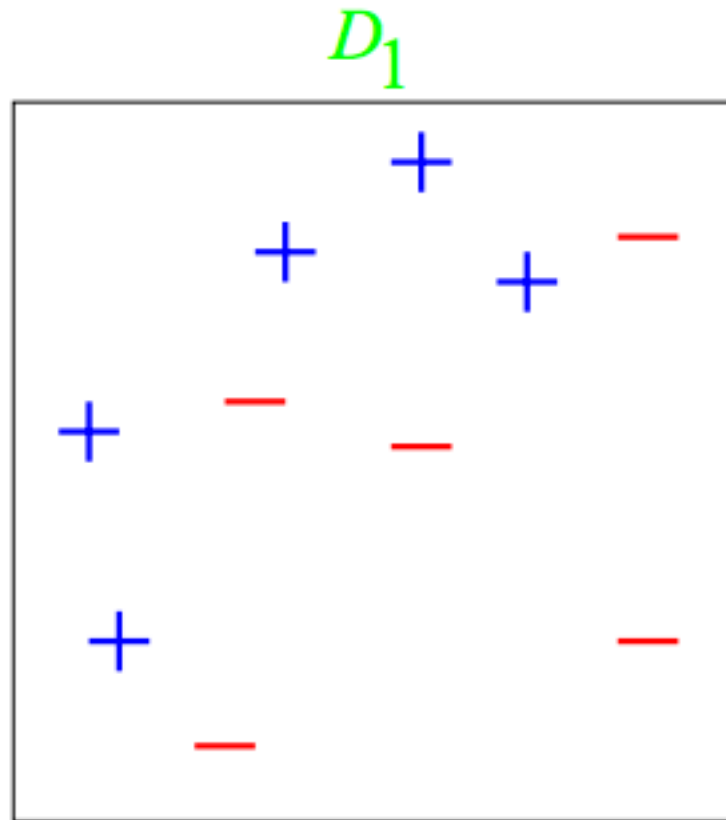
$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Output the final classifier:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

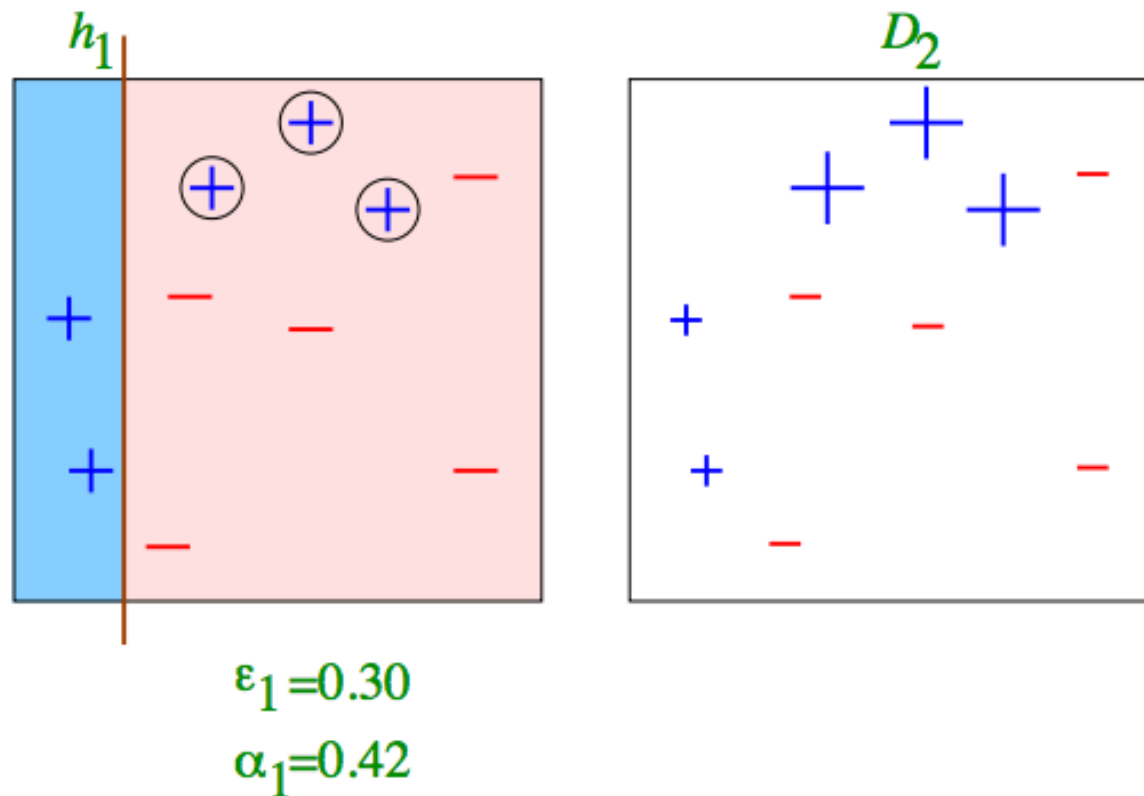
Figure 1: The boosting algorithm AdaBoost.

Boosting: A toy example



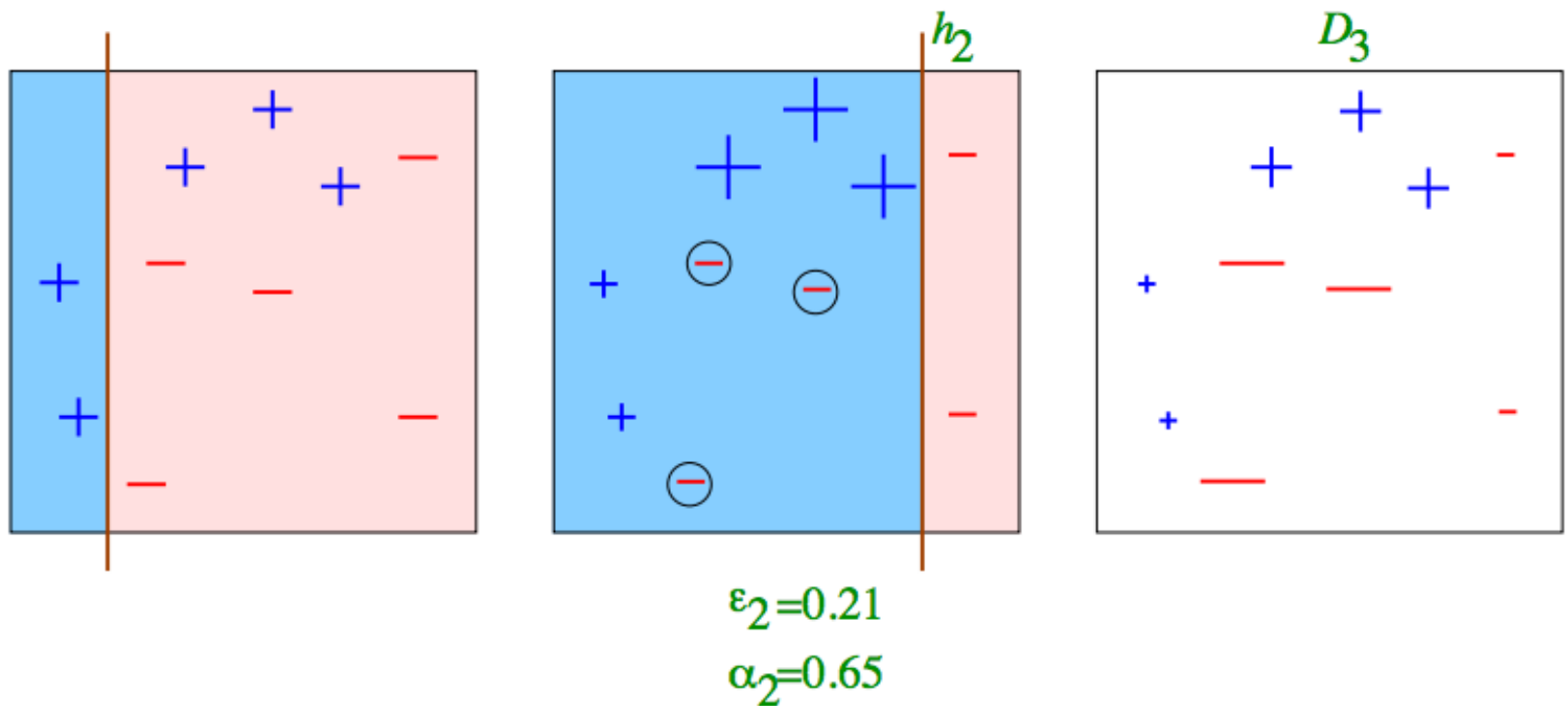
Boosting: A toy example

Round 1



Boosting: A toy example

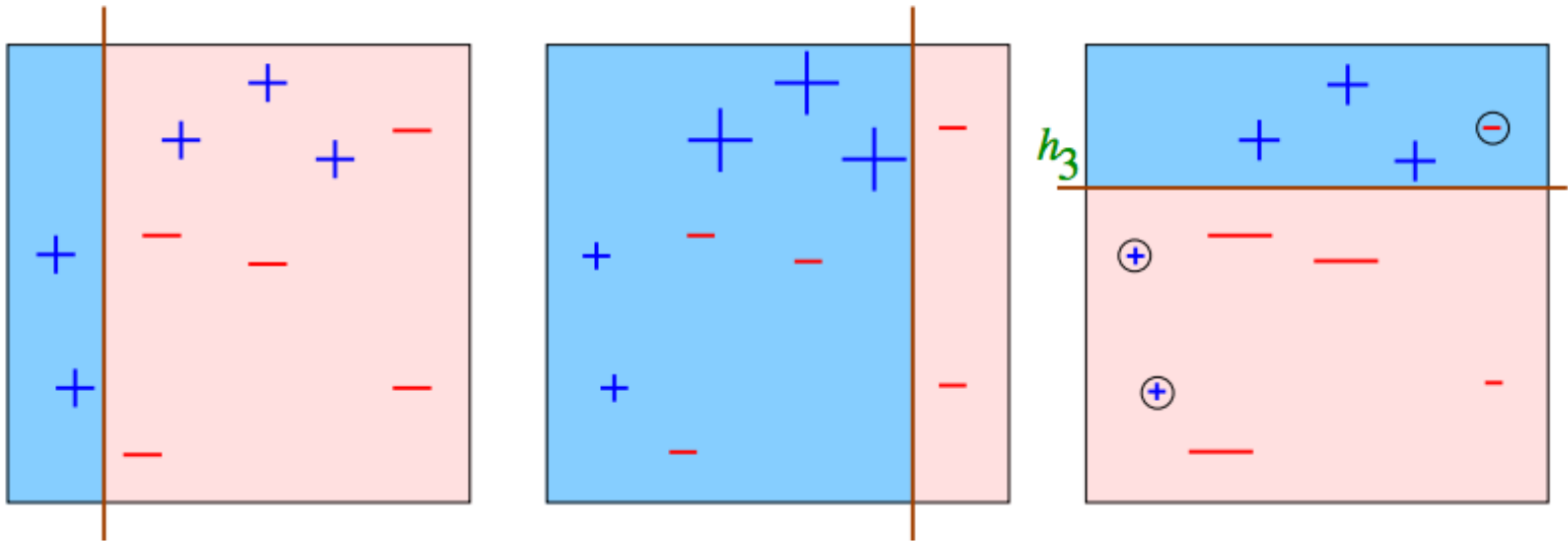
Round 2



Thanks, Rob Schapire

Boosting: A toy example

Round 3



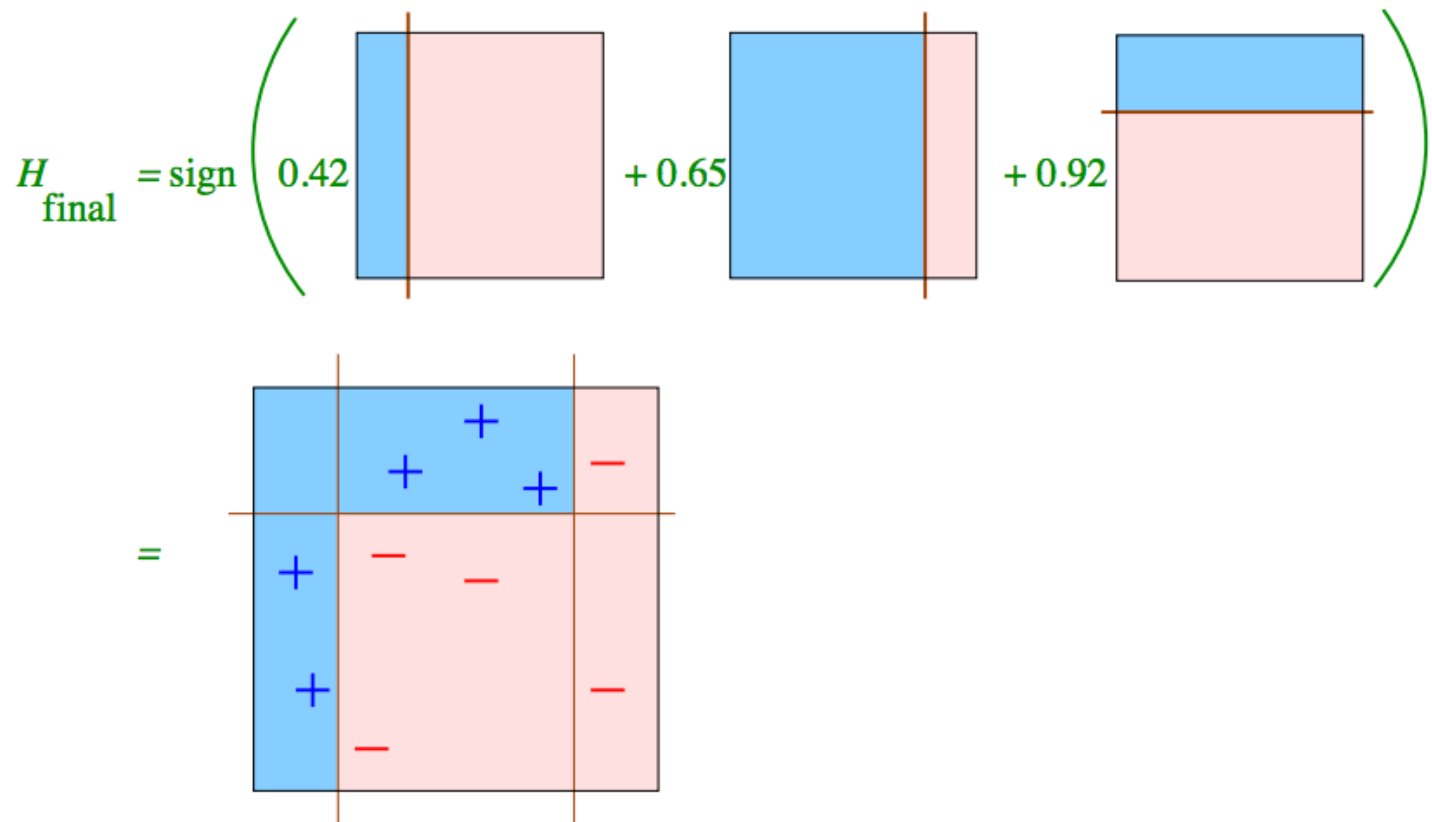
$$\epsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Thanks, Rob Schapire

Boosting: A toy example

Final Classifier



What α_t to choose for hypothesis h_t ?

[Schapire, 1989]

Training error of final classifier is bounded by:


$$\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^m \exp(-y_i f(x_i))$$

Where $f(x) = \sum_t \alpha_t h_t(x)$; $H(x) = \text{sign}(f(x))$

What α_t to choose for hypothesis h_t ?

[Schapire, 1989]

Training error of final classifier is bounded by:


$$\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_i \exp(-y_i f(x_i)) = \prod_t Z_t$$

Where $f(x) = \sum_t \alpha_t h_t(x)$; $H(x) = \text{sign}(f(x))$

If we minimize $\prod_t Z_t$, we minimize our training error

We can tighten this bound greedily, by choosing α_t and h_t on each iteration to minimize Z_t .

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

What α_t to choose for hypothesis h_t ?

[Schapire, 1989]

We can minimize this bound by choosing α_t on each iteration to minimize Z_t .

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Define

$$\epsilon_t = \sum_{i=1}^m D_t(i) \delta(h_t(x_i) \neq y_i)$$

We can show that:

$$Z_t = (1 - \epsilon_t) \exp^{-\alpha_t} + \epsilon_t \exp^{\alpha_t}$$

What α_t to choose for hypothesis h_t ?

[Schapire, 1989]

We can minimize this bound by choosing α_t on each iteration to minimize Z_t .

$$Z_t = (1 - \epsilon_t) \exp^{-\alpha_t} + \epsilon_t \exp^{\alpha_t}$$

For boolean target function, this is accomplished by [Freund & Schapire '97]:

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

Where:

$$\epsilon_t = \sum_{i=1}^m D_t(i) \delta(h_t(x_i) \neq y_i)$$

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \dots, T$:

- Train base learner using distribution D_t .
- Get base classifier $h_t : X \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

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

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

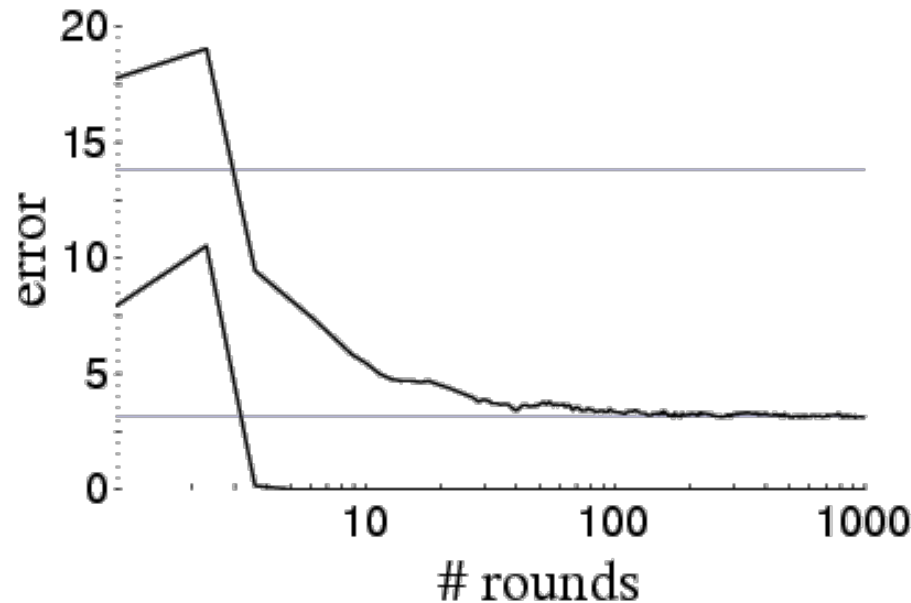
Strong, weak classifiers

- If each classifier is (at least slightly) better than random
 - $\epsilon_t < 0.5$
- With a few extra steps it can be shown that AdaBoost will achieve zero *training error* (exponentially fast):

$$\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \prod_t Z_t \leq \exp \left(-2 \sum_{t=1}^T (1/2 - \epsilon_t)^2 \right)$$

Boosting results – Digit recognition

[Schapire, 1989]

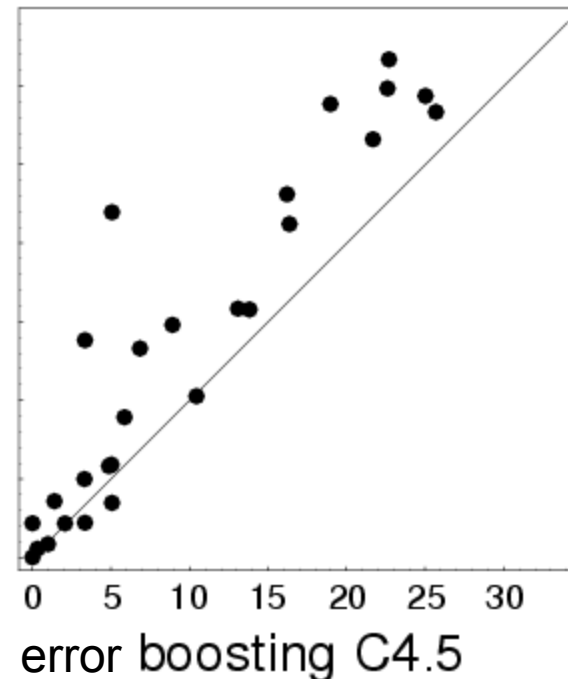
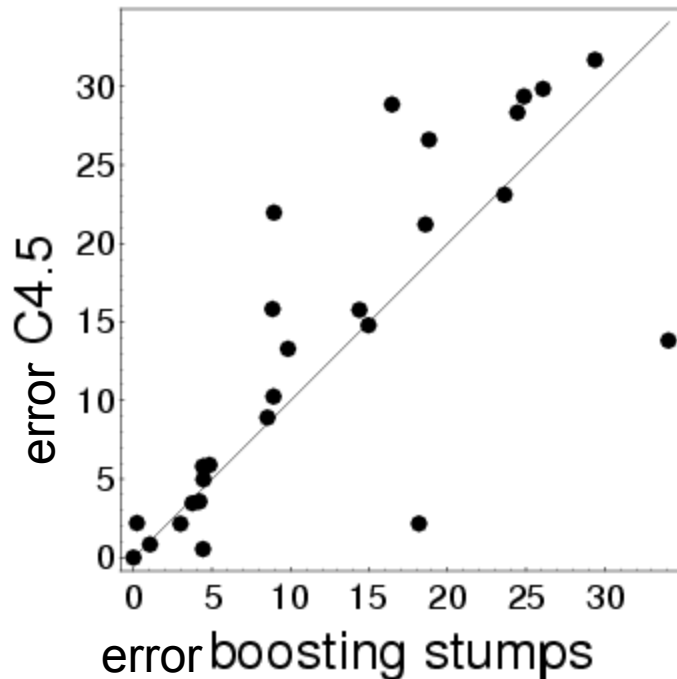


- Boosting often
 - Robust to overfitting
 - Test set error decreases even after training error is zero

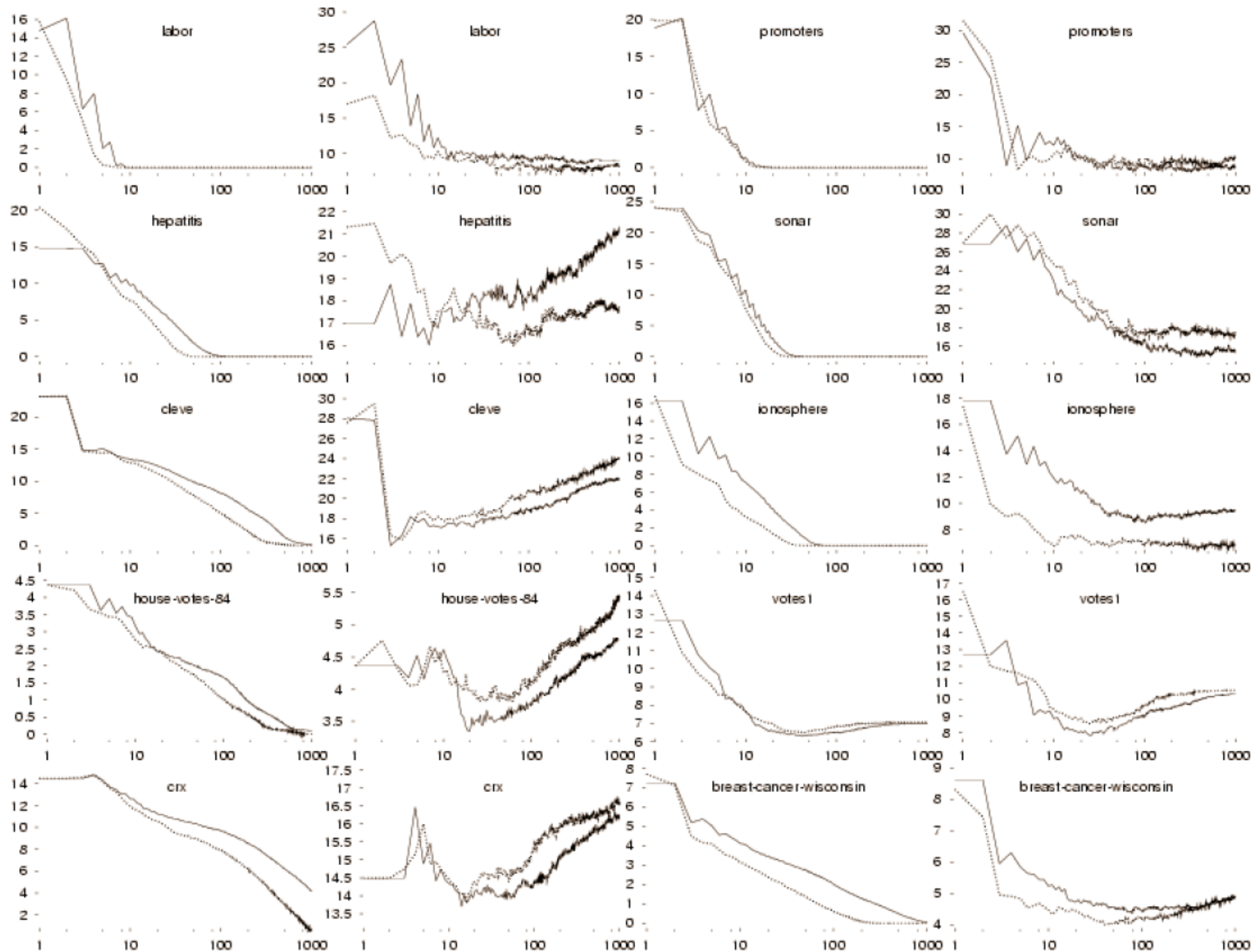
Boosting: Experimental Results

[Freund & Schapire, 1996]

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets




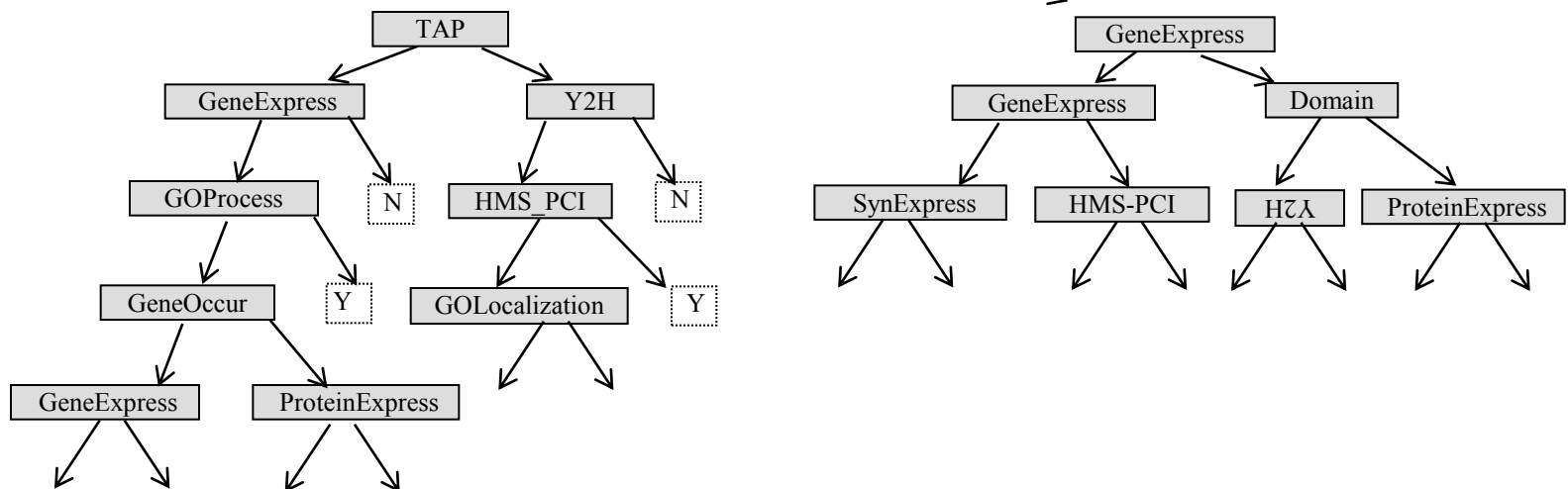
AdaBoost and AdaBoost.MH on Train (left) and Test (right) data from Irvine repository. [Schapire and Singer, ML 1999]



Random forest

- A collection of decision trees
- For each tree we select a subset of the attributes (recommended square root of $|A|$) and build tree using just these attributes

- An input sample is sing majority voting



What you need to know about Boosting

- Combine weak classifiers to obtain very strong classifier
 - Weak classifier – slightly better than random on training data
 - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Most popular application of Boosting:
 - Boosted decision stumps!
 - Very simple to implement, very effective classifier

Boosting and Logistic Regression

Logistic regression assumes:

$$P(Y = 1|X) = \frac{1}{1 + \exp(f(x))}$$

And tries to maximize data likelihood:

$$P(\mathcal{D}|H) = \prod_{i=1}^m \frac{1}{1 + \exp(-y_i f(x_i))}$$

Equivalent to minimizing log loss

$$\sum_{i=1}^m \ln(1 + \exp(-y_i f(x_i)))$$

Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss

$$\sum_{i=1}^m \ln(1 + \exp(-y_i f(x_i)))$$

Boosting minimizes similar loss function!!

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

Both smooth approximations of 0/1 loss!

Logistic regression and Boosting

Logistic regression:

- Minimize loss fn

$$\sum_{i=1}^m \ln(1 + \exp(-y_i f(x_i)))$$

- Define

$$f(x) = \sum_j w_j x_j$$

where x_j predefined

Boosting:

- Minimize loss fn

$$\sum_{i=1}^m \exp(-y_i f(x_i))$$

- Define

$$f(x) = \sum_t \alpha_t h_t(x)$$

where $h_t(x_i)$ defined
dynamically to fit data
(not a linear classifier)

- Weights α_j learned incrementally