COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Ensemble learning

Reminders/comments

- Practice final released in announcement: please do not distribute
- I will be away next Wednesday for a conference
  - Class still scheduled for Monday, but office hours canceled for me
  - Class canceled on Wednesday
- Scheduled practice final review for Monday, December 14 at 11:00 a.m. in this room

Thought questions

- Many questions about
  - neural networks parameter selection, including number of hidden layers, number of nodes within a layer, transfer functions
  - hypothesis testing parameter selection, including how to split, etc.
  - dimensionality reduction versus sparse coding, and again how to choose between them, and how to choose k
  - central limit theorem
  - bias-variance (we will talk about that today)

Though question: normal assumption

- General theme: why do we consider the errors to be normally distributed, and get confidence intervals using normal distributions? Can we/should we use other distributions?

- Central limit theorem: for iid RVs {x1, …, xn} with finite variance

$$\sqrt{n}\left(\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu\right) \xrightarrow{d} N(0, \sigma$$

  - Law of large numbers: the sample average converges to mu

- How quickly does the above RV converge? Some results have finite sample convergence rate (e.g., O(1/sqrt(n))); 30 is the usual rule of thumb for “enough” samples

Thought question: types of tests

- Student's t-test is based on assumption that variance of two samples should be equal hence my understanding is that this method is not appropriate to check our hypothesis. Should we use Welch's t-test to be safe (which doesn't require variances to be same)?

- I am taking the intro to statistics class and there we went through many kinds of tests and at the end the moral of the story was to always use the Welch's t-test. Is this the same conclusion in machine learning?
  - This is definitely a more robust option, and part of the standard tools for t-tests (matlab, python)

Thought question

- When is it necessary to test for statistical significance of the out-peformance of Algorithm A over Algorithm B? Is it only when the two algorithms are performing closely on your dataset? If Algorithm A was predicting with an accuracy of greater than > 5% or more would it still be necessary to do something like a paired t-test?
  - Should always compute standard error (confidence interval)
  - If the interval do not overlap, can say they are different
  - Even with 5% difference in mean, the intervals might overlap and you might have to do a paired t-test (equal variance or unequal variances)

Ensembles

- Can a set of **weak learners** create a single strong learner?
- Answer: yes! See seminal paper: “The Strength of Weak Learnability” Schapire, 1990
- Why do we care?
    - can be easier to specify weak learners e.g., shallow decision trees, set of neural networks with varying number of layers, etc.
    - fighting the bias-variance trade-off

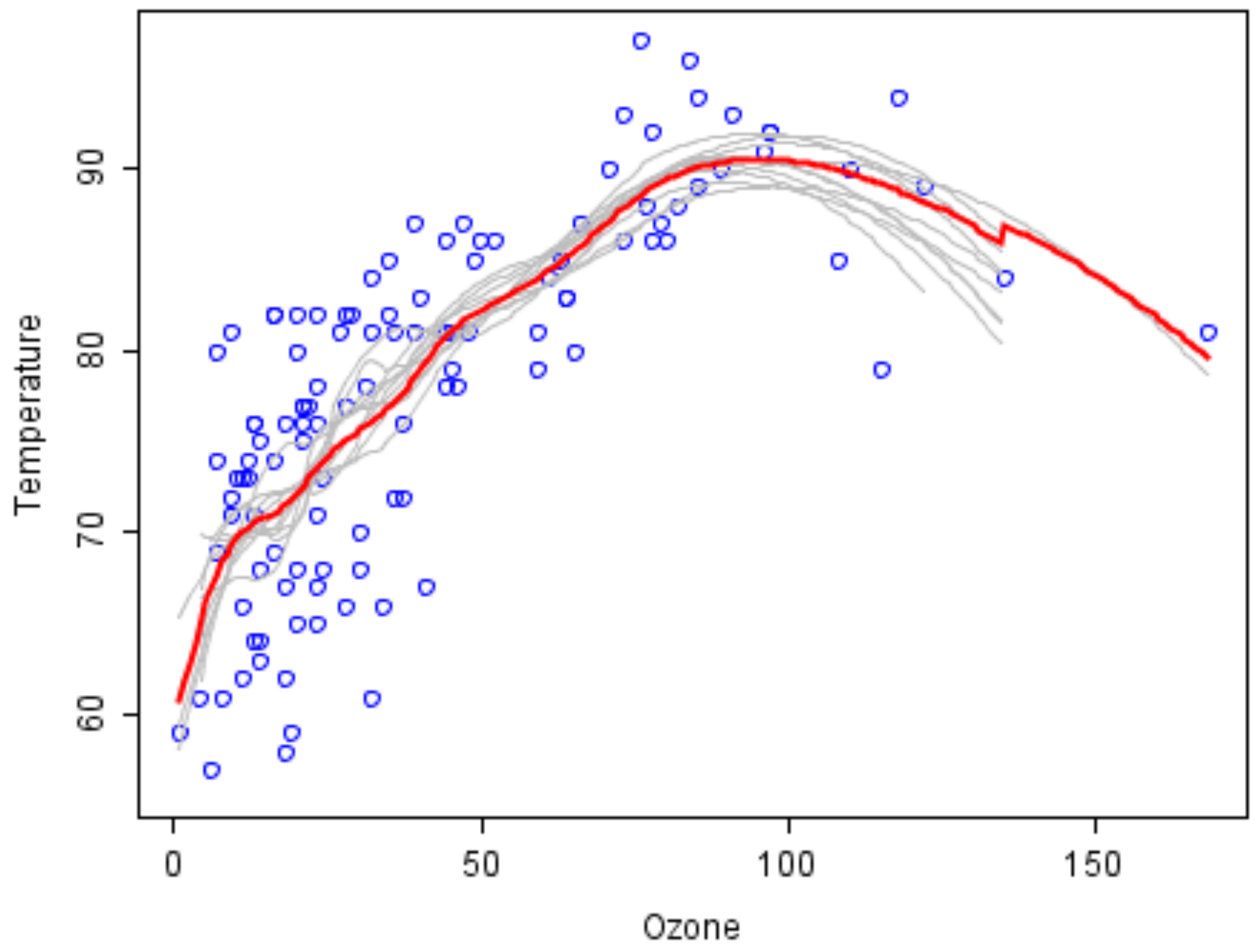
Bias-variance tradeoff

- We encountered this trade-off for weights in linear regression
- Regularizing introduced bias, but reduced variance

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{Var}(\hat{\theta}) + \left(\mathrm{Bias}(\hat{\theta}, \theta)\right)^2.$$

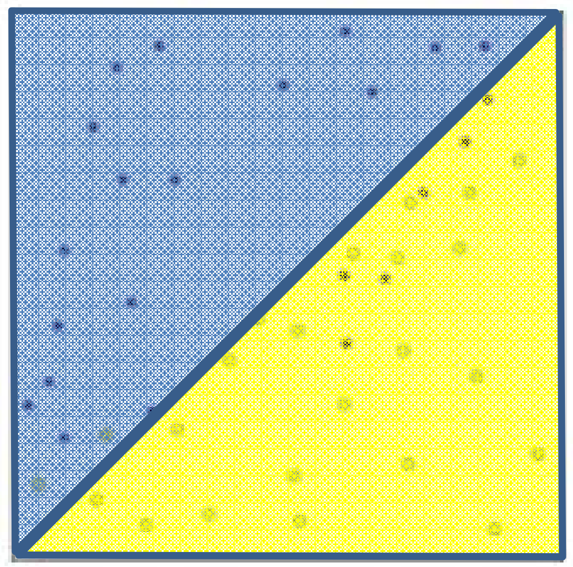
Simple approach: Bagging

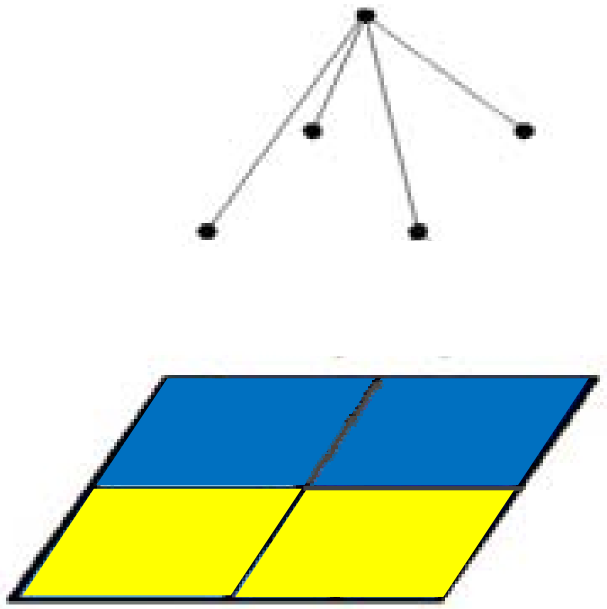
- **Bootstrap aggregating:**
  - subsample the data into m sets (bootstrapped samples)
  - learn m models on these subsets
  - average their values for predictions



Weak learners

- **Weak learners:** naive Bayes, logistic regression, decision stumps (or shallow decision trees)





Are good ☺ - Low variance, don't usually overfit

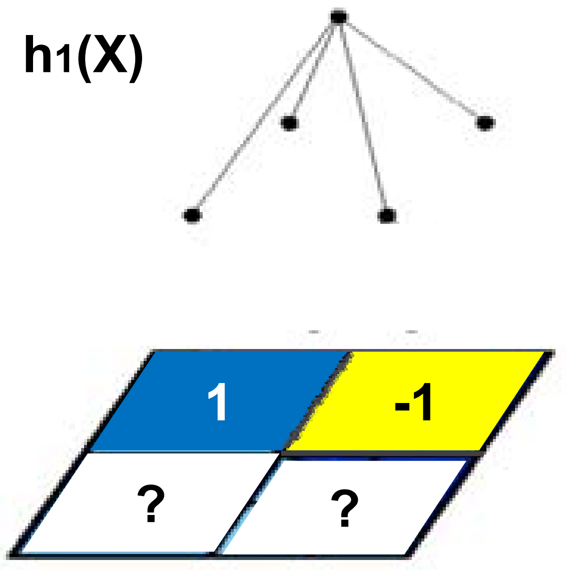
Are bad ☹ - High bias, can't solve hard learning problems

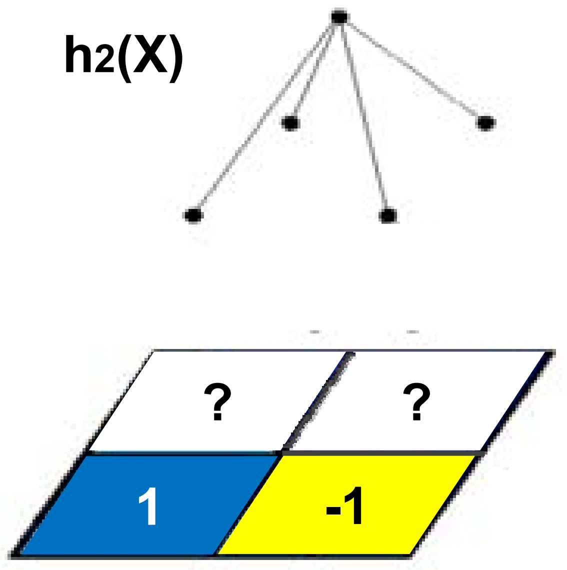
*some material from slides by Eric Xing: http://www.cs.cmu.edu/~epxing/Class/10701-11f/Lecture/lecture22.pdf

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!





H: X → Y (-1,1)

$H(X) = h_1(X)+h_2(X)$

$H(X) = \text{sign}(\sum_t \alpha_t h_t(X))$

weights

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 h_t to learn about different parts of the input space? weight the votes of different classifiers? α_t

Rationale

- There is no algorithm that is always the most accurate
- We can select simple “weak” classification or regression methods and combine them into a single “strong” method
- Different learners use different
  - Algorithms
  - Parameters
  - Representations
  - Training sets
  - Subproblems
- The problem: how to combine them

Boosting [Schapire 89]

- **Idea:** given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote
- On each iteration *t*:
  - weight each training example by how incorrectly it was classified
  - Learn a weak hypothesis – h_t
  - Obtain a strength for this hypothesis – α_t
- Final classifier: $\boxed{\mathbf{H(X) = sign(\sum \alpha_t h_t(X))}}$
- Practically useful, and theoretically interesting

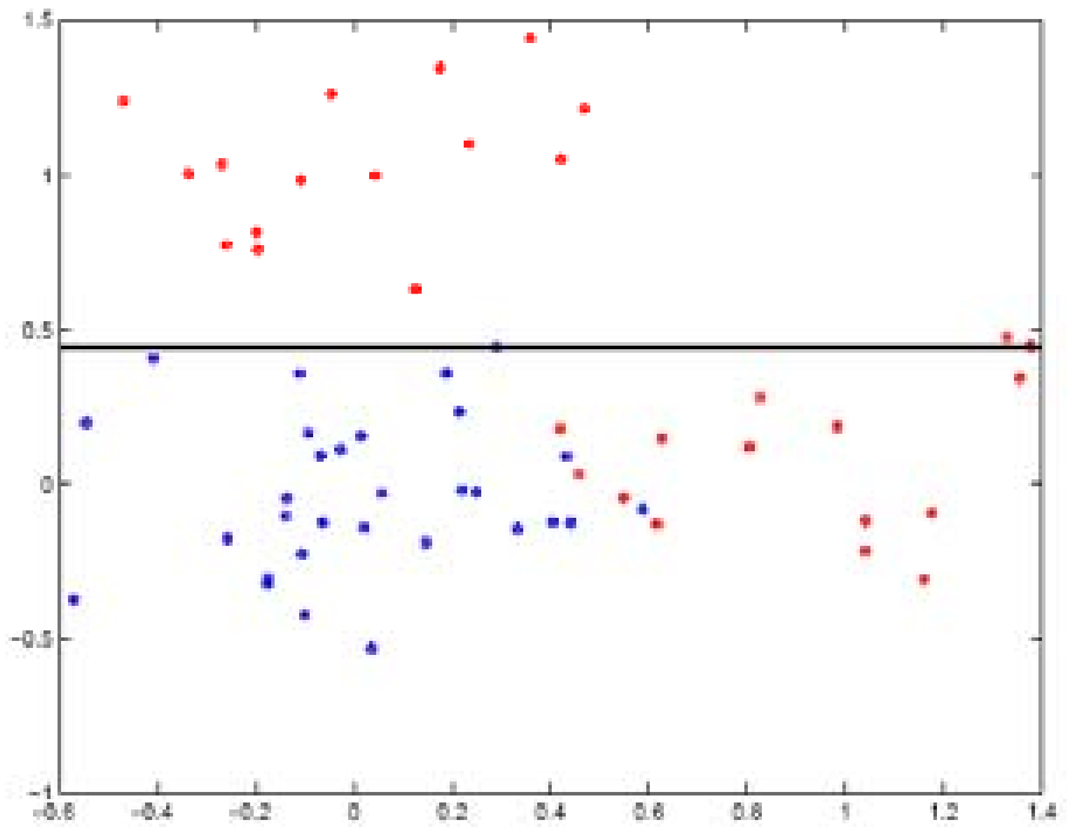
Combination of classifiers

- Suppose we have a family of component classifiers (generating ±1 labels) such as decision stumps:

$$h(x;\theta) = \text{sign}\left(wx_k + b\right)$$

where $\theta = \{k,w,b\}$

- Each decision stump pays attention to only a single component of the input vector



Combination of classifiers

- We'd like to combine the simple classifiers additively so that the final classifier is the sign of

$$\hat{h}(\mathbf{x}) = \alpha_1 h(\mathbf{x};\theta_1) + \ldots + \alpha_m h(\mathbf{x};\theta_m)$$

where the "votes" $\{\alpha_i\}$ emphasize component classifiers that make more reliable predictions than others

AdaBoost

- **Input:**
  - ***N*** examples $S_N = \{(x_1,y_1),\ldots,(x_N,y_N)\}$
  - a weak base learner $h = h(x,\theta)$
- **Initialize:** equal example weights $w_i = 1/N$ for all $i = 1..N$
- **Iterate for $t = 1\ldots T$:**
  1. train base learner according to weighted example set (w_t, x) and obtain hypothesis $h_t = h(x,\theta_t)$
  2. compute hypothesis error ε_t
  3. compute hypothesis weight α_t
  4. update example weights for next iteration w_{t+1}
- **Output:** final hypothesis as a linear combination of h_t

Adaboost

- At the kth iteration we find (any) classifier $h(\mathbf{x};\ \theta_k^*)$ for which the weighted classification error:

$$\varepsilon_k = \sum_{i=1}^{n} W_i^{k-1} I(y_i \neq h(\mathbf{x}_i;\theta_k^*)) \Big/ \sum_{i=1}^{n} W_i^{k-1}$$

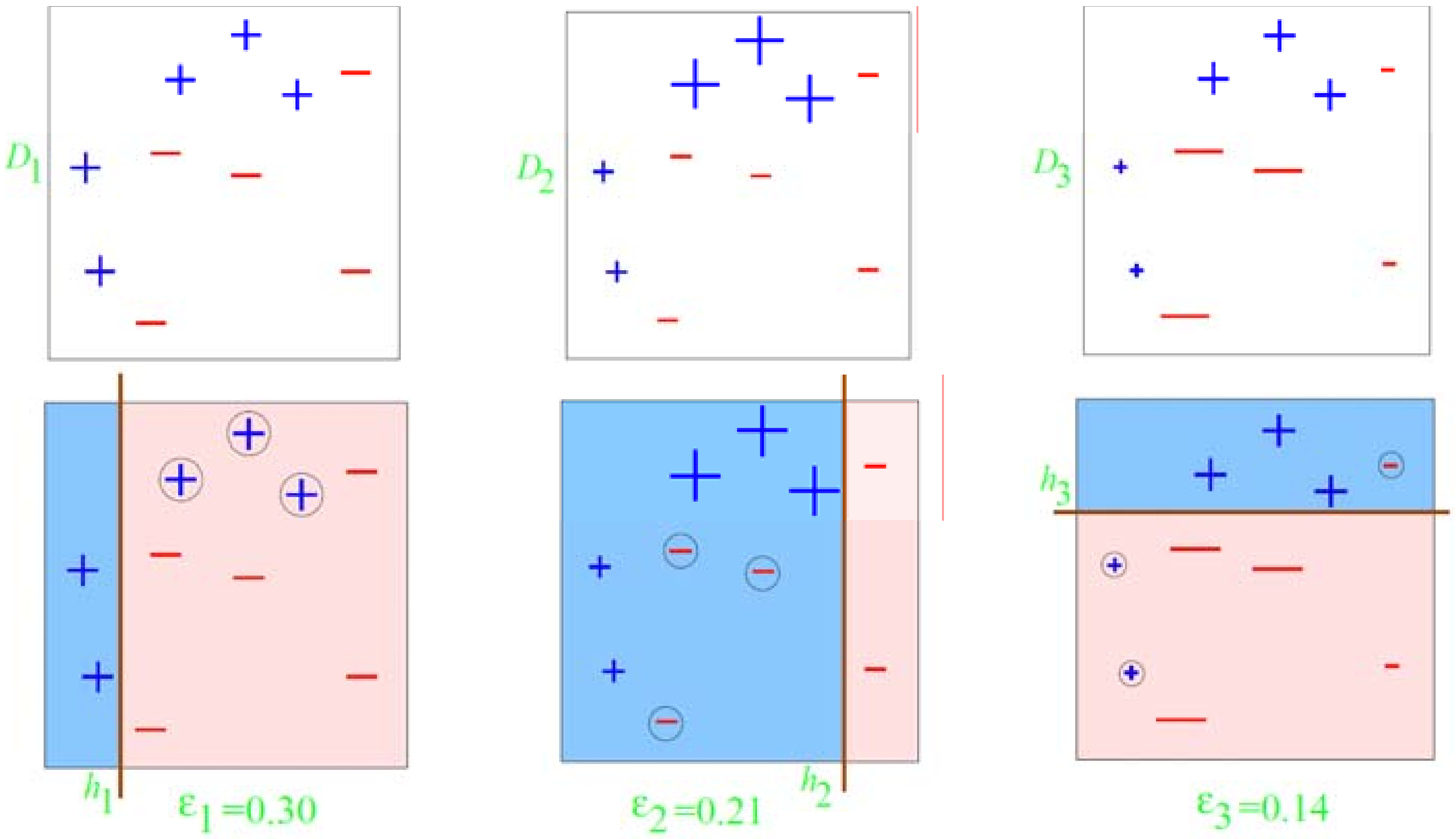
  is better than change.
  - This is meant to be "easy" --- weak classifier
- Determine how many “votes” to assign to the new component classifier:

$$\alpha_k = 0.5 \log\left((1-\varepsilon_k)/\varepsilon_k\right)$$

  - stronger classifier gets more votes
- Update the weights on the training examples:

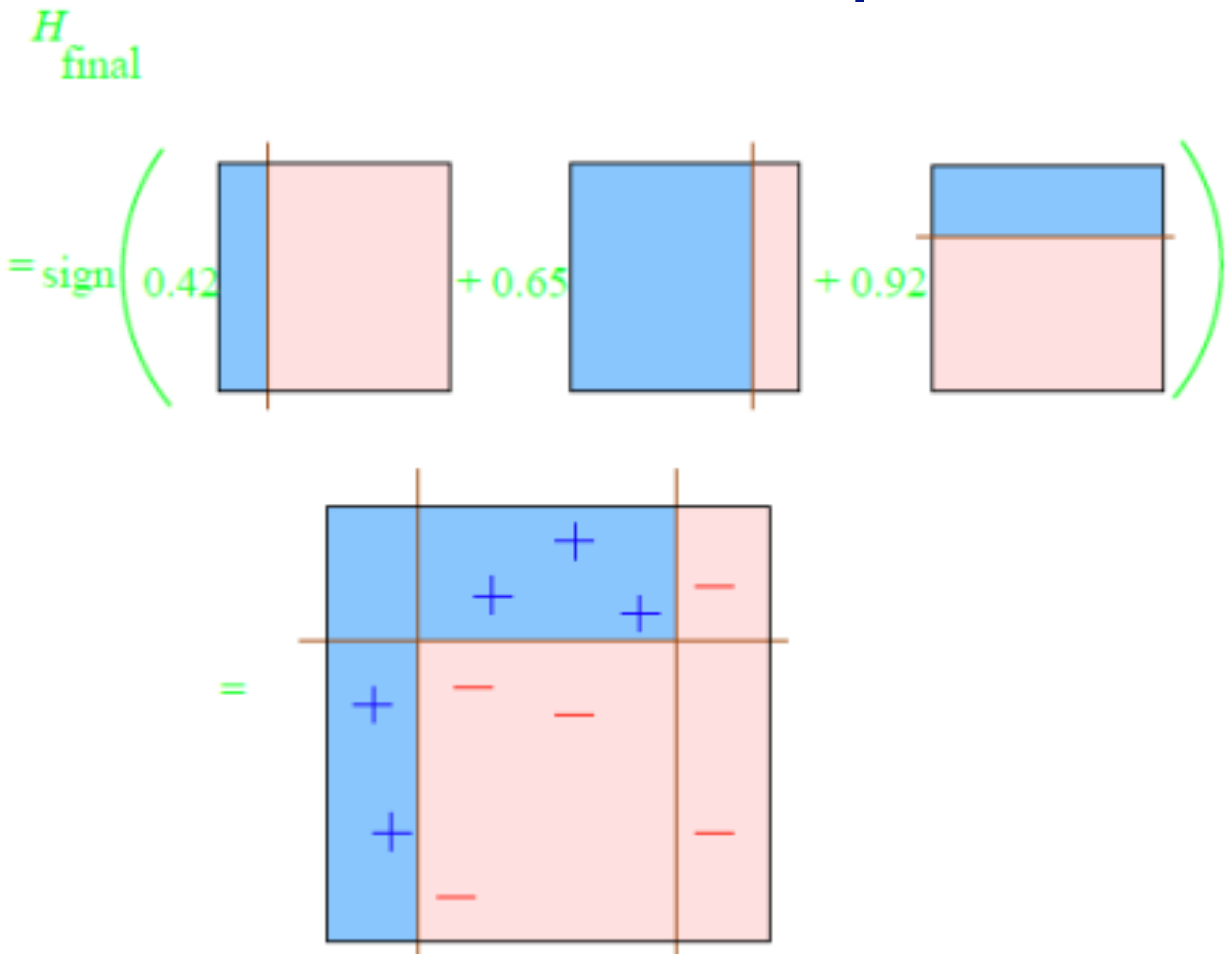
$$W_i^k = W_i^{k-1} \exp\{-y_i a_k h(\mathbf{x}_i;\theta_k)\}$$

Boosting example with decision stumps



$\alpha_1 = 0.42$ $\alpha_2 = 0.65$ $\alpha_3 = 0.92$

Boosting example with decision stumps



Base learners

- Weak learners used in practice:
  - Decision stumps
  - Decision trees (e.g. C4.5 by Quinlan 1996)
  - Multi-layer neural networks
  - Radial basis function networks
- Can base learners operate on weighted examples?
  - In many cases they can be modified to accept weights along with the In many cases they can be modified to accept weights along with the examples
  - In general, we can sample the examples (with replacement) according to the distribution defined by the weights

Thought question

- PCA tries to "compress" data, noise may be deducted as well as useful information(I guess it is possible). On the contrary, sparse coding tries to extract "higher" features from original data without any information lost. Does that mean features extracted by sparse coding are always more discriminative than PCA?
  - their goal for the new representation is quite different
  - sparse representations can also often be thought of as local
  - PCA obtains (typically) global representations
  - Both approaches are to help make learning easier
  - If PCA keeps almost all singular values, then just as discriminative; if aggressively remove many, then are throwing away information
  - For sparse coding has an aggressive regularizer, could also throw away information

Thought questions

- [Learning representations] Is there any relation between dimension reduction and data compression algorithms, like zip, mp3, mp4? What I can see is that they both try to represent things in less information. If there is some relation, can we improve compression process using machine learning techniques, or can we utilize knowledge of compression algorithms to enhance our learning algorithms?
  - Several compression algorithms based on prediction

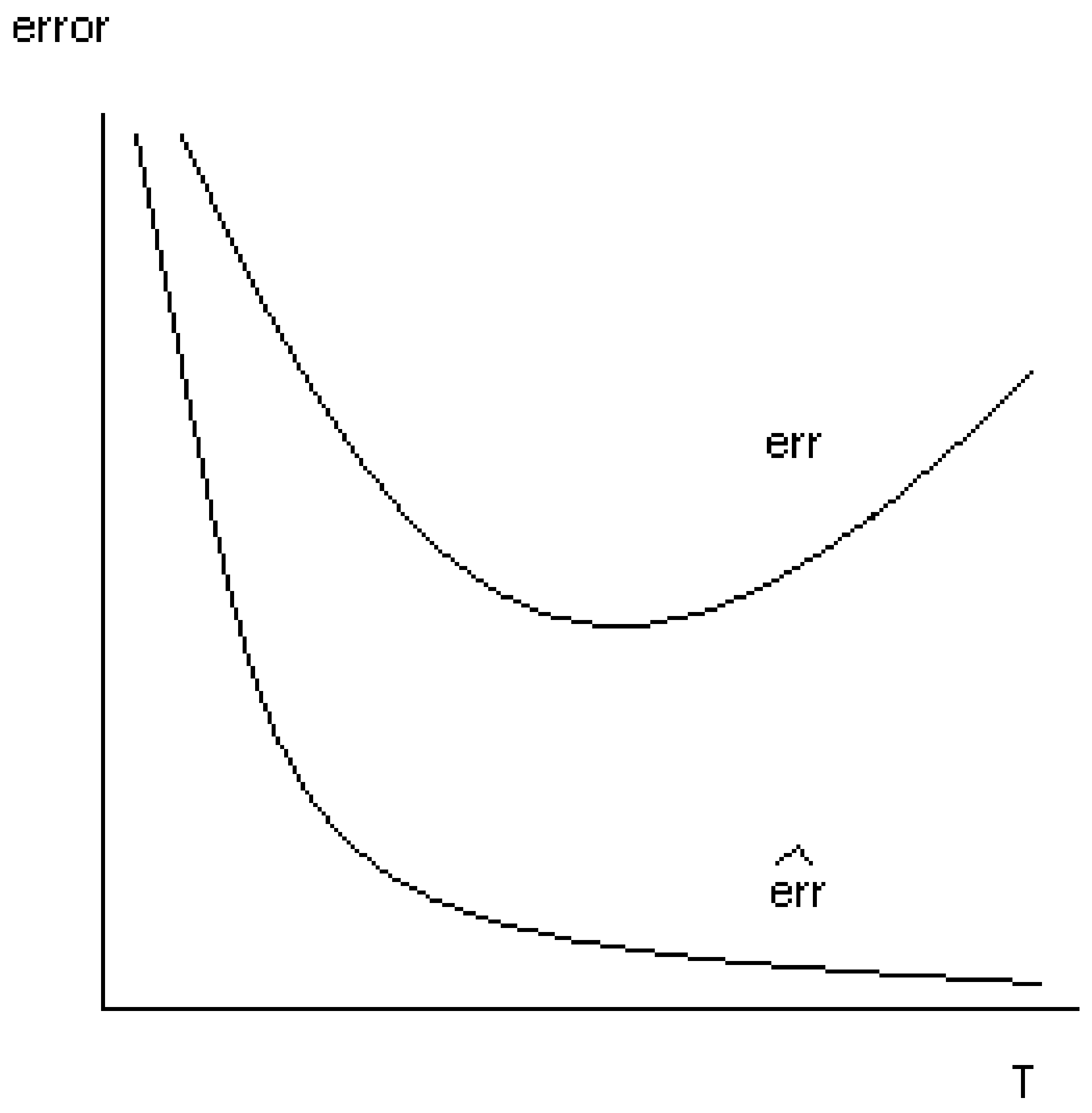
Generalization error bounds for Adaboost

$$error_{true}(H) \le error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

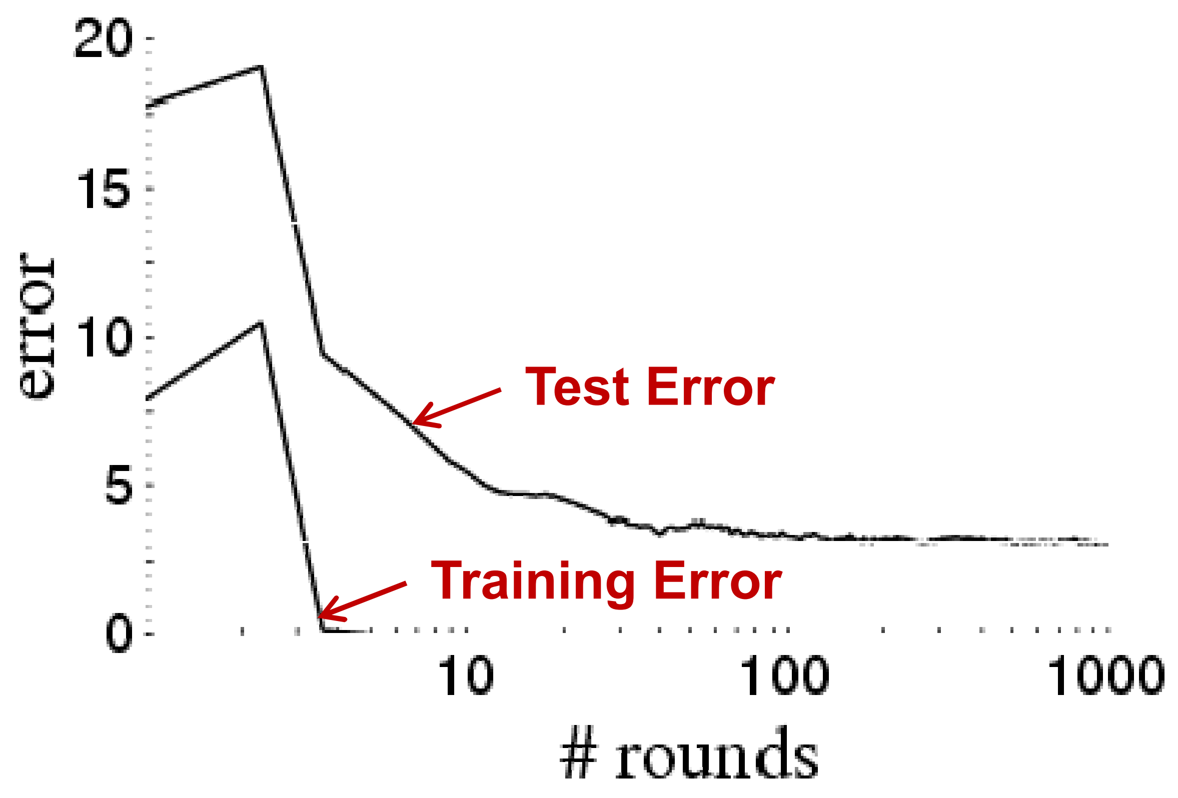
	bias	variance	
$\updownarrow$ tradeoff	large	small	T small
	small	large	T large

- T – number of boosting rounds
- d – VC dimension of weak learner, measures complexity of classifier
- m – number of training examples

Expected Adaboost behavior due to overfitting



Adaboost in practice



- Boosting often, **but not always**
  - Robust to overfitting
  - Test set error decreases even after training error is zero

Why does this seem to contradict the generalization bound?

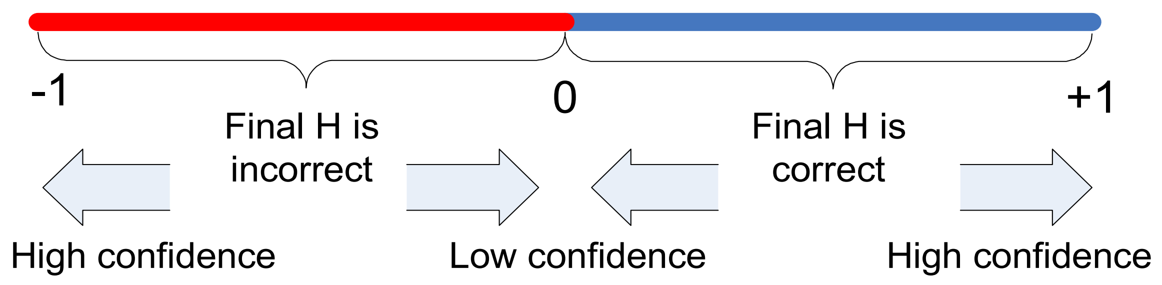
Intuition

- Even when training error becomes zero, the confidence in the hypotheses continues to increase
- This increase reduces the generalization error (rather than causing overfitting)
- Quantify with margin bound, to measure confidence of a hypothesis: when a vote is taken, the more predictors agreeing, the more confident you are in your prediction

Margin

$$\begin{aligned}\text{margin}(x,y) &= yf(x) \\ &= y\sum_t a_t h_t(x) \\ &= \sum_t a_t y h_t(x) \\ &= \sum_{t:h_t(x)=y} a_t - \sum_{t:h_t(x)\neq y} a_t\end{aligned}$$

where y is the correct label of instance x, and a_t is a normalized version of α_t such that $\alpha_t \geq 0$ and $\sum_t a_t = 1$. The expression $\sum_{t:h_t(x)=y} a_t$ stands for the weighted fraction of correct votes, and $\sum_{t:h_t(x)\neq y} a_t$ stands for the weighted fraction of incorrect votes. Margin is a number between -1 and 1 as shown in Figure 4.



* from http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0305.pdf

Margins and Adaboost

- AdaBoost increases the margins
- Large margin in training indicates lower generalization error
- For any γ, the generalization error is less than:

$$\Pr\left(\text{margin}_h(\mathbf{x},y) \le \gamma\right) + O\left(\sqrt{\frac{d}{m\gamma^2}}\right)$$

Robert E. Schapire, Yoav Freund, Peter Bartlett and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651-1686, 1998.

- It does not depend on T!!! $\longleftarrow$ The number of boosting rounds

General Boosting

- "Boosting algorithms as gradient descent": http://papers.nips.cc/paper/1766-boosting-algorithms-as-gradient-descent.pdf
- Adaboost is only one of many choices, with exponential loss
- Other examples: LogitBoost, confidenceBoost, ARC-X4
- Main idea: given some loss L, (implicit) set of hypotheses and a weak learning algorithm,
    - generate hypothesis ht that point in a descent direction
    - assign weight relative to how much pointing in descent direction

Boosting and logistic regression

Logistic regression equivalent to minimizing log loss

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

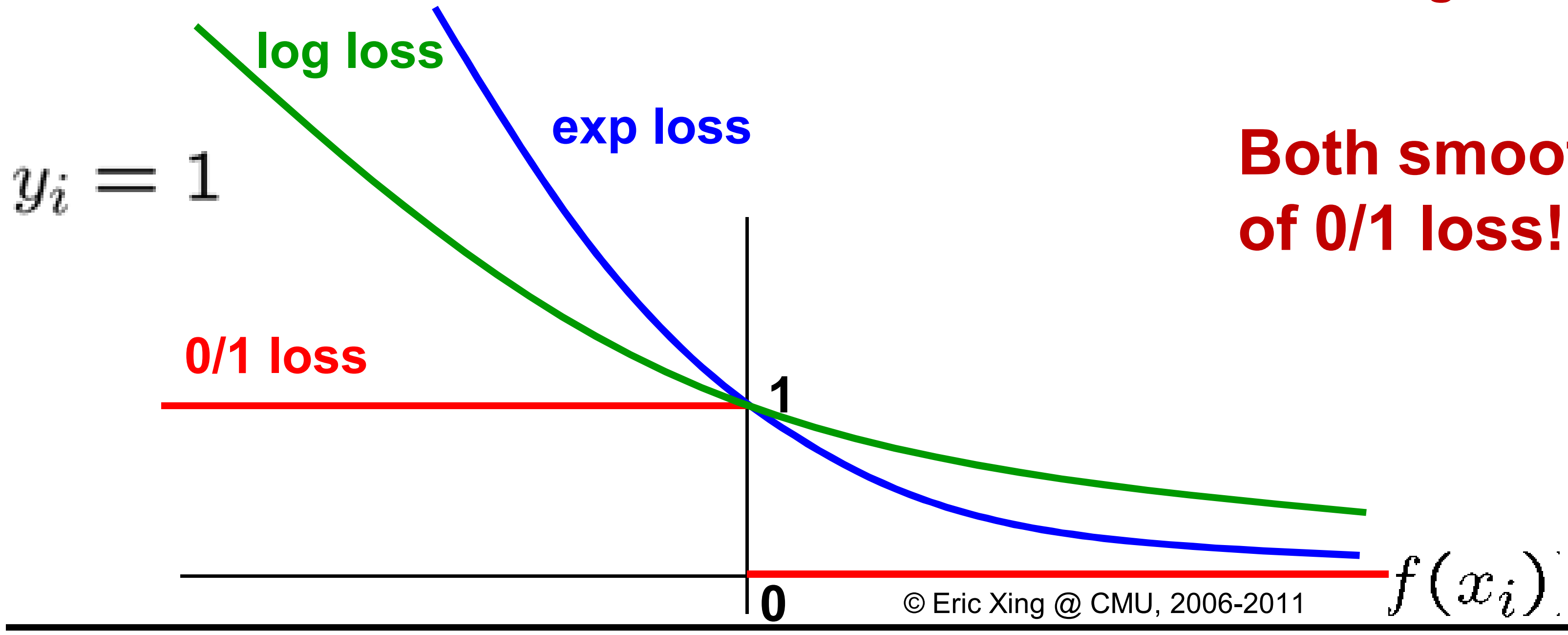
$$f(x) = w_0 + \sum_j w_j x_j$$

Boosting minimizes similar loss function!!

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

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

Weighted average of weak learners



Both smooth approximations of 0/1 loss!

Relationship to Bayes optimal classifier

- The Bayes optimal classifier is an ensemble of all the possible hypotheses
- No other ensemble can outperform it
- Usually not practical to use, but well-motivated theoretically and approximations to it (i.e., ensembles) work well in practice
- Up to now, we have mostly been using only one model
    - Can you think of any ensemble models we have considered?

Diversity of the ensemble

- An important property appears to be diversity of the ensemble
- We get to define the hypothesis space: does not have to be homogenous (e.g., the set of linear classifiers)
- Strategies to promote this include:
    - using different types of learners (e.g., naive Bayes, logistic regression and decision trees)
    - pruning learners that are similar
    - random learners, which are more likely to be different than strong/ deliberate algorithms which might learn similar predictions