CSC4007 Advanced Machine Learning

Lesson 07: Ensemble Methods

by Vien Ngo
EEECS / ECIT / DSSC

Outline

- Ensemble methods introduction
- Bias/variance trade-off
- Bagging
- Boosting

Outline

- **Ensemble methods introduction**
- Bias/variance trade-off
- Bagging
- Boosting

No Free Lunch Theorem

Our model is a simplification of reality

Simplification is based on assumptions

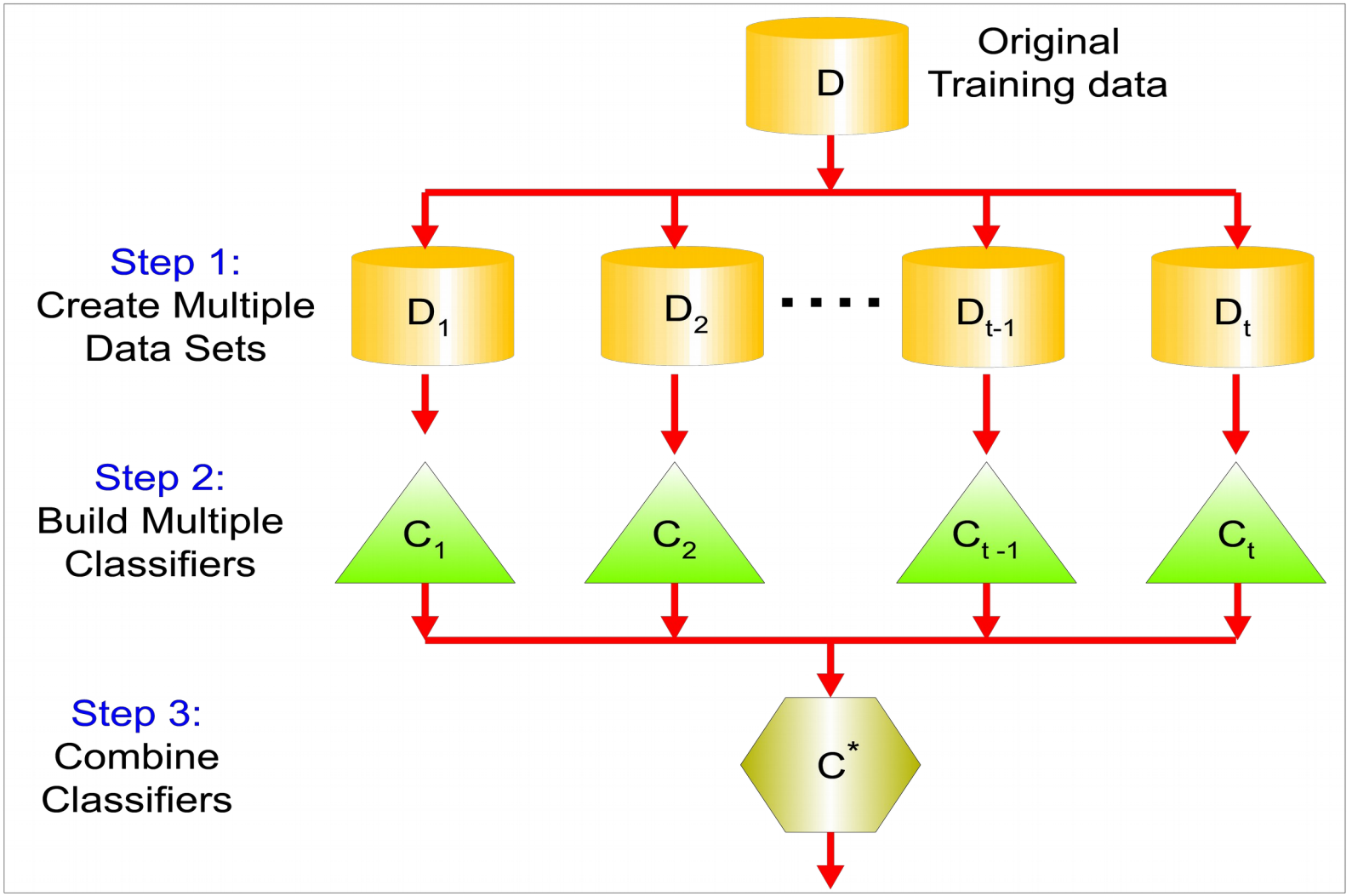
Assumptions fail in certain situations

No single algorithm wins all the time!

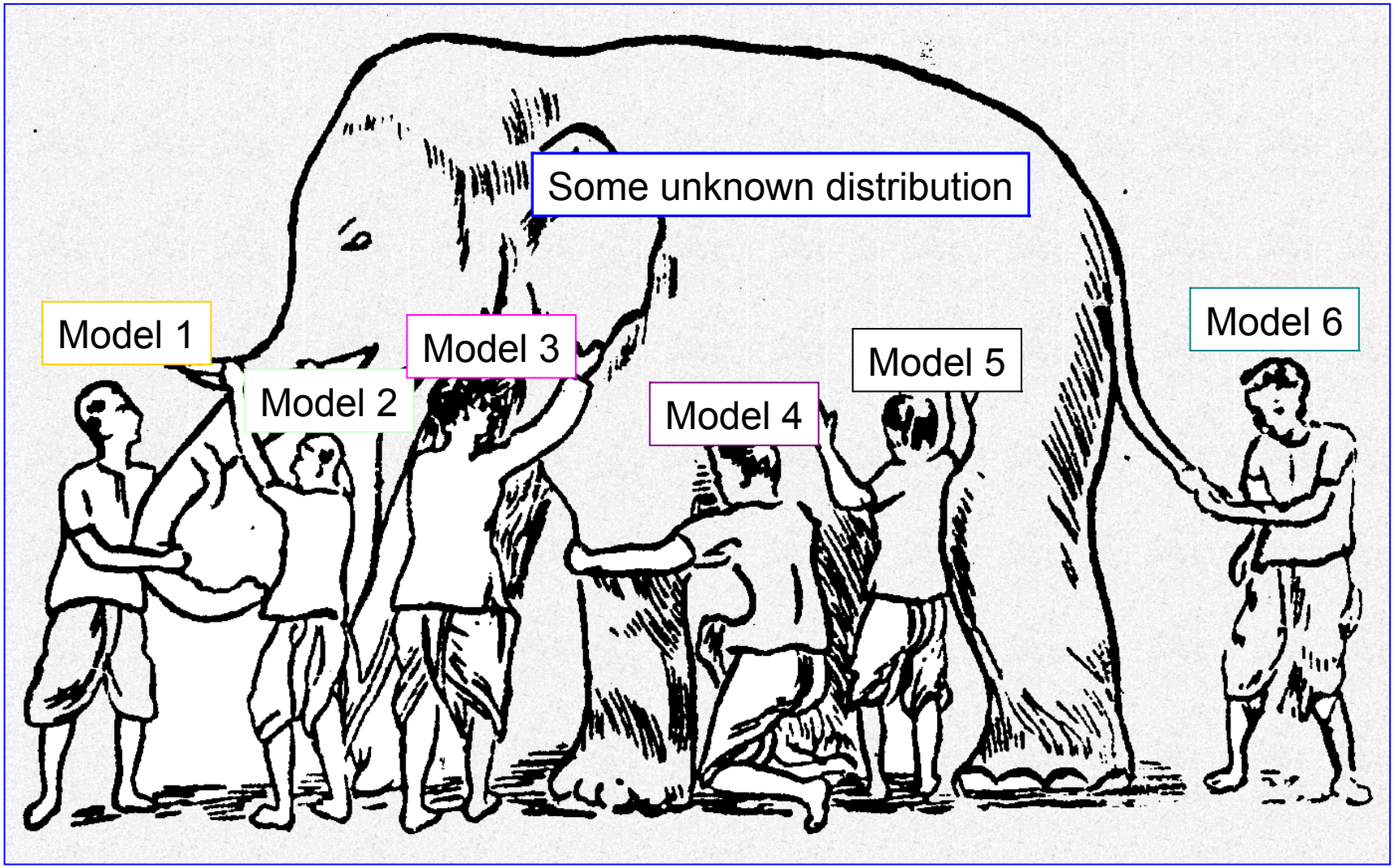
Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, 67.

Slide credit: D. Hoiem

Ensemble Methods: General Idea



Why Ensemble Works?



Ensemble gives the global picture!

from W. Fan, Random Decision Tree.

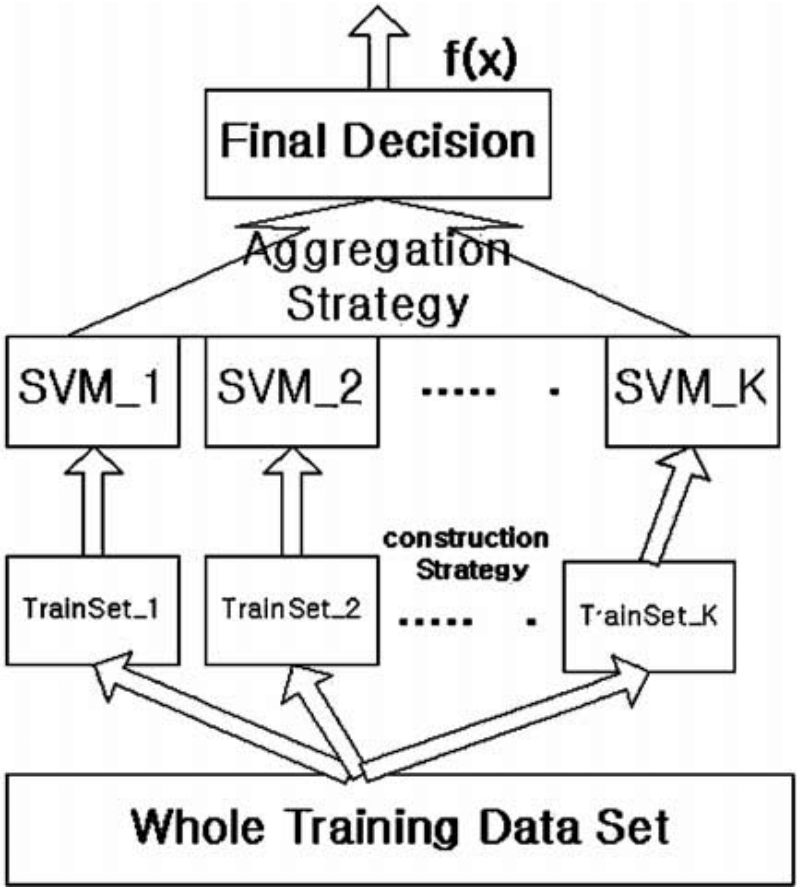
Value of Ensembles: the Wisdom of Crowds

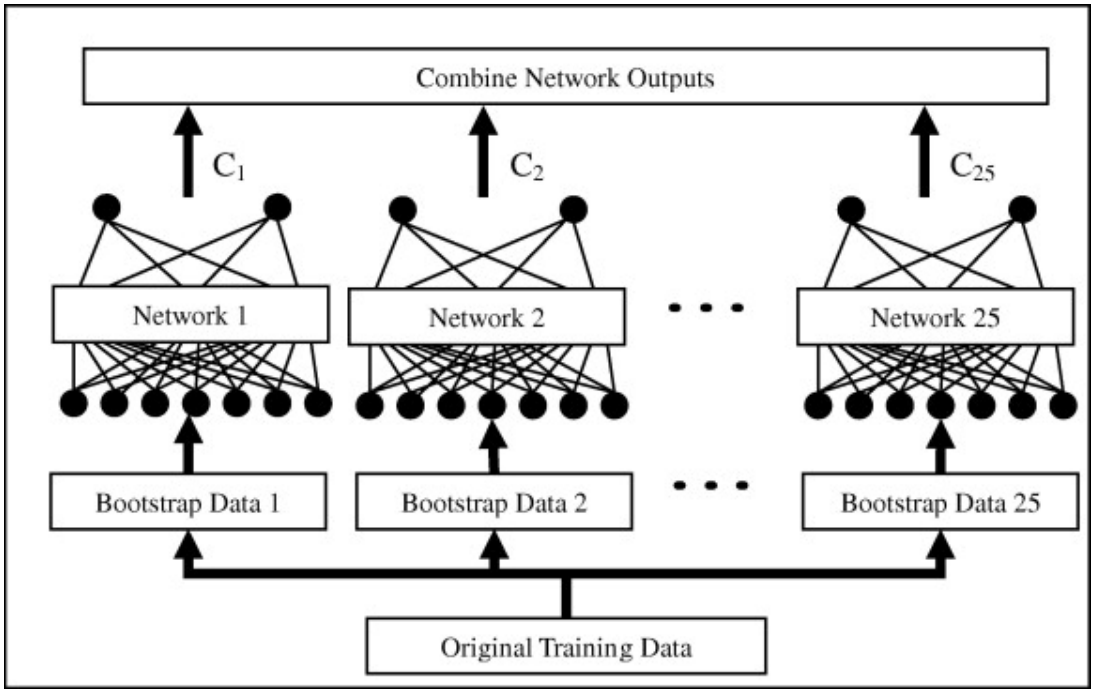
- "No Free Lunch" Theorem
  - No single algorithm wins all the time!
- When combining multiple independent decisions, each of which is at least more accurate than random guessing, random errors cancel each other out (e.g. by taking the **majority vote**) and correct decisions are reinforced.

- Human ensembles are demonstrably better
  - Which of these is not a web browser?
  - **Who Wants to be a Millionaire**: "Ask the audience"

Ensemble Methods: the Wisdom of Crowds

- One can mix-and-match classification or regression algorithms within the ensemble framework:
  - *E.g. ensembles of SVMs, logistic regression classifiers, linear discriminant classifiers, neural networks for classification, etc.*

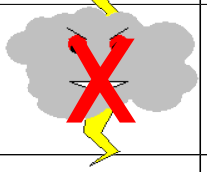




by Hong-Mo Je

By Myoung-JongKim

Example: Weather Forecast

Reality							
Classifier 1							
Classifier 2							
Classifier 3							
Classifier 4							
Classifier 5							
Combine							

By Carla P. Gomes

What is Ensemble Learning?

- **Ensemble:** collection of base learners
  - Each learns the target function
  - Combine their outputs for a final prediction
  - Often called “meta-learning”

- How can you get **different** learners?
- How can you **combine** learners?

Outline

- **Bias/variance trade-off**

- Ensemble methods that minimize variance
  - Bagging

- Ensemble methods that minimize bias and variance
  - Boosting

The machine learning framework: Steps

Training

Training Images

Training Labels

Image Features

Training

Learned model

Testing

Test Image

Image Features

Learned model

Prediction

Slide credit: D. Hoiem and L. Lazebnik

Generalization

Training set (labels known)

Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

Slide credit: L. Lazebnik

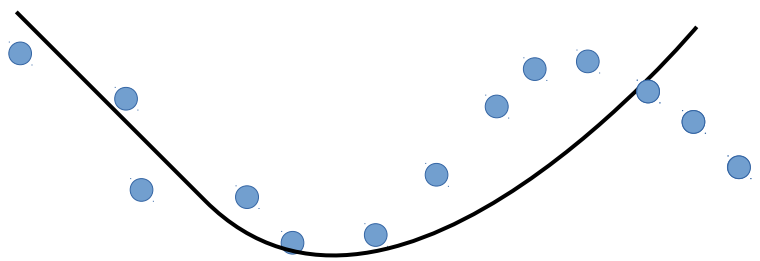
Generalization

- Components of generalization error
  - **Bias:** how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model
  - **Variance:** how much models estimated from different training sets differ from each other
- **Underfitting:** model is too “simple” to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

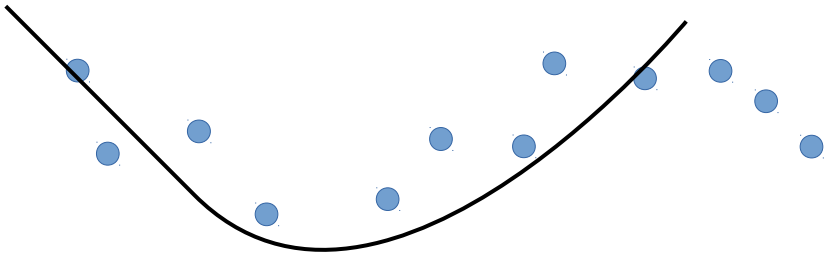
Slide credit: L. Lazebnik

Generalization: Underfitting vs. Overfitting

- **Underfitting:** Variance is low because models estimated from different training sets is not much different from each other (not enough flexibility). High bias = high fitting error.

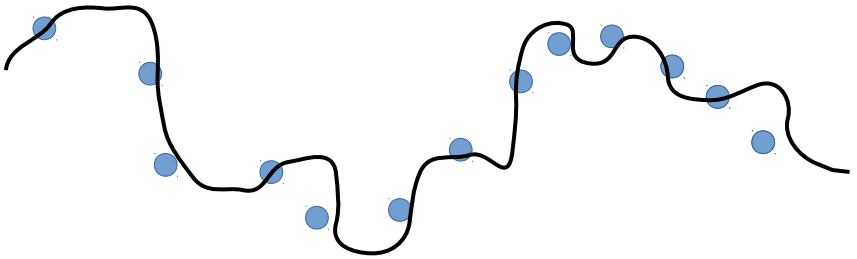


Fitting model on training set 1

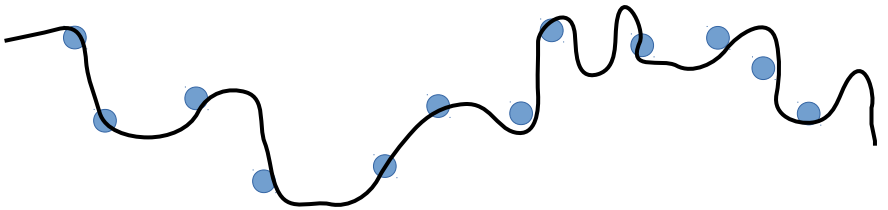


Fitting model on training set 2

- **Overfitting:** Variance is large because models estimated from different training sets is very much different from each other (too much sensitivity to the sample). Low bias = low fitting error.



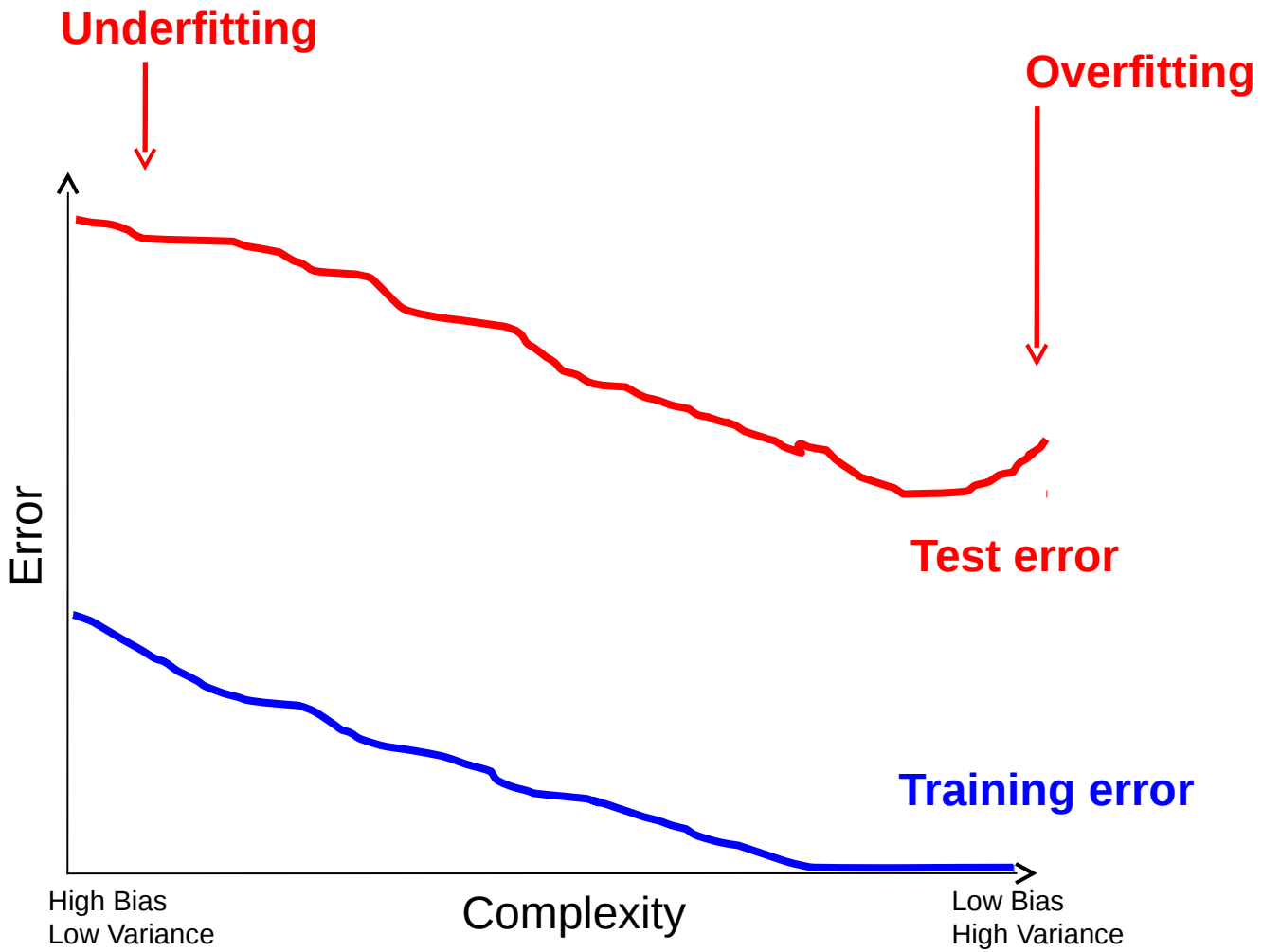
Fitting model on training set 1



Fitting model on training set 2

Bias-Variance Trade-off

Training and testing errors vs. model complexity



Slide credit: D. Hoiem

Bias-Variance Trade-off

Expectation of the mean square error (MSE) (w.r.t dataset and noise)

$$E(MSE) = noise^2 + bias^2 + variance$$

Unavoidable error

Error due to incorrect assumptions

Error due to variance of training samples

Where:

$$E(MSE) = \frac{1}{k}\left[MSE(dataset\,1) + MSE(dataset\,2) + \ldots + MSE(dataset\,k)\right]$$

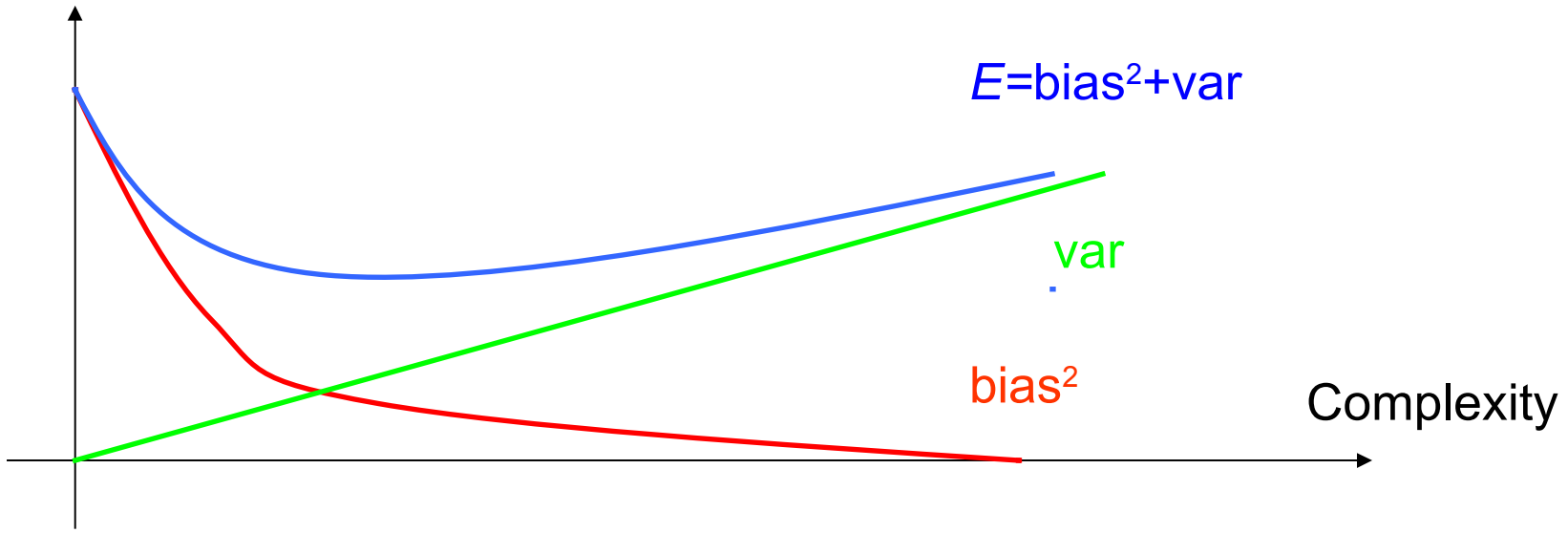
MSE measure on the second data set

See the following for explanations of bias-variance (also Bishop's "Neural Networks" book):

- http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

Slide credit: D. Hoiem

Complexity of the Model



Usually, the bias is a decreasing function of the complexity,
while variance is an increasing function of the complexity.

Remember…

- No classifier is inherently better than any other: you need to make assumptions to generalize.

- Three kinds of error:
  - Inherent (noise): unavoidable
  - Bias: due to over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data

Outline

- Bias/variance trade-off

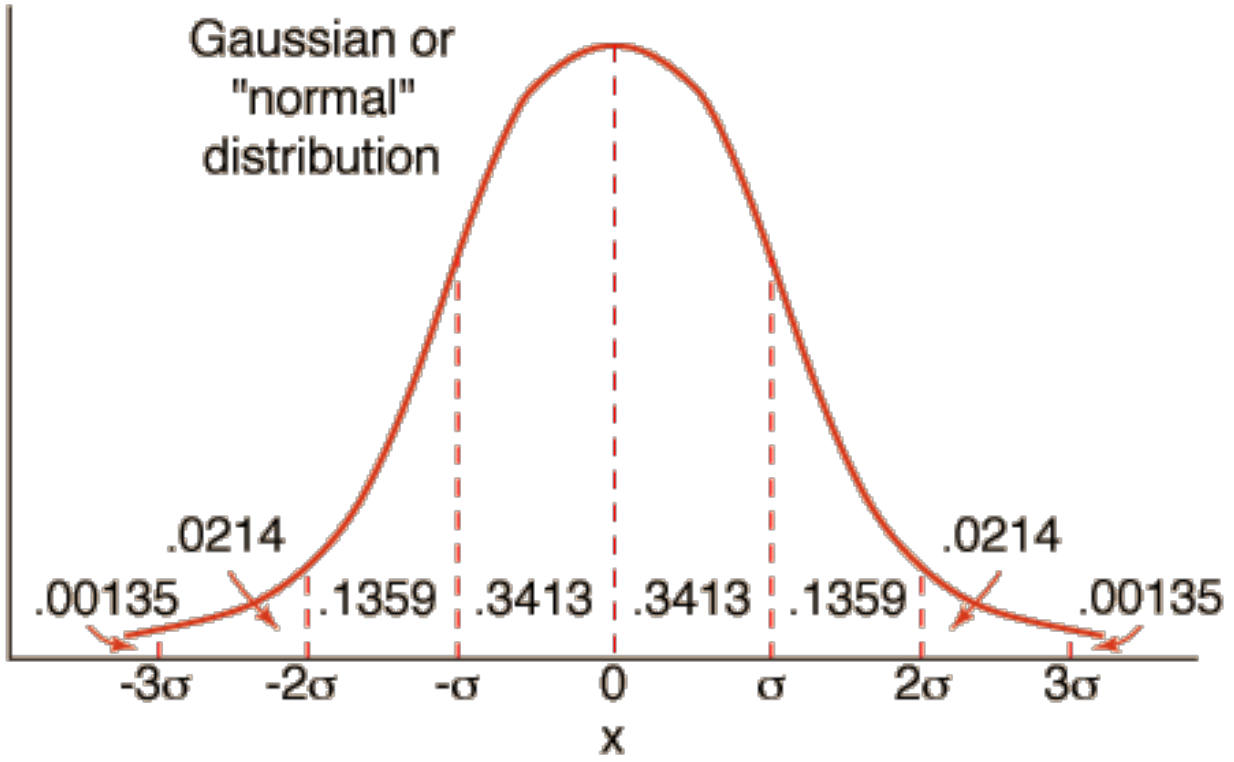
- **Ensemble methods that minimize variance**
  - **Bagging**

- Ensemble methods that minimize bias and variance
  - Boosting

Averaging decreases variance

- Motivating example:

  – Assume we measure a random variable x with a Gaussian distribution $N(\mu, \sigma^2)$



Averaging decreases variance

- Example

  – Assume we measure a random variable x with a Gaussian distribution $N(\mu, \sigma^2)$

- If only one measurement x_1 is done,
  - The expected mean of the measurement is μ
  - Variance is $\mathrm{Var}(x_1) = \sigma^2$
- If random variable x is measured K times (x_1, x_2, … , x_K) and the value is estimated as: $(x_1 + x_2 + \ldots + x_K)/K = K\mu/K = \mu$,
  - Mean of the estimate is still μ
  - But, variance of the mean is smaller:

    $[\mathrm{Var}(x_1) + \mathrm{Var}(x_2) + \ldots + \mathrm{Var}(x_K)]/K^2 = K\sigma^2/K^2 = \sigma^2/K$
- **Proposal: Bagging is a kind of averaging!**

Averaging decreases variance: Example

- Coin tossing?
- Three sequences of tossing (each sequence is 10 tossing times) (0 means tail, 1 means head). Let's calculate the probability of being head:
  - Experiment 1: 0, 1, 0, 1, 1, 0, 1, 0, 0, 1

$$mean_1 = \frac{5}{10} = 0.5; var_1 = \frac{10 \times 0.5^2}{10} = 0.25$$

  - Experiment 2: 1, 1, 0, 1, 1, 0, 1, 0, 0, 1

$$mean_2 = \frac{6}{10} = 0.6; var_2 = \frac{4 \times 0.6^2 + 6 \times 0.4^2}{10} = 0.24$$

  - Experiment 3: 0, 1, 0, 1, 0, 0, 1, 0, 0, 1

$$mean_2 = \frac{4}{10} = 0.4; var_2 = \frac{6 \times 0.6^2 + 4 \times 0.4^2}{10} = 0.28$$

- The average/mean and its variance over three experiments:

$$mean = \frac{(mean_1 + mean_2 + mean_3)}{3} = 0.5; var = \frac{0.25 + 0.28 + 0.24}{3^2} = 0.085$$

Bagging: Ideas

- Invented by Leo & Breiman (1994)
- Sampling with replacement

Training Data

Data ID

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample
- Bagging reduces variance by averaging
- Bagging has little effect on bias

- Why sampling with replacement? Create more variability of data combination

Bagging Algorithm: Ideas

- Leo & Breiman (1994)
- Sampling with replacement

Training Data

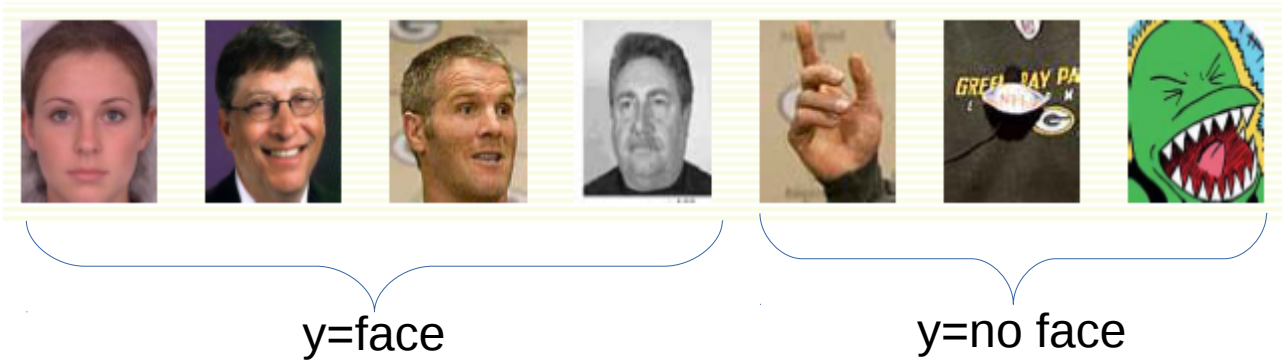
Data ID

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Bootstrap Aggregation (Bagging):
  - Train a classifier on each bootstrap sample
  - Use majority voting to determine the class label of ensemble classifier

Bagging: Algorithm

- **Input**: N labeled data (x_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)



Bagging : Formal Algorithm

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (e.g. size N) (by sampling from D with replacement)

Bagging : Formal Algorithm

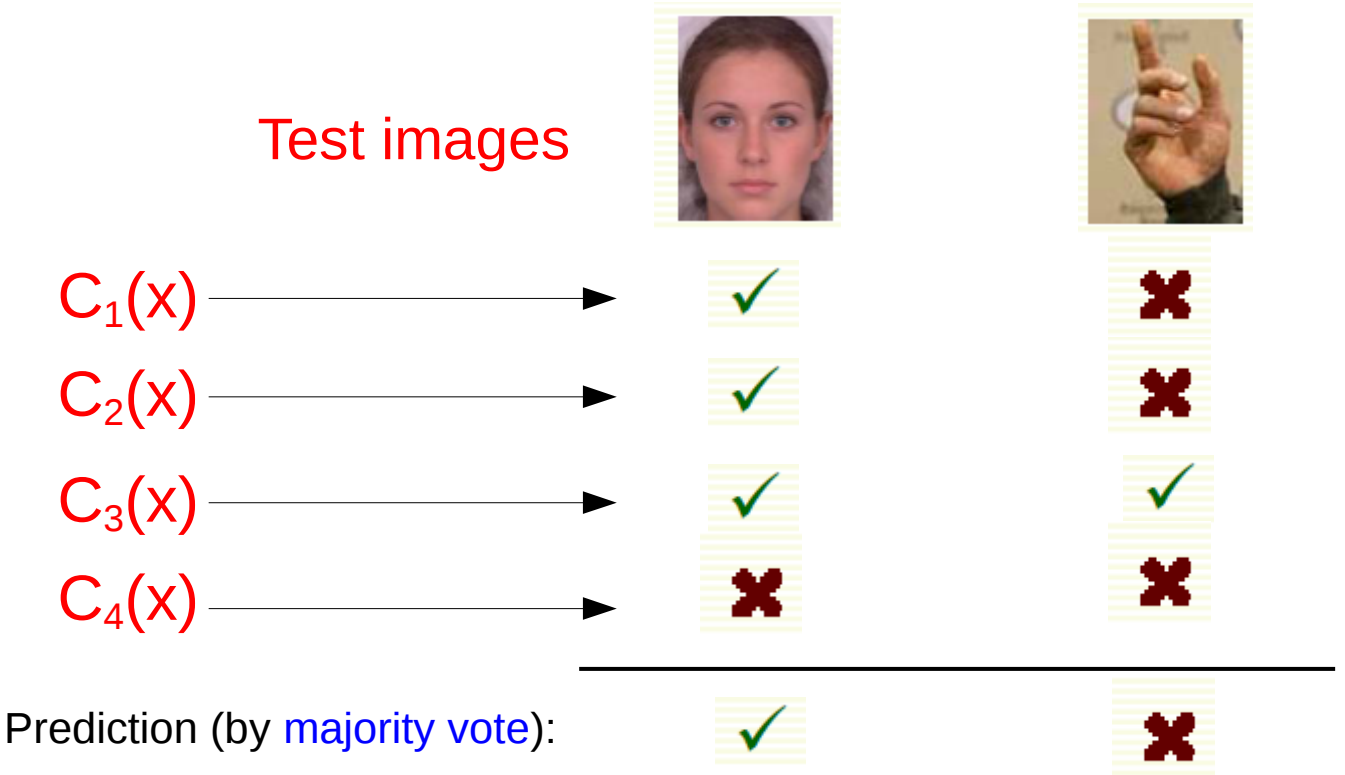
- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (e.g. size N) (by sampling from D with replacement)
    - Generate a classifier C_t using data D_t

$C_t(x_i)$ →

- Output: the set (C_1,C_2,C_3 , ..., C_T)

Bagging : Prediction

- **Given**: the set (C_1,C_2,C_3 , ..., C_T) returned from training
- **Prediction** for a new test point:



Bagging: The 0.632 bootstrap

- Bagging is also called the ***0.632 bootstrap***
  - A particular training data has a probability of 1-1/*n* of *not* being picked
  - Thus its probability of ending up in the test data (not selected) is:

$$\left[1-\frac{1}{n}\right]^{n} \approx e^{-1} = 0.368$$

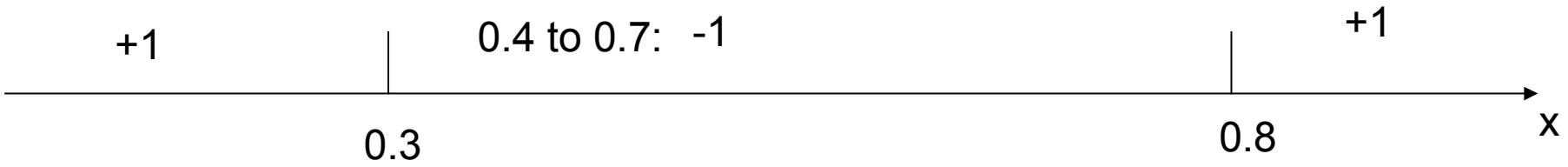
  - This means the training data will contain approximately 63.2% (1-0.368=0.632) of the instances

Euler's number: e =2.71828182845904523536028747135 27

Example of Bagging

Assume that the training data is:

D={0.1:+1; 0.2:+1; 0.3:+1; 0.4:-1; 0.5:-1; 0.6:-1; 0.7:-1; 0.8:+1; 0.9:+1;1:+1}



Goal: find a collection of **10** simple thresholding **classifiers** that collectively can classify correctly.

Each simple (base or weak) classifier is:

(x<=Delta → class = +1 or -1 depending on which value yields the lowest error; where Delta is determined by entropy minimization)

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = -1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

Bagging (applied to training data)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy of ensemble classifier: 100% ☺

Bagging- Summary

- Works well if the base classifiers are unstable (complement each other)
- Increased accuracy because it ***reduces the variance*** of the individual classifier
- Does not focus on any particular instance of the training data
  - Typically help when applied with an over-fitted base model

- Methods that reduce the bias: What if we want to focus on a particular instances of training data?

Outline

- Bias/variance trade-off

- Ensemble methods that minimize variance
  - Bagging

- **Ensemble methods that minimize bias and variance**
  - **Boosting**

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all n training records are assigned equal weights
  - Unlike Bagging, weights may change at the end of a boosting round.
  - Similar to Bagging, Boosting is also based on averaging, therefore it can reduce the variance.
  - Unlike Bagging, at each round a classifier is built based on the prediction errors in the previous round. Therefore it helps reduce the bias.

Boosting

- Records that are wrongly classified will have their weights increased
  - The higher the weight the "more attention" a training sample receives, i.e. via sampling with replacement.
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Boosting: Sampling with Replacement

- Sampling from the training set D **with replacement**, according to the sample distribution $D(x_i)$

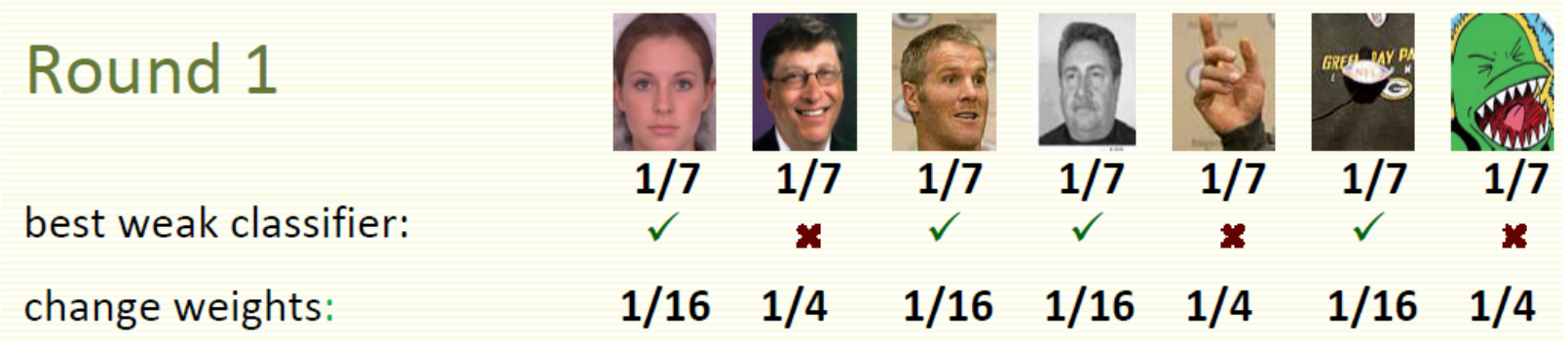
- Draw k samples, each x with probability equal to D(x):

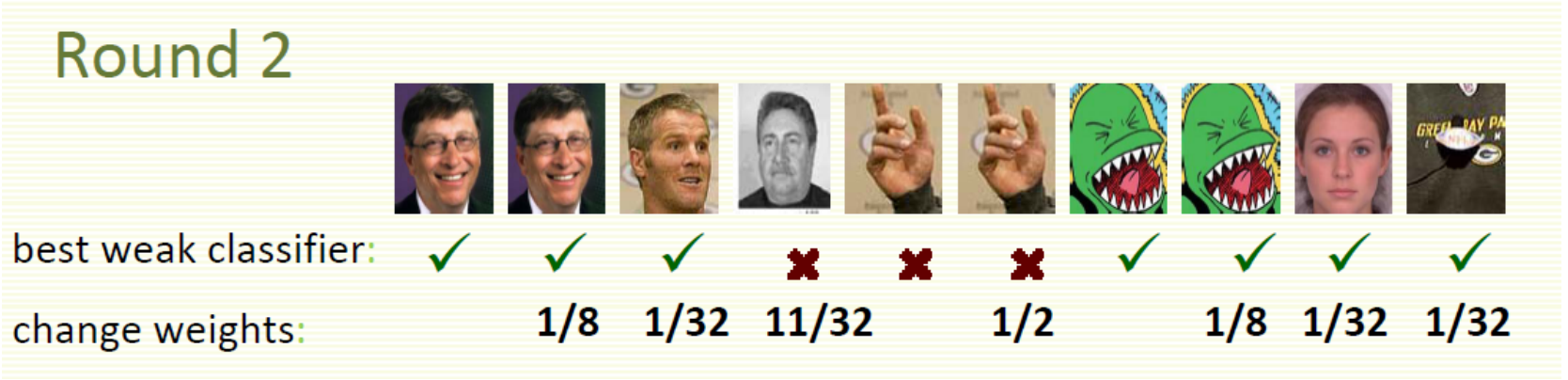
Boosting

- Equal weights are assigned to each training instance (1/N for round 1) at first
- After a classifier C_i is learned, the weights are adjusted to allow the subsequent classifier C_{i+1} to “pay more attention” to data that were misclassified by C_i.
- Final boosted classifier C* combines the votes of each individual classifier
  - Weight of each classifier’s vote is a function of its accuracy
- **AdaBoost (Adaptive Boosting) – Most popular boosting algorithm**

Boosting: Example

- Examples of high weight are shown more often at later rounds
- Face/nonface classification problem:





Olga Veksler

AdaBoost for Classification: Formal Algorithm

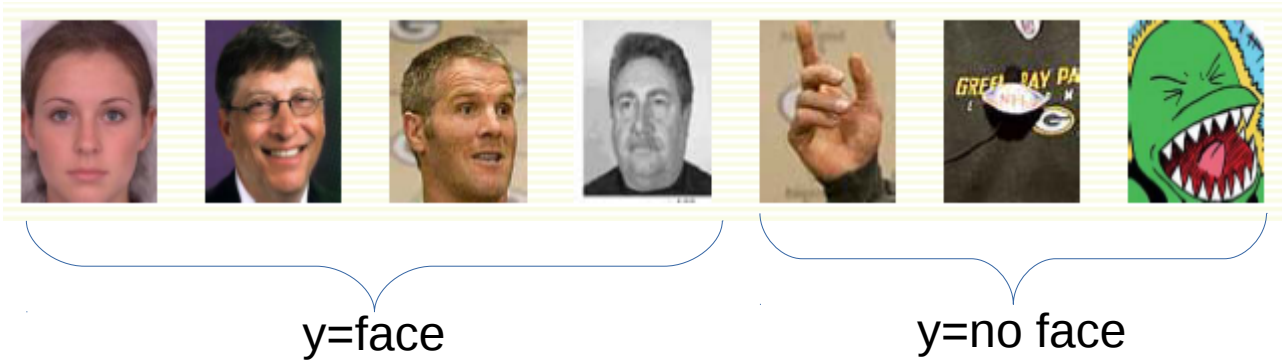
- **Input**:
  - Training set *D* containing *N* instances
  - *T* rounds (generate T classifiers)
  - *A* classification learning scheme (weak or base classifier)
- **Output**:
  - A composite model (an ensemble model)

AdaBoost Outline: Training Phase

- **Input**: Training data D contain N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initially assign equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T
    - Construct a dataset D_t (e.g. size *N*) (by sampling from D with replacement using the distribution $D_t(i)$)
    - Generate *a classifier* C_t *using data* D_t
    - *Update the distribution* $D_{t+1}(i)$ *from* $D_t(i)$ *by:*
      - *Increase if instance i misclassified by* C_t
      - *Decrease if instance i correctly classified by* C_t
- Each data's chance of being selected in the next rounds depends on its weight
  - Each time the new sample is generated directly from the training data D with different sampling probability according to the weights; these weights are not zero

AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (x_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)

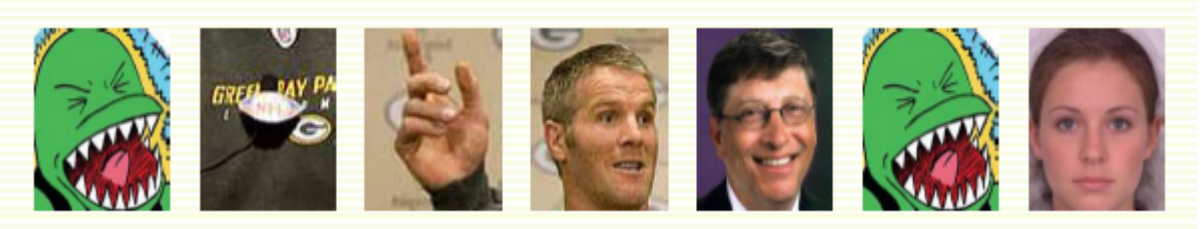


AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N

AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (size N) (by sampling from D with replacement using the distribution $D_t(i)$)



AdaBoost Algorithm: Training Phase

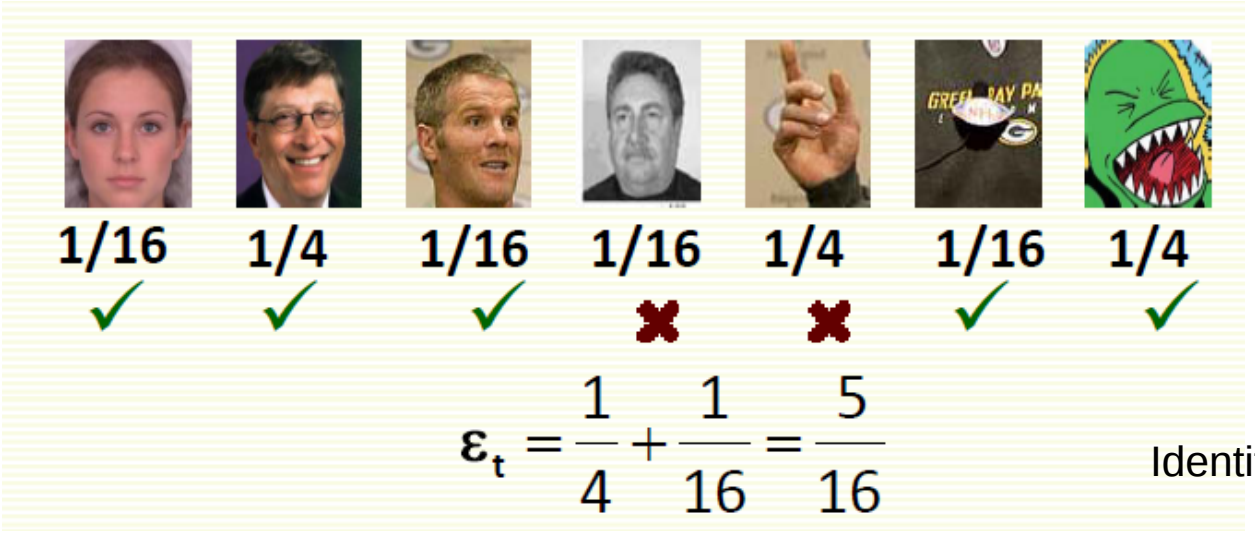
- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (e.g. size N) (by sampling from D with replacement using the distribution $D_t(i)$)
    - Generate a classifier C_t using data D_t

$C_t(X_i)$ →

Prediction/testing on the training set D_t

AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (e.g. size N) (by sampling from D with replacement using the distribution $D_t(i)$)
    - Generate a classifier C_t using data D_t
    - Compute the weighted training error: $\epsilon_t = \sum_{i=1}^{N} D_t(i) I(y_t \neq C_t(x_i))$



$$\varepsilon_t = \frac{1}{4} + \frac{1}{16} = \frac{5}{16}$$

Identity function: = 1 if true
= 0 otherwise

AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (size N) (by sampling from D with replacement using the distribution $D_t(i)$)
    - Generate a classifier C_t using data D_t
    - Compute the weighed training error: $\epsilon_t = \sum_{i=1}^{N} D_t(i) I(y_t \neq C_t(x_i))$
    - Set: $\alpha_t = \eta \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

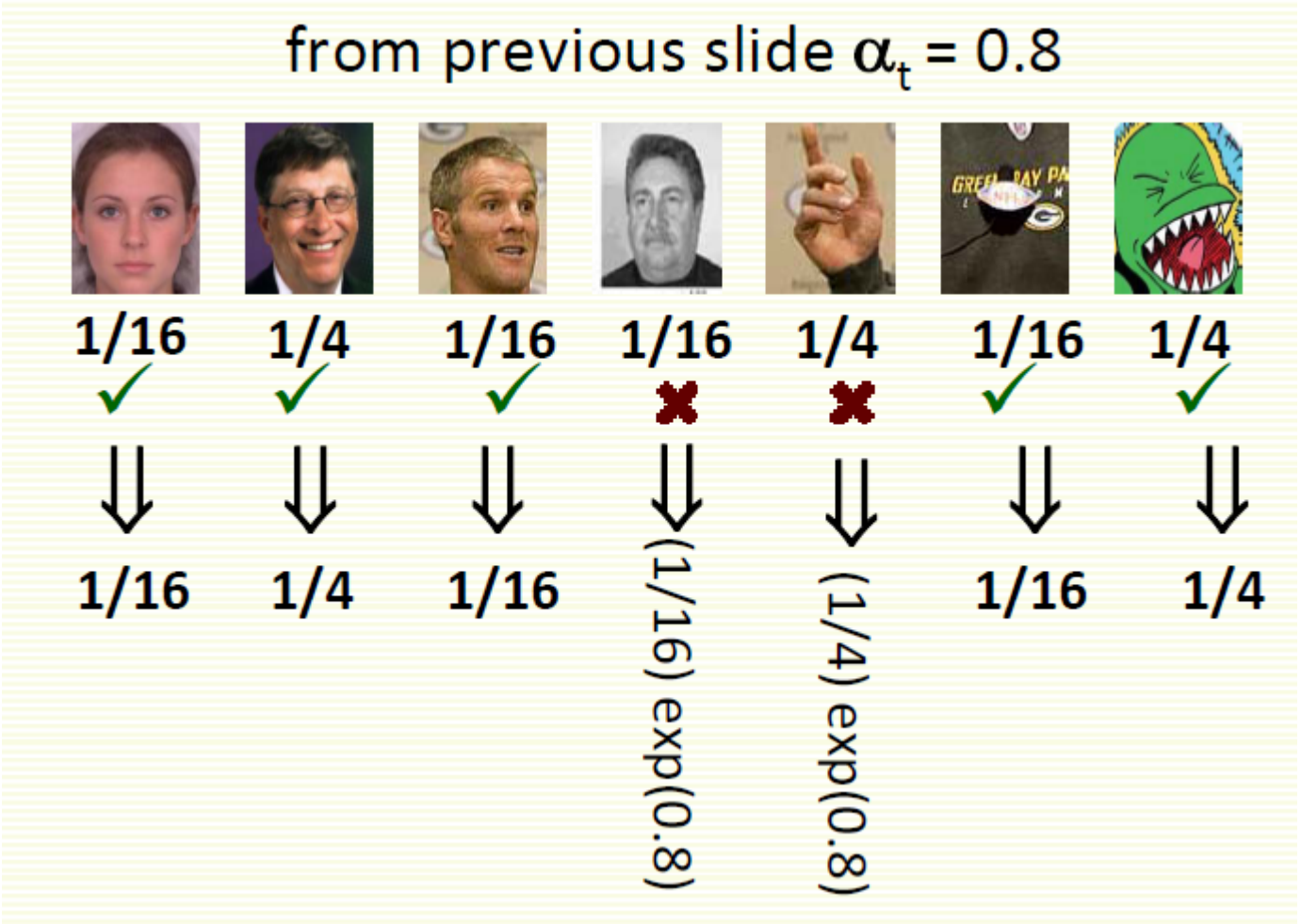
In example from previous slide:

$$\text{set}: \eta=1 \quad \varepsilon_t = \frac{5}{16} \quad \Rightarrow \quad \alpha_t = \log \frac{1-\frac{5}{16}}{\frac{5}{16}} = \log \frac{11}{5} \approx 0.8$$

AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (size N) (by sampling from D with replacement using the distribution $D_t(i)$)
    - Generate a classifier C_t using data D_t
    - Compute the weighed training error: $\epsilon_t = \sum_{i=1}^{N} D_t(i) I(y_t \neq C_t(x_i))$
    - Set: $\alpha_t = \eta \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    - Update the weights: $D_{t+1}(i) = D_t(i) \exp(\alpha_t I(y_i \neq C_t(x_i)))$

- Update the weights: $D_{t+1}(i) = D_t(i) \exp(\alpha_t I(y_i \neq C_t(x_i)))$



weights of misclassified examples is increased

AdaBoost Algorithm: Training Phase

- **Input**: N labeled data (X_1,y_1), (x_2,y_2), (x_3,y_3),….(x_N,y_N)
- **Training**:
  - Initialize equal weight $D_1(i)=1/N$ to each data, i=1,2,…,N
  - For each t =1, 2,…, T:
    - Construct a dataset D_t (size N) (by sampling from D with replacement using the distribution $D_t(i)$)
    - Generate a classifier C_t using data D_t
    - Compute the weighed training error: $\epsilon_t = \sum_{i=1}^{N} D_t(i) I(y_t \neq C_t(x_i))$
    - Set: $\alpha_t = \eta \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    - Update the weights: $D_{t+1}(i) = D_t(i) \exp(\alpha_t I(y_i \neq C_t(x_i)))$
    - Re-normalize the weights D_{t+1}: $D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_{j=1}^{N} D_{t+1}(j)}$

- Re-normalize the weights D_{t+1}: $$D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_{j=1}^{N} D_{t+1}(j)}$$

from previous slide:

1/16 1/4 1/16 0.14 0.56 1/16 1/4

after normalization

0.05 0.18 0.05 0.10 0.40 0.05 0.18

AdaBoost: Testing Phase

- The lower a classifier error rate, the more accurate it is, and therefore, the higher its weight for voting should be
- Weight of a classifier C_t's vote is

$$\alpha_t = \eta \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

- Testing/Prediction:
  - For each class k, sum the weights of each classifier that assigned class k to x test (unseen data)
  - The class with the highest sum is the WINNER! (using the principle of weighted majority voting)

$$C^*(x_{test}) = \underset{class\, k \in K}{\operatorname{argmax}} \sum_{t}^{T} \alpha_t I\left(C_t(x_{test}) = k\right)$$

We compute the sum for every class label in the set of class labels, and return as our final prediction the class k that gives the largest summation.

AdaBoost: Tuning Parameters

- The number of rounds T. Bagging does not over-fit as T increases. Boosting can over-fit! **Cross Validation**
- The hyper-parameter η, a small positive number, specifying the rate of change (smaller η means slower weight change).
- Base classifier: the classification learning scheme affects the convergence rate and computational time.

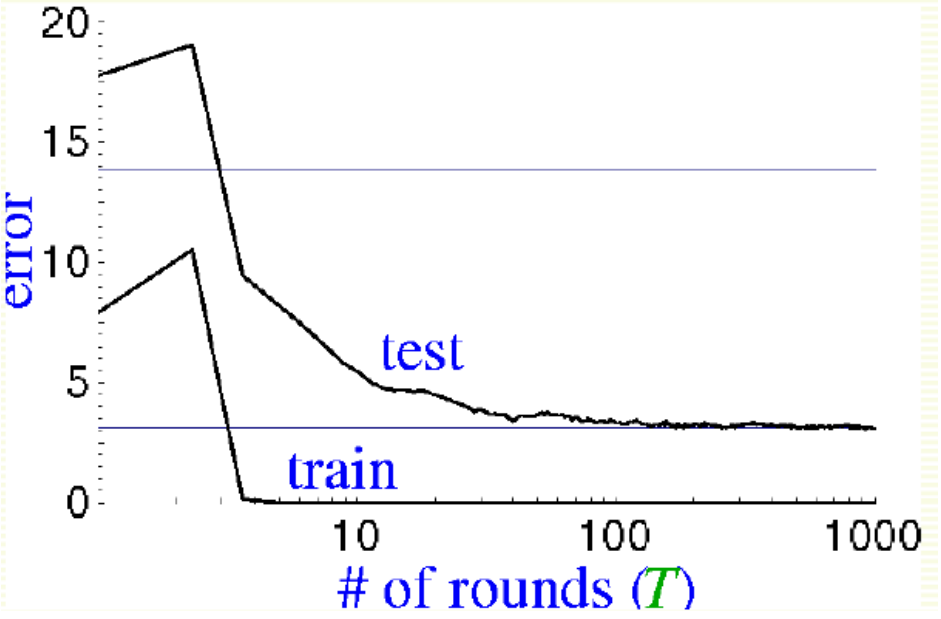
AdaBoost: Theoretical Properties

- It can be shown that the training error drops exponentially fast:

$$Training\ error \leq \exp\left(-2\sum_t \left(\epsilon_t - 0.5\right)^2\right)$$

- For example: if the first 5 rounds have following errors {0.3, 0.14, 0.06, 0.03, 0.01}

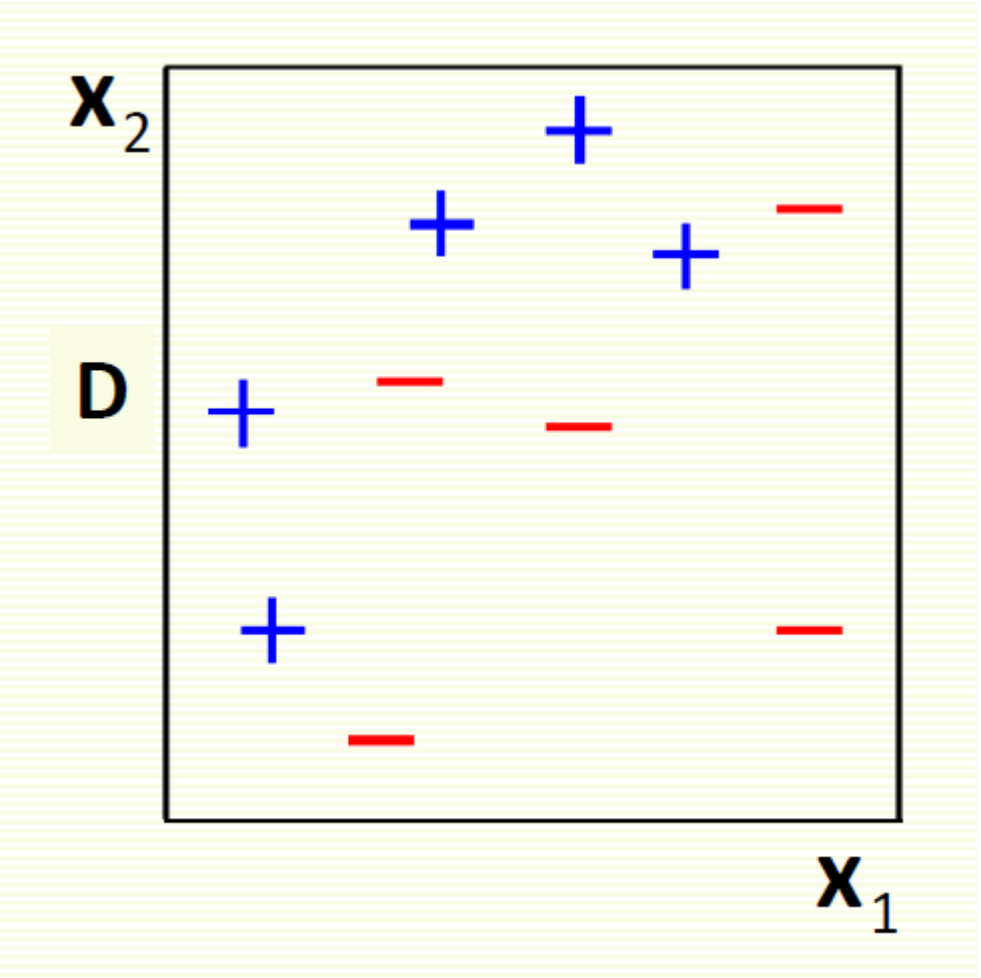
$$Training\ error \leq \exp\left(-2\left(0.2^2+0.36^2+0.44^2+0.47^2+0.49^2\right)\right) \approx 0.19$$



AdaBoost: Theoretical Properties

- We are really interested in the generalization properties of the final classifier, not the training error
- AdaBoost was shown to have excellent generalization properties in practice
  - the more rounds, the more complex is the final classifier, so overfitting is expected as the training proceeds
  - but in the beginning researchers observed no overfitting of the data
  - It turns out it does overfit data eventually, if you run it really long
- It can be shown that boosting increases the margins of training examples, as iterations proceed
  - larger margins help better generalization
  - margins continue to increase even when training error reaches zero
  - helps to explain empirically observed phenomena: test error continues to drop even after training error reaches zero

AdaBoost: Example

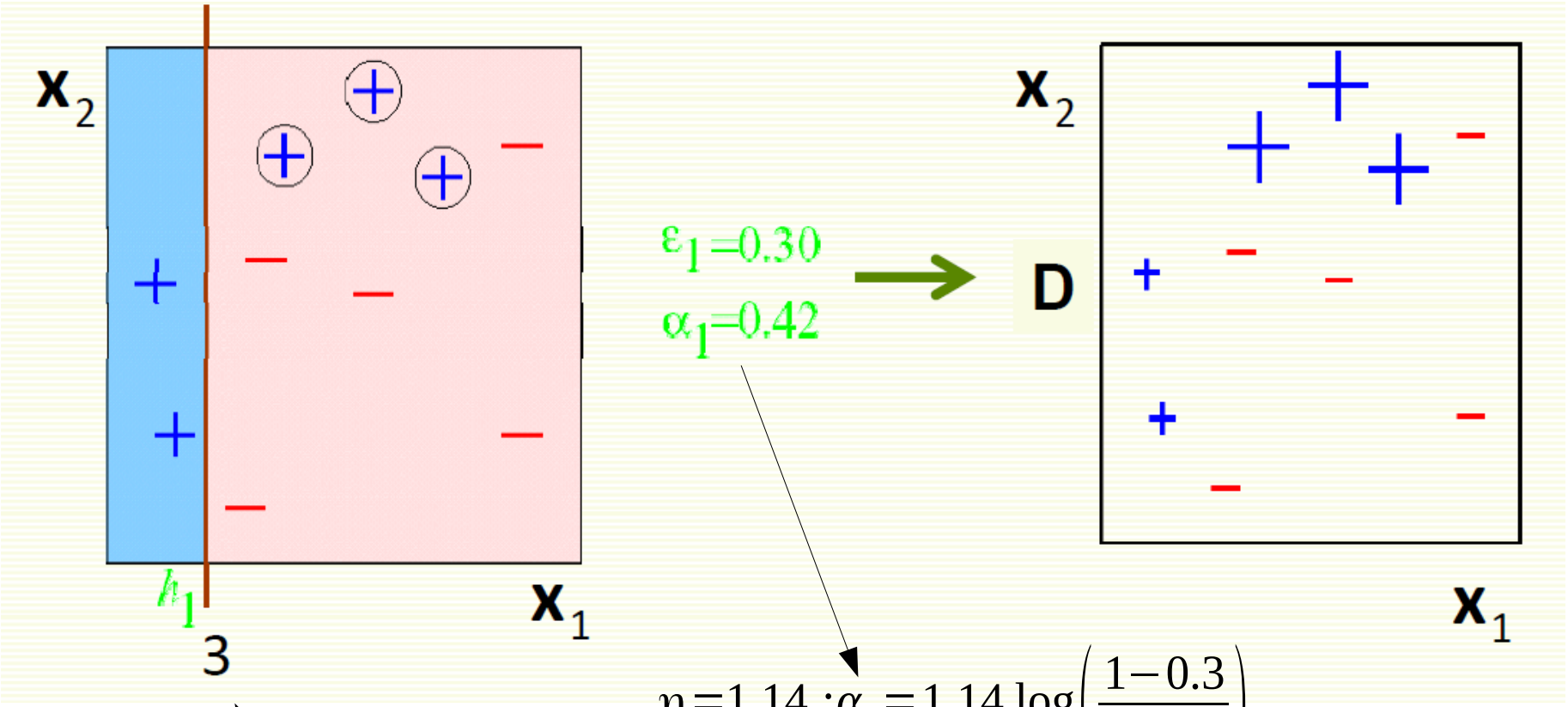


Initialization: all examples have equal weights

Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire

AdaBoost: Example

ROUND 1: Build classifier t = 1

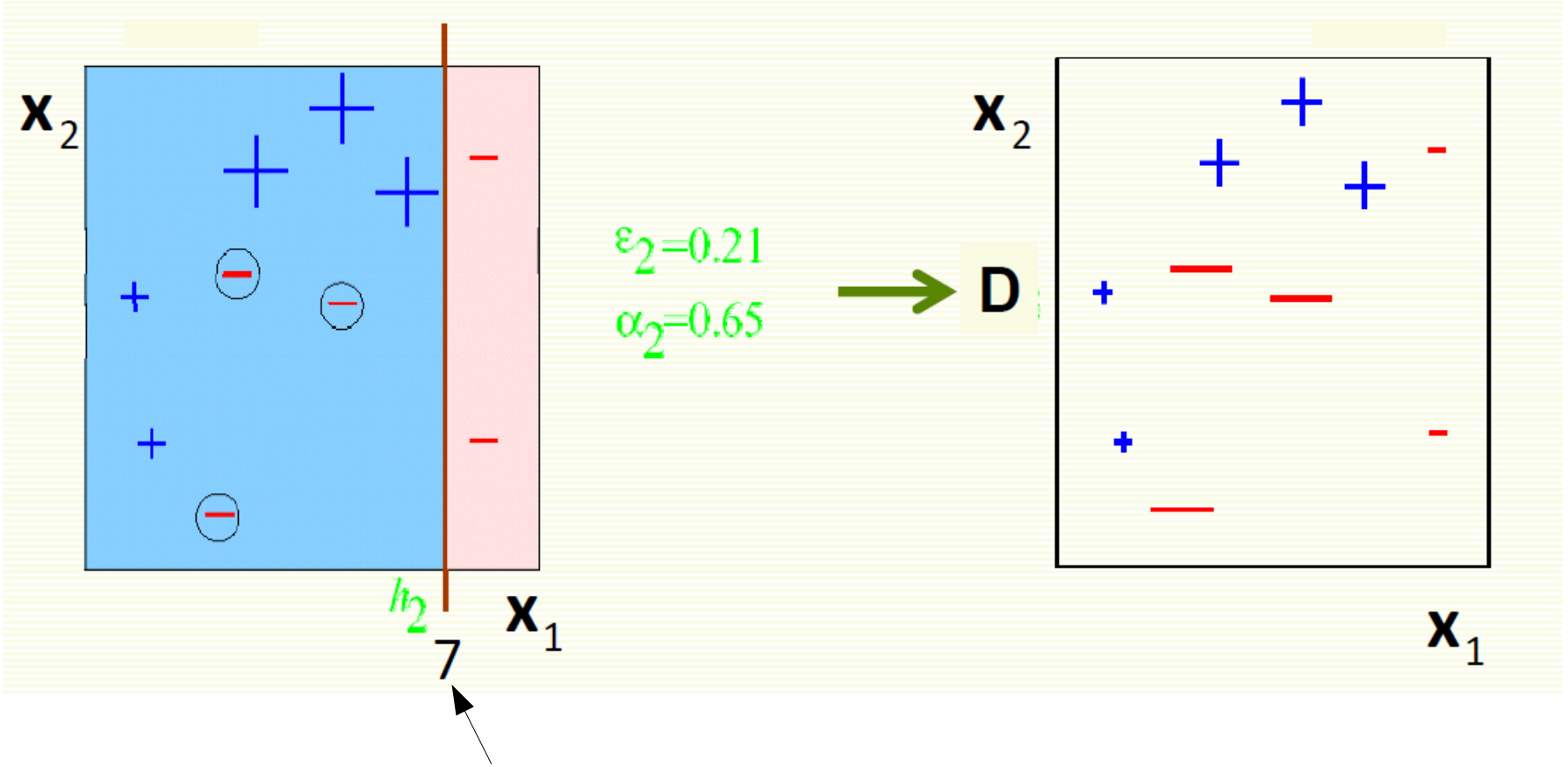


$$\eta = 1.14 \; ; \alpha_1 = 1.14 \log\left(\frac{1-0.3}{0.3}\right)$$

Classifier C_1 is a level function at h_1=3: $C_1(x) = sign(3 - x_1)$

AdaBoost: Example

ROUND 2: Build classifier t = 2



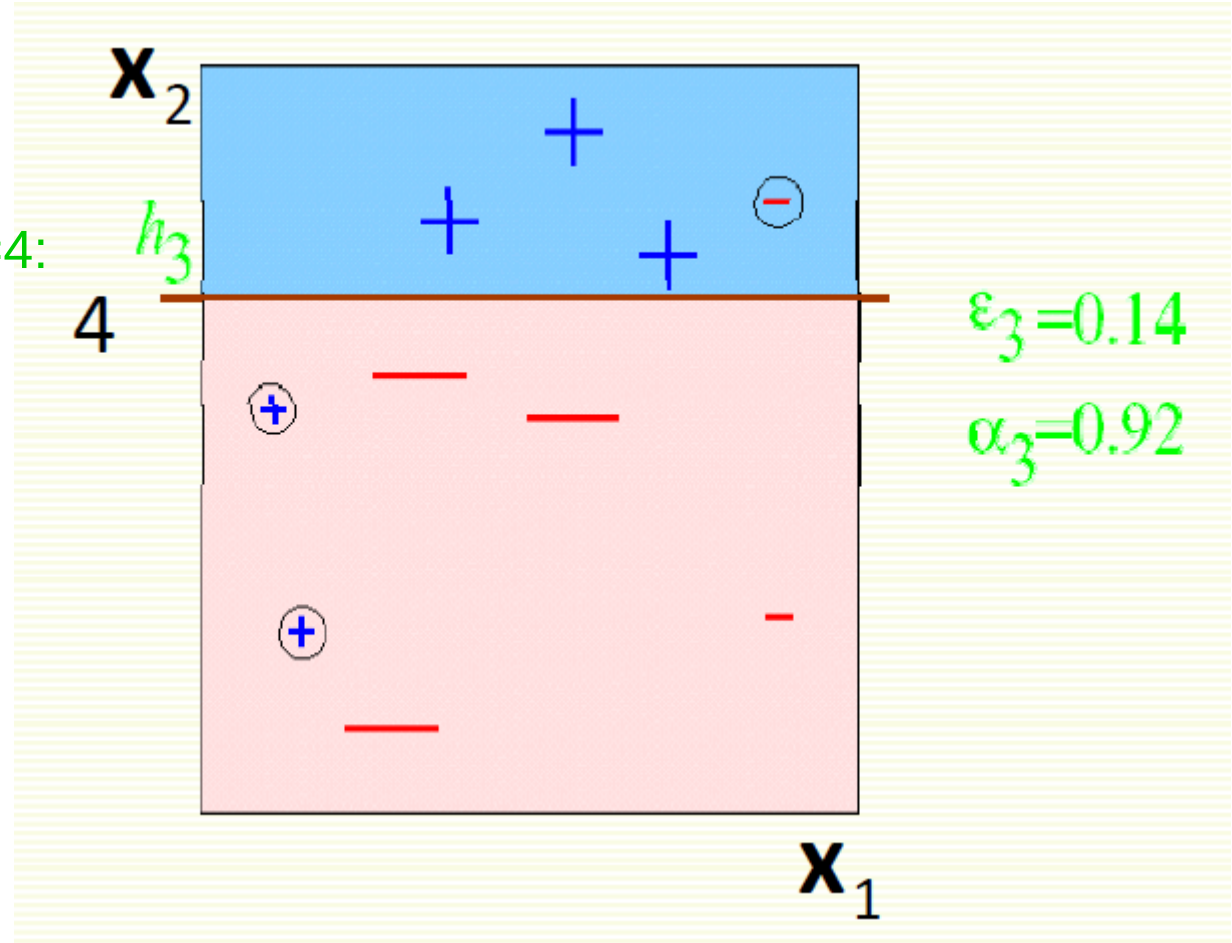
Classifier C_2 is a level function at h_2=7: $C_2(x) = sign(7 - x_1)$

AdaBoost: Example

ROUND 3: Build classifier t = 3

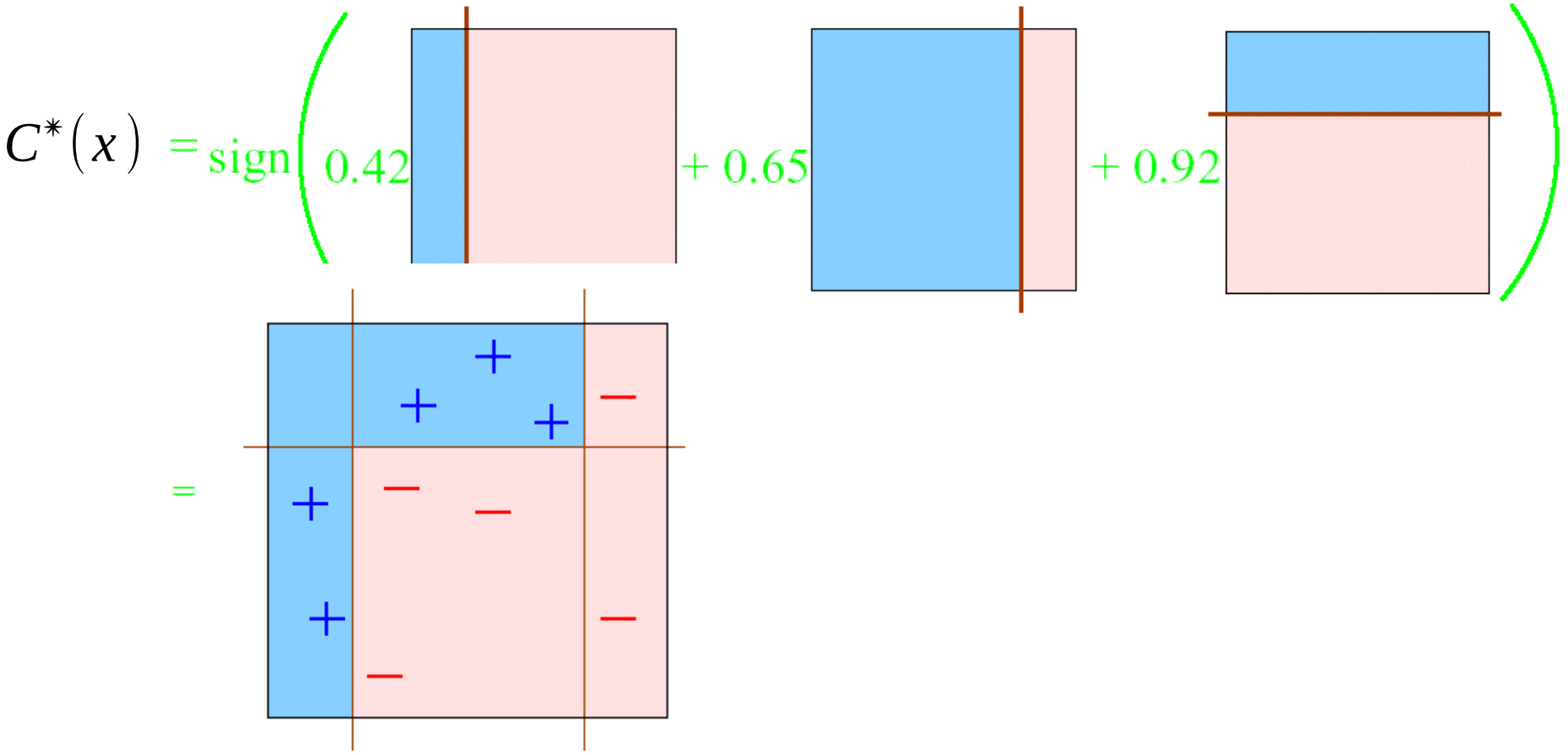
Classifier C_3 is a level function at h_3=4:

$$C_3(x) = sign(x_2 - 4)$$



AdaBoost: Example

- Testing/Prediction:



$$C^*(x) = sign\left(0.42\,sign\left(3-x_1\right)+0.65\,sign\left(7-x_1\right)+0.92\,sign\left(x_2-4\right)\right)$$

Note: This is a non-linear decision boundary

Boosting: Summary

- Boosting can:
  - Reduce variance (the same as Bagging)
  - But also to eliminate the effect of high bias of the weak learner (unlike Bagging)
- Train versus test errors performance:
  - Train errors can be driven close to 0
  - But test errors do not show over-fitting

Further Topics …

- Ensemble methods for regression
- Ensemble methods that minimize variance
  - **Bagging**
  - Random forest
- Ensemble methods that minimize bias
  - Functional gradient boosting
  - **Boosting**
  - Ensemble selection