

# Boosting

Yoav Freund

## Probably Approximately Correct (PAC) Learning

- ▶ Sample space  $X$  with a fixed but unknown distribution  $P$ .

## Probably Approximately Correct (PAC) Learning

- ▶ Sample space  $X$  with a fixed but unknown distribution  $P$ .
- ▶ Concept class  $C$ , with  $c \in C$  and  $c : X \rightarrow \{0, 1\}$ .

## Probably Approximately Correct (PAC) Learning

- ▶ Sample space  $X$  with a fixed but unknown distribution  $P$ .
- ▶ Concept class  $C$ , with  $c \in C$  and  $c : X \rightarrow \{0, 1\}$ .
- ▶ **Learning Input:** A training set  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  drawn i.i.d. according to  $P$ , labeled by some  $c \in C$ .

## Probably Approximately Correct (PAC) Learning

- ▶ Sample space  $X$  with a fixed but unknown distribution  $P$ .
- ▶ Concept class  $C$ , with  $c \in C$  and  $c : X \rightarrow \{0, 1\}$ .
- ▶ **Learning Input:** A training set  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  drawn i.i.d. according to  $P$ , labeled by some  $c \in C$ .
- ▶ **Learning Output:** A concept  $c'$  such that

$$P(c(x) \neq c'(x)) \leq \epsilon$$

## Probably Approximately Correct (PAC) Learning

- ▶ Sample space  $X$  with a fixed but unknown distribution  $P$ .
- ▶ Concept class  $C$ , with  $c \in C$  and  $c : X \rightarrow \{0, 1\}$ .
- ▶ **Learning Input:** A training set  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  drawn i.i.d. according to  $P$ , labeled by some  $c \in C$ .
- ▶ **Learning Output:** A concept  $c'$  such that

$$P(c(x) \neq c'(x)) \leq \epsilon$$

- ▶ **Strong PAC Learning** of  $C$ : There exists an algorithm such that for all  $\epsilon, \delta$ , the algorithm runs in time polynomial in  $1/\epsilon$  and  $1/\delta$  and outputs  $c'$  with

$$P(c(x) \neq c'(x)) \leq \epsilon$$

## Probably Approximately Correct (PAC) Learning

- ▶ Sample space  $X$  with a fixed but unknown distribution  $P$ .
- ▶ Concept class  $C$ , with  $c \in C$  and  $c : X \rightarrow \{0, 1\}$ .
- ▶ **Learning Input:** A training set  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  drawn i.i.d. according to  $P$ , labeled by some  $c \in C$ .
- ▶ **Learning Output:** A concept  $c'$  such that

$$P(c(x) \neq c'(x)) \leq \epsilon$$

- ▶ **Strong PAC Learning** of  $C$ : There exists an algorithm such that for all  $\epsilon, \delta$ , the algorithm runs in time polynomial in  $1/\epsilon$  and  $1/\delta$  and outputs  $c'$  with

$$P(c(x) \neq c'(x)) \leq \epsilon$$

- ▶ **Weak PAC Learning:** Same as strong PAC, but only required to hold for a single  $\epsilon < \frac{1}{2}$ .

# Boosting

- ▶ A boosting algorithm can translate a weak PAC learner into a strong pac learner.



# Boosting

- ▶ A boosting algorithm can translate a weak PAC learner into a strong pac learner.
- ▶ How it is done: by giving the weak learner different distributions.

## Zero sum games in matrix form

- ▶ Game between two players.

## Zero sum games in matrix form

- ▶ Game between two players.
- ▶ Defined by  $n \times m$  matrix  $\mathbf{M}$

## Zero sum games in matrix form

- ▶ Game between two players.
- ▶ Defined by  $n \times m$  matrix **M**
- ▶ **Row** player chooses  $i \in \{1, \dots, n\}$

## Zero sum games in matrix form

- ▶ Game between two players.
- ▶ Defined by  $n \times m$  matrix  $\mathbf{M}$
- ▶ **Row** player chooses  $i \in \{1, \dots, n\}$
- ▶ **Column** player chooses  $j \in \{1, \dots, m\}$

## Zero sum games in matrix form

- ▶ Game between two players.
- ▶ Defined by  $n \times m$  matrix  $\mathbf{M}$
- ▶ **Row** player chooses  $i \in \{1, \dots, n\}$
- ▶ **Column** player chooses  $j \in \{1, \dots, m\}$
- ▶ **Row** player gains  $\mathbf{M}(i, j) \in [0, 1]$

## Zero sum games in matrix form

- ▶ Game between two players.
- ▶ Defined by  $n \times m$  matrix  $\mathbf{M}$
- ▶ Row player chooses  $i \in \{1, \dots, n\}$
- ▶ Column player chooses  $j \in \{1, \dots, m\}$
- ▶ Row player gains  $\mathbf{M}(i, j) \in [0, 1]$
- ▶ Column player loses  $\mathbf{M}(i, j)$

## Zero sum games in matrix form

- ▶ Game between two players.
- ▶ Defined by  $n \times m$  matrix  $\mathbf{M}$
- ▶ Row player chooses  $i \in \{1, \dots, n\}$
- ▶ Column player chooses  $j \in \{1, \dots, m\}$
- ▶ Row player gains  $\mathbf{M}(i, j) \in [0, 1]$
- ▶ Column player loses  $\mathbf{M}(i, j)$
- ▶ Game repeated many times.



## Pure vs. mixed strategies

- ▶ Choosing a **single** action = **pure** strategy.

## Pure vs. mixed strategies

- ▶ Choosing a **single** action = **pure** strategy.
- ▶ Choosing a **Distribution** over actions = **mixed** strategy.

## Pure vs. mixed strategies

- ▶ Choosing a **single** action = **pure** strategy.
- ▶ Choosing a **Distribution** over actions = **mixed** strategy.
- ▶ **Row** player chooses dist. over rows **P**

## Pure vs. mixed strategies

- ▶ Choosing a **single** action = **pure** strategy.
- ▶ Choosing a **Distribution** over actions = **mixed** strategy.
- ▶ **Row** player chooses dist. over rows **P**
- ▶ **Column** player chooses dist. over columns **Q**

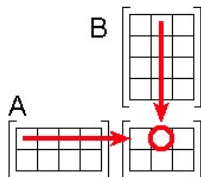
## Pure vs. mixed strategies

- ▶ Choosing a **single** action = **pure** strategy.
- ▶ Choosing a **Distribution** over actions = **mixed** strategy.
- ▶ **Row** player chooses dist. over rows **P**
- ▶ **Column** player chooses dist. over columns **Q**
- ▶ **Row** player gains  **$M(P, Q)$** .

## Pure vs. mixed strategies

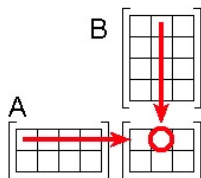
- ▶ Choosing a **single** action = **pure** strategy.
- ▶ Choosing a **Distribution** over actions = **mixed** strategy.
- ▶ **Row** player chooses dist. over rows **P**
- ▶ **Column** player chooses dist. over columns **Q**
- ▶ **Row** player gains  **$M(P, Q)$** .
- ▶ **Column** player loses  **$M(P, Q)$** .

# Mixed strategies in matrix notation



$$(A \times B)_{12} = \sum_{r=1}^4 a_{1r} b_{r2} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42}$$

# Mixed strategies in matrix notation

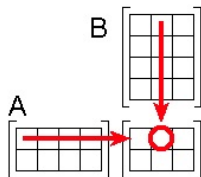


$$(A \times B)_{12} = \sum_{r=1}^4 a_{1r} b_{r2} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42}$$

**Q** is a **column** vector. **P**<sup>T</sup> is a row vector.



## Mixed strategies in matrix notation



$$(A \times B)_{12} = \sum_{r=1}^4 a_{1r} b_{r2} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42}$$

**Q** is a **column** vector. **P<sup>T</sup>** is a row vector.

$$\mathbf{M}(\mathbf{P}, \mathbf{Q}) = \mathbf{P}^T \mathbf{M} \mathbf{Q} = \sum_{i=1}^n \sum_{j=1}^m \mathbf{P}(i) \mathbf{M}(i, j) \mathbf{Q}(j)$$

# The minmax Theorem

When using pure strategies, second player has an advantage.

# The minmax Theorem

When using pure strategies, second player has an advantage.

John von Neumann, 1928.

$$\min_P \max_Q \mathbf{M}(\mathbf{P}, \mathbf{Q}) = \max_Q \min_P \mathbf{M}(\mathbf{P}, \mathbf{Q})$$

In words: for **mixed** strategies, choosing second gives no advantage.

## The learning game matrix

	Example 1	Example 2	Example 3
Rule 1	0	1	0
Rule 2	1	1	0
Rule 3	0	0	1
Rule 4	1	0	1
Rule 5	0	1	1

entries: 1 = rule is correct on example, 0= incorrect

## Boosting is implied by min/max theorem

- ▶ For any distribution  $Q$  over the examples there exists a row (rule) that is correct on  $\frac{1}{2} + \gamma$  of the (dist over the) examples.

## Boosting is implied by min/max theorem

- ▶ For any distribution **Q** over the examples there exists a row (rule) that is correct on  $\frac{1}{2} + \gamma$  of the (dist over the) examples.
- ▶ From min/max theorem we get that there exists a distribution **P** over the rules such that for any example at least  $\frac{1}{2} + \gamma$  of the rules are correct.

## Boosting is implied by min/max theorem

- ▶ For any distribution **Q** over the examples there exists a row (rule) that is correct on  $\frac{1}{2} + \gamma$  of the (dist over the) examples.
- ▶ From min/max theorem we get that there exists a distribution **P** over the rules such that for any example at least  $\frac{1}{2} + \gamma$  of the rules are correct.
- ▶ The weighted majority is always correct.

## Boosting is implied by min/max theorem

- ▶ For any distribution **Q** over the examples there exists a row (rule) that is correct on  $\frac{1}{2} + \gamma$  of the (dist over the) examples.
- ▶ From min/max theorem we get that there exists a distribution **P** over the rules such that for any example at least  $\frac{1}{2} + \gamma$  of the rules are correct.
- ▶ The weighted majority is always correct.
- ▶ Existence proof, but not an algorithm.



## Schapire's boosting algorithm

Calls the weak learner 3 times, on 3 different distributions, combines the rules using a majority.

The distributions are

- ▶  $h_1$ : use the training set as is.

## Schapire's boosting algorithm

Calls the weak learner 3 times, on 3 different distributions, combines the rules using a majority.

The distributions are

- ▶  $h_1$ : use the training set as is.
- ▶  $h_2$ : Filter examples so that  $P(h_1(x) = c(x)) = \frac{1}{2}$

## Schapire's boosting algorithm

Calls the weak learner 3 times, on 3 different distributions, combines the rules using a majority.

The distributions are

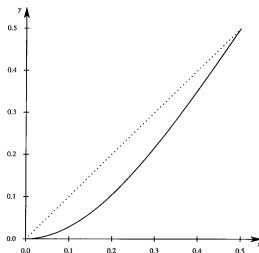
- ▶  $h_1$ : use the training set as is.
- ▶  $h_2$ : Filter examples so that  $P(h_1(x) = c(x)) = \frac{1}{2}$
- ▶  $h_3$  Filter out examples such that  $h_1(x) = h_2(x)$

## Idea of proof

- ▶ If errors of weak rules are at most  $x < 1/2$  then error of combined rule is at most  $3x^2 - 2x^3$ .

## Idea of proof

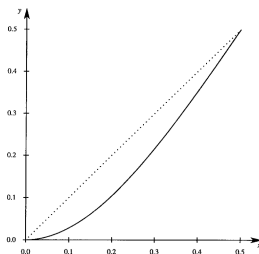
- If errors of weak rules are at most  $x < 1/2$  then error of combined rule is at most  $3x^2 - 2x^3$ .



- Figure 1. A graph of the function  $g(x) = 3x^2 - 2x^3$ .

## Idea of proof

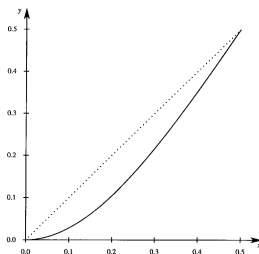
- ▶ If errors of weak rules are at most  $x < 1/2$  then error of combined rule is at most  $3x^2 - 2x^3$ .



- ▶ *Figure 1.* A graph of the function  $g(x) = 3x^2 - 2x^3$ .
- ▶ Let the available rules have error  $\frac{1}{2} - \gamma$  and assume we want a rule whose error is  $\epsilon$ .

## Idea of proof

- ▶ If errors of weak rules are at most  $x < 1/2$  then error of combined rule is at most  $3x^2 - 2x^3$ .



- ▶ *Figure 1.* A graph of the function  $g(x) = 3x^2 - 2x^3$ .
- ▶ Let the available rules have error  $\frac{1}{2} - \gamma$  and assume we want a rule whose error is  $\epsilon$ .
- ▶ Using 3-combiner recursively for depth at most  $O(\frac{1}{\gamma^2} \log \frac{1}{\epsilon})$  achieves the error  $\epsilon$ .

# Boost By Majority

Majority vote over many weak rules, rather than 3.

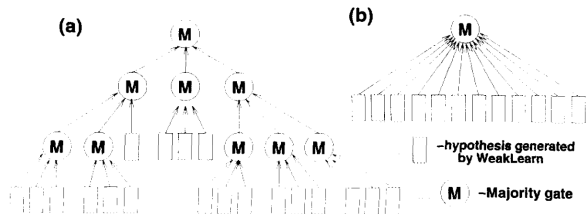


FIG. 1. Final concepts structure: (a) Schapire, (b) a one-layer majority circuit.



# Game between booster and learner

- Booster chooses distribution over examples.

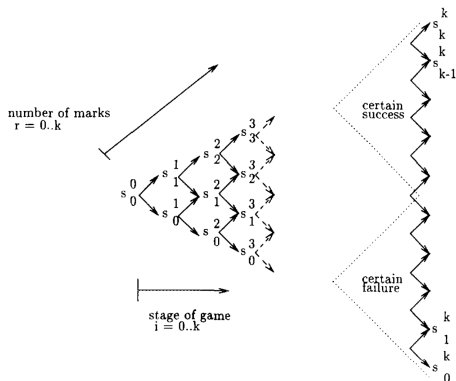


FIG. 2. Transitions between consecutive partitions.

## Game between booster and learner

- ▶ Booster chooses distribution over examples.
- ▶ Weak learner chooses where weak rules makes a mistake.

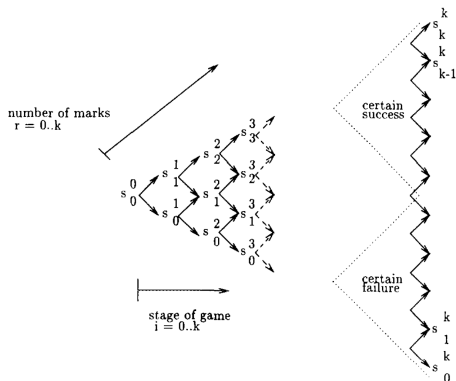


FIG. 2. Transitions between consecutive partitions.

## Game between booster and learner

- ▶ Booster chooses distribution over examples.
- ▶ Weak learner chooses where weak rules makes a mistake.
- ▶ Weak learner constrained to make weighted error smaller than  $(1/2) - \gamma$

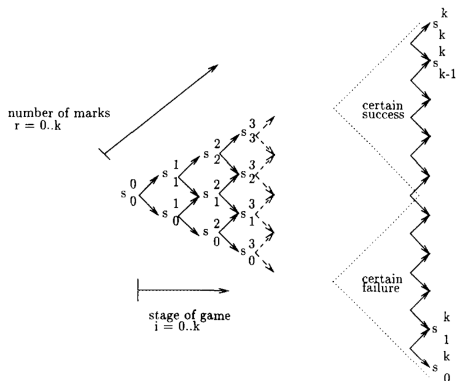


FIG. 2. Transitions between consecutive partitions.

## Potential function

loss set and which are in the reward set; it is reasonable to define the potential for  $i = k$  as

$$\beta_r^k = \begin{cases} 0 & \text{if } r > \frac{k}{2} \\ 1 & \text{if } r \leq \frac{k}{2}. \end{cases} \quad (4)$$

For  $i < k$  we define the potential recursively:

$$\beta_r^i = (\tfrac{1}{2} - \gamma) \beta_r^{i+1} + (\tfrac{1}{2} + \gamma) \beta_{r+1}^{i+1}. \quad (5)$$

## Weight function

The weighting factor is defined inductively as

$$\alpha_r^{k-1} = \begin{cases} 1 & \text{if } r = \left\lfloor \frac{k}{2} \right\rfloor \\ 0 & \text{otherwise.} \end{cases}$$

and for  $0 \leq i \leq k-2$ ,

$$\alpha_r^i = \left(\frac{1}{2} - \gamma\right) \alpha_r^{i+1} + \left(\frac{1}{2} + \gamma\right) \alpha_{r+1}^{i+1}.$$

## Potential is non increasing

- ▶ Let  $q_r^i$  be the fraction of the examples that have  $r$  mistakes on iteration  $i$ .

## Potential is non increasing

- ▶ Let  $q_r^i$  be the fraction of the examples that have  $r$  mistakes on iteration  $i$ .
- ▶ Then if the booster uses the weights  $\alpha_r^i$  then

$$\beta_0^0 > \sum_{r=0}^1 q_r^1 \beta_r^1 > \sum_{r=0}^2 q_r^2 \beta_r^2 > \cdots > \sum_{r=0}^k q_r^k \beta_r^k.$$

## Potential is non increasing

- ▶ Let  $q_r^i$  be the fraction of the examples that have  $r$  mistakes on iteration  $i$ .
- ▶ Then if the booster uses the weights  $\alpha_r^i$  then

$$\beta_0^0 > \sum_{r=0}^1 q_r^1 \beta_r^1 > \sum_{r=0}^2 q_r^2 \beta_r^2 > \cdots > \sum_{r=0}^k q_r^k \beta_r^k.$$

- ▶ Idea of proof, consider last and next to last steps.



## Error bound

Given a weak learner with error  $(1/2) - \gamma$ , find  $k$  that satisfies

$$\sum_{j=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{j} \left(\frac{1}{2} + \gamma\right)^j \left(\frac{1}{2} - \gamma\right)^{k-j} \leq \epsilon.$$

Then running Boost-by-majority for  $k$  iterations will generate a rule with error at most  $\epsilon$ .

# Adaboost

## Algorithm AdaBoost (Setup)

### Input:

- ▶ Sequence of  $N$  labeled examples  $\langle (x_1, y_1), \dots, (x_N, y_N) \rangle$

### Initialize:

$$w_i^1 = \frac{1}{N} \quad \text{for } i = 1, \dots, N.$$

## Algorithm AdaBoost (Setup)

### Input:

- ▶ Sequence of  $N$  labeled examples  $\langle (x_1, y_1), \dots, (x_N, y_N) \rangle$
- ▶ Weak learning algorithm *WeakLearn*

### Initialize:

$$w_i^1 = \frac{1}{N} \quad \text{for } i = 1, \dots, N.$$

## Algorithm AdaBoost (Setup)

### Input:

- ▶ Sequence of  $N$  labeled examples  $\langle (x_1, y_1), \dots, (x_N, y_N) \rangle$
- ▶ Weak learning algorithm *WeakLearn*
- ▶ Integer  $T$  specifying number of iterations

### Initialize:

$$w_i^1 = \frac{1}{N} \quad \text{for } i = 1, \dots, N.$$

## Algorithm AdaBoost (Main Loop)

**For**  $t = 1, 2, \dots, T$ :

1.  $p^t = \frac{w^t}{\sum_{i=1}^N w_i^t} \cdot$

## Algorithm AdaBoost (Main Loop)

**For**  $t = 1, 2, \dots, T$ :

1.  $p^t = \frac{w^t}{\sum_{i=1}^N w_i^t}$ .
2. Call *WeakLearn*, providing the distribution  $p^t$ . Get back a hypothesis  $h_t : X \rightarrow \{0, 1\}$ .

## Algorithm AdaBoost (Main Loop)

**For**  $t = 1, 2, \dots, T$ :

1.  $p^t = \frac{w^t}{\sum_{i=1}^N w_i^t}$ .
2. Call *WeakLearn*, providing the distribution  $p^t$ . Get back a hypothesis  $h_t : X \rightarrow \{0, 1\}$ .
3. Calculate the error of  $h_t$ :

$$\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|.$$



## Algorithm AdaBoost (Main Loop)

**For**  $t = 1, 2, \dots, T$ :

1.  $p^t = \frac{w^t}{\sum_{i=1}^N w_i^t}$ .
2. Call *WeakLearn*, providing the distribution  $p^t$ . Get back a hypothesis  $h_t : X \rightarrow \{0, 1\}$ .
3. Calculate the error of  $h_t$ :

$$\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|.$$

4. Set

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}.$$

## Algorithm AdaBoost (Main Loop)

**For**  $t = 1, 2, \dots, T$ :

1.  $p^t = \frac{w^t}{\sum_{i=1}^N w_i^t}$ .
2. Call *WeakLearn*, providing the distribution  $p^t$ . Get back a hypothesis  $h_t : X \rightarrow \{0, 1\}$ .
3. Calculate the error of  $h_t$ :

$$\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|.$$

4. Set

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}.$$

5. Update the weights:

$$w_i^{t+1} = w_i^t \beta_t^{(1 - |h_t(x_i) - y_i|)}.$$

## Algorithm AdaBoost (Final Output)

**Output the final hypothesis**  $h_{final}$ , defined by:

$$h_{final}(x) = \begin{cases} 1, & \text{if } \sum_{t=1}^T (\ln \frac{1}{\beta_t}) h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \ln \frac{1}{\beta_t}, \\ 0, & \text{otherwise.} \end{cases}$$

# Main Theorem

**Theorem 6** Suppose the weak learning algorithm **WeakLearn**, when called by **AdaBoost**, generates hypotheses with errors  $\epsilon_1, \dots, \epsilon_T$ . Then the error  $\epsilon = \frac{1}{N} \# [h_{final}(x_i) \neq y_i]$  of the final hypothesis  $h_{final}$  is bounded above by

$$\epsilon \leq 2^T \prod_{t=1}^T \sqrt{\epsilon_t (1 - \epsilon_t)}.$$

## Upper bound on total weight

$$\begin{aligned}\sum_{i=1}^N w_i^{t+1} &= \sum_{i=1}^N w_i^t \beta_t^{(1 - |h_t(x_i) - y_i|)} \\ &\leq \sum_{i=1}^N w_i^t \left(1 - (1 - \beta_t)(1 - |h_t(x_i) - y_i|)\right) \\ &\leq \left(\sum_{i=1}^N w_i^t\right) \left(1 - (1 - \epsilon_t)(1 - \beta_t)\right).\end{aligned}$$

## Combining over iterations

Combining the weight-update inequality over  $t = 1, \dots, T$ , we get

$$\sum_{i=1}^N w_i^{T+1} \leq \prod_{t=1}^T \left( 1 - (1 - \epsilon_t)(1 - \beta_t) \right). \quad (16)$$

## Lower bound on total weight

The final hypothesis  $h_{final}$  makes a mistake on instance  $i$  only if

$$\prod_{t=1}^T \beta_t^{(1-|h_t(x_i)-y_i|)} \geq \left(\prod_{t=1}^T \beta_t\right)^{-\frac{1}{2}}. \quad (17)$$

The final weight of instance  $i$  is

$$w_i^{T+1} = D(i) \prod_{t=1}^T \beta_t^{(1-|h_t(x_i)-y_i|)}. \quad (18)$$

By comparing the sum of all final weights to those on examples where  $h_{final}$  is incorrect, one obtains

$$\sum_{i=1}^N w_i^{T+1} \geq \sum_{i: h_{final}(x_i) \neq y_i} w_i^{T+1} \geq e \left(\prod_{t=1}^T \beta_t\right)^{1/2},$$

where  $e$  is the error of  $h_{final}$ .

## Resulting Error Bound

Combining (16) and the above,

$$e \leq \prod_{t=1}^T \frac{1 - (1 - \epsilon_t)(1 - \beta_t)}{\sqrt{\beta_t}}. \quad (20)$$

Minimizing each factor leads to  $\beta_t = \epsilon_t / (1 - \epsilon_t)$ . Plugging back yields

$$e \leq 2^T \prod_{t=1}^T \sqrt{\epsilon_t (1 - \epsilon_t)},$$



## Alternative forms of the bound

$$\begin{aligned} e &\leq \prod_{t=1}^T \sqrt{1 - 4\gamma_t^2} = \exp\left(-\sum_{t=1}^T \text{KL}\left(\frac{1}{2} \parallel \frac{1}{2} - \gamma_t\right)\right) \\ &\leq \exp\left(-2\sum_{t=1}^T \gamma_t^2\right). \end{aligned}$$

# Comparing Hedge vs Adaboost

# Comparing Hedge vs Adaboost

## Hedge

# Comparing Hedge vs Adaboost

## Hedge

- ▶ Each iteration adds an Example

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.
- ▶ Lower bound on score: Loss of best expert

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.
- ▶ Lower bound on score: Loss of best expert

### Adaboost



## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.
- ▶ Lower bound on score: Loss of best expert

### Adaboost

- ▶ Each iteration adds a Weak Rule

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.
- ▶ Lower bound on score: Loss of best expert

### Adaboost

- ▶ Each iteration adds a Weak Rule
- ▶ Weights assigned to examples.

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.
- ▶ Lower bound on score: Loss of best expert

### Adaboost

- ▶ Each iteration adds a Weak Rule
- ▶ Weights assigned to examples.
- ▶ Upper bound on Score: Edges of weak rules.

## Comparing Hedge vs Adaboost

### Hedge

- ▶ Each iteration adds an Example
- ▶ Weights assigned to Experts
- ▶ Upper bound on score: Loss of alg.
- ▶ Lower bound on score: Loss of best expert

### Adaboost

- ▶ Each iteration adds a Weak Rule
- ▶ Weights assigned to examples.
- ▶ Upper bound on Score: Edges of weak rules.
- ▶ Lower bound on Score: Error of majority vote.

## General Concepts

- ▶  $T$  step game, final configuration associated with a real value.

## General Concepts

- ▶  $T$  step game, final configuration associated with a real value.
- ▶ The **potential** of an configuration is an approximation/upper bound/lower bound of the final value given the current configuration.

## General Concepts

- ▶  $T$  step game, final configuration associated with a real value.
- ▶ The **potential** of an configuration is an approximation/upper bound/lower bound of the final value given the current configuration.
- ▶ The Weight is the difference between potential of reachable next steps.

## General Concepts

- ▶  $T$  step game, final configuration associated with a real value.
- ▶ The **potential** of an configuration is an approximation/upper bound/lower bound of the final value given the current configuration.
- ▶ The Weight is the difference between potential of reachable next steps.
- ▶ similar to board evaluation in Chess.



## General Concepts

- ▶  $T$  step game, final configuration associated with a real value.
- ▶ The **potential** of an configuration is an approximation/upper bound/lower bound of the final value given the current configuration.
- ▶ The Weight is the difference between potential of reachable next steps.
- ▶ similar to board evaluation in Chess.
- ▶ Our special case: state is a vector, Potential is defined as

$$\Phi(\mathbf{u}) = \psi \left( \sum_{i=1}^n \phi(u_i) \right)$$

## General Concepts

- ▶  $T$  step game, final configuration associated with a real value.
- ▶ The **potential** of an configuration is an approximation/upper bound/lower bound of the final value given the current configuration.
- ▶ The Weight is the difference between potential of reachable next steps.
- ▶ similar to board evaluation in Chess.
- ▶ Our special case: state is a vector, Potential is defined as

$$\Phi(\mathbf{u}) = \psi \left( \sum_{i=1}^n \phi(u_i) \right)$$

- ▶ I will (mis)use the word **potential** to refer to  $\phi$  and **Score** to refer to  $\Phi$

# AdaBoost: The symmetric version

- **Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$   
**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$

## AdaBoost: The symmetric version

- ▶ **Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$   
**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$
- ▶ **For**  $t = 1, \dots, T$ :

## AdaBoost: The symmetric version

- ▶ **Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$   
**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$
- ▶ **For**  $t = 1, \dots, T$ :
  - ▶ Train weak learner using dist.  $D_t$  to get  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$

## AdaBoost: The symmetric version

- ▶ **Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$   
**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$
- ▶ **For**  $t = 1, \dots, T$ :
  - ▶ Train weak learner using dist.  $D_t$  to get  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$
  - ▶ Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$

## AdaBoost: The symmetric version

- ▶ **Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$   
**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$
- ▶ **For**  $t = 1, \dots, T$ :
  - ▶ Train weak learner using dist.  $D_t$  to get  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$
  - ▶ Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
  - ▶ Update, for  $i = 1, \dots, m$ :

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

## AdaBoost: The symmetric version

- ▶ **Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$   
**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$
- ▶ **For**  $t = 1, \dots, T$ :
  - ▶ Train weak learner using dist.  $D_t$  to get  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$
  - ▶ Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
  - ▶ Update, for  $i = 1, \dots, m$ :

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

- ▶ **Output** the final hypothesis:

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



# show boosting examples

# Concepts when analyzing boosting

- ▶ booster tries to minimize Potential, adversary - to maximize.

## Concepts when analyzing boosting

- ▶ booster tries to minimize Potential, adversary - to maximize.
- ▶ **Adaboost: Configuration:** Margins  $m(x) = yH(x)$ : larger than zero = correct.

$$\Phi(x_1, \dots, x_n) = \sum_i \exp(-m(x_i))$$

## Concepts when analyzing boosting

- ▶ booster tries to minimize Potential, adversary - to maximize.
- ▶ **Adaboost: Configuration:** Margins  $m(x) = yH(x)$ : larger than zero = correct.

$$\Phi(x_1, \dots, x_n) = \sum_i \exp(-m(x_i))$$

- ▶ **Boost by majority: Configuration** = number of marks on each example

$$\Phi(x_1, \dots, x_n) = \sum_j \beta_{r(x_j)}^j$$

$$\beta_r^i = \sum_{j=0}^{\lfloor k/2 \rfloor - r} \binom{k-i}{j} \left(\frac{1}{2} + \gamma\right)^j \left(\frac{1}{2} - \gamma\right)^{k-1-j}$$

- **Weight assigned to examples:** The gradient of the potential.

$$\nu = \frac{\partial}{\partial x_i} \Phi(x_1, \dots, x_n) = \frac{\partial}{\partial x_i} \phi(x_i)$$

- **Weight assigned to examples:** The gradient of the potential.

$$\nu = \frac{\partial}{\partial x_i} \Phi(x_1, \dots, x_n) = \frac{\partial}{\partial x_i} \phi(x_i)$$

- Adaboost - same as potentials

- **Weight assigned to examples:** The gradient of the potential.

$$\nu = \frac{\partial}{\partial x_i} \Phi(x_1, \dots, x_n) = \frac{\partial}{\partial x_i} \phi(x_i)$$

- Adaboost - same as potentials
- Boost by majority: the binomial PMF.

Show boosting margin figures.



## Concepts when analyzing online learning

- Configuration:  $R_i$ : regret relative to expert  $i$

## Concepts when analyzing online learning

- ▶ Configuration:  $R_i$ : regret relative to expert  $i$
- ▶ Potential:

$$\Phi(R_1, \dots, R_n) = \frac{1}{\eta} \log \left( \sum_i \exp(-\eta R_i) \right)$$

## Concepts when analyzing online learning

- **Configuration:**  $R_i$ : regret relative to expert  $i$
- **Potential:**

$$\Phi(R_1, \dots, R_n) = \frac{1}{\eta} \log \left( \sum_i \exp(-\eta R_i) \right)$$

- **Weight:** The gradient of the potential. assigned to experts. For exponential potential has the same form as the potential.

## Concepts when analyzing online learning

- ▶ **Configuration:**  $R_i$ : regret relative to expert  $i$
- ▶ **Potential:**

$$\Phi(R_1, \dots, R_n) = \frac{1}{\eta} \log \left( \sum_i \exp(-\eta R_i) \right)$$

- ▶ **Weight:** The gradient of the potential. assigned to experts. For exponential potential has the same form as the potential.
- ▶ **Score:** Average potential:  $\Psi(\vec{m}) = \frac{1}{n} \sum_{i=1}^n \phi(m_i)$  experts try to maximize, learner, to minimize.

## Concepts when analyzing online learning

- **Configuration:**  $R_i$ : regret relative to expert  $i$
- **Potential:**

$$\Phi(R_1, \dots, R_n) = \frac{1}{\eta} \log \left( \sum_i \exp(-\eta R_i) \right)$$

- **Weight:** The gradient of the potential. assigned to experts. For exponential potential has the same form as the potential.
- **Score:** Average potential:  $\Psi(\vec{m}) = \frac{1}{n} \sum_{i=1}^n \phi(m_i)$  experts try to maximize, learner, to minimize.
- score upper bounds error of combined rule.