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 $c : X \to \{0, 1\}$.
- **Learning Input**: A training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ 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 $c : X \to \{0, 1\}$.
- **Learning Input**: A training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ 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 $c : X \to \{0, 1\}$.
- **Learning Input**: A training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ 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 ϵ, δ, 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 $c : X \to \{0, 1\}$.
- **Learning Input**: A training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ 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 ϵ, δ, 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 $\mathbf{M}$
- Row player chooses $i \in \{1, \ldots, n\}$

Zero sum games in matrix form

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

Zero sum games in matrix form

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

Zero sum games in matrix form

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

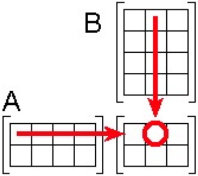
Pure vs. mixed strategies

- 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 $\mathbf{M}(\mathbf{P}, \mathbf{Q})$.

Pure vs. mixed strategies

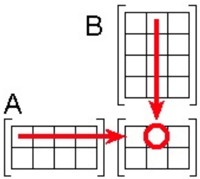
- Choosing a single action = pure strategy.
- Choosing a Distribution over actions = mixed strategy.
- Row player chooses dist. over rows $\mathbf{P}$
- Column player chooses dist. over columns $\mathbf{Q}$
- Row player gains $\mathbf{M}(\mathbf{P}, \mathbf{Q})$.
- Column player looses $\mathbf{M}(\mathbf{P},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

Mixed strategies in matrix notation



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

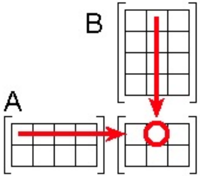
Mixed strategies in matrix notation



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

Q is a column vector. **P**^{T} 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. $\mathbf{P}^T$ 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.

$$\min_{\mathbf{P}} \max_{\mathbf{Q}} \mathbf{M}(\mathbf{P}, \mathbf{Q}) = \max_{\mathbf{Q}} \min_{\mathbf{P}} \mathbf{M}(\mathbf{P}, \mathbf{Q})$$

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 my 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 my 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 rhw (dist over the) rules are correct.

Boosting is implied my 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 rhw (dist over the) rules are correct.
- The weighted majority is always correct.

Boosting is implied my 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 rhw (dist over 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

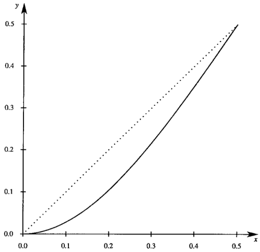
- h_1: use the training set as is.
- h_2: Filter examples so that $P(h_1(x) = c(x)) = \frac{1}{2}$
- h_3 Filter out examples such that $h_1(x) = h_2(x)$

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

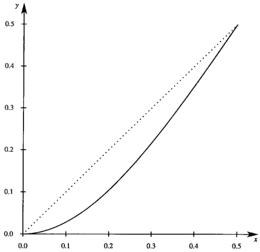
- If errors of weak rules are at most $x < 1/2$ then error of combined rule is at most $3x^2 - 2x^3$.



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

Idea of proof

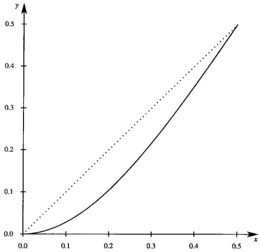
- If errors of weak rules are at most $x < 1/2$ then error of combined rule is at most $3x^2 - 2x^3$.



- *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 ϵ.

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 ϵ.
- Using 3-combiner recursively for depth at most $O(\frac{1}{\gamma^2} \log \frac{1}{\epsilon})$ achieves the error ϵ.

Boost By Majority

Majority vote over many weak rules, rather than 3.

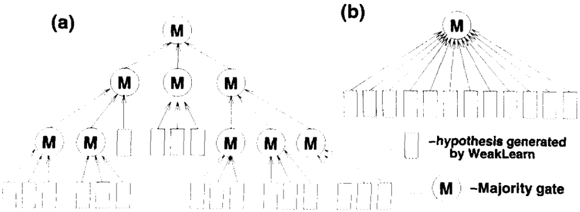


FIG. 1. Final concepts structure: (a) Schapire, (b) a one-layer majority circuit.

Game between booster and learner

- Booster chooses distribution over examples.

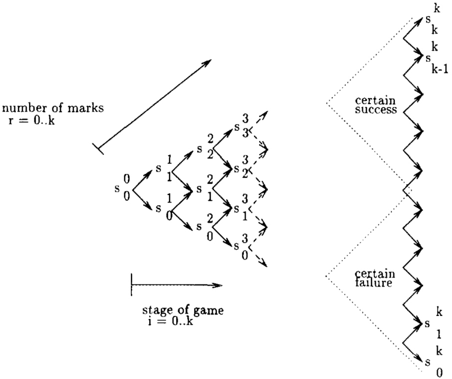


FIG. 2. Transitions between consecutive partitions.

Game between booster and learner

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

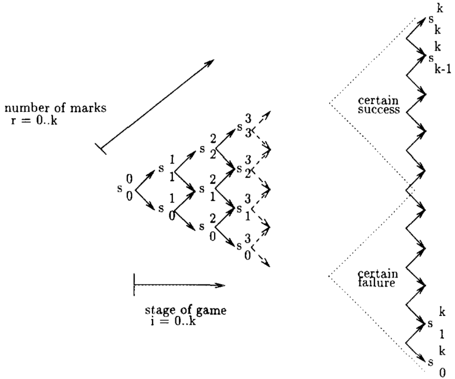


FIG. 2. Transitions between consecutive partitions.

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 $(1/2) - \gamma$

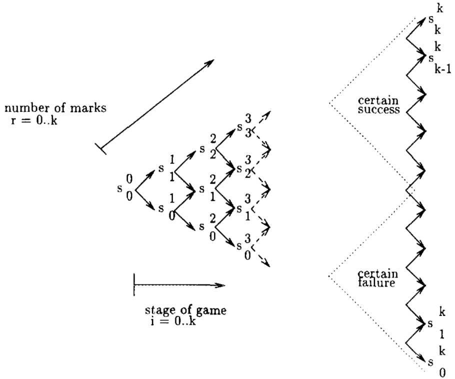


FIG. 2. Transitions between consecutive partitions.

Potential function

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

$$\beta_r^k = \begin{cases} 0 & \text{if} \quad r > \dfrac{k}{2} \\ 1 & \text{if} \quad r \leqslant \dfrac{k}{2}. \end{cases} \tag{4}$$

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

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

Weight function

The weighting factor is defined inductively as

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

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

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

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^i_r be the fraction of the examples that have r mistakes on iteration i.
- Then if the booster uses the weights α^i_r then

$$\beta^0_0 > \sum_{r=0}^{1} q^1_r \, \beta^1_r > \sum_{r=0}^{2} q^2_r \, \beta^2_r > \cdots > \sum_{r=0}^{k} q^k_r \, \beta^k_r .$$

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 α_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 ϵ.

Adaboost

Algorithm AdaBoost (Setup)

Input:

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

Initialize:

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

Algorithm AdaBoost (Setup)

Input:

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

Initialize:

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

Algorithm AdaBoost (Setup)

Input:

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

Initialize:

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

Algorithm AdaBoost (Main Loop)

For $t = 1, 2, \ldots, T$:

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

Algorithm AdaBoost (Main Loop)

For $t = 1, 2, \ldots, T$:

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 $t = 1, 2, \ldots, T$:

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 $h_t : X \to \{0, 1\}$.
3. Calculate the error of h_t:

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

Algorithm AdaBoost (Main Loop)

For $t = 1, 2, \ldots, T$:

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:

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

4. Set

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

Algorithm AdaBoost (Main Loop)

For $t = 1, 2, \ldots, T$:

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 $h_t : X \to \{0, 1\}$.
3. Calculate the error of h_t:
$$\epsilon_t = \sum_{i=1}^{N} p_i^t \mid h_t(x_i) - y_i \mid.$$
4. Set
$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}.$$
5. Update the weights:
$$w_i^{t+1} = w_i^t \, \beta_t^{\left(1 - \mid h_t(x_i) - y_i \mid\right)}.$$

Algorithm AdaBoost (Final Output)

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

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

Main Theorem

Theorem 6 Suppose the weak learning algorithm WeakLearn, when called by AdaBoost, generates hypotheses with errors $\epsilon_1, \ldots, \epsilon_T$. 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

$$
\begin{aligned}
\sum_{i=1}^{N} w_i^{t+1} &= \sum_{i=1}^{N} w_i^t \, \beta_t^{\left(1 - |\, h_t(x_i) - y_i \,|\right)} \\
&\leq \sum_{i=1}^{N} w_i' \left(1 - (1 - \beta_t)(1 - |\, h_t(x_i) - y_i \,|)\right) \\
&\leq \left(\sum_{i=1}^{N} w_i^t\right)\left(1 - (1 - \epsilon_t)\,(1 - \beta_t)\right).
\end{aligned}
$$

Combining over iterations

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

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

Lower bound on total weight

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

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

The final weight of instance i is

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

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,

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

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

$$
\begin{aligned}
e &\le \prod_{t=1}^{T}\sqrt{1-4\gamma_t^2} = \exp\left(-\sum_{t=1}^{T}\mathrm{KL}\left(\tfrac{1}{2}\,\|\,\tfrac{1}{2}-\gamma_t\right)\right) \\
&\le \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).
\end{aligned}
$$

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.

Algorithm 1.1: The boosting algorithm AdaBoost

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

Algorithm 1.1: The boosting algorithm AdaBoost

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

Algorithm 1.1: The boosting algorithm AdaBoost

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

Algorithm 1.1: The boosting algorithm AdaBoost

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

Algorithm 1.1: The boosting algorithm AdaBoost

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

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

Algorithm 1.1: The boosting algorithm AdaBoost

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

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

- **Output** the final hypothesis:

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

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 in score 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 in score 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 in score 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 in score 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 sometimes (mis)use the word potential to refer to ϕ and Score to refer to Φ

Concepts when analyzing boosting

- Adaboost: COnfiguration: Margins$m(x) = yH(x)$: larger than zero = correct.

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

Concepts when analyzing boosting

- Adaboost: COnfiguration: Margins$m(x) = yH(x)$: larger than zero = correct.

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

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

$$\Phi(x_1, \ldots, x_n) = \sum_j (\beta^i_{r(x_i)})$$

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

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

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

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

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

- booster tries to minimize Potential, weak rules - to maximize.

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:

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

Concepts when analyzing online learning

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

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

- Weight: The gradient of the potential. In this case has the same form as the potential.

Concepts when analyzing online learning

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

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

- Weight: The gradient of the potential. In this case 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:

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

- Weight: The gradient of the potential. In this case 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.