FUNDAMENTALS OF LEARNING AND INFORMATION PROCESSING

SESSION 15: STATISTICAL MACHINE LEARNING (V)

Gang Li

Deakin University, Australia

2021-10-01

Table of Content

Paradox

Learning for Decipher

Let F_n be a class of trapdoor functions over $\{0,1\}^n$: $F_n = \{f_{s_n} : s_n \text{ is the key}\}$. Now given a set of encrypted messages:

$$(x_1, f(x_1)), \cdots, (x_n, f(x_n))$$

Learn the class $F_n^{-1} = \{f^{-1} : f \in F_n\}$, whether it is feasible?

Learning for Decipher

Let F_n be a class of trapdoor functions over $\{0,1\}^n$: $F_n = \{f_{s_n} : s_n \text{ is the key}\}$. Now given a set of encrypted messages:

$$(x_1, f(x_1)), \cdots, (x_n, f(x_n))$$

Learn the class $F_n^{-1} = \{f^{-1} : f \in F_n\}$, whether it is feasible?

Proof.

- $|F_n| \leq 2^{Poly(n)}$, so $VCDim(F_n) = Poly(n)$

Learning for Decipher

Let F_n be a class of trapdoor functions over $\{0,1\}^n$: $F_n = \{f_{s_n} : s_n \text{ is the key}\}$. Now given a set of encrypted messages:

$$(x_1, f(x_1)), \cdots, (x_n, f(x_n))$$

Learn the class $F_n^{-1} = \{f^{-1} : f \in F_n\}$, whether it is feasible?

Proof.

- $|F_n| \leq 2^{Poly(n)}$, so $VCDim(F_n) = Poly(n)$
- So the number of examples we need will be polynomial in n.

Learning for Decipher

Let F_n be a class of trapdoor functions over $\{0,1\}^n$: $F_n = \{f_{s_n} : s_n \text{ is the key}\}$. Now given a set of encrypted messages:

$$(x_1, f(x_1)), \cdots, (x_n, f(x_n))$$

Learn the class $F_n^{-1} = \{f^{-1} : f \in F_n\}$, whether it is feasible?

Proof.

- $|F_n| \leq 2^{Poly(n)}$, so $VCDim(F_n) = Poly(n)$
- So the number of examples we need will be polynomial in n.
- There is no efficient learner for this class. Otherwise, ...?

□

Computational Complexity of Learning

Computational Complexity of Learning

How much computation is involved in carrying out a learning task?

Sample Complexity as in *the Fundamental Theorem of PAC Learning*

Computational Complexity analyses the runtime of algorithms in an asymptotic sense.

Computational Complexity of Learning

How much computation is involved in carrying out a learning task?

Sample Complexity as in *the Fundamental Theorem of PAC Learning*
Computational Complexity analyses the runtime of algorithms in an asymptotic sense.

Algorithmic Consideration.

For a task with *input size* n, it is common to use the term *feasible* or *efficiently computable* for tasks that can be performed by an algorithm whose running time is $O(p(n))$ for some polynomial function p.

Computational Complexity of Learning

How much computation is involved in carrying out a learning task?

Sample Complexity as in *the Fundamental Theorem of PAC Learning*
Computational Complexity analyses the runtime of algorithms in an asymptotic sense.

Algorithmic Consideration.

For a task with *input size* n, it is common to use the term *feasible* or *efficiently computable* for tasks that can be performed by an algorithm whose running time is $O(p(n))$ for some polynomial function p.

Input size **of the learning algorithm** $\mathscr{A}$

Computational Complexity of Learning

How much computation is involved in carrying out a learning task?

Sample Complexity as in *the Fundamental Theorem of PAC Learning*
Computational Complexity analyses the runtime of algorithms in an asymptotic sense.

Algorithmic Consideration.

For a task with *input size* n, it is common to use the term *feasible* or *efficiently computable* for tasks that can be performed by an algorithm whose running time is $O(p(n))$ for some polynomial function p.

***Input size* of the learning algorithm $\mathscr{A}$**

- It should not be the sample size, as if the size is larger than the sample complexity, the algorithm can safely ignore those extra examples;

Computational Complexity of Learning

How much computation is involved in carrying out a learning task?

Sample Complexity as in *the Fundamental Theorem of PAC Learning*
Computational Complexity analyses the runtime of algorithms in an asymptotic sense.

Algorithmic Consideration.

For a task with *input size* n, it is common to use the term *feasible* or *efficiently computable* for tasks that can be performed by an algorithm whose running time is $O(p(n))$ for some polynomial function p.

***Input size* of the learning algorithm $\mathscr{A}$**

- It should not be the sample size, as if the size is larger than the sample complexity, the algorithm can safely ignore those extra examples;
- Instead, we consider ϵ and δ, as well as the hypothesis class complexity n. We hope that the running time is $O(p(\frac{1}{\epsilon}, \frac{1}{\delta}, n))$ for some polynomial function p.

Computational Complexity of Learning

How much computation is involved in carrying out a learning task?

Sample Complexity as in *the Fundamental Theorem of PAC Learning*
Computational Complexity analyses the runtime of algorithms in an asymptotic sense.

Algorithmic Consideration.

For a task with *input size* n, it is common to use the term *feasible* or *efficiently computable* for tasks that can be performed by an algorithm whose running time is $O(p(n))$ for some polynomial function p.

Input size **of the learning algorithm** $\mathscr{A}$

- It should not be the sample size, as if the size is larger than the sample complexity, the algorithm can safely ignore those extra examples;
- Instead, we consider ϵ and δ, as well as the hypothesis class complexity n. We hope that the running time is $O(p(\frac{1}{\epsilon}, \frac{1}{\delta}, n))$ for some polynomial function p.

Output **of the learning algorithm** $\mathscr{A}$ must be efficient
Runtime **of the learning algorithm** $\mathscr{A}$
max(time for $\mathscr{A}$ to output some h, time for h to output a label for any given X)

□

Computational Complexity of Learning

Given a function $f : (0,1)^2 \to \mathcal{N}$, a learning task $(Z, \mathcal{H}, l)$, and a learning algorithm $\mathcal{A}$ solves the learning task in time $O(f)$ if there exists a constant number c, such that for every probability distribution $\mathcal{D}$ over $\mathcal{A}$, and input $\epsilon, \delta \in (0,1)$, when $\mathcal{A}$ has access to samples generated i.i.d. by $\mathcal{D}$.

1. $\mathcal{A}$ terminates after performing at most $cf(\epsilon,\delta)$ operations
2. The output of $\mathcal{A}$, h_A, can be applied to predict the label of a new example while performing at most $cf(\epsilon,\delta)$ operations
3. The output of $\mathcal{A}$ is probably approximately correct: with probability of at least $1-\delta$, $L_{\mathcal{D}} \leq \min_{h^* \in \mathcal{H}} L_{\mathcal{D}(h^*)} + \epsilon$

Computational Complexity of Learning

Given a function $f : (0,1)^2 \to \mathcal{N}$, a learning task $(Z, \mathcal{H}, l)$, and a learning algorithm $\mathcal{A}$ solves the learning task in time $O(f)$ if there exists a constant number c, such that for every probability distribution $\mathcal{D}$ over $\mathcal{A}$, and input $\epsilon, \delta \in (0,1)$, when $\mathcal{A}$ has access to samples generated i.i.d. by $\mathcal{D}$.

1. $\mathcal{A}$ terminates after performing at most $cf(\epsilon,\delta)$ operations
2. The output of $\mathcal{A}$, h_A, can be applied to predict the label of a new example while performing at most $cf(\epsilon,\delta)$ operations
3. The output of $\mathcal{A}$ is probably approximately correct: with probability of at least $1-\delta$, $L_{\mathcal{D}} \leq \min_{h^* \in \mathcal{H}} L_{\mathcal{D}(h^*)} + \epsilon$

Consider a sequence of learning problems, $(Z_n, \mathcal{H}_n, l_n)_{n=1}^{\infty}$, where n is defined by a domain Z_n, a hypothesis class $\mathcal{H}_n$ and a loss function l_n. Let $\mathcal{A}$ be a learning algorithm designed for solving learning problems of this form. Given a function $g : \mathcal{N} \times (0,1)^2 \to \mathcal{N}$, we say that the runtime of $\mathcal{A}$ with respect to the preceding sequence is $O(g)$, if for all n, $\mathcal{A}$ solves the problem in time $O(f_n)$, where $f_n : (0,1)^2 \to \mathcal{N}$ is defined by $f_n(\epsilon,\delta) = g(n,\epsilon,\delta)$.

- $\mathcal{A}$ is an efficient algorithm with respect to a sequence $(Z_n, \mathcal{H}_n, l_n)$ if its runtime is $O(p(\frac{1}{\epsilon}, \frac{1}{\delta}, n))$ for some polynomial p.

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

- ERM over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn if the sample size is in the order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

- ERM over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn if the sample size is in the order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$
- If using exhaustive search, the implementation of ERM will be in time $O(|\mathcal{H}| m_{\mathcal{H}}(\epsilon, \delta))$

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

- ERM over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn if the sample size is in the order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$
- If using exhaustive search, the implementation of ERM will be in time $O(|\mathcal{H}| m_{\mathcal{H}}(\epsilon, \delta))$
- So for any fixed finite $\mathcal{H}$, the exhaustive search runs in polynomial time.

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

- ERM over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn if the sample size is in the order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$
- If using exhaustive search, the implementation of ERM will be in time $O(|\mathcal{H}| m_{\mathcal{H}}(\epsilon, \delta))$
- So for any fixed finite $\mathcal{H}$, the exhaustive search runs in polynomial time.

A Sequence of Problem

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

- ERM over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn if the sample size is in the order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$
- If using exhaustive search, the implementation of ERM will be in time $O(|\mathcal{H}| m_{\mathcal{H}}(\epsilon, \delta))$
- So for any fixed finite $\mathcal{H}$, the exhaustive search runs in polynomial time.

A Sequence of Problem

- If we define a sequence of problems in which $|\mathcal{H}_n| = n$, then the exhaustive search is still efficient;

Computational Complexity of Learning

Whether a general learning problem can be solved efficiently depends on *how it can be broken into a sequence of specific learning problems*.

Examples.

Learning Finite Hypothesis Class

- ERM over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn if the sample size is in the order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$
- If using exhaustive search, the implementation of ERM will be in time $O(|\mathcal{H}| m_{\mathcal{H}}(\epsilon, \delta))$
- So for any fixed finite $\mathcal{H}$, the exhaustive search runs in polynomial time.

A Sequence of Problem

- If we define a sequence of problems in which $|\mathcal{H}_n| = n$, then the exhaustive search is still efficient;
- If we define a sequence of problems in which $|\mathcal{H}_n| = 2^n$, then the exhaustive search is not efficient;

□

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d}) = 2d$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d}) = 2d$
- $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d}) = 2d$
- $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$
- For any sample S with $m_{\mathcal{H}}(\epsilon, \delta)$ examples, it is sufficient to consider rectangles that have points of S on every boundary edge.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d}) = 2d$
- $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$
- For any sample S with $m_{\mathcal{H}}(\epsilon, \delta)$ examples, it is sufficient to consider rectangles that have points of S on every boundary edge.

Step 2: Upper bound the time -

- Every such rectangle is determined by at most $2d$ points from S; there are $\leq m^{2d}$ such tuples, so the run time $\sim [C \frac{2d + \ln(2/\delta)}{\epsilon^2}]^{2d}$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d}) = 2d$
- $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$
- For any sample S with $m_{\mathcal{H}}(\epsilon, \delta)$ examples, it is sufficient to consider rectangles that have points of S on every boundary edge.

Step 2: Upper bound the time -

- Every such rectangle is determined by at most $2d$ points from S; there are $\leq m^{2d}$ such tuples, so the run time $\sim [C \frac{2d + \ln(2/\delta)}{\epsilon^2}]^{2d}$

Step 3: Algorithm -

- So for any fixed finite $\mathcal{H}_d$, the exhaustive search runs in *polynomial* time.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Axis aligned rectangles $\mathcal{X} = \mathcal{R}^d$, let $\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d} = \{h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(x) : a_i < b_i\}$, where $h_{a_1,\cdots,a_d,b_1,\cdots,b_d}(\vec{x}, y) = 1$ iff $x_1 \in \forall i, x_i \in [a_i, b_i]$.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}_{a_1,\cdots,a_d,b_1,\cdots,b_d}) = 2d$
- $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$
- For any sample S with $m_{\mathcal{H}}(\epsilon, \delta)$ examples, it is sufficient to consider rectangles that have points of S on every boundary edge.

Step 2: Upper bound the time -

- Every such rectangle is determined by at most $2d$ points from S; there are $\leq m^{2d}$ such tuples, so the run time $\sim [C \frac{2d + \ln(2/\delta)}{\epsilon^2}]^{2d}$

Step 3: Algorithm -

- So for any fixed finite $\mathcal{H}_d$, the exhaustive search runs in *polynomial* time.
- So for a variable d, the exhaustive search runs in *exponential* time.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Boolean Conjunctions is a mapping from $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ that can be expressed as $x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_r}$. The function is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \text{if } x_{i_1} = \cdots = x_{i_k} = 1 \text{ and } x_{j_1} = \cdots = x_{j_r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Boolean Conjunctions is a mapping from $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ that can be expressed as $x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_r}$. The function is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \text{if } x_{i_1} = \cdots = x_{i_k} = 1 \text{ and } x_{j_1} = \cdots = x_{j_r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq 2^{2d}$ or $\leq 3^d$. Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Boolean Conjunctions is a mapping from $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ that can be expressed as $x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_r}$. The function is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \text{if } x_{i_1} = \cdots = x_{i_k} = 1 \text{ and } x_{j_1} = \cdots = x_{j_r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq 2^{2d}$ or $\leq 3^d$. Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$

Step 2: Algorithm - *Realizable* cases

- Let $h_0 = x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{i_1} \wedge \cdots \wedge \neg x_{i_k}$ and $L_S(h_0) = \frac{|h(\vec{x}_i, y_i) : y_i = 1|}{m}$.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Boolean Conjunctions is a mapping from $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ that can be expressed as $x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_r}$. The function is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \text{if } x_{i_1} = \cdots = x_{i_k} = 1 \text{ and } x_{j_1} = \cdots = x_{j_r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq 2^{2d}$ or $\leq 3^d$. Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$

Step 2: Algorithm - *Realizable* cases

- Let $h_0 = x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{i_1} \wedge \cdots \wedge \neg x_{i_k}$ and $L_S(h_0) = \frac{|h(\vec{x}_i, y_i) : y_i = 1|}{m}$.
- $h_{t+1} = \begin{cases} h_t & h_t(x_t) = y_t \\ \text{delete any literal that fails in } h_t & \text{otherwise} \end{cases}$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Boolean Conjunctions is a mapping from $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ that can be expressed as $x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_r}$. The function is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \text{if } x_{i_1} = \cdots = x_{i_k} = 1 \text{ and } x_{j_1} = \cdots = x_{j_r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq 2^{2d}$ or $\leq 3^d$. Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$

Step 2: Algorithm - *Realizable* cases

- Let $h_0 = x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{i_1} \wedge \cdots \wedge \neg x_{i_k}$ and $L_S(h_0) = \frac{|h(\vec{x}_i, y_i) : y_i = 1|}{m}$.
- $h_{t+1} = \begin{cases} h_t & h_t(x_t) = y_t \\ \text{delete any literal that fails in } h_t & \text{otherwise} \end{cases}$
- h_{t+1} is the minimal (the most demanding) conjunction that accepts all t labelled examples among $(x_1, y_1), \cdots, (x_t, y_t)$.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

Boolean Conjunctions is a mapping from $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ that can be expressed as $x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_r}$. The function is defined as:

$$h(\vec{x}) = \begin{cases} 1 & \text{if } x_{i_1} = \cdots = x_{i_k} = 1 \text{ and } x_{j_1} = \cdots = x_{j_r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq 2^{2d}$ or $\leq 3^d$. Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{2d + \ln(2/\delta)}{\epsilon^2}$

Step 2: Algorithm - *Realizable* cases

- Let $h_0 = x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \neg x_{i_1} \wedge \cdots \wedge \neg x_{i_k}$ and $L_S(h_0) = \frac{|h(\vec{x}_i, y_i) : y_i = 1|}{m}$.
- $h_{t+1} = \begin{cases} h_t & h_t(x_t) = y_t \\ \text{delete any literal that fails in } h_t & \text{otherwise} \end{cases}$
- h_{t+1} is the minimal (the most demanding) conjunction that accepts all t labelled examples among $(x_1, y_1), \cdots, (x_t, y_t)$.

Step 3: Upper bound the time -

- $m \times 2d = C \frac{2d + \ln(2/\delta)}{\epsilon^2} \times 2d$, so it is polynomial.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq (3^d)^3 = 3^{3d}$.

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq (3^d)^3 = 3^{3d}$.
- Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{d + \ln(2/\delta)}{\epsilon^2}$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq (3^d)^3 = 3^{3d}$.
- Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{d + \ln(2/\delta)}{\epsilon^2}$

Step 2: NP Hard even in realizable case

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq (3^d)^3 = 3^{3d}$.
- Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{d + \ln(2/\delta)}{\epsilon^2}$

Step 2: NP Hard even in realizable case

Step 3: An Non-ERM algorithm

- Note that each $h = A_1 \vee A_2 \vee A_3 = \wedge_{u \in A_1, v \in A_2, w \in A_3} (u \vee v \vee w)$

Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq (3^d)^3 = 3^{3d}$.
- Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{d + \ln(2/\delta)}{\epsilon^2}$

Step 2: NP Hard even in realizable case

Step 3: An Non-ERM algorithm

- Note that each $h = A_1 \vee A_2 \vee A_3 = \wedge_{u \in A_1, v \in A_2, w \in A_3} (u \vee v \vee w)$
- Define new $(2d)^3$ variables X_{uvw} to replace $u \vee v \vee w$

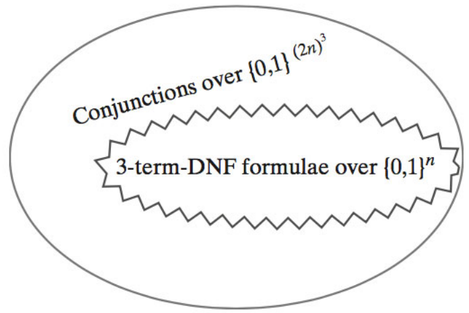
Implementing the ERM Rule

On a finite input sample $S \in \mathcal{Z}^m$, the ERM rule outputs some $h \in \mathcal{H}$ that minimizes the empirical loss $L_S(h) = \frac{1}{|S|} \sum_{z \in S} l(h, z)$

3-Term DNF each $h \in \mathcal{H}$ has the form $h = A_1 \vee A_2 \vee A_3$, where each A_i is a conjunction.

Step 1: Upper bound the sample size -

- $VCdim(\mathcal{H}) \leq \log(|\mathcal{H}|)$, and $|\mathcal{H}| \leq (3^d)^3 = 3^{3d}$.
- Therefore, $m_{\mathcal{H}}(\epsilon, \delta) \sim C \frac{d + \ln(2/\delta)}{\epsilon^2}$



Step 2: NP Hard even in realizable case

Step 3: An Non-ERM algorithm

- Note that each $h = A_1 \vee A_2 \vee A_3 = \wedge_{u \in A_1, v \in A_2, w \in A_3} (u \vee v \vee w)$
- Define new $(2d)^3$ variables X_{uvw} to replace $u \vee v \vee w$
- Such conjunction can be learned in time $poly(\frac{1}{\epsilon}, \frac{1}{\delta}, 8d^3)$

Hardness of Learning

Hardness of Learning

Table 1: Hardness of Learning

	Realizable	**Agnostic**
Unrestricted Proper	3-term DNF `Conjunctions` over d variables with complexity $O(\frac{d}{\epsilon})$	$\mathscr{H}_{rec}^{d}$ with complexity $O(\frac{d}{\epsilon^2})^d$

It is *hard* from left to right, or from top to bottom.

NP-Hardness

NP-Hardness Unless there is a big surprise in the mathematics or CS, there is no polynomial time algorithm that solves the problem for ALL inputs.

NP-Hardness

NP-Hardness Unless there is a big surprise in the mathematics or CS, there is no polynomial time algorithm that solves the problem for ALL inputs.

Examples.

Learning Axis-aligned Rectangles $\mathscr{H}_{rec}^{d}$

NP-Hardness

NP-Hardness Unless there is a big surprise in the mathematics or CS, there is no polynomial time algorithm that solves the problem for ALL inputs.

Examples.

Learning Axis-aligned Rectangles $\mathscr{H}_{rec}^{d}$

- If we wish the algorithm to be also polynomial in d, it is NP-Hard even in *realizable* cases for *proper* learning.

NP-Hardness

NP-Hardness Unless there is a big surprise in the mathematics or CS, there is no polynomial time algorithm that solves the problem for ALL inputs.

Examples.

Learning Axis-aligned Rectangles $\mathscr{H}_{rec}^{d}$

- If we wish the algorithm to be also polynomial in d, it is NP-Hard even in *realizable* cases for *proper* learning.

Learning Half-Spaces $\mathscr{H}_{linear}^{d}$

- It is easy for *realizable* case.

NP-Hardness

NP-Hardness Unless there is a big surprise in the mathematics or CS, there is no polynomial time algorithm that solves the problem for ALL inputs.

Examples.

Learning Axis-aligned Rectangles $\mathscr{H}_{rec}^{d}$

- If we wish the algorithm to be also polynomial in d, it is NP-Hard even in *realizable* cases for *proper* learning.

Learning Half-Spaces $\mathscr{H}_{linear}^{d}$

- It is easy for *realizable* case.
- But it is NP Hard in the *agnostic* case, for *proper* learning and *unrestricted* learning.

□

Cryptographic-Hardness

Cryptographic-Hardness Many cryptographic systems assume that there exists a one way function $f : \{0,1\}^n \to \{0,1\}^n$, that is easy to compute but is hard to invert: f can be computed in time polynomial $p(n)$, but no polynomial time algorithm can invert it: for any randomized polynomial time algorithm $\mathscr{A}$, for any every polynomial $p(\cdot)$,

$$P[f(A(f(\vec{x}))) = f(\vec{x})] < \frac{1}{p(n)}$$

Cryptographic-Hardness

Cryptographic-Hardness Many cryptographic systems assume that there exists a one way function $f : \{0,1\}^n \to \{0,1\}^n$, that is easy to compute but is hard to invert: f can be computed in time polynomial $p(n)$, but no polynomial time algorithm can invert it: for any randomized polynomial time algorithm $\mathscr{A}$, for any every polynomial $p(\cdot)$,

$$P[f(A(f(\vec{x}))) = f(\vec{x})] < \frac{1}{p(n)}$$

Examples.

ATM Pin

- Save the $f(pin)$ in the stripe on the back of the card.

Cryptographic-Hardness

Cryptographic-Hardness Many cryptographic systems assume that there exists a one way function $f : \{0,1\}^n \to \{0,1\}^n$, that is easy to compute but is hard to invert: f can be computed in time polynomial $p(n)$, but no polynomial time algorithm can invert it: for any randomized polynomial time algorithm $\mathscr{A}$, for any every polynomial $p(\cdot)$,

$$P[f(A(f(\vec{x}))) = f(\vec{x})] < \frac{1}{p(n)}$$

Examples.

ATM Pin

- Save the $f(pin)$ in the stripe on the back of the card.

Cryptographic-Hardness

Cryptographic-Hardness Many cryptographic systems assume that there exists a one way function $f : \{0,1\}^n \to \{0,1\}^n$, that is easy to compute but is hard to invert: f can be computed in time polynomial $p(n)$, but no polynomial time algorithm can invert it: for any randomized polynomial time algorithm $\mathscr{A}$, for any every polynomial $p(\cdot)$,

$$P[f(A(f(\vec{x}))) = f(\vec{x})] < \frac{1}{p(n)}$$

Examples.

ATM Pin

- Save the $f(pin)$ in the stripe on the back of the card.

Trapdoor Function

- One way function with a secret key s_n for every n such that it is easy to invert $f(x)$ if s_n is known, otherwise hard if s_n is unknown.

Cryptographic-Hardness

Cryptographic-Hardness Many cryptographic systems assume that there exists a one way function $f : \{0,1\}^n \to \{0,1\}^n$, that is easy to compute but is hard to invert: f can be computed in time polynomial $p(n)$, but no polynomial time algorithm can invert it: for any randomized polynomial time algorithm $\mathscr{A}$, for any every polynomial $p(\cdot)$,

$$P[f(A(f(\vec{x}))) = f(\vec{x})] < \frac{1}{p(n)}$$

Examples.

ATM Pin

- Save the $f(pin)$ in the stripe on the back of the card.

Trapdoor Function

- One way function with a secret key s_n for every n such that it is easy to invert $f(x)$ if s_n is known, otherwise hard if s_n is unknown.
- Discrete log function

□

Validation and Model Selection

Model Validation

We have already learned some hypothesis h, now we want to estimate how good is h. When we can randomly partition the set of examples into two parts, one as training set S, and one as validation set V, also known as hold-out set.

Model Validation

We have already learned some hypothesis h, now we want to estimate how good is h. When we can randomly partition the set of examples into two parts, one as training set S, and one as validation set V, also known as hold-out set.

Solution.

- Take a fresh i.i.d. sample $V = \{(x_1, y_1), \cdots, (x_{m_V}, y_{m_V})\}$

Model Validation

We have already learned some hypothesis h, now we want to estimate how good is h. When we can randomly partition the set of examples into two parts, one as training set S, and one as validation set V, also known as hold-out set.

Solution.

- Take a fresh i.i.d. sample $V = \{(x_1, y_1), \cdots, (x_{m_V}, y_{m_V})\}$
- Output $L_V(h)$ as an estimator of $L_{\mathscr{D}}(h)$

Model Validation

We have already learned some hypothesis h, now we want to estimate how good is h. When we can randomly partition the set of examples into two parts, one as training set S, and one as validation set V, also known as hold-out set.

Solution.

- Take a fresh i.i.d. sample $V = \{(x_1, y_1), \cdots, (x_{m_V}, y_{m_V})\}$
- Output $L_V(h)$ as an estimator of $L_{\mathscr{D}}(h)$
  - Using the *Hoeffding's inequality*, for any $\epsilon > 0$, $P(|L_V(h) - L_{\mathscr{D}}(h)| > \epsilon) \leq 2e^{-2m_V\epsilon^2} = \delta$

Model Validation

We have already learned some hypothesis h, now we want to estimate how good is h. When we can randomly partition the set of examples into two parts, one as training set S, and one as validation set V, also known as hold-out set.

Solution.

- Take a fresh i.i.d. sample $V = \{(x_1, y_1), \cdots, (x_{m_V}, y_{m_V})\}$
- Output $L_V(h)$ as an estimator of $L_{\mathscr{D}}(h)$
  - Using the *Hoeffding's inequality*, for any $\epsilon > 0$, $P(|L_V(h) - L_{\mathscr{D}}(h)| > \epsilon) \leq 2e^{-2m_V\epsilon^2} = \delta$
  - Given δ and m_V, we have $\epsilon = \sqrt{\frac{\log(2/\delta)}{2m_V}}$, so we have

$$L_{\mathscr{D}}(h) \leq L_V(h) + \sqrt{\frac{\log(2/\delta)}{2m_V}}$$

Model Validation

We have already learned some hypothesis h, now we want to estimate how good is h. When we can randomly partition the set of examples into two parts, one as training set S, and one as validation set V, also known as hold-out set.

Solution.

- Take a fresh i.i.d. sample $V = \{(x_1, y_1), \cdots, (x_{m_V}, y_{m_V})\}$
- Output $L_V(h)$ as an estimator of $L_{\mathscr{D}}(h)$
  - Using the *Hoeffding's inequality*, for any $\epsilon > 0$, $P(|L_V(h) - L_{\mathscr{D}}(h)| > \epsilon) \leq 2e^{-2m_V\epsilon^2} = \delta$
  - Given δ and m_V, we have $\epsilon = \sqrt{\frac{\log(2/\delta)}{2m_V}}$, so we have

$$L_{\mathscr{D}}(h) \leq L_V(h) + \sqrt{\frac{\log(2/\delta)}{2m_V}}$$

- This is in general more accurate than the estimate on the *training* set S, which is from fundamental theorem of learning:

$$L_{\mathscr{D}}(h) \leq L_S(h) + \sqrt{C\frac{d + \log(1/\delta)}{m}}$$

Model Selection

Let $\mathscr{H} = \{h_1, \cdots, h_r\}$ be the set of all output predictors of the different algorithms. Assume a validation set V of size m_V is sampled independent of $\mathscr{H}$. Then, with probability of at least $1-\delta$ over the choice of V, we have $\forall h \in \mathscr{H}$,

$$|L_{\mathscr{D}}(h) - L_V(h)| \leq \sqrt{\frac{\log(2|\mathscr{H}|/\delta)}{2m_V}}$$

Model Selection

Let $\mathscr{H} = \{h_1, \cdots, h_r\}$ be the set of all output predictors of the different algorithms. Assume a validation set V of size m_V is sampled independent of $\mathscr{H}$. Then, with probability of at least $1-\delta$ over the choice of V, we have $\forall h \in \mathscr{H}$,

$$|L_{\mathscr{D}}(h) - L_V(h)| \leq \sqrt{\frac{\log(2|\mathscr{H}|/\delta)}{2m_V}}$$

Implications.

- The proof is directly from each single hypothesis h on V, combined with union bounds.

Model Selection

Let $\mathscr{H} = \{h_1, \cdots, h_r\}$ be the set of all output predictors of the different algorithms. Assume a validation set V of size m_V is sampled independent of $\mathscr{H}$. Then, with probability of at least $1-\delta$ over the choice of V, we have $\forall h \in \mathscr{H}$,

$$|L_{\mathscr{D}}(h) - L_V(h)| \leq \sqrt{\frac{\log(2|\mathscr{H}|/\delta)}{2m_V}}$$

Implications.

- The proof is directly from each single hypothesis h on V, combined with union bounds.
- This theorem tells us that the error on the validation set approximates the true error as long as $\mathscr{H}$ is not too large.

Model Selection

Let $\mathscr{H} = \{h_1, \cdots, h_r\}$ be the set of all output predictors of the different algorithms. Assume a validation set V of size m_V is sampled independent of $\mathscr{H}$. Then, with probability of at least $1-\delta$ over the choice of V, we have $\forall h \in \mathscr{H}$,

$$|L_{\mathscr{D}}(h) - L_V(h)| \leq \sqrt{\frac{\log(2|\mathscr{H}|/\delta)}{2m_V}}$$

Implications.

- The proof is directly from each single hypothesis h on V, combined with union bounds.
- This theorem tells us that the error on the validation set approximates the true error as long as $\mathscr{H}$ is not too large.
- Suppose the chosen one is $h_S \in ERM_{\mathscr{H}}(V)$, then from the uniform convergence property, we have:

$$L_{\mathscr{D}}(h_S) \leq \min_{h^* \in \mathscr{H}} L_{\mathscr{D}}(h^*) + \sqrt{\frac{2\log(2|\mathscr{H}|/\delta)}{m_V}}$$

Model Selection in Practise

When data is plentiful, the best model selection approach is the training-validation-test split:

Training Set apply the learning algorithm with different parameters on the training set to produce $\mathscr{H} = \{h_1, \cdots, h_r\}$

Validation Set Choose h_S from $\mathscr{H}$ based on the validation set

Test Set Estimate the error of h_S using the test set

Model Selection in Practise

When data is plentiful, the best model selection approach is the training-validation-test split:

Training Set apply the learning algorithm with different parameters on the training set to produce $\mathscr{H} = \{h_1, \cdots, h_r\}$
Validation Set Choose h_S from $\mathscr{H}$ based on the validation set
Test Set Estimate the error of h_S using the test set

Implications.

- When data is scarce, we use k-fold cross validation.

□

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule
- Changing the parameters you consider

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule
- Changing the parameters you consider
- Get a larger sample

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule
- Changing the parameters you consider
- Get a larger sample
- Change the hypothesis class by
  - Enlarging it

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule
- Changing the parameters you consider
- Get a larger sample
- Change the hypothesis class by
  - Enlarging it
  - Reducing it

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule
- Changing the parameters you consider
- Get a larger sample
- Change the hypothesis class by
  - Enlarging it
  - Reducing it
  - Completely changing it

Machine Learning Diagnosis

Consider a scenario:

> You were given a learning task and have approached it with a choice of hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on the test set. The test result turns out to be unsatisfactory.

What went wrong? and what should you do next?

How to Fix?

- Change the optimization algorithm used to apply your learning rule
- Changing the parameters you consider
- Get a larger sample
- Change the hypothesis class by
  - Enlarging it
  - Reducing it
  - Completely changing it
- Change the feature representation of the data

Error Decomposition Using Training Sample Only

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \text{argmin}_{h \in \mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*)$. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*) + L_{\mathcal{D}}(h^*) - L_{\mathcal{D}}(f) + L_{\mathcal{D}}(f) = \epsilon_{est} + \epsilon_{app} + L_{\mathcal{D}}(f)$$

Error Decomposition Using Training Sample Only

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \text{argmin}_{h \in \mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*)$. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*) + L_{\mathcal{D}}(h^*) - L_{\mathcal{D}}(f) + L_{\mathcal{D}}(f) = \epsilon_{est} + \epsilon_{app} + L_{\mathcal{D}}(f)$$

How to Fix?

- If the approximation error ϵ_{app} is large, it will not help us to enlarge the sample size, nor to reduce the hypothesis class. We should enlarge the hypothesis class or change it, or on a different feature representation of the data;

Error Decomposition Using Training Sample Only

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \text{argmin}_{h \in \mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*)$. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*) + L_{\mathcal{D}}(h^*) - L_{\mathcal{D}}(f) + L_{\mathcal{D}}(f) = \epsilon_{est} + \epsilon_{app} + L_{\mathcal{D}}(f)$$

How to Fix?

- If the approximation error ϵ_{app} is large, it will not help us to enlarge the sample size, nor to reduce the hypothesis class. We should enlarge the hypothesis class or change it, or on a different feature representation of the data;
- The estimation error ϵ_{est} depends on the sample size. If it is large, we should obtain more training examples; or reduce the hypothesis class.

□

Error Decomposition Using Validation Sample

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*)$, and V is the validation set. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_V(h_S) + L_V(h_S) - L_S(h_S) + L_S(h_S)$$

Error Decomposition Using Validation Sample

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*)$, and V is the validation set. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_V(h_S) + L_V(h_S) - L_S(h_S) + L_S(h_S)$$

How to Fix?

- $L_{\mathcal{D}}(h_S) - L_V(h_S)$ can be bounded quite tightly

Error Decomposition Using Validation Sample

Let h_S be an $ERM_{\mathscr{H}}$ hypothesis: $h_S \in \operatorname{argmin}_{h\in\mathscr{H}} L_S(h)$. Assume $f \in \mathscr{Y}^{\mathscr{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathscr{H}$: $h^* = \min_{h^*\in\mathscr{H}} L_{\mathscr{D}}(h^*)$, and V is the validation set. Then we can have:

$$L_{\mathscr{D}}(h_S) = L_{\mathscr{D}}(h_S) - L_V(h_S) + L_V(h_S) - L_S(h_S) + L_S(h_S)$$

How to Fix?

- $L_{\mathscr{D}}(h_S) - L_V(h_S)$ can be bounded quite tightly
- When $L_V(h_S) - L_S(h_S)$ is *large*, the algorithm suffers from "overfitting"

□

Error Decomposition Using Validation Sample

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \operatorname{argmin}_{h\in\mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^*\in\mathcal{H}} L_{\mathcal{D}}(h^*)$, and V is the validation set. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_V(h_S) + L_V(h_S) - L_S(h_S) + L_S(h_S)$$

How to Fix?

- $L_{\mathcal{D}}(h_S) - L_V(h_S)$ can be bounded quite tightly
- When $L_V(h_S) - L_S(h_S)$ is *large*, the algorithm suffers from "overfitting"
- When $L_S(h_S)$ is *large*, the algorithm suffers from "underfitting"

$$L_S(h_S) = (L_S(h_S) - L_S(h^*)) + (L_S(h^*) - L_{\mathcal{D}}(h^*)) + L_{\mathcal{D}}(h^*)$$

  - When h_S is an ERM result, we have $L_S(h_S) - L_S(h^*) \le 0$
  - Since h^* is not dependent on S, $(L_S(h^*) - L_{\mathcal{D}}(h^*))$ can be bounded tightly.
  - $L_{\mathcal{D}}(h^*)$ is the approximation error. So if $L_S(h_S)$ is large, the approximation error $L_D(h^*)$ is generally large, though not always (why?)

□

Error Decomposition Using Validation Sample

Let h_S be an $ERM_{\mathcal{H}}$ hypothesis: $h_S \in \text{argmin}_{h \in \mathcal{H}} L_S(h)$. Assume $f \in \mathcal{Y}^{\mathcal{X}}$ be the true hypothesis, and h^* is the best hypothesis in $\mathcal{H}$: $h^* = \min_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*)$, and V is the validation set. Then we can have:

$$L_{\mathcal{D}}(h_S) = L_{\mathcal{D}}(h_S) - L_V(h_S) + L_V(h_S) - L_S(h_S) + L_S(h_S)$$

How to Fix?

- $L_{\mathcal{D}}(h_S) - L_V(h_S)$ can be bounded quite tightly
- When $L_V(h_S) - L_S(h_S)$ is *large*, the algorithm suffers from "overfitting"
- When $L_S(h_S)$ is *small*, it does not necessarily imply that $L_D(h^*)$ is small.

$$L_S(h_S) = (L_S(h_S) - L_S(h^*)) + (L_S(h^*) - L_{\mathcal{D}}(h^*)) + L_{\mathcal{D}}(h^*)$$

  - When the $VCDim(h_S)$ is d, but the training sample size $m < d$, how is the approximation error?
  - When the $VCDim(h_S)$ is d, but the training sample size $m > 2d$, how is the approximation error?

□

Boosting

Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Remarks.

- Intuitively, one can think of a weak learner as an algorithm that uses a simple '*rule of thumb*' to output a hypothesis that performs just slightly better than a random guess.

Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Remarks.

- Intuitively, one can think of a weak learner as an algorithm that uses a simple '*rule of thumb*' to output a hypothesis that performs just slightly better than a random guess.
- We will focus on weak learners of the type ERM_B for some basic class B, instead of the original $\mathscr{H}$.

Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Remarks.

- Intuitively, one can think of a weak learner as an algorithm that uses a simple '*rule of thumb*' to output a hypothesis that performs just slightly better than a random guess.
- We will focus on weak learners of the type ERM_B for some basic class B, instead of the original $\mathscr{H}$.
- ERM_B is efficiently implementable

Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Remarks.

- Intuitively, one can think of a weak learner as an algorithm that uses a simple '*rule of thumb*' to output a hypothesis that performs just slightly better than a random guess.
- We will focus on weak learners of the type ERM_B for some basic class B, instead of the original $\mathscr{H}$.
- ERM_B is efficiently implementable
- For every sample that is labelled by some hypothesis from $\mathscr{H}$, any ERM_B hypothesis will have an error of at most $\frac{1}{2} - \gamma$.

□

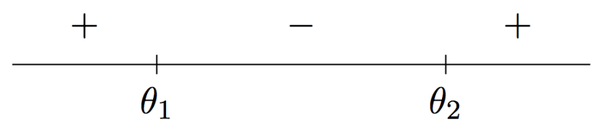
Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Example.

3-Piece Classifier $\mathscr{H}_3$ $X = \mathscr{R}$, $\mathscr{H}_3$ is the class of 3-piece classifiers,



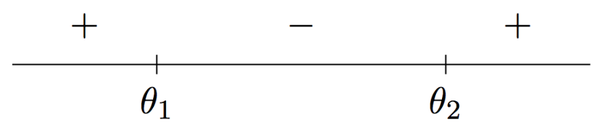
Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Example.

3-Piece Classifier $\mathscr{H}_3$ $X = \mathscr{R}$, $\mathscr{H}_3$ is the class of 3-piece classifiers,



Decision Stumps $\mathscr{H}_{DS}$ Let $B = \{x \mapsto sign(x - \theta) \cdot b : \theta \in \mathscr{R}, b \in \{\pm 1\}\}$ be the class.

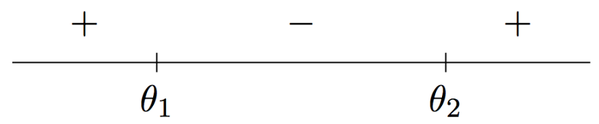
Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Example.

3-Piece Classifier $\mathscr{H}_3$ $X = \mathscr{R}$, $\mathscr{H}_3$ is the class of 3-piece classifiers,



Decision Stumps $\mathscr{H}_{DS}$ Let $B = \{x \mapsto sign(x-\theta)\cdot b : \theta \in \mathscr{R}, b \in \{\pm 1\}\}$ be the class.

- For any sample S labelled by $f \in \mathscr{H}_3$, $\exists h \in \mathscr{B}$ such that $L_S(h) \leq \frac{1}{3} = \frac{1}{2} - \frac{1}{6}$

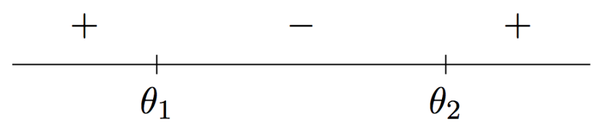
Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Example.

3-Piece Classifier $\mathscr{H}_3$ $X = \mathscr{R}$, $\mathscr{H}_3$ is the class of 3-piece classifiers,



Decision Stumps $\mathscr{H}_{DS}$ Let $B = \{x \mapsto sign(x-\theta)\cdot b : \theta \in \mathscr{R}, b \in \{\pm 1\}\}$ be the class.

- For any sample S labelled by $f \in \mathscr{H}_3$, $\exists h \in \mathscr{B}$ such that $L_S(h) \leq \frac{1}{3} = \frac{1}{2} - \frac{1}{6}$
- ERM_B is a γ-weak learner for $\mathscr{H}_3$, where $\gamma = ...?$

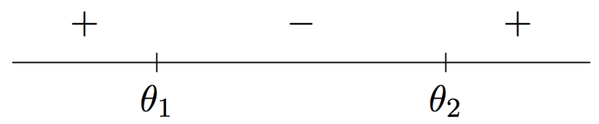
Weak Learner

An algorithm $\mathscr{A}$ is a γ-weak learner for a class $\mathscr{H}$ if for some function $m_{\mathscr{H}}(\delta)$ for every probability distribution D over $\mathscr{X}$ and every $f \in \mathscr{H}$, for every $m > m_{\mathscr{H}}(\delta)$:

$$P_{S \sim (\mathscr{D},f)^m}[L_{(\mathscr{D},f)}A(S) > \frac{1}{2} - \gamma] < \delta$$

Example.

3-Piece Classifier $\mathscr{H}_3$ $X = \mathscr{R}$, $\mathscr{H}_3$ is the class of 3-piece classifiers,



Decision Stumps $\mathscr{H}_{DS}$ Let $B = \{x \mapsto sign(x-\theta)\cdot b : \theta \in \mathscr{R}, b \in \{\pm 1\}\}$ be the class.

- For any sample S labelled by $f \in \mathscr{H}_3$, $\exists h \in \mathscr{B}$ such that $L_S(h) \leq \frac{1}{3} = \frac{1}{2} - \frac{1}{6}$
- ERM_B is a γ-weak learner for $\mathscr{H}_3$, where $\gamma = ...?$
  - Given $\gamma < \frac{1}{6}$, let ϵ be such that $\gamma + \epsilon < \frac{1}{6}$, and let $m(\gamma) = m_{\mathscr{B}}(\epsilon, \delta)$, it will guarantee that $L_{(D,f)}(ERM_B(S)) \leq L_S(ERM_B(S)) + \epsilon \leq \frac{1}{6}$

One Popular Weak Learner $\mathscr{H}_{DS}^{d}$

In practise, the most popular weak learner class is $\mathscr{H}_{DS}^{d}$, the class of Decision Stumps over $\mathscr{R}^{d}$.

One Popular Weak Learner $\mathcal{H}_{DS}^{d}$

In practise, the most popular weak learner class is $\mathcal{H}_{DS}^{d}$, the class of Decision Stumps over $\mathcal{R}^{d}$.

Remarks.

- Every $h \in \mathcal{H}_{DS}^{d}$ is determined by two parameters: $(i, r, b \in \{\pm 1\})$:

$$h_{(i,r,+1)}(x) = \begin{cases} b & \text{if } x_i < r \\ -b & \text{if } x_i \geq r \end{cases}$$

One Popular Weak Learner $\mathscr{H}_{DS}^{d}$

In practise, the most popular weak learner class is $\mathscr{H}_{DS}^{d}$, the class of Decision Stumps over $\mathscr{R}^{d}$.

Remarks.

- Every $h \in \mathscr{H}_{DS}^{d}$ is determined by two parameters: $(i, r, b \in \{\pm 1\})$:

$$h_{(i,r,+1)}(x) = \begin{cases} b & \text{if } x_i < r \\ -b & \text{if } x_i \geq r \end{cases}$$

- For every d, the class $\mathscr{H}_{DS}^{d}$ is efficiently learnable, more specifically, $ERM_{\mathscr{H}_{DS}^{d}}$ can be implemented in time $O(m \times d)$.

  Proof.

One Popular Weak Learner $\mathscr{H}_{DS}^{d}$

In practise, the most popular weak learner class is $\mathscr{H}_{DS}^{d}$, the class of Decision Stumps over $\mathscr{R}^{d}$.

Remarks.

- Every $h \in \mathscr{H}_{DS}^{d}$ is determined by two parameters: $(i, r, b \in \{\pm 1\})$:

$$h_{(i,r,+1)}(x) = \begin{cases} b & \text{if } x_i < r \\ -b & \text{if } x_i \geq r \end{cases}$$

- For every d, the class $\mathscr{H}_{DS}^{d}$ is efficiently learnable, more specifically, $ERM_{\mathscr{H}_{DS}^{d}}$ can be implemented in time $O(m \times d)$.

  Proof.

  - Input a sample with m examples: $((x_1, y_1), \cdots, (x_m, y_m)))$ where each x_i is a vector in $\mathscr{R}^{d}$ and $y_i \in \{-1, +1\}$.

One Popular Weak Learner $\mathscr{H}^d_{DS}$

In practise, the most popular weak learner class is $\mathscr{H}^d_{DS}$, the class of Decision Stumps over $\mathscr{R}^d$.

Remarks.

- Every $h \in \mathscr{H}^d_{DS}$ is determined by two parameters: $(i, r, b \in \{\pm 1\})$:

$$h_{(i,r,+1)}(x) = \begin{cases} b & \text{if } x_i < r \\ -b & \text{if } x_i \geq r \end{cases}$$

- For every d, the class $\mathscr{H}^d_{DS}$ is efficiently learnable, more specifically, $ERM_{\mathscr{H}^d_{DS}}$ can be implemented in time $O(m \times d)$.

  Proof.

  - Input a sample with m examples: $((x_1, y_1), \cdots, (x_m, y_m)))$ where each x_i is a vector in $\mathscr{R}^d$ and $y_i \in \{-1, +1\}$.
  - For each $1 \leq i \leq d$, and for each $1 \leq j \leq m$, consider $h_{(i,x_j(i),+1)}$ and $h_{(i,x_j(i),-1)}$

One Popular Weak Learner $\mathscr{H}^d_{DS}$

In practise, the most popular weak learner class is $\mathscr{H}^d_{DS}$, the class of Decision Stumps over $\mathscr{R}^d$.

Remarks.

- Every $h \in \mathscr{H}^d_{DS}$ is determined by two parameters: $(i, r, b \in \{\pm 1\})$:

$$h_{(i,r,+1)}(x) = \begin{cases} b & \text{if } x_i < r \\ -b & \text{if } x_i \geq r \end{cases}$$

- For every d, the class $\mathscr{H}^d_{DS}$ is efficiently learnable, more specifically, $ERM_{\mathscr{H}^d_{DS}}$ can be implemented in time $O(m \times d)$.

  Proof.

  - Input a sample with m examples: $((x_1, y_1), \cdots, (x_m, y_m)))$ where each x_i is a vector in $\mathscr{R}^d$ and $y_i \in \{-1, +1\}$.
  - For each $1 \leq i \leq d$, and for each $1 \leq j \leq m$, consider $h_{(i,x_j(i),+1)}$ and $h_{(i,x_j(i),-1)}$
  - Compute $L_S(h)$ for each such h, output the one with minimum loss. The time cost will be $O(2(m+1) \times d \times m)$, which is polynomial, hence efficient.

The Problem of Boosting

Suppose an algorithm $\mathscr{A}$ is a γ-weak learner for some class $\mathscr{H}$. Can we use $\mathscr{A}$ to construct a strong learner? If $\mathscr{A}$ is computationally efficient, can we boost it efficiently?

The Problem of Boosting

Suppose an algorithm $\mathscr{A}$ is a γ-weak learner for some class $\mathscr{H}$. Can we use $\mathscr{A}$ to construct a strong learner? If $\mathscr{A}$ is computationally efficient, can we boost it efficiently?

Two questions.

- Boosting the confidence δ;

The Problem of Boosting

Suppose an algorithm $\mathscr{A}$ is a γ-weak learner for some class $\mathscr{H}$. Can we use $\mathscr{A}$ to construct a strong learner? If $\mathscr{A}$ is computationally efficient, can we boost it efficiently?

Two questions.

- Boosting the confidence δ;
- Boosting the accuracy ϵ;

□

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Steps. Aim: at least $1-\delta$, we have $L_D(\hat{h}) \leq \epsilon_0 + \epsilon$.

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Steps. Aim: at least $1-\delta$, we have $L_D(\hat{h}) \leq \epsilon_0 + \epsilon$.

1. Apply $\mathscr{A}$ on k i.i.d. samples, each of $m_{\mathscr{H}}(\delta_0)$ examples, to obtain $h_1, \cdots, h_k$.

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Steps. Aim: at least $1-\delta$, we have $L_D(\hat{h}) \leq \epsilon_0 + \epsilon$.

1. Apply $\mathscr{A}$ on k i.i.d. samples, each of $m_{\mathscr{H}}(\delta_0)$ examples, to obtain $h_1, \cdots, h_k$.
2. Take additional validation sample of size m_V and output $\hat{h} \in \arg min_{h_i} L_V(h_i)$.

□

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Proof.

- First, when we have $m_V \geq \frac{2\log(4k/\delta)}{\epsilon^2}$, the validation procedure guarantees

$$P[L_{\mathscr{D}}(\hat{h}) > \min_i L_{\mathscr{D}}(\hat{h}_i) + \epsilon] \leq \frac{\delta}{2}$$

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Proof.

- First, when we have $m_V \geq \frac{2\log(4k/\delta)}{\epsilon^2}$, the validation procedure guarantees

$$P[L_{\mathscr{D}}(\hat{h}) > \min_i L_{\mathscr{D}}(\hat{h}_i) + \epsilon] \leq \frac{\delta}{2}$$

- From the weak-learner, we have:

$$P[\min_i L_{\mathscr{D}}(\hat{h}_i) > \epsilon_0] = P[\forall_i L_{\mathscr{D}}(\hat{h}_i) > \epsilon_0] = \prod_{i=1}^{k} P[L_{\mathscr{D}}(\hat{h}_i) > \epsilon_0] \leq \delta_0^k \leq \frac{\delta}{2}$$

This is valid when $k \geq \lceil \frac{\log(2/\delta)}{\log(1/\delta_0)} \rceil$.

Boosting the Confidence

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario, it is $m_{\mathscr{H}}(\epsilon_0, \delta_0)$ with $\epsilon_0 = \frac{1}{2} - \gamma_0$ is fixed. For any $\epsilon, \delta \in (0,1)$, we show how to learn $\mathscr{H}$ to accuracy $\epsilon_0 + \epsilon$ with confidence δ.

Proof.

- First, when we have $m_V \geq \frac{2\log(4k/\delta)}{\epsilon^2}$, the validation procedure guarantees

$$P[L_{\mathscr{D}}(\hat{h}) > \min_i L_{\mathscr{D}}(\hat{h}_i) + \epsilon] \leq \frac{\delta}{2}$$

- From the weak-learner, we have:

$$P[\min_i L_{\mathscr{D}}(\hat{h}_i) > \epsilon_0] = P[\forall_i L_{\mathscr{D}}(\hat{h}_i) > \epsilon_0] = \prod_{i=1}^{k} P[L_{\mathscr{D}}(\hat{h}_i) > \epsilon_0] \leq \delta_0^k \leq \frac{\delta}{2}$$

  This is valid when $k \geq \lceil \frac{\log(2/\delta)}{\log(1/\delta_0)} \rceil$.
- Apply the union bound, for at least $1-\delta$, we have $L_{\mathscr{D}}(\hat{h}) \leq \epsilon_0 + \epsilon$.

□

Boosting the Accuracy

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario:

Input $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, where for each i, $y_i = f(x_i)$

Output call $\mathscr{A}$ and get hypothesis h with small error $L_S(h)$ on S

Boosting the Accuracy

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario:

Input $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, where for each i, $y_i = f(x_i)$
Output call $\mathscr{A}$ and get hypothesis h with small error $L_S(h)$ on S

History.

- Problem raised by *Kearns* and *Valiant* in 1988

Boosting the Accuracy

Suppose an algorithm $\mathscr{A}$ is a γ_0-weak learner for some class $\mathscr{H}$ that requires $m_{\mathscr{H}}(\delta_0)$ examples, namely in the PAC learning scenario:

Input $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, where for each i, $y_i = f(x_i)$
Output call $\mathscr{A}$ and get hypothesis h with small error $L_S(h)$ on S

History.

- Problem raised by *Kearns* and *Valiant* in 1988
- Solved in 1990 by then MIT student, Robert *Schapire*; In 1995 improved into AdaBoost by *Schapire* and *Freund*

AdaBoost Algorithm

Input Training set $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, a weaker learner WL, number of rounds T

Output $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

AdaBoost Algorithm

Input Training set $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, a weaker learner WL, number of rounds T

Output $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

Algorithm.

- Initialize $D^{(1)} = (\frac{1}{m}, \cdots, \frac{1}{m})$

AdaBoost Algorithm

Input Training set $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, a weaker learner WL, number of rounds T
Output $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

Algorithm.

- Initialize $D^{(1)} = (\frac{1}{m}, \cdots, \frac{1}{m})$
- For $t = 1, \cdots, T$:
  - invoke weak learner $h_t = WL(D^{(t)}, S)$
  - compute $\epsilon_t = L_{D^{(t)}}(h_t) = \sum_{i=1}^{m} D_i^{(t)} 1_{[y_i \neq h_t(x_i)]}$
  - let $\omega_t = \frac{1}{2} \log(\frac{1}{\epsilon_t} - 1)$
  - update $D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\omega_t y_i h_t(x_i)}}{\sum_{j=1}^{m} D_j^{(t)} e^{-\omega_t y_j h_t(x_j)}}$ for all $i = 1, \ldots, m$

AdaBoost Algorithm

Input Training set $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, a weaker learner WL, number of rounds T
Output $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

Algorithm.

- Initialize $D^{(1)} = (\frac{1}{m}, \cdots, \frac{1}{m})$
- For $t = 1, \cdots, T$:
  - invoke weak learner $h_t = WL(D^{(t)}, S)$
  - compute $\epsilon_t = L_{D^{(t)}}(h_t) = \sum_{i=1}^{m} D_i^{(t)} 1_{[y_i \neq h_t(x_i)]}$
  - let $\omega_t = \frac{1}{2} \log(\frac{1}{\epsilon_t} - 1)$
  - update $D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\omega_t y_i h_t(x_i)}}{\sum_{j=1}^{m} D_j^{(t)} e^{-\omega_t y_j h_t(x_j)}}$ for all $i = 1, \ldots, m$
- Output the hypothesis $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

□

AdaBoost Algorithm

Input Training set $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, a weaker learner WL, number of rounds T

Output $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

Intuition. AdaBoost forces WL to focus on problematic examples

- The error of h_t with respect to $D^{(t+1)}$ is exactly 1/2.

AdaBoost Algorithm

Input Training set $S = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, a weaker learner WL, number of rounds T
Output $h_S(x) = sign(\sum_{t=1}^{T} \omega_t h_t(x))$

Intuition. AdaBoost forces WL to focus on problematic examples

- The error of h_t with respect to $D^{(t+1)}$ is exactly 1/2.
  - Considering the fact that:

$$\sum_{i=1}^{m} D_i^{(t+1)} 1_{[y_i \neq h_t(x_i)]} = \frac{\sum_{i=1}^{m} D_i^{(t)} e^{-\omega_t y_i h_t(x_i)} 1_{[y_i \neq h_t(x_i)]}}{\sum_{j=1}^{m} D_j^{(t)} e^{-\omega_t y_j h_t(x_j)}} = \frac{e^{\omega_t} \epsilon_t}{e^{\omega_t} \epsilon_t + e^{-\omega_t}(1 - \epsilon_t)}$$

$$= \frac{\epsilon_t}{\epsilon_t + e^{-2\omega_t}(1 - \epsilon_t)} = \frac{\epsilon_t}{\epsilon_t + \frac{\epsilon_t}{1-\epsilon_t}(1 - \epsilon_t)} = \frac{1}{2}$$

□

AdaBoost Algorithm

Let S be a training set and assume that at each iteration of AdaBoost, the γ-weak learner $\mathscr{A}$ returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$, then the training error of the output hypothesis of AdaBoost is at most

$$L_S(h_S) = \frac{1}{m}\sum_{i=1}^{m} 1_{[h_s(x_i)\neq y_i]} \leq e^{-2\gamma^2 T}$$

AdaBoost Algorithm

Let S be a training set and assume that at each iteration of AdaBoost, the γ-weak learner $\mathscr{A}$ returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$, then the training error of the output hypothesis of AdaBoost is at most

$$L_S(h_S) = \frac{1}{m}\sum_{i=1}^{m} 1_{[h_s(x_i)\neq y_i]} \leq e^{-2\gamma^2 T}$$

Remarks. Proof to be seen in UML, P106-107

- For any $\epsilon > 0$ and $\gamma \in (0, 1/2)$, if $T \geq \frac{\log(1/\epsilon)}{2\gamma^2}$, then AdaBoost will output a hypothesis h_S with $L_S(h_S) \leq \epsilon$.

AdaBoost Algorithm

Let S be a training set and assume that at each iteration of AdaBoost, the γ-weak learner $\mathscr{A}$ returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$, then the training error of the output hypothesis of AdaBoost is at most

$$L_S(h_S) = \frac{1}{m}\sum_{i=1}^{m} 1_{[h_s(x_i)\neq y_i]} \leq e^{-2\gamma^2 T}$$

Remarks. Proof to be seen in UML, P106-107

- For any $\epsilon > 0$ and $\gamma \in (0, 1/2)$, if $T \geq \frac{\log(1/\epsilon)}{2\gamma^2}$, then AdaBoost will output a hypothesis h_S with $L_S(h_S) \leq \epsilon$.
- Since the weak learner is involved on a distribution over S, in many cases δ can be 0. In any case, by "Boosting the confidence", we can assume without loss of generality that δ is very small.

AdaBoost Algorithm

Let S be a training set and assume that at each iteration of AdaBoost, the γ-weak learner $\mathscr{A}$ returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$, then the training error of the output hypothesis of AdaBoost is at most

$$L_S(h_S) = \frac{1}{m}\sum_{i=1}^{m} 1_{[h_s(x_i)\neq y_i]} \leq e^{-2\gamma^2 T}$$

Remarks. Proof to be seen in UML, P106-107

- For any $\epsilon > 0$ and $\gamma \in (0, 1/2)$, if $T \geq \frac{\log(1/\epsilon)}{2\gamma^2}$, then AdaBoost will output a hypothesis h_S with $L_S(h_S) \leq \epsilon$.
- Since the weak learner is involved on a distribution over S, in many cases δ can be 0. In any case, by "Boosting the confidence", we can assume without loss of generality that δ is very small.
- How about $L_D(h_S)$? Need to check VC Dimension.

□

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Remarks.

- Since the weak learner $\mathscr{A}$ is invoked only on distributions over S, we can assume without loss of generality that $B = \{g_1, \cdots, g_d\}$ for some $d \leq 2^m$.

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Remarks.

- Since the weak learner $\mathscr{A}$ is invoked only on distributions over S, we can assume without loss of generality that $B = \{g_1, \cdots, g_d\}$ for some $d \leq 2^m$.
- Denote $\psi(x) = (g_1(x), \cdots, g_d(x))$, we have

  $$L(B,T) = \{x \mapsto sign(< \omega, \psi(x) >) : \omega \in \mathscr{R}^d, ||\omega||_0 \leq T\}$$

  where $||\omega||_0 = |\{i : \omega_i \neq 0\}|$

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Remarks.

- Since the weak learner $\mathscr{A}$ is invoked only on distributions over S, we can assume without loss of generality that $B = \{g_1, \cdots, g_d\}$ for some $d \leq 2^m$.
- Denote $\psi(x) = (g_1(x), \cdots, g_d(x))$, we have

  $$L(B,T) = \{x \mapsto sign(< \omega, \psi(x) >) : \omega \in \mathscr{R}^d, ||\omega||_0 \leq T\}$$

  where $||\omega||_0 = |\{i : \omega_i \neq 0\}|$
- That is, AdaBoost learns a composition of the class of halfspaces with sparse coefficients over the mapping $x \mapsto \psi(x)$.

□

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Example. Suppose $X = \mathscr{R}$ and B is Decision Stumps:

$$B = \{x \mapsto sign(x - \theta) \cdot \omega : \theta \in B, \omega \in \{\pm 1\}\}$$

- Composing half spaces on top of simple classes can be very expressive!

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Example. Suppose $X = \mathscr{R}$ and B is Decision Stumps:

$$B = \{x \mapsto sign(x - \theta) \cdot \omega : \theta \in B, \omega \in \{\pm 1\}\}$$

- Composing half spaces on top of simple classes can be very expressive!
- Let G_T be the class of piece-wise constant functions with T pieces. Then we have $G_T \subseteq L(B,T)$.

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Example. Suppose $X = \mathscr{R}$ and B is Decision Stumps:

$$B = \{x \mapsto sign(x - \theta) \cdot \omega : \theta \in B, \omega \in \{\pm 1\}\}$$

- Composing half spaces on top of simple classes can be very expressive!
- Let G_T be the class of piece-wise constant functions with T pieces. Then we have $G_T \subseteq L(B,T)$.
- We can design the function and weights as below:

AdaBoost Algorithm

Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Example. Suppose $X = \mathscr{R}$ and B is Decision Stumps:

$$B = \{x \mapsto sign(x - \theta) \cdot \omega : \theta \in B, \omega \in \{\pm 1\}\}$$

- Composing half spaces on top of simple classes can be very expressive!
- Let G_T be the class of piece-wise constant functions with T pieces. Then we have $G_T \subseteq L(B,T)$.
- We can design the function and weights as below:
  - we set $\omega_0 = 0.5$

AdaBoost Algorithm

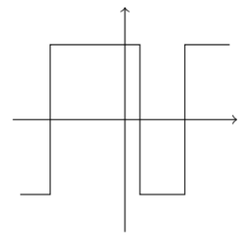
Given a base hypothesis class B for the weak learner $\mathscr{A}$, the output of AdaBoost will be a member of the following class:

$$L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$$

Example. Suppose $X = \mathscr{R}$ and B is Decision Stumps:

$$B = \{x \mapsto sign(x - \theta) \cdot \omega : \theta \in B, \omega \in \{\pm 1\}\}$$

- Composing half spaces on top of simple classes can be very expressive!
- Let G_T be the class of piece-wise constant functions with T pieces. Then we have $G_T \subseteq L(B,T)$.
- We can design the function and weights as below:
  - we set $\omega_0 = 0.5$
  - for $t > 1$, we set $\omega_t = (-1)^t$.



AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Remarks.

- We have argued that the expressiveness of $L(B,T)$ grows with T

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Remarks.

- We have argued that the expressiveness of $L(B,T)$ grows with T
- Namely, the approximation error decreases with T

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Remarks.

- We have argued that the expressiveness of $L(B,T)$ grows with T
- Namely, the approximation error decreases with T
- Here, VC Dimensional shows that the estimation error increases with T

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Remarks.

- We have argued that the expressiveness of $L(B,T)$ grows with T
- Namely, the approximation error decreases with T
- Here, VC Dimensional shows that the estimation error increases with T
- Therefore, the parameter T of AdaBoost enables us to control the bias-complexity tradeoff.

□

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Proof.

- Let $C = \{x_1, \cdots, x_m\}$ be a set that is shattered by $L(B,T)$; Each labelling of C by $h \in L(B,T)$ is obtained by first choosing base hypothesis $h_i \in B$, and then applied the halfspace hypothesis over the vector $(h_1(x), \cdots, h_T(x))$.

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Proof.

- Let $C = \{x_1, \cdots, x_m\}$ be a set that is shattered by $L(B,T)$; Each labelling of C by $h \in L(B,T)$ is obtained by first choosing base hypothesis $h_i \in B$, and then applied the halfspace hypothesis over the vector $(h_1(x), \cdots, h_T(x))$.

1. By Sauer's lemma, at most $(em/d)^d$ different dichotomies; at most $(em/d)^{dT}$ ways to select T from them

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Proof.

- Let $C = \{x_1, \cdots, x_m\}$ be a set that is shattered by $L(B,T)$; Each labelling of C by $h \in L(B,T)$ is obtained by first choosing base hypothesis $h_i \in B$, and then applied the halfspace hypothesis over the vector $(h_1(x), \cdots, h_T(x))$.
  1. By Sauer's lemma, at most $(em/d)^d$ different dichotomies; at most $(em/d)^{dT}$ ways to select T from them
  2. For the second step, by Sauer's lemma, at most $(em/T)^T$ different dichotomies;

  Namely, $(em/d)^{dT} \times (em/T)^T \leq m^{(d+1)T}$

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Proof.

- Let $C = \{x_1, \cdots, x_m\}$ be a set that is shattered by $L(B,T)$; Each labelling of C by $h \in L(B,T)$ is obtained by first choosing base hypothesis $h_i \in B$, and then applied the halfspace hypothesis over the vector $(h_1(x), \cdots, h_T(x))$.
  1. By Sauer's lemma, at most $(em/d)^d$ different dichotomies; at most $(em/d)^{dT}$ ways to select T from them
  2. For the second step, by Sauer's lemma, at most $(em/T)^T$ different dichotomies;

  Namely, $(em/d)^{dT} \times (em/T)^T \leq m^{(d+1)T}$
- Since we assume C is shattered, so we have $2^m \leq (em/d)^d T \times (em/T)^T \leq m^{(d+1)T}$

AdaBoost Algorithm

Let B be a base class with VC dimension $VCDim(B) = d$, and let $L(B,T)$ be: $L(B,T) = \{x \mapsto sign(\sum_{t=1}^{T} \omega_t h_t(x)) : \omega \in \mathscr{R}^T, \forall t, h_t \in B\}$. Assume that both T and d are at least 3, then

$$VCDim(L(B,T)) \leq T(d+1)(3\log(T(d+1)) + 2)$$

Proof.

- Let $C = \{x_1, \cdots, x_m\}$ be a set that is shattered by $L(B,T)$; Each labelling of C by $h \in L(B,T)$ is obtained by first choosing base hypothesis $h_i \in B$, and then applied the halfspace hypothesis over the vector $(h_1(x), \cdots, h_T(x))$.
  1. By Sauer's lemma, at most $(em/d)^d$ different dichotomies; at most $(em/d)^{dT}$ ways to select T from them
  2. For the second step, by Sauer's lemma, at most $(em/T)^T$ different dichotomies;

  Namely, $(em/d)^{dT} \times (em/T)^T \leq m^{(d+1)T}$
- Since we assume C is shattered, so we have $2^m \leq (em/d)^d T \times (em/T)^T \leq m^{(d+1)T}$
- m can not be too big: $m \leq \log(m)\frac{(d+1)T}{\log(2)}$

□

Quiz

The VC Dimension

- Let $\mathcal{H}_1,\ldots,\mathcal{H}_r$ be hypothesis classes over some fixed domain set $\mathcal{X}$. Let $d = max_i VCDim(\mathcal{H}_i)$ and assume for simplicity that $d \geq 3$.

The VC Dimension

- Let $\mathcal{H}_1, \ldots, \mathcal{H}_r$ be hypothesis classes over some fixed domain set $\mathcal{X}$. Let $d = max_i VCDim(\mathcal{H}_i)$ and assume for simplicity that $d \geq 3$.

  1. Prove that for $r = 2$, it holds that

  $$VCDim(\mathcal{H}_\infty \cup \mathcal{H}_\in) \leq 2d + 1$$

The VC Dimension

- Let $\mathcal{H}_1, \ldots, \mathcal{H}_r$ be hypothesis classes over some fixed domain set $\mathcal{X}$. Let $d = max_i VCDim(\mathcal{H}_i)$ and assume for simplicity that $d \geq 3$.

  1. Prove that for $r = 2$, it holds that

     $$VCDim(\mathcal{H}_\infty \cup \mathcal{H}_\in) \leq 2d + 1$$

  2. Prove that

     $$VCDim(\cup_{i=1}^{r} \mathcal{H}_i \leq 4d \log(2d) + 2\log(r)$$

Questions?

Contact Information

Associate Professor **Gang Li**
School of Information Technology
Deakin University
Geelong, Victoria 3216, Australia

gangli@tulip.org.au
Open Resources of TULIP-Lab
Team for Universal Learning and Intelligent Processing