MACHINE LEARNING

Vasant Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science
Bioinformatics and Computational Biology Program
Center for Computational Intelligence, Learning, & Discovery
Iowa State University
honavar@cs.iastate.edu
www.cs.iastate.edu/~honavar/
www.cild.iastate.edu/

Mistake and Loss Bound Models of Learning

- Outline
- Machine learning and theories of learning
- Mistake bound model of learning
- Mistake bound analysis of conjunctive concept learning
- Weighted majority and related multiplicative update algorithms
- Applications
Computational Models of Learning

- **Model of the Learner**: Computational capabilities, sensors, effectors, knowledge representation, inference mechanisms, prior knowledge, etc.
- **Model of the Environment**: Tasks to be learned, information sources (teacher, queries, experiments), performance measures
- **Key questions**: Can a learner with a certain structure learn a specified task in a particular environment? Can the learner do so efficiently? If so, how? If not, why not?

Models of Learning: What are they good for?

- To make explicit relevant aspects of the learner and the environment
- To identify easy and hard learning problems (and the precise conditions under which they are easy or hard)
- To guide the design of learning systems
- To shed light on natural learning systems
- To help analyze the performance of learning systems
Mistake Bound Analysis

Example – Learning Conjunctive Concepts

- Given an arbitrary, noise-free sequence of labeled examples \((X_1, C(X_1)), (X_2, C(X_2)) \ldots (X_m, C(X_m))\) of an unknown binary conjunctive concept \(C\) over \(\{0,1\}^N\), the learner's task is to predict \(C(X)\) for a given \(X\).

Theorem: Exact online learning of conjunctive concepts can be accomplished with at most \((N+1)\) prediction mistakes.

Online learning of conjunctive concepts

Algorithm A.1

- Initialize \(L = \{X_1, \neg X_1, \ldots, X_N, \neg X_N\}\)
- Predict according to match between an instance and the conjunction of literals in \(L\)
- Whenever a mistake is made on a positive example, drop the offending literals from \(L\)

Example

\((0111, 1)\) will result in \(L = \{\neg X_1, X_2, X_3, X_4\}\)

\((1110, 1)\) will yield \(L = \{X_2, X_3\}\)
Mistake bound analysis of conjunctive concept learning

Proof Sketch
• No literal in \( C \) is ever eliminated from \( L \)
• Each mistake eliminates at least one literal from \( L \)
• The first mistake eliminates \( N \) of the \( 2N \) literals
• Conjunctive concepts can be learned with at most \((N+1)\) mistakes

Conclusion
• Conjunctive concepts are easy to learn in the mistake bound model

Optimal Mistake Bound Learning Algorithms

Definition: An **optimal mistake bound** \( m_{bound}(C) \) for a concept class \( C \) is the **lowest possible** mistake bound in the worst case (considering all concepts in \( C \), and all possible sequences \( D \) of examples).

\[
m_{bound}(C) = \min_{\text{learners}} \max_{c \in C} \max_{\text{sequences } D} \text{mistakes } (c^*, L, D)
\]

where mistakes \((c^*, L, D)\) is the number of mistakes made by \( L \) in its attempt to learn \( c^* \) based on the sequence of examples provided.
Mistake Bounds and optimal mistake bounds

\[ m\text{bound}(C) \leq 2^N \] (why?) \text{trivial bounds}

\[ m\text{bound}(C) \leq |C| - 1 \] (why?)

\[ m\text{bound}(C) \leq \log(|C|) \text{ we will prove this} \]

Definition: An optimal learning algorithm for a concept class \( C \) (in the mistake bound framework) is one that is \textit{guaranteed to exactly} learn any concept in \( C \), using any noise-free example sequence, with at most \( O(\text{mbound}(C)) \) mistakes.

Version space and Halving algorithm

\( V_i = \{ c \in C | c \) is consistent with the first \( i \) examples \} \)

\( V_0 = C \)

\( \xi_0(C, X) = \{ c \in C : c(X) = 0 \} \)

\( \xi_1(C, X) = \{ c \in C : c(X) = 1 \} \)

For \( i > 0 \),

\[ V_i = \begin{cases} \xi_0(V_{i-1}, X_i) & \text{if } c^*(X_i) = 0 \\ \xi_1(V_{i-1}, X_i) & \text{if } c^*(X_i) = 1 \end{cases} \]

\( V_i \) is a subset of \( C \) and \( \xi_0 \) and \( \xi_1 \) are functions that select concepts.

Halving Algorithm: On input \( X_i \), Predict 1 if

\[ |\xi_1(V_{i-1}, X_i)| \geq |\xi_0(V_{i-1}, X_i)| \]

and 0 otherwise. Eliminate the concepts (majority or minority) that were wrong.
Halving Algorithm

Definition: The halving algorithm predicts according to the majority of concepts in the current version space and a mistake results in elimination of all the offending concepts from the version space.

\[ V_i = \{ c \in C \mid c \text{ is consistent with the first } i \text{ examples} \} \]

\[ V_0 = C \]

\[ \xi_0(C, X) = \{ c \in C : c(X) = 0 \} \]

\[ \xi_1(C, X) = \{ c \in C : c(X) = 1 \} \]

For \( i > 0 \), \( V_i = \begin{cases} 
\xi_0(V_{i-1}, X_i) & \text{if } c^*(X_i) = 0 \\
\xi_1(V_{i-1}, X_i) & \text{if } c^*(X_i) = 1 
\end{cases} \)

---

The Halving Algorithm

Theorem: \( m_{\text{bound}}(C) \leq \log(|C|) \)

Proof:
- The halving algorithm predicts according to majority of concepts in the version space
- Each mistake eliminates at least half of the candidate hypotheses in the version space
- The number of mistakes is bounded by \( \log(|C|) \)

The halving algorithm can be computationally feasible if there is a way to compactly represent and efficiently manipulate the version space. Otherwise it is not computationally feasible.
The Halving Algorithm

- The halving algorithm is not optimal with respect to the number of mistakes. In order to minimize the number of mistakes, the learner has to guess according to the subset of the version space that is expected to yield the fewest mistakes.

- The optimal mistake bound algorithm has to predict 1 if

\[
m_{bound}(\xi_1(V_{i-1}, X_i)) \geq m_{bound}(\xi_0(V_{i-1}, X_i))
\]

and 0 otherwise. However this algorithm is even less efficient.

---

The Halving Algorithm

- **Question:** Are there any efficiently implementable learning algorithms with mistake bounds comparable to that of the halving algorithm?

- **Answer:** Littlestone's algorithm for learning *monotone disjunctions* of at most \(k\) of \(n\) literals using the hypothesis class of threshold functions with at most \((k \lg n)\) mistakes. More on this later.
Randomized Halving Algorithm

- The predictions made by the halving algorithm may not be based on any concept in $C$. There may not exist in $C$ a concept that is consistent with the majority vote.
- The randomized halving algorithm due to Maass predicts according to a randomly selected concept $c \in C$.
- All concepts in $C$ that are inconsistent with the example are eliminated from further consideration.
- **Theorem:** The expected number of mistakes made by the randomized halving algorithm is at most $\log |C| + O(1)$.

WLOG assume that the order of presentation of the examples is independent of the learner’s actions.

Suppose the concepts in the version space are ordered by when they are going to be eliminated by examples.

Let $c_i, ..., c_r$ be the order (so $r = |V_i|$).

Let $M_r$ be the expected number of mistakes.
Randomized Halving Algorithm

- The randomized halving algorithm picks one of the $r$ concepts at random with probability equal to $1/r$.
- Suppose $c_r$ is chosen
  - $c_r$ is the target concept and hence there can be no further mistakes.
- One of the other concepts is chosen with probability $(r-1)/r$
  In this case,
  - There will be one mistake (at least)
  - Plus the expected number of mistakes for the remaining concepts

Randomized Halving Algorithm

\[
M_i = 0
\]

\[
M_r = \left( \frac{1}{r} \right) (0) + \left( \frac{r-1}{r} \right) \left( 1 + \frac{\sum_{i=1}^{r-1} M_i}{(r-1)} \right) = \left( \frac{r-1}{r} \right) + \frac{\sum_{i=1}^{r-1} M_i}{r}
\]

\[
rM_r = (r-1) + \sum_{i=1}^{r-1} M_i;
\]

\[
(r-1)M_{r-1} = (r-2) + \sum_{i=1}^{r-2} M_i
\]

\[
r(M_r - M_{r-1}) + M_{r-1} = 1 + M_{r-1}
\]

\[
M_r = M_{r-1} + \frac{1}{r}
\]

\[
M_r = \sum_{i=1}^{r} \left( \frac{1}{r} \right) = \ln r + O(1)
\]
Learning monotone disjunctions when irrelevant attributes abound

\[ C = \{x_i \lor x_{i_2} \lor \ldots \lor x_{i_k} \mid i_j \in \{1, \ldots, N\}; j \in \{1, \ldots, k\}\} \]

\[ |C| = \binom{N}{k} + \binom{N}{k-1} + \ldots + \binom{N}{0} \]

\[ \lg |C| = \Theta(k \lg N) \]

How can we design an algorithm that achieves this mistake bound?

Winnow Algorithm

- **Observation** – Monotone disjunctions are a subset of threshold functions
- **Idea** – Use threshold neurons to learn monotone disjunctions

Initialize \( \theta = \left(\frac{N}{2}\right); \ W = (1, \ldots, 1) \)

**Predict** \( y(X) = 1 \) iff \( W.X > \theta \) otherwise predict \( y(X) = 0 \)

If \( c(X) = 1 \) but \( y(X) = 0 \), double all \( w_i \) where \( x_i = 1 \)

If \( c(X) = 0 \) but \( y(X) = 1 \), zero out all \( w_i \) where \( x_i = 1 \)

**Theorem** – Winnow makes \( O(k \lg N) \) mistakes

Copyright Vasant Honavar, 2006.
Winnow Algorithm

\[ 0 \leq \left( \sum_{i=1}^{N} w_i \right) \leq (N + u\theta - v\theta) \]

\[ \forall i \ w_i \leq 2\theta \]

\[ \exists w_i \ lg \ w_i \geq \left( \frac{u}{k} \right) \]

\[ \left( \frac{u}{k} \right) \leq (lg \ w_i) \leq (lg \theta + 1) \]

\[ u \leq k(\lg \theta + 1) \]

\[ \nu \leq \left( \frac{N}{\theta} \right) + k(\lg \theta + 1) \]

\[ (u + \nu) \leq 2 + 2k \lg N \]

\( u \) – number of times weights are doubled

\( v \) – number of times weights are zeroed out

Each weight doubling adds at most \( \theta \) to the sum of weights and each zeroed out weight subtracts at least \( \theta \) from the sum of weights.

No weight that is greater than \( \theta \) is ever doubled.

Each weight doubling has to affect at least one of the weights. Each weight doubling adds at least 1 to the logarithm of the weight that got doubled.

Theorem – Winnow makes \( O(k \lg N) \) mistakes.

Generalizations of Winnow

- Winnow algorithm and its variants and generalizations can be used to learn concepts from more expressive concept classes by preprocessing the input patterns
  - e.g. by transforming an \( n \)-bit pattern into an \( O(n^k) \) bit pattern that encodes all conjunctions of at most \( k \) literals (negated or un negated)
- Winnow algorithm and its variants can be made to generalize better by incorporating regularization
Weighted majority learning algorithm (WML)

Motivations
• Robust algorithm for learning monotone disjunctions
• Suppose we have a pool of predictors – features, experts, algorithms… The optimal predictor – that is, the predictor that makes the fewest mistakes depends on the data and is not known a priori
• Basic idea
  – make predictions based on a weighted majority of predictions of all predictors in the pool
  – if a mistake is made, adjust the weights

Initialize \( W = (1, ..., 1); \quad \theta_0 = \theta_1 = 0 \)

For each training example \((X, c(X))\)
  If \( x_i = 0 \), \( \theta_0 \leftarrow \theta_0 + w_i \)
  If \( x_i = 1 \), \( \theta_1 \leftarrow \theta_1 + w_i \)
  Predict \( y(X) = 1 \) if \( \theta_1 > \theta_0 \)
  Predict \( y(X) = 0 \) if \( \theta_1 < \theta_0 \)
  Predict \( y(X) = \text{Random} \{0, 1\} \) if \( \theta_1 = \theta_0 \)
  If \( c(X) \neq y(X) \), \( w_i \leftarrow \beta w_i \) \((0 < \beta < 1)\)
Weighted majority learning algorithm (WML)

**Theorem:**
- Let \( D \) be any sequence of training examples.
- Let \( A = \{A_1 \ldots A_n\} \) be any pool of \( n \) predictors.
- Let \( k \) be the number of mistakes made by the best predictor in the pool \( A \) on the sequence of examples \( D \).
- Then

\[
m = \text{mbound}_{WML} \leq \left( k \log \left( \frac{1}{\beta} \right) + \log n \right) \frac{\log \frac{2}{1+\beta}}{\log \frac{1}{1-\beta}}
\]

**Proof:** The total weight associated with the \( n \) predictors at any time is \( W = (\theta_0 + \theta_1) \) where \( \theta_0 \) and \( \theta_1 \) are as defined in WML.

Consider an example on which a mistake is made. WLOG, assume that the prediction was 0. Then the total weight after the update is

\[
W_{\text{after}} \leq (\theta_0 + \theta_1) - (1-\beta) \left( \frac{\theta_0 + \theta_1}{2} \right) \quad \text{(why?)}
\]

\[
W_{\text{after}} \leq \left( \frac{1 + \beta}{2} \right) (\theta_0 + \theta_1) = \left( \frac{1 + \beta}{2} \right) W_{\text{before}}
\]

The predictors in the weighted majority must have held at least half the total weight. After update, this portion of the weight gets reduced by a factor \((1-\beta)\).

Each mistake causes sum of weights after an update to be no more than \( \left( \frac{1 + \beta}{2} \right) \) times the value before the update.
Weighted majority learning algorithm (WML)

\[ W_{after} \leq \left( \frac{1 + \beta}{2} \right) W_{before} \]

where \( m \) is the number of updates

\[ W_{final} \leq \left( \frac{1 + \beta}{2} \right)^m W_{init} = \left( \frac{1 + \beta}{2} \right)^n \]

The best predictor (say \( w_j \)) makes \( k \) mistakes and hence undergoes \( k \) weight updates. So \( w_{j, final} = \beta^k \)

Clearly, the weight of the best predictor can be no greater than the sum of weights

\[ \beta^k \leq \left( \frac{1 + \beta}{2} \right)^n \]

Implication: WML makes almost as few mistakes as the optimal learner in the pool – We can use weighted majority when it is unclear which learner in the pool is optimal
Some Variations on WML

Selection from countably infinite pool of predictors

- No algorithm (that halts) can obtain predictions from an infinite number of predictors.
- However, we can modify WML so that it considers successively larger pools of predictors.
- The modified algorithm behaves very much like WML with a degradation in mistake bound of the order of \((\log i)\) where \(i\)th predictor in the pool is the optimal predictor.

Randomized predictions

- Predict 1 with probability \(\frac{\theta_1}{\theta_1 + \theta_0}\)
- Predict 0 with probability \(\frac{\theta_0}{\theta_1 + \theta_0}\)

- The update equations can be modified so that the rate of mistakes approaches arbitrarily close to the rate of mistakes of the best predictor in the pool.
Some Variations on WML – Balanced Winnow

Inputs and outputs are bipolar

Balanced Winnow \((W^+, W^-, (X, y))\)

\[0 < \beta < 1\]

If \(\text{sign}(W^+ \cdot X - W^- \cdot X) \neq y\)

If \(y = 1\), \(\forall i\) \(w_i^+ \leftarrow w_i^+ + \beta^{-x_i}w_i^+; w_i^- \leftarrow w_i^- + \beta^{x_i}w_i^-\)

If \(y = -1\), \(\forall i\) \(w_i^- \leftarrow w_i^- + \beta^xw_i^-; w_i^+ \leftarrow w_i^+ + \beta^{-x_i}w_i^+\)

Balanced Winnow is equivalent to keeping a scaled sum of updates as perceptron does but with a scale factor of \(\eta = \log \left(\frac{1}{\beta}\right)\)

Balanced Winnow \((W, Z, (X, y))\)

\[0 < \beta < 1\]

If \(\text{sign}(W \cdot X) \neq y\)

\[Z \leftarrow Z + y \left(\log \frac{1}{\beta}\right) X\]

\[W \leftarrow 2 \sinh(Z)\] (That is, \(\forall i\) \(w_i \leftarrow 2 \sinh z_i = e^{z_i} - e^{-z_i}\))
Generalized Perceptron algorithms

Generalized Perceptron \((W, Z, f, (X, y))\)

If \(\text{sign}(W \cdot X) \neq y\)

\[ Z \leftarrow Z + \eta y X \]

\[ W \leftarrow f(Z) \] (That is, \(\forall i \quad w_i \leftarrow f(z_i)\))

See Grove, Littlestone, Schuurmans

Quasi additive update algorithms for function approximation

Generalized Gradient Descent \((W, Z, f, (X, y))\)

\[ y \leftarrow W \cdot X \]

\[ Z \leftarrow Z + \eta y X \]

\[ W \leftarrow f(Z) \] (That is, \(\forall i \quad w_i \leftarrow f(z_i)\))

By choosing the learning rate \(\eta\) and \(f\) appropriately, we can obtain gradient-based learning algorithms that work well in the presence of irrelevant attributes (See Kivinen and Warmuth)
Applications of Multiplicative Update Algorithms

Multiplicative update algorithms constitute an example of theoretical analysis of simple algorithms leading to a new powerful family of algorithms that are useful in practice

• Spelling correction
• Text processing (SPAM filters)
• Face recognition
• Portfolio selection
• Learning in game-theoretic settings

Probably Approximately Correct (PAC) Learning

• Distribution-free models of learning
• Probably Approximately Correct (PAC) Learning
• Sample Complexity Analysis of Concept Classes
• Efficient PAC Learners – polynomial sample learning, polynomial time learning
• Vapnik-Chervonenkis (VC) dimension and Sample Complexity
• Occam’s razor
• Learning under simple distributions
• Brief tour of other key results
The Learning Game

- We assume
  - An instance space \( \mathcal{X} \)
  - A concept space \( C = \{ c : \mathcal{X} \rightarrow \{0,1\} \} \)

- A hypothesis space \( H = \{ h : \mathcal{X} \rightarrow \{0,1\} \} \)

- An unknown, arbitrary, not necessarily computable, stationary probability distribution \( D \) over the instance space \( \mathcal{X} \)
Rules of the Game

• An adversary selects a distribution $D$ over a given instance space $X$ and a target concept $c$ from a given concept class $C$
• An oracle samples the instance space according to $D$ and provides a set $S$ of labeled examples of an unknown concept $c$ to the learner
• The learner's task is to output a hypothesis $h$ from $H$ that closely approximates the unknown concept $c$ based on the examples it has encountered
• The learner is tested on samples drawn from the instance space according to the same probability distribution $D$

Measuring the error of a hypothesis

• The error of a hypothesis $h$ with respect to a concept $c$ and distribution $D$

$$\text{error}_{c,D}(h) = \Pr_{x \in D}(c(x) \neq h(x))$$
Probably Approximately Correct Learning – Why?

Impossibility of learning with 0% error

• Because instances are sampled according to an unknown, arbitrary probability distribution $D$ over the instance space, there is no way to be certain that the learner will see all the necessary examples to exactly learn an unknown concept – exact learning is impossible!

Impossibility of approximate learning with 100% confidence

• Approximate learning (with a specified error $\varepsilon$) cannot be guaranteed hundred percent of the time because of the vagaries of the sampling process

$\varepsilon$-approximation of a concept $c$

• We say that a hypothesis $h$ is an $\varepsilon$-approximation of a concept $c$, with respect to an instance distribution $D$ if and only if the probability that $h$ and $c$ disagree on an instance from the instance space drawn at random according to the distribution $D$ is less than $\varepsilon$. That is,

$$error_{c,D}(h) < \varepsilon$$
PAC Learning – A preliminary definition

• A concept class $C$ is said to be PAC-learnable using a hypothesis class $H$ if there exists a learning algorithm $L$ such that for all concepts $c \in C$, for all distributions $D$ on an instance space $\mathcal{X}$, $\forall \varepsilon, \delta \ (0 < \varepsilon, \delta < 1)$, $L$, when allowed access to the Example oracle (that is, a finite set $S$ of labeled examples of a target concept $c$), outputs with probability at least $(1 - \delta)$, a hypothesis $h \in H$ which is an $\varepsilon$-approximation of $c$. That is,

$$\forall D \text{ over } \mathcal{X}, \forall c \in C, \forall \varepsilon, \delta : 0 < \varepsilon < 1, 0 < \delta < 1, \Pr_{S \in D} (error_{c, D}(h) < \varepsilon) \geq (1 - \delta)$$

Such a learning algorithm $L$ is called a PAC learning algorithm for the concept class $C$.

Notes on the definition of PAC Learnability

• The definition of PAC learnability of a specified concept class $C$ requires that there be a learning algorithm $L$ that produces an $\varepsilon$-approximation of any concept in the concept class $C$, under any instance distribution, and any choice of the error ($\varepsilon$) and confidence ($\delta$) parameters.

• Specifying a learning algorithm requires
  – the choice of an instance representation
  – the choice of a hypothesis (concept) representation and
  – an algorithm for determining the membership of an instance in a hypothesis (concept).

• More on this later
How can we show that a concept class is PAC Learnable?

- In order to prove the PAC learnability of a concept class we have to demonstrate the existence of a learning algorithm which meets the necessary criteria specified in the definition of PAC learnability.
- It is even better if we can offer a constructive proof – that is, provide an algorithm that meets the PAC criteria.
- It turns out that we can often get away with using a rather dumb learning algorithm – one that simply outputs a hypothesis that is consistent with the training examples. (We assume that \( H \) is expressive enough to guarantee the existence of a consistent hypothesis).

PAC Learnability of Finite Concept Classes

- **Definition:** A **consistent learner** is one that returns some hypothesis \( h \in H \) that is consistent with a training set \( S \) of cardinality \( m \).
- **Theorem:** A consistent learner \( L \) is a PAC learner. That is, given a sufficiently large number \( m \) of examples of \( c \), the hypothesis produced by \( L \) is guaranteed, with probability at least \( 1-\delta \), to be an \( \varepsilon \)-approximation of \( c \) – for any choice of \( c \in C \), any instance distribution \( D \), and any choice of \( \varepsilon, \delta \) such that \( 0<\varepsilon, \delta<1 \). Specifically, it suffices if

\[
  m > \frac{1}{\varepsilon} \ln \left( \frac{|H|}{\delta} \right)
\]
A consistent learner

\[ V_{H,S} = \{ h \in H \mid h \text{ is consistent with examples in } S \} \]

Proof that a consistent learner is a PAC learner

Proof sketch

- There are two kinds of hypothesis in \( H \), and hence in the version space \( V_{H,S} \):
  - good (\( \varepsilon \)-approximations of the target concept)
  - bad (not \( \varepsilon \)-approximations of the target concept).
- Given a sufficiently large number of examples of a target concept \( c \), a sufficiently large fraction of the bad hypotheses get eliminated from the version space maintained by a consistent learner.
- Consequently, a randomly selected hypothesis from \( V_{H,S} \) has a high probability (at least \( 1-\delta \)) of being an \( \varepsilon \)-approximation of the target concept.
A consistent learner is a PAC learner

**Definition:** A version space $V_{H,S}$ is said to be $\epsilon$-exhausted with respect to an instance distribution $D$ and a concept $c$ if every hypothesis $h \in V_{H,S}$ is an $\epsilon$-approximation of $c$. That is,

$$\forall h \in V_{H,S} \text{ error}_{c,D}(h) < \epsilon$$

Our goal is to make the training set $S$ large enough to ensure that the probability that the version space is not $\epsilon$-exhausted with respect to $c$ and $D$ is sufficiently small (less than $\delta$) regardless of the choice of $c \in C$ and instance distribution $D$ by an adversary.

---

**Theorem:** Suppose $H$ is a finite hypothesis space, and $S$ a set of $m$ ($m \geq 1$) examples of some $c \in C$. Then for any $\epsilon$ ($0 < \epsilon \leq 1$), the probability that the version space $V_{H,S}$ is not $\epsilon$-exhausted with respect to an instance distribution $D$ and a concept $c$ is at most

$$|H|e^{-\epsilon m}$$

**Proof:**

- Let $H_{Bad}$ be the subset of hypothesis in $V_{H,S}$ that are not $\epsilon$-approximations of $c$.

$$\forall h \in H_{Bad}, \text{ error}_{c,D}(h) \geq \epsilon$$
A consistent learner is a PAC learner

• The probability that a hypothesis $h \in H_{\text{Bad}}$ agrees with $c$ on a random instance drawn according to $D$ is at most $(1 - \varepsilon)$

• The probability that a hypothesis $h \in H_{\text{Bad}}$ is consistent $m$ independently drawn random examples is at most $(1 - \varepsilon)^m$

• The probability that some hypothesis in $V_{H,S}$ survives $m$ independently drawn random examples is at most

$$|H_{\text{Bad}}|(1-\varepsilon)^m \leq |H|(1-\varepsilon)^m \text{ since } H_{\text{Bad}} \subseteq H$$

• PAC learning requires that the probability of $L$ returning a bad hypothesis is small. That is, $|H|(1-\varepsilon)^m < \delta$

PAC learning requires that the probability of $L$ returning a bad hypothesis is small (at most $\delta$). That is,

$$|H|(1-\varepsilon)^m \leq \delta$$

$(0 < \varepsilon \leq 1) \Rightarrow \{ (1 - \varepsilon) \leq e^{-\varepsilon} \}$

Hence, to ensure that a consistent learner is a PAC learner, it suffices that

$$m \geq \left( \frac{1}{\varepsilon} \left( \ln |H| + \ln \left( \frac{1}{\delta} \right) \right) \right)$$
Sample complexity of PAC Learning for finite hypothesis classes

The smallest integer $m$ that satisfies the inequality

$$m > \frac{1}{\varepsilon} \ln \frac{|H|}{\delta}$$

is called the sample complexity of $H$.

---

PAC- Easy and PAC-Hard Concept Classes for Consistent Learners

- Conjunctive concepts are easy to learn
- Use the same algorithm as the one used in the mistake bound setting
- Sample complexity
  $$O\left(\frac{1}{\varepsilon} \left( N \ln 3 + \ln \frac{1}{\delta} \right) \right)$$
- Time complexity is polynomial in the relevant parameters of interest
- The class of all Boolean concepts is hard to learn (Why?)
- Remark: Polynomial sample complexity is necessary but not sufficient for efficient (polynomial time) PAC learning – producing a consistent hypothesis may be NP-Hard
Representation

Distinction between a concept and its representation
• A concept is simply a set of instances – **extensional** definition
• A representation of a concept is a symbolic encoding of that set – **intensional** definition

Example
• A concept can be represented as a
  • Boolean formula $\phi$, or a Boolean formula $\varphi$ that is logically equivalent to $\phi$, or
  • a truth table

Different representations of the same concept may differ radically in **size**

Example
• Boolean parity function

\[ f(x_1, x_2, ..., x_n) = x_1 \oplus x_2 \oplus ... \oplus x_n \]

where $\oplus$ denotes the exclusive OR can be computed by a circuit of $\land$, $\lor$, and $\neg$ gates whose size is bounded by a fixed polynomial in $n$ but a DNF (disjunction of conjunctions) representation of the same function has size that is exponential in $n$. 
Representation

• A given target concept has many representations
• The learner is oblivious to which, if any, representation is being used by the teacher or adversary to encode the target concept
• Yet it matters a great deal which of the many representations of hypotheses that the learner chooses – the size of the representation of a hypothesis \( h \) is a lower bound on the running time of an algorithm that outputs \( h \).

Representation

• A representation scheme for a concept class \( \mathbf{C} \) is a function \( R : \Sigma^* \rightarrow \mathbf{C} \) where \( \Sigma \) is a finite alphabet of symbols.
• Any string \( \sigma \in \Sigma^* \) such that \( R(\sigma) = c \) is called a representation of \( c \) under \( R \).
• There may be many representations for a concept \( c \) under representation \( R \).
• When we need to use real numbers to represent concepts, we may allow \( R : (\Sigma \cup \mathbb{R})^* \rightarrow \mathbf{C} \).
Representation size

\[ R : \Sigma^* \rightarrow C \]

\[ \text{size} : \Sigma^* \rightarrow \mathbb{N} \]

assigns a natural number \( \text{size}(\sigma) \) to each representation \( \sigma \)

The results obtained under a particular definition of \( \text{size} \) are meaningful only if the definition is \textit{natural}.

Example

\[ \Sigma = \{0,1\} \]

\( \text{size}(\sigma) \) is the length of \( \sigma \) in bits

If real numbers are used to encode a concept, we may charge one unit of size to each real number – cannot translate this measure of size into size in bits unless the real numbers are finite precision

---

Size of a concept \( c \) under a representation \( R \)

\[ \text{size}(c) = \min_{R(\sigma) = c} \{ \text{size}(\sigma) \} \]

Size of a concept \( c \in C \) under a representation scheme \( R \) for \( C \) is the size of the smallest representation of \( c \) under \( R \)

The larger the value of \( \text{size}(c) \), the more complex the concept \( c \) under the chosen representation

From now on, when we speak of learning a concept class \( C \), we will mean learning \( C \) under a chosen representation \( R \)
Size of instances

- In a Boolean instance space $X_n = \{0,1\}^n$ the size of each instance is $n$
- In $X_n = \mathbb{R}^n$ the size of each instance may be taken to be $n$ (with the usual caveat).
- In $X_n = A^*n$ where $A$ is a finite alphabet, the size of an instance is the length of the corresponding string (with maximum size being $n$).

Efficient (Polynomial Time) PAC Learning

**Definition:** Let $C_n$ be a concept class (actually a representation class) over $X_n$.

- Let $X = \bigcup_{n \geq 1} X_n$ and $C = \bigcup_{n \geq 1} C_n$.
- $C$ is said to be efficiently PAC-learnable if $C$ is PAC-learnable using a learning algorithm $L$ which runs in time that is polynomial in $n$ (size of the instance representation), $\text{size}(c)$ (size of the representation of the target concept $c$), $\left(\frac{1}{\delta}\right)$ and $\left(\frac{1}{\varepsilon}\right)$.

- We assume that the learner is given $n$ and $\text{size}(c)$ as input – however, these assumptions can be relaxed.
Efficient (Polynomial Time) PAC Learning

• **Necessary:** Sample complexity must be polynomial in the relevant parameters
• **Sufficient:** Polynomial sample complexity and a polynomial time consistent learner
  • More examples allowed to achieve lower error
  • More examples allowed for achieving higher confidence
  • More examples allowed for learning more complex concepts
  • More examples allowed for learning from bigger instances

Conjunctive Concepts are Efficiently PAC Learnable

• Conjunctive concepts are efficiently PAC-learnable under a natural representation of conjunctions

  • Sample complexity \( O\left(\frac{1}{\varepsilon} \left( N \ln 3 + \ln \left( \frac{1}{\delta} \right) \right) \right) \)

  • Time complexity \( O\left(\frac{1}{\varepsilon} \left( N \ln 3 + \ln \left( \frac{1}{\delta} \right) \right) \right) \)
3-Term DNF concepts are not efficiently PAC learnable unless P=RP

- **Theorem:** 3-term DNF concept class (disjunctions of at most 3 conjunctions) are not efficiently PAC-learnable using the same hypothesis class (although it has polynomial sample complexity) unless P=RP.

- **Proof:** By polynomial time reduction of graph 3-colorability (a well-known NP-complete problem) to the problem of deciding whether a given set of labeled examples is consistent with some 3-term DNF formula.

---

Transforming Hard Problems to Easy ones

- **Theorem:** 3-term DNF concepts are efficiently PAC-learnable using 3-CNF (conjunction of disjunctions (clauses) with at most 3 literals per clause) hypothesis class.

- **Proof:** 3-term DNF $\subseteq$ 3-CNF

- Transform each example over $N$ boolean variables into a corresponding example over $N^3$ variables (one for each possible clause in a 3-CNF formula).

$$T_1 \lor T_2 \lor T_3 = \bigwedge_{u \in T_1, v \in T_2, w \in T_3} (u \lor v \lor w)$$

- The problem reduces to learning a conjunctive concept over the transformed instance space.
Transforming Hard Problems to Easy ones

- **Theorem**: For any $k \geq 2$ $k$-term DNF are efficiently PAC-learnable using the $k$-CNF hypothesis class.

- **Remark**: In this case, enlarging the search space by using a hypothesis class that is larger than strictly necessary, actually makes the problem easy!

- **Remark**: No, we have not proved that $P=NP$.

- **Summary**:

  \[
  \text{Conjunctive} \subseteq \text{k-term DNF} \subseteq \text{k-CN} \subseteq \text{CNF}
  \]

  Easy   Hard   Easy   Hard

Occam Learning Algorithm

- **Definition**: Let $\alpha \geq 0$ & $0 \leq \beta < 1$ be constants. A learning algorithm $L$ is said to be an $\alpha - \beta$ Occam algorithm for a concept class $C$ using a hypothesis class $H$ if $L$, given a set $S$ of $m$ random examples of an unknown concept $c \in C$ outputs a hypothesis $h \in H$ such that $h$ is consistent with $S$ and

\[
\text{size}(h) \leq \left(\text{Nsize}(c)^\alpha m^\beta\right)
\]

Effective hypothesis space size

\[
H_{nm} \leq 2^{(\text{Nsize}(c)^\alpha m^\beta)}
\]
Occam Learning Algorithm outputs succinct hypothesis

\[ \text{size} (h) \leq \left\{ \text{Nsize} (c) \right\}^\alpha m^\beta \]

When \( m >> N \), \( \text{size} (h) = O\left( (\text{size} (c))^\alpha m^\beta \right) \)

- \( m \) labels have to be compressed into \( O(m)\beta \) bits
- A mild requirement because we can always obtain a consistent hypothesis that is \( O(mn) \) bits long (why?)
- We have to allow \( \text{size}(h) \) to depend linearly on \( \text{size}(c) \) in the event the shortest hypothesis in \( H \) may in fact be the target concept \( c \).
- We allow a generous dependence on \( m \) – which often makes it easier to find a consistent hypothesis – finding the shortest hypothesis is often computationally intractable

Occam Learning Algorithm

- An Occam learning algorithm \( L \) for a concept class \( C \) is said to be an efficient \( \alpha - \beta \) Occam learning algorithm for \( C \) if its running time is bounded by a polynomial in \( n, m, \) and \( \text{size}(c) \).
- The simple algorithm we considered for learning conjunctive concepts is an efficient Occam learning algorithm (Prove this!).
Sample complexity of an Occam Algorithm

- Theorem: An Occam algorithm is guaranteed to be PAC if the number of samples

\[
m = O\left(\frac{1}{\epsilon} \lg \frac{1}{\delta} + \left[\frac{\text{Nsize}(c)^a}{\epsilon}\right]^{\frac{1}{1-\beta}}\right)
\]

- Proof: Left as an exercise.

---

k-decision lists

- k decision list over Boolean variables \(x_1...x_N\) is an ordered sequence

\[
l = ((c_1, b_1)...(c_l, b_l), b)
\]

Where each \(c_i\) is a conjunction of at most k literals chosen from \(x_1...x_N\) (and their negations) and each \(b_i\) and \(b\) is 0 or 1.

On a given \(N\)-bit input, \(l\) is evaluated like a nested if-then-else statement with \(b\) corresponding to the default output.
Occam algorithm is PAC for K-decision lists

- **Theorem:** For any fixed $k$, the concept class of $k$-decision lists is efficiently PAC-learnable using the same hypothesis class.

- **Algorithm** – Greedily find conjunctions of at most $k$ literals that cover the largest subset of examples with the same class label.

- **Remark:** $k$-decision lists constitute the most expressive Boolean concept class over the Boolean instance space $\{0,1\}^N$ that are known to be efficiently PAC learnable.

PAC Learnability of Infinite Concept Classes

- How many random examples does a learner need to draw before it has sufficient information to learn an unknown target concept chosen from a concept class $C$?

- Sample complexity results derived previously answer this question for the case of finite concept classes.

- Are there any non-trivial infinite concept classes that are PAC learnable from a finite set of examples?

- Can we quantify the complexity of an infinite concept class? – yes, using Vapnik-Chervonenkis Dimension!
Vapnik-Chervonenkis (VC) Dimension

- Let $C$ be a concept class over an instance space $X$.
- Both $C$ and $X$ may be infinite.
- We need a way to describe the behavior of $C$ on a finite set of points $S \subseteq X$. 
  \[ S = \{X_1, X_2, \ldots, X_m\} \]
- For any concept class $C$ over $X$, and any $S \subseteq X$,
  \[ \Pi_C(S) = \{c \cap S : c \in C\} \]
- Equivalently, with a little abuse of notation, we can write
  \[ \Pi_C(S) = \{(c(X_1), \ldots, c(X_m)) : c \in C\} \]

$\Pi_C(S)$ is the set of all dichotomies or behaviors on $S$ that are induced or realized by $C$.

If $|\Pi_C(S)| = (0,1)^m$, where $|S| = m$, or equivalently, $|\Pi_C(S)| = 2^m$, we say that $S$ is shattered by $C$.

A set $S$ of instances is said to be *shattered* by a hypothesis class $H$ if and only if for every dichotomy of $S$, there exists a hypothesis in $H$ that is consistent with the dichotomy.
VC Dimension of a hypothesis class

- **Definition:** The VC-dimension $V(H)$, of a hypothesis class $H$ defined over an instance space $X$ is the cardinality $d$ of the largest subset of $X$ that is shattered by $H$. If arbitrarily large finite subsets of $X$ can be shattered by $H$, $V(H) = \infty$.

How can we show that $V(H)$ is at least $d$?
- Find a set of cardinality at least $d$ that is shattered by $H$.

How can we show that $V(H) = d$?
- Show that $V(H)$ is at least $d$ and no set of cardinality $(d+1)$ can be shattered by $H$.

VC Dimension of a Hypothesis Class - Examples

- Example: Let the instance space $X$ be the 2-dimensional Euclidean space. Let the hypothesis space $H$ be the set of linear 1-dimensional hyperplanes in the 2-dimensional Euclidean space.

  \[ \text{\includegraphics[width=0.5\textwidth]{example.png}} \]

- Then $V(H) = 3$ (a set of 3 points can be shattered by a hyperplane as long as they are not co-linear but a set of 4 points cannot be shattered). For the concept class of linear hyperplanes, VC dimension is $n+1$.
VC Dimension and Sample complexity

A concept class \( C \subseteq 2^X \) is trivial if it contains a single concept or 2 disjoint concepts which partition \( X \).

**Theorem:** Let \( C \) be a non trivial concept class. Then \( C \) is PAC learnable if and only if \( V(C) \) is finite.

If \( V(C) = d \) and \( d < \infty \), then the bounds on sample complexity of \( C \) are given by

\[
m = O\left( \frac{1}{\varepsilon} \log \frac{1}{\delta} + \frac{d}{\varepsilon} \log \frac{1}{\varepsilon} \right)
\]

\[
m = \Omega\left( \frac{d}{\varepsilon} \right)
\]

**Proof:** See Readings

---

Some Useful Properties of VC Dimension

\[ (C_1 \subseteq C_2) \Rightarrow V(C_1) \leq V(C_2) \]

If \( C \) is a finite concept class, \( V(C) \leq \log |C| \)

\[ (C = \{X - c : c \in C\}) \Rightarrow V(C) = V(C) \]

\[ (C = C_1 \cup C_2) \Rightarrow V(C) \leq V(C_1) + V(C_2) + 1 \]

If \( C_j \) is formed by a union or intersection of \( l \) concepts from \( C \), \( V(C_j) = O(V(C)l \log l) \)

If \( V(C) = d \), \( \Pi_c(m) = \max \{\Pi_c(S) : |S| = m\} \)

\[ \Pi_c(m) \leq \Phi_d(m) \text{ where} \]

\[ \Phi_d(m) = 2^m \text{ if } m \geq d \text{ and } \Phi_d(m) = O(m^d) \text{ if } m < d \]

**Proof:** Left as an exercise
Sample complexity of a multilayer perceptron

- Acyclic, layered multi-layer networks of \( s \) threshold logic units, each with \( r \) inputs, has VC dimension

\[
d = O(r + 1)s \log(s)
\]

Hence, we have:

\[
m = O\left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right) + \frac{d}{\epsilon} \log\left(\frac{1}{\delta}\right)\right)
\]

\[
= O\left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right) + \left(\frac{r+1}{\epsilon}\right) s \log\left(\frac{s}{\epsilon}\right)\right)
\]

Learning when the size of the target concept is unknown

- Results on efficient PAC learnability of concept classes are derived under the assumption that the size of the target concept is one of the inputs to the learning algorithm
- Can we guarantee efficient PAC learnability when the size of the target concept is unknown? – Yes, using the doubling trick and hypothesis testing
Hoffding Bounds

- Let $X_1, \ldots, X_m$ be outcomes of independent Bernoulli trials each with probability of success $p$. Let

$$S = \sum_{i=1}^{m} X_i, \quad \text{So} \quad E(S) = pm$$

$$\Pr(S \geq pm + t) \leq e^{-2mt^2}$$
$$\Pr(S \geq \alpha m) \leq e^{-2m(\alpha - p)^2} \quad \text{where} \quad \alpha \geq p$$
$$\Pr(S \leq \alpha m) \leq e^{-2m(\alpha - p)^2} \quad \text{where} \quad \alpha \leq p$$

Chernoff Bounds

Let $X_1, \ldots, X_m$ be independent outcomes of independent Bernoulli trials each with probability of success $p$. Let

$$S = \sum_{i=1}^{m} X_i$$

and

$$LE(p, m, r) = \Pr(S \leq r)$$
$$GE(p, m, r) = \Pr(S \geq r)$$

$$LE(p, m, (1 - \alpha)pm) \leq e^{-\alpha^2 mp^1/2} \quad 0 \leq \alpha \leq 1$$
$$GE(p, m, (1 + \alpha)pm) \leq e^{-\alpha^2 mp^3/3}$$

Chernoff Bounds are tighter than Hoffding Bounds when $p < 1/4$
How to determine if a hypothesis is $\varepsilon$-good

- We cannot distinguish with certainty between an $\varepsilon$-good hypothesis and one that has error slightly greater than $\varepsilon$ by testing the hypotheses on a finite set of examples.
- However, we can distinguish between an $(\varepsilon/2)$-good hypothesis and an $\varepsilon$-bad hypothesis with high confidence.

Algorithm $\text{Test}(h,n,\varepsilon,\delta)$

1. Make $m = \left\lceil \frac{32}{\varepsilon^2} \left( n \ln 2 + \ln \left( \frac{2}{\delta} \right) \right) \right\rceil$ calls to $\text{Example}(c,D)$ ($n$ is the size of the instances).
2. Accept $h$ if it misclassifies at most $\left( \frac{3\varepsilon}{4} \right) m$ examples; Otherwise, reject $h$.

$\text{Test}(h,n,\varepsilon,\delta)$ has the property:

- If $\text{error}_{c,D}(h) \geq \varepsilon$, then $\Pr(h \text{ is accepted}) \leq \frac{\delta}{2^{n+1}}$.
- If $\text{error}_{c,D}(h) \leq \frac{\varepsilon}{2}$, then $\Pr(h \text{ is rejected}) \leq \frac{\delta}{2^{n+1}}$. 

Learning when the size of the target concept is unknown

**Algorithm** $B(n, c, \delta)$

1. $i \leftarrow 0$
2. UNTIL $h$ is accepted by $\text{Test}(h, n, c, \delta)$ DO
3. $i \leftarrow i + 1$
4. $3 \leftarrow \left \lfloor 2^{(i-1)/\ln(3)} \right \rfloor$
5. $h_i \leftarrow \text{hypothesis output by } A(n, 3, c/2, 1/2)$
6. Output $h = h_i$

$A$ – requires the target concept size as a parameter

$B$ – works for an unknown target concept size

Learning in the presence of noise

- Types of noise
- Random misclassification noise
- Random attribute noise – uniform, non-uniform
- Malicious noise – examples selected and corrupted by an omnipotent adversary who may have access to the internal state of the learner
- .......

Copyright Vasant Honavar, 2006.
Learning in the presence of random misclassification noise

- Random misclassification noise – with probability $\eta$ the instance is correctly labeled. With probability $(1 - \eta)$, the label is flipped.

Assume WLOG that $0 \leq \eta \leq \eta_0 < 1/2$

Draw

$$m \geq \frac{2}{\varepsilon^2 (1 - 2\eta_0)^2} \ln\left(\frac{2|C|}{\delta}\right)$$

examples from Example$_\eta$

Output a hypothesis $h \in C$ that minimizes the training error.

The method can be adapted to the case of unknown $\eta_0$.

Alternative methods are available for specific concept classes.
PAC learning using weak learners

Weak learner
• Confidence lower than $(1-\delta)$
  \(\rightarrow\) Boost confidence
• Error greater than \(\varepsilon\)
  \(\rightarrow\) Boost accuracy
• Error greater than \(\varepsilon\) and Confidence lower than $(1-\delta)$
  \(\rightarrow\) Boost accuracy and confidence

We can turn weak learners into strong (PAC) learners using accuracy and confidence boosting algorithms

Confidence Boosting

• Run the algorithm several times on independently drawn training sets to obtain a set of hypotheses – The number of independent runs is chosen to be large enough to ensure that the probability that at least one of the resulting hypothesis has error less than \(\varepsilon\) is at least $(1-\delta^2)$
• Use hypothesis testing to select the best hypothesis in the pool with high confidence – alternatively use weighted majority classification
Accuracy Boosting

• Learn a sequence of hypotheses
• The first hypothesis is based on the original training set
• Each subsequent hypothesis is based on a sampling of the training set according to a distribution which assigns higher probability to training examples that were misclassified by the previously learned hypotheses and perhaps a different error parameter
• Classification is based on majority or weighted majority of the hypotheses
• More on Accuracy Boosting Later..

Learning under helpful distributions

• PAC Learning requires success under all possible probability distributions
• Some concept classes are hard to learn under all distributions – e.g., regular languages or deterministic finite state automata (DFA), yet they are readily learned by humans
• Question – can natural settings be modeled by more benign or helpful distributions? E.g., can DFA be learned under helpful distributions?
• What precisely are helpful distributions?
Digression – Kolmogorov Complexity

- **Kolmogorov complexity** $K(x)$ is a machine independent i.e. universal measure of the complexity of description of an object
- $K(x) = \text{the number of bits in the shortest universal Turing machine program for } x$
- **Example 1**
  - Object – 01010101010101... 0101010101 = (01)$^{500}$
  - Program – Print “01” 500 times
- **Example 2**
  - Object – 1100110111… 100101110111 (random string)
  - Program – Print “11001101 … 01110111”
- Simple objects have low Kolmogorov complexity

Universal distribution

We fix a universal Turing machine $U$

\[
K(\alpha) = \min_{\pi} \{ \text{length } (\pi) \mid U(\pi) = \alpha \}
\]

\[
K(\alpha \mid \beta) = \min_{\pi} \{ \text{length } (\pi) \mid U(\pi, \beta) = \alpha \}
\]

$K(\alpha \mid \beta) \leq K(\alpha)$

Universal distribution $M$ assigns higher probabilities to simpler objects

\[
M(x) \propto 2^{-K(x)} \quad M(x \mid \alpha) \propto 2^{-K(x \mid \alpha)}
\]
Learning Under Universal distribution

- Universal distribution $M$ multiplicatively dominates all enumerable distributions
- Enumerable distributions include finite precision Poisson, Gaussian, and many other distributions

**Theorem:** A concept class is Probably approximately learnable under each enumerable distribution iff it is Probably approximately learnable under the universal distribution assuming during learning examples are drawn according to $M(x)$

Learning under universal distribution

- Li and Vitanyi (1991) showed that log $n$-term DNF are learnable under $M(x | c)$ where $c$ is the target concept
- Parekh and Honavar (1999, 2001) showed that
  - Simple DFA (with encoding of size $O(\log N)$ where $N$ is the number of states) are efficiently learnable under the universal distribution $M(x)$
  - DFA are efficiently learnable with a helpful teacher – examples are drawn according to $M(x | c)$ where $c$ is the target concept
  - Denis (2001) showed that DFA are efficiently learnable from positive examples alone under $M(x | c)$
Additional Possibilities

- PAC learning model assumes that target concepts are selected uniformly at random from $C$
- **Benign teacher** – How about if target concepts are selected according to universal distribution over the concept class, namely $M(c)$?
- **Occam Learner** – Impose a preference bias over the set of consistent hypotheses – Select hypothesis $h$ according to $M(h)$
- **Bayesian learner** – Assume priors given by $M(h)$

Summary of Main Results in Distribution-Independent Learning Theory

- PAC-Easy learning problems lend themselves to a variety of efficient algorithms.
- PAC-Hard learning problems can often be made PAC-easy through appropriate instance transformation and choice of hypothesis space
- Occam's razor often helps
- Weak learning algorithms can be turned into strong PAC learners through accuracy and confidence boosting
- Learning under restricted classes of instance distributions (e.g., universal distribution) and priors offers new possibilities