Generic Mistake Bound Learning

Machine Learning
Spring 2018
Generic Mistake Bound Learning

• How good is a learning algorithm?

• Online learning
Where are we?

• How good is a learning algorithm?
  – Different answers based on different learning protocols

• Online learning
Quantifying Performance

• How can we rigorously quantify the performance of our learning algorithm?

• **One approach**: Compute how many examples should the learning algorithm see before we can say that our learned hypothesis is *good* (or *good enough*)

  This number will depend on the learning protocol
Example: Learning Conjunctions

There is a hidden (monotone) conjunction for the learner (you) to learn

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

How many examples are needed to learn it? How does learning proceed?

There are 100 Boolean variables. But you don’t know that only these five are relevant.
How good is our learning algorithm?

Example: Learning Conjunctions

There is a hidden (monotone) conjunction for the learner (you) to learn

\[ f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100} \]

How many examples are needed to learn it? How does learning proceed?

– **Protocol I**: The learner proposes instances as queries to the teacher

There are 100 Boolean variables. But you don’t know that only these five are relevant
Example: Learning Conjunctions

There is a hidden (monotone) conjunction for the learner (you) to learn

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

How many examples are needed to learn it? How does learning proceed?

- **Protocol I**: The learner proposes instances as queries to the teacher
- **Protocol II**: The teacher (who knows \( f \)) provides training examples

There are 100 Boolean variables. But you don’t know that only these five are relevant.
How good is our learning algorithm?

Example: Learning Conjunctions

There is a hidden (monotone) conjunction for the learner (you) to learn

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

How many examples are needed to learn it? How does learning proceed?

- **Protocol I**: The learner proposes instances as queries to the teacher
- **Protocol II**: The teacher (who knows f) provides training examples
- **Protocol III**: Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels \(f(x)\)
Example: Learning Conjunctions

There is a hidden (monotone) conjunction for the learner (you) to learn

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

How many examples are needed to learn it? How does learning proceed?

- **Protocol I**: The learner proposes instances as queries to the teacher
- **Protocol II**: The teacher (who knows f) provides training examples
- **Protocol III**: Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels \( f(x) \)

Let us compare these three protocols.
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]
Learning Conjunctions

**Protocol I:** The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

- Is $x_{100}$ in? $(1,1,1...,1,0)$, ? $f(x)=0$ (conclusion: Yes, $x_{100}$ is in $f$)

How good is our learning algorithm?
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

– Is $x_{100}$ in? $<(1,1,1...1,0), ?>$ $f(x)=0$ (conclusion: Yes, $x_{100}$ is in $f$)
– Is $x_{99}$ in? $<(1,1,...1,0,1), ?>$ $f(x)=1$ (conclusion: No, $x_{99}$ is not in $f$)
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

- Is $x_{100}$ in? $<(1,1,1...,1,0), ?> \quad f(x)=0$ (conclusion: Yes, $x_{100}$ is in $f$)
- Is $x_{99}$ in? $<(1,1,...1,0,1), ?> \quad f(x)=1$ (conclusion: No, $x_{99}$ is not in $f$)
- ...

$\ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}$

How good is our learning algorithm?
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

- Is \( x_{100} \) in? \( \langle 1,1,1...,1,0 \rangle, \?) \( f(x)=0 \) (conclusion: Yes, \( x_{100} \) is in \( f \))
- Is \( x_{99} \) in? \( \langle 1,1,...1,0,1 \rangle, \?) \( f(x)=1 \) (conclusion: No, \( x_{99} \) is not in \( f \))
- ...
- Is \( x_{2} \) in? \( \langle 1,0,...1,1,1 \rangle, \?) \( f(x)=0 \) (conclusion: Yes, \( x_{2} \) is in \( f \))
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

- Is $x_{100}$ in? $\langle 1,1,1...,1,0 \rangle$, $\triangleright$ $f(x)=0$ (conclusion: Yes, $x_{100}$ is in $f$)
- Is $x_{99}$ in? $\langle 1,1,...1,0,1 \rangle$, $\triangleright$ $f(x)=1$ (conclusion: No, $x_{99}$ is not in $f$)
- ...
- Is $x_{2}$ in? $\langle 1,0,...1,1,1 \rangle$, $\triangleright$ $f(x)=0$ (conclusion: Yes, $x_{2}$ is in $f$)
- Is $x_{1}$ in? $\langle 0,1,...1,1,1 \rangle$, $\triangleright$ $f(x)=1$ (conclusion: No, $x_{1}$ is not in $f$)
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a monotone conjunction:

- Is \( x_{100} \) in? \(<(1,1,1\ldots,1,0), \geq \) f(x)=0 (conclusion: Yes, \( x_{100} \) is in f)
- Is \( x_{99} \) in? \(<(1,1,\ldots,1,0,1), \geq \) f(x)=1 (conclusion: No, \( x_{99} \) is not in f)
- ...
- Is \( x_2 \) in? \(<(1,0,\ldots,1,1,1), \geq \) f(x)=0 (conclusion: Yes, \( x_2 \) is in f)
- Is \( x_1 \) in? \(<(0,1,\ldots,1,1,1), \geq \) f(x)=1 (conclusion: No, \( x_1 \) is not in f)

A straightforward algorithm requires \( n=100 \) queries, and will produce the hidden conjunction (exactly)

\[
h = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}
\]
Learning Conjunctions

Protocol I: The learner proposes instances as queries to the teacher

Since we know we are after a **monotone conjunction**:

- Is \( x_{100} \) in? \(<(1,1,1...1,0), ?>\) f(x)=0 (conclusion: Yes, \( x_{100} \) is in f)
- Is \( x_{99} \) in? \(<(1,1,...1,0,1), ?>\) f(x)=1 (conclusion: No, \( x_{99} \) is not in f)
- ...
- Is \( x_{2} \) in? \(<(1,0,...1,1,1), ?>\) f(x)=0 (conclusion: Yes, \( x_{2} \) is in f)
- Is \( x_{1} \) in? \(<(0,1,...1,1,1), ?>\) f(x)=1 (conclusion: No, \( x_{1} \) is not in f)

A straightforward algorithm requires \( n=100 \) queries, and will produce the hidden conjunction (exactly)

\[
    h = x_{2} \land x_{3} \land x_{4} \land x_{5} \land x_{100}
\]

What happens here if the conjunction is not known to be monotone? If we know of a positive example, the same algorithm works.
How good is our learning algorithm?

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

**Learning Conjunctions**

**Protocol II:** The teacher (who knows f) provides training examples
Learning Conjunctions

Protocol II: The teacher (who knows $f$) provides training examples

- **First**: Teacher gives a superset of the good variables
  
  $<(0,1,1,1,0,...,0,1), 1>$

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]
Learning Conjunctions

**Protocol II:** The teacher (who knows f) provides training examples

- **First:** Teacher gives a superset of the good variables
  \[(0,1,1,1,1,0,...,0,1), 1\]

- **Next:** Teacher proves that each of these variables are required
  - \[(0,0,1,1,1,0,...,0,1), 0\] need \(x_2\)
  - \[(0,1,0,1,1,0,...,0,1), 0\] need \(x_3\)
  - ...
  - \[(0,1,1,1,1,0,...,0,0), 0\] need \(x_{100}\)
Learning Conjunctions

**Protocol II**: The teacher (who knows f) provides training examples

- **First**: Teacher gives a superset of the good variables
  \[<(0,1,1,1,1,0,...,0,1), 1>\]
  These variables are sufficient

- **Next**: Teacher proves that each of these variables are required
  - \[<(0,0,1,1,1,0,...,0,1), 0>\] need \(x_2\)
  - \[<(0,1,0,1,1,0,...,0,1), 0>\] need \(x_3\)
  - ...  
  - \[<(0,1,1,1,1,0,...,0,0), 0>\] need \(x_{100}\)
  All the variables are necessary
Learning Conjunctions

Protocol II: The teacher (who knows $f$) provides training examples

• **First**: Teacher gives a superset of the good variables
  
  $$<(0,1,1,1,1,0,...,0,1), 1>$$

• **Next**: Teacher proves that each of these variables are required
  
  - $$<(0,0,1,1,0,...,0,1), 0>$$ need $x_2$
  - $$<(0,1,0,1,1,0,...,0,1), 0>$$ need $x_3$
  - $$<(0,1,1,1,1,0,...,0,0), 0>$$ need $x_{100}$

A straightforward algorithm requires $k = 6$ examples to produce the hidden conjunction (exactly)
Learning Conjunctions

**Protocol II**: The teacher (who knows f) provides training examples

- **First**: Teacher gives a superset of the good variables
  \((0,1,1,1,0,...,0,1), 1\)

- **Next**: Teacher proves that each of these variables are required
  
  - \((0,0,1,1,0,...,0,1), 0\) need \(x_2\)
  - \((0,1,0,1,1,0,...,0,1), 0\) need \(x_3\)
  - ... 
  - \((0,1,1,1,1,0,...,0,0), 0\) need \(x_{100}\)

A straightforward algorithm requires \(k = 6\) examples to produce the hidden conjunction (exactly)

\[
h = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}
\]

Modeling teaching can be very difficult, unfortunately

\[
f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}
\]
Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels \( f(x) \)

- \((1,1,1,1,1,...,1,1), 1\)
- \((1,1,1,0,0,...,0,0), 0\)
- \((1,1,1,1,1,0,...0,1,1), 1\)
- \((1,0,1,1,1,0,...0,1,1), 0\)
- \((1,1,1,1,0,...0,0,1), 1\)
- \((0,1,0,1,0,0,...0,1,1), 0\)
- \((1,1,1,1,1,...,0,1), 1\)
- \((1,1,1,1,1,...,0,1,1), 0\)
Learning Conjunctions

Protocol III: Some random source (nature) provides training examples

Teacher (Nature) provides the labels (f(x))

- \<(1,1,1,1,1,\ldots,1,1), 1>\
- \<(1,1,1,0,0,\ldots,0,0), 0>\
- \<(1,1,1,1,0,\ldots,0,1,1), 1>\
- \<(1,0,1,1,1,0,\ldots,0,1,1), 0>\
- \<(1,1,1,1,0,\ldots,0,0,1), 1>\
- \<(1,0,1,0,0,\ldots,0,1,1), 0>\
- \<(1,1,1,1,1,\ldots,0,1), 1>\
- \<(0,1,0,1,0,\ldots,0,1,1), 0>\

How good is our learning algorithm?

\[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

Look for the variables that are present in all positive examples.
How good is our learning algorithm?

Learning Conjunctions

**Protocol III**: Some random source (nature) provides training examples

Teacher (Nature) provides the labels \( f(x) \)

- \(<(1,1,1,1,1,\ldots,1,1), 1>\)
- \(<(1,1,1,0,0,\ldots,0,0), 0>\)
- \(<(1,1,1,1,1,\ldots,0,1,1), 1>\)
- \(<(0,1,1,1,1,\ldots,0,1,1), 0>\)
- \(<(1,0,1,1,1,0,\ldots,0,1), 1>\)
- \(<(1,1,1,1,1,0,\ldots,0,0,1), 1>\)
- \(<(1,0,1,0,0,0,\ldots,0,1,1), 0>\)
- \(<(1,1,1,1,1,1,\ldots,0,1), 1>\)
- \(<(0,1,0,1,0,0,\ldots,0,1,1), 0>\)

For a reasonable learning algorithm (by **elimination**), the final hypothesis will be

\[
h = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_{100}
\]
How good is our learning algorithm?

**Learning Conjunctions**

**Protocol III:** Some random source (nature) provides training examples

Teacher (Nature) provides the labels \( f(x) \)

- \( <(1,1,1,1,1,...,1,1), 1> \)
- \( <(1,1,1,0,0,...,0,0), 0> \)
- \( <(1,1,1,1,1,0,...0,1,1), 1> \)
- \( <(1,0,1,1,1,...0,1,1), 0> \)
- \( <(1,1,1,1,1,0...0,0,1), 1> \)
- \( <(1,0,1,0,0,...0,1,1), 0> \)
- \( <(1,1,1,1,1,...,0,1), 1> \)
- \( <(0,1,0,1,0,...0,1,1), 0> \)

For a reasonable learning algorithm (by **elimination**), the final hypothesis will be

\[ h = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]

Whenever the output is 1, \( x_1 \) is present.
Learning Conjunctions

**Protocol III**: Some random source (nature) provides training examples

Teacher (Nature) provides the labels \( f(x) \)

- \((1,1,1,1,1,...,1,1), 1\)
- \((1,1,1,0,0,...,0,0), 0\)
- \((1,1,1,1,0,...0,1,1), 1\)
- \((1,0,1,1,1,...0,1,1), 0\)
- \((1,1,1,1,0,...0,0,1), 1\)
- \((1,0,1,0,0,...0,1,1), 0\)
- \((1,1,1,1,1,...,0,1), 1\)
- \((0,1,0,1,0,...0,1,1), 0\)

For a reasonable learning algorithm (by elimination), the final hypothesis will be

\[
h = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_{100}
\]

Whenever the output is 1, \( x_1 \) is present.

With the given data, we only learned an approximation to the true concept.

Is it good enough?
Two Directions for How good is our learning algorithm?

• Can analyze the probabilistic intuition
  – Never saw $x_1=0$ in positive examples, maybe we’ll never see it
  – And if we do, it will be with small probability, so the concepts we learn may be *pretty good*
    • *Pretty good*: In terms of performance on future data
  – *Probably Approximately Correct (PAC) framework*
Two Directions for How good is our learning algorithm?

• Can analyze the probabilistic intuition
  – Never saw $x_1=0$ in positive examples, maybe we’ll never see it
  – And if we do, it will be with small probability, so the concepts we learn may be pretty good
    • Pretty good: In terms of performance on future data
  – Probably Approximately Correct (PAC) framework

• The number of mistakes made in learning
  – Mistake bound driven learning: Update your hypothesis only when you make mistakes
  – Define good in terms of how many mistakes you make before you stop
Where are we?

• How good is a learning algorithm?

• Online learning
  – The mistake bound model/algorithm
  – A proof of concept mistake bound algorithm: The Halving algorithm
  – Examples
  – Representations and ease of learning
Big picture
Big picture

Last lecture: Linear models
Big picture

Last lecture: Linear models

How good is a learning algorithm?
Big picture

Last lecture: Linear models

Mistake
Driven
learning

How good is a learning algorithm?
Last lecture: Linear models

Mistake Driven learning

Perceptron, Winnow

How good is a learning algorithm?
How good is a learning algorithm?

Last lecture: Linear models

Mistake Driven learning

Perceptron, Winnow

Support Vector Machines

PAC, Empirical Risk Minimization

…. 
Mistake-Driven Learning

Coming up

• Online Learning

• Learning algorithms for linear models
  – Perceptron (with many variations)
  – Winnow (if time permits)
  – General Gradient Descent view
Where are we?

• How good is a learning algorithm?

• Online learning
  – The mistake bound model
  – A proof of concept mistake bound algorithm: The Halving algorithm
  – Examples
  – Representations and ease of learning
Mistake bound algorithms

- **Setting:**
  - Instance space: $X$ (dimensionality $n$)
  - Target $f: X \to \{0,1\}$, $f \in C$, the concept class (parameterized by $n$)

- Performance: learner makes a mistake when $h(x) \neq f(x)$

- $M_A(f, S)$: Number of mistakes algorithm $A$ makes on sequence $S$ of examples for the target function $f$

- $M_A(C) = \max_{f \in C} M_A(f, S)$: The maximum possible number of mistakes made by $A$ for any target function in $C$ and any sequence $S$ of examples

- Algorithm $A$ is a mistake bound algorithm for the concept class $C$ if $M_A(C)$ is a polynomial in $n$
Mistake bound algorithms

• **Setting:**
  – Instance space: $X$ (dimensionality $n$)
  – Target $f: X \rightarrow \{0,1\}$, $f \in C$, the concept class (parameterized by $n$)

• **Learning Protocol (online):**
  – Learner is given $x \in X$, randomly chosen
  – Learner predicts $h(x)$, and is then given $f(x)$ (feedback)
Mistake bound algorithms

- **Setting:**
  - Instance space: $X$ (dimensionality $n$)
  - Target $f: X \rightarrow \{0,1\}$, $f \in C$, the concept class (parameterized by $n$)

- **Learning Protocol (online):**
  - Learner is given $x \in X$, randomly chosen
  - Learner predicts $h(x)$, and is then given $f(x)$ (feedback)

- **Performance:** learner makes a mistake when $h(x) \neq f(x)$
Mistake bound algorithms

• **Setting:**
  – Instance space: $X$ (dimensionality $n$)
  – Target $f: X \rightarrow \{0,1\}$, $f \in C$, the concept class (parameterized by $n$)

• **Learning Protocol (online):**
  – Learner is given $x \in X$, randomly chosen
  – Learner predicts $h(x)$, and is then given $f(x)$ (feedback)

• **Performance:** learner makes a mistake when $h(x) \neq f(x)$
  – $M_A(f, S)$: Number of mistakes algorithm $A$ makes on sequence $S$ of examples for the target function $f$
Mistake bound algorithms

• Setting:
  – Instance space: \( X \) (dimensionality \( n \))
  – Target \( f: X \rightarrow \{0,1\}, f \in C \), the concept class (parameterized by \( n \))

• Learning Protocol (online):
  – Learner is given \( x \in X \), randomly chosen
  – Learner predicts \( h(x) \), and is then given \( f(x) \) (feedback)

• Performance: learner makes a mistake when \( h(x) \neq f(x) \)
  – \( M_A(f, S) \): Number of mistakes algorithm \( A \) makes on sequence \( S \) of examples for the target function \( f \)
  – \( M_A(C) = \max_{f \in C, S} M_A(f, S) \): The maximum possible number of mistakes made by \( A \) for any target function in \( C \) and any sequence \( S \) of examples
Mistake bound algorithms

• **Setting:**
  – Instance space: $X$ (dimensionality $n$)
  – Target $f: X \rightarrow \{0,1\}$, $f \in C$, the concept class (parameterized by $n$)

• **Learning Protocol (online):**
  – Learner is given $x \in X$, randomly chosen
  – Learner predicts $h(x)$, and is then given $f(x)$ (feedback)

• **Performance:** learner makes a mistake when $h(x) \neq f(x)$
  – $M_A(f, S)$: Number of mistakes algorithm $A$ makes on sequence $S$ of examples for the target function $f$
  – $M_A(C) = \max_{f \in C, S} M_A(f, S)$: The maximum possible number of mistakes made by $A$ for any target function in $C$ and any sequence $S$ of examples

• Algorithm $A$ is a **mistake bound algorithm** for the concept class $C$ if $M_A(C)$ is a polynomial in $n$
Mistake bound algorithms

- **Setting:**
  - Instance space: $X$ (dimensionality $n$)
  - Target $f: X \rightarrow \{0,1\}$, $f \in C$, the concept class (parameterized by $n$)

- **Learning Protocol (online):**
  - Learner is given $x \in X$, randomly chosen
  - Learner predicts $h(x)$, and is then given $f(x)$ (feedback)

- **Performance:** learner makes a mistake when $h(x) \neq f(x)$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence $S$ of examples for the target function $f$
  - $M_A(C) = \max_{f \in C, S} M_A(f, S)$: The maximum possible number of mistakes made by A for any target function in $C$ and any sequence $S$ of examples

- **Algorithm A is a mistake bound algorithm** for the concept class $C$ if $M_A(C)$ is a polynomial in $n$
Online/Mistake Bound Learning

- No assumptions about the distribution of examples

- Examples are presented to the learning algorithm in a sequence. For each example:
  1. Learner gets an unlabeled example
  2. Learner makes a prediction
  3. Then, the true label is revealed

- Count the number of mistakes

- A concept class is learnable in the **mistake bound model** if there exists an learning algorithm that makes a polynomial number of mistakes for any sequence of training examples
  - Polynomial in the dimensionality of the examples
Online/Mistake Bound Learning

• Simple and intuitive, widely applicable
  – Robot in an assembly line, language learning,…

• Efficient in the case of very large data sets, when the data cannot fit memory (streaming data)

• No (or not much) memory: get example, update hypothesis, get rid of it

• **Goal**: We will try to make the smallest number of mistakes in the long run.
Online/Mistake Bound Learning

• **Advantages:**
  – Simple
  – Generic conversion to other learning models (online-to-batch conversion)

• **Drawbacks:**
  – Too simple
  – Global behavior: not clear when will the mistakes be made
Is counting mistakes enough?

• Under the mistake bound model, we are not concerned about the number of examples needed to learn a function

• We only care about not making mistakes

• Eg: Suppose the learner is presented the same example over and over
  – Under the mistake bound model, it is okay
  – We won’t be able to learn the function, but we won’t make any mistakes either!
Where are we?

• How good is a learning algorithm?

• Online learning
  – The mistake bound model
  – **Proof of concept mistake driven algorithms**
  – Examples
  – Representations and ease of learning
Generic Mistake Driven Algorithms

Let $C$ be a finite concept class.

**Goal**: To learn a hidden $f \in C$

- Initialize $C_0 = C$, the set of all possible functions.
- When an example $x$ arrives:
  - Predict the label for $x$ as 1 if a majority of the functions in $C_i$ predict 1. Otherwise 0. That is, output = 1 if
    - If prediction $\neq f(x)$:
      - Update $C_{i+1} = \text{all elements of } C_i$ that agree with $f(x)$.
- Learning ends when there is only one element in $C_i$.

We will construct a series of sets of functions $C_i$.
Generic Mistake Driven Algorithms

• Let $C$ be a finite concept class
• Goal: Learn $f \in C$

• Algorithm $\text{CON}$ (short for consistent):
Generic Mistake Driven Algorithms

- Let C be a finite concept class
- Goal: Learn \( f \in C \)

- Algorithm \text{CON} (short for consistent):
  - In the \( i^{\text{th}} \) stage of the algorithm:
    - \( C_i \) is a subset of concepts in C
Generic Mistake Driven Algorithms

- Let $C$ be a finite concept class
- Goal: Learn $f \in C$

- Algorithm **CON** (short for consistent):
  In the $i^{th}$ stage of the algorithm:
  - $C_i$ is a subset of concepts in $C$
  - Randomly choose $f \in C_i$ and use it to predict the next example

Clearly, $C_{i+1} \subseteq C_i$

If a mistake is made on the $i^{th}$ example, then $|C_{i+1}| < |C_i|$

Does this mean we can do better?
Generic Mistake Driven Algorithms

• Let $C$ be a finite concept class
• Goal: Learn $f \in C$

• Algorithm $\text{CON}$ (short for consistent):
  In the $i^{th}$ stage of the algorithm:
  – $C_i$ is a subset of concepts in $C$
  – Randomly choose $f \in C_i$ and use it to predict the next example
  – If a mistake is made, remove $f$ from $C_i$ to obtain $C_{i+1}$
Generic Mistake Driven Algorithms

• Let $C$ be a finite concept class
• Goal: Learn $f \in C$

• Algorithm $\text{CON}$ (short for consistent):
  In the $i^{th}$ stage of the algorithm:
  – $C_i$ is a subset of concepts in $C$
  – Randomly choose $f \in C_i$ and use it to predict the next example
  – If a mistake is made, remove $f$ from $C_i$ to obtain $C_{i+1}$
• Clearly, $C_{i+1} \subseteq C_i$
Generic Mistake Driven Algorithms

- Let C be a finite concept class
- Goal: Learn $f \in C$

- Algorithm **CON** (short for consistent):
  - In the $i^{th}$ stage of the algorithm:
    - $C_i$ is a subset of concepts in C
    - Randomly choose $f \in C_i$ and use it to predict the next example
    - If a mistake is made, remove $f$ from $C_i$ to obtain $C_{i+1}$
  - Clearly, $C_{i+1} \subseteq C_i$

  - If a mistake is made on the $i^{th}$ example, then $|C_{i+1}| < |C_i|$ progress is made
Generic Mistake Driven Algorithms

- Let $C$ be a finite concept class
- Goal: Learn $f \in C$

- Algorithm $\text{CON}$ (short for consistent):
  - In the $i^{th}$ stage of the algorithm:
    - $C_i$ is a subset of concepts in $C$
    - Randomly choose $f \in C_i$ and use it to predict the next example
    - If a mistake is made, remove $f$ from $C_i$ to obtain $C_{i+1}$
  - Clearly, $C_{i+1} \subseteq C_i$

- If a mistake is made on the $i^{th}$ example, then $|C_{i+1}| < |C_i|$
  progress is made

- The $\text{CON}$ algorithm makes at most $|C|-1$ mistakes
Generic Mistake Driven Algorithms

• Let C be a finite concept class
• Goal: Learn \( f \in C \)

• Algorithm **CON** (short for consistent):
  In the \( i^{th} \) stage of the algorithm:
  – \( C_i \) is a subset of concepts in C
  – Randomly choose \( f \in C_i \) and use it to predict the next example
  – If a mistake is made, remove \( f \) from \( C_i \) to obtain \( C_{i+1} \)
• Clearly, \( C_{i+1} \subseteq C_i \)

• If a mistake is made on the \( i^{th} \) example, then \( |C_{i+1}| < |C_i| \)
  progress is made

• The **CON** algorithm makes at most \( |C|-1 \) mistakes

  *Is this a mistake bound algorithm? Can we do better?*
The Halving Algorithm

Let $C$ be a finite concept class.

**Goal**: To learn a hidden $f \in C$

- Initialize $C_0 = C$, the set of all possible functions
- When an example $x$ arrives:
  - Predict the label for $x$ as 1 if a majority of the functions in $C_i$ predict 1. Otherwise 0. That is, output = 1 if
    - If prediction $\neq f(x)$:
      - Update $C_{i+1} = \text{all elements of } C_i \text{ that agree with } f(x)$

We will construct a series of sets of functions $C_i$

- Learning ends when there is only one element in $C_i$
The Halving Algorithm

Let $C$ be a finite concept class.

**Goal**: To learn a hidden $f \in C$

- Initialize $C_0 = C$, the set of all possible functions
- When an example $x$ arrives:
  - Predict the label for $x$ as 1 if a majority of the functions in $C_i$ predict 1. Otherwise 0. That is, output $= 1$ if
    $$\left| \{h(x) = 1 : h \in C_i \} \right| > \left| \{h(x) = 0 : h \in C_i \} \right|$$

- Learning ends when there is only one element in $C_i$
The Halving Algorithm

Let $C$ be a finite concept class.

**Goal**: To learn a hidden $f \in C$

- Initialize $C_0 = C$, the set of all possible functions
- When an example $x$ arrives:
  - Predict the label for $x$ as 1 if a **majority** of the functions in $C_i$ predict 1. Otherwise 0. That is, output = 1 if
    \[
    |\{h(x) = 1 : h \in C_i\}| > |\{h(x) = 0 : h \in C_i\}|
    \]
  - If prediction $\neq f(x)$: *(i.e error)*
    - Update $C_{i+1} = $ all elements of $C_i$ that agree with $f(x)$
    - Learning ends when there is only one element in $C_i$
The Halving Algorithm

Let $C$ be a finite concept class.

**Goal**: To learn a hidden $f \in C$

- Initialize $C_0 = C$, the set of all possible functions.
- When an example $x$ arrives:
  - Predict the label for $x$ as 1 if a majority of the functions in $C_i$ predict 1. Otherwise 0. That is, output $= 1$ if
    $$\left|\{h(x) = 1 : h \in C_i\}\right| > \left|\{h(x) = 0 : h \in C_i\}\right|$$
  - If prediction $\neq f(x)$: (i.e error)
    - Update $C_{i+1} = \text{all elements of } C_i \text{ that agree with } f(x)$
  - Learning ends when there is only one element in $C_i$

How many mistakes will the Halving algorithm make?
How many mistakes will the Halving algorithm make?

Suppose it makes $n$ mistakes. Finally, we will have the final set of concepts $C_n$ with one element $C_n$ was created when a majority of the functions in $C_{n-1}$ were incorrect.

$$1 = |C_n| < \frac{1}{2}|C_{n-1}|$$
How many mistakes will the Halving algorithm make?

Suppose it makes $n$ mistakes. Finally, we will have the final set of concepts $C_n$ with one element

$C_n$ was created when a majority of the functions in $C_{n-1}$ were incorrect

$$1 = |C_n| < \frac{1}{2} |C_{n-1}|$$
$$< \frac{1}{2} \cdot \frac{1}{2} |C_{n-2}|$$
$$< \vdots$$
$$< \frac{1}{2^n} |C_0| = \frac{1}{2^n} |C|$$
How many mistakes will the Halving algorithm make?

Suppose it makes \( n \) mistakes. Finally, we will have the final set of concepts \( C_n \) with one element

\( C_n \) was created when a majority of the functions in \( C_{n-1} \) were incorrect

\[
1 = |C_n| < \frac{1}{2} |C_{n-1}|
< \frac{1}{2} \cdot \frac{1}{2} |C_{n-2}|
< \ldots
< \frac{1}{2^n} |C_0| = \frac{1}{2^n} |C|
\]

\(|C| > 2^n\)
How many mistakes will the Halving algorithm make?

Suppose it makes \( n \) mistakes. Finally, we will have the final set of concepts \( C_n \) with one element.

\( C_n \) was created when a majority of the functions in \( C_{n-1} \) were incorrect.

The Halving algorithm will make at most \( \log(|C|) \) mistakes.
The Halving Algorithm

• Hard to compute

• In some concept classes, Halving is \textit{optimal}
  – Eg: for class of all conjunctions
The Halving Algorithm

• Hard to compute

• In some concept classes, Halving is *optimal*
  – Eg: for class of all conjunctions

For the most difficult concept in the class,

For the most difficult sequence of examples,

The *optimal* mistake bound algorithm makes the fewest number of mistakes
The Halving Algorithm

• Hard to compute

• In some concept classes, Halving is *optimal*
  – Eg: for class of all conjunctions

• In general, to be optimal, instead of guessing in accordance with the majority of the valid concepts, we should guess according to the concept group that gives the least number of expected mistakes (even harder to compute)

For the most difficult concept in the class,
For the most difficult sequence of examples,
The *optimal* mistake bound algorithm makes the fewest number of mistakes
Where are we?

• How good is a learning algorithm?

• Online learning
  – The mistake bound model
  – A proof of concept mistake bound algorithm: The Halving algorithm
  – Examples
  – Representations and ease of learning
Learning Conjunctions

• Hidden function: **conjunctions**
  – The learner is to learn functions like $f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}$
• Number of conjunctions with $n$ variables = $??$

$\log(|C|) = O(n)$

The elimination algorithm makes at most $n$ mistakes

Assume that only $k << n$ attributes occur in the conjunction

$\log(|C|) = k \log(n)$

Can we learn efficiently with this number of mistakes?
Learning Conjunctions

- Hidden function: conjunctions
  - The learner is to learn functions like $f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100}$
- Number of conjunctions with $n$ variables = $3^n$
Learning Conjunctions

• Hidden function: **conjunctions**
  – The learner is to learn functions like \( f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \)

• Number of conjunctions with \( n \) variables = \( 3^n \)
  – \( \log(|C|) = O(n) \)
Learning Conjunctions

• Hidden function: **conjunctions**
  – The learner is to learn functions like \( f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \)
• Number of conjunctions with \( n \) variables = \( 3^n \)
  – \( \log(|C|) = O(n) \)

• Hidden function: **\( k \)-conjunctions**
  – Assume that only \( k \ll n \) attributes occur in the conjunction
• Number of \( k \)-conjunctions: \( 2^k C(n,k) \approx 2^k n^k \)
  – \( \log(|C|) = k \log(n) \)
  – Can we learn efficiently with this number of mistakes?
Where are we?

• How good is a learning algorithm?

• Online learning
  – The mistake bound model
  – Proof of concept mistake driven algorithms
  – Examples
  – Representations and ease of learning
Motivation

• Consider a learning problem in a very high dimensional space
  \( \{x_1, x_2, x_3, \ldots, x_{1000000} \} \)

• And assume that the function space is very sparse (the function of interest depends on a small number of attributes.)

  \[ f = x_2 \land x_3 \land x_4 \land x_5 \land x_{100} \]
Representation and efficient learning

• Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?

- Theorem [Haussler 1988]: Given a sample on n attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the sample and has the minimum number of attributes.

- Same holds for Disjunctions

Proof intuition: Reduction to minimum set cover problem

Given a collection of sets that cover X, define a set of examples so that learning the best (dis/con)junction implies a minimal cover.

We cannot learn the concept efficiently as a (dis/con)junction

• But, we will see that we can do that, if we are willing to learn the concept as a Linear Threshold function.

In a more expressive class, the search for a good hypothesis sometimes becomes combinatorially easier
Representation and efficient learning

• Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?

• **Theorem** [Haussler 1988]: Given training data on n attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the data *and has the minimum number of attributes.*
  
  – Same holds for Disjunctions
Representation and efficient learning

• Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?

• **Theorem** [Haussler 1988]: Given training data on n attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the data *and has the minimum number of attributes.*
  — Same holds for Disjunctions

• Proof by reduction to minimum set cover problem
  ⇒ We cannot learn the concept efficiently as a *(dis/con)*junction
Representation and efficient learning

• Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?

• **Theorem** [Haussler 1988]: Given training data on n attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the data and has the minimum number of attributes.
  
  — Same holds for Disjunctions

• Proof by reduction to minimum set cover problem
  
  ⇒ We cannot learn the concept efficiently as a (dis/con)junction

• But, we will see that we can do that, if we are willing to learn the concept as a Linear Threshold function.
Representation and efficient learning

• Assume that you want to learn conjunctions. Should your hypothesis space be the class of conjunctions?

• **Theorem** [Haussler 1988]: Given training data on n attributes that is consistent with a conjunctive concept, it is NP-hard to find a pure conjunctive hypothesis that is both consistent with the data and has the minimum number of attributes.
  – Same holds for Disjunctions

• Proof by reduction to minimum set cover problem
  ⇒ We cannot learn the concept efficiently as a (dis/con)junction

• But, we will see that we can do that, if we are willing to learn the concept as a Linear Threshold function.

In a more expressive class, the search for a good hypothesis sometimes becomes combinatorially easier
Summary: The Halving algorithm

• A simple algorithm for *finite* concept spaces
  – Stores a set of hypotheses that it iteratively refines
    • Receive an input
    • **Prediction:** the label of the majority of hypotheses still under consideration
    • **Update:** If incorrect, remove all inconsistent hypotheses

• Makes $\log |C|$ mistakes for a concept class $C$
What you should know

• What is the mistake bound model?

• Simple *proof-of-concept* mistake bound algorithms
  – CON: Makes $O(|C|)$ mistakes
  – The Halving algorithm
    • Can learn a concept with at most $\log(|C|)$ mistakes
    • Sadly, for non-trivial functions, only useful if we don’t care about storage or computation time

• Even for simple Boolean functions (conjunctions and disjunctions), learning them as *linear threshold units* is computationally easier