Online Learning

Machine Learning

THE UNIVERSITY OF UTAH

Some slides based on lectures from Dan Roth, Avrim Blum and others

Big picture

Big picture

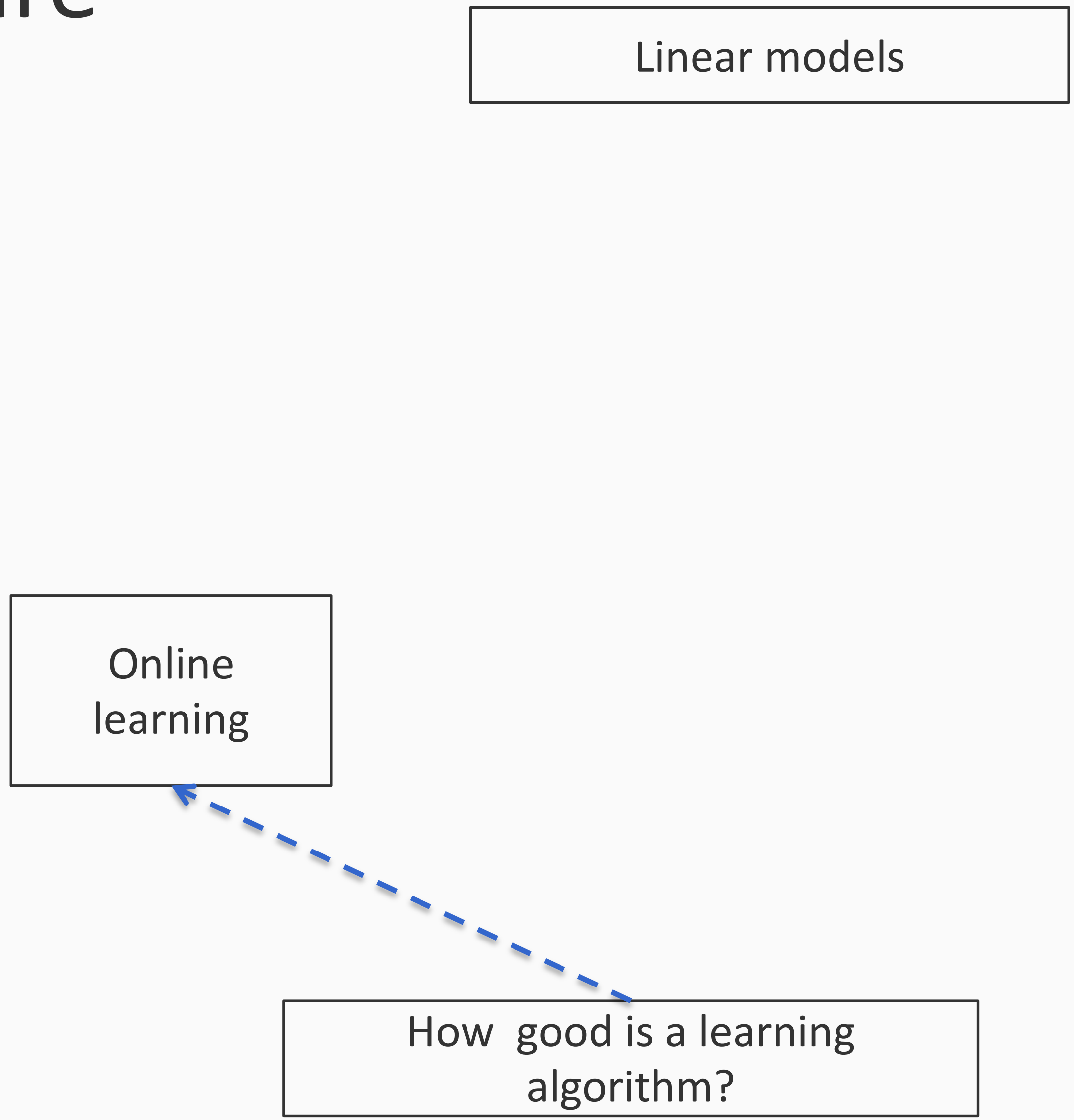
Last lecture: Linear models

Big picture

Linear models

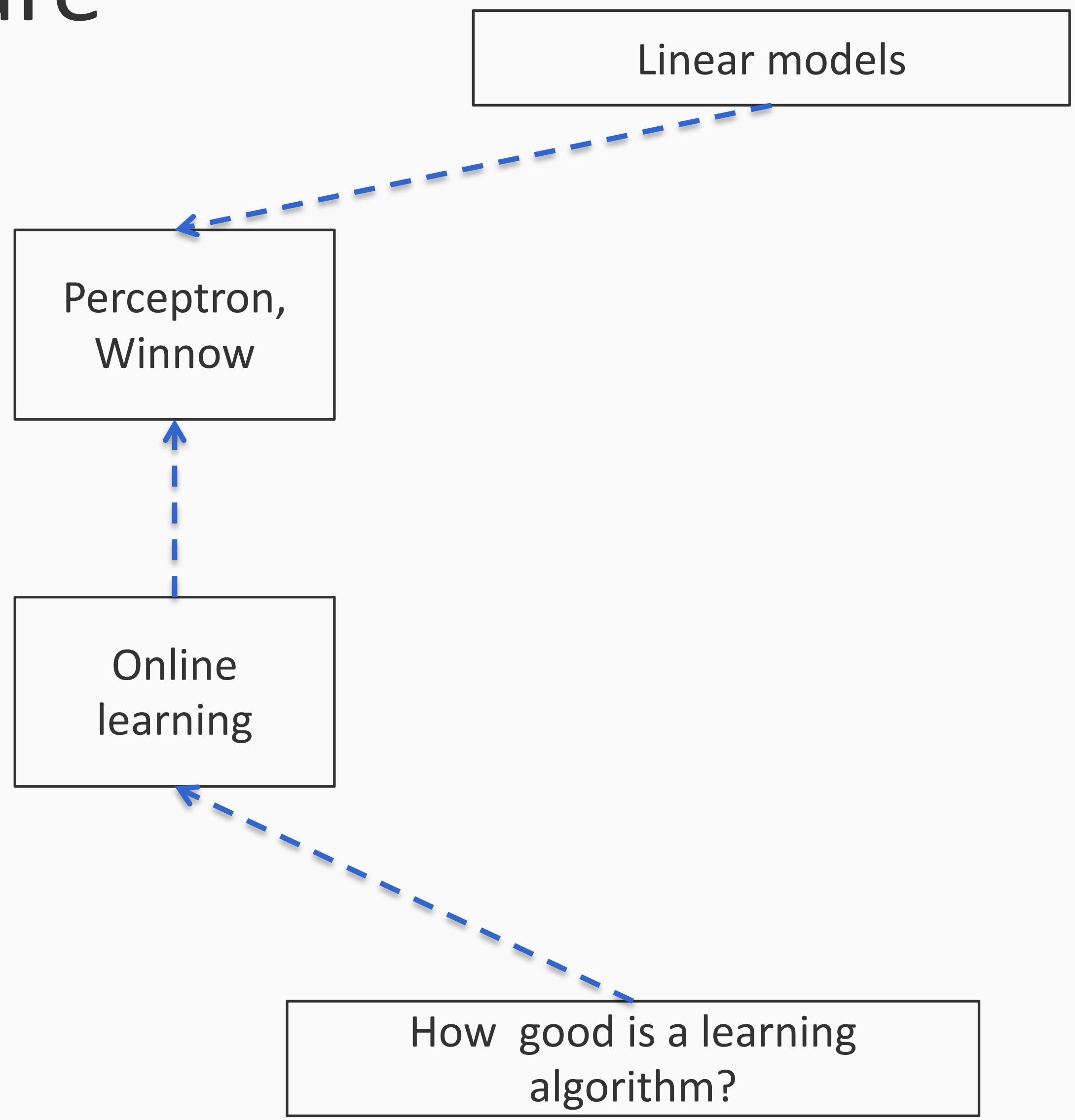
How good is a learning algorithm?

Big picture

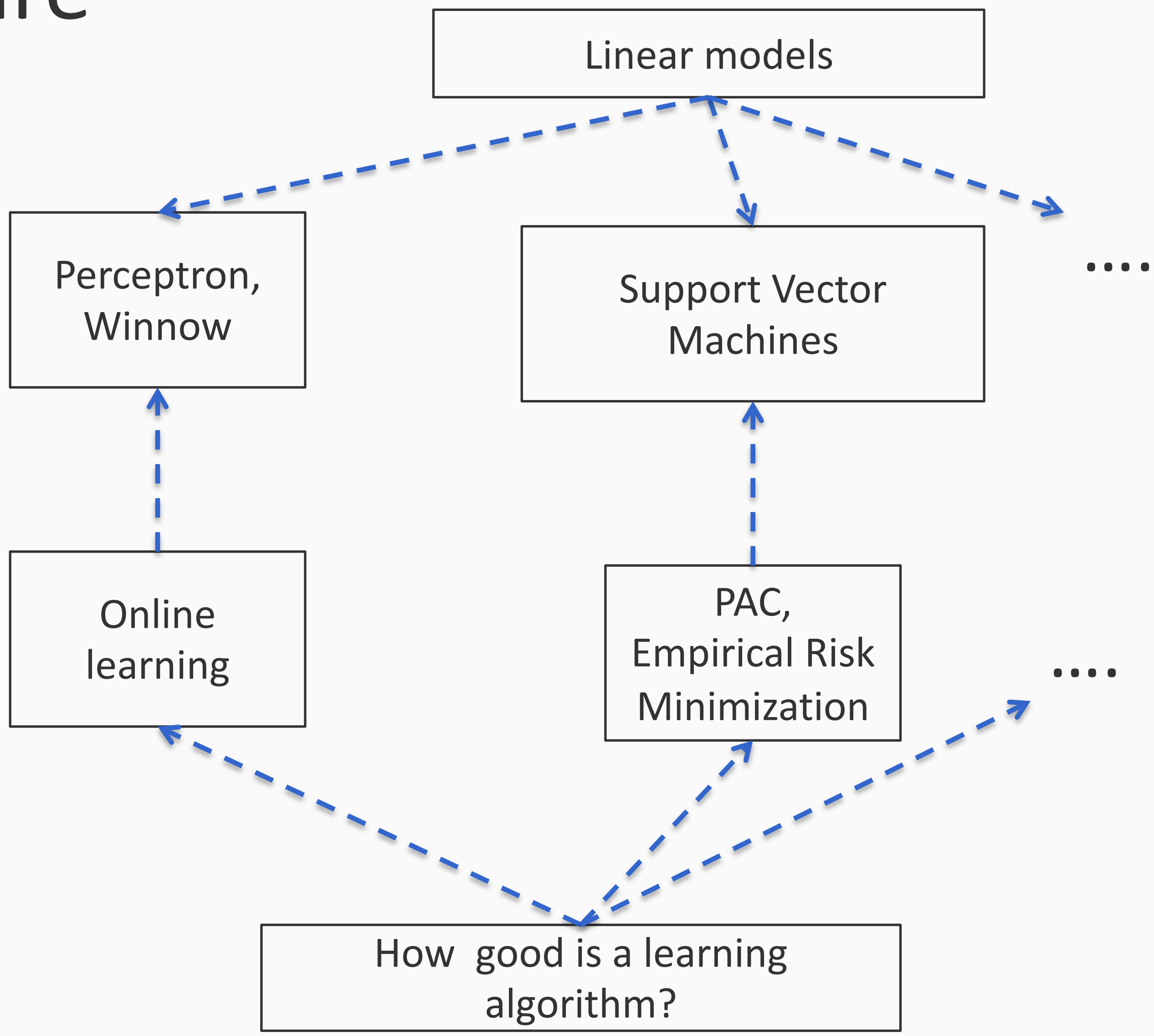


Linear models
Online learning
How good is a learning algorithm?

Big picture


Linear models
Perceptron, Winnow
Online learning
How good is a learning algorithm?

Big picture


Linear models
Perceptron, Winnow
Support Vector Machines
....
Online learning
PAC, Empirical Risk Minimization
....
How good is a learning algorithm?

Mistake bound learning

- The mistake bound model
- A proof of concept mistake bound algorithm: The Halving algorithm
- Examples
- Representations and ease of learning

Coming up…

- Mistake-driven learning

- Learning algorithms for learning a linear function over the feature space
  - Perceptron (with many variants)
  - General Gradient Descent view

Issues to watch out for

- Importance of Representation
- Complexity of Learning
- More about features

Mistake bound learning

- The mistake bound model
- A proof of concept mistake bound algorithm: The Halving algorithm
- Examples
- Representations and ease of learning

Motivation

Consider a learning problem in a very high dimensional space

$$[x_1, x_2, \cdots, x_{1000000}]$$

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

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

Motivation

Consider a learning problem in a very high dimensional space

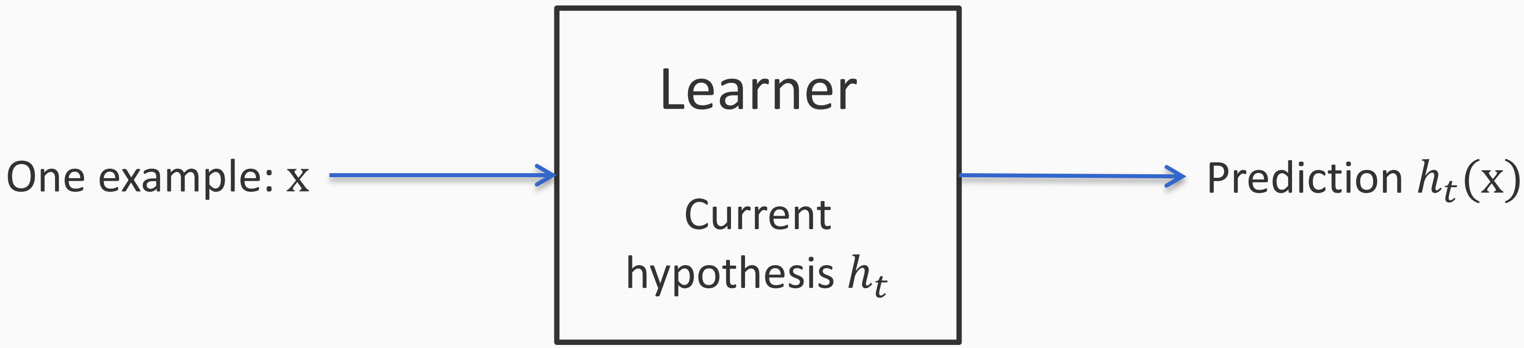
$$[x_1, x_2, \cdots, x_{1000000}]$$

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

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

- Can we develop an algorithm that depends only *weakly* on the dimensionality and mostly on the number of relevant attributes?
- How should we represent the hypothesis?

An illustration of mistake driven learning



Loop forever:

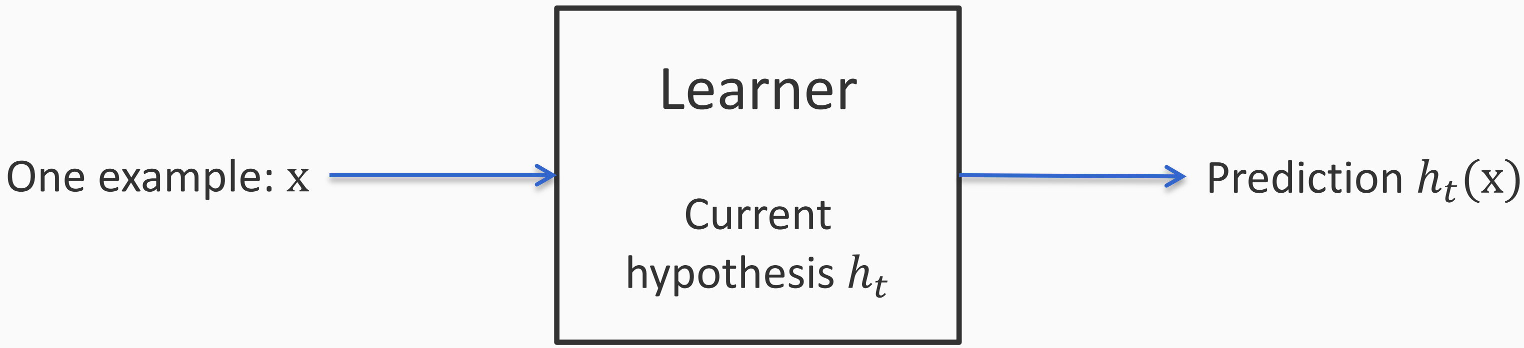
1. Receive example x

2. Make a prediction using the current hypothesis $h_t(\mathrm{x})$

3. Receive the true label for x.

4. If $h_t(\mathrm{x})$ is not correct, then:
    - Update h_t to h_{t+1}

An illustration of mistake driven learning



Loop forever:

1. Receive example x

2. Make a prediction using the current hypothesis $h_t(\mathrm{x})$

3. Receive the true label for x.

4. If $h_t(\mathrm{x})$ is not correct, then:
   - Update h_t to h_{t+1}

Only need to define how prediction and update behave

Can such a simple scheme work? How do we quantify what "work" means?

Mistake bound algorithms

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

Mistake bound algorithms

- Setting:
  - Instance space: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \to \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ ⟵ *the feedback*

Mistake bound algorithms

- Setting:
  - Instance space: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \rightarrow \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ ⟵ *the feedback*
- Performance: learner makes a mistake when $h(\mathbf{x}) \neq f(x)$

Mistake bound algorithms

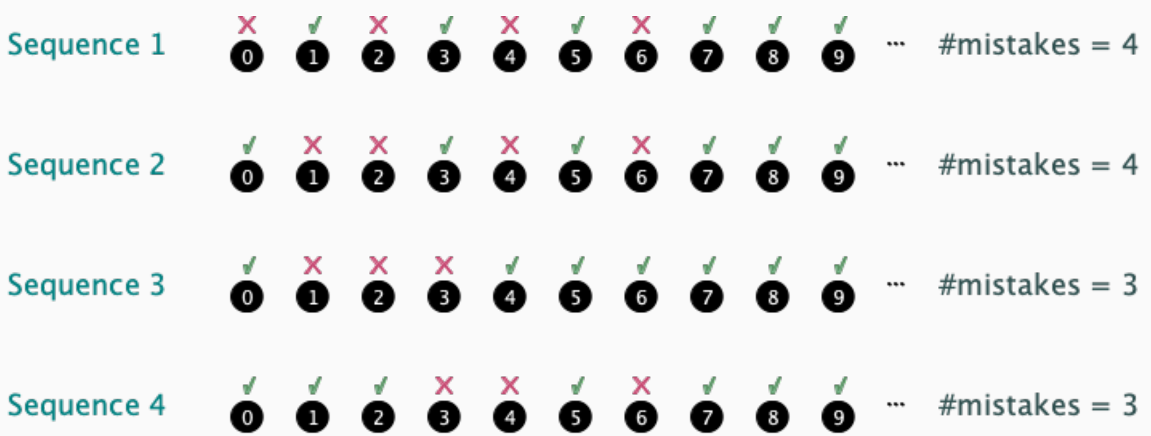
- Setting:
  - Instance space: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \to \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ ⟵ *the feedback*
- Performance: learner makes a mistake when $h(\mathbf{x}) \neq f(x)$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f

Mistake bound algorithms

- Setting:
  - Instance space: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \rightarrow \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ $\longleftarrow$ *the feedback*
- Performance: learner makes a mistake when $h(\mathbf{x}) \neq f(x)$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f

Mistake bound algorithms

- **Setting**:
  - Instance space: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \to \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- **Learning Protocol**:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ $\quad\longleftarrow$ *the feedback*
- **Performance**: learner makes a mistake when $h(\mathbf{x}) \neq f(x)$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f



Mistake bound algorithms

- Setting:
  - Instance space: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \rightarrow \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ ⟵ *the feedback*
- Performance: learner makes a mistake when $h(\mathbf{x}) \neq f(x)$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f
  - $M_A(C) = \max_{f \in S} M_A(f, S)$: 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: $\mathcal{X}$ (dimensionality n)
  - Target f: $\mathcal{X} \to \{0,1\}$, $f \in C$ the concept class (parameterized by n)
- Learning Protocol:
  - Learner is given $\mathbf{x} \in \mathcal{X}$, randomly chosen
  - Learner predicts $h(\mathbf{x})$ and is then given $f(\mathbf{x})$ ⟵ *the feedback*
- Performance: learner makes a mistake when $h(\mathbf{x}) \neq f(x)$
  - $M_A(f, S)$: Number of mistakes algorithm A makes on sequence S of examples for the target function f
  - $M_A(C) = \max_{f \in S} M_A(f, S)$: 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 the dimensionality n

Learnability in the mistake bound model

- Algorithm A is a ***mistake bound algorithm*** for the concept class C if $M_A(C)$ is a polynomial in the dimensionality n
  - That is, the maximum number of mistakes it makes for any sequence of inputs (perhaps even an adversarially chosen one) is polynomial in the dimensionality

Learnability in the mistake bound model

- Algorithm A is a ***mistake bound algorithm*** for the concept class C if $M_A(C)$ is a polynomial in the dimensionality n
  - That is, the maximum number of mistakes it makes for any sequence of inputs (perhaps even an adversarially chosen one) is polynomial in the dimensionality

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

Learnability in the mistake bound model

- Not the most general setting for online learning
- Not the most general metric
- Other metrics: Regret, cumulative loss

- Algorithm A is a ***mistake bound algorithm*** for the concept class C if $M_A(C)$ is a polynomial in the dimensionality n
  - That is, the maximum number of mistakes it makes for any sequence of inputs (perhaps even an adversarially chosen one) is polynomial in the dimensionality

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

Online Learning

- No assumptions about the distribution of examples

- Examples are presented to the learning algorithm in a sequence. *Could be adversarial!*

  For each example:

  1. Learner gets an unlabeled example
  2. Learner makes a prediction
  3. Then, the true label is revealed

- In the mistake bound model, we only count the number of mistakes

Online Learning

- Simple and intuitive model, widely applicable

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

- Evaluation: We will try to make the smallest number of mistakes in the long run.

  – Some things to think about:
    - What is the relation to the “real” goal? What is the real goal of learning?
    - Does online learning generate a hypothesis that does well on previously unseen data?

Online/Mistake Bound Learning

- No notion of data distribution; a worst case model

- No (or not much) memory: get example → update hypothesis → get rid of it

- Drawbacks:
  - Too simple
  - Global behavior: not clear when will the mistakes be made

- Advantages:
  - Simple
  - Many issues arise already in this setting
  - Generic conversion to other learning models (online-to-batch conversion)

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!*

Mistake bound learning

- The mistake bound model
- A proof of concept mistake bound algorithm: The Halving algorithm
- Examples
- Representations and ease of learning

Can mistake bound algorithms exist?

Before getting to a more useful mistake bound algorithm, let's see a proof-of-concept mistake bound algorithm

The Halving algorithm

Generic Mistake Bound Algorithms

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

Generic Mistake Bound 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 = all concepts in C consistent with all i – 1 previously seen examples

Generic Mistake Bound 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 = all concepts in C consistent with all i – 1 previously seen examples
  - Choose randomly $f \in C_i$ and use it to predict the next example

Generic Mistake Bound 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 = all concepts in C consistent with all i – 1 previously seen examples
  - Choose randomly $f \in C_i$ and use it to predict the next example

- It is not hard to show that $C_{i+1} \subseteq C_i$

Generic Mistake Bound 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 = all concepts in C consistent with all i – 1 previously seen examples
  - Choose randomly $f \in C_i$ and use it to predict the next example

- It is not hard to show that $C_{i+1} \subseteq C_i$

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

Generic Mistake Bound 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 = all concepts in C consistent with all i – 1 previously seen examples
  - Choose randomly $f \in C_i$ and use it to predict the next example

- It is not hard to show that $C_{i+1} \subseteq C_i$

- 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

Questions?

Generic Mistake Bound 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 = all concepts in C consistent with all i – 1 previously seen examples
  - Choose randomly $f \in C_i$ and use it to predict the next example

- It is not hard to show that $C_{i+1} \subseteq C_i$

- 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? Depends on what C is

Can we do better than CON?

The Halving Algorithm

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

- Initialize $C_0 = C$, the set of all possible functions

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: Learn $f \in C$

- 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

$$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$

- Learning ends when there is only one element in C_i

The Halving Algorithm

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

- 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

$$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$

  - If prediction ≠ *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: Learn $f \in C$

- 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

$$|\{h(\mathbf{x}) = 1 : h \in C_i\}| > |\{h(\mathbf{x}) = 0 : h \in C_i\}|$$

  - If prediction ≠ $f(\mathbf{x})$: (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

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

$$1 = |C_n| \quad < \quad \frac{1}{2}|C_{n-1}|$$

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

$$
\begin{aligned}
1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\
&< \frac{1}{2} \cdot \frac{1}{2}|C_{n-2}| \\
&< \vdots \\
&< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C|
\end{aligned}
$$

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

$$
\begin{aligned}
1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\
&< \frac{1}{2} \cdot \frac{1}{2}|C_{n-2}| \\
&< \vdots \\
&< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C|
\end{aligned}
$$

$$|C| > 2^n$$

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

$$\begin{aligned} 1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\ &< \frac{1}{2}\cdot\frac{1}{2}|C_{n-2}| \\ &< \vdots \\ &< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C| \end{aligned}$$

$$|C| > 2^n$$

The Halving algorithm will make at most $\log|C|$ mistakes

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

$$
\begin{aligned}
1 = |C_n| &< \frac{1}{2}|C_{n-1}| \\
&< \frac{1}{2}\cdot\frac{1}{2}|C_{n-2}| \\
&< \vdots \\
&< \frac{1}{2^n}|C_0| = \frac{1}{2^n}|C|
\end{aligned}
\qquad \Rightarrow \qquad |C| > 2^n
$$

The Halving algorithm will make at most $\log|C|$ mistakes

Questions?

The Halving Algorithm

- Hard to compute
- In some concept classes, Halving is *optimal*
  - Eg: for class of all Boolean functions

The Halving Algorithm

- Hard to compute
- In some concept classes, Halving is *optimal*
  - Eg: for class of all Boolean functions

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 Boolean functions

- 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

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 O(log|C|) mistakes for a concept class C
- Not always optimal because we care about minimizing the number of mistakes in the future!
  - What if, instead of eliminating functions that disagree with this example, we down-weight them
  - Perhaps via multiplicative or additive updates…

Mistake bound 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 $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

- Number of conjunctions with n variables = $|\mathrm{C}| = ???$

Learning Conjunctions

Hidden function: conjunctions

- The learner is to learn functions like $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

- Number of conjunctions with n variables = $|\mathrm{C}| = 3^n$

Learning Conjunctions

Hidden function: conjunctions

- The learner is to learn functions like $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

- Number of conjunctions with n variables = $|\mathrm{C}| = 3^n$
  - $\log|C| = O(n)$

Learning Conjunctions

Hidden function: conjunctions

- The learner is to learn functions like $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

- Number of conjunctions with n variables = $|\mathrm{C}| = 3^n$
  - $\log|C| = O(n)$
- There is a practical algorithm that can achieve this bound
  - Elimination: Learn from positive examples by eliminating inactive literals.

The Halving algorithm is not efficient.

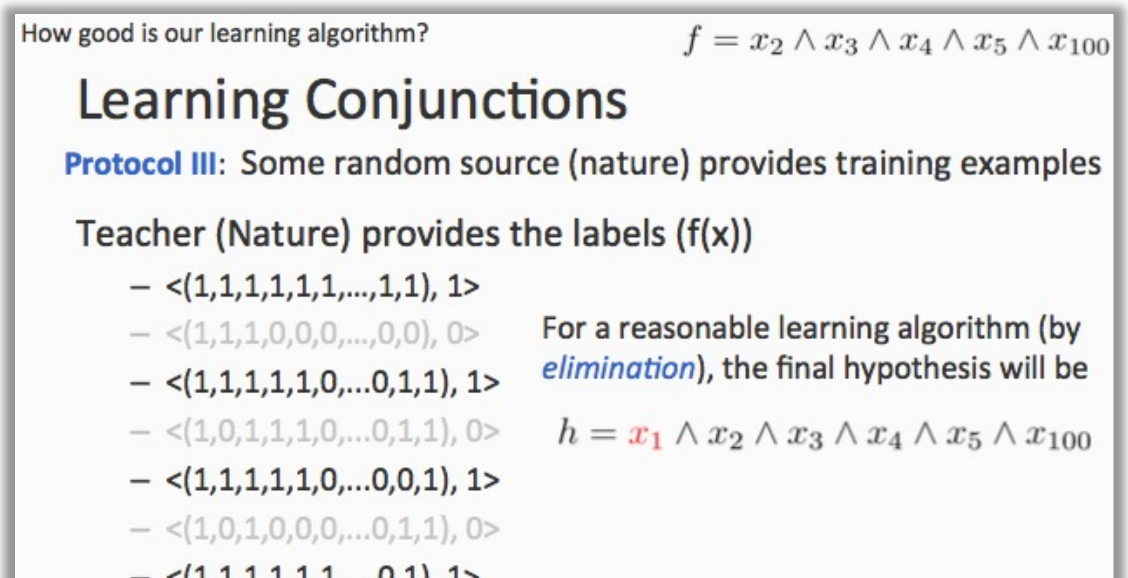
Elimination is an efficient algorithm that realizes the mistake bound of the Halving algorithm

Learning Conjunctions

Hidden function: conjunctions

- The learner is to learn functions like $f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$

- Number of conjunctions with n variables = $|\mathrm{C}| = 3^n$
  - $\log|C| = O(n)$
- There is a practical algorithm that can achieve this bound
  - Elimination: Learn from positive examples by eliminating inactive literals.



Learning Conjunctions: Elimination

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

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

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

Notation: <example, label>

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

Learning Conjunctions: Elimination

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

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

Look for the variables that are present in *every* positive example.

All other variables can be eliminated

Why?

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

Learning Conjunctions: Elimination

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

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

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

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

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

Learning Conjunctions: Elimination

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

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

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

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Whenever the output is 1, x_1 is present

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

Learning Conjunctions: Elimination

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

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

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

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

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?

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 = $|C| = 3^n$
  - $\log|C| = O(n)$
- The elimination algorithm makes at most n mistakes
  - Learn from positive examples; eliminate inactive literals.

Hidden function: *k-conjunctions*

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

- Number of k-conjunctions = $2^k \binom{n}{k} \approx 2^k n^k$ Why?
  - $\log|C| = O(k \log n)$
  - <u>*Can we learn efficiently with this number of mistakes ?*</u>

Mistake bound learning

- The mistake bound model
- A proof of concept mistake bound algorithm: The Halving algorithm
- Examples
- Representations and ease of learning

Representation and efficient learning

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

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

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 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 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 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 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 by reduction to minimum set cover problem

  $\Rightarrow$ 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

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
    - How to apply this bound to simple function classes
- Even for simple Boolean functions (conjunctions and disjunctions), learning them as linear threshold units is computationally easier