Week 1: Why Learning Theory?

The No-Free-Lunch Theorem

Tianhao Wang
tianhaowang@ucsd.edu

UCSD · Spring 2026
DSC 190/291 Topics: Learning Theory

Today

Course Overview

Logistics

Week 1: Online Prediction

The No-Free-Lunch Theorem

How Structure Helps

Course Overview

What is this course about?

Machine Learning is Everywhere

Examples:

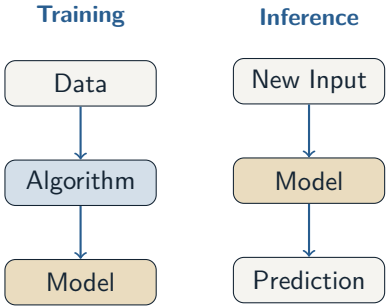
- Image classification
- Speech recognition
- Large language models (GPT, Claude)
- Recommendations (Netflix, Spotify)
- Medical diagnosis
- Fraud detection

Machine Learning is Everywhere

Examples:

- Image classification
- Speech recognition
- Large language models (GPT, Claude)
- Recommendations (Netflix, Spotify)
- Medical diagnosis
- Fraud detection

All involve learning a **prediction rule** from data.



The Central Question

Machine learning works...

The Central Question

Machine learning works... but **when**? And **why**?

The Central Question

Machine learning works... but **when**? And **why**?

Natural questions arise:

- **Statistical:** How much data do we need to learn?

The Central Question

Machine learning works... but **when**? And **why**?

Natural questions arise:

- **Statistical:** How much data do we need to learn?
- **Computational:** Can we learn efficiently?

The Central Question

Machine learning works... but **when**? And **why**?

Natural questions arise:

- **Statistical:** How much data do we need to learn?
- **Computational:** Can we learn efficiently?

But first: **Is learning always possible?**

Learning Without Assumptions

Setup: Predict labels for instances arriving one by one.

Learning Without Assumptions

Setup: Predict labels for instances arriving one by one.

No-Free-Lunch Theorem (informal):

For any learner, there exists a target function f such that the learner is wrong on ***every single prediction****.*

Learning Without Assumptions

Setup: Predict labels for instances arriving one by one.

No-Free-Lunch Theorem (informal):

For any learner, there exists a target function f such that the learner is wrong on ***every single prediction****.*

Why? An adversary can define f point by point:

- See learner's prediction on new point x
- Set $f(x) =$ opposite of prediction
- Labels come from one fixed function—but learner always loses

Learning Without Assumptions

Setup: Predict labels for instances arriving one by one.

No-Free-Lunch Theorem (informal):

For any learner, there exists a target function f such that the learner is wrong on ***every single prediction****.*

Why? An adversary can define f point by point:

- See learner's prediction on new point x
- Set $f(x) =$ opposite of prediction
- Labels come from one fixed function—but learner always loses

There is no free lunch for learning.

What This Course Studies

The resolution:

- Real problems have **structure**

What This Course Studies

The resolution:

- Real problems have **structure**
- Target functions aren't arbitrary

What This Course Studies

The resolution:

- Real problems have **structure**
- Target functions aren't arbitrary
- With the right assumptions, learning becomes possible

What This Course Studies

The resolution:

- Real problems have **structure**
- Target functions aren't arbitrary
- With the right assumptions, learning becomes possible

Learning Theory: Rigorous study of what can be learned, how much data is needed, and how efficiently.

What's Different About This Course

Learning theory is traditionally taught with paper proofs.

What's Different About This Course

Learning theory is traditionally taught with paper proofs.

This course experiments with formal verification:

- Machine-check our proofs using Lean 4

What's Different About This Course

Learning theory is traditionally taught with paper proofs.

This course experiments with formal verification:

- Machine-check our proofs using Lean 4
- AI tools to assist proof development

What's Different About This Course

Learning theory is traditionally taught with paper proofs.

This course experiments with formal verification:

- Machine-check our proofs using Lean 4
- AI tools to assist proof development
- Create human-readable explanations alongside

Paper Proof vs Formal Proof

Theorem: For all $n \in \mathbb{N}$, $\quad 0 + n = n$.

Paper proof:

"By definition of addition."

Paper Proof vs Formal Proof

Theorem: For all $n \in \mathbb{N}$, $\quad 0 + n = n$.

Paper proof:

"By definition of addition."

(Seems obvious—done!)

Lean proof:

```
theorem zero_add (n : Nat) :
    0 + n = n := by
  induction n with
  | zero => rfl
  | succ n ih => simp [ih]
```

Paper Proof vs Formal Proof

Theorem: For all $n \in \mathbb{N}$, $\quad 0 + n = n$.

Paper proof:

"By definition of addition."

(Seems obvious—done!)

Lean proof:

```
theorem zero_add (n : Nat) :
    0 + n = n := by
  induction n with
  | zero => rfl
  | succ n ih => simp [ih]
```

(Requires induction—why?)

Why the Difference?

Lean's addition is defined by recursion on the *second* argument:

```
def add : Nat -> Nat -> Nat
  | n, 0      => n                  -- base case
  | n, succ m => succ (add n m)     -- recursive case
```

Why the Difference?

Lean's addition is defined by recursion on the *second* argument:

```
def add : Nat -> Nat -> Nat
  | n, 0      => n                  -- base case
  | n, succ m => succ (add n m)     -- recursive case
```

Consequence:

- `n + 0 = n` is true *by definition* (just unfold!)

Why the Difference?

Lean's addition is defined by recursion on the *second* argument:

```
def add : Nat -> Nat -> Nat
  | n, 0      => n                    -- base case
  | n, succ m => succ (add n m)       -- recursive case
```

Consequence:

- `n + 0 = n` is true *by definition* (just unfold!)
- `0 + n = n` is *not* definitional—must prove by induction on `n`

Why the Difference?

Lean's addition is defined by recursion on the *second* argument:

```
def add : Nat -> Nat -> Nat
  | n, 0      => n                    -- base case
  | n, succ m => succ (add n m)       -- recursive case
```

Consequence:

- `n + 0 = n` is true *by definition* (just unfold!)
- `0 + n = n` is *not* definitional—must prove by induction on `n`

Formal proof exposes details that paper proofs gloss over.

Why Formal Proof?

Natural language proofs

- Checked by humans
- Can have subtle gaps
- "Remaining cases similar"

Why Formal Proof?

Natural language proofs

- Checked by humans
- Can have subtle gaps
- "Remaining cases similar"

Formal proofs (Lean, Coq, ...)

- Checked by machine
- Every step verified
- If it compiles, it's correct

Why Formal Proof?

Natural language proofs

- Checked by humans
- Can have subtle gaps
- "Remaining cases similar"

Formal proofs (Lean, Coq, ...)

- Checked by machine
- Every step verified
- If it compiles, it's correct

Formal proof forces explicit assumptions and dependencies.

Proof Assistants: A Brief History

Proof Assistants: A Brief History

Why now?

- Mature libraries (mathlib: 150k+ theorems)

Proof Assistants: A Brief History

Why now?

- Mature libraries (mathlib: 150k+ theorems)
- AI tools assist proof generation

Proof Assistants: A Brief History

Why now?

- Mature libraries (mathlib: 150k+ theorems)
- AI tools assist proof generation
- Lower barrier to entry than ever before

AI + Formal Proof: Rapid Progress

Formalizing Learning Theory

Recent efforts:

- Statistical learning theory in Lean 4 (empirical processes, Rademacher complexity) [Zhang, Lee & Liu, 2026]

Formalizing Learning Theory

Recent efforts:

- Statistical learning theory in Lean 4 (empirical processes, Rademacher complexity)
  [Zhang, Lee & Liu, 2026]
- Reinforcement learning in Coq (value/policy iteration convergence)
  [Vajjha et al., CPP 2021]

Formalizing Learning Theory

Recent efforts:

- Statistical learning theory in Lean 4 (empirical processes, Rademacher complexity)
  [Zhang, Lee & Liu, 2026]
- Reinforcement learning in Coq (value/policy iteration convergence)
  [Vajjha et al., CPP 2021]
- VC dimension and PAC learning: work in progress

Formalizing Learning Theory

Recent efforts:

- Statistical learning theory in Lean 4 (empirical processes, Rademacher complexity)
  [Zhang, Lee & Liu, 2026]
- Reinforcement learning in Coq (value/policy iteration convergence)
  [Vajjha et al., CPP 2021]
- VC dimension and PAC learning: work in progress

Why formalize?

- Exposes hidden assumptions

Formalizing Learning Theory

Recent efforts:

- Statistical learning theory in Lean 4 (empirical processes, Rademacher complexity)
  [Zhang, Lee & Liu, 2026]
- Reinforcement learning in Coq (value/policy iteration convergence)
  [Vajjha et al., CPP 2021]
- VC dimension and PAC learning: work in progress

Why formalize?

- Exposes hidden assumptions
- Clarifies dependencies between lemmas

Formalizing Learning Theory

Recent efforts:

- Statistical learning theory in Lean 4 (empirical processes, Rademacher complexity)
  [Zhang, Lee & Liu, 2026]
- Reinforcement learning in Coq (value/policy iteration convergence)
  [Vajjha et al., CPP 2021]
- VC dimension and PAC learning: work in progress

Why formalize?

- Exposes hidden assumptions
- Clarifies dependencies between lemmas
- Catches subtle errors in published proofs

From Proving to Understanding

AI can now prove IMO-level theorems.

From Proving to Understanding

AI can now prove IMO-level theorems.

But a proof that *compiles* is not the same as a proof we can *learn from*.

From Proving to Understanding

AI can now prove IMO-level theorems.

But a proof that *compiles* is not the same as a proof we can *learn from*.

New challenge: How do we make formal proofs accessible to humans?

Verification $\neq$ Understanding

Two worlds:

- **Paper proofs:** readable, but may have gaps

Verification $\neq$ Understanding

Two worlds:

- **Paper proofs:** readable, but may have gaps
- **Formal proofs:** correct, but hard to follow

Verification $\neq$ Understanding

Two worlds:

- **Paper proofs:** readable, but may have gaps
- **Formal proofs:** correct, but hard to follow

A proof is not finished when it compiles.

Proof Presentation: Another Experimental Part

Paper proofs can also be hard to read.

Proof Presentation: Another Experimental Part

Paper proofs can also be hard to read.

Can we do better than both?

Proof Presentation: Another Experimental Part

Paper proofs can also be hard to read.

Can we do better than both?

Goal: Machine-verified correctness + human-readable explanations.

Proof Presentation: Another Experimental Part

Paper proofs can also be hard to read.

Can we do better than both?

Goal: Machine-verified correctness + human-readable explanations.

Presentation forms:

- Annotated proof pages

Proof Presentation: Another Experimental Part

Paper proofs can also be hard to read.

Can we do better than both?

Goal: Machine-verified correctness + human-readable explanations.

Presentation forms:

- Annotated proof pages
- Proof blueprints (lemma dependencies)

Proof Presentation: Another Experimental Part

Paper proofs can also be hard to read.

Can we do better than both?

Goal: Machine-verified correctness + human-readable explanations.

Presentation forms:

- Annotated proof pages
- Proof blueprints (lemma dependencies)
- Visualizations and trace tables

Three Strands of the Course

1. Learning Theory

- Definitions
- Theorems
- Calculations

Three Strands of the Course

1. Learning Theory

- Definitions
- Theorems
- Calculations

2. Formal Proof

- Lean 4
- AI-assisted
- Verification

Three Strands of the Course

1. Learning Theory

- Definitions
- Theorems
- Calculations

2. Formal Proof

- Lean 4
- AI-assisted
- Verification

3. Presentation

- Human-readable
- Visualizations
- Communication

Three Strands of the Course

1. Learning Theory

- Definitions
- Theorems
- Calculations

2. Formal Proof

- Lean 4
- AI-assisted
- Verification

3. Presentation

- Human-readable
- Visualizations
- Communication

Complete artifact = Verified proof + Clear explanation

Three Strands of the Course

1. Learning Theory
- Definitions
- Theorems
- Calculations

2. Formal Proof
- Lean 4
- AI-assisted
- Verification

3. Presentation
- Human-readable
- Visualizations
- Communication

Complete artifact = Verified proof + Clear explanation

AI assistance opens new possibilities for proof and presentation.

Logistics

How this course runs

Weekly Schedule

Wednesdays, 5:00 – 7:50 PM

Lecture		Lecture		Workshop
5:00 – 5:50	break	6:00 – 6:50	break	7:00 – 7:50

Weekly Schedule

Wednesdays, 5:00 – 7:50 PM

Lecture		Lecture		Workshop
5:00 – 5:50	break	6:00 – 6:50	break	7:00 – 7:50

Workshop: Student presentations and feedback

Quarter-Long Project

Milestones:

1. Choose a learning theory result to formalize

Quarter-Long Project

Milestones:

1. Choose a learning theory result to formalize
2. Thorough understanding of its paper proof

Quarter-Long Project

Milestones:

1. Choose a learning theory result to formalize
2. Thorough understanding of its paper proof
3. Formalize in Lean

Quarter-Long Project

Milestones:

1. Choose a learning theory result to formalize
2. Thorough understanding of its paper proof
3. Formalize in Lean
4. Present and revise

Quarter-Long Project

Milestones:

1. Choose a learning theory result to formalize
2. Thorough understanding of its paper proof
3. Formalize in Lean
4. Present and revise
5. Final deliverable: Lean proof + proof presentation

Quarter-Long Project

Milestones:

1. Choose a learning theory result to formalize
2. Thorough understanding of its paper proof
3. Formalize in Lean
4. Present and revise
5. Final deliverable: Lean proof + proof presentation

Weekly assignments: Theory problems + brief project progress report

Project Output

1. **Formalized results** — compiling Lean proofs

Project Output

1. **Formalized results** — compiling Lean proofs
2. **Presentation** — your design; format is up to you

Project Output

1. **Formalized results** — compiling Lean proofs
2. **Presentation** — your design; format is up to you
3. **AI workflow** — `CLAUDE.md`, `AGENTS.md`, skills, and other AI artifacts you develop along the way

Project Repository

Each team sets up a Git repo to track their work.

Repo should contain:

- Lean project files
- Presentation materials
- AI workflow artifacts (`CLAUDE.md`, `AGENTS.md`, skills, etc.)
- README

Project Repository

Each team sets up a Git repo to track their work.

Repo should contain:

- Lean project files
- Presentation materials
- AI workflow artifacts (`CLAUDE.md`, `AGENTS.md`, skills, etc.)
- README

I will maintain a **course repo** with a list of all project repos.

Grading

Component	Weight
Weekly Assignments	30%
Quarter-Long Project	40%
Participation	30%

Grading

Component	Weight
Weekly Assignments	30%
Quarter-Long Project	40%
Participation	30%

Participation includes:

- Attending and engaging in lecture
- Workshop presentations
- Giving feedback to peers

AI Usage

AI use is **encouraged** — for both assignments and the project.

AI Usage

AI use is **encouraged** — for both assignments and the project.

Use AI for learning the material, exploring proof structure, checking candidate steps, writing and debugging Lean code, improving explanation and presentation.

AI Usage

AI use is **encouraged** — for both assignments and the project.

Use AI for learning the material, exploring proof structure, checking candidate steps, writing and debugging Lean code, improving explanation and presentation.

Develop your AI workflow throughout the quarter — refine your `CLAUDE.md`, `AGENTS.md`, and skills as you learn what works.

AI Usage

AI use is **encouraged** — for both assignments and the project.

Use AI for learning the material, exploring proof structure, checking candidate steps, writing and debugging Lean code, improving explanation and presentation.

Develop your AI workflow throughout the quarter — refine your `CLAUDE.md`, `AGENTS.md`, and skills as you learn what works.

Verification responsibility is yours — stand behind every proof step, formalization, and explanation you submit.

AI Usage

AI use is **encouraged** — for both assignments and the project.

Use AI for learning the material, exploring proof structure, checking candidate steps, writing and debugging Lean code, improving explanation and presentation.

Develop your AI workflow throughout the quarter — refine your `CLAUDE.md`, `AGENTS.md`, and skills as you learn what works.

Verification responsibility is yours — stand behind every proof step, formalization, and explanation you submit.

AI is a collaborator, not an oracle.

Week 1's Topic

The first model of the course

The Basic Prediction Setup

Instance space $\mathcal{X}$: set of possible inputs

The Basic Prediction Setup

Instance space $\mathcal{X}$: set of possible inputs
Label space $\mathcal{Y} = \{0, 1\}$: binary labels

The Basic Prediction Setup

Instance space $\mathcal{X}$: set of possible inputs
Label space $\mathcal{Y} = \{0, 1\}$: binary labels
Hypothesis $h : \mathcal{X} \to \mathcal{Y}$: a prediction rule

The Basic Prediction Setup

Instance space $\mathcal{X}$: set of possible inputs
Label space $\mathcal{Y} = \{0, 1\}$: binary labels
Hypothesis $h : \mathcal{X} \to \mathcal{Y}$: a prediction rule

Running example: Threshold classifier on $\mathbb{R}$

$$h_\theta(x) = \mathbf{1}[x \geq \theta]$$

The Basic Prediction Setup

Instance space $\mathcal{X}$: set of possible inputs
Label space $\mathcal{Y} = \{0, 1\}$: binary labels
Hypothesis $h : \mathcal{X} \to \mathcal{Y}$: a prediction rule

Running example: Threshold classifier on $\mathbb{R}$

$$h_\theta(x) = \mathbf{1}[x \geq \theta]$$

- $h_5(3) = 0$ (below threshold)
- $h_5(7) = 1$ (above threshold)
- $h_5(5) = 1$ (at threshold)

Online Prediction Protocol

Round t:

1. Instance x_t arrives

Online Prediction Protocol

Round t:

1. Instance x_t arrives
2. Learner predicts $\hat{y}_t \in \{0, 1\}$

Online Prediction Protocol

Round t:

1. Instance x_t arrives
2. Learner predicts $\hat{y}_t \in \{0, 1\}$
3. True label y_t revealed

Online Prediction Protocol

Round t:

1. Instance x_t arrives
2. Learner predicts $\hat{y}_t \in \{0, 1\}$
3. True label y_t revealed
4. Repeat for $t = 1, 2, \ldots, T$

Online Prediction Protocol

Round t:

1. Instance x_t arrives
2. Learner predicts $\hat{y}_t \in \{0, 1\}$
3. True label y_t revealed
4. Repeat for $t = 1, 2, \ldots, T$

History: $H_t = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$

Online Prediction Protocol

Round t:

1. Instance x_t arrives
2. Learner predicts $\hat{y}_t \in \{0, 1\}$
3. True label y_t revealed
4. Repeat for $t = 1, 2, \ldots, T$

History: $H_t = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$

Prediction comes before label is revealed.

What is a Learner?

A **learner** (or learning rule) is a mapping:

$$A : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}$$

What is a Learner?

A **learner** (or learning rule) is a mapping:

$$A : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}$$

Given history, output a hypothesis $h_t = A((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$

Prediction: $\hat{y}_t = h_t(x_t)$

What is a Learner?

A **learner** (or learning rule) is a mapping:

$$A : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}$$

Given history, output a hypothesis $h_t = A((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$

Prediction: $\hat{y}_t = h_t(x_t)$

Examples:

- **Constant:** $h_t(x) = 1$ for all x

What is a Learner?

A **learner** (or learning rule) is a mapping:

$$A : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}$$

Given history, output a hypothesis $h_t = A((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$

Prediction: $\hat{y}_t = h_t(x_t)$

Examples:

- **Constant:** $h_t(x) = 1$ for all x
- **Memorize:** $h_t(x) =$ last label seen for x, else default

Counting Mistakes

A **mistake** occurs when $\hat{y}_t \neq y_t$.

Total mistakes: $M_T = \sum_{t=1}^{T} \mathbf{1}[\hat{y}_t \neq y_t]$

Counting Mistakes

A **mistake** occurs when $\hat{y}_t \neq y_t$.

Total mistakes: $M_T = \sum_{t=1}^{T} \mathbf{1}[\hat{y}_t \neq y_t]$

Goal: Make few mistakes.

Counting Mistakes

A **mistake** occurs when $\hat{y}_t \neq y_t$.

Total mistakes: $M_T = \sum_{t=1}^{T} \mathbf{1}[\hat{y}_t \neq y_t]$

Goal: Make few mistakes.

But bounded under what conditions?

- Against all possible label sequences?
- Only when labels follow some structure?

Counting Mistakes

A **mistake** occurs when $\hat{y}_t \neq y_t$.

Total mistakes: $M_T = \sum_{t=1}^{T} \mathbf{1}[\hat{y}_t \neq y_t]$

Goal: Make few mistakes.

But bounded under what conditions?

- Against all possible label sequences?
- Only when labels follow some structure?

This is exactly the question we address next.

No-Free-Lunch

Why learning needs assumptions

The Key Question

Is there a learner that makes few mistakes
no matter what labeling rule is used?

The Key Question

Is there a learner that makes few mistakes
no matter what labeling rule is used?

If yes: identify universal strategy, done.

The Key Question

Is there a learner that makes few mistakes
no matter what labeling rule is used?

If yes: identify universal strategy, done.

If no: we need to understand **what assumptions** make learning possible.

Theorem

For any finite $\mathcal{X}$ and any deterministic learner A, there exists:

- a function $f\colon \mathcal{X} \to \{0,1\}$, and
- a sequence $x_1, \ldots, x_n$ with $n = |\mathcal{X}|$ (all distinct)

such that A makes n mistakes on the sequence $(x_t, y_t = f(x_t))_{t=1}^{n}$.

Theorem

For any finite $\mathcal{X}$ and any deterministic learner A, there exists:

- a function $f : \mathcal{X} \to \{0, 1\}$, and
- a sequence $x_1, \ldots, x_n$ with $n = |\mathcal{X}|$ (all distinct)

such that A makes n mistakes on the sequence $(x_t, y_t = f(x_t))_{t=1}^{n}$.

In words: No learner can guarantee fewer than $|\mathcal{X}|$ mistakes in the worst case.

NFL Proof: Construction

Strategy: Build f point by point to defeat the learner.

NFL Proof: Construction

Strategy: Build f point by point to defeat the learner.

Fix any ordering $x_1, \ldots, x_n$ of $\mathcal{X}$. In round t:

1. History so far: $H_{t-1} = ((x_1, f(x_1)), \ldots, (x_{t-1}, f(x_{t-1})))$

NFL Proof: Construction

Strategy: Build f point by point to defeat the learner.

Fix any ordering $x_1, \ldots, x_n$ of $\mathcal{X}$. In round t:

1. History so far: $H_{t-1} = ((x_1, f(x_1)), \ldots, (x_{t-1}, f(x_{t-1})))$
2. Learner predicts: $\hat{y}_t = A(H_{t-1})(x_t)$

NFL Proof: Construction

Strategy: Build f point by point to defeat the learner.

Fix any ordering $x_1, \ldots, x_n$ of $\mathcal{X}$. In round t:

1. History so far: $H_{t-1} = ((x_1, f(x_1)), \ldots, (x_{t-1}, f(x_{t-1})))$
2. Learner predicts: $\hat{y}_t = A(H_{t-1})(x_t)$
3. Define: $f(x_t) := 1 - \hat{y}_t$ (opposite of prediction)

NFL Proof: Construction

Strategy: Build f point by point to defeat the learner.

Fix any ordering $x_1, \ldots, x_n$ of $\mathcal{X}$. In round t:

1. History so far: $H_{t-1} = ((x_1, f(x_1)), \ldots, (x_{t-1}, f(x_{t-1})))$
2. Learner predicts: $\hat{y}_t = A(H_{t-1})(x_t)$
3. Define: $f(x_t) := 1 - \hat{y}_t$ (opposite of prediction)

Why is f well-defined? Each x_t is new — no previous value is overwritten.

NFL Proof: Construction

Strategy: Build f point by point to defeat the learner.

Fix any ordering $x_1, \ldots, x_n$ of $\mathcal{X}$. In round t:

1. History so far: $H_{t-1} = ((x_1, f(x_1)), \ldots, (x_{t-1}, f(x_{t-1})))$
2. Learner predicts: $\hat{y}_t = A(H_{t-1})(x_t)$
3. Define: $f(x_t) := 1 - \hat{y}_t$ (opposite of prediction)

Why is f well-defined? Each x_t is new — no previous value is overwritten.

Why n mistakes? By construction, $\hat{y}_t \neq f(x_t)$ on every round.

The Lesson from NFL

Without assumptions: Any learner can be forced to make $|\mathcal{X}|$ mistakes.

The Lesson from NFL

Without assumptions: Any learner can be forced to make $|\mathcal{X}|$ mistakes.

The way out: Assume the labeling rule belongs to a known hypothesis class $\mathcal{H}$.

The Lesson from NFL

Without assumptions: Any learner can be forced to make $|\mathcal{X}|$ mistakes.

The way out: Assume the labeling rule belongs to a known hypothesis class $\mathcal{H}$.

What if we have **prior knowledge** about the labeling rule?

How Structure Helps

When assumptions rescue learning

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Assumption: There exists $\theta^* \in [0, 1]$ such that $y = \mathbf{1}[x > \theta^*]$

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Assumption: There exists $\theta^* \in [0, 1]$ such that $y = \mathbf{1}[x > \theta^*]$

Strategy: Binary search

1. Maintain interval $[a, b]$ containing θ^*

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Assumption: There exists $\theta^* \in [0, 1]$ such that $y = \mathbf{1}[x > \theta^*]$

Strategy: Binary search

1. Maintain interval $[a, b]$ containing θ^*
2. For point x: predict $\hat{y} = \mathbf{1}[x > \frac{a+b}{2}]$

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Assumption: There exists $\theta^* \in [0, 1]$ such that $y = \mathbf{1}[x > \theta^*]$

Strategy: Binary search

1. Maintain interval $[a, b]$ containing θ^*
2. For point x: predict $\hat{y} = \mathbf{1}[x > \frac{a+b}{2}]$
3. On mistake at x: update interval to exclude wrong half
   - If predicted 1 but $y = 0$: $\theta^* < x$, so $[a, b] \leftarrow [a, x]$
   - If predicted 0 but $y = 1$: $\theta^* > x$, so $[a, b] \leftarrow [x, b]$

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Assumption: There exists $\theta^* \in [0, 1]$ such that $y = \mathbf{1}[x > \theta^*]$

Strategy: Binary search

1. Maintain interval $[a, b]$ containing θ^*
2. For point x: predict $\hat{y} = \mathbf{1}[x > \frac{a+b}{2}]$
3. On mistake at x: update interval to exclude wrong half
   - If predicted 1 but $y = 0$: $\theta^* < x$, so $[a, b] \leftarrow [a, x]$
   - If predicted 0 but $y = 1$: $\theta^* > x$, so $[a, b] \leftarrow [x, b]$

After k mistakes: interval has length $(b - a)/2^k$

Structure Helps: Threshold Example

Setup: $\mathcal{X} = [0, 1]$, predict labels $y \in \{0, 1\}$ for points $x \in \mathcal{X}$

Assumption: There exists $\theta^* \in [0, 1]$ such that $y = \mathbf{1}[x > \theta^*]$

Strategy: Binary search

1. Maintain interval $[a, b]$ containing θ^*
2. For point x: predict $\hat{y} = \mathbf{1}[x > \frac{a+b}{2}]$
3. On mistake at x: update interval to exclude wrong half
   - If predicted 1 but $y = 0$: $\theta^* < x$, so $[a, b] \leftarrow [a, x]$
   - If predicted 0 but $y = 1$: $\theta^* > x$, so $[a, b] \leftarrow [x, b]$

After k mistakes: interval has length $(b - a)/2^k$

At most $\lceil \log_2(1/\varepsilon) \rceil$ mistakes for precision ε

Hypothesis Class = Prior Knowledge

A **hypothesis class** $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is a set of candidate labeling functions.

Hypothesis Class = Prior Knowledge

A **hypothesis class** $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is a set of candidate labeling functions.

Assumption (realizability): $\exists\, h^* \in \mathcal{H}$ such that $y_t = h^*(x_t)$ for all t.

- Learner knows $\mathcal{H}$, but not which $h^* \in \mathcal{H}$

Hypothesis Class = Prior Knowledge

A **hypothesis class** $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is a set of candidate labeling functions.

Assumption (realizability): $\exists\, h^* \in \mathcal{H}$ such that $y_t = h^*(x_t)$ for all t.

- Learner knows $\mathcal{H}$, but not which $h^* \in \mathcal{H}$

Interpretation: $\mathcal{H}$ encodes our *prior knowledge* about the problem

- Smaller $\mathcal{H}$ = stronger assumption = easier to learn
- Larger $\mathcal{H}$ = weaker assumption = harder to learn

Generalizing: Finite Hypothesis Classes

Setup: Hypothesis class $\mathcal{H} \subseteq \{f : \mathcal{X} \to \{0, 1\}\}$ with $|\mathcal{H}| = N$

Generalizing: Finite Hypothesis Classes

Setup: Hypothesis class $\mathcal{H} \subseteq \{f : \mathcal{X} \to \{0, 1\}\}$ with $|\mathcal{H}| = N$

Assumption: There exists $h^* \in \mathcal{H}$ generating the labels

Generalizing: Finite Hypothesis Classes

Setup: Hypothesis class $\mathcal{H} \subseteq \{f : \mathcal{X} \to \{0, 1\}\}$ with $|\mathcal{H}| = N$

Assumption: There exists $h^* \in \mathcal{H}$ generating the labels

Question: How many mistakes are needed to identify h^*?

The Halving Algorithm

Idea: Maintain *version space* $V_t \subseteq \mathcal{H}$ = hypotheses consistent with history

The Halving Algorithm

Idea: Maintain *version space* $V_t \subseteq \mathcal{H}$ = hypotheses consistent with history

Algorithm:

1. Initialize $V_1 = \mathcal{H}$

The Halving Algorithm

Idea: Maintain *version space* $V_t \subseteq \mathcal{H}$ = hypotheses consistent with history

Algorithm:

1. Initialize $V_1 = \mathcal{H}$
2. At round t: predict $\hat{y}_t$ = majority vote of $\{h(x_t) : h \in V_t\}$

The Halving Algorithm

Idea: Maintain *version space* $V_t \subseteq \mathcal{H}$ = hypotheses consistent with history

Algorithm:

1. Initialize $V_1 = \mathcal{H}$
2. At round t: predict $\hat{y}_t$ = majority vote of $\{h(x_t) : h \in V_t\}$
3. Update: $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

The Halving Algorithm

Idea: Maintain *version space* $V_t \subseteq \mathcal{H}$ = hypotheses consistent with history

Algorithm:

1. Initialize $V_1 = \mathcal{H}$
2. At round t: predict $\hat{y}_t$ = majority vote of $\{h(x_t) : h \in V_t\}$
3. Update: $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Key observation: True h^* is never eliminated (always in V_t)

Halving Algorithm: Analysis

Claim: Each mistake eliminates at least half of V_t

Halving Algorithm: Analysis

Claim: Each mistake eliminates at least half of V_t

Proof:

- If $\hat{y}_t \neq y_t$, then majority of V_t predicted wrong
- All wrong hypotheses are eliminated
- So $|V_{t+1}| \leq |V_t|/2$

Halving Algorithm: Analysis

Claim: Each mistake eliminates at least half of V_t

Proof:

- If $\hat{y}_t \neq y_t$, then majority of V_t predicted wrong
- All wrong hypotheses are eliminated
- So $|V_{t+1}| \leq |V_t|/2$

Mistake bound:

- Start with $|V_1| = |\mathcal{H}|$
- After k mistakes: $|V| \leq |\mathcal{H}|/2^k$
- Since $|V| \geq 1$ always: $k \leq \log_2 |\mathcal{H}|$

Halving Algorithm: Analysis

Claim: Each mistake eliminates at least half of V_t

Proof:

- If $\hat{y}_t \neq y_t$, then majority of V_t predicted wrong
- All wrong hypotheses are eliminated
- So $|V_{t+1}| \leq |V_t|/2$

Mistake bound:

- Start with $|V_1| = |\mathcal{H}|$
- After k mistakes: $|V| \leq |\mathcal{H}|/2^k$
- Since $|V| \geq 1$ always: $k \leq \log_2 |\mathcal{H}|$

Halving makes at most $\lfloor \log_2 |\mathcal{H}| \rfloor$ mistakes

Complexity of Hypothesis Classes

Observation: Halving's mistake bound depends on $\log_2 |\mathcal{H}|$

Complexity of Hypothesis Classes

Observation: Halving's mistake bound depends on $\log_2 |\mathcal{H}|$

Interpretation: The "size" of $\mathcal{H}$ measures learning difficulty

Complexity of Hypothesis Classes

Observation: Halving's mistake bound depends on $\log_2 |\mathcal{H}|$

Interpretation: The "size" of $\mathcal{H}$ measures learning difficulty

But what about infinite hypothesis classes?

- Threshold functions: $|\mathcal{H}| = \infty$, yet $O(\log(1/\varepsilon))$ mistakes

Complexity of Hypothesis Classes

Observation: Halving's mistake bound depends on $\log_2 |\mathcal{H}|$

Interpretation: The "size" of $\mathcal{H}$ measures learning difficulty

But what about infinite hypothesis classes?

- Threshold functions: $|\mathcal{H}| = \infty$, yet $O(\log(1/\varepsilon))$ mistakes

Key insight: Need a notion of "complexity" beyond cardinality

Complexity of Hypothesis Classes

Observation: Halving's mistake bound depends on $\log_2 |\mathcal{H}|$

Interpretation: The "size" of $\mathcal{H}$ measures learning difficulty

But what about infinite hypothesis classes?

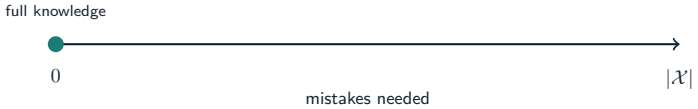
- Threshold functions: $|\mathcal{H}| = \infty$, yet $O(\log(1/\varepsilon))$ mistakes

Key insight: Need a notion of "complexity" beyond cardinality

Coming up: VC dimension, Littlestone dimension, ...

The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$



Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H}$ $\Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

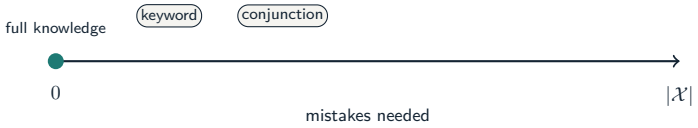
The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$

Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H}$ $\Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

The Tradeoff: Expressiveness vs. Mistakes

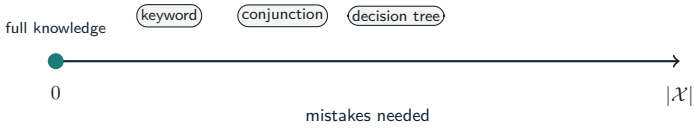
$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$



Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H} \Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

The Tradeoff: Expressiveness vs. Mistakes

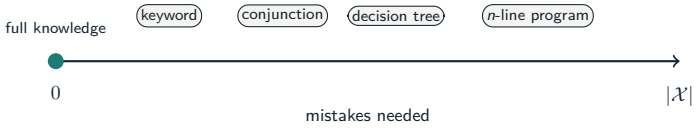
$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$



Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H}$ $\Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

The Tradeoff: Expressiveness vs. Mistakes

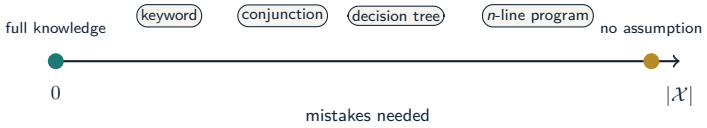
$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$



Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H}$ $\Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$

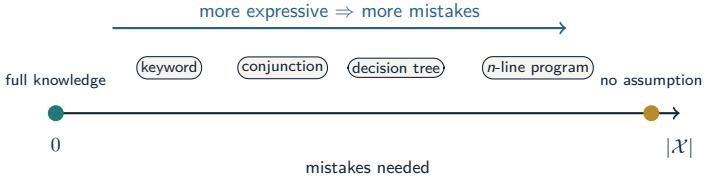


Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H} \Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)

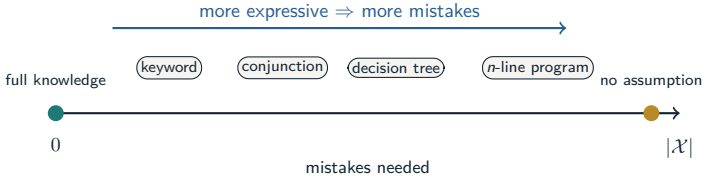
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$



The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)

keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$

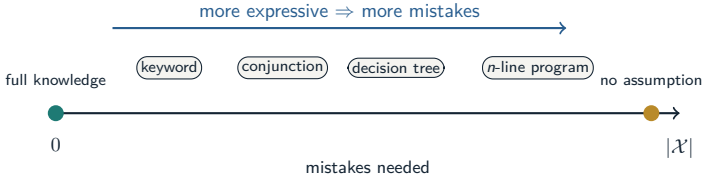


Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes

The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)

keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$

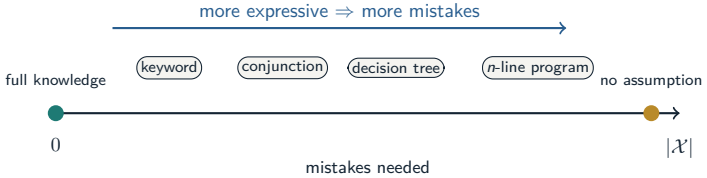


Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes

Smaller $\mathcal{H}$ $\Rightarrow$ fewer mistakes needed

The Tradeoff: Expressiveness vs. Mistakes

$\mathcal{X} = 2^D$ (emails as subsets of dictionary D), $\mathcal{Y} = \{0, 1\}$ (spam/not spam)
keyword $=$ "free" $\in x$; conjunction $=$ "free" $\wedge$ "click" $\wedge$ "win" $\in x$



Recall: Halving achieves $\leq \log_2 |\mathcal{H}|$ mistakes
Smaller $\mathcal{H}$ $\Rightarrow$ fewer mistakes needed
So why not pick the smallest $\mathcal{H}$ that still contains the truth?

Why Not Use $\mathcal{H} = \{\textbf{short programs}\}$?

Idea: Let $\mathcal{H}$ = all 100-line Python programs

Why Not Use $\mathcal{H} = \{\textbf{short programs}\}$?

Idea: Let $\mathcal{H}$ = all 100-line Python programs

Mistake bound: $\leq \log_2 128^{100 \cdot 80} \approx 56{,}000$ mistakes

Why Not Use $\mathcal{H} = \{\textbf{short programs}\}$?

Idea: Let $\mathcal{H}$ = all 100-line Python programs

Mistake bound: $\leq \log_2 128^{100 \cdot 80} \approx 56{,}000$ mistakes

Problem: Running Halving requires checking each program on each example

- That's not even computable! (Halting problem)

Why Not Use $\mathcal{H} = \{\textbf{short programs}\}$?

Idea: Let $\mathcal{H}$ = all 100-line Python programs

Mistake bound: $\leq \log_2 128^{100 \cdot 80} \approx 56{,}000$ mistakes

Problem: Running Halving requires checking each program on each example

- That's not even computable! (Halting problem)

Even for "nice" classes like decision trees:

- Mistake bound: $O(k \log |D|)$ where k = tree size, D = features
- Runtime of Halving: $O(|D|^k \cdot n \cdot k)$ — exponential in k!

Why Not Use $\mathcal{H} = \{\textbf{short programs}\}$?

Idea: Let $\mathcal{H}$ = all 100-line Python programs

Mistake bound: $\leq \log_2 128^{100 \cdot 80} \approx 56{,}000$ mistakes

Problem: Running Halving requires checking each program on each example

- That's not even computable! (Halting problem)

Even for "nice" classes like decision trees:

- Mistake bound: $O(k \log |D|)$ where k = tree size, D = features
- Runtime of Halving: $O(|D|^k \cdot n \cdot k)$ — exponential in k!

We want hypothesis classes that are:

1. Expressive (capture interesting concepts with low $\log |\mathcal{H}|$)
2. Computationally efficiently learnable

An Important Class: Linear Predictors

Setup: Choose features $\phi_1(x), \phi_2(x), \ldots, \phi_d(x)$

$$\phi : \mathcal{X} \to \mathbb{R}^d, \quad \phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_d(x))$$

An Important Class: Linear Predictors

Setup: Choose features $\phi_1(x), \phi_2(x), \ldots, \phi_d(x)$

$$\phi : \mathcal{X} \to \mathbb{R}^d, \quad \phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_d(x))$$

Linear classifiers (halfspaces):

$$\mathcal{H} = \left\{ h_{w,\theta}(x) = \mathbf{1}[\langle w, \phi(x) \rangle > \theta] \;\middle|\; w \in \mathbb{R}^d, \theta \in \mathbb{R} \right\}$$

An Important Class: Linear Predictors

Setup: Choose features $\phi_1(x), \phi_2(x), \ldots, \phi_d(x)$

$$\phi : \mathcal{X} \to \mathbb{R}^d, \quad \phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_d(x))$$

Linear classifiers (halfspaces):

$$\mathcal{H} = \left\{ h_{w,\theta}(x) = \mathbf{1}[\langle w, \phi(x) \rangle > \theta] \;\middle|\; w \in \mathbb{R}^d, \theta \in \mathbb{R} \right\}$$

Interpretation:

- The feature map $\phi(x)$ encodes our prior knowledge
- The linear separator w is what we learn

Why Linear Predictors?

Highly expressive with the right features:

Why Linear Predictors?

Highly expressive with the right features:

- $\phi_i(x) = [\text{word } i \in x]$ (bag of words)

Why Linear Predictors?

Highly expressive with the right features:

- $\phi_i(x) = [\text{word } i \in x]$ (bag of words)
- Can encode **conjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 2.5]$

Why Linear Predictors?

Highly expressive with the right features:

- $\phi_i(x) = [\text{word } i \in x]$ (bag of words)
- Can encode **conjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 2.5]$
- Can encode **disjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 0.5]$

Why Linear Predictors?

Highly expressive with the right features:

- $\phi_i(x) = [\text{word } i \in x]$ (bag of words)
- Can encode **conjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 2.5]$
- Can encode **disjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 0.5]$
- Can encode **weighted votes**: $[3\phi_{\text{free}} + 2\phi_{\text{click}} - \phi_{\text{meeting}} > 1]$

Why Linear Predictors?

Highly expressive with the right features:

- $\phi_i(x) = [\text{word } i \in x]$ (bag of words)
- Can encode **conjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 2.5]$
- Can encode **disjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 0.5]$
- Can encode **weighted votes**: $[3\phi_{\text{free}} + 2\phi_{\text{click}} - \phi_{\text{meeting}} > 1]$

Key question: Can we (online) learn linear predictors?

Why Linear Predictors?

Highly expressive with the right features:

- $\phi_i(x) = [\text{word } i \in x]$ (bag of words)
- Can encode **conjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 2.5]$
- Can encode **disjunctions**: $[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 0.5]$
- Can encode **weighted votes**: $[3\phi_{\text{free}} + 2\phi_{\text{click}} - \phi_{\text{meeting}} > 1]$

Key question: Can we (online) learn linear predictors?

Problem: $|\mathcal{H}| = \infty$ (uncountably many choices of w, θ)

The Challenge: Infinite Hypothesis Class

Example: Threshold functions in 1D

$$\mathcal{H} = \left\{ h_\theta(x) = \mathbf{1}[x \leq \theta] \;\middle|\; \theta \in \mathbb{R} \right\}$$

The Challenge: Infinite Hypothesis Class

Example: Threshold functions in 1D

$$\mathcal{H} = \left\{ h_\theta(x) = \mathbf{1}[x \leq \theta] \;\middle|\; \theta \in \mathbb{R} \right\}$$

Theorem: For *any* learning rule A, there exists a sequence realized by $\mathcal{H}$ on which A makes a mistake on *every* round.

The Challenge: Infinite Hypothesis Class

Example: Threshold functions in 1D

$$\mathcal{H} = \left\{ h_\theta(x) = \mathbf{1}[x \le \theta] \;\middle|\; \theta \in \mathbb{R} \right\}$$

Theorem: For *any* learning rule A, there exists a sequence realized by $\mathcal{H}$ on which A makes a mistake on *every* round.

Proof idea: Binary search construction

- $x_1 = 0.5, \quad y_t = -\hat{y}_t$ (always wrong!)
- $x_{t+1} = x_t + y_t \cdot 2^{-(t+1)}$
- Realized by $\theta = 0.5 + \sum_t y_t \cdot 2^{-(t+1)}$

The Challenge: Infinite Hypothesis Class

Example: Threshold functions in 1D

$$\mathcal{H} = \left\{ h_\theta(x) = \mathbf{1}[x \le \theta] \;\middle|\; \theta \in \mathbb{R} \right\}$$

Theorem: For *any* learning rule A, there exists a sequence realized by $\mathcal{H}$ on which A makes a mistake on *every* round.

Proof idea: Binary search construction

- $x_1 = 0.5, \quad y_t = -\hat{y}_t$ (always wrong!)
- $x_{t+1} = x_t + y_t \cdot 2^{-(t+1)}$
- Realized by $\theta = 0.5 + \sum_t y_t \cdot 2^{-(t+1)}$

Key observation: Adversary exploits *infinite precision* — points get arbitrarily close to the threshold.

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

Restrict $\theta \in G_r = \{-1, -\frac{r-1}{r}, \ldots, -\frac{1}{r}, 0, \frac{1}{r}, \ldots, 1\}$

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

Restrict $\theta \in G_r = \{-1, -\frac{r-1}{r}, \ldots, -\frac{1}{r}, 0, \frac{1}{r}, \ldots, 1\}$

- 1D thresholds: $|\mathcal{H}| = 2r + 1 \quad \Rightarrow \quad \log_2 |\mathcal{H}| = O(\log r)$ mistakes

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

Restrict $\theta \in G_r = \{-1, -\frac{r-1}{r}, \ldots, -\frac{1}{r}, 0, \frac{1}{r}, \ldots, 1\}$

- 1D thresholds: $|\mathcal{H}| = 2r + 1 \quad \Rightarrow \quad \log_2 |\mathcal{H}| = O(\log r)$ mistakes
- d-dimensional: $w \in G_r^d \quad \Rightarrow \quad \log |\mathcal{H}| = O(d \log r) = O(d \cdot \#\text{bits})$

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

Restrict $\theta \in G_r = \{-1, -\frac{r-1}{r}, \ldots, -\frac{1}{r}, 0, \frac{1}{r}, \ldots, 1\}$

- 1D thresholds: $|\mathcal{H}| = 2r + 1 \quad \Rightarrow \quad \log_2 |\mathcal{H}| = O(\log r)$ mistakes
- d-dimensional: $w \in G_r^d \quad \Rightarrow \quad \log |\mathcal{H}| = O(d \log r) = O(d \cdot \#\text{bits})$

This matches our earlier framework! Finite $\mathcal{H} \Rightarrow$ Halving works.

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

Restrict $\theta \in G_r = \{-1, -\frac{r-1}{r}, \ldots, -\frac{1}{r}, 0, \frac{1}{r}, \ldots, 1\}$

- 1D thresholds: $|\mathcal{H}| = 2r + 1 \quad \Rightarrow \quad \log_2 |\mathcal{H}| = O(\log r)$ mistakes
- d-dimensional: $w \in G_r^d \quad \Rightarrow \quad \log |\mathcal{H}| = O(d \log r) = O(d \cdot \#\text{bits})$

This matches our earlier framework! Finite $\mathcal{H} \Rightarrow$ Halving works.

But: Runtime of Halving is $\Omega(|G_r|^d) = \Omega(r^d)$ — **exponential in d!**

From Infinite to Finite: Discretization

Idea: Make $\mathcal{H}$ finite by limiting precision

Restrict $\theta \in G_r = \{-1, -\frac{r-1}{r}, \ldots, -\frac{1}{r}, 0, \frac{1}{r}, \ldots, 1\}$

- 1D thresholds: $|\mathcal{H}| = 2r + 1 \quad \Rightarrow \quad \log_2 |\mathcal{H}| = O(\log r)$ mistakes
- d-dimensional: $w \in G_r^d \quad \Rightarrow \quad \log |\mathcal{H}| = O(d \log r) = O(d \cdot \#\text{bits})$

This matches our earlier framework! Finite $\mathcal{H} \Rightarrow$ Halving works.

But: Runtime of Halving is $\Omega(|G_r|^d) = \Omega(r^d)$ — **exponential in d!**

Question: Can we avoid both infinite mistakes AND exponential runtime?

So We Can't Learn Linear Predictors?

Summary of approaches:

So We Can't Learn Linear Predictors?

Summary of approaches:

Approach 1: Continuous $\mathcal{H}$ (no discretization)

- Adversary exploits infinite precision $\Rightarrow$ infinite mistakes

So We Can't Learn Linear Predictors?

Summary of approaches:

Approach 1: Continuous $\mathcal{H}$ (no discretization)

- Adversary exploits infinite precision $\Rightarrow$ infinite mistakes

Approach 2: Discretize with resolution r

- $O(d \log r)$ mistakes with Halving
- But runtime $\Omega(r^d) \Rightarrow$ exponential

So We Can't Learn Linear Predictors?

Summary of approaches:

Approach 1: Continuous $\mathcal{H}$ (no discretization)

- Adversary exploits infinite precision $\Rightarrow$ infinite mistakes

Approach 2: Discretize with resolution r

- $O(d \log r)$ mistakes with Halving
- But runtime $\Omega(r^d) \Rightarrow$ exponential

Approach 3: The Perceptron

- Works with continuous $\mathcal{H}$ directly
- $O(1/\gamma^2)$ mistakes where $\gamma =$ margin
- Runtime $O(d)$ per round $\Rightarrow$ **efficient!**

So We Can't Learn Linear Predictors?

Summary of approaches:

Approach 1: Continuous $\mathcal{H}$ (no discretization)

- Adversary exploits infinite precision $\Rightarrow$ infinite mistakes

Approach 2: Discretize with resolution r

- $O(d \log r)$ mistakes with Halving
- But runtime $\Omega(r^d) \Rightarrow$ exponential

Approach 3: The Perceptron

- Works with continuous $\mathcal{H}$ directly
- $O(1/\gamma^2)$ mistakes where $\gamma =$ margin
- Runtime $O(d)$ per round $\Rightarrow$ **efficient!**

The key: Margin γ is the "right" complexity measure for linear predictors

The Perceptron Algorithm (Rosenblatt 1958)

Algorithm:

1. Initialize $w_1 = 0$
2. At round t:
   - Receive x_t, predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t) \rangle)$
   - Receive true label $y_t \in \{-1, +1\}$
   - If mistake ($y_t \neq \hat{y}_t$): $w_{t+1} \leftarrow w_t + y_t \phi(x_t)$
   - Else: $w_{t+1} \leftarrow w_t$

The Perceptron Algorithm (Rosenblatt 1958)

Algorithm:

1. Initialize $w_1 = 0$
2. At round t:
   - Receive x_t, predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t) \rangle)$
   - Receive true label $y_t \in \{-1, +1\}$
   - If mistake ($y_t \neq \hat{y}_t$): $w_{t+1} \leftarrow w_t + y_t \phi(x_t)$
   - Else: $w_{t+1} \leftarrow w_t$

Key insight: Update moves w toward correct classification

The Perceptron Algorithm (Rosenblatt 1958)

Algorithm:

1. Initialize $w_1 = 0$
2. At round t:
   - Receive x_t, predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t) \rangle)$
   - Receive true label $y_t \in \{-1, +1\}$
   - If mistake ($y_t \neq \hat{y}_t$): $w_{t+1} \leftarrow w_t + y_t \phi(x_t)$
   - Else: $w_{t+1} \leftarrow w_t$

Key insight: Update moves w toward correct classification

No need to enumerate hypotheses — just update weights!

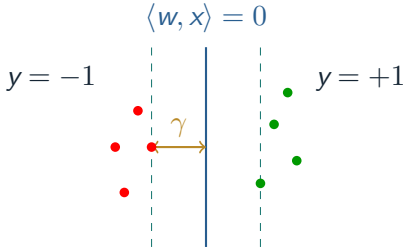
What is Margin? Geometric Intuition

Margin: $\gamma = \min_t \ y_t \langle w^*, \phi(x_t) \rangle$ where $\|w^*\| = 1$.

What is Margin? Geometric Intuition

Margin: $\gamma = \min_t \; y_t \langle w^*, \phi(x_t) \rangle$ where $\|w^*\| = 1$.

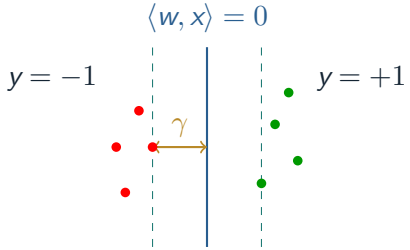
Geometrically: minimum distance from any point to the decision boundary.



What is Margin? Geometric Intuition

Margin: $\gamma = \min_t \; y_t \langle w^*, \phi(x_t) \rangle$ where $\|w^*\| = 1$.

Geometrically: minimum distance from any point to the decision boundary.

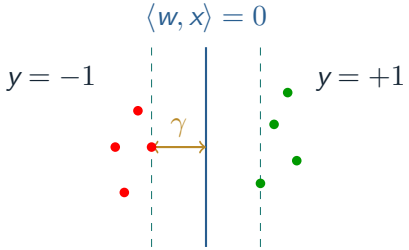


- **Large** γ = well-separated = "easy" **Small** γ = nearly touching = "hard"

What is Margin? Geometric Intuition

Margin: $\gamma = \min_t \; y_t \langle w^*, \phi(x_t) \rangle$ where $\|w^*\| = 1$.

Geometrically: minimum distance from any point to the decision boundary.



- **Large** γ = well-separated = "easy" **Small** γ = nearly touching = "hard"

The threshold counterexample has $\gamma \to 0$ — adversary pushes points arbitrarily close to boundary!

Perceptron: Mistake Bound

Theorem (Novikoff 1962): If data is linearly separable with margin γ, and $\|\phi(x_t)\| \leq R$ for all t, then Perceptron makes at most

$$M \leq \frac{R^2}{\gamma^2} \quad \text{mistakes}$$

Perceptron: Mistake Bound

Theorem (Novikoff 1962): If data is linearly separable with margin γ, and $\|\phi(x_t)\| \leq R$ for all t, then Perceptron makes at most

$$M \leq \frac{R^2}{\gamma^2} \quad \text{mistakes}$$

Why does this make sense?

- Large margin $\gamma \Rightarrow$ few mistakes (easy problem)
- Small margin $\gamma \Rightarrow$ many mistakes (hard problem)
- $\gamma \to 0 \Rightarrow$ unbounded mistakes (threshold counterexample!)

Perceptron: Mistake Bound

Theorem (Novikoff 1962): If data is linearly separable with margin γ, and $\|\phi(x_t)\| \leq R$ for all t, then Perceptron makes at most

$$M \leq \frac{R^2}{\gamma^2} \quad \text{mistakes}$$

Why does this make sense?

- Large margin $\gamma \Rightarrow$ few mistakes (easy problem)
- Small margin $\gamma \Rightarrow$ many mistakes (hard problem)
- $\gamma \to 0 \Rightarrow$ unbounded mistakes (threshold counterexample!)

Remarkable: Bound is **independent of dimension** d!

Perceptron: Mistake Bound

Theorem (Novikoff 1962): If data is linearly separable with margin γ, and $\|\phi(x_t)\| \leq R$ for all t, then Perceptron makes at most

$$M \leq \frac{R^2}{\gamma^2} \quad \text{mistakes}$$

Why does this make sense?

- Large margin $\gamma \Rightarrow$ few mistakes (easy problem)
- Small margin $\gamma \Rightarrow$ many mistakes (hard problem)
- $\gamma \to 0 \Rightarrow$ unbounded mistakes (threshold counterexample!)

Remarkable: Bound is **independent of dimension** d!

A problem in $d = 1{,}000{,}000$ dimensions with large margin is easier than a problem in $d = 2$ with tiny margin.

Perceptron Analysis: Claim 1

Recall: $w_1 = 0$; predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle)$; on mistake, $w_{t+1} \leftarrow w_t + y_t\phi(x_t)$.
Assume: $\|\phi(x)\| \leq R$ and $\exists\, w^*$ with $\|w^*\| = 1$ and $y_t\langle w^*, \phi(x_t)\rangle \geq \gamma$ for all t.

Perceptron Analysis: Claim 1

Recall: $w_1 = 0$; predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle)$; on mistake, $w_{t+1} \leftarrow w_t + y_t\phi(x_t)$.
Assume: $\|\phi(x)\| \le R$ and $\exists\, w^*$ with $\|w^*\| = 1$ and $y_t\langle w^*, \phi(x_t)\rangle \ge \gamma$ for all t.

Claim 1: $\langle w^*, w_{t+1}\rangle \ge \gamma M_t$ (alignment grows with mistakes)

Perceptron Analysis: Claim 1

Recall: $w_1 = 0$; predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle)$; on mistake, $w_{t+1} \leftarrow w_t + y_t\phi(x_t)$.
Assume: $\|\phi(x)\| \le R$ and $\exists\, w^*$ with $\|w^*\| = 1$ and $y_t\langle w^*, \phi(x_t)\rangle \ge \gamma$ for all t.

Claim 1: $\langle w^*, w_{t+1}\rangle \ge \gamma M_t$ (alignment grows with mistakes)

Proof: Since $w_1 = 0$ and updates only happen on mistakes:

$$w_{t+1} = \sum_{i \le t:\ \text{mistake}} y_i\phi(x_i)$$

Perceptron Analysis: Claim 1

Recall: $w_1 = 0$; predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle)$; on mistake, $w_{t+1} \leftarrow w_t + y_t\phi(x_t)$.
Assume: $\|\phi(x)\| \le R$ and $\exists\, w^*$ with $\|w^*\| = 1$ and $y_t\langle w^*, \phi(x_t)\rangle \ge \gamma$ for all t.

Claim 1: $\langle w^*, w_{t+1}\rangle \ge \gamma M_t$ (alignment grows with mistakes)

Proof: Since $w_1 = 0$ and updates only happen on mistakes:

$$w_{t+1} = \sum_{i \le t:\ \text{mistake}} y_i\phi(x_i)$$

Taking the inner product with w^*:

$$\langle w^*, w_{t+1}\rangle = \sum_{i:\ \text{mistake}} y_i\langle w^*, \phi(x_i)\rangle$$

Perceptron Analysis: Claim 1

Recall: $w_1 = 0$; predict $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle)$; on mistake, $w_{t+1} \leftarrow w_t + y_t\phi(x_t)$.
Assume: $\|\phi(x)\| \le R$ and $\exists\, w^*$ with $\|w^*\| = 1$ and $y_t\langle w^*, \phi(x_t)\rangle \ge \gamma$ for all t.

Claim 1: $\langle w^*, w_{t+1}\rangle \ge \gamma M_t$ (alignment grows with mistakes)

Proof: Since $w_1 = 0$ and updates only happen on mistakes:

$$w_{t+1} = \sum_{i \le t:\ \text{mistake}} y_i\phi(x_i)$$

Taking the inner product with w^*:

$$\langle w^*, w_{t+1}\rangle = \sum_{i:\ \text{mistake}} y_i\langle w^*, \phi(x_i)\rangle$$

The margin assumption gives $y_i\langle w^*, \phi(x_i)\rangle \ge \gamma$ for every i, so:

$$\langle w^*, w_{t+1}\rangle \ge \gamma M_t$$

Perceptron Analysis: Claim 2

Claim 2: $\|w_{t+1}\|^2 \leq R^2 M_t$ (norm grows slowly)

Perceptron Analysis: Claim 2

Claim 2: $\|w_{t+1}\|^2 \leq R^2 M_t$ (norm grows slowly)

Proof: On a mistake, $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle) \neq y_t$, so $y_t \langle w_t, \phi(x_t)\rangle \leq 0$.

Perceptron Analysis: Claim 2

Claim 2: $\|w_{t+1}\|^2 \leq R^2 M_t$ (norm grows slowly)

Proof: On a mistake, $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle) \neq y_t$, so $y_t \langle w_t, \phi(x_t)\rangle \leq 0$.

Expanding:

$$\|w_{t+1}\|^2 = \|w_t\|^2 + \underbrace{2y_t\langle w_t, \phi(x_t)\rangle}_{\leq 0} + \underbrace{\|\phi(x_t)\|^2}_{\leq R^2} \leq \|w_t\|^2 + R^2$$

Perceptron Analysis: Claim 2

Claim 2: $\|w_{t+1}\|^2 \leq R^2 M_t$ (norm grows slowly)

Proof: On a mistake, $\hat{y}_t = \text{sign}(\langle w_t, \phi(x_t)\rangle) \neq y_t$, so $y_t \langle w_t, \phi(x_t)\rangle \leq 0$.

Expanding:

$$\|w_{t+1}\|^2 = \|w_t\|^2 + \underbrace{2y_t\langle w_t, \phi(x_t)\rangle}_{\leq 0} + \underbrace{\|\phi(x_t)\|^2}_{\leq R^2} \leq \|w_t\|^2 + R^2$$

Telescoping from $\|w_1\|^2 = 0$: each mistake adds at most R^2, so $\|w_{t+1}\|^2 \leq R^2 M_t$.

Perceptron Analysis: Conclusion

Claim 1: $\langle w^*, w_{t+1} \rangle \geq \gamma M_t$ **Claim 2:** $\|w_{t+1}\|^2 \leq R^2 M_t$

Perceptron Analysis: Conclusion

Claim 1: $\langle w^*, w_{t+1} \rangle \geq \gamma M_t$ **Claim 2:** $\|w_{t+1}\|^2 \leq R^2 M_t$

Combining: By Cauchy–Schwarz ($\|w^*\| = 1$),

$$\gamma M_t \leq \langle w^*, w_{t+1} \rangle \leq \|w^*\| \cdot \|w_{t+1}\| = \|w_{t+1}\| \leq R\sqrt{M_t}$$

Perceptron Analysis: Conclusion

Claim 1: $\langle w^*, w_{t+1} \rangle \geq \gamma M_t$ **Claim 2:** $\|w_{t+1}\|^2 \leq R^2 M_t$

Combining: By Cauchy–Schwarz ($\|w^*\| = 1$),

$$\gamma M_t \leq \langle w^*, w_{t+1} \rangle \leq \|w^*\| \cdot \|w_{t+1}\| = \|w_{t+1}\| \leq R\sqrt{M_t}$$

Dividing both sides by $\sqrt{M_t}$:

$$\gamma\sqrt{M_t} \leq R \quad \Rightarrow \quad M_t \leq \frac{R^2}{\gamma^2}$$

Linear Predictors: The Big Picture

What we learned:

Linear Predictors: The Big Picture

What we learned:

1. **Cardinality isn't everything**
   $|\mathcal{H}| = \infty$ doesn't mean learning is impossible!

Linear Predictors: The Big Picture

What we learned:

1. **Cardinality isn't everything**
   $|\mathcal{H}| = \infty$ doesn't mean learning is impossible!

2. **Margin is a complexity measure**
   Perceptron: $M \leq R^2/\gamma^2$ — depends on geometry, not cardinality

Linear Predictors: The Big Picture

What we learned:

1. **Cardinality isn't everything**
   $|\mathcal{H}| = \infty$ doesn't mean learning is impossible!

2. **Margin is a complexity measure**
   Perceptron: $M \leq R^2/\gamma^2$ — depends on geometry, not cardinality

3. **The counterexample exploits** $\gamma \to 0$
   Adversary squeezes points to boundary $\Rightarrow$ margin vanishes

Linear Predictors: The Big Picture

What we learned:

1. **Cardinality isn't everything**
   $|\mathcal{H}| = \infty$ doesn't mean learning is impossible!

2. **Margin is a complexity measure**
   Perceptron: $M \leq R^2/\gamma^2$ — depends on geometry, not cardinality

3. **The counterexample exploits $\gamma \to 0$**
   Adversary squeezes points to boundary $\Rightarrow$ margin vanishes

4. **Efficient algorithms exist**
   Perceptron: $O(d)$ per round vs Halving: $O(|\mathcal{H}|)$ per round

What's Next?

The adversarial counterexamples rely on a **carefully chosen sequence** in a **specific order**.

What's Next?

The adversarial counterexamples rely on a **carefully chosen sequence** in a **specific order**.

What if examples come in **random order**?

What's Next?

The adversarial counterexamples rely on a **carefully chosen sequence** in a **specific order**.

What if examples come in **random order**?

This motivates the statistical learning model — next week.

Week 1 Summary

Main lesson: There is no free lunch for learning.

Week 1 Summary

Main lesson: There is no free lunch for learning.

Course structure:

- Three viewpoints: Statistical, Computational, Online
- Three strands: Theory, Formal proof, Presentation

Week 1 Summary

Main lesson: There is no free lunch for learning.

Course structure:

- Three viewpoints: Statistical, Computational, Online
- Three strands: Theory, Formal proof, Presentation

Concepts introduced:

- Instance space, label space, hypothesis
- Online prediction model, mistake counting
- No-Free-Lunch theorem