Computational and Statistical Learning Theory

TTIC 31120

Prof. Nati Srebro

Lecture 1:

What is Learning?

Formalizing Learning

Online Learning

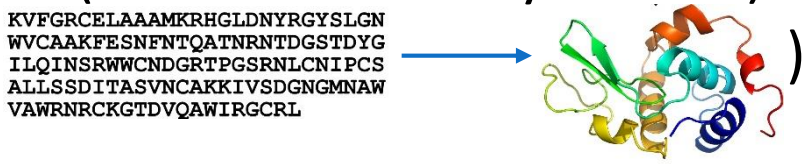
What is “Machine Learning”?

“Learning” (in nature): Using past experience to make future decisions or guide future actions

“Machine Learning” as an Engineering Paradigm: Use data and examples, instead of expert knowledge, to automatically create systems that perform complex tasks

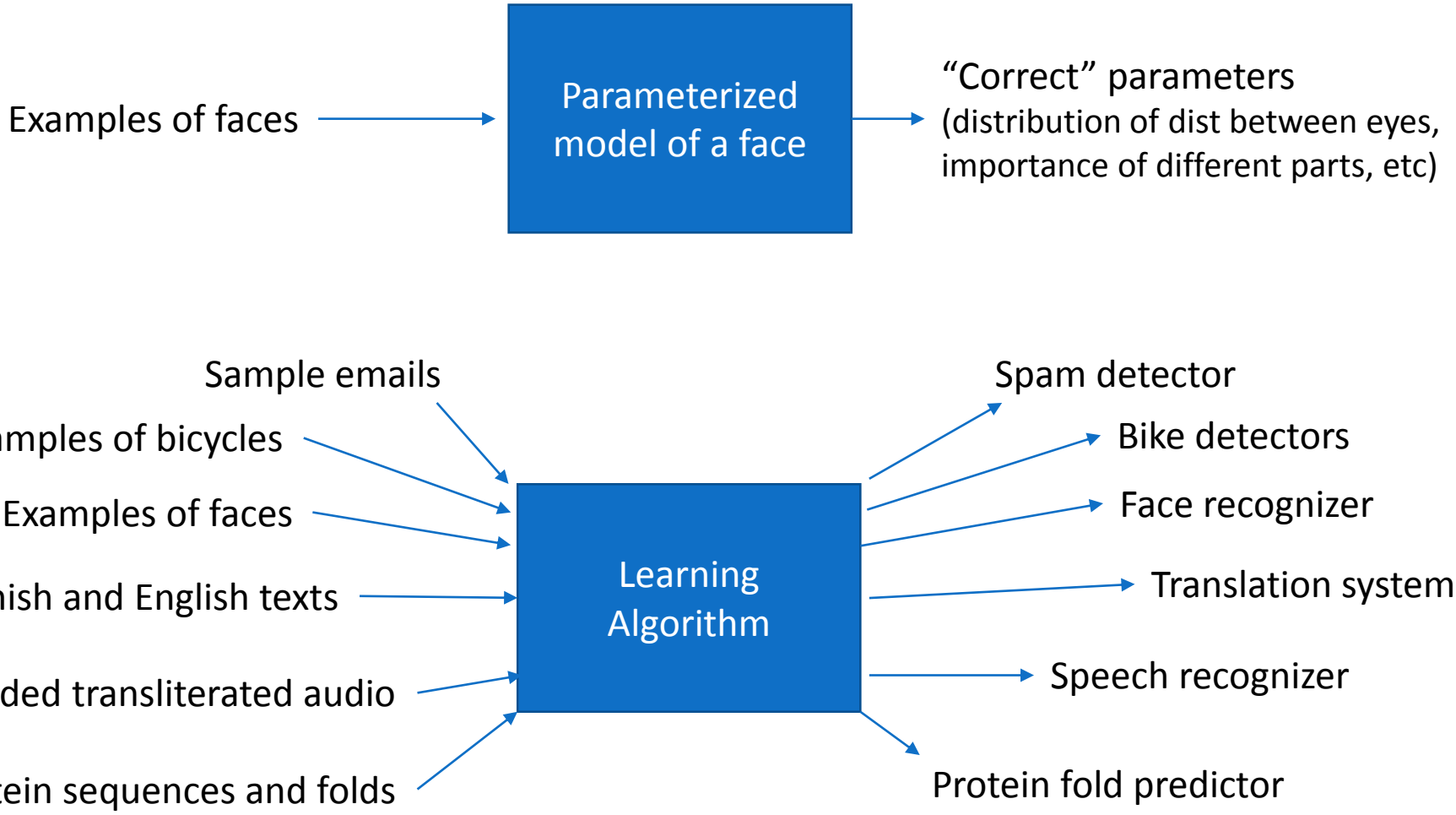
Machine Learning Everywhere

- OCR (addresses, checks, books, forms, pen input, …)
- Spam Filtering
- Machine Translation
- Speech Recognition
- Vision:
  - Face detection and face recognition
  - Object detection (search by object, pedestrian detection, …)
  - Pose estimation in Kineckt
  - Driving assistance systems and self driving cars: pedestrian and hazard detection, staying in-lane, reading road signs, …
- Control: helicopters, robots, self-driving cars
- Search (which is the relevant page?)
- Advertising and ad placement
- Recommendation systems (what movie will you like?)
- Protein fold prediction (



)

Generic Learning



The ability to learn grammars is **hard-wired** into the brain. It is not possible to "learn" linguistic ability—rather, we are born with a brain apparatus specific to language representation.

There exists some "universal" learning algorithm that can learn ***anything***: language, vision, speech, etc. The brain is based on it, and we're working on uncovering it. (Hint: the brain uses neural networks)

There is no "free lunch": no learning is possible without *some* prior assumption about the structure of the problem (prior knowledge)

More Data, Less Expert Knowledge

Expert knowledge:
full specific knowledge

Machine Learning

none

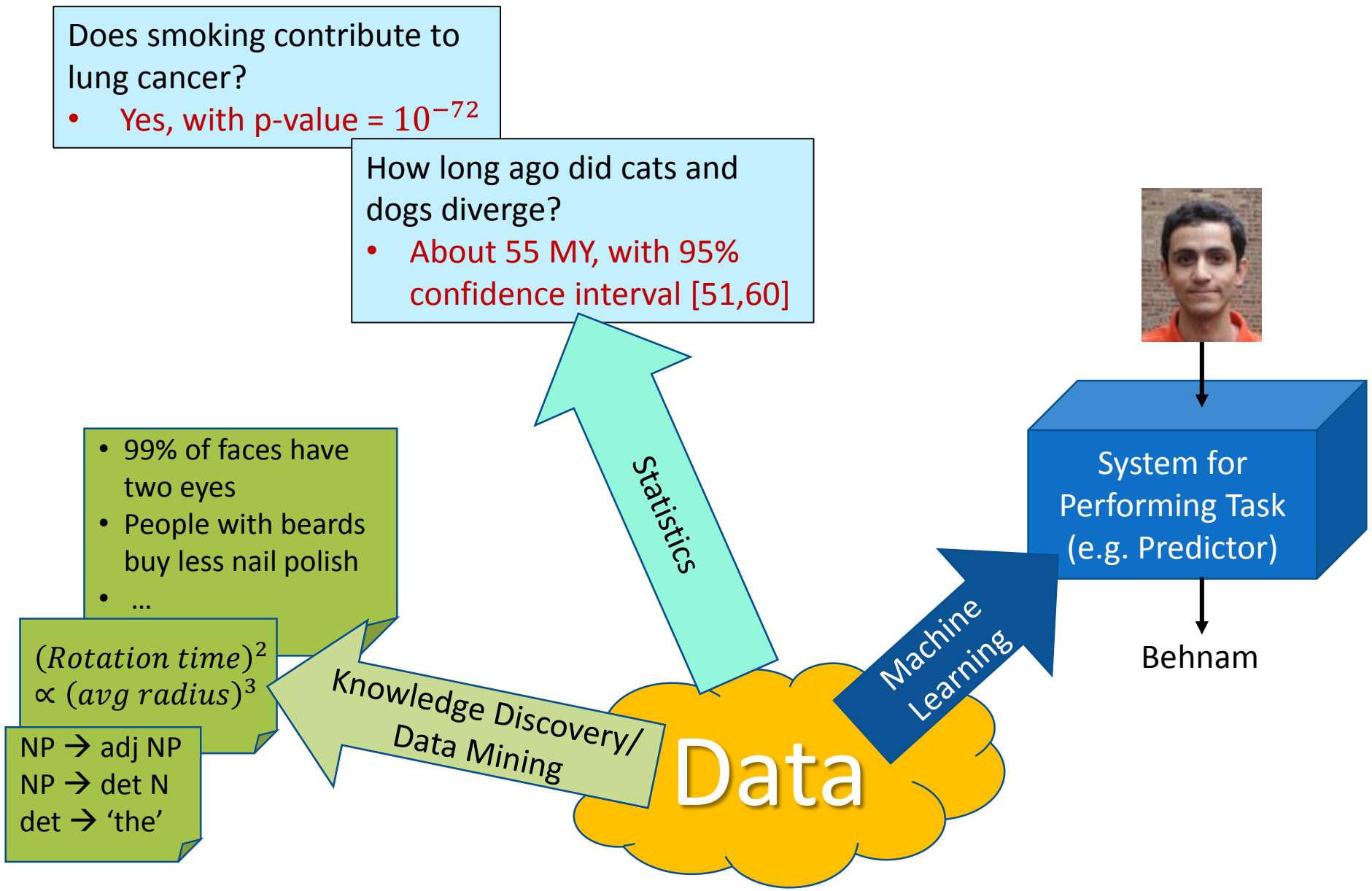
more data →

Use data to fit
specific model

Expert Systems
(no data at all)

no free lunch

"Machine Learning": Use data and examples, instead of expert knowledge, to automatically create systems that perform complex tasks



This course: Computational and Statistical Learning *Theory*

- Assumes you are already familiar with machine learning, and with common learning methods (eg nearest neighbor, SVMs, feed-forward networks)

- Main Goals:
  - Strengthen understanding of learning. What effects how well we can learn, and the required resources (data, computation) for learning? How should we think of and evaluate different learning approaches?
  - Obtain formal and quantitative understanding of learning: Formalize learning, "no free lunch", universal learning, computational limits on learning; Understand learning guarantees and bounds and what they tell us.
  - Understand relationship between learning and optimization, and explore modern optimization techniques in the context of learning.

- Secondary Goal:
  - Learn techniques and develop skills for analyzing learning and proving learning guarantees.

Learning Predictors

- Domain $\mathcal{X}$
  - each $x \in \mathcal{X}$ is called an "instance"
  - e.g. set of all possible email messages

- Label set $\mathcal{Y}$
  - We will focus on binary classification $\mathcal{Y} = \{\pm 1\}$
  - e.g. +1 means "SPAM", -1 means "not SPAM"

- Predictor: $h: \mathcal{X} \to \mathcal{Y}$, i.e. $h \in \mathcal{Y}^{\mathcal{X}}$
  - also called "classifier" or "hypothesis"
  - e.g. $h(x) = \begin{cases} +1, & x \text{ contains the word "free"} \\ -1, & otherwise \end{cases}$

Online Learning Process

- At each time $t = 1,2, \ldots$
  - We receive an instance $x_t \in \mathcal{X}$ (receive an email)
  - We predict a label $\hat{y}_t = h_t(x_t)$ (predict if its spam)
  - We see the correct label y_t of x_t (user tells us if it was really spam)
  - We update the predictor h_{t+1} based on (x_t, y_t)

- Learning rule: mapping A: $(\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}}$
  - $h_t = A\big((x_1, y_1), (x_2, y_2), \ldots, (x_{t-1}, y_{t-1})\big)$

- Goal: make few mistakes $\hat{y}_t \neq y_t$

- Is this possible?
  - E.g. $\mathcal{X} = \{students\ in\ class\}$, $\mathcal{Y} = \{\pm 1\}$
  - Lets play a game: try to learn my labels for you…

No Free Lunch: Online Version

- For **any finite $\mathcal{X}$ with n elements**, and **any learning rule A**, there exists a mapping $f(x)$ and a sequence $\{(x_t, y_t = f(x_t))\}_t$ on which A **makes at least n mistakes**
  - $x_1, \ldots, x_n$ all different
  - $f(x_t) = -\hat{y}_t, t = 1..n$

- For any **infinite $\mathcal{X}$**, and any learning rule A, there exists a mapping $f(x)$ and a sequence $\{(x_t, y_t = f(x_t))\}_t$ on which A **makes a mistake on every round**

- If $\mathcal{X}$ is small, we can limit ourselves to $|\mathcal{X}|$ mistakes by memorizing $h(x_t)$, but "memorizing" doesn't quite feel like "learning"….

Prior Knowledge

- Assume $y_t = f(x_t)$ for some $f \in \mathcal{H}$
- $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is a **"hypothesis class"**
  - Learner knows $\mathcal{H}$, but of course doesn't know f
- $\mathcal{H}$ represents our "Prior Knowledge" or "Expert Knowledge"
- We say the sequence $\{(x_t, y_t)\}_t$ is **realizable by** $\mathcal{H}$

E.g.:

$$\mathcal{H} = all\ predictors\ based\ on\ single\ word\ occurence$$

$$\mathcal{H} = \{h_i \mid dictionary\ word\ i\}, \quad h_i(x) = [[\text{word } i \text{ appears in } x]]$$

$$[[condition]] = \begin{cases} +1, & condition \text{ is } true \\ -1, & \text{otherwise} \end{cases}$$

- What if this assumption is wrong??

  ➔ Later…

Learning Finite Hypothesis Classes

- How can we learn problems realizable by a finite hypothesis class?

- The learning rule **CONSISTENT**:
  - use $h \in \mathcal{H}$ consistent with examples so far
  - $\mathbf{CONSISTENT}_{\mathcal{H}}(S) = \left(some\ h \in \mathcal{H}\ s.t. \forall_{(x,y)\in S}(h(x) = y)\right)$

  (strictly speaking: not a specific function—we will refer to any rule returning a consistent h as "**CONSISTENT**")

- Iterative implementation:
  - Initialize $V_1 = \mathcal{H}$
  - For $t = 1,2, \ldots$
    - Output some $h_t \in V_t$ (and predict $\hat{y}_t = h_t(x_t)$)
    - Based on (x_t, y_t), update $V_{t+1} = \{h \in V_t | h(x_t) = y_t\}$

- Theorem:
  If $\{(x_t, y_t)\}_t$ is realizable by $\mathcal{H}$, $\mathbf{CONSISTENT}_{\mathcal{H}}$ will make $< |\mathcal{H}|$ mistakes

- Proof:
  If $h_t(x_t) \neq y_t$, h_t is removed from V_t, hence $|V_{t+1}| \leq |V_t| - 1$. Since true f always remains in V_t, $|V_t| \geq 1$. Hence, #mistakes $\leq |V_1| - 1$.

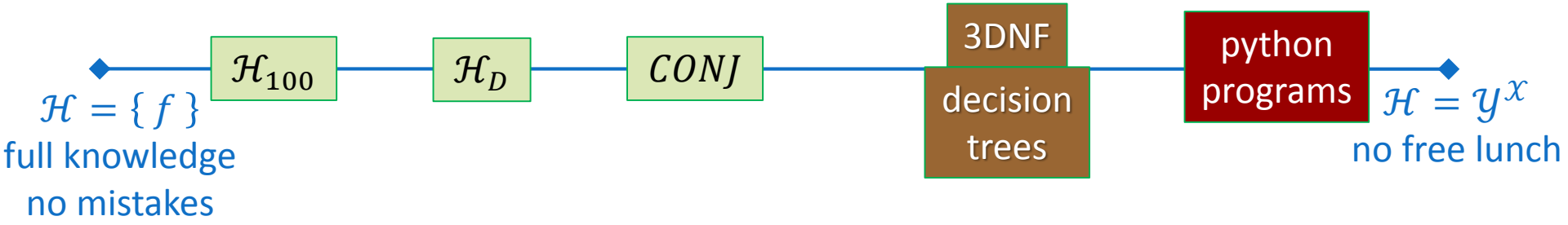
"Halving"

- The $\mathbf{HALVING}_{\mathcal{H}}$ learning rule:
  - Initialize $V_1 = \mathcal{H}$
  - For $t = 1,2, \ldots$
    - Output h_t, where $h_t(x) = \text{MAJORITY}(\, h(x) : h \in V_t \,)$
      (predict $\hat{y}_t = \text{MAJORITY}(\, h(x_t) : h \in V_t \,)$)
    - Based on (x_t, y_t), update $V_{t+1} = \{h \in V_t | h(x_t) = y_t\}$
- Theorem:
  If $\{(x_t, y_t)\}_t$ is realizable by $\mathcal{H}$, $\mathbf{HALVING}_{\mathcal{H}}$ will make $< \log_2 |\mathcal{H}|$ mistakes
- Proof:
  If $h_t(x_t) \neq y_t$, then at least half of the functions $h \in V_t$ are wrong and will be removed, hence $|V_{t+1}| \leq |V_t|/2$.
  Since true f always remains in V_t, $|V_t| \geq 1$. Hence, #mistakes $\leq \log_2 |V_1|$.

- Question: is $\mathbf{HALVING}_{\mathcal{H}}$ a specific function?

The Complexity of $\mathcal{H}$

- $\log_2 |\mathcal{H}|$ measures the "complexity" of the hypothesis class
  - More complex ➔ more mistakes ➔ more data until we learn
  - More specific "expert knowledge" ➔ smaller $\mathcal{H}$ ➔ less mistakes, learn quicker



$\mathcal{H}_{100} = \{ [[\text{word } i \text{ in } x]] \mid i \in \textit{special list of } 100 \textit{ words} \}$ $\log_2 |100|$

$\mathcal{H}_D = \{ [[\text{word } i \text{ in } x]] \mid i \in \textit{dictionary } D \}$ $\log_2 |D|$

$CONJ3 = \{ [[\text{word } i_1, i_2 \text{ OR } i_3 \text{ in } x]] \mid i_1, i_2, i_3 \in D \}$ $\log_2 \binom{|D|}{3} \leq 3 \log_2 |D|$

k-term 3DNF over the literals $\phi_i(x) = [[\text{word } i \text{ in } x]]$ $\leq \log_2 (2|D|)^{3k} = O(k \log |D|)$

e.g. $\left(\phi_{free}(x) \wedge \phi_{special}(x) \wedge \overline{\phi_{equation}(x)}\right) \vee \left(\phi_{email}(x) \wedge \phi_{expire}(x) \wedge \phi_{important}(x)\right)$

Decision trees with k nodes over $\phi_i(x)$ $O(k \log k + k \log |D|)$

Python programs with n lines that take x as input $\leq \log_2 128^{(80\,n)} = O(n)$

Why not use $\mathcal{H} = \{$ short programs $\}$?

- Learn SPAM detectable by 100-line program with $\leq \log_2 128^{100 \cdot 80} = 56{,}000$ mistakes (that's nothing!)
- Running HALVING requires checking, at each step, and for each program, whether it returns the right answer. ***That's not even computable!***
- Even for classes where predictors are easily computable, such as decision trees:
  - #mistakes (≈data needed to learn) $\leq \log_2 |\mathcal{H}|$
  - But runtime scales linearly with $|\mathcal{H}|$ (need to check all $h \in \mathcal{H}$)
  - E.g. for decision trees: #mistakes $\leq O(k \log k + k \log |D|)$
    Runtime = $O\left(k(k|D|)^k\right)$
- We want hypothesis classes that:
  - Capture lots of interesting stuff with low complexity (e.g. low cardinality)
  - Are computationally efficiently learnable

Interim Summary

- $\log_2 |\mathcal{H}|$ measures complexity, gives bounds on number of mistakes ($\approx$ data required for learning), at least in realizable case
- Lots of interesting "universal" classes with bounded cardinality
- … but runtime is exponential (or worse)

- Issues we still need to worry about:
  - Computational efficiency
  - Errors (non-realizability)

But first…

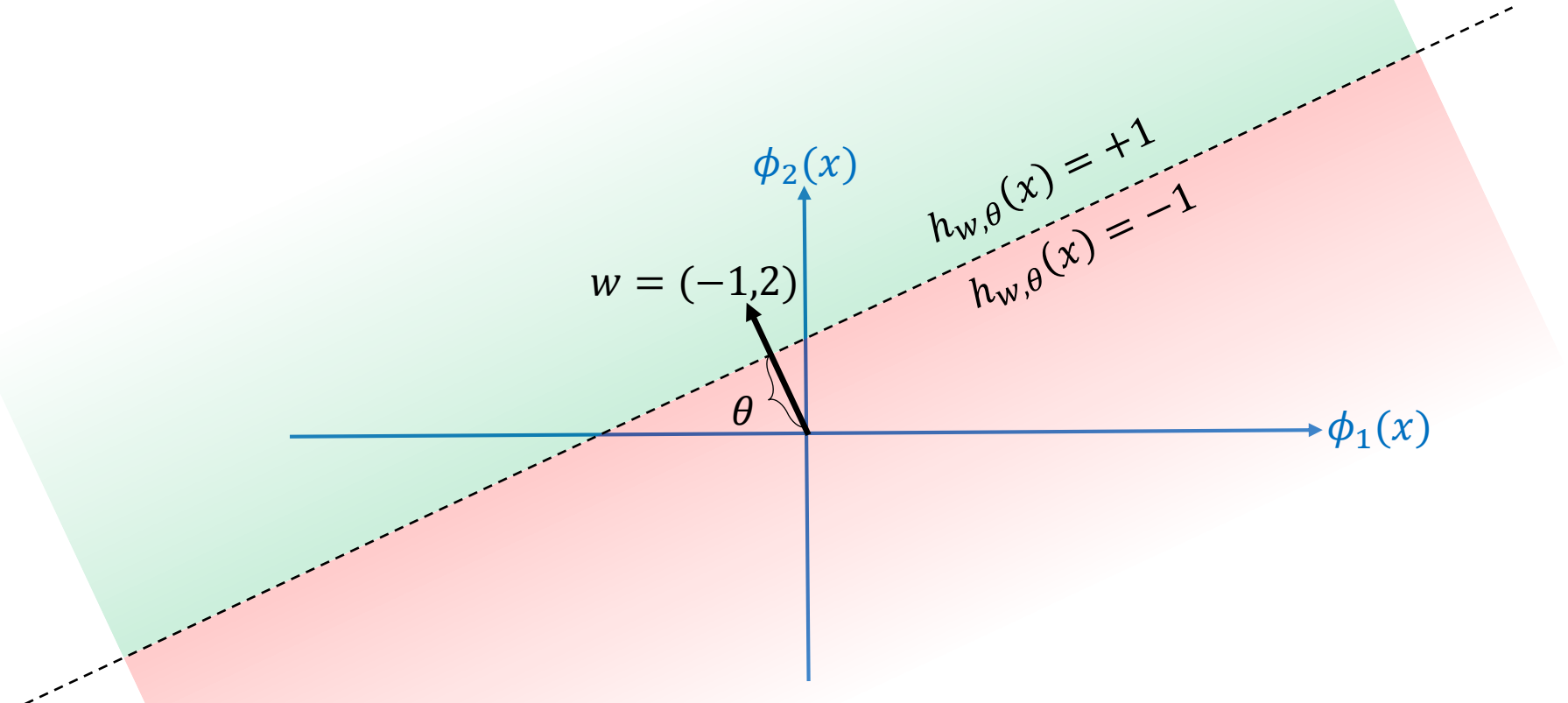
An Important Class: Linear Predictors

- Decide on some *features* $\phi_1(x), \phi_2(x), \ldots$

$$\phi_i : \mathcal{X} \to \mathbb{R}, \qquad \phi(x) = \left(\phi_1(x), \phi_2(x), \ldots, \phi_d(x)\right) \in \mathbb{R}^d$$

- Linear classifiers (halfspaces) over ϕ:

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



An Important Class: Linear Predictors

- Decide on some *features* $\phi_1(x), \phi_2(x), \ldots$

$$\phi_i : \mathcal{X} \to \mathbb{R}, \qquad \phi(x) = \left(\phi_1(x), \phi_2(x), \ldots, \phi_d(x)\right) \in \mathbb{R}^d$$

- Linear classifiers (halfspaces) over ϕ:

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

- The feature map $\phi(x)$, and the decision to use linear classifiers, encodes our prior knowledge

- E.g. $\phi_i(x) = [[\text{word } i \text{ in } x]]$ or $\phi_i(x) = (\#\text{occurrences of } i \text{ in } x)$

… or maybe $\phi_{CAPS}(x) = \#all\ caps$, or $\phi_{i \text{ in subject}}(x)$

… or #occurrences of phrases or pairs of words

- Can encode conjunctions: $\left[\left[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 0.5\right]\right]$

… and disjunctions: $\left[\left[\phi_{i_1} + \phi_{i_2} + \phi_{i_3} > 2.5\right]\right]$

… and negations, and weights

- Can we (online) learn linear predictors? How many mistakes?

Bias term or no bias term?

- Linear classifiers (halfspaces) over $\phi: \mathcal{X} \to \mathbb{R}^d$:
$$\mathcal{H} = \left\{ h_{w,\theta}(x) = \left[\left[\langle w, \phi(x)\rangle > \theta\right]\right] \mid w \in \mathbb{R}^d, \theta \in \mathbb{R} \right\}$$

- Instead, consider augmented feature map with constant feature map $\tilde{\phi}: \mathcal{X} \to \mathbb{R}^{d+1}$:
$$\tilde{\phi}(x) = [\, \phi(x), 1 \,]$$

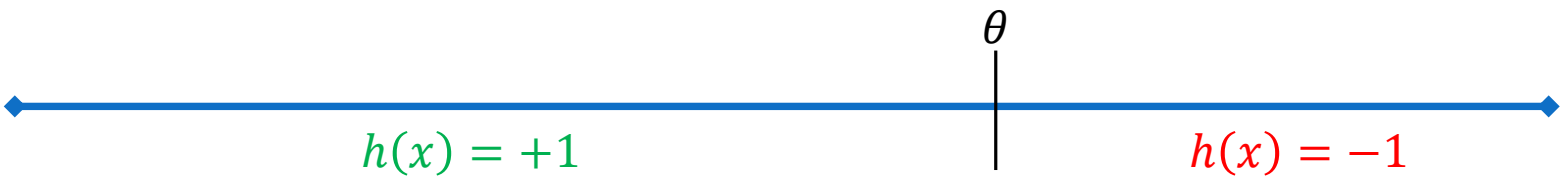
$$\mathcal{H} = \left\{ h_w(x) = \left[\left[\langle w, \tilde{\phi}(x)\rangle > 0\right]\right] \mid w \in \mathbb{R}^{d+1} \right\}$$

- Will often be easier to work with homogeneous definition.

Linear Predictors in 1d: Initial Segments

$$\mathcal{H} = \{ [\![\phi(x) \leq \theta]\!] \mid \theta \in \mathbb{R} \} \qquad \phi(x) \in [0,1]$$

e.g. $\phi(x) = \frac{\#\text{CAPS in } x}{\text{total \#chars}}$ (or just think of $\mathcal{X} = [0,1]$)



- **Theorem**: For any learning rule A, there exists a sequence realized by $\mathcal{H}$, on which A makes a mistake on every round
- Proof:
  - $x_1 = 0.5$
  - $y_t = -\hat{y}_t$
  - $x_{t+1} = x_t + y_t 2^{-(t+1)}$
  - Realized by $\theta = 0.5 + \sum_t y_t 2^{-(t+1)}$

So can we really can't learn linear predictors?

- Answer 1:
  - Counterexample based on extremely high resolution
  - If we discretize $\theta \in \left\{0, \frac{1}{r}, \frac{2}{r}, \frac{3}{r}, \ldots, 1\right\}$, $\log_2|\mathcal{H}| = \log_2(r+1)$
  - More generally, for linear predictors over $\phi(x) \in \mathbb{R}^d$:

$$\log|\mathcal{H}_{\text{linear}}| = O(d\ \log r) = O\big(d \cdot (\#\text{bits per number})\big)$$

Half-Spaces With Discrete Weights

$$G_r = \{-1, -\frac{(r-1)}{r}, -\frac{r-2}{r}, \dots, -\frac{1}{r}, 0, \frac{1}{r}, \dots, \frac{r-2}{r}, \frac{r-1}{r}, 1\}$$

$$\phi: \mathcal{X} \to G_r^d$$

$$\mathcal{H} = \left\{ h_w(x) = \left[\left[\langle w, \phi(x)\rangle > 0\right]\right] \mid w \in G_r^d \right\}$$

- $|\mathcal{H}| = (2r+1)^d$, hence HALVING will make at most $O(d \log r)$ mistakes.
- How do we implement HALVING?
  - Keep track of $\Omega(r^d)$ hypothesis?
  - Runtime exponential in d.
- Instead: we will show $poly(d)$-time algorithm, with slightly worse mistake bound

What's the problem with HALVING?

- Recall that HALVING maintains a "version space" V_t of hypothesis consistent with examples so far, and so still under consideration.

$$V_t = \{w \mid \forall_{i=1..(t-1)} y_i \langle w, \phi(x_i) \rangle > 0 \}$$

(V_t is a polytope in $\mathbb{R}^d$ with $(t-1)$ facets)

- Predictions:

$$h_t(x) = MAJORITY\big(h_{w(x)} : w \in V_t \big)$$

- Problem: calculating $MAJORITY\big(h_{w(x)} : w \in V_t \big)$

- Instead: we will maintain ellipsoid $\mathcal{E}_t \supseteq V_t$
  - Easy to calculate majority prediction (just use center of ellipsoid)
  - We will show how to easily update $\mathcal{E}_t$
  - We will bound the number of mistakes we can make

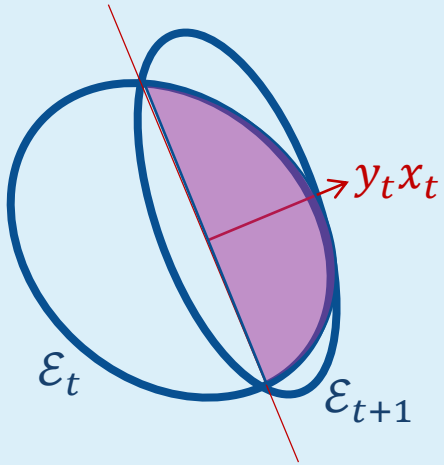
Ellipsoids

- Unit ball in $\mathbb{R}^d$: $B = \{ w \in \mathbb{R}^d \mid \|w\| \leq 1 \}$
- Ellipsoid: image of B under the affine transform $x \mapsto Mx + v$, for some matrix $M \in \mathbb{R}^{d \times d}$, and vector $v \in \mathbb{R}^d$

$$\mathcal{E}(M, v) = \{Mw + v \mid \|w\| \leq 1\}$$

The Ellipsoid Learner

Maintain $\mathcal{E}_t = \mathcal{E}\left(A_t^{1/2}, w_t\right)$, s.t. it always contains all consistent hypothesis



The **ELLIPSOID** learning rule:

- Initialize $A_1 = d \cdot I_d$, $w_1 = 0$
- For each $t = 1,2, \ldots$
  - Output $h_t = \left(x \mapsto \left[\left[\langle w_t, \phi(x)\rangle > 0\right]\right]\right)$
  - Receive (x_t, y_t).
  - If $h_t(x_t) \neq y_t$, update $\mathcal{E}_{t+1}$ to minimum volume ellipsoid that contains $\mathcal{E}_t \cap \{w \mid y_t\langle w, \phi(x_t)\rangle > 0\}$:

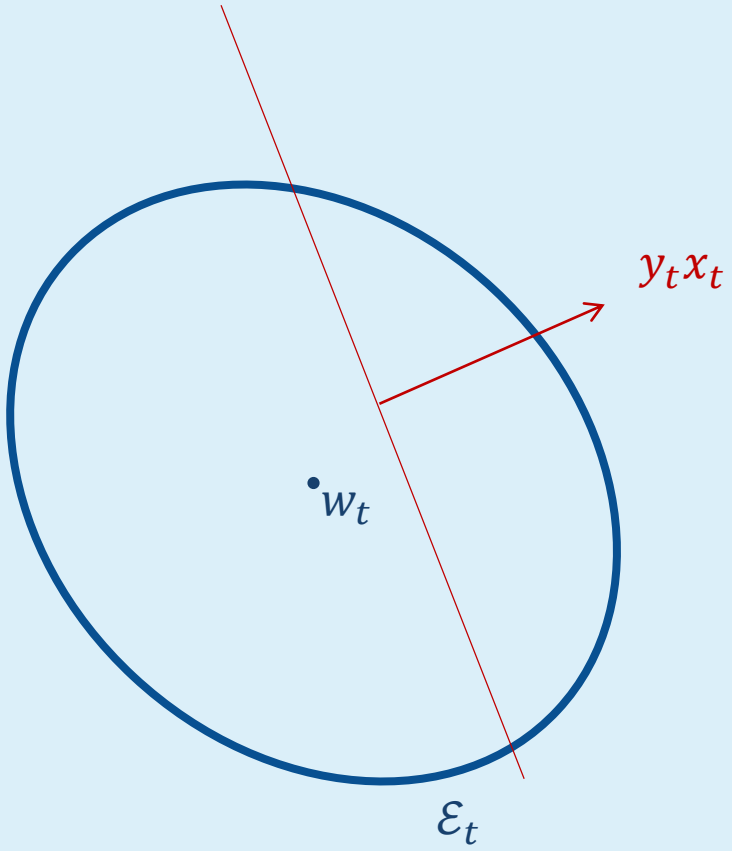
$$w_{t+1} \leftarrow w_t + \frac{y_t}{d+1}\frac{A_t x_t}{\sqrt{x_t' A_t x_t}} \qquad A_{t+1} = \frac{d^2}{d^2-1}\left(A_t - \frac{2}{d+1}\frac{A_t x_t x_t' A_t}{x_t' A_t x_t}\right)$$

  - Otherwise, keep $w_{t+1} = w_t,\ A_{t+1} = A_t$

The Ellipsoid Learner: Analysis

- Lemma: whenever we make a mistake,

$$Vol(\mathcal{E}_{t+1}) \leq e^{\frac{1}{2d+2}} \cdot Vol(\mathcal{E}_t)$$



The Ellipsoid Learner: Analysis

- Lemma: whenever we make a mistake,

$$Vol(\mathcal{E}_{t+1}) \leq e^{\frac{1}{2d+2}} \cdot Vol(\mathcal{E}_t)$$

- Lemma: if $\phi(x) \in G_r^d$ and there is a consistent hypothesis in G_r^d, then

$$Vol(\mathcal{E}_t) \geq \frac{1}{r^{2d}} Vol(B)$$

- And: $Vol(\mathcal{E}_1) = d^{d/2} Vol(B)$

- **Conclusion**: number of mistakes is at most

$$\log_{e^{\frac{1}{2d+2}}} \frac{Vol(\mathcal{E}_1)}{Vol(\mathcal{E}_{t+1})} = (2d+2)\left(2d \log r + \frac{d}{2} \log d\right) = O(d^2 \log(rd))$$

Recall: HALVING makes at most $O(d \log r)$ mistakes.

So can we really can't learn linear predictors?

- Answer 1:
  - Counterexample based on extremely high resolution
  - If we discretize $\theta \in \left\{0, \frac{1}{r}, \frac{2}{r}, \frac{3}{r}, \ldots, 1\right\}$, $\log_2|\mathcal{H}| = \log_2(r+1)$
  - More generally, for linear predictors over $\phi(x) \in \mathbb{R}^d$:

    $$\log|\mathcal{H}_{\text{linear}}| = O(d\ \log r) = O\big(d \cdot (\#\text{bits per number})\big)$$
  - Runtime of HALVING is $\Omega\left(r^d\right)$
  - Can ensure $O(d^2 \log(rd))$ mistakes in time $poly(d)$
  - (Can improve mistake bound using sophisticated randomized algorithm, with worse but still $poly(d)$ runtime)
  - But is the discretization and the sophisticated methods really necessary???

- Answer 2:
  - Counterexample based on very specific sequence, in very specific order
  - What happens if examples (x_t, y_t) come in a random order?