Introduction to Machine Learning
TTIC 31020

Prof. Nati Srebro

Lecture 13:

Feature Selection

Hardness of Proper Learning

Feature Selection

- Lots of features, i.e. very high dimensional $\phi(x) \in \mathbb{R}^D$
- Predict using few features $\phi(x)|_I \;\; I \subset D, |I| \ll D$
- In the context of linear prediction:
  $h_w = \langle w, \phi(x) \rangle$, $supp(w) \subseteq I$, i.e. $\|w\|_0 \ll D$

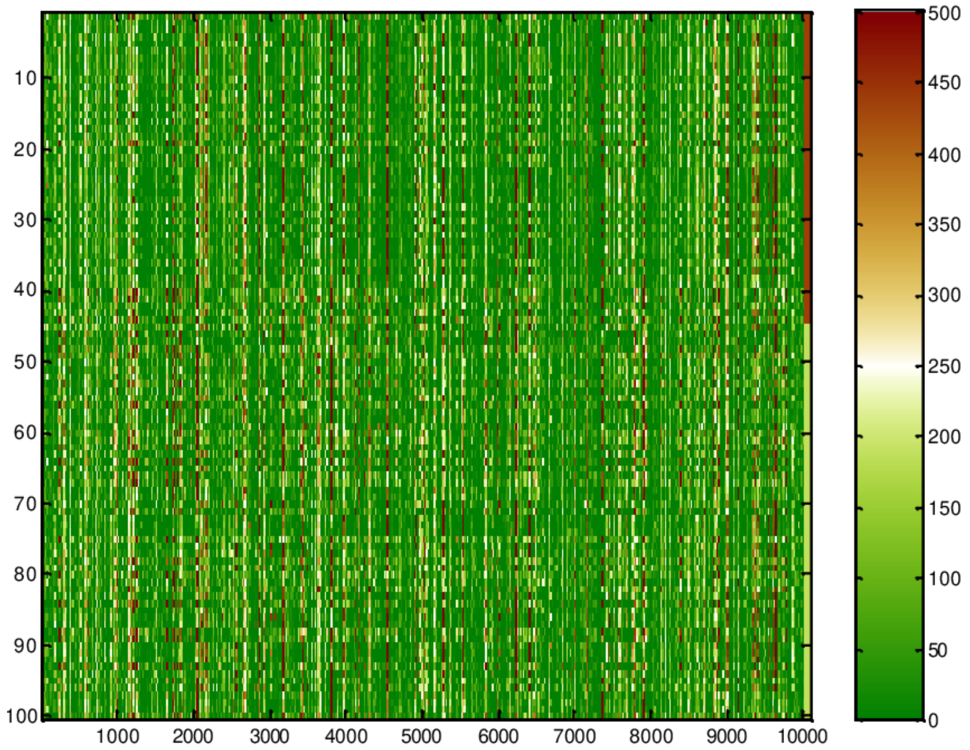
$\|w\|_0 = |\{i | w[i] \;\neq 0\}|$

➔ $\arg\min \; L_s(w) \quad , \;\; \|w\|_0$

E.g. $\quad \arg\min L_S(w) \;$ s.t. $\|w\|_0 \leq k$

$\quad \arg\min \|w\|_0 \;$ s.t. $L_S(w) = 0$

- NP-hard, use heuristic approaches instead



$\phi(x)[i]$=abundance of protein i in blood
y= cancer?

Forward Greedy Selection (eg Coordinate Descent)

$$\arg\min \; L_S(w) \quad , \qquad \|w\|_0$$

Initialize $w^{(0)} = 0, I^{(0)} = \emptyset$

At each iteration k:

- **Find "good" feature $\boldsymbol{i}$**

  Highest directional derivative: $\arg\max \left| \frac{\partial L_S\left(w^{(t)}\right)}{\partial w[i]} \right|$ [AdaBoost]

  Biggest benefit: $\underset{supp(w) \subseteq I^{(k)} \cup \{i\}}{\arg\min} \; L_S(w)$

  …
- **Add feature: $\boldsymbol{I^{(k+1)} = I^{(k)} \cup \{i\}}$**
- **Update $\boldsymbol{w}$ to include $\boldsymbol{w[i]}$**

  Incrementally: $w^{(k+1)} = \arg\min L_S(w) \;\; s.t. \; \forall_{i' \neq i} w[i'] = w^{(k)}[i']$ [AdaBoost]

  Fully Corrective: $w^{(k+1)} = \arg\min L_S(w) \;\; s.t. \; \mathrm{supp}(w) \subseteq I^{(k+1)}$

  …

Variations: Consider adding 2 or 3 features at a time (look-ahead)
Also allow removing (pruning) or replacing features

Forward Greedy Selection as a "Wrapper" for any learning rule $A(S)$

Input: learning rule $A(\cdot)$ and training set S

Initialize $I^{(0)} = \emptyset$

At each iteration k:

- **Find "good" feature $\boldsymbol{i}$**

  Biggest empirical benefit: $\arg\min_i L_S\left(A\left(S|_{I^{(k)}\cup\{i\}}\right)\right)$

  Best validation benefit: $\arg\min_i L_{S_{val}}\left(A\left(S_{train}|_{I^{(k)}\cup\{i\}}\right)\right)$ (or cross validation)
- **Add feature: $\boldsymbol{I^{(k+1)} = I^{(k)} \cup \{i\}}$**
- **Apply learning rule with new feature set: $\boldsymbol{h^{(k+1)} = A\left(S|_{I^{(k+1)}}\right)}$**

Variations: Consider adding 2 or 3 features at a time (look-ahead)
Also allow removing (pruning) or replacing features
Start with I ={all features} are remove features gradually

"Filter" Approaches to Feature Selection

- Before applying learning rule, select which feature to use

Rank features based on correlation with target

$$\rho_i^2 = \frac{\mathbb{E}[\phi(x)[i]y]^2}{\mathbb{E}\big[\phi(x)[i]\big]\mathbb{E}[y]}$$

Use I = k features with largest ρ_i^2

Return $h = \boldsymbol{A}(S|_I)$

or after centering,

or covariance (ie taking into account feature scale or variance),

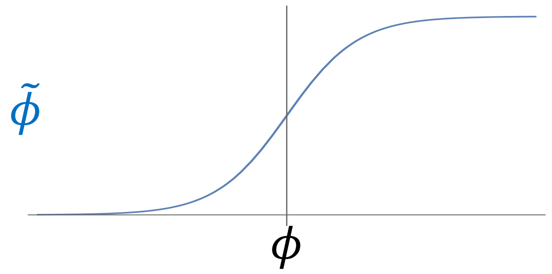
or mutual information,

or some other indicator

- Unlike "wrapper" approaches, selected features *do not* depend on learning rule A

Feature Selection vs Feature Manipulation

- <u>Feature Selection</u>: from a given set of features $\{\phi(x)[i]\}$ select a subset $\{\phi(x)[i] | i \in I\}$ to use
- <u>Feature "Manipulation"</u>: modify features $\phi(x)$ to obtained different features $\tilde{\phi}(x)$
  - Centering: $\tilde{\phi}(x)[i] = \phi(x)[i] - \mu_i$
  - Scaling/Normalization: $\tilde{\phi}(x)[i] = \frac{1}{\sigma_i}\phi(x)[i]$
  - Clipping: $\tilde{\phi}(x)[i] = \text{clip}_{[\text{minval},\text{maxval}]}(\phi(x)[i])$
  - Discretization/quantization: create buckets, $\tilde{\phi}(x)[i] = \text{bucket of } \phi(x)[i]$ or thresholding $\tilde{\phi}(x)[i] = [[\phi(x)[i] > \theta_i]]$
  - Non-linear transformation such as $\tilde{\phi}(x)[i] = \log(b + \phi(x)[i])$ or $\tilde{\phi}(x)[i] = \frac{1}{1+e^{-\phi(x)[i]}}$
  - Possibly acting on entire feature vector, e.g. whitening, PCA
  - **Usually without regard to the target labels**



Convex Surrogate: L_1 Regularization

$$\arg\min_{w} L_S(w) \quad , \quad \|w\|_0$$

Convex Surrogate: L_1 Regularization

$$\arg\min_w L_S(w) \quad , \quad \|w\|_1$$

$\|w\|_1 = \sum_i |w[i]|$

- Original Lasso: $\ell^{\text{sq}}(\langle w, \phi(x)\rangle, y) = \frac{1}{2}(y - \langle w, \phi(x)\rangle)^2$; could use any other loss functions
- L_1 regularization:
  - Ensures generalization (effective dimension $\propto \|w\|_1^2 \|\phi\|_\infty^2 \log D$)
  - Can be thought of as convex surrogate for sparsity
  - Induces sparsity due to non-differentiability at 0

If $\phi(x) \in \{\pm 1\}^D$: $\|\phi(x)\|_2^2 = D$
but $\|\phi(x)\|_\infty^2 = \max_i |\phi(x)[i]| = 1$

ℓ_1

$$\|(1,0,\ldots,0)\|_1^2 \ll \left\|\left(\frac{1}{\sqrt{D}},\frac{1}{\sqrt{D}},\ldots,\frac{1}{\sqrt{D}}\right)\right\|_1^2 = D$$

ℓ_2

$$\|(1,0,\ldots,0)\|_2^2 = \left\|\left(\frac{1}{\sqrt{D}},\frac{1}{\sqrt{D}},\frac{1}{\sqrt{D}},\ldots,\frac{1}{\sqrt{D}}\right)\right\|_2^2 = 1$$

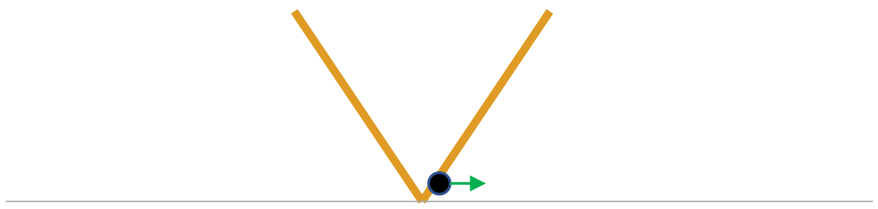
$$F_1(w) = L_S(w) + \lambda\|w\|_1$$

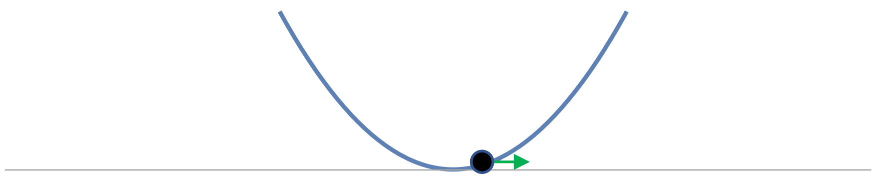
$$F_2(w) = L_S(w) + \lambda\|w\|_2^2$$

what happens when $w[i]$ is already very close to zero:

$$\partial_i F_w(w) = \partial_i L_S(w) + \lambda\, sign(w[i])$$

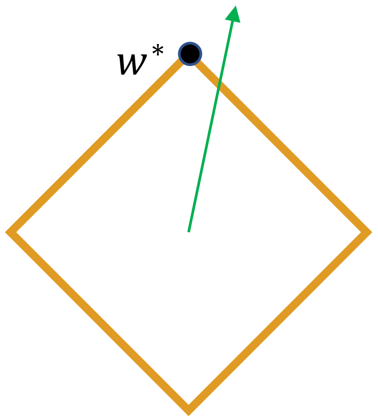
$$\partial_i F_w(w) = \partial_i L_S(w) + 2\lambda\, w[i]$$

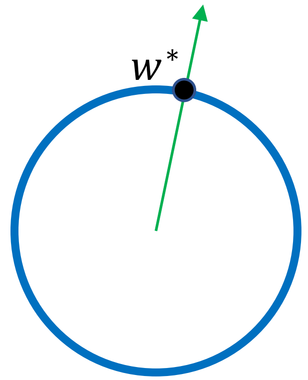




$$\arg\min L_S(w) \quad s.t.\ \|w\|_1 \le B$$

$$\arg\min L_S(w) \quad s.t.\ \|w\|_2 \le B$$





Feature Selection vs Feature Learning

- Feature Selection: from a given set of features $\{\phi(x)[i]\}$ select a subset $\{\phi(x)[i] | i \in I\}$ to use
- Feature Learning: method to construct ***new*** features $\psi(x)$

  …based on x, i.e. $\psi(x)[i] = g_i(x)$

  E.g. linear combinations of features, products or monomials in the features, other combinations of features

  …choose from a class $\mathcal{B} = \{g: \mathcal{X} \to \mathbb{R}\}$ of possible "feature generators"

  E.g. linear functions, stumps, small decision trees, …

  $\equiv$ "feature selection" from $\phi(x) \in \mathbb{R}^{\mathcal{B}}$, $\phi(x)[g] = g(x)$

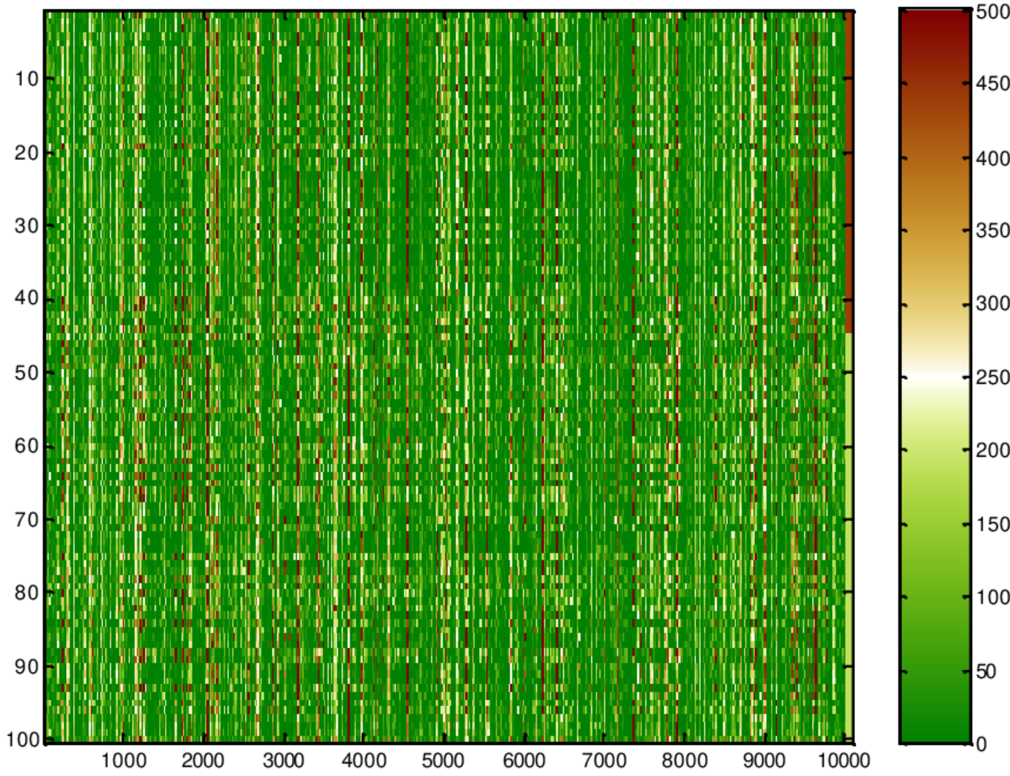
  ➔ Selection from infinitely many features, but that doesn't scare us!

  Can find low $\|w\|_0$ or $\|w\|_1$ predictor over ∞-dim w, e.g. using Boosting, or forward greedy
- Either way: key is sparsity, sparsity-related complexity control/generalization, or a sparsity inducing regularizer/constraint

Why do Feature Selection?

- We want to know what the relevant features are

  ➔ not a learning goal



$\phi(x)[i]$=abundance of protein i in blood
y= cancer?

Why do Feature Selection?

- We want to know what the relevant features are

  ➔ not a learning goal

- Inductive bias / complexity control

  ➔ enough to compete with best sparse predictor:

$$L_{\mathcal{D}}(\hat{w} = A(S)) \leq \inf_{\|w\|_0 \leq k} L_{\mathcal{D}}(w) + \epsilon$$

  - ℓ_1 regularized learning often does this, even if not sparse
  - Bayesian approach: integrate over posterior (over uncertainty) ➔ dense predictor
  - Don't need to worry about getting ONLY correct features (especially If there are many features correlated with the ``correct'' feature)

- Small memory footprint of predictor and fast prediction runtime

  ➔ often better to learn dense predictor (even if reality sparse), then try to sparsify while preserving accuracy (as much as possible)

Method (Team)	test error	$\%\|w\|_0$	% decoy features
BayesNN-DFT (*Neal/Zhang*)	6.48	80.3	47.8
BayesNN-large (*Neal*)	7.27	60.3	28.5
BayesNN-small (*Neal*)	7.13	4.7	2.9
final_2-3 (*Chen*)	7.91	24.9	9.9
BayesNN-large (*Neal*)	7.83	60.3	28.5
final2-2 (*Chen*)	8.80	24.6	6.7
Ghostminer1 (*Ghostminer*)	7.89	80.6	36.1
RF+RLSC (*Torkkola/Tuv*)	8.04	22.4	17.5
Ghostminer2 (*Ghostminer*)	7.86	80.6	36.1
RF+RLSC (*Torkkola/Tuv*)	8.23	22.4	17.5
FS+SVM (*Lal*)	8.99	20.9	17.3
Ghostminer3 (*Ghostminer*)	8.24	80.6	36.1
CBAMethod3E (*CBAGroup*)	8.14	12.8	0.1
CBAMethod3E (*CBAGroup*)	8.14	12.8	0.1
Nameless (*Navot/Bachrach*)	7.78	32.3	16.2

$$\arg\min \quad L_S(w) \quad , \qquad \|w\|_0$$

- ERM for sparse predictors is NP-hard

$$\arg\min L_S(w) \qquad \text{s.t. } \|w\|_0 \le k$$

$$\arg\min \|w\|_0 \text{ s.t. } L_S(w) = 0$$

Or even: does there exists w s.t. $L_S(w) = 0$ and $\|w\|_0 \le k$

➔ not doable in time $poly$(input size)=$poly(mD)$, for any $k \le D$ (unless P=NP)

- Only known method: enumerate over $\binom{D}{k} = O(D^k)$ ➔ overall time $O(mD \cdot D^k) = O(mD^{k+1})$
- All methods suggested (coordinate descent, forward greedy, ℓ_1 relaxation) only guaranteed to return optimum under additional assumptions, and can frequently fail

- Does this mean that learning sparse predictors (without additional assumptions) is computationally *intractable*?

polynomial

"tractable" = can be done in ~~*reasonable*~~ runtime

Efficient PAC (Probably Approximately Correct) Learning

"efficiently" in the textbook and more traditionally

Definition (attempt): A hypothesis class $\mathcal{H}$ is **tractably** learnable (in the realizable case) if there exists a **poly-time computable** learning rule A such that $\forall\epsilon > 0$, $\exists m(\epsilon)$, $\forall\mathcal{D}$ s.t. $\exists_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*) = 0$,

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[L_{\mathcal{D}}\big(A(S)\big)\right] \leq \epsilon$$

- **Runtime polynomial in what?**
- A(S) polynomial in S
  - Can inflate m. If we really need only $m(\epsilon)$ samples, but computing $A(\cdot)$ is requires exponential runtime, instead ask for $m'(\epsilon) = 2^{m(\epsilon)}$ samples.
- A(S) polynomial in ϵ
  - For sparse linear predictors: runtime is $O\left(\binom{d}{k}|S|d\right) = O\left(d^{k+1}\frac{VC}{\epsilon^2}\right) = O\left(\frac{d^{k+1}k\log d}{\epsilon^2}\right)$
  - In fact, Boosting ➔ runtime always $poly(1/\epsilon)$, so enough to ask about $\epsilon = 0.4$
- What we really want is "polynomial in the size of the problem"

Efficient PAC (Probably Approximately Correct) Learning

Consider a **family** of hypothesis classes $\{\mathcal{H}_n\}_{n=1}^{\infty}$, over $\mathcal{X}_n$ (e.g. $\mathcal{X}_n = \{0,1\}^n$ or $\mathcal{X}_n = \mathbb{R}^n$)

E.g. Linear predictors over $\mathbb{R}^n$

Decision trees with at most n nodes

Programs of length $\leq n$

"efficiently" in the textbook and more traditionally

Definition: A **family** $\mathcal{H}_n$ of hypothesis classes is **tractably** learnable (in the realizable case) if there exists a learning rule A such that $\forall_n \forall \epsilon, \exists m(n,\epsilon), \forall \mathcal{D}$ s.t. $\exists_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*) = 0$,

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[L_{\mathcal{D}}\big(A(S)\big)\right] \leq \epsilon$$

and ***A* can be computed in time $\boldsymbol{poly(n, {}^1/_\epsilon)}$** [due to Boosting: $poly(n)$ for $\epsilon = 0.4$ is enough]

- In particular, this implies $m(n,\epsilon) \leq poly(n, {}^1/_\epsilon)$
- Alternative view: instead of algorithm $A(S)$ taking S as input, algorithm $A(\mathcal{D},\epsilon)$, that is allowed to sample from $\mathcal{D}$ in unit time.

What can we learn tractably?

In realizable setting (assuming $\exists_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*) = 0$):

- Linear Predictors

  Using an LP feasibility problem:
  Find w s.t. $\forall_i \; y_i \langle w, \phi(x_i) \rangle > 1$
- Polynomials

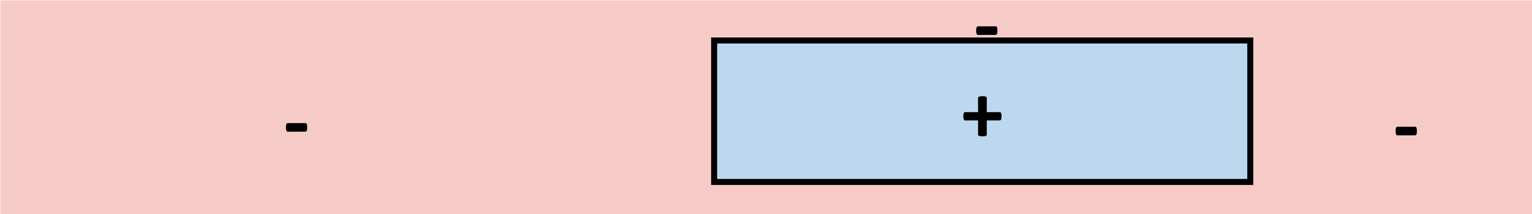
  As linear predictors over expanded feature space
- Conjunctions of variables (e.g. $x_7 \text{ AND } x_{12} \text{ AND } x_{33}$)

  Use conjunction of all variables≠0 when $y = 1$

  … or view as linear predictor
- Axis Aligned Rectangles

  By finding minimal enclosing rectangle of positive points

Can implement CONSISTENT:
“find $h \in \mathcal{H}$ with $L_S(h) = 0$”



- For any family $\mathcal{H}_n$ of hypothesis classes consider the "find consistent" problem:

$\boldsymbol{FINDCONS_{\mathcal{H}}}$

Input: $S = \{(x_1, y_1), \dots, (x_m, y_m)\} \subset \mathcal{X}_n \times \mathcal{Y}$
Output: consistent $h \in \mathcal{H}_n$, i.e such that $L_S(h) = 0$,
or "no" if no consistent $h \in \mathcal{H}_n$ exists

- Claim: If
  - $\mathrm{VCdim}(\mathcal{H}_n) \leq poly(n)$, and } **required**
  - There is a poly-time algorithm for $\mathrm{FINDCONS}_{\mathcal{H}}$ } **???**
    (polynomial in size of input)

  then $\mathcal{H}_n$ is tractably learnable.
- **Converse?**

SQRT-SPARSE

$$\mathcal{H}_n = \left\{x \mapsto \langle w, x\rangle \,\middle|\, w \in \mathbb{R}^n, \|w\|_0 \leq \sqrt{n}\right\} \qquad \mathcal{X}_n = \mathbb{R}^n \text{ (or even } \mathcal{X}_n = \{\pm 1\}^n\text{)}$$

- VC-dim($\mathcal{H}_n$) = $O\left(\sqrt{n} \cdot \log(n)\right)$
- **$\mathrm{FINDCONS}_{\mathcal{H}}$ for SQRT-SPARSE is NP-Hard**

➔ We can't learn SQRT-SPARSE in $poly(n)$ time?

Definition: A family $\mathcal{H}_n$ of hypothesis classes is **tractably properly learnable** (in the realizable case) if there exists a learning rule A such that $\forall_n \forall \epsilon, \exists m(n,\epsilon), \forall \mathcal{D}$ s.t. $\exists_{h^* \in \mathcal{H}} L_{\mathcal{D}}(h^*) = 0$,

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[L_{\mathcal{D}}\big(A(S)\big)\right] \le \epsilon$$

$\boldsymbol{A}$ **can be computed in time** $\boldsymbol{poly(n, {}^1/_\epsilon)}$ and $\boldsymbol{A}$ **always outputs a predictor in** $\boldsymbol{\mathcal{H}_n}$

Theorem: if $\mathcal{H}_n$ is tractably **properly** PAC learnable ➔ $\exists$ poly-time randomized algorithm for $\mathrm{FINDCONS}_{\mathcal{H}}$

More precisely $FINDCONST_{\mathcal{H}} \in RP$:

For *any* input S: $\Pr[\, return\ cons\ h \mid \exists\ cons\ h] \ge \frac{3}{4}$

$\Pr[\, return\ no \mid no\ cons\ h] = 1$

$\boldsymbol{FINDCONS_{\mathcal{H}}}$

Input: $S = \{(x_1, y_1), \dots, (x_m, y_m)\} \subset \mathcal{X}_n \times \mathcal{Y}$

Output: consistent $h \in \mathcal{H}_n$, i.e such that $L_S(h) = 0$, *or* "no" if no consistent $h \in \mathcal{H}_n$ exists

Proof: using $A(\cdot)$ construct rand alg For $\mathrm{FINDCONS}_{\mathcal{H}}$:

- If no consistent h ➔ no $h = A(S)$ will have $L_S(h) = 0$ ➔ "no"
- $\exists$ cons. ➔ $\mathcal{D}$ =Unif[S] realizable ➔ $\mathbb{E}\left[L_S\big(A(\tilde{S})\big)\right] \le \frac{1}{8|S|}$

Markov

➔ w.p. $\ge \frac{3}{4}$, $L_S\big(A(\tilde{S})\big) \le \frac{1}{2|S|}$, i.e. $L_S(h) = 0$

$h = A(\tilde{S})$

- We assume here the size of each x is n. $m(n,\epsilon) = poly(n,\epsilon)$ ➔ $m = poly(n, |S|)$, and runtime of A is poly in its input size, runtime is $poly(mn) = poly(n|S|)$ ie overall runtime poly in input size.

1. Sample $\tilde{S} \sim Unif[S]^m$, $m = m\left(n, \frac{1}{8|S|}\right)$ ie m samples from $\mathcal{D} = Unif(S)$
2. $h \leftarrow A(\tilde{S})$
3. Check if $L_S(h) = 0$ by evaluating h on each point in S
4. If yes, output h, otherwise output "no"

Conclusion: No efficient poly-time alg for $\mathrm{FINDCONS}_{\mathcal{H}}$ ➔ $\mathcal{H}_n$ not tractably properly PAC learnbale

SQRT-SPARSE: $\mathcal{H}_n = \{x \mapsto sign(\langle w, x\rangle) \mid w \in \mathbb{R}^n, \|w\|_0 \le \sqrt{n}\}$

$\text{FINDCONS}_{\mathcal{H}}$ is NP-Hard

➔ Unless $\text{NP} = \text{RP}$, $\text{FINDCONS}_{\mathcal{H}} \notin RP$ (ie no poly-time rand alg.)

➔ If $\text{NP} \neq \text{RP}$, SQRT-SPARSE NOT tractably **properly** learnable (returning $\hat{h} = A(S) \in \mathcal{H}_n$)

But is it tractably ***improperly*** learnable?
I.e. by a learning rule possible returning $A(S) \notin \mathcal{H}_n$
but running in time poly(n), and thus using poly(n) samples ??

Use FINDCONS_{LIN_n} for $LIN_n = \{x \mapsto sign(\langle w, x\rangle) \mid w \in \mathbb{R}^n\} \supset \mathcal{H}_n$

Actually $O\left(\frac{n \log m}{m}\right)$ ➔ need only $O\left(\frac{n \log \epsilon}{\epsilon}\right)$ samples

$\mathcal{D}$ realizable by SQRT-SPARSE ➔ $\mathcal{D}$ realizable by LIN

➔ $\mathbb{E}_{\mathcal{D}}\left[\text{FINDCONS}_{LIN_n}(A(S))\right] \le O\left(\sqrt{\frac{VC(LIN_n)}{m}}\right) = O\left(\sqrt{\frac{n}{m}}\right)$

➔ learn with $O\left({}^{n}/_{\epsilon^2}\right) = poly(n, \epsilon)$ samples

SQRT-SPARSE: $\mathcal{H}_n = \{x \mapsto sign(\langle w, x\rangle) \,|\, w \in \mathbb{R}^n, \|w\|_0 \le \sqrt{n}\}$

Can tractably learn with $m = O(n)$ samples using $\mathrm{FINDCONS}_{LIN}$

Contrast to VC(SQRT-SPARSE)=$\sqrt{n}\log n$ ➔

Using $\mathrm{FINDCONS}_{SQRT-SPARSE}$ (eg exhaustive search),
only $m = O(\sqrt{n}\log n)$ samples

Relax statistically easy but computationally hard class
to larger class that's statistically harder but computationally easier:

	$\boldsymbol{CONS_{SQRT-SPARSE}}$	$\boldsymbol{CONS_{LIN}}$
Sample complexity	$\boldsymbol{O(\sqrt{n}\log n)}$	$\boldsymbol{O(n)}$
Runtime	$\boldsymbol{n^{O(n)}}$	$\boldsymbol{O(n^3)}$

$\mathcal{H}_{n,k,C} = \{\, sign(\langle w, x\rangle) \mid w \in \mathbb{R}^n, \|w\|_0 \leq k, |\boldsymbol{w[i]}| \leq \boldsymbol{C}\}$ $\quad\quad \mathcal{X}_n = [-1,1]^n$

$VC(\mathcal{H}_{n,k,C}) = O(k \log n)$, but FIND-CONS NP-Hard if $k = \omega(1)$

Relax to: $\mathcal{H}_{n,k,B} \subseteq \mathcal{F}_{n,B}$, $B = C \cdot k$

$$\boldsymbol{\mathcal{F}_{n,B}} = \{\, \boldsymbol{sign}(\langle \boldsymbol{w}, \boldsymbol{x}\rangle) \mid \boldsymbol{w} \in \mathbb{R}^{\boldsymbol{n}}, \|\boldsymbol{w}\|_1 \leq \boldsymbol{B}\}$$

If there is separation with margin (or with $\ell^{hinge}, \ell^{sq}, \ell^{lgstc}$):

can learn with $O\left(\|w\|_1^2 \log n\right)$ samples

	$\|w\|_0 \leq k, \lvert w[i]\rvert \leq C$	$\|w\|_1 \leq B$
Sample complexity	$\boldsymbol{O(k \log(n))}$	$O(B^2 \log n) = \left(\boldsymbol{k^2 C^2 \log n}\right)$
Computation	Full enumeration: n^k no efficient method	Linear Programming, Boosting or stochastic method

Gap arbitrarily high if $C \gg 1$

Hardness of CONSISTENT

- Axis-aligned rectangles in n dimensions
- **Halfspaces in $\boldsymbol{n}$ dimensions**

CONSISTENT:
Poly-time
Tractably Properly Learnable

- Sparse predictors in high dimensions:

$$\mathcal{H}_n = \left\{x \mapsto \langle w, x\rangle \middle| w \in \mathbb{R}^{2^n}, \|w\|_0 \leq n\right\}$$

- Decision trees of size at most $\mathrm{poly}(n)$
- Logical formulas of size poly(n)
- Neural nets with at most poly(n) units
- Python programs of size at most poly(n)
- Functions computable in poly(n) time

CONSISTENT:
NP-Hard
Not Tractably Properly Learnable
(unless $\mathrm{NP} = \mathrm{RP}$)

Improperly Learnable?