On the tradeoff between computational complexity and sample complexity in learning

Shai Shalev-Shwartz

School of Computer Science and Engineering
The Hebrew University of Jerusalem

Joint work with Sham Kakade, Ambuj Tewari, Ohad Shamir, Karthik Sridharan, Nati Srebro

CS Seminar, Hebrew University, December 2009

PAC Learning

- $\mathcal{X}$ - domain set

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \rightarrow \mathcal{Y}$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$
- Learning (formally)

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$
- Learning (formally)
  - $\mathcal{D}$ - a distribution over $\mathcal{X}$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$
- Learning (formally)
  - $\mathcal{D}$ - a distribution over $\mathcal{X}$
  - Assumption: instances of S are chosen i.i.d. from $\mathcal{D}$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \rightarrow \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$
- Learning (formally)
  - $\mathcal{D}$ - a distribution over $\mathcal{X}$
  - Assumption: instances of S are chosen i.i.d. from $\mathcal{D}$
  - Error: $\mathrm{err}(h) = \mathbb{P}[h(\mathbf{x}) \neq h^\star(\mathbf{x})]$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$
- Learning (formally)
  - $\mathcal{D}$ - a distribution over $\mathcal{X}$
  - Assumption: instances of S are chosen i.i.d. from $\mathcal{D}$
  - Error: $\text{err}(h) = \mathbb{P}[h(\mathbf{x}) \neq h^\star(\mathbf{x})]$
  - Goal: use S to find h s.t. w.p. $1 - \delta$, $\text{err}(h) \leq \epsilon$

PAC Learning

- $\mathcal{X}$ - domain set
- $\mathcal{Y} = \{\pm 1\}$ - target set
- Predictor: $h : \mathcal{X} \to \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, h^\star(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, h^\star(\mathbf{x}_m))$
- Learning (informally): Use S to find some $h \approx h^\star$
- Learning (formally)
  - $\mathcal{D}$ - a distribution over $\mathcal{X}$
  - Assumption: instances of S are chosen i.i.d. from $\mathcal{D}$
  - Error: $\text{err}(h) = \mathbb{P}[h(\mathbf{x}) \neq h^\star(\mathbf{x})]$
  - Goal: use S to find h s.t. w.p. $1 - \delta$, $\text{err}(h) \leq \epsilon$
- Prior knowledge: $h^\star \in \mathcal{H}$

Complexity of Learning

- Sample complexity — How many examples are needed ?

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H}) \log(1/\delta)}{\epsilon}$

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H}) \log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H}) \log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)
- Computational complexity — How much time is needed ?

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H}) \log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)
- Computational complexity — How much time is needed ?
  - Naively: it takes $\Omega(|\mathcal{H}|)$ to implement the ERM

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H})\ \log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)
- Computational complexity — How much time is needed ?
  - Naively: it takes $\Omega(|\mathcal{H}|)$ to implement the ERM
  - Exponential gap between time and sample complexity (?)

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H})\ \log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)
- Computational complexity — How much time is needed ?
  - Naively: it takes $\Omega(|\mathcal{H}|)$ to implement the ERM
  - Exponential gap between time and sample complexity (?)
- This talk — joint time-sample dependency

$$\mathrm{err}(m',\tau) \stackrel{\mathrm{def}}{=} \min_{m \le m'} \ \min_{A:\mathrm{time}(A)\le\tau} \ \mathbb{E}[\mathrm{err}(A(S))]$$

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H})\,\log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)
- Computational complexity — How much time is needed ?
  - Naively: it takes $\Omega(|\mathcal{H}|)$ to implement the ERM
  - Exponential gap between time and sample complexity (?)
- This talk — joint time-sample dependency

$$\mathrm{err}(m', \tau) \stackrel{\mathrm{def}}{=} \min_{m \le m'} \; \min_{A:\mathrm{time}(A) \le \tau} \; \mathbb{E}[\mathrm{err}(A(S))]$$

- Sample complexity — $\arg\min\{m' : \mathrm{err}(m', \infty) \le \epsilon\}$

Complexity of Learning

- Sample complexity — How many examples are needed ?
  - Vapnik: exactly $\frac{\mathrm{VC}(\mathcal{H})\,\log(1/\delta)}{\epsilon}$
  - Using the ERM (empirical risk minimization)
- Computational complexity — How much time is needed ?
  - Naively: it takes $\Omega(|\mathcal{H}|)$ to implement the ERM
  - Exponential gap between time and sample complexity (?)
- This talk — joint time-sample dependency

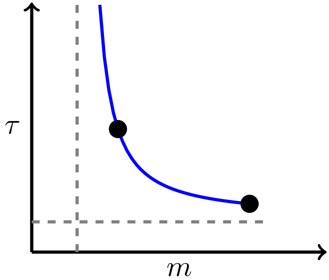
$$\mathrm{err}(m',\tau) \stackrel{\mathrm{def}}{=} \min_{m\le m'} \; \min_{A:\mathrm{time}(A)\le\tau} \; \mathbb{E}[\mathrm{err}(A(S))]$$

  - Sample complexity — $\arg\min\{m' : \mathrm{err}(m',\infty) \le \epsilon\}$
  - Data laden — $\mathrm{err}(\infty,\tau)$

Main Conjecture

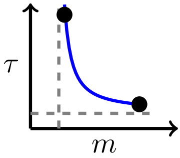
Main Question

How much time, τ, is needed to achieve error $\leq \epsilon$ as a function of sample size, m?



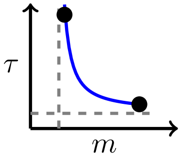
Warmup example

- $\mathcal{X} = \{0, 1\}^d$



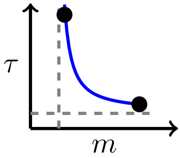
Warmup example

- $\mathcal{X} = \{0,1\}^d$
- $\mathcal{H}$ is 3-term DNF formulae:



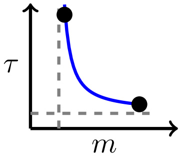
Warmup example

- $\mathcal{X} = \{0,1\}^d$
- $\mathcal{H}$ is 3-term DNF formulae:
  - $h(\mathbf{x}) = T_1(\mathbf{x}) \vee T_2(\mathbf{x}) \vee T_3(\mathbf{x})$, where each T_i is a conjunction



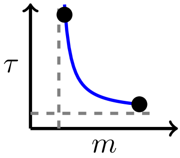
Warmup example

- $\mathcal{X} = \{0,1\}^d$
- $\mathcal{H}$ is 3-term DNF formulae:
  - $h(\mathbf{x}) = T_1(\mathbf{x}) \vee T_2(\mathbf{x}) \vee T_3(\mathbf{x})$, where each T_i is a conjunction
  - E.g. $h(\mathbf{x}) = (x_1 \wedge \neg x_3 \wedge x_7) \vee (x_4 \wedge x_2) \vee (x_5 \wedge \neg x_9)$



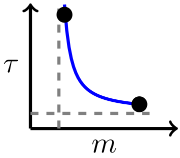
Warmup example

- $\mathcal{X} = \{0,1\}^d$
- $\mathcal{H}$ is 3-term DNF formulae:
  - $h(\mathbf{x}) = T_1(\mathbf{x}) \vee T_2(\mathbf{x}) \vee T_3(\mathbf{x})$, where each T_i is a conjunction
  - E.g. $h(\mathbf{x}) = (x_1 \wedge \neg x_3 \wedge x_7) \vee (x_4 \wedge x_2) \vee (x_5 \wedge \neg x_9)$
  - $|\mathcal{H}| \leq 3^{3d}$ therefore sample complexity is order d/ϵ



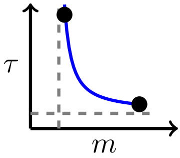
Warmup example

- $\mathcal{X} = \{0,1\}^d$
- $\mathcal{H}$ is 3-term DNF formulae:
  - $h(\mathbf{x}) = T_1(\mathbf{x}) \vee T_2(\mathbf{x}) \vee T_3(\mathbf{x})$, where each T_i is a conjunction
  - E.g. $h(\mathbf{x}) = (x_1 \wedge \neg x_3 \wedge x_7) \vee (x_4 \wedge x_2) \vee (x_5 \wedge \neg x_9)$
  - $|\mathcal{H}| \le 3^{3d}$ therefore sample complexity is order d/ϵ
  - Kearns & Vazirani: If RP$\neq$NP, it is not possible to efficiently find $h \in \mathcal{H}$ s.t. $\mathrm{err}(h) \le \epsilon$



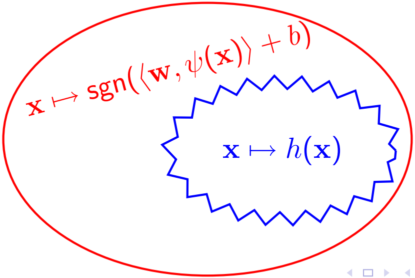
Warmup example

- $\mathcal{X} = \{0,1\}^d$
- $\mathcal{H}$ is 3-term DNF formulae:
  - $h(\mathbf{x}) = T_1(\mathbf{x}) \vee T_2(\mathbf{x}) \vee T_3(\mathbf{x})$, where each T_i is a conjunction
  - E.g. $h(\mathbf{x}) = (x_1 \wedge \neg x_3 \wedge x_7) \vee (x_4 \wedge x_2) \vee (x_5 \wedge \neg x_9)$
  - $|\mathcal{H}| \leq 3^{3d}$ therefore sample complexity is order d/ϵ
  - Kearns & Vazirani: If RP≠NP, it is not possible to efficiently find $h \in \mathcal{H}$ s.t. $\text{err}(h) \leq \epsilon$
- Claim: if $m \geq d^3/\epsilon$ it is possible to find a predictor with error $\leq \epsilon$ in polynomial time



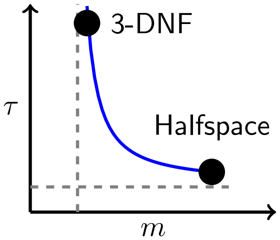
How more data reduces time?

- Observation: $T_1 \vee T_2 \vee T_3 = \wedge_{u \in T_1, v \in T_2, w \in T_3} (u \vee v \vee w)$
- Define: $\psi : \mathcal{X} \to \{0,1\}^{2(2d)^3}$ s.t. for each triplet of literals u, v, w there are two variables indicating if $u \vee v \vee w$ is true or false
- Observation: Exists Halfspace s.t. $h^\star(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \psi(\mathbf{x}) \rangle + b)$
- Therefore, can solve ERM w.r.t. Halfspaces (linear programming)
- VC dimension of Halfspaces is the dimension
- Sample complexity is order d^3/ϵ



Trading samples for runtime

Algorithm	samples	runtime
3-DNF	$\frac{d}{\epsilon}$	2^d
Halfspace	$\frac{d^3}{\epsilon}$	$\text{poly}(d)$



But,

- The lower bound on the computational complexity is only for *proper* learning — there's no lower bound on the computational complexity of improper learning with d/ϵ examples
- The lower bound on the sample complexity of Halfspaces is in the general case — here we have a specific structure

The interesting questions:

- Is the curve really true ? Can one construct 'correct' lower bounds ?
- If the curve is true, one should be able to construct more algorithms on the curve. How?

Second example: Online Ads Placement

For $t = 1, 2, \ldots, m$

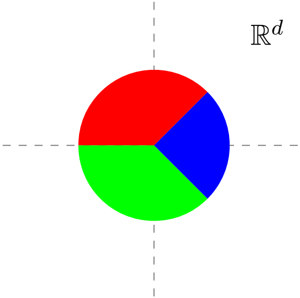
- Learner receives side information $\mathbf{x}_t \in \mathbb{R}^d$
- Learner predicts $\hat{y}_t \in [k]$
- Learner pay cost $\mathbf{1}[\hat{y}_t \neq h^\star(\mathbf{x}_t)]$
- "Bandit setting" — learner does not know $h^\star(\mathbf{x}_t)$

Goal: Minimize error rate:

$$\text{err} = \frac{1}{m} \sum_{t=1}^{m} \mathbf{1}[\hat{y}_t \neq h^\star(\mathbf{x}_t)] \ .$$

Linear Hypotheses

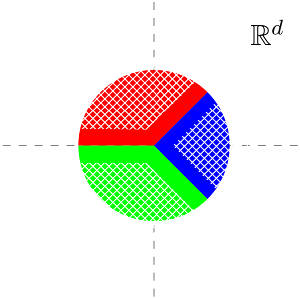
$$\mathcal{H} = \{\mathbf{x} \mapsto \underset{r}{\text{argmax}}\,(W\,\mathbf{x})_r \;\;:\;\; W \in \mathbb{R}^{k,d},\; \|W\|_F \le 1\}$$



Large margin assumption

Assumption: Data is separable with margin μ:

$$\forall t,\ \forall r \neq y_t,\ (W\mathbf{x}_t)_{y_t} - (W\mathbf{x}_t)_r \geq \mu$$



First approach – Halving

Halving for Bandit Multiclass categorization

Initialize: $V_1 = \mathcal{H}$

For $t = 1, 2, \ldots$

- Receive $\mathbf{x}_t$
- For all $r \in [k]$ let $V_t(r) = \{h \in V_t : h(\mathbf{x}_t) = r\}$
- Predict $\hat{y}_t \in \arg\max_r |V_t(r)|$
- If $\mathbf{1}[\hat{y}_t \neq y_t]$ set $V_{t+1} = V_t \setminus V_t(\hat{y}_t)$

First approach – Halving

Halving for Bandit Multiclass categorization

Initialize: $V_1 = \mathcal{H}$

For $t = 1, 2, \ldots$

- Receive $\mathbf{x}_t$
- For all $r \in [k]$ let $V_t(r) = \{h \in V_t : h(\mathbf{x}_t) = r\}$
- Predict $\hat{y}_t \in \arg\max_r |V_t(r)|$
- If $\mathbf{1}[\hat{y}_t \neq y_t]$ set $V_{t+1} = V_t \setminus V_t(\hat{y}_t)$

Analysis:

- Whenever we err $|V_{t+1}| \leq \left(1 - \frac{1}{k}\right) |V_t| \leq \exp(-1/k)\, |V_t|$
- Therefore: $\text{err} \leq \frac{k \log(|\mathcal{H}|)}{m}$
- Equivalently, sample complexity is $\frac{k \log(|\mathcal{H}|)}{\epsilon}$

Using Halving

- Step 1: Dimensionality reduction to $d' = O(\frac{\ln(m+k)}{\mu^2})$
- Step 2: Discretize $\mathcal{H}$ to $(1/\mu)^{kd'}$ hypotheses
- Apply Halving on the resulting finite set of hypotheses

Using Halving

- Step 1: Dimensionality reduction to $d' = O(\frac{\ln(m+k)}{\mu^2})$
- Step 2: Discretize $\mathcal{H}$ to $(1/\mu)^{kd'}$ hypotheses
- Apply Halving on the resulting finite set of hypotheses

Analysis:

- Sample complexity is order of $\frac{k^2/\mu^2}{\epsilon}$
- But runtime grows like $(1/\mu)^{kd'} = (m+k)^{\tilde{O}(k/\mu^2)}$

How can we improve runtime?

- Halving is not efficient because it does not utilize the structure of $\mathcal{H}$
- In the full information case: Halving can be made efficient because each version space V_t can be made convex !
- The Perceptron is a related approach which utilizes convexity and works in the full information case
- Next approach: Lets try to rely on the Perceptron

The Mutliclass Perceptron

For $t = 1, 2, \ldots, m$

- Receive $\mathbf{x}_t \in \mathbb{R}^d$
- Predict $\hat{y}_t \;=\; \arg\max_r (W^t\,\mathbf{x}_t)_r$
- Receive $y_t = h^\star(\mathbf{x}_t)$
- If $\hat{y}_t \neq y_t$ update: $W^{t+1} = W^t + U^t$

The Mutliclass Perceptron

For $t = 1, 2, \ldots, m$

- Receive $\mathbf{x}_t \in \mathbb{R}^d$
- Predict $\hat{y}_t \;=\;$ arg max$_r (W^t\,\mathbf{x}_t)_r$
- Receive $y_t = h^\star(\mathbf{x}_t)$
- If $\hat{y}_t \neq y_t$ update: $W^{t+1} = W^t + U^t$

$$U^t = \begin{bmatrix} 0 & \ldots & 0 \\ & \vdots & \\ 0 & \ldots & 0 \\ \ldots & \mathbf{x}_t & \ldots \\ 0 & \ldots & 0 \\ & \vdots & \\ 0 & \ldots & 0 \\ \ldots & -\mathbf{x}_t & \ldots \\ 0 & \ldots & 0 \\ & \vdots & \\ 0 & \ldots & 0 \end{bmatrix}$$

Row y_t

Row $\hat{y}_t$

Problem: In the bandit case, we're blind to value of y_t

The Banditron (Kakade, S, Tewari 08)

- Explore: From time to time, instead of predicting $\hat{y}_t$ guess some $\tilde{y}_t$
- Suppose we get the feedback 'correct', i.e. $\tilde{y}_t = y_t$
- Then, we have full information for Perceptron's update: $(\mathbf{x}_t, \hat{y}_t, \tilde{y}_t = y_t)$

The Banditron (Kakade, S, Tewari 08)

- Explore: From time to time, instead of predicting $\hat{y}_t$ guess some $\tilde{y}_t$
- Suppose we get the feedback 'correct', i.e. $\tilde{y}_t = y_t$
- Then, we have full information for Perceptron's update: $(\mathbf{x}_t, \hat{y}_t, \tilde{y}_t = y_t)$
- Exploration-Exploitation Tradeoff:
  - When exploring we may have $\tilde{y}_t = y_t \neq \hat{y}_t$ and can learn from this
  - When exploring we may have $\tilde{y}_t \neq y_t = \hat{y}_t$ and then we had the right answer in our hands but didn't exploit it

The Banditron (Kakade, S, Tewari 08)

For $t = 1, 2, \ldots, m$

- Receive $\mathbf{x}_t \in \mathbb{R}^d$
- Set $\hat{y}_t = \arg\max_r (W^t \mathbf{x}_t)_r$
- Define: $P(r) = (1 - \gamma)\mathbf{1}[r = \hat{y}_t] + \frac{\gamma}{k}$
- Randomly sample $\tilde{y}_t$ according to P
- Predict $\tilde{y}_t$
- Receive feedback $\mathbf{1}[\tilde{y}_t = y_t]$
- Update: $W^{t+1} = W^t + \tilde{U}^t$

The Banditron (Kakade, S, Tewari 08)

For $t = 1, 2, \ldots, m$

- Receive $\mathbf{x}_t \in \mathbb{R}^d$
- Set $\hat{y}_t = \arg\max_r (W^t \mathbf{x}_t)_r$
- Define: $P(r) = (1-\gamma)\mathbf{1}[r = \hat{y}_t] + \frac{\gamma}{k}$
- Randomly sample $\tilde{y}_t$ according to P
- Predict $\tilde{y}_t$
- Receive feedback $\mathbf{1}[\tilde{y}_t = y_t]$
- Update: $W^{t+1} = W^t + \tilde{U}^t$ where

$$\tilde{U}^t = \begin{bmatrix} 0 & \ldots & 0 \\ & \vdots & \\ 0 & \ldots & 0 \\ \ldots & \frac{\mathbf{1}[y_t = \tilde{y}_t]}{P(\tilde{y}_t)} \mathbf{x}_t & \ldots \\ 0 & \ldots & 0 \\ & \vdots & \\ 0 & \ldots & 0 \\ \ldots & -\mathbf{x}_t & \ldots \\ 0 & \ldots & 0 \\ & \vdots & \\ 0 & \ldots & 0 \end{bmatrix}$$

Row $\tilde{y}_t$

Row $\hat{y}_t$

The Banditron (Kakade, S, Tewari 08)

Theorem

- *Banditron's sample complexity is order of* $\frac{k/\mu^2}{\epsilon^2}$
- *Banditron's runtime is* $O(k/\mu^2)$

The Banditron (Kakade, S, Tewari 08)

Theorem

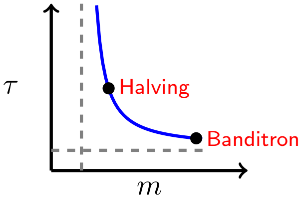
- *Banditron's sample complexity is order of* $\frac{k/\mu^2}{\epsilon^2}$
- *Banditron's runtime is* $O(k/\mu^2)$

The crux of difference between Halving and Banditron:

- Without having the full information, the version space is non-convex and therefore it is hard to utilize the structure of $\mathcal{H}$
- Because we relied on the Perceptron we did utilize the structure of $\mathcal{H}$ and got an efficient algorithm
- We managed to obtain 'full-information examples' by using exploration
- The price of exploration is a higher regret

Trading samples for runtime

Algorithm	samples	runtime
Halving	$\frac{k^2/\mu^2}{\epsilon}$	$(m+k)^{\tilde{O}(k/\mu^2)}$
Banditron	$\frac{k/\mu^2}{\epsilon^2}$	k/μ^2



Next example: Agnostic PAC learning of fuzzy halfspaces

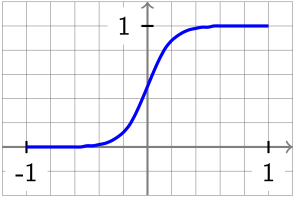
Agnostic PAC:

- $\mathcal{D}$ - arbitrary distribution over $\mathcal{X} \times \mathcal{Y}$
- Training set: $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
- Goal: use S to find h_S s.t. w.p. $1 - \delta$,

$$\text{err}(h_S) \leq \min_{h \in \mathcal{H}} \text{err}(h) + \epsilon$$

Hypothesis class

$$\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle) : \|\mathbf{w}\|_2 \leq 1\}, \qquad \phi(z) = \frac{1}{1+\exp(-z/\mu)}$$



- Probabilistic classifier: $\mathbb{P}[h_{\mathbf{w}}(\mathbf{x}) = 1] = \phi(\langle \mathbf{w}, \mathbf{x} \rangle)$
- Loss function: $\mathrm{err}(\mathbf{w}; (\mathbf{x}, y)) = \mathbb{P}[h_{\mathbf{w}}(\mathbf{x}) \neq y] = \left| \phi(\langle \mathbf{w}, \mathbf{x} \rangle) - \frac{y+1}{2} \right|$
- Remark: Dimension can be infinite (kernel methods)

First approach — sub-sample covering

- Claim: exists $1/(\epsilon\mu^2)$ examples from which we can efficiently learn $\mathbf{w}^\star$ up to error of ϵ
- Proof idea:
  - $S' = \{(\mathbf{x}_i, y'_i) : y'_i = y_i$ if $y_i\langle\mathbf{w}^\star, \mathbf{x}_i\rangle < -\mu$ and else $y'_i = -y_i\}$
  - Use surrogate convex loss $\frac{1}{2}\max\{0, 1 - y\langle\mathbf{w}, x\rangle/\gamma\}$
  - Minimizing surrogate loss on $S' \Rightarrow$ minimizing original loss on S
  - Sample complexity w.r.t. surrogate loss is $1/(\epsilon\mu^2)$

Analysis

- Sample complexity: $1/(\epsilon\mu)^2$
- Time complexity: $m^{1/(\epsilon\mu^2)} = \left(\frac{1}{\epsilon\mu}\right)^{1/(\epsilon\mu^2)}$

Second Approach – IDPK (S, Shamir, Sridharan)

Learning fuzzy halfspaces using **I**nfinite-**D**imensional-**P**olynomial-**K**ernel

- Original class: $\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle)\}$

Second Approach – IDPK (S, Shamir, Sridharan)

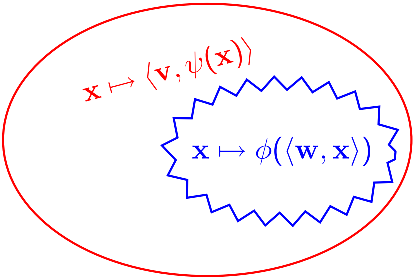
Learning fuzzy halfspaces using **I**nfinite-**D**imensional-**P**olynomial-**K**ernel

- Original class: $\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle)\}$
- Problem: Loss is non-convex w.r.t. $\mathbf{w}$

Second Approach – IDPK (S, Shamir, Sridharan)

Learning fuzzy halfspaces using **I**nfinite-**D**imensional-**P**olynomial-**K**ernel

- Original class: $\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle)\}$
- Problem: Loss is non-convex w.r.t. $\mathbf{w}$
- Main idea: Work with a larger hypothesis class for which the loss becomes convex



Step 2 – Learning fuzzy halfspaces with IDPK

- Original class: $\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle) : \|\mathbf{w}\| \le 1\}$
- New class: $\mathcal{H}' = \{\mathbf{x} \mapsto \langle \mathbf{v}, \psi(\mathbf{x}) \rangle : \|\mathbf{v}\| \le B\}$ where $\psi : \mathcal{X} \to \mathbb{R}^{\mathbb{N}}$ s.t. $\forall j,\ \forall (i_1, \ldots, i_j),\ \psi(\mathbf{x})_{(i_1,\ldots,i_j)} = 2^{j/2}\, x_{i_1} \cdots x_{i_j}$

Step 2 – Learning fuzzy halfspaces with IDPK

- Original class: $\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle) : \|\mathbf{w}\| \le 1\}$
- New class: $\mathcal{H}' = \{\mathbf{x} \mapsto \langle \mathbf{v}, \psi(\mathbf{x}) \rangle : \|\mathbf{v}\| \le B\}$ where $\psi : \mathcal{X} \to \mathbb{R}^{\mathbb{N}}$ s.t. $\forall j,\ \forall (i_1, \ldots, i_j),\ \psi(\mathbf{x})_{(i_1,\ldots,i_j)} = 2^{j/2}\, x_{i_1} \cdots x_{i_j}$

Lemma (S, Shamir, Sridharan 2009)

If $B = \exp(\tilde{O}(1/\mu))$ then for all $h \in \mathcal{H}$ exists $h' \in \mathcal{H}'$ s.t. for all $\mathbf{x}$, $h(\mathbf{x}) \approx h'(\mathbf{x})$.

Step 2 – Learning fuzzy halfspaces with IDPK

- Original class: $\mathcal{H} = \{\mathbf{x} \mapsto \phi(\langle \mathbf{w}, \mathbf{x} \rangle) : \|\mathbf{w}\| \le 1\}$
- New class: $\mathcal{H}' = \{\mathbf{x} \mapsto \langle \mathbf{v}, \psi(\mathbf{x}) \rangle : \|\mathbf{v}\| \le B\}$ where $\psi : \mathcal{X} \to \mathbb{R}^{\mathbb{N}}$ s.t. $\forall j,\ \forall (i_1, \ldots, i_j),\ \psi(\mathbf{x})_{(i_1,\ldots,i_j)} = 2^{j/2}\, x_{i_1} \cdots x_{i_j}$

Lemma (S, Shamir, Sridharan 2009)

If $B = \exp(\tilde{O}(1/\mu))$ then for all $h \in \mathcal{H}$ exists $h' \in \mathcal{H}'$ s.t. for all $\mathbf{x}$, $h(\mathbf{x}) \approx h'(\mathbf{x})$.

Remark: The above is a pessimistic choice of B. In practice, smaller B suffices. Is it tight? Even if it is, are there natural assumptions under which a better bound holds ?
(e.g. Kalai, Klivans, Mansour, Servedio 2005)

Proof idea

- Polynomial approximation: $\phi(z) \approx \sum_{j=0}^{\infty} \beta_j z^j$

Proof idea

- Polynomial approximation: $\phi(z) \approx \sum_{j=0}^{\infty} \beta_j z^j$
- Therefore:

$$
\begin{aligned}
\phi(\langle \mathbf{w}, \mathbf{x} \rangle) &\approx \sum_{j=0}^{\infty} \beta_j (\langle \mathbf{w}, \mathbf{x} \rangle)^j \\
&= \sum_{j=0}^{\infty} \sum_{k_1,\ldots,k_j} 2^{-j/2} \beta_j 2^{j/2} w_{k_1} \cdots w_{k_j} x_{k_1} \cdot \cdots \cdot x_{k_j} \\
&= \langle \mathbf{v}_{\mathbf{w}}, \psi(\mathbf{x}) \rangle
\end{aligned}
$$

Proof idea

- Polynomial approximation: $\phi(z) \approx \sum_{j=0}^{\infty} \beta_j z^j$
- Therefore:

$$\begin{aligned}\phi(\langle \mathbf{w}, \mathbf{x} \rangle) &\approx \sum_{j=0}^{\infty} \beta_j (\langle \mathbf{w}, \mathbf{x} \rangle)^j \\ &= \sum_{j=0}^{\infty} \sum_{k_1,\ldots,k_j} 2^{-j/2} \beta_j 2^{j/2} w_{k_1} \cdots w_{k_j} x_{k_1} \cdots \cdot x_{k_j} \\ &= \langle \mathbf{v}_{\mathbf{w}}, \psi(\mathbf{x}) \rangle\end{aligned}$$

- To obtain a concrete bound we use Chebyshev approximation technique: Family of orthogonal polynomials w.r.t. inner product:

$$\langle f, g \rangle = \int_{x=-1}^{1} \frac{f(x) g(x)}{\sqrt{1 - x^2}} dx$$

Infinite-Dimensional-Polynomial-Kernel

- Although the dimension is infinite, can be solved using the kernel trick
- The corresponding kernel (a.k.a. Vovk's infinite polynomial):

$$\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = K(\mathbf{x}, \mathbf{x}') = \frac{1}{1 - \frac{1}{2}\langle \mathbf{x}, \mathbf{x}' \rangle}$$

- Algorithm boils down to linear regression with the above kernel
- Convex! Can be solved efficiently
- Sample complexity: $(B/\epsilon)^2 = 2^{\tilde{O}(1/\mu)}/\epsilon^2$
- Time complexity: m^2

Trading samples for time

Algorithm	sample	time
Covering	$\frac{1}{\epsilon^2\mu^2}$	$\left(\frac{1}{\epsilon\mu}\right)^{1/(\epsilon\mu^2)}$
	$\Uparrow$	$\Downarrow$
IDPK	$\left(\frac{1}{\epsilon\mu}\right)^{1/\mu} \frac{1}{\epsilon^2}$	$\left(\frac{1}{\epsilon\mu}\right)^{2/\mu} \frac{1}{\epsilon^4}$

Summary

- Trading data for runtime (?)
- There are more examples of the phenomenon ....

Open questions:

- More points on the curve (new algorithms)
- Lower bounds ??? Can you help ?