A complexity theoretic perspective on density estimation

Rocco A. Servedio
Columbia University

Anindya De
UC Berkeley/IAS

Ilias Diakonikolas
U. Edinburgh

STOC workshop
May 31, 2014

Learning Probability Distributions

- Big topic in statistics literature ("density estimation") for decades
- Exciting work in the last decade+ in TCS, largely on learning continuous distributions (mixtures of Gaussians & more)
- This talk: distribution learning from a **complexity theoretic perspective**
  - What about **distributions over the hypercube?**
  - Can we formalize intuition that "simple distributions are easy to learn"?
  - Insights into classical density estimation questions

What do we mean by "learn a distribution"?

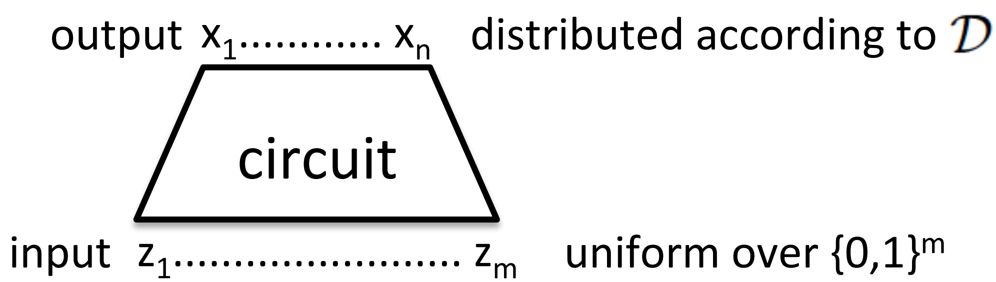
- Unknown target distribution $\mathcal{D}$
- Algorithm gets i.i.d. draws from $\mathcal{D}$
- With probability 9/10, must output (a sampler for a) distribution $\mathcal{D}'$ such that **statistical distance** between $\mathcal{D}$ and $\mathcal{D}'$ is small:

$$d_{\mathrm{TV}}(\mathcal{D}, \mathcal{D}') = \frac{1}{2} \sum_x |\mathcal{D}(x) - \mathcal{D}'(x)| \leq \varepsilon$$

(Natural analogue of Boolean function learning.)

Previous work: [KRRSS94]

- Looked at learning distributions over $\{0,1\}^n$ in terms of n-output circuits that **generate** distributions:



- [AIK04] showed it's hard to learn even very simple distributions from this perspective: already hard even if each output bit is a 4-junta of input bits.

This work: A different perspective

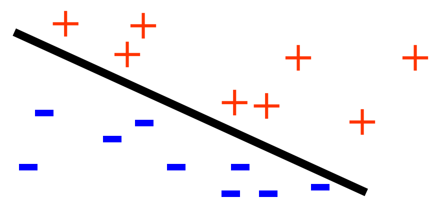
Our notion of a "simple" distribution over $\{0,1\}^n$:
uniform distribution over **satisfying assignments of a "simple" Boolean function.**

What kinds of Boolean functions can we learn from their satisfying assignments?

Want algorithms that have polynomial runtime and # of samples required.

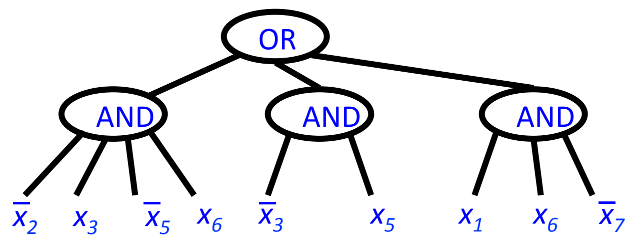
What are “simple” functions?

Halfspaces:



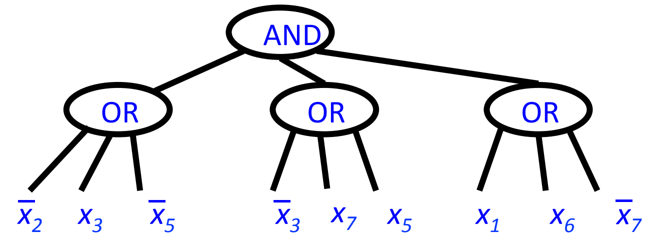
$$f(x) = \mathrm{sign}(w_1 x_1 + \cdots + w_n x_n - \theta)$$

DNF formulas:

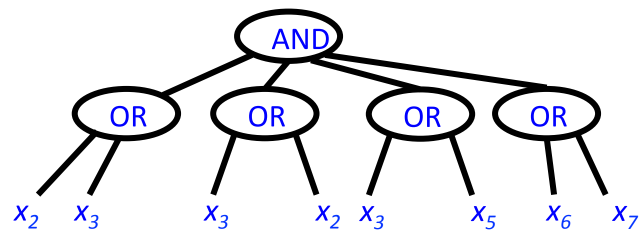


Simple functions, cont.

3-CNF formulas:

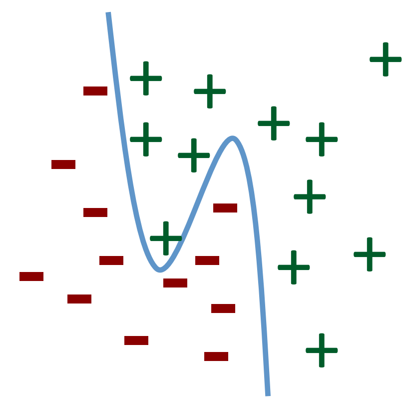


Monotone 2-CNF:



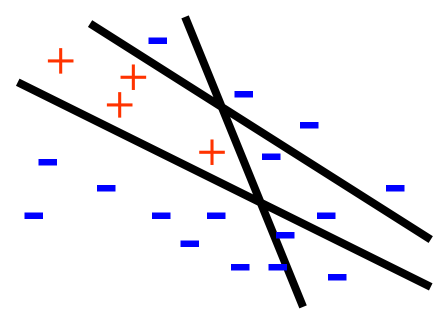
Yet more simple functions

Low-degree polynomial threshold functions:



$$f(x) = \mathrm{sign}(p(x_1, \ldots, x_n))$$

Intersections of k halfspaces:



The model, more precisely

- Let $\mathcal{C}$ be a fixed class of Boolean functions over $\{0,1\}^n$
- There is some unknown $f \in \mathcal{C}$. Learning algorithm sees samples drawn uniformly from $f^{-1}(1)$. Target distribution: $U_{f^{-1}(1)}$.
- Goal : With probability 9/10, output a sampler for a hypothesis distribution $\mathcal{D}$ such that

$$d_{\mathrm{TV}}(\mathcal{D}, U_{f^{-1}(1)}) \leq \varepsilon$$

We'll call this a ***distribution learning algorithm for*** $\mathcal{C}$.

Relation to function learning

Q: How is this different from learning $\mathcal{C}$ (function learning) under the uniform distribution?

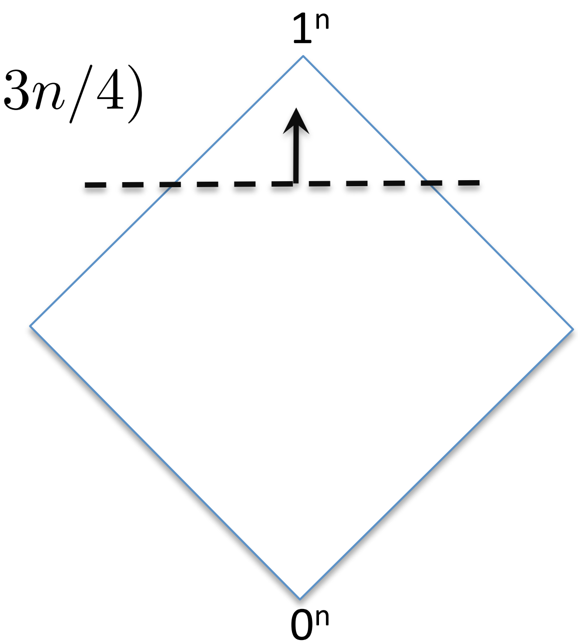
A: Only get positive examples. Some other ways:

- (not so major) Output a hypothesis *distribution* rather than a hypothesis *function*
- (really major) **Much more demanding guarantee** than usual uniform-distribution learning.

Example: Halfspaces

Usual uniform-distribution model for learning functions:
Hypothesis h allowed to be wrong on $\varepsilon 2^n$ points in $\{0,1\}^n$.

$$f = sign(\sum_{i=1}^{n} x_i - 3n/4)$$



$$\Pr_{x \in U_n}[f(x) \neq 0] = 2^{-\Omega(n)}$$

For highly biased target function like f, constant-0 function is a fine hypothesis for any $\varepsilon = 1/\mathrm{poly}(n)$.

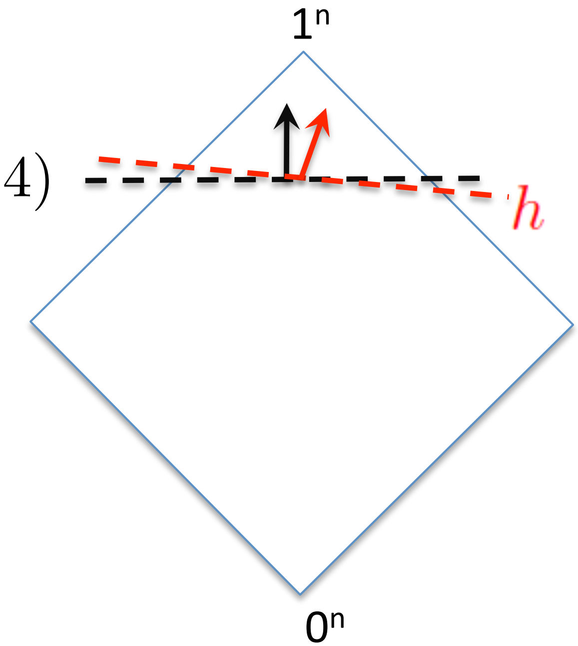
A stronger requirement

Our distribution-learning model: "constant-0 hypothesis" is meaningless!

In this example, for $U_{h^{-1}(1)}$ to be a good hypothesis distribution, $f^{-1}(1)\Delta h^{-1}(1)$ must be only a $2^{-\Omega(n)}$ fraction of $\{0,1\}^n$.

$$f = sign(\sum_{i=1}^{n} x_i - 3n/4)$$

$$\Pr_{x \in U_n}[f(x) \neq 0] = 2^{-\Omega(n)}$$



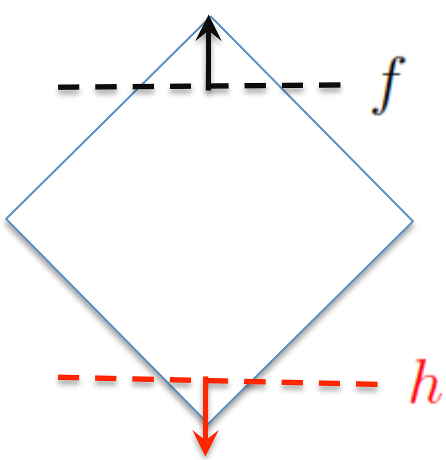
Essentially, we require hypothesis function with **multiplicative** rather than **additive** ε-accuracy relative to f.

Usual function-learning setting

Given: random labeled examples from $\{0,1\}^n$, must

Output: hypothesis h such that

$$\Pr_{x \in U_n}[f(x) \neq h(x)] \leq \epsilon$$



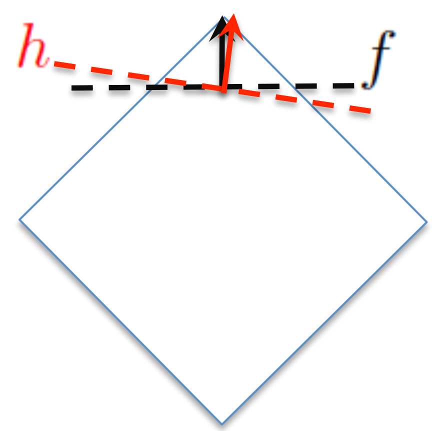
If both regions are small, this h is fine!

Our setting

Given: draws from $U_{f^{-1}(1)}$, must

Output: hypothesis $\mathcal{D}$ with the following guarantee :

$$d_{\mathrm{TV}}(\mathcal{D}, U_{f^{-1}(1)}) \leq \varepsilon$$



h must satisfy

$$\Pr_{x \in U_n}[f(x) \neq h(x)] \leq \epsilon \Pr_{x \in U_n}[f(x) = 1]$$

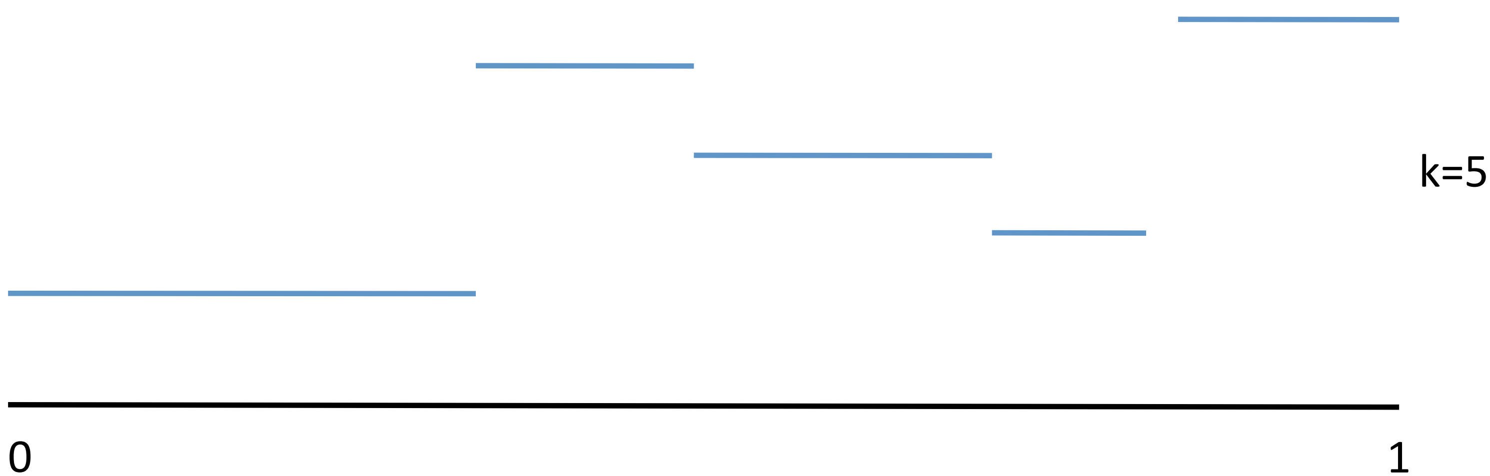
Brief motivational digressions: (1) Real-world language learning

People typically learn new languages by being exposed to correct utterances (**positive examples**), which are a **sparse subset** of all possible vocalizations (all examples).

Goal is to be able to **generate new correct utterances** (generate draws from a distribution similar to the one the samples came from).

(2) Connection to continuous density estimation questions

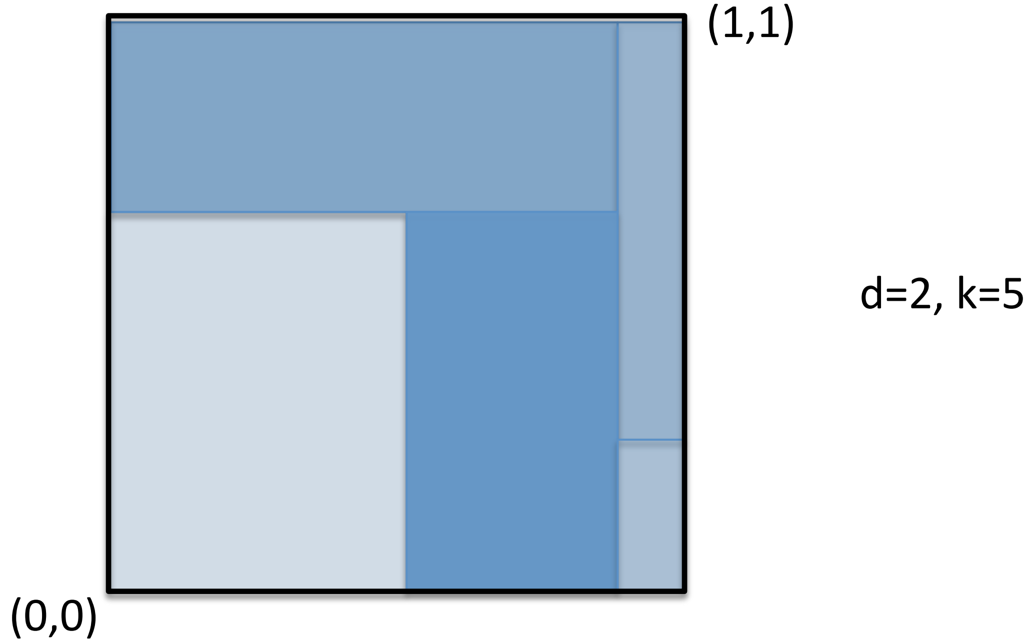
A basic question in continuous 1-dimensional density estimation:
Target distribution (say over [0,1]) is a “k-bin histogram” -- pdf is piecewise constant with k pieces.



Easy to learn such a distribution with poly(k,1/e) samples and runtime.

Multi-dimensional histograms

Target distribution over $[0,1]^d$ is specified by k hyper-rectangles that cover $[0,1]^d$; pdf is constant within each rectangle.



Question: Can we learn such distributions without incurring the "curse of dimensionality"? (Don't want runtime, # samples to be exponential in d)

Connection with our problem

Our "learning from satisfying assignments" problem for the class $\mathcal{C}$ = {all k-leaf decision trees over d Boolean variables} is a (very) special case of learning k-bin d-dimensional histograms.

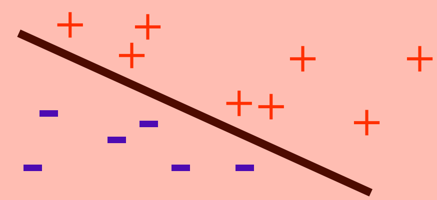
One of the k hyper-rectangles	$\longleftrightarrow$	set of inputs reaching one of the k decision tree leaves
Rectangle with 0 weight in the distribution	$\longleftrightarrow$	decision tree leaf that's labeled 0

For this special case, we beat the "curse of dimensionality" and achieve runtime $d^{O(\log k)}$.

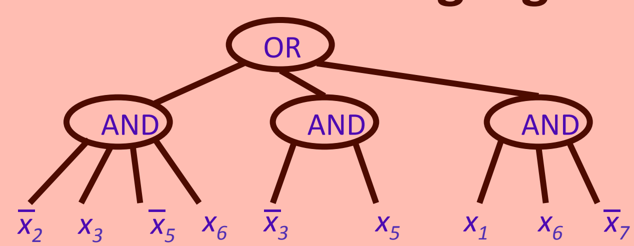
Results

Positive results

Theorem 1: We give an **efficient distribution learning algorithm** for $\mathcal{C}$ = { halfspaces }.
Runtime is $\mathrm{poly}(n, 1/\varepsilon)$.



Theorem 2: We give a (pretty) **efficient distribution learning algorithm** for $\mathcal{C}$ = { poly(n)-term DNFs }.
Runtime is $\mathrm{quasipoly}(n, 1/\varepsilon)$.

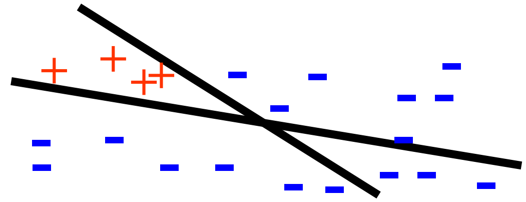


Both results obtained via a **general approach**, plus $\mathcal{C}$ -specific work.

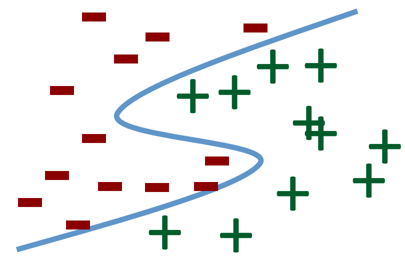
Negative results

Assuming crypto-hardness (essentially RSA), there are **no efficient distribution learning algorithms** for:

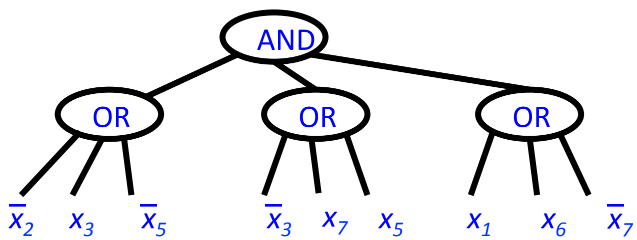
- Intersections of two halfspaces



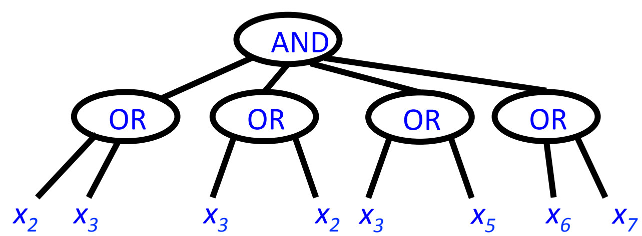
- Degree-2 polynomial threshold functions



- 3 – CNFs ,
  or even



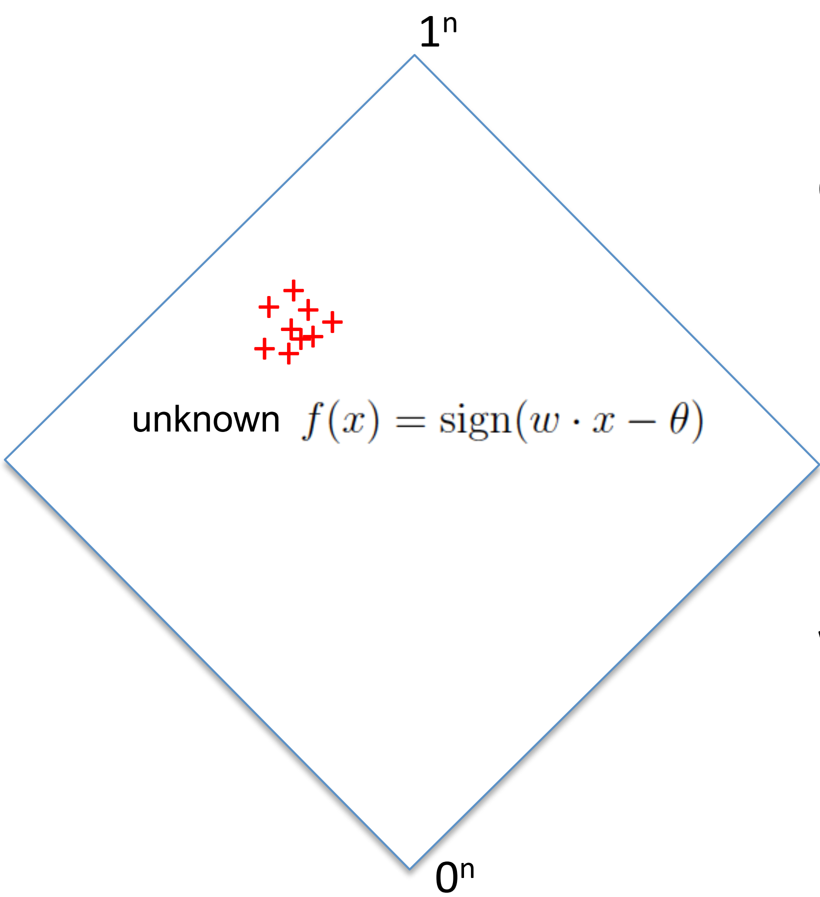
- Monotone 2-CNFs



Rest of talk

- Positive results
- General approach, illustrated through specific case of halfspaces
- Touch on DNFs

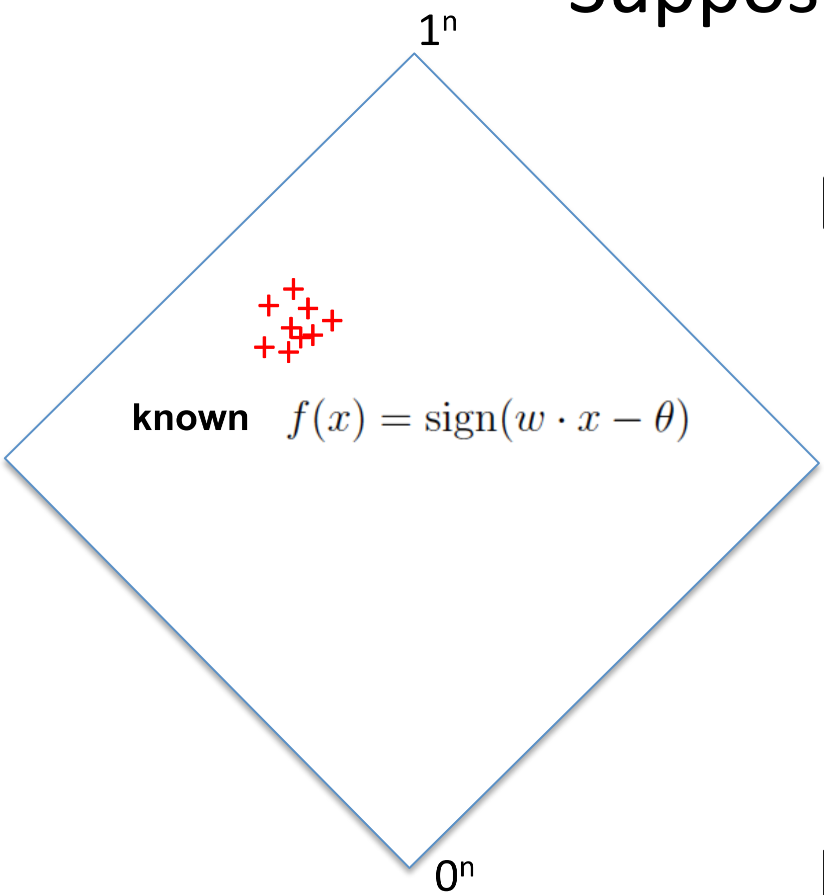
Learning halfspace distributions



Given positive examples drawn uniformly from $f^{-1}(1)$ for some unknown halfspace f,

We need to (whp) output a sampler for a distribution that's close to $U_{f^{-1}(1)}$.

Let's fantasize



Suppose somebody **gave us** f.

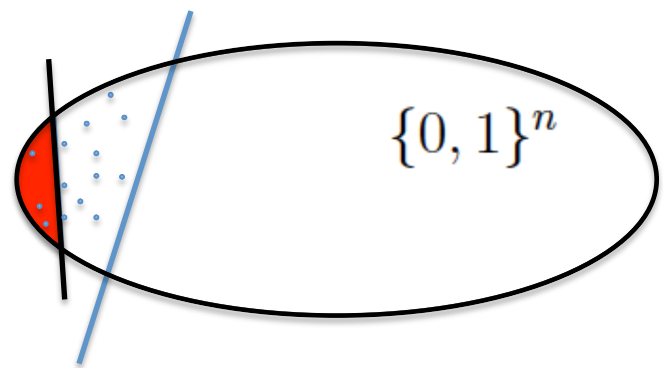
Even then, we need to output a **sampler** for a distribution close to uniform over $f^{-1}(1)$.

Is this doable? Yes.

Approximate sampling for halfspaces

Theorem: Given $f(x) = \mathrm{sign}(w \cdot x - \theta)$ over $\{0,1\}^n$, can return a uniform point from $f^{-1}(1)$ in time $\mathrm{poly}(n, \log(1/\varepsilon))$ (with failure probability ε)

- [MorrisSinclair99]: sophisticated MCMC analysis
- [Dyer03]: elementary randomized algorithm & analysis using "dart throwing"

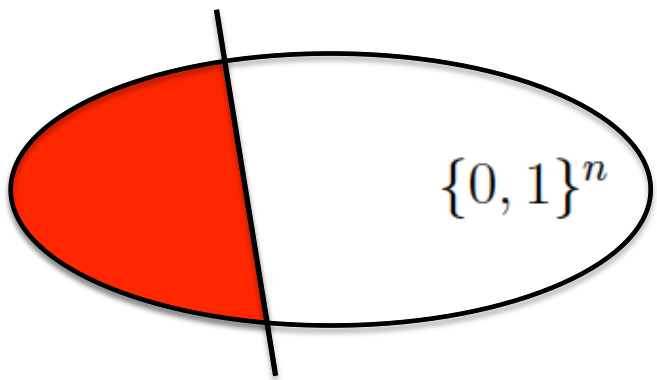


Of course, in our setting we are not given f.

But, we should expect to use (at least) this machinery for our general problem.

A potentially easier case…?

For approximate sampling problem (where we're given f), problem is much easier if $p = |f^{-1}(1)|/2^n$ is large: sample uniformly & do rejection sampling.



Maybe our problem is easier too in this case?

In fact, yes. Let's consider this case first.

Halfspaces: the high-density case

- Let $p = |f^{-1}(1)|/2^n$.

- We will first consider the case that $p \geq n^{-c}$.

- We'll solve this case using Statistical Query learning & hypothesis testing for distributions.

First Ingredient for the high-density case: SQ

Statistical Query (SQ) learning model:

- SQ oracle $\mathcal{O}_{f,D}(\cdot)$: given poly-time computable $\chi : \{0,1\}^n \times \{0,1\} \to \{0,1\}$, outputs $\mathbf{E}_{x \sim D}[\chi(x, f(x))] \pm \tau$ where $\tau = n^{-O(1)}$.

- An algorithm $\mathcal{A}$ is said to be a SQ learner for $\mathcal{C}$ (under distribution D) if $\mathcal{A}$ can learn f given access to $\mathcal{O}_{f,D}(\cdot)$.

SQ learning for halfspaces

Good news: [BlumFriezeKannanVempala97] gave an efficient SQ learning algorithm for halfspaces.

Outputs halfspace hypotheses!

Of course, to run it, need access to oracle for $\mathcal{O}_{f,D}(\cdot)$ for the unknown halfspace f.

So, we need to simulate this given our examples from $U_{f^{-1}(1)}$.

The high-density case: first step

Lemma: Given access to uniform random samples from $U_{f^{-1}(1)}$ and $\widehat{p}$ such that $|\widehat{p} - p| \leq \tau$, queries to $\mathcal{O}_{f,U_n}$ can be simulated up to error 2τ in time $poly(n/\tau)$.

Proof sketch: $\mathbf{E}_{x\sim D}[\chi(x, f(x))] =$

$$\underbrace{\mathbf{E}_{x\sim D}[\chi(x,-1)]}_{\text{Estimate using samples from } \{0,1\}^n} + \underbrace{\mathbf{E}_{x\sim D_{f,+}}[\chi(x,1) - \chi(x,-1)]}_{\text{Estimate using samples from } U_{f^{-1}(1)}} \cdot \underbrace{\mathbf{Pr}_{x\sim D}[f(x) = 1]}_{p=}. \quad \blacksquare$$

The high-density case: first step

Lemma: Given access to uniform random samples from $U_{f^{-1}(1)}$ and $\widehat{p}$ such that $|\widehat{p} - p| \leq \tau$, queries to $\mathcal{O}_{f,U_n}$ can be simulated up to error 2τ in time $poly(n/\tau)$.

Recall promise: $p \geq n^{-c} \quad (p = |f^{-1}(1)|/2^n)$

Additionally, we assume that we have $\widehat{p}$ $= p \pm n^{-2c}$.

A halfspace!

Lemma lets us use the halfspace SQ-learner to get h such that

$$\Pr_{x \in U_n}[h(x) \neq f(x)] \leq n^{-2c}$$

Handling the high-density case

- Since $\Pr_{x \in U_n}[h(x) \neq f(x)] \leq n^{-2c}$, have that
  - $d_{TV}(U_{h^{-1}(1)}, U_{f^{-1}(1)}) \leq n^{-c}$
  - $|h^{-1}(1)| \geq (1/2) \cdot n^{-c} \cdot 2^{-n}$

- Hence using rejection sampling, we can easily sample $U_{h^{-1}(1)}$.

Caveat : We don't actually have an estimate $\hat{p}$ for $|f^{-1}(1)|/2^n$.

Ingredient #2: Hypothesis testing

- Try all possible values of $\widehat{p}$ in a sufficiently fine multiplicative grid $1, \frac{1}{1+\gamma}, \frac{1}{(1+\gamma)^2}, \ldots$

- We will get a list of candidate distributions $U_{h_1^{-1}(1)}, \ \ldots \ , U_{h_m^{-1}(1)}$ such that at least one of them is ϵ-close to $U_{f^{-1}(1)}$.

- Run a "distribution hypothesis tester" to return $U_{h_i^{-1}(1)}$ which is 6ϵ- close to $U_{f^{-1}(1)}$.

Distribution hypothesis testing

Theorem: Given

- Sampler for target distribution $\widehat{D}$
- Approximate samplers for distributions $D_1, \ldots, D_m$
- Approximate evaluation oracles for $D_1, \ldots, D_m$
- Promise : $\exists i \in [m] \; d_{TV}(D_i, \widehat{D}) \leq \epsilon$

Hypothesis tester guarantee: Outputs D_j such that $d_{TV}(D_j, \widehat{D}) \leq 6\epsilon$ in time $poly(m, 1/\epsilon)$

Having **evaluators** as well as samplers for the hypotheses is crucial for this.

Distribution hypothesis testing, cont.

We need samplers & evaluators for our hypothesis distributions $U_{h_i^{-1}(1)}$

All our hypotheses are dense, so can do **approximate counting** easily (rejection sampling) to estimate $|h_i^{-1}(1)|$

Note that $U_{h_i^{-1}(1)}(x) = \begin{cases} 1/|h_i^{-1}(1)| & \text{if } h_i(x) = 1 \\ 0 & \text{if } h_i(x) = 0 \end{cases}$

So we get the required (approximate) evaluators.

Similarly, (approximate) samples are easy via rejection sampling.

Recap

So we handled the high-density case using

- **SQ learning** (for halfspaces)
- **Hypothesis testing** (generic).

(Also used approximate sampling & counting, but they were trivial because we were in the dense case.)

Now let's consider the low-density case (the interesting case).

Low density case: A new ingredient

New ingredient for the low-density case:
A new kind of algorithm called a **densifier**.

- **Input:** $\widehat{p}$ such that $\widehat{p}/p \in [1-\epsilon, 1+\epsilon]$, and samples from $U_{f^{-1}(1)}$
- **Output:** A function g such that:
  - $\Pr_{x \in f^{-1}(1)}[g(x) \neq 1] \leq \epsilon$
  - $\Pr_{x \in U_n}[g(x) = 1] \leq n^c \cdot p$

For simplicity, assume that $g \in \mathcal{C}$ (like f)

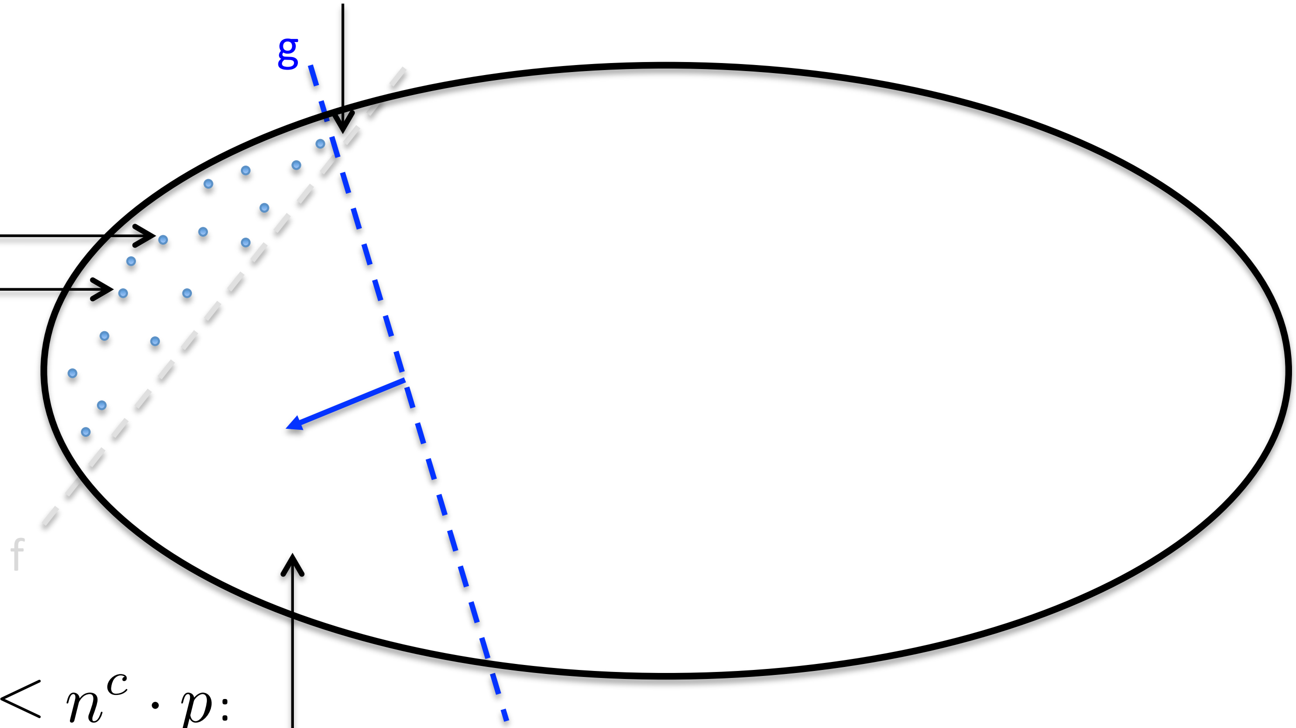
Densifier illustration

$$\Pr_{x \in f^{-1}(1)}[g(x) \neq 1] \leq \epsilon\text{:}$$

$f^{-1}(1) \setminus g^{-1}(1)$ is tiny compared to $f^{-1}(1)$

Samples from $U_{f^{-1}(1)}$

Good estimate $\widehat{p}$



$$\Pr_{x \in U_n}[g(x) = 1] \leq n^c \cdot p\text{:}$$

$g^{-1}(1)$ is not too large compared to $f^{-1}(1)$

Low-density case (cont.)

To solve the low-density case, we need **approximate sampling** and **approximate counting** algorithms for the class $\mathcal{C}$.

This, plus previous ingredients (**SQ learning**, **hypothesis testing**, & **densifier**) suffices: given all these ingredients, we get a distribution learning algorithm for $\mathcal{C}$.

How does it work?

The overall algorithm: (recall that $f \in \mathcal{C}$)

Needs good estimate $\widehat{p}$ of p

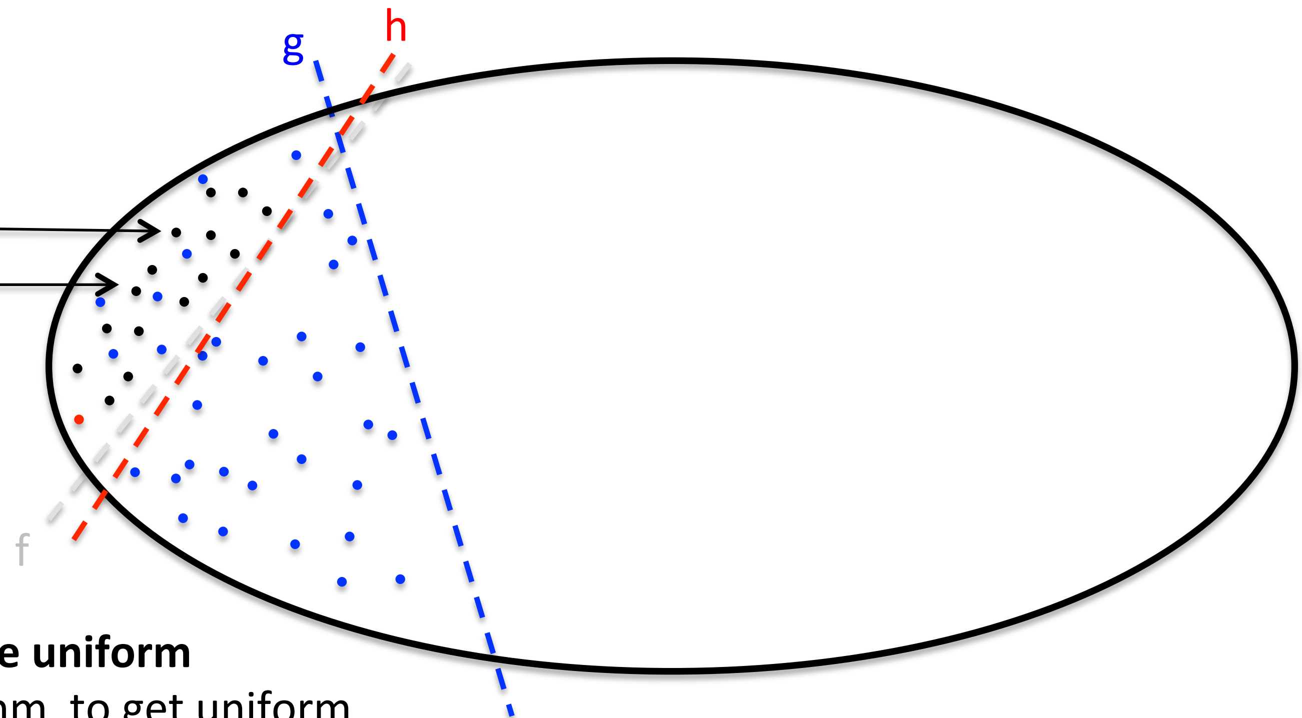
1. Run **densifier** to get $g \in \mathcal{C}$
2. Use **approximate sampling** algorithm for g to get samples from $U_{g^{-1}(1)}$
3. Run **SQ-learner** for f under distribution $U_{g^{-1}(1)}$ to get hypothesis h for f
4. Sample from $U_{g^{-1}(1)}$ till get x such that $h(x) = 1$; output this x.

Repeat with different guesses for $\widehat{p}$, & use **hypothesis testing** to choose $U_{h_i^{-1}(1)}$ that's close to $U_{f^{-1}(1)}$

A picture of one stage

Note: This all assumed we have a good estimate $\hat{p}$

1. Using samples from $U_{f^{-1}(1)}$, run **densifier** to get g



2. Run **approximate uniform generation** algorithm to get uniform positive examples of g

3. Run **SQ-learner** on distribution $U_{g^{-1}(1)}$ to get high-accuracy hypothesis h for f (under $U_{g^{-1}(1)}$)

4. Sample from $U_{g^{-1}(1)}$ till get point x where $h(x) = 1$, and output it.

How it works, cont.

Recall that to carry out hypothesis testing, we need samplers & evaluators for our hypothesis distributions $U_{h_i^{-1}(1)}$

Now some hypotheses h_i may be very sparse…

- Use **approximate counting** to estimate $|h_i^{-1}(1)|$

As before, $U_{h_i^{-1}(1)}(x) = \begin{cases} 1/|h_i^{-1}(1)| & \text{if } h_i(x) = 1 \\ 0 & \text{if } h_i(x) = 0 \end{cases}$

so we get (approximate) evaluator.

- Use **approximate sampling** to get samples from $h_i^{-1}(1)$.

Recap: a general method

Theorem: Let $\mathcal{C}$ be a class of Boolean functions such that:

(i) $\mathcal{C}$ is **efficiently SQ-learnable**;

(ii) $\mathcal{C}$ has a **densifier** with an output in $\mathcal{C}$; and

(iii) $\mathcal{C}$ has **efficient approximate counting** and **sampling** algorithms.

Then there is an **efficient distribution learning algorithm** for $\mathcal{C}$.

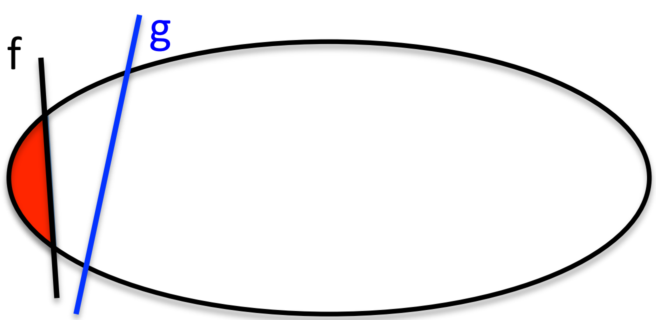
Back to halfspaces: what have we got?

- Saw earlier we have **SQ learning** [BlumFriezeKannanVempala97]
- [MorrisSinclair99,Dyer03] give **approximate counting** and **sampling**.

So we have all the necessary ingredients….except a **densifier**.

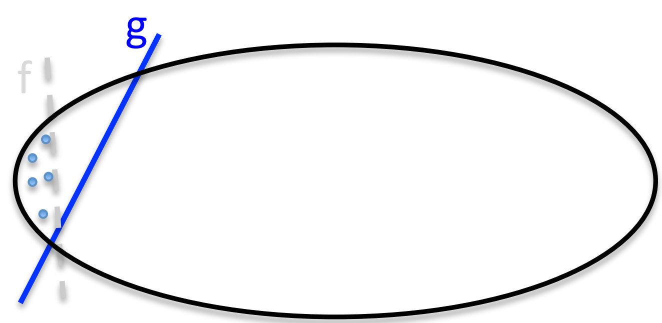
Reminiscent of [Dyer03] "dart throwing" approach to approximate counting – but in that setting, **we are given** f

Approximate counting setting:



Given f, come up with g

Densifier setting:



Can we come up with a suitable g **given only samples from** $U_{f^{-1}(1)}$ **?**

A densifier for halfspaces

Theorem: There is an algorithm running in time $poly(n/\epsilon)$ such that for any halfspace f, if the algorithm gets as input $\widehat{p}$ such that $\widehat{p}/p \in [1-\epsilon, 1+\epsilon]$ and access to $U_{f^{-1}(1)}$, it outputs a halfspace g with the following properties :

1. $\Pr_{x \in f^{-1}(1)}[g(x) \neq 1] \leq \epsilon$, and
2. $\Pr_{x \in U_n}[g(x) = 1] \leq n^{10} \cdot p$

Getting a densifier for halfspaces

Key ingredients:

- Online learner of [MaassTuran90]
- Approximate sampling for halfspaces [MorrisSinclair,Dyer03]

Towards a densifier for halfspaces

Recall our goals:

1. $\Pr_{x \in f^{-1}(1)}[g(x) \neq 1] \leq \epsilon$
2. $\Pr_{x \in U_n}[g(x) = 1] \leq n^{10} \cdot p$

Fact: Let $S_+ \subseteq_R f^{-1}(1)$ be of size n^3/ϵ. Then, with probability $1 - 2^{-\Omega(n)}$, condition (1) holds for **any** halfspace g such that $S_+ \subseteq g^{-1}(1)$.

Proof: If (1) fails for a halfspace g, then $\mathbf{Pr}[S_+ \subseteq g^{-1}(1)] < (1-\epsilon)^{|S_+|}$.
Fact follows from union bound over all (at most 2^{n^2} many) halfspaces g. ■

So ensuring (1) is easy – choose $S_+ \subseteq_R f^{-1}(1)$ and ensure S_+ is consistent with g.
How to ensure (2)?

Online learning as a two-player game

Imagine a two player game in which Alice has a halfspace f and Bob wants to learn f:

i. Bob initializes S to the empty set

ii. Bob runs a (specific polytime) algorithm $\mathcal{A}$ on the set S and returns halfspace h consistent with S

iii. Alice either says "yes, $h \equiv f$" or else returns an $x \in \{0,1\}^n$ such that $h(x) \neq f(x)$

iv. Bob adds $(x, f(x))$ to S and returns to step (ii).

Guarantee of the game

Theorem: [MaassTuran90] There is a specific algorithm $\mathcal{A}$ that Bob can run so that the game terminates in at most $O(n^5)$ rounds. At the end, either $h \equiv f$ or Bob can certify that there is no halfspace meeting all the constraints.

(Algorithm $\mathcal{A}$ is essentially the ellipsoid algorithm.)

Q: How is this helpful for us ?

A: Bob seems to have a powerful strategy ☺ We will exploit it.

Using the online learner

- Choose S_+ as defined earlier. Start with $S = \phi$.
- "Bob" simulation: i^{th} stage – Run Bob's strategy and return h_i consistent with S.
- "Alice" simulation: If $h_i(x) = 0$ for some $x \in S_+$, then return x.
  - Else, if $|h_i^{-1}(1)| \leq n^{10} \cdot \widehat{p} \cdot 2^n$ (**approx counting**) then we are done and return h_i.
  - Else use **approx sampling** to randomly choose a point $x \in h_i^{-1}(1)$ and return x.

Why is the simulation correct?

- If $h_i(x) = 0$ for $x \in S_+$, then the simulation step is indeed correct.
- The other case in which Alice returns a point is that $|h_i^{-1}(1)| \geq n^{10}|f^{-1}(1)|$. This means that the simulation at every step is correct with probability $1 - n^{-10}$.
- Since the simulation lasts $O(n^5)$ steps, all the steps are correct with probability $1 - n^{-5}$.

Finishing the algorithm

- Provided the simulation is correct, h_i which gets returned always satisfies the conditions:

1. $$\Pr_{x \in f^{-1}(1)} [g(x) \neq 1] \leq \epsilon$$
2. $$\Pr_{x \in U_n} [g(x) = 1] \leq n^{10} \cdot p$$

So, we have a densifier – and a distribution learning algorithm – for halfspaces.

DNFs

Recall general result:

Theorem: Let $\mathcal{C}$ be a class of Boolean functions such that:

(i) $\mathcal{C}$ is **efficiently SQ-learnable**;

(ii) $\mathcal{C}$ has a **densifier** with an output in $\mathcal{C}$; and

(iii) $\mathcal{C}$ has **efficient approximate counting** and **sampling** algorithms.

Then there is an **efficient distribution learning algorithm** for $\mathcal{C}$.

Get (iii) from [KarpLubyMadras89].
What about densifier and SQ learning?

Sketch of the densifier for DNFs

- Consider a DNF $f = T_1 \vee \cdots \vee T_s$. For concreteness, suppose each $|T_i| = \sqrt{n}$.
- Key observation: for each i, $\Pr_{x \sim U_{f^{-1}(1)}}[T_i(x) = 1] \geq 1/s.$

  So Pr[$2 \log n$ consecutive samples from $U_{f^{-1}(1)}$ all satisfy same T_i] is $\geq 1/s^{2 \log n} = 1/n^{2 \log s}$.

- If this happens, whp these $2 \log n$ samples **completely identify** T_i
- The densifier finds candidate terms in this way, outputs OR of all candidate terms.

SQ learning for DNFs

- Unlike halfspaces, no efficient SQ algorithm for learning DNFs under arbitrary distributions is known; best known runtime is $2^{\tilde{O}(n^{1/3})}$.

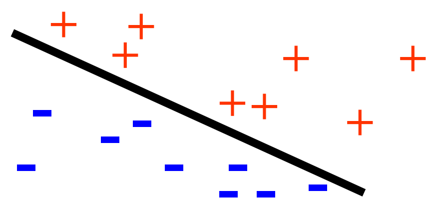
- But: our densifier identifies $n^{2\log s}$ "candidate terms" such that f is (essentially) an OR of at most s of them.

- Can use **noise-tolerant SQ learner for sparse disjunctions**, applied over $n^{2\log s}$ "metavariables" (the candidate terms).

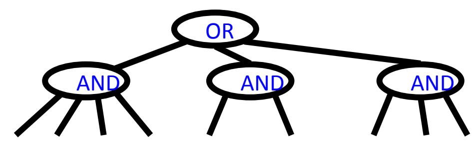
- Running time is poly(# metavariables).

Summary of talk

- New model: Learning distribution $U_{f^{-1}(1)}$
- "Multiplicative accuracy" learning
- Positive results:

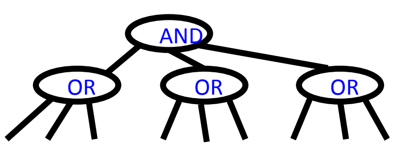


Halfspaces

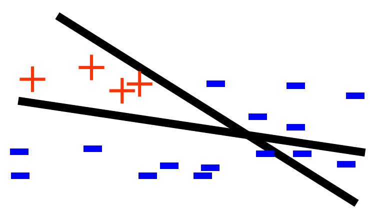


DNFs

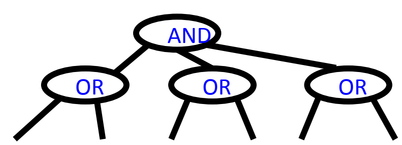
- Negative results:



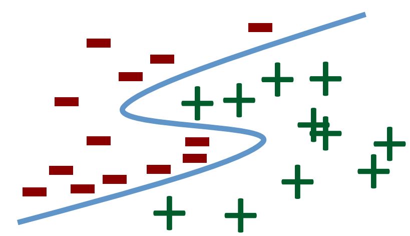
3-CNFs



Intersection
of 2 halfspaces



Monotone
2-CNFs



Degree-2 PTFs

Future work

- Extensions to agnostic / semi-agnostic setting?
- Other formalizations of "simple distributions"?
- Beating the "curse of dimensionality" for d-dimensional histogram distributions?

Thank you!

Hardness results

Secure signature schemes

- G: (randomized) **key generation algorithm**; produces (pk, sk) key pairs
- S: **signing algorithm**; $\sigma = S(m, sk)$ is signature for
  $m$ssage sk
  V using secret key . $V(m, pk, \sigma) = 1$
- $\sigma = S(m, sk)$ **algorithm**; if

$(m_1, \sigma_1), \ldots, (m_t, \sigma_t)$

Security guarantee: Given signed mess(m', σ') ,
$V(m', pk, \sigma') = 1$thm can prm'ice such that
for a new

Connection with our problem

Intuition: View $U_{f^{-1}(1)}$ as uniform distribution over **signed messages** $S(m, sk)$.

If, given signed messages, you can (approximately) sample from $U_{f^{-1}(1)}$, this means you can generate new signed messages – contradicts security guarantee!

Need to work with a refinement of signature schemes – unique signature schemes [MicaliRabinVadhan99] – for intuition to go through.

Unique signature schemes known to exist under various crypto assumptions (RSA', Diffie-Hellman', etc.)

Signature schemes + Cook-Levin

Lemma: For any secure signature scheme, there is a secure signature scheme with the same security **where the verification algorithm is a 3-CNF**.

$f^{-1}(1)$ corresponds to $V^{-1}(1)$, so security of signature scheme → no distribution learning algorithm for 3-CNF.

More hardness

Same approach yields hardness **for intersections of 2 halfspaces** & **degree-2 PTFs**. (Require parsimonious reductions, efficiently computable/invertible maps between sat. assignments of $\mathcal{C}$ and sat. assignments of 3-CNF.)

For **monotone 2CNFs**: use the "Blow-up" reduction used in proving hardness of approximate counting for monotone-2-SAT. Roughly, most sat. assignments of monotone-2-CNF correspond to sat. assignments of 3-CNF.