Binary classification and Queries: Theory

vishvAs vAsuki

November 29, 2010

Contents

Based on [1], courses by Adam Klivans, Pradeep Ravikumar.

Part I

Introduction

1 Themes

Consider the classification task described in the statistics survey. Computational learning theory is about binary classification problem, and getting the best classifier using 0/1 loss.

Hardness of (accurately/ approximately) learning a concept in a concept class in terms of time, space, sample size: Upper (**Efficient Learning Algorithm design**) and Lower bounds. Doing this: Agnostic or gnostic of distribution; In the presence or absence of noise (in label or in input); relative to

ease of evaluation of a concept; Proper learning vs learning using a different Hypothesis concept class.

What concept classes are more general?

How many mistakes does an algorithm make whilst efficiently learning a concept class? Learning using bunch of experts.

Judge the importance of problem/ model: Consider real-life learning scenarios where they are useful. Consider if lower / upper bound for some important scenario may be obtained.

1.1 Interplay with other areas.

Observe similarities between compression and learning: learning which is more than mere memorization usually (kNN being an example to the contrary) involves learning some concise, general things from the examples.

Cryptography and learning: opposites.

1.2 Folk theorems

The only C we know to learn are polynomials (Eg PTF).

1.3 Characterization of research effort

See computational complexity theory ref and algorithms ref; emphasis is on the former.

1.4 Notation

1.4.1 The sample

Universal set of possible inputs X. An example := ($x \in X$, label l). Label is $\{1, 0\}$ or $\{1, -1\}$ depending on context. Bits in x: $x_1 \dots x_n$. Sample set S, $|S| = m$

1.4.2 Concepts and their sets

Target concept (classifier function) in a certain representation: c; Concept class C; Concept representation class R, representation dimension (f(input size)) or number of binary vars in x: n, sampling distribution D; hypothesis concept h, hypothesis class H.

$|H_{n,m}|$: H restricted to
algorithms with m n-dimensional examples. [**Check**]

1.4.3 Learning algorithm

Learning algorithm L, attribute/ variable list V.

1.5 The binary classifier c

For different ways of looking at c, see boolean fn ref. For VCD proofs/ measuring complexity of representation classes, it is profitable to view a hypothesis as the dichotomy it induces on X.

2 Representation classes of classifiers

2.1 Properties of representation classes

2.1.1 Error regions

Class of error regions: $\Delta(c) = \{c\Delta h]\}$.

2.1.1.1 epsilon net of examples

ϵ **net**: A set of examples which hits all error regions ($\Delta_\epsilon(c)$) of weight $\geq \epsilon$ under D. A bad hypothesis h: $c\Delta h \in \Delta_\epsilon(c)$.

2.1.2 Monotonicity

To understand monotonicity of boolean functions, see boolean functions reference.

Transformation to non-monotone R by introducing n new literals, if n finite.

2.1.3 Closures

2.1.3.1 Projection closure

Take partial assignment P; let P/A: an assignment to V first using P, then using A for unassigned vars. Projection of an example using P. Projection closed concept class: Take any f in C, fix values of some variables using P, get another $f_P = f(P/A)$ in C.

2.1.3.2 Embedding closure

Embedding one domain of variables into another; embedding a concept into a domain of variables. Embedding an example. Embedding closed concept class: Take f, rename variables, add irrelevant variables to the domai$n$, f is still in C.

2.1.3.3 PEC classes

Most useful concept classes are projection and embedding
closed (pec), or are contained in some projection and embedding closed C.

2.2 Some representation classes

2.2.1 Geometric classes

Also see boolean functions ref for halfspaces etc.. Axis aligned rectangles (good basic example).

2.2.2 Boolean functions

See boolean functions ref: k-DNF. k-CNF, Conjunctions. Disjunctions. Decision lists. Decision trees. T-augmented decision trees. Polynomial size DNF. D-PTF. Halfspaces: common benchmark for machine learning algs. Intervals in R. Polyhedra in R^n. The class of parity functions with length k: PARITY(k).

2.2.3 Classes of Continuous fns

See complex analysis ref.

2.2.4 Restricted concept classes

C_n: concepts depending on a set of n vars. $C_{r,n}$ Concepts in C_n with at most r relevant variables.

2.2.5 Some PAC Reductions

k-CNF to Conjunctions. k-decision list to 1-decision list.

Boolean formula to Log-space turing machine: Parse the circuit, Have identifier for each node.

Log-space turing machine concept class (max klogn space) to Deterministic Finite Automaton (DFA) concept class: DFA state = <state, possible tape content, head location>, Transitions = L/R, $x \rightarrow x.x...x$.

2.3 Measure complexity of C

Consider n, size(c) ($\approx \log|C|$). Find VCD; Plot dichotomy count $D_C(m)$: these are described in the boolean functions survey. Find sample complexity.

2.3.1 SQ dimension (sqd or d)

2.3.1.1 Definition

Take c:
$\{-1,1\}^n \rightarrow \{-1,1\}$, $C = \{c\}$. $sqd_D(C) = \max d | \exists S_D = \{f_1,..,f_d\} \subseteq C$ with $|corr_{x\sim D}(f_i f_j)| = |E_{x\sim D}(f_i(x)f_j(x))| \leq d^{-1}$;
$sqd(C) = \max_D sqd_D(C)$.

Applications: Learning in the presense of noise.

2.3.1.2 Bounds on SQD

Max sqd = max $|C| = 2^n$. Parity fn $p_S(x) = \prod_{x_i \in S} x_i$; $C = \{p_S : |S| \leq n\}$ has max sqd: Sets S, T differ in some $x_d : E_{x_i \sim U}[p_S(x)p_T(x)] = E[x_d]E[..] = 0$.

$sqd(C) = \Omega(VCD(C))$: Construct the set $\{f_i\}$: Take uniform distr on the shattered set; select two fns $\{f_i\}$ solving it for set of 2 examples $\{x_i\}$; assume for 2^i; make $\{f_i\}$ vs $\{x_i\}$ sign matrix M; get matrix for 2^{i+1}: $\left[\begin{smallmatrix} M & M \\ M & -M \end{smallmatrix}\right]$; extend proof for those between 2^{i-1} and 2^i by using matreces for powers of 2.

$sqd(C) = 2^{O(VCD(C))}$. [**Find proof**]

2.3.1.3 Other properties

All in the corr-shattered set $S_D = \{f_i\}$ linearly independent. Proof by contr: If d is sqd, $c_i f_i = 0$, take $c_m = max(\{c_i\})$, so $f_m = \sum_{i \neq m} \frac{c_i}{c_m} f_i$, so $1 = E[f_m f_m] = \sum_{i \neq m} \frac{c_i}{c_m} E[f_m f_i]$; but $\frac{c_i}{c_m} \in [-1, 1]$, $E[f_m f_i] \leq d^{-1}$, so absurdity.

2.3.2 Other measures

Unique -ve dimension: UND(C) = Max Subclass $C' \in C$ with unique -ve examples.

Can use covering and packing numbers. These measures of the size of sets are described in the topology ref.

3 Resources used by the learner

3.1 Queries and oracles

An oracle provides examples according to D. A teacher answers membership and equivalence queries.

Example oracle $EX_D(c)$.
Membership query (mq): mq(x) yields c(x). Equivalence query (eq): eq(h) yields counterexample. MQ(c), EQ(c) oracles.

$EX_U(c)$ can be simulated using MQ(c). In PAC setting, EQ(c) can be simulated by the EX oracle.

3.1.1 Probabilistic oracle

PEX(f) for fn $f : \{0,1\}^n \to [-1, 1]$: produces (x, b) where $x \sim U$, b is ± 1 binary RV with mean f(x). For boolean f, this is UPAC oracle EX(f, U). Models random noise under U: PEX(tf) is an oracle for f(x) with random noise of rate $1/2 - t/2$: simply see how to make binary RV b with mean tE[f].

Statistical dist: $D(PEX(f), PEX(g)) = E_x[|f(x) - g(x)|] \leq \|f - g\|$. So, upon querying, an algorithm notices difference between PEX(f) and PEX(g) with prob $\leq \|f - g\|$. (See probability ref.)

3.2 Non adaptive mq (namq)

Points on which mq's are made do not depend on concept being learnt. So, many concepts can be learnt from same set of query points: like the brain. Also, easier parallizability.

3.3 Relative power of various queries

As they solve iterated products problem, polynomial sized circuits are not learnable using random examples. By modifying the circuits to encode the ckt at certain fixed points, ye get a class learnable with namq but not with random examples alone. By placing the encoding at varying locations, but encoding the location of the encodings at certain fixed points, ye get a class learnable with mq, but not with namq.

3.4 Attribute efficient (ae) learning

r: Number of relevant attributes.
Alg is poly time, calls to oracle is $p(r, |c|)h(n), h(n) = o(n)$.

Strong ae learning: $h(n) = O((\log n)^k)$. Or, calls to oracle polynomial in $|c|$, not in n.

Part II

Complexity of classification

4 Learning the best hypothesis

Aka Agnostic learning model.

4.1 The goal

Let error in best fit, $\eta = min_{c \in C} Pr(c(x) \neq l)$. Given δ, ϵ; get h where $Pr(h(x) \neq l) \leq \eta + \epsilon$.

4.1.1 Underlying distribution

Naught known to the algorithm about process generating data; just asking for approximately best fit or ϵ approx of the c with error η. Even same example s can have different label l at different times: this could either be because of noise/ corruption of the label or because of the underlying distribution.

In some settings, it may be clear that the underlying $Pr(x)$ distribution is of a particular type. This setting is important in practice.

4.2 Connection to PAC model

PAC model a special case. So, lower bounds which apply to PAC learning apply to agnostic learning too.

4.3 Common algorithms

Very few positive results known. Usually empirical risk minimization used.

4.3.1 Under U, learn f with Fourier concentration

If target $f \in C$ has good fourier concentration:
$\sum_{|S|\geq d} \hat{f}(S)^2 \leq \epsilon$; for best $g \in H$: $Pr_x(g(x) \neq f(x)) = \eta$; then use low degree algorithm to learn g: get $h = \sum_{|S|<d} \hat{g}(S)p_S$; then error of h wrt g:
$\left\|\sum_{|S|\geq d} \hat{g}(S)p_S\right\|^2 \leq \left\|g - \sum_{|S|<d} \hat{f}(S)p_S\right\|^2 \leq \|g - f + f_{\geq d}\|^2 \leq O(\eta+\epsilon)$; final error of h wrt f also $O(\eta + \epsilon)$.

4.3.2 Learning parities

Equivalent to decoding random linear codes from adversarial errors: $xG = c$; make H in $N(G^T)$; $(c+e)H = eH$. Finding e from eH is equivalent to learning noisy parities. NP hard.

If possible, enables weak parity learning.

4.3.2.1 Reduces to learning with random noise

Let f have t-heavy component s. This is also a weak parity learner.

Let A-projection of f: $f_A = \sum_{a\in\{0,1\}^n : Aa=1^m} \hat{f}(a)p_a(x)$: So, picking 2^{-m+n} basis parities. $f_A(x) = E_{p\in\{0,1\}^m}[f(x \oplus A^T p)p_{1^m}(p)]$. Thus, Can simulate $PEX(f_A)$ with PEX(f).

Take [-1, 1] ranged f, $s \neq 0$, random A: with probability 2^{-m-1}, $\hat{f}_A(s) = \hat{f}(s)$ and $\sum_{a\neq s} \hat{f_A}(a)^2 \leq 2^{-m+1} \|f\|^2$: latter by finding expectation, using Markov inequality.

Algorithm WP: Take learner L for parities over k vars for every noise $\eta < 1/2$ in time $T(\mathrm{n}, k, (1-2\eta)^{-}1)$ and $S = S(\mathrm{n}, k, (1-2\eta)^{-}1)$ examples. Take

PEX(f); simulate $PEX(f_A)$ which $\approx PEX(\hat{f}(a)(s)p_s(x))$ which is s with noise $\eta = \frac{1-t}{2}$; run L to get parity s, output s if s is $t/2$ heavy.

Statistical distance $\Delta(PEX(f_A(x)), PEX(\hat{f}(a)(s)p_s(x)) \leq 1/(2S)$; L uses S examples; so Probability that A notices difference between these is at most $1/2$.

Also, f has at most $u = 1/t^2$ t-heavy coefficients: so, by coupon collector, with $u \log u$ repetitions, identify all t-heavy coefficients.

Can learn t-heavy p_s for f with range beyond $[-1, 1]$, given oracle $EX(f)$ with correct expected value for $f(x)$: scale f by $\|f\|_\infty$; learn $\frac{t}{\|f\|_\infty}$ correlated parity.

4.3.2.2 Learn DNF, k-juntas

For DNF, use p-smooth booster. Distributions D generated by booster; learning parity p_s correlated with f under D is same as learning correlated parity for $2^n D(x)f(x)$; scale this: $F(x) = \frac{2^n D(x)f(x)}{\|2^n D(x)\|_\infty}$; use oracle $EX_D(f(x))$, add noise to get $PEX(F(x))$.

Learning k-juntas: A clean formulation of the problem of efficient learning in the presence of irrelevant attributes. k-junta has at most 2^k non 0 coefficients; Each of them is 2^{-k+1} heavy (See boolean fn ref); use WP to learn full spectrum.

Noise tolerance: Oracle for f with noise η is same as $PEX((1-2\eta)f)$. Use this oracle in above algorithm to get oracle $PEX((1-2\eta)f_a p_a)$, use it as above.

4.4 Useful relations

$(1-x) \leq e^{-x}$. To solve $am^b \leq 2^{cm}$ for m: make both sides look alike. To simplify $f(x) - f(x+t) = ctf'(x)$.

5 PAC learning

5.1 The goal

5.1.1 Distribution and the oracle

Distribution agnostic. Assumption: training distribution = testing distribution. 2 Oracle PAC model variant: Algorithm can use both D_c^+ and D_c^-, $error \leq \epsilon$ wrt both.

EQ oracle can be simulated by the EX oracle.

5.1.2 Strong PAC learning

Given ϵ, δ, C, we want to find $h \in C$ such that, with probability $1-\delta$, $Pr(c(x) \neq h(x)) \leq \epsilon$.

5.1.3 Weak PAC learning algorithm

p, q polynomials; $\epsilon \leq 2^{-1} - g = 2^{-1} - \frac{1}{p(n,size(c))}$, $\delta \leq \frac{1}{q(n,size(c))}$. g: the advantage of L over random guessing. Minimally better than random guessing: any g subexponential, can be achieved by memorizing previously seen examples.

h is weak hyp for c iff:
$Pr_{x\sim D}[h(x)f(x) = -1] \leq 2^{-1} + g$; or $E_{x\sim D}[h(x)c(x)] \geq 2g$.

5.2 PAC learning model

5.2.1 Make PAC learning algorithm

For C using H (containing C) with m finite: Cast a **multivalued discrete classification function** into PAC model: Make a binary classifier concept for each bit.

Decide initial h (maybe null, universal); decide update to hypothesis corresponding to example provided by the sampling oracle : check if +ve examples enough.

If H finite: Create an Occam algorithm; maybe use size(h) to bound $|H|$; use Occam's razor. If H finite: Create an Occamish algorithm inconsistent with at most $\epsilon|S|/2$ examples; maybe use size(h) to bound $|H|$; use Chernoff to bound $Pr(\exists$bad h; $|[c\Delta h] \cap S| \leq m\epsilon/2)$; thence continue Occam razor proof.

Try to find an algorithm which will grow the hypothesis decision list efficiently, as if solving a set cover problem.

Any $|H|$: Make algorithm to find h consistent with S, use VCD - Occam razor.

Make an algorithm to learn C using only EQ(c), simulate EQ using EX.

5.2.2 PAC reduce C over X to C' over X'

Efficiently change $x \in X_n \rightarrow g(x) \in X'_{p(n)}$; so that there exists proper image concept c' for every c with size(c') = q(size(c)).

5.2.3 Prove efficient PAC learnability

Make PAC algorithm, bound m for (ϵ, δ) approximation of c; prove that running time is polynomial in $(\epsilon^{-1}, \delta^{-1}, size(c)$,m, $n)$.

Reduce to a known efficient PAC learning algorithm.

Show efficient learnability in the mistake bound model; bound m using Occam razor or VCD Occam razor.

Find weak learner; boost.

If learning on uniform distribution, see U distribution Fourier analysis section.

5.2.4 Lower bound on sample complexity in PAC

5.2.4.1 Shatter dimension

m = $\Omega(\frac{d}{\epsilon})$.

[**Proof**]:Take S shattered by C', let $|S| = d$ and distribution $D = U$ over S. Then, we show that no algorithm L can ensure ϵ error whp. As D is the support of C', without loss of generality, we will consider C to be the finite set of 2^d concepts with different dichotomies induced on S.

Draw S' with $|S'| = \frac{d}{2}$; suppose that L learns h. $h = f_L(S')$. But there are $|2^{S-S'}| = 2^{d/2}$ different concepts c consistent with the labellings in S' and so, h is chosen independently of these different concepts. Does h have low error wrt all of these?

We want to show that there exists a c such that the h learned would have a high error rate with non-negligible probability. We will do this by considering the following process: Fix S and h, pick $c \sim U(2^{S-S'})$. We then show that $E_c[Pr_x(h(x) \neq c(x))] > 1/4$, which implies the existence of c for which h is a bad approximation.

$E_c[Pr_x[h(x) \neq c(x)]] = E_c[Pr_D(x \in S - S')Pr_c(h(x) \neq c(x)|x \in S - S')] \geq 2^{-1} \times 2^{-1}$. Thus, there exists a c, for which the h learned using S' as the sample would be bad.

We then need to show that L cannot find a good hypothesis for this c with high probability. [**Incomplete**]

5.2.4.2 From halving algorithm

Halving algorithm L mistakes in MB model is an approximate lower bound: take set S on which L makes mistakes, make uniform distr on S, use halving algorithm in PAC model, then error probability is $(|S| - m)/(2 \log |S|) \leq \epsilon$, so $|S|(1 - 1/(n^k)) \leq m \forall k$.

5.2.5 PAC with mq

Any R efficiently PAC learnable using mq's and eq's is efficiently learnable with mq's only.

5.3 PAC learnability of some R

Also see list of R which are PAC learnable with noise, learnable in MB.

5.3.1 Halfspaces

$f = sgn(w.x)$, $\forall$x, $\langle w, x \rangle \neq 0$, $\exists x_i : |E_{x \sim D}[f(x)x_i]| \geq W^{-1}$; so some x_i weak hypothesis; then use ADAboost.

5.3.2 Parities

Use the MB algorithm with Occam razor.

ae learning of parities wrt Uniform distribution is equivalent to decoding high rate random linear codes from low number of errors. So, believed to be hard.

5.4 Approximate f with orthogonal polynomials

Aka Fourier coefficients. Must have orthogonal polynomials for D.

5.5 Uniform distribution PAC learning (UPAC)

Easier than learning under general distributions: can use Fourier analysis.

Harder than learning Gaussian distr. [**Find proof**]

[**OP**]:What is the lower bound for UPAC learning? Does ae learning DL' s make sense?

5.5.1 Shortness of DNF term length, Decn tree depth

DNF term-length or 1-DL size is limited to $\log \frac{1}{\epsilon}$: Probability of longer term being satisfied $< \epsilon$.

5.5.1.1 Learn DNF

Collect all terms of size $log(s/\epsilon)$ consistent with target DNF: use feature expansion, disjunction learning winnow algorithm.

5.5.2 The function space

Basis of the input space: $\{\pm 1\}^n$. Basis of the function space: Parity functions p_S. $f = \sum_{S \subseteq [n]} \hat{f}(S) p_S$. See Boolean function ref for details.

5.5.3 Alg to learn Fourier coefficient

Learn $\hat{f}(S)$
using $\hat{f}(S) = E_{x \in_U \{\pm 1\}^n}[f(x)p_S(x)]$: Choose sample of size m; by Chernoff $m = \Omega(\frac{\log \delta^{-1}}{l^2})$ for (δ, l) approx of $\hat{f}(S)$.

Best fitting polynomial g of degree d: $\sum_{|S| \leq d} \hat{f}(S) p_S$ (See Boolean function ref). $Pr_x(sgn(g(x)) \neq f(x)) = E_x([sgn(g(x)) \neq f(x)]) \leq E_x((f(x) - g(x))^2)$.

5.5.4 Low degree algorithm

If f is (ϵ, δ) concentrated, f learnable wrt U to accuracy 1-a in time $n^{O(\delta)}$: Sample to estimate $\hat{f}(S) : |S| \leq \delta$, make g, output $sgn(g)$. Agnostically learning big correlated parities.

5.5.4.1 Learning some R under U

Polysize DNF formulae learnable in
$n^{O(\log|c|\log\frac{1}{\epsilon}))}$. Halfspaces learnable in $n^{O(\epsilon^{-2})}$. Any function of k halfspaces learnable in $n^{O(k^2\epsilon^{-2})}$. [**OP**]:Do better for $\cap$ of k halfspaces.

Depth d ckts of size $|c|$. Also Decision trees. [**Find proof**]

5.5.5 Based on total influence

f learnable in $n^{O(\frac{d}{\epsilon})}$ to accuracy $1-\epsilon$. [**Find proof**]

So, monotone f learnable in $n^{O(\frac{\sqrt{n}}{\epsilon})}$.

5.5.6 Hardness of ae learning of PARITY(k)

This is equivalent to decoding high rate random linear code C from low number of errors, a hard open problem. Also equivalent to learning PARITY with noise, in simpler case where noise is random and independent.

If you can decode C, you can ae learn PARITY: Simply take the examples $\{x_i\}$ you get, turn it into H, get a random G from $null(H^T)$, give it to the decoder along with the syndrome $\{p_e(x_i)\}$, return hypothesis p_e.

If you can learn PARITY(k) using algorithm L, you can learn it properly whp. Take the hypothesis h returned by L, turn it into an mq oracle which evaluates h(y) by taking h(x+y)+h(y) over many x. Thence make hypothesis parity p attribute efficiently.

By properly ae learning parity you can decode random linear code: Given G and y, make H and yH, use learner.

5.6 UPAC with mq

5.6.1 Learning PARITY(k) using namq

Let H be check matrix for a code C; corrupted code y = c+e for some code c and error e with $wt(e) < w$. Take BCR decoding algorithm (see coding theory ref) A which accepts y and the syndrome yH = eH, and returns e. So, using namq's on the columns of H, A learns unknown $e \in PARITY(k)$ attribute efficiently. Another view: using e, A learns parity c using noisy namq's.

5.6.2 Sparse approach

For t-sparse f; or $\frac{\|f\|_1^2}{\epsilon}$ sparse g: approximator for f.

Thus, using K-M algorithm, f is learnable to accuracy $1-\epsilon$ in time $poly(\mathrm{n}, \|f\|_1, \epsilon^{-1})$ using membership queries: find coords of g = coords of f of size $\geq \frac{\epsilon}{\|f\|_1}$; use sgn(g) as hypothesis.

5.6.3 Weak parity learning

Given f with some t-heavy coefficient, given MQ oracle, find vector z with t/2 heavy $|\hat{f}(z)|$. $|\hat{f}(y)| \geq t$ iff $Pr(f = p_y) \geq 1/2 + t/2$; so the weakly correlated parity p_y is $1/2 - t/2$ approximator of f, so called weak parity learning.

Can also view this as learning parities in agnostic setting. See agnostic learning section.

5.6.3.1 ae-namq algorithm: Like a sieve

Take an aena-mq parity learning algorithm L. $f_a : f(x \oplus a)$; S: the set of mq points of L. Estimate Fourier coefficients of f with t/4 accuracy. Also, for every y in S, find all Fourier coefficients of $f_{R,y}$ to estimate coefficients of f_y. $\hat{f}_c(a) = \hat{f}(a)p_a(c)$; so $\hat{f}_c(a)\hat{f}(a)$ and $p_a(c)$ have same sign.

Run L for every $z \in 0,1^m$ with $\hat{f}_R(z) \geq 3t/4$, using $\hat{f}_{R,c}(z)\hat{f}_R(z)$ to answer mq's and learn p_a. If $a = zR$, add p_a to a set of possible hypotheses. Repeat the process many times with different R. Really good p_a occur in hypothesis sets at much higher rate. Thus, find a with $\hat{f}(a) \geq t/2$.

Can also be fixed to find t/2 heavy component of randomized fn ψ. Only, m required for good approximation whp depends on $var(\psi)$.

5.6.3.2 K-M algorithm to find big components using membership queries

Let $|a| = k$.
$f_a(x) := \sum_{Z \in \{\pm 1\}^{n-k}} \hat{f}(a, Z)p_{a,Z}(a, x)$.
Then $f_a(x) = E_y[f(yx)p_a(y)]$: Let $Z = Z_1Z_2, Z_1 \in \{\pm 1\}^k$; $E_y[f(yx)p_a(y)] = E_y[\sum_Z \hat{f}(Z)p_Z(yx)p_a(y)] = \sum_{Z_1,Z_2} \hat{f}(Z_1Z_2)p_{Z_2}(x)E_y[p_{Z_1}(y)p_a(y)] = \sum_{Z_2} \hat{f}(aZ_2)p_{Z_2}(x)$.

(Kushilevitz, Mansour). Input t; output all $|\hat{f}(S)| \geq t$ whp. Make tree: at root find $\|f\|^2$; at left child find $\|f_0\|^2$; at right child find $\|f_1\|^2$; if any branch has $\|f_a\|^2 \leq t^2$ prune; repeat.

Leaves of the tree are individual coefficients $\hat{f}(S)$. Height $\leq n$. Breadth $\leq t^{-2}$: every level is a partition of $\left\{\hat{f}(S)\right\}$; t^{-2} of $\{f_a\}$ have $\|f_a\|^2 > t^{-2}$.

To find $\|f_a\|^2 = E_x[f_a(x)^2]$: Find $f_a(x) = E_y[f(yx)p_a(y)]$: pick random y's, use membership queries.

Opcount: poly(t, n).

5.6.3.3 Results

Decision trees with t leaves approximable by t-sparse function; so learnable in polynomial time with membership queries.

5.6.4 Learning DNF

Aka Harmonic sieve. DNFs have a $(2s+1)^{-1}$ heavy component: For any D, there is p_a such that $|E_D[fp_a]| \geq (2s+1)^{-1}$ and $wt(a) \leq \log((2s+1)\left\|D2^n\right\|_\infty)$. [**Find proof**].

$E_D[f(x)p_a(x)] = E_U[f(x)2^nD(x)p_a(x)] = \hat{\psi}(a)$. If ye give oracle access to D, you can evaluate ψ. So, use ae-namq algorithm to learn a with $\hat{\psi}(a) \geq t/2$. As $var(\psi) \leq \|2^nD(x)\|$, m required will depend on this. This is efficient only for D polynomially close to U.

Then boost using p-smooth algorithm.

5.7 Boosting weak efficient PAC algorithm L

5.7.1 Practical applications

Frequently used: boost decision trees.

5.7.2 Boost confidence

Run L k times to gather k hypotheses (Get ϵ approx with prob $1-(1-\delta_0)^k$), select best h by using many examples to gauge $h\Delta c$.

5.7.3 Boosting accuracy

So you are using the guarantee that polynomial time L works with any distribution to produce a hypothesis which produces very accurate results by playing with input distribution.

General idea for all techniques: put more weight on x on which weak hyp h makes mistake; rerun weak learner.

5.7.3.1 Lower bound

Number of iterations to get ϵ accurate using γ weak learner is $\Omega(g^{-2}\log(\frac{1}{\epsilon}))$.

5.7.4 Properties of booster

Noise tolerance. Smoothness: How far does the artificially generated distribution deviate from original?: Affects speed.

5.7.5 Boost accuracy by filtering

Reweight points using online algorithm.

5.7.5.1 Majority tree of Hypotheses Booster

(Schapire). Error b bosted to $g(b) = 3b^2 - 2b^3$: Run L to get h_1; filter D to get D_2 where h_1 is useless like random guessing; run L to get h_2 - learn

something new; filter D to get D_3 where $h_1(x) \neq h_2(x)$; run L to get h_3; return $majority(h_1, h_2, h_3)$. Gauge errors of h_1 and h_2 to ensure efficient filterability.

Recursive accuracy boosting For target accuracy b and distribution D: Maybe invoke self thrice for accuracy $g^{-1}(b)$ and distributions D, D_2, D_3. Final h like 3-ary tree; only leaves use samples for learning; others sample for error gauging.

5.7.6 Boost accuracy by sampling

Reweight points using offline algorithm.

Get sample S; boost accuracy till h consistant with S; bound size of h; bound $|S|$ using Occam razor.

5.7.7 Adaptive booster (AdaBoost)

Get sample S; use U distr on it. At time t=0: wt on s_j is $w_j = 1$; $W_0 = \sum w_i = m$; prob distr $D(x_i) = \frac{w_i}{W}$. On iteration i: run L, get hypothesis $h_i : X \to \{\pm 1\}$, $err(h_i) = E_i$, accuracy $a_i = 1 - E_i$, $\beta_i = \frac{E_i}{a_i}$; for each s_j where h_i is correct, reduce their weight: $w_j := w_j \beta_i$. Output after time T: $sgn(\sum_i a_i h_i - 2^{-1})$; $a_i = \frac{\lg \beta_i}{\sum_j \lg \beta_j}$. If $E = \max_i E_i$, $\beta = max_i \beta_i$, this is $sgn(T^{-1} \sum_i h_i - 2^{-1})$.

Final error $E_T \leq e^{-2Tg^2}$: $W_1 = m(2E); W_T = m(2E)^T$. Weight of x_i misclassified at iteration T $\geq \beta^{T/2}$: At $\geq \frac{T}{2}$ of $\{h_i\}$ could've been right on it. Total wt on misclassified points $\leq \epsilon \mathrm{m} \beta^{T/2} \leq m(2E)^T$; so $\epsilon \leq (\frac{4E^2}{\beta})^{T/2} = (4E(1-E))^{T/2} = (4(4^{-1} - g^2))^{T/2} \leq exp(-2g^2 T)$.

$e^{-2Tg^2} < m^{-1}$ when $T > \frac{\ln m}{2g^2}$. So, final $|h| = O(\frac{\ln m}{2g^2})|c|$.

5.7.7.1 Contrast with maj tree booster

Gives simple hyphothesis; takes advantage of varying error rates of weak hypotheses. [**Find proof**]

5.7.7.2 Noise tolerance properties

Sensitive to noise even if L resistant to noise: Distributions created by looking at actual label.

5.7.8 Boosting by branching programs

h = Branching program: DAG of hypotheses $\{h_i\}$. Labels x according to leaf reached. Run weak learner, get h_0. From distr $D_{h_0^-}$ and $D_{h_0^+}$, get $h_{1,1}$ and $h_{1,2}$. From $D_{h_{i,1}^+}$ get $h_{i+1,1}$; from $D_{h_{i,j}^+} \vee D_{h_{i,j+1}^-}$ get $h_{i+1,j+1}$: diamonds in the DAG; from $D_{h_{i,i+1}^+}$ get $h_{i+1,i+2}$.

Assume that both $err_{D_{c^-}}(h_{i,j})$ and $err_{D_{c^-}}(h_{i,j}) \leq 2^{-1} - g$: can be removed. [**Find proof**]

At node (j,k); k-1 hypotheses have said h(x)=1; j-k+1 hypotheses have said h(x)=0. So, in final level T; first T/2 leaves labelled 0; rest labelled 1.

Take x with c(x)=1: $h_i = h_{i,j}$ seen in iteration i: In worst case, $E_x[[h_i(x) = 1]] = 2^{-1} + g$; so E_x[leaf where x ends up] $= \frac{T}{2} + gT$. A biased random walk.

Show $Pr_x(h(x) = 1) \geq 1 - \epsilon$: due to lack of independence of events h(x) = 1, martingale is the right tool; let $X_i = [h_i(x) = 1]$, $Y = \sum_{i=1}^{T} X_i$; $Y_i = E[Y|X_1, ..X_i]$ is a Doob martingale with $Y_i - Y_{i-1} = 1$; $Y_i = \sum_{j=1}^{i} X_i + \frac{T-i}{2} + g(T-i)$; by Azuma: $Pr(|Y - E[Y]| \geq gT) \leq 2e^{-g^2T} \leq \epsilon$ for $T \geq \frac{10 \log \frac{1}{t}}{g^2}$.

Resistant to noise if L resistant to noise: Boosting program creates distributions without looking at $c(x)$.

5.7.9 Consequences

PAC algorithm memory bound: $O(\frac{1}{\epsilon})$. [**Find proof**]

5.8 Hardness of PAC learning

5.8.1 General techniques

Specify distr where learning hard.

Any C which includes functions which can be efficiently used to solve iterated products [cryptography ref] is inherently unpredictable.

PAC-reduce a C' known to be hard to C.

Show that VCD is ∞. Eg: $C = \{convex\ polygons\}$.

Reduce it to a hard problem in another area: eg: decoding random linear codes.

5.8.1.1 If RP neq NP

By contradiction: Take NP complete language A; efficiently reduce input a to labelled sample set S_a where $S_a \equiv$c$\in C$ iff $a \in A$; assume efficient PAC learning algorithm L; with suitable $\epsilon \stackrel{?}{=} |S_a|^{-1}$, use L with timer to learn ϵ close h whp; see if h is consistant with S_a; if yes, $a \in A$; otherwise $a \notin A$; we have an RP algorithm for A!

5.8.2 Hardness results for proper learning

$k \geq 3$: k-term DNF learning intractable (H is restricted to equal C), but predicting classification accurately tractable: Hardness by reduction to graph coloring problem [**Find proof**]; k-term DNF = some k-CNF by distr law: so improper learning using k-CNF as H is easy. So, conversion from k-CNF learnt to k-term DNF is hard.

Usually the $RP \neq NP$ assumption is used.

5.8.3 Cryptographic hardness results for improper learning

Aka inherent tractable-unpredictability. Learning hard despite: Polynomial examples being sufficient; or concept evaluation easy.

5.8.3.1 C which include concepts which can find Discrete cube roots

Take n bit N's. Make discrete cube root hardness assumption [see cryptography ref]. Consider uniform distr over Z_N^*; any algorithm can generate examples by itself - so doesn't learn anything new; otherwise contradicts assumption. Learning remains hard even if supplied n repeated squares of a number y mod N : no clue about d in this $O(n^2)$ input.

By PAC reduction, anything which can compute iterated products is hard to learn.

5.8.4 C which include concepts to find ith bit of BBS pseudorandom generator outputs

$f_{s_0,N,t}(i)$: ith output bit of BBS pseudorandom generator with seed s_0 and N, if $i \leq t$ (see randomized algs ref). If $\exists\ O(n^{ck})$ time algorithm A to learn C with error $\leq 2^{-1} - n^{-k}$; you can distinguish BBS from random string b of $n^{(d+1)ck}$ bits, where $d \in Z, dc \neq 1$; and thence factor N.

5.8.4.1 Distinguisher D

Input random string; Use Uniform distr over b; use (i, b_i) as examples; learn with n^{ck} examples; get hypothesis h with error $\leq 2^{-1} - n^{-k}$; Pick another bit index j; try predicting b_j; if guess correct, output 1 or 'generator'. On truly random b, $Pr(D^{rand} = 1) \geq 2^{-1} + \frac{n^{ck}}{n^{(d+1)ck}}$; but $Pr(D^{f_{s_0,N,t}} = 1) \leq 2^{-1} + n^{-k}$: difference not negligible.

By PAC reduction, anything which can compute iterated products is hard to learn.

5.8.5 Classes which include RSA decryptors

$C = \{f_{d,N}\}$, with private key d, public key e: see Cryptography ref. Not learnable in polytime: else make examples over $U(\{0,1\}^n)$: $(x^e \mod N, x)$; learn and violate RSA unbreakability assumption.

5.8.6 Inherently tractably-unpredictable classes

Polynomial sized boolean circuits of depth log n and n^2 inputs: Can solve Iterated products with ckt of depth $\log^2 n$ (using binary tree like mult ckt). Beame et al: even with log n deep ckt. So, Poly-sized Boolean formulae inherently unpredictable.

Also, Neural networks of constant depth with boolean weights:
(Reif). By PAC reduction from Poly-sized Boolean formulae, Log-space turing machines and DFA inherently unpredictable (but DFA learnable with membership queries).

Intersections of halfspaces. [**Find proof**]Agnostically learning single halfspace. [**Find proof**]

[**OP**]:DNF formulae (can learn under U if RP= NP) : hard to learn? : too weak to encode pseudorandom generators, many cryptographic primitives; if you unconditionally show DNF hard to learn under U, you show $RP \neq NP$!

6 Mistake bounded models

6.1 Mistake bound (MB) learning model

6.1.1 Problem

L learns C: L given sample point x, returns h(x), told if it is correct; this is repeated; has mistake bound m (over any sequence of examples).

6.1.1.1 Adversarial nature and randomization

The mistake bounded model is adversarial in nature - we are concerned with the number of mistakes made in the worst case. So, randomization helps : the adversary decides on the input before the coin is tossed, so not knowing the algorithm's output, its attempts to cause mistakes are less successful.

6.1.2 Traits of MB learners

6.1.2.1 General traits

Any L with finite mb can be written as sober algorithm L' which makes its hypothesis consistant with all examples seen so far: Otherwise, can feed inconsistant hypothesis repeatedly, and L would exceed mb.

Careful algorithm L_c: updates hypothesis only if it makes mistake. L without careless updates is L_c, which has same mistake bound: Else, if L_c makes m+1 mistakes, L would exceed mistake bound m on that sequence.

6.1.2.2 Efficient learnability in MB model

Show runtime per trial = poly(n, size(c)), show polynomial mistake bounds. Reduce to efficiently learnable problem.

6.1.3 Learnability in EQ only model

MB oracle can be simulated using an EQ oracle.

6.1.4 Learnability in PAC model

Any C which is efficiently learnable in the MB model is PAC learnable. We use this PAC algorithm: Take sober algorithm, get hypothesis consistant with large enough sample S drawn from the given distribution D.

Or translate to EQ algorithm, then convert to PAC algorithm.

6.1.5 Lower bound

$mb = \Omega(VCD(C_{r,n}))$: Else you'd be able to learn in the PAC model with an impossibly low sample complexity.

6.1.5.1 Halving algorithm

For $|C|$ finite: At max $O(\log |C|)$ mistakes needed, ignoring efficiency: L replies to x with $maj(h_1(x), \dots)$: falsifies atleast $1/2$ the concepts with every mistake.

mb of Halving algorithm is a lower bound: no algorithm can make better use of sample [**Check**]. $mb = \Omega(|C|)$ not lower bound: Take k point functions; halving algorithm has mb = 1, not log k.

[**OP**]:Compare halving algorithm running time with VCD.

6.1.6 Make Mistake bounded learning algorithm for C using H

Decide initial h (maybe null, universal); decide update to hypothesis corresponding to example provided by the sampling oracle: Eg: Learn Disjunctions with mistake bound 2n.

6.1.7 Find mistake bound

Enumerate cases where algorithm makes mistake; find max num of improvements to h possible in each case.

Start with some money; show that you never get to 0.

[**OP**]:In MB model: Does ae learnability imply strong ae learnability?

6.2 Mistake bounds (mb) for some R

6.2.1 Disjunctions (so Conjunctions)

n vars, k terms.

6.2.1.1 Simplified Winnow

Mb = O(k log n), runtime O(n).

Algorithm: Initial h is the sign of the halfspace $f = \sum x_i - n$ of weight $W = 0$; if mistake on +ve x, double weights of all $Wt(x_i) = 1$ upto max n; if mistake on -ve x, $\forall x_i = 1$, set $Wt(x_i) = 0$.

Analysis: On +ve x: From f def$n$, $\sum_{x_i=1} Wt(x_i) \leq n$, Max wt added: n, so max mistakes $k \log n$. On -ve x: From f def$n$, $\sum_{x_i=1} Wt(x_i) \geq n$, Min wt removed: n, so mb $= k \log n$.

6.2.2 Decision lists

Length k. 1-Decn lists: Have bag of candidates for 1st position (bag 1); Demote to bag 2 if guess is wrong; etc.. ; $mb = O(nk) = O(n^2)$. [**OP**]:: Get closer to $\Omega(|C|) = \Omega(k \log n)$.

So, t-Decision lists efficiently learnable, by **Feature expansion**; mb $= O(n^t k) = n^{O(t)}$. As $|C| = n^{tk}, \Omega(tk \log n) = O(tn^t \log n)$ needed.

6.2.3 Decision trees

Rank of dec tree is $r \leq \log n$, so dec tree reduced to $\log n + 1$ dec list learnt with mb$= n^{O(r)} = n^{O(\log n)}$. Information theoretically $O(n(\log n))$ enough. [**Find proof**][**OP**]:: Can we get to $O(n^k)$?. Similarly l augmented dec tree of rank r has $mb = n^{O(l+r)}$.

6.2.4 Polynomial size DNF f

Reduce to l-augmented dec tree of rank $\leq \frac{2n}{l} \log s + 1$;
taking $l = \sqrt{n \log s}$, f reduced to $O(\sqrt{n \log s})$ decn list, so mb $= n^{O(\sqrt{n \log s})}$.

$mb = O(n^{O(n^{1/3} \log s)})$: reduce to $O(n^{1/3} \log s)$ degree PTF.

6.2.5 Halfspace

Given sample set $S = \{(x_i, y_i)\}$. The target classifier has the form: $\sum a_i x_i - a_0 \leq 0 \forall i : y_i = -1$. We want to learn some possible $a_i, a_0 \in R$. This can be solved with linear programming; max mistakes: $O(n^c)$.

See statistics survey for the winnow algorithm, the perceptron algorithm, SVM's etc.

6.2.6 Learn parity

Use the GF(2) representation of assignments and parity functions. See mq only algorithm to learn parity.

Could be important in learning Fourier coefficients.

[**OP**]:Parity: Show learnability in IMB model.

6.2.6.1 Halfspaces: Sample net algorithm for Uniform distr

Take S labeled points; given random point p, choose points with positive inner products; return label of the majority.

6.2.7 Halfspaces: Averaging algorithm

6.2.7.1 Problem

With $U(\mu, \sigma)$. Target hyperplane c through origin; unit vector $u \perp c$ defines c. Draw S points uniformly from unit sphere S^{n-1}'s surface.

6.2.7.2 Algorithm

Reflect x_i with $c(x_i) = -1$; get all +ve S. $u_h = avg_{x_i \in_U S} x_i$.

6.2.7.3 Proof of goodness

Angle between u, u_h be θ: $Pr(sgn(\langle u, x\rangle) \neq sgn(\langle u_h, x\rangle)) = \frac{\theta}{\pi}$. Let u'_h be component of $u_h \perp u$. $\theta = tan^{-1} \frac{\|u'_h\|}{\langle u, u_h\rangle}$.

Show $\langle u_h, u\rangle$ large;
so see $\langle u_h, u\rangle = m^{-1} \sum \langle u_h, x_i\rangle$; so $E[\langle u_h, u\rangle] = E[\langle u_h, x_i\rangle]$. But, for u fixed and x uniform from unit sphere,
$Pr(a \leq \langle u, x\rangle \leq b) = \sqrt{n} \int_a^b (\sqrt{1 - z^2}^{n-3} dz)$ [**Find proof**]. So, $E[\langle u_h, x_i\rangle] = 2\sqrt{n} \int_0^1 (1-z^2)^{\frac{n-3}{2}} z dz \geq 2\sqrt{n} \int_0^{\sqrt{\frac{1}{\sqrt{n}}}} (1-z^2)^{\frac{n-3}{2}} z dz \geq c$. Also, can see $E[\langle u_h, u\rangle]$ big whp. [**Find proof**]

Show u'_h small: Wlog, take $u = (1, 0.., 0)$, let $u'_h = v$. $v_1 = 0$; get other v_i by choosing points uniformly at random from S^{n-2}: get each $v_i \sim N(0, n^{-1})$. So, each $v_i = O(\frac{\sqrt{t}}{\sqrt{n}})$ whp. $\|u'_h\| \leq m^{-0.5}$. [**Find proof**]

So, $\theta = \frac{\sqrt{n}}{\sqrt{m}\pi}$. [**Check**]. For $m = \frac{n}{\epsilon^2}$, error $= \frac{\theta}{\pi} \leq \frac{\epsilon}{\pi}$.

[**OP**]:Prove that $\bigcap$ of halfspaces using averaging algorithm works for arbit distr.

6.2.8 PTF of degree d

$mb = O(n^d)$, Time: $O(n^{dc}) = O(n^{O(d)})$: reduce to Halfspace.

6.2.9 Of a panel of k experts

6.2.9.1 The problem

You have a panel of experts (e_i), who on input x return their verdict $(e_i(x))$ on whether $c(x) = 1$ or $c(x) = -1$.

Best expert for input sequence It is possible that no $e_i(x)$ works perfectly $forall x$. So, for every e_i, there is always an input sequence, for which $e_i(x)$ is always wrong. However, for a given input sequence, the 'best expert' can easily discerned from hindsight.

Performance goal If you knew who the least erroneous expert was, you would use the verdict of that expert alone; but you do not know this. You want an algorithm which combines the verdicts of all these experts in such a way that you do not perform much worse than the best expert.

6.2.9.2 Weighted majority

Classifier returns $sgn(\sum_i w_i e_i)$.

Learning algorithm: init: $w_i = 1$; mb(best expert) = m_{min}. When wrong, halve w_i.

Analysis: When the algorithm is wrong, $\geq \frac{1}{2}$ of total weight W is on wrong experts and $\frac{1}{4}W$ is lost due to the corrective update. So, after m mistakes, $W \leq (\frac{3}{4})^m k$. So, considering the weight of the best expert after m mistakes, $(\frac{1}{2})^{m_{min}} \leq W \leq (0.75)^m k$; so $m \leq O(m_{min} + \log k)$.

Matter of focusing wt on the best expert: see also external regret minimization analysis in Game Theory ref.

6.2.9.3 Randomized weighted majority

In this version of the algorithm, the classifier returns +1 with probability $\frac{\sum_{i:e_i(x)=1} w_i}{\sum_i w_i}$. This is the same as returning the vote of the expert e_i with probability w_i.

The expected number of mistakes after a certain number of rounds (rather than a certain number of mistakes) can then be analysed using a similar analysis. It turns out to be better. For intuition as to why this is the case, see note on the adversarial nature of the mistake bounded learning model.

6.2.9.4 Comparison with halfspace algorithms

By vieweing input bits as experts, learning a halfspace $w^T x + w_0$ used to classify x can be cast as one of learning weights to be assigned to a panel of experts.

For comparison with some related half-space learning algorithms, like the winnow and the perceptron learning algorithm, see sections about them in the statistics survey.

6.2.10 Intersection F of k halfspaces

As $NS_a \leq k\sqrt{a}$. [**Find proof**]

6.3 Infinite attribute MB learning model (IMB)

6.3.1 Problem and notation

Infinite literals $\{x_i\}$ out there. A sample point is specified by a list of attributes present in the sample.

r is the number of variables c actually depends on. L: MB model algorithm for learning C. n, when not used in the MB sense: size of the largest example seen. L' : algorithm to learn in infinite attribute model. $p(r, |c|)h(n) =$ mistake bound of L; $p'(r, |c|)h'(n) =$ mistake bound of L'. N_i : set of literals used by L' at step i.

We can assume that L' knows r and p: else, r and p can be doubled or squared till L' stops making more than $p(r, |c|)h(n)$ mistakes.

6.3.1.1 The key problem

Which attributes are relevant for classification?

6.3.1.2 Importance

Eg: A rover in mars trying to understand the properties of life there.

6.3.2 ae learning

Same as in MB model, except use n as size of largest example.

6.3.3 Lower bound

Using MB model lower bounds: $\Omega(VCD(C_{r,n}))$.

6.3.4 Sequential learning algorithm

L' uses L and an attribute set N to label the examples.

6.3.4.1 Algorithm

Init: N empty. When L makes $\geq p(r, |c|)h(|N_i|)$ mistakes, we know that N_i doesn't have all relevant vars; during mistakes, we would have seen ≥ 1 relevant literal not already in N_i. So, we set N_{i+1} to include all variables seen during mistakes, along with N_i.

6.3.4.2 Analysis

After r iterations, L' gets all relevant literals. So, mistake bound is $O(rp(r, |c|)h(|N_r|)) = rp(r, |c|)h(n)p(r, |c|)h(|N_{r-1}|))) \ldots$

6.3.5 Learning by mapping

Works for pec C. Keep mapping/ projecting variables to a list of variables, N: yields better bound, simpler analysis. We use $s = |c|$ below.

Whenever a mistake is made, any new attribute is immediately added to N. Now, if the MB algorithm made at most $p(r, s)h(|N|)$ mistakes, the new algorithm makes at most $p(r, s)h(|N|)$ mistakes, where $|N|$ solves $|N| \geq np(r, s)h(|N|) + n$.

So, if pec class ae learnable in MB model, it is also learnable in the IMB model. Also, if pec class strongly ae learnable in MB model, it is also strongly ae learnable in the IMB model.

[**OP**]:Show that learnability in IMB implies ae learnability in MB model.

6.3.6 Results

Using $O(r \log n)$ winnow algorithm from MB, disjunctions
learnable with mb $O(r^3 \log rn)$ and time per trial $O(rn \log rn)$. So, size s k-CNF and k-DNF learnable using winnow with time per trial $\tilde{O}(n^k)$ and mb $O(s^3 k \log sn)$.

[**OP**]:Learn decision lists in the IMB in time and mistake bound poly(n, r)?

6.3.7 Expensive Sequential halving algorithm

Sequential learning algorithm where L = expensive halving algorithm with $mb = \log |C(N)|$, $C(N) = \{c' | c'(x) = c(x \cap N\}$. $C(r) = \bigcup C(S') : |S'| \in [0, r]$. $C(N) \leq \binom{|N|}{k} C(k)$ as all c based on max r literals. So, at any step, $p(r, |c|)h(n) = r \log(|N||C(r)|)$. So, $p'(r, |c|)h'(n) = r^2 \log(nr|C(r)|)$.

6.4 MBQ and IMBQ models

MB and IMB models with MQ oracles. Trying to learn pec class. An example of Active Learning. Motivating scenarios: password cracking.

If $f(s_1) = 1$ and $f(s_2) = 0$, with $\log n$ membership queries (mq), you can identify the relevant variable in $s_1 \Delta s_2$.

Lower bound: $\Omega(VCD(C_{r,n}))$. [**Find proof**]

Convert an efficient MBQ algorithm (L) with bound $q(\text{n}, |c|)$ into an algorithm L' that strongly ae learns the class in MBQ or the IMBQ model with bound $2(r+1)q(\text{n}, |c|) + r(\log m + 1)$. Pf: N: set of possibly relevant attributes; Init: $N = \phi$; keep growing N to include all relevant attributes. Take arbit partial assignment P for V-N; Run L to learn c_P. Keep projecting examples $(s \to P/s)$ and mq to and fro; Respond to mq in the obvious way. If it makes a mistake on some s, check using mq if $c(s) \neq c(P/s)$; if so, find and add the relevant variable therein. So, bound of the resultant algorithm L' is roughly r $\times$ bound of L: , L' is ae.

6.5 Predictive power of Consistent (maybe small) h

6.5.1 (a,b) Occam algorithm for C using H

Give h consistant with sample S, $size(h) \leq (n.size(c))^a m^b$.

So, the size of the h generated is not too big. This property turns out to be important in bounding the size of H, which in turn helps us prove the goodness of the PAC learning algorithm which we craft using the Occamish algorithm.

This is reminiscent of the predictive power of Occam razor used in elucidating the philosophy of science after the rennaissance: make as few assumptions in your theory as possible.

6.5.2 Occam razor: Goodness from consistency

Any efficient Occam algorithm can be used to construct a good PAC algorithm: simply draw $m = O(\epsilon^{-1}(\log|H_{n,m}| + \log(\delta^{-1})))$ samples from the distribution D and use the Occam algorithm to learn a hypothesis h consistent with this sample set.

Proof. : Using the Chernoff bounds, we can say: The empirical estimate of the goodness of a fixed h on a large sample set S is, with high probability, very close to its actual goodness. But there can be many h which are ϵ close to c, and we want to be able to say that, irrespective of which h the Occam algorithm outputs, it is likely to be good. To do this, we use the union bound from probability theory: $Pr(\exists \text{ bad h}; [c\Delta h] \cap S = \phi) \leq |\Delta_\epsilon(c)|(1-\epsilon)^m \leq |H|(1-\epsilon)^m \leq |H|e^{-m\epsilon} \leq \delta$.

In other words, we found a way of saying that S is likely to be an ϵ net around c.

6.5.2.1 VCD - Occam razor: Extension to unbounded C

Even if $|H| = \infty$, $m = \Omega(\epsilon^{-1}\log(\delta^{-1}) + \frac{d}{\epsilon}\log(\epsilon^{-1}))$ examples likely to form ϵ net: use bound on $\Pi_C(m)$. So, whp:
$\epsilon = O(m^{-1}\log(\Pi_C(2m)) + m^{-1}\log d^{-1})$: ϵ decreases with increase in m, decrease in $\Pi_C(2m)$.

6.5.2.2 Almost consistent h

Suppose that an Occam-like algorithm produces a h with a small, non zero empirical error on S: ϵ_S.

Using the Hoeffding's inequalityuality, for a fixed h, we see that this is likely to be a good estimate of the actual error. $Pr(|\frac{\epsilon}{2} - \epsilon_S| \leq \frac{\epsilon}{2}) \leq 2e^{-\frac{\epsilon^2}{2}m}$; so any h with $\epsilon_S = \frac{\epsilon}{2}$ on $m = \Omega(\epsilon^{-2}\log\frac{1}{\delta})$ good whp.

As before, the union bound using either the VCD d or using $|H|$ can then be applied to get: $\Omega(\epsilon^{-2}(\log|H| + \log\frac{1}{\delta}))$ and $\Omega(\epsilon^{-2}(d + \log\frac{1}{\delta}))$.

6.5.2.3 Occam with Approximate set cover

Take set U of m examples seen so far; $|c|$:= size (num of literals) of smallest c to cover U. Greedily, repeatedly alter c (add literal) to cover most of uncovered part of U at step i (U_i): as c covers U_i, atleast 1 literal in c covers $\frac{U_i}{|c|}$; so $U_{i+1} = |U_i| - \frac{U_i}{|c|} \leq m(1-|c|^{-1})^{-i}$; so, $|h| = O(|c|\log m\log n)$; by Occam razor $m = O(\frac{|c|\log^2 n + \log\frac{1}{d}}{\epsilon})$. Eg: learn disjunctions of size k.

6.5.3 Converse to Occam razor

For any pac learnable C, can use Adaboost with L, do boosting by sampling to find small hypothesis consistent with any sample S.

7 Dealing with noise

7.1 Classification noise

Random noise vs Adversarial noise. Getting (x, l) instead of (x, c(x)).

Malicious noise: both x and c(x) corrupted.

7.1.1 Random Classification noise model

Uniform noise $\eta \leq \eta_0 \in [0, .5)$; L given η_0, also polynomial in $\frac{1}{.5-\eta_0}$.

7.1.2 Statistical Query (SQ) learning model

(Kearns). A **statistical query** (criterion, tolerance) $= (\chi, \tau)$ yields probability $Pr_x(\chi)$ or $E_x(\chi)$ of χ over examples. χ efficiently evaluatable, τ not very tiny. L forms h using only statistical queries.

7.1.3 Show non-constructive SQ learnability

If $d = sqd_D(C)$ and $S_D = \{f_i\}$ the corr shattered set; $\forall f \in$C, $\exists f_i \in S_D$: $|corr_D(f_i, f)| > (d+1)^{-1}$. Let
$g_S = max(|corr_D(f_i, f_j)|)$; $f_i, f_j \in S$; $g^* = \min\{g_S : |S| = d\}$: pick such S; if $g^* \leq (d+1)^{-1}$ $sqd_D \geq d+1$: absurd; so, $g^* > (d+1)^{-1}$; so swapping any $s \in S$ with $s' \in S_D$, get result.

7.1.3.1 Non uniform algorithm

So make SQs: $\forall f_i \in S_D, E[f_i c] =?$ to find $f_j : E[f_j c] > (d+1)^{-1}$; $Pr(f_j \neq c) \geq 2^{-1} + (d+1)^{-1}$. So, $\exists$ weak learning algorithm for C with running time $O(d^t)$, t constant.

Also $\Omega(d^{t'})$: as $sqd = \Omega(vcd)$.

7.1.4 Simulate SQ learning

Noise absent: Take many examples to find P_χ, use Chernoff. **Classification Noise** present: Sample η_i from [0,1/2) at sufficiently small intervals; Express P_χ as combo of probabilities over noisy examples, η; for each η_i get h_i; gauge noisy errors, return h with least noisy error (which grows like actual error).

7.1.5 Efficient PAC learnability with noise

Any C efficiently learnable using SQ is efficiently PAC learnable with classification noise. So, PAC model more powerful than SQ: $\exists$ C learnable in PAC+noise, but not learnable in SQ. (Find proof.) But, Folk theorem: most existing PAC algs extensible to SQ algs.

(Also using statistical queries): Conjunctions. k-CNF: by feature expansion.

Halfspaces ([**Find proof**]: Blum etal): Important subroutine for various learning algs.

7.1.6 Learning Parity

$c = p_S$; classification noise. Best algorithm: $2^{\frac{n}{\log n}}$. [**Find proof**]

Learning k-PARITY with random noise: Measure errors of all $O(n^k)$ h over a random sample set of size $O(\frac{1}{1-2\eta}k \log n)$.

If parity learnable in polytime, DNFs efficiently learnable under U: see reduction from agnostic parity learning.

[**OP**]:: If you can learn parity when there is constant noise, can you learn it when $\eta = 1/n^r$?

7.1.7 Random persistant classification noise

Classification noise model with modification: $\forall x$: once mq(x) is returned, all future queries mq(x) elicit the same label.

Motivation: Random classification noise can be beaten if MQ(c) oracle present: For each example, make $poly(\frac{1}{.5-\eta_0})$ mq and take majority answer.

8 Active learning

8.1 mq only model

Finite attributes around. Exact learning using only membership queries (mq). Scenario: Robot explores labyrinth.

Constrained instance oracle for f: Takes partial assignment P and prediction b, if possible, extends P to a complete assignment A such that f(A) = 1-b.

Lower bound: By information theory, atleast $\log |C|$ mq needed.

8.1.1 ae learning

8.1.1.1 Monotone functions

Learnability in mq only model does not always imply ae learnability for deterministic algorithms.

But, implied for monotone functions. Pf: Let constrained instance oracle be simulated using $t(\mathrm{n}, r, |c|)$ mq. Make mq only algorithm with $q(\mathrm{n}, |c|)$ bound, get mq algorithm with $2(r+1)t(\mathrm{n}, r, |c|)q(\mathrm{n}, |c|) + r(\log n + 1)$ bound. Use

the MBQ algorithm to aeMBQ algorithm conversion, but use this trick to find relevant vars: When mq is made on partial assignment P to N, extend P to A, find c(A), use constrained instance query oracle to find an assignment B with $c(B) \neq c(A)$. t(n,r, —c—) for monotone functions is a constant.

8.1.1.2 Parity fn

To learn any parity fn f, n linearly independent mq are both necessary and sufficient: Consider the GF_2 representation of parity funcitons and assignments in Boolean fn ref.

But, rand algorithm can strongly ae learn: Simulate constrained instance oracle by uniformly sampling assignments to unassigned variables and making mq's with them.

8.2 mq and eq model

8.2.1 Learning DFA with membership queries

Keep / update a binary classification tree: leaves are states in actual DFA denoted by access strings; internal nodes are distinguishing strings which lead to accptence from one bunch of states and rejection from another. Make hypothesis DFA thence: sift access string + alphabet + dist. string; Use equivalence queries to get counterexample; simulate prefixes of counterexample in both hypothesis DFA and using membership queries with distinguishing strings in classification tree to distinguish some new state; repeat.

(**Myhill-Nerode**) So, minimal DFA unique!

8.2.1.1 Learning without reset, using homing sequence h

Only strongly connected component learnt. Run algorithm in parallel with many start states; For every membership query, execute h first to return to some state; awaken the right algorithm; run the reminder of the pending membership query and other processing.

8.2.1.2 Find homing sequence h

Homing sequence in a DFA: A string executed on DFA gives a sequence of outputs (+/-); if homing sequence, output sequence determines destination state irrespective of output.

Start with empty h; run no-reset learning algorithm to get a generalized binary classification tree (root can be one of states corresponding to some output sequence for h); if tree outgrows size(DFA) use randomization to find equivalent states: access string1 + dist str and string1 + dist str yield same result (accept / reject); append their distinguishing string to h; repeat.

Bibliography

[1] Michael J Kearns and Umesh V Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, USA, 1994.