Summary of Computational Learning Theory

Kui Tang

December 2012

Prepared from notes and homeworks from Professor Rocco Servedio's Fall 2012 course in preparation for the final. No guarantees of completeness or correctness.

Contents

1 Online Learning

1.1 Elementary Algorithms

To learn a one-decision list, use extended decision (multiple rules; if multiple antecedents are satisfied, arbitrarily choose one) lists are your hypotheses. Begin with all $4n + 2$ rules. Each time a rule fires and returns a wrong answer, such that that rule is *responsible* for the error, we push the rule down to the next level. A rule is pushed iff a mistake is made (show the converse side), so we have at most $O(n)$ mistakes.

Decision lists can be encoded as LTFs with exponentially decaying weights. Recall that a matching rule short-circuits the rest of the decision list evaluation. In an LTF with exponentially-decaying weights, the first matching rule overwhelm all subsequent coefficients [HW1.1b].

[HW1.2] Any OLMB algorithm can be made conservative: Suppose algorithm A has mistake bound M. Algorithm A' predicts with its current hypothesis. If it turned out to be a mistake, A' gives the current example to A and takes A's output hypothesis.

The two algorithms make the same number of mistakes on the *subsequence* that causes mistakes. This is because A only sees this sequence of mistake examples. Therefore, A' can make no more mistakes than A itself would have.

Randomized halving algorithm: if we remove the oblivious adversary, a more efficient algorithm (in expectation) can be obtained: maintain the set of consistent-so-far hypothesis, and on each mistake, pick h uniformly at random from the new set of consistent hypotheses. Then for any fixed target c and training sequence $\left\{x^i\right\}$ (this is the *oblivious adversary* assumption), we have $\mathbf{E}[M] \leq \ln|\mathcal{C}| + O(1)$. Proof idea: order the concepts in CONSIST by the order in which the fixed training sequence will eliminate them. Define $M_r = \mathbf{E}$ [mistakes when $r$concepts left]. Each of the r remaining examples has *uniform* probability of causing a mistake, giving $M_r \leq \sum_{t=1}^{r-1} \frac{1}{r}(1 + M_{r-t})$. Some worst case analysis gives you a rational sequence, thus summing to $\ln r$.

1.2 Perceptron

Wrong examples x pull the normal vector of the hypothesis hyperplane toward v.

$w \leftarrow 0$
for $(x, y) \in$ examples:
  $\hat{y} \leftarrow w^\top x$
  if $\hat{y} \neq y$: $w \leftarrow w + yx$

Let v be the concept. If $\delta = \inf_x \left|v^\top x\right| > 0$, then Perceptron achieves $O\left(1/\delta^2\right)$ mistake bound. The proof is obtained by simultaneous upper- and lower-bounding of $w^\top v$ and $w^\top w$.

1. *After M mistakes $\delta M \leq w^\top v$.* Proof: initially, $M = 0$ and $w^\top v = 0$ since $w = 0$. Each mistake increases $w^\top v$ by at least δ (write out the update equations for both FP and FN cases)

2. *After M mistakes, $\|w\|^2 \leq M$.* Proof: $(w + x)^\top (w + x) = w^\top w + 2w^\top x + x^\top x \leq w^\top w + 1$ for the FN case, since $w^\top x < 0$. Symmetric for the FN case.

Put them together with Cauchy-Schwarz: $\delta M \leq w^\top v \leq \|w\| \leq \sqrt{M}$.

Perceptron is easy to code and works well in practice, but is not optimal. Linear programming can solve LTFs in poly $(\log 1/\delta)$ time.

Example [HW2.4]: if i is the first index on which $x_i \neq y_i$, then $f(x, y) = \begin{cases} 1 & x_i = 1, y_i = -1 \vee x = y \\ 0 & \text{ow} \end{cases}$ requires exponential time to learn.

Dual perceptron: $w^\top x$ is really $\sum_{i \in MISTAKES} \left(x_i^\top x\right) y_i = \sum_{i \in MISTAKES} y_i K(x_i, x)$, i.e. a kernel or inner product.

1.3 Winnow

Winnow learns k-sparse disjunctions: the target disjunction contains $\leq k$ variables out of $\{x_1, \ldots, x_n\}$. The Winnow algorithm demotes (eliminates) the coefficients for all dimensions that are responsible for false positive while doubling the coefficient for all dimensions that are potentially responsible for false negatives. This eventually converges to weight on those coefficients which are truly responsible for the correct concept. More formally,

$w_i \leftarrow 1, \forall i$
for $(x, y) \in$ examples:
  if $h(x) = 1$ **and** $c(x) = 0$: *# FP*
    for i s.t. $x_i = 1$: $w_i \leftarrow 0$
  if $h(x) = 0$ **and** $c(x) = 1$: *# FN*
    for i s.t. $x_i = 1$: $w_i \leftarrow 2w_i$

We prove the lemmas:

1. No weight is ever negative (duh), each promotion step doubles at least one weight (because it happens on an FP, so the true hypothesis was positive, so at least one dimension must be 1), and $w_i \leq 2n$, for if $w_i \geq n$, then $h(x) = 1$ and we can't ever have an FN).

2. Total number of promotion steps is $p \leq k \lg 2n$. This is because no dimension belonging to the correct c is ever demoted (because it can never cause a false positive), which each promotion doubles at least one weight (from Lemma 1). Each variable can be promoted at most $\lg 2n$ times (Lemma 1) and there are k variables in the target, giving us this.

3. (Elevator Bound) The number of demotions satisfies $d \leq p + 1$.

Naive Winnow (above) does not tolerate noise. What is *no* disjunction is consistent with all examples? Blum modifies Winnow to do a $w_i \leftarrow w_i/2$ update on an FP. If we define m_c to be the mistakes made by the best expert with respect to hypothesis c and A_c to be the number of *attribute errors,* that is, labels that contradict c. More precisely, we count the positive examples that satisfy nothing in c and the negative examples that satisfy k relevant variables of c. Then the Winnow mistake bound is $O(A_c + k \log n)$. The attribute errors satisfy $m_c \leq A_c \leq k m_c$ because each attribute error is a mistake and one can mess up on at most k variables (that's all there is). Then this Winnow achieves $O(A_c + k \log n)$, which means it is $O(r)$-competitive with respect to the *best* k-disjunction. This holds even if no disjunction is consistent with all the examples.

1.4 (Randomized) Weighted Majority

Given N experts and $0 < \beta < 1$, assign each expert a weight $0 \leq w_i \leq 1$. Suppose the best expert makes m mistakes. Then the WM algorithm

$w_i \leftarrow 1, \forall i$
for $(x, y) \in$ examples:
  for $e_i \in$ experts:
    $\hat{y}_i \leftarrow e_i(x)$
    if $\hat{y}_i \neq y$: $w_i \leftarrow w_i \beta$
  predict $\sum_{i=1}^{N} w_i \hat{y}_i$

makes

$$m \leq \frac{\lg N + m \lg \frac{1}{\beta}}{\lg \frac{2}{1+\beta}}$$

mistakes. Quantitatively, a high β decreases the constant factor to its limit 2 which pushing up the variable factor to infinity. For instance, with $\beta = 1/2$, the bound is $2.41\,(m \lg N)$ while the corresponding bound for $\beta = 1 - \epsilon$ is $2m + \frac{2}{\epsilon} \lg N$.

The proof is apparent in the multiplicative weight updates: weights decay exponentially with respect to the number of mistakes made. Define $W = \sum_i w_i$ to be the total weight of the system at some iteration. Initially $W = N$. Now suppose the entire algorithm makes a mitake. Then at least half of experts made a mistake, so $W^* \leq \frac{W}{2} + \beta\frac{W}{2} = W\left(\frac{1+\beta}{2}\right)$. After M mistakes, we have $W \leq N\left(\frac{1+\beta}{2}\right)^N$, while the best expert has weight $w_* \geq \beta^m$ by its mistake bound, and by definition, $w_* \leq W$. Glue the inequalities together and do a little algebra.

Weighted majority ignores confidence and makes the maximally greedy choice: if 49% of the weight is on 0 but only 51% is on 1, WM will return 1 even though it has almost equally strong reason to return 0. We can thus derive a randomized algorithm with better *expected* mistake bound. The ONLY change is that instead of picking the output with maximum weight, we normalize weights (so they become probabilities), and select sample an expert i from the multinomial distribution of their weights, so i's prediction. The randomized algorithm has

$$\mathbf{E}[M] \leq \frac{m \ln 1/\beta + \ln N}{1-\beta}$$

Notice that with $\beta = 1 - \epsilon$, we get $\mathbf{E}[M] \leq m + \frac{\ln N}{\epsilon}$: the multiplicative constant disappeared!

Proof: suppose we run for T epochs, and put F_t as the fraction of weight that make a mistake at time t. Then $\mathbf{P}(\text{Mistake on trial } t) = F_t$. This is a Bernoulli rv, so $\mathbf{E}[M] = \sum_{t=1}^{T} F_t$. Before seeing example t, we have decompose the weight into the share for experts about to be right and about to be wrong: $W = (1 - F_t)\,W + F_t W$. After, $W^* = (1 - F_t)\,W + \beta F_t W = (1 - (1-\beta)\,F_t)\,W$. After T trials, the final total weight of the system is $W_T = N\prod_{t=1}^{T}(1 - (1-\beta)\,F_i)$. On the other hand, $\beta^M \leq W_T$. Take logarithms and use Best Identity Ever.

1.5 General Bounds

VC Dimension Let $\mathcal{C}$ be a concept class over X and $S \subset X$. We say $\mathcal{C}$ *shatters* S if for each $T \subset S$, there exists a $C \in \mathcal{C}$ such that $C \cap S = T$. The VC-dimension of $\mathcal{C}$ is the size of the largest shatterable set.

Intuitively (due to Vapnik), a concept class with infinite (or $2^{|X|}$ if finite) VC-dimension can explain ANY data, and thus are not falsifiable (and unscientific). More weakly, the theory $\mathcal{C}$ can explain ANY observation of fewer than $VC(\mathcal{C})$ points. This interpretation leads to various lower bounds of learning (because nothing can be said for sets that are "too small").

The VC dimension has the following properties:

1. The VC dimension of n-dimensional half-spaces is $n + 1$.
2. On a finite hypothesis class $\mathcal{C}$,

   $$VC(\mathcal{C}) \leq M_{HALVING}(\mathcal{C}) \leq \lceil \lg |\mathcal{C}| \rceil$$

   Equality with powerset hypothesis class and strict with $X = \{1 : N\}$ and $\mathcal{C} = \{[n, n+3] : -2 \leq n \leq N - 3\}$. [HW2.3]
3. If $VC(\mathcal{C}) = d$, then $|\mathcal{C}| \geq 2^d$ (consequence of above).
4. The VC dimension lower-bounds *any* OLMB algorithm with an adversary (the enemy picks points to be explainable by ANY hypothesis and always answers the opposite of what your algorithm answers. This can last up to d points). This has an intuitive interpretation: *until you get enough points, the data is not sufficient to identify the true concept.*
5. Even with an oblivious adversary, $\mathbf{E}[M] \leq \frac{VC(\mathcal{C})}{2}$. Same reason as above, but you'll only make mistakes half instead of all the time instead of all of the time.

2 PAC Learning

If algorithm L is given

- X, an instance space,
- $\mathcal{C}$, a hypothesis class,
- $0 < \epsilon < 1/2$, an accuracy parameter,
- $0 < \delta < 1/2$, a confidence parameter,
- $EX(c, \mathcal{D})$, an oracle returning pairs $(x, c(x))$ with $x \sim \mathcal{D}$ and $c \in \mathcal{C}$ the true hypothesis

then it *is a PAC algorithm for* $\mathcal{C}$ if with probability $\geq 1 - \delta$, L returns a hypothesis h with $error(h) \leq \epsilon$.

In addition, given a concrete representation of $\mathcal{C}$ and a function $size$ measuring elements of $\mathcal{C}$, and L is given $size(c)$, we say it is *efficiently PAC* if it runs in time poly $\left(\frac{1}{\epsilon}, \frac{1}{\delta}, size(c)\right)$.

Example: Conjuctions *Begin the* $h = x_1 \wedge \overline{x}_1 \wedge \ldots \wedge x_n \wedge \overline{x}_n$*, which has no satisfying assignments. Ignore negative examples, and delete any literals that contradict the positive examples.*

Notice that h always contains at least as many literals as c (i.e. h is more specific). So a literal z appearing in $h \backslash c$ causes an error on positive examples in which $z = 0$. Now, each error (false negative), can be "blamed" on at least one literal z (which was zero when it shouldn't have been). This gives $error(h) \leq \sum_{z \in h} p(z) = \sum_{z \in h} \mathbf{P}_{\mathcal{D}}[c(a) = 1 \wedge a_z = 0]$. There are total of $2n$ literals, so as long as each literal z satisfies $p(z) < \epsilon/2n$, we're good. So define z as a *bad* literal if it takes more than its share of error, i.e. if $p(z) \geq \epsilon/2n$.

We find a sample size that bounds the probability that any literal in h is bad. Note that if z is bad, then the probability it survives for m trials is bounded by $(1 - \epsilon/2n)^m \leq e^{-m\epsilon/2n}$. Multiply this bound by $2n$ (union bound) and solve for m in $\ldots \leq \delta$ to get $m \geq \frac{2n}{\epsilon}\left(\ln 2n + \ln \frac{1}{\delta}\right)$, which is efficiently PAC.

2.1 Probability Inequalities

The PAC guarantee bounds the probability of an error (itself a probability, but over $\mathcal{D}$) being too high. One way to prove an algorithm is PAC is to measure the empirical error $\hat{p}$ and compute the sample size necessary so that the probability of the true error p is less than δ.

1. Markov: for nonnegative X with finite mean μ, we have $\mathbf{P}[X \geq k\mu] \leq 1/k$.
2. Hoeffding: for a sequence $\{X_i\}$ of independent Bernoulli rvs with theoretical probability p and observed probability $\hat{p}$, and given $0 \leq \gamma \leq 1$, we have $\mathbf{P}[\hat{p} > p + \gamma] \leq e^{-2m\gamma^2}$ and similarly for $<$.
3. Chernoff: under the same hypotheses,

$$\begin{aligned} \mathbf{P}[\hat{p} > (1+\gamma)p] &\leq e^{-mp\gamma^2/3} \\ \mathbf{P}[\hat{p} < (1-\gamma)p] &\leq e^{-mp\gamma^2/2} \end{aligned}$$

4. Best inequality ever: used in probability problems to simplify binomial products, remember $1 - x \leq e^{-x}$. Draw the picture, and it's obvious from convexity.

2.2 OLMB to PAC

WLOG, assume A is a conservative OLMB algorithm. Then

```
forever:
    draw (x, c(x)) from EX
    give (x, c(x)) to A and update h if needed
    if (1/ε) ln((m+1)/δ) in a row were correct
        return the current hypothesis h
```

is PAC with sample complexity $\geq m + \frac{m+1}{\epsilon} \ln\left(\frac{m+1}{\delta}\right)$. The proof is simple: if $err(h) \geq \epsilon$, i.e. if h is bad, then the probability that h achieves $\frac{1}{\epsilon} \ln\left(\frac{m+1}{\delta}\right)$ correct examples in a row is bounded by by $\frac{\delta}{m+1}$ (Best Identity Ever). Moreover, since A is conservative, it will one hypothesis for each mistake, so at most $m+1$ hypothesis (include the initial hypothesis), so taking the union bound, we get the probability of a bad hypothesis surviving $\leq \delta$.

With a bit more work, we can get an $O\left(\frac{m}{\epsilon} \ln\left(\frac{1}{\delta}\right)\right)$ bound.

Clearly a PAC algorithm's time complexity is bounded below by its sample complexity.

With one-way functions, we can construct concept classes that]are efficiently PAC learnable but not efficiently OLMB learnable. WHY? But if runtime doesn't matter, then any FINITE hypothesis class that can be learned by PAC can also be learned by OLMB (in particular, halving) [HW4.4]. More concretely, let $X := \{0,1\}^n$. Then $|\Pi_{\mathcal{C}}(X)| = |\{c \cap X | c \in \mathcal{C}\}| = |\mathcal{C}|$. By Sauer's lemma, $|\mathcal{C}| = |\Pi_{\mathcal{C}}(X)| \leq \left(\frac{e2^n}{d}\right)^d$. The Halving algorithm will run with mistake bound

$$\begin{aligned} M &\leq \lg|\mathcal{C}| = d \lg\left(\frac{e2^n}{d}\right) \\ &= d\left[\lg e + n - \lg d\right] \end{aligned}$$

This expression is polynomial in n and d (in particular, since $poly(d)$ dominates $\lg d$), so it remains to show $d = poly(n)$. Denote by m the sample complexity for the PAC algorithm. Fix $\epsilon, \delta = 0.01$ arbitrarily. Then $m = poly(n, 100, 100) = poly(n)$. By the lower bound on PAC learning, $\Omega(100d) \leq m$. This means d can increase no faster than m, so $d = poly(n)$.

2.3 Consistent Hypothesis Finder; Occam's Razor

We can view this as a special case of OLMB learning. Suppose we have a consistent hypothesis finder (i.e. Halving algorithm). If we draw $m \geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right)$ examples, give it to the CHF, and output CHF's return hypothesis. The analysis is trivial. First, define the *bad* hypotheses to be the ones for which $err(h) > \epsilon$. Then the probability that any bad hypothesis survives m samples is $(1-\epsilon)^m$. Best identity ever gives the bound $\frac{\delta}{|\mathcal{H}|}$ and the union bound cancels the denominator.

OCCAM'S RAZOR, CARDINALITY VERSION [KV, pp. 35] IS JUST A REFORMULATION OF CHF, with an arbitrary constant b touching m. (This is to account for representational issues)

The subtlety is that $\mathcal{H}$ is an a-priori fixed concept class. So if your algorithm is "build a lookup table," you must define $\mathcal{H}$ to be the class of all lookup tables. For instance, if $\mathcal{C} = 2^X$ and $|X| = 2^n$ and our algorithm is to just build a lookup table and return 0 for unseen examples, then $d = VC\left(2^X\right) = |X| = 2^n$ and we do not have any PAC algorithm (because the PAC lower bound requires $\Omega\left(\frac{d}{\epsilon}\right)$). This is a case of the size of $\mathcal{H}$, 2^{2^n} being too large to be useful [HW3.5].

Interlude: Greedy Set-Cover Heuristic Given $U = \{1, \ldots, m\}$ and $\mathcal{S} = \{S_1, \ldots, S_J\}$, do:

1. Initialize $\mathcal{R} = \emptyset$, the index set of our chosen cover. (Abuse of notation)
2. Pick the biggest S_i. For each S_j with $i \neq j$, update $S_j \leftarrow S_j \backslash S_i$. (Delete from everyone else the stuff belonging to S_i). And, update $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$.
3. Repeat 2 until the sets indexed by $\mathcal{S}$ cover U.

Let $opt(\mathcal{S})$ denote the cardinality of the optimal cover. Then any subset $U^* \subset U$ has a cover of size at most $opt(\mathcal{S})$ because after all, then entire set U can be covered by a collection of that size. In the worst (highest entropy) case, each set in the optimal cover covers a $1/opt(\mathcal{S})$ fraction of the points in U^*. So then there always exists some $S \in \mathcal{S}$ such that $|S \cap U^*| \geq |U^*| / opt(\mathcal{S})$.

Now let $U_i \subset U$ denote the uncovered elements after step i. Since step i covers at least $|U_i| / opt(\mathcal{S})$ many points, we have $|U_{i+1}| \leq |U_i| - \dfrac{|U_i|}{opt(\mathcal{S})} = |U_i| \left(1 - \dfrac{1}{opt(\mathcal{S})}\right)$. By induction, $|U_i| \leq \left(1 - \dfrac{1}{opt(\mathcal{S})}\right)^i m$, and choosing $i > opt(\mathcal{S}) \ln m$ drives this bound under 1, guaranteeing that all elements are covered after using that many sets.

Example: k-sparse Monotone Disjunctions

1. Draw m labelled examples (m calculated at the end). NOTE: There are 2^n monotone disjunctions, so naively applying CHF would require $O\left(\dfrac{n}{\epsilon}\right)$ draws, which is worse than the bound we will prove below.
2. Run halving algorithm (or any algorithm that uses only negative examples), giving a monotone disjunction of length r consistent with all positive examples.
3. Convert the positive examples to a set cover problem in the method of [HW4.1]. Denote $x^{(j)}$ as the jth positive example. Given (at most) m positive examples and r variables, construct an $m \times r$ binary matrix $(M)_{ij} = x_j^{(i)}$. Define $U = \{1 : m\}$ and S_r as the rth column of M.
   (a) Now, if a k-sparse monotone disjunction consistent with the example exists, then $opt(\mathcal{S}) \leq k$.
   (b) Run the greedy heuristic to get $opt(\mathcal{S}) \ln m \leq k \ln m < r$ subset (selection of columns) that also covers all the positive examples (and hence is consistent with the required sample size m).

Thus, the greedy set-cover heuristic bounds the *length* of hypotheses. How to convert the length into size? Count: $|\mathcal{H}_m| \leq \sum_{j=0}^{k \ln m} \binom{n}{j} \leq n^{k \ln m}$. Some algebra gives $m = 2\left(\dfrac{1}{\epsilon} k \ln n \cdot \ln \dfrac{1}{\delta}\right) \ln \left(\dfrac{1}{\epsilon} k \ln n \cdot \ln \dfrac{1}{\delta}\right)$.

2.4 Hardness of 3-Term DNF Learning; Proper and Improper Learning

Proper learning requires the hypothesis to belong to the concept class; improper learning makes no such restriction. To improperly learn a 3-Term DNF, we just convert it to a 3CNF by "multiplying out:"

$$T_1 \vee T_2 \vee T_3 = \bigwedge_{(u,v,w) \in T_1 \times T_2 \times T_3} (u \vee v \vee w)$$

resulting in $8n^3$ clauses. Regard each clause as a single variable in a larger feature space. Then run the conjuction algorithm. This is an $O(n^3)$ slowdown, which is fine.

But proper 3DNF learning is hard. We appeal to graph coloring:

Graph 3-Colorability *Given an undirected graph G, is there an assignment of 3 colors to the vertices of the graph such that for any edge (i, j), the vertices i and j have different colors?* This problem is NP-complete.

Reduction to 3-Term DNF Learning In general, we can reduce to consistent hypothesis finding. Because we have a finite instance space X, we can set the PAC algorithm to $\epsilon < \frac{1}{2|X|}$ so that PAC satisfaction $\Leftrightarrow$ consistency.

Given n vertices in the graph, take $X = \{0,1\}^n$. The idea is to create a clause for the COMPLEMENT of each color. We will create both positive samples S^+ and negative samples S^- that so that a 3-Term DNF consistent with those samples will encode a valid 3-coloring.

For vertex i, put a 1-bitstring x_i except whose ith digit is zero into S^+. For edge (i,j), put a 1-bitstring x_{ij} except whoe i- and j-th bitstrings are zero. We prove G is 3-colorable $\Leftrightarrow \exists$ a 3-term DNF labelling S^+ and S^- correctly.

$\Rightarrow$: Suppose G is 3-colorable. Put the variables that are NOT red into T_R. (Our DNF does not negate any variable.) Now consider a red vertex i. Only its ith digit is zero, so the only clause on which it could possibly yield zero is T_R. But $i \notin T_R$, so in fact x_i satisfies T_R as well. The analysis is similar for the other colors. Further, no two adjacent vertices are both red, so for each negative example x_{ij}, it cannot be that $i, j \in T_R$ (or any color). Thus, x_{ij} satisfies no clause, and is thus labelled negatively.

$\Leftarrow$: Conversely, suppose a hypothesis $h = T_R \vee T_G \vee T_B$ is found consistent with S^+ and S^-. If x_i satisfies T_R, color i red. Do the same for each of the other colors, and break ties arbitrarily. Since every $x_i \in S^+$ satisfies h, each x_i satisfies at least one clause, so each vertex gets at least one color. Now suppose on the contrary i and j are both red. Then x_i and x_j both satisfy T_R, and hence their bitwise AND, that is, x_{ij}, satisfies T_R and hence h. That's a contradiction because $x_{ij} \in S^-$.

2.5 VC Dimension Lower-Bounds PAC Sample Complexity

If $d = VC(\mathcal{C})$, then any PAC algorithm for $\mathcal{C}$ must use at least $\frac{1}{32}\left(\frac{d-1}{\epsilon}\right) = \Omega\left(\frac{d}{\epsilon}\right)$ *for* $\delta \leq \frac{1}{9}$.

Let S be a size-d set shattered by $\mathcal{C}$. The proof technique is to construct a nasty distribution $\mathcal{D}$. But first, we will show a bound of $d/2$ for the uniform distribution over S (and measure zero on $X \backslash S$). Thus, without loss of generality, we assume $X = S$ (because we will never get any draws from $X \backslash S$). Then $\mathcal{C} = 2^S$ so that for each dichotomy of S, there is exactly one concept in $\mathcal{C}$ (indeed, itself) that induces the dichotomy.

Pick a target concept uniformly at random. This is equivalent to flipping a fair coin d times to generate a dichotomy of S. So suppose the learning algorithm drew $d/2$ examples. Then at best it can memorize these $d/2$ first examples. It cannot yet generalize to the remaining $d/2$ examples; indeed, its expected performance is just getting half of the remaining examples right, i.e. $\frac{d/2}{2d} = \frac{1}{4}$. By Markov's inequality, with probability at least $1/2$, the error rate at least $1/8$. So with only $d/2$ samples, we will fail to achieve the PAC guarantee.

To get the dependence on ϵ, we give $1 - 8\epsilon$ weight to $x_1$and $8\epsilon/(d-1)$ weight to the rest of the points in S (and still no weight outside). Applying the above proportions, if we draw $d/2$ distinct points, then with probability at least $1/2$, we get error at least $\frac{1}{8}8\epsilon = \epsilon$. But we need $\Omega(d/\epsilon)$ trials to get even this far (since we need points with probability only $O(\epsilon/d)$ of being drawn; geometric distribution).

3 Bounds for Infinite Hypothesis Classes

3.1 Growth Functions and Examples

The *growth function* Π relaxes the definition of shattering. Rather than require that all subsets of S be realized (that is, $T = C \cap S$) by some concept in a hypothesis class, we simply count how many of them are: $\Pi_{\mathcal{C}}(S) = \{C \cap S : C \in \mathcal{C}\}$. We can define $\Pi_{\mathcal{C}}(m) = \max_{|S|=m} \Pi_{\mathcal{C}}(S)$.

Let's compute some growth functions.

Given $\mathcal{C} = [a,b]$ and $X = \mathbb{R}$, compute $\Pi_{\mathcal{C}}(m)$. So consider some subset S of m points on the line (the max doesn't matter much here; any set is as good as any other). What subsets of S are realized by the intervals? Clearly, the empty set, singletons (bound a point in S by a narrow interval), and discrete intervals with at

least two points ($I \cap S$ for some interval I). The last class is defined by the number of left/right pairs of S, which is given by $\binom{m}{2}$. Then $\Pi_{\mathcal{C}}(m) = 1 + m + \binom{m}{2} = O(m^2)$. Notice that the VC dimension of intervals on a line is also 2. Coincidence? Methinks not.

[HW4.5] The growth function of half-spaces in $\mathbb{R}^n$ is $O(n^2)$. Suppose some points S were separated by the hyperplane w (suppose the plane is homogeneous). Translate w so that it passed through some example x. Now rotate w so that it passes through two examples. Label this line $\left(x_1^+, x_2^+, b\right)$ where x_1 and x_2 are the two points the line crossed, and b describes which side of the line is positive. There are at most $2\binom{m}{2} = O(m^2)$ such descriptions. The cases for two negative and one positive one negative are similar, retaining the overall $O(m^2)$ bound.

3.2 Sauer's Lemma

Main Result $\Pi_{\mathcal{C}}(m) \leq \Phi_d(m) \leq \left(\frac{em}{d}\right)^d$. Moreover, for $m \leq d$, we have $\Pi_{\mathcal{C}}(m) = 2^m$ and for $m > d$, we have $\Pi_{\mathcal{C}}(m) \leq \left(\frac{em}{d}\right)^d$, which is poly$(m)$.

Proof Define

$$\Phi_d(m) = \begin{cases} 1 & d = 0, m = 0 \\ \Phi_d(m-1) + \Phi_{d-1}(m-1) & d, m > 0 \end{cases}$$

We first prove $\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$. The base cases are just the definition of the binomial coefficients; the inductive steps appeal to the binomial identity $\binom{m-1}{i} + \binom{m-1}{i-1} = \binom{m}{i}$. This is just the definition of Pascal's triangle. Also, remember this combinatorial story: Suppose we have m people and want committees of i people each. Designate a dictator. Each committee either has or does not have the dictator. There are $\binom{m-1}{i}$ committees without the dictator, because you can choose i people from a set of $m-1$, and $\binom{m-1}{i-1}$ committees with the dictator: choose $i-1$ among the $m-1$ non-dictators.

Now $\sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{em}{d}\right)^d$ for $d < m$. This is because $\left(\frac{d}{m}\right)^d < 1$, and multiplying gives

$$\begin{aligned} \left(\frac{d}{m}\right)^d \sum_{i=0}^{d} \binom{m}{i} &\leq \sum_{i=0}^{d} \left(\frac{d}{m}\right)^i \binom{m}{i} \\ &\leq \sum_{i=0}^{m} \left(\frac{d}{m}\right)^i \binom{m}{i} \\ &= \left(1 + \frac{d}{m}\right)^m \\ &= \left(1 + \frac{d}{m}\right)^{\frac{m}{d} d} \leq e^d \end{aligned}$$

where we used $\left(1 + \frac{1}{x}\right)^x \leq e$.

Let $d = VC(\mathcal{C})$. Sauer's lemma says $\Pi_{\mathcal{C}}(m) \leq \Phi_d(m)$. Proof: induction (base case easy), contradiction, and some arguments about combinatorially, how realizing dichotomies work.

3.3 Infinite CHF (Method of Double Sampling)

Give a CHF finder at least

$$m \geq \frac{2}{\epsilon}\left(d \ln\left(\frac{2em}{d}\right) + \ln\frac{2}{\delta}\right) \geq \frac{2}{\epsilon}\left(\ln \Pi_{\mathcal{C}}(2m) + \ln\frac{2}{\delta}\right)$$

Explicitly,

$$m \geq K\left(\frac{d}{\epsilon}\ln\frac{1}{\epsilon}+\frac{1}{\epsilon}\ln\frac{1}{\delta}\right)$$

Proof idea: Draw $2m$ points; put the first m into S_1 and the second m into S_2. We will use S_2 to test our hypothesis that a bad hypothesis survived in S_1. Recall that in the finite case, we took a union bound over ALL hypotheses to bound the probability that any one bad hypothesis survived. Since $\mathcal{C}$ is now infinite, we can no longer take the union bound. But we can still use probability to bound the probability that a bad hypothesis got away.

That is, define events

- A: some bad hypothesis ($error(h) > \epsilon$) is consistent with S_1
- B: some bad hypothesis h is consistent with S_1, but h mislabels $\geq \frac{m\epsilon}{2}$ examples in S_2

One can show with multiplicative Chernoff bounds that $\mathbf{P}(B|A) \geq \frac{1}{2}$ for $m \geq \frac{8}{\epsilon}$. Then $\mathbf{P}(B) \geq \mathbf{P}(A \cap B) \geq \mathbf{P}(A)\mathbf{P}(B|A) \geq \frac{1}{2}\mathbf{P}(A)$

Now we show $\mathbf{P}(B) \leq \frac{\delta}{2}$. An equivalent way to think about this event is some hypothesis mislabeled at least $K \geq \frac{m\epsilon}{2}$ points, but when the points where randomly split into S_1 and S_2, none of them happened to fall into S_1. That's simply the number of ways K black balls can be selected from m divided by the number of ways K black balls can be selected from $2m$ balls, i.e.

$$\frac{\binom{m}{K}}{\binom{2m}{K}} \leq \frac{1}{2^K}$$

(write out factorials and use Stirling's approximation). Now the $2m$ balls are not homogeneous, but can instead be labelled $\Pi_{\mathcal{C}}(2m)$ ways. Multiplying,

$$\mathbf{P}(B) \leq \frac{\Pi_{\mathcal{C}}(2m)}{2^K} \leq \frac{\Pi_{\mathcal{C}}(2m)}{2^{\frac{m\epsilon}{2}}} \leq \frac{\delta}{2}$$

and solve algebra gets the above bound.

Application: Efficient Linear Programming There exists polynomial time linear programming algorithms (consistent hypothesis finders). Since the VC dimension of half-spaces is $n+1$, there exists a poly(n, m, ℓ)-time algorithm to PAC-learn an LTF, where each example is ℓ bits long. (Just draw m samples according to the bound.)

Note that perceptron can take up to 2^ℓ runtime (examples in HW2).

4 Boosting

We say L is a *weak PAC learning algorithm for* $\mathcal{C}$ if with probability at least $1/\text{poly}(n, size(c))$, L outputs a hypothesis with error $\leq 1/2 - 1/p(n, size(c))$.

The polynomial bounds are the weakest demands possible to ensure that any learning happens at all, i.e. at least some ability to generalize outside of the training set.

4.1 Three-Stage Boosting

1. Run L on $\mathcal{D}$; obtain h_1.

2. Filter $\mathcal{D}$ to get $\mathcal{D}_2$: flip a fair coin. If heads, draw labelled examples until we get $(x, c(x))$ with $h_1(x) = c(x)$, and output it. If tails, then draw labelled examples until $h_1(x) \neq c(x)$ and output it. In this way, we get $\mathbf{P}_{\mathcal{D}_2}[h_1(x) \neq c(x)] = \frac{1}{2}$. Now run L on $\mathcal{D}_2$; obtain h_2.

3. Filter $\mathcal{D}_2$ to get $\mathcal{D}_3$, this time so that $h_1(x) \neq h_2(x)$ occur with equal probability.

4. Output $h = MAJ(h_1, h_2, h_3)$.

Denote by β_i the error rate of the ith hypothesis. The following technical lemma will useful:

$$\mathbf{P}_{\mathcal{D}}[x \in S] = 2(1-\beta_1)\mathbf{P}_{\mathcal{D}_2}[h_1(x) = c(x) \wedge x \in S] + 2\beta_1 \mathbf{P}_{\mathcal{D}_2}[h_1(x) \neq c(x) \wedge x \in S]$$

The 3-step booster obtains a modest error decrease. Namely, if each $\beta_i \leq \beta$, then $err_{\mathcal{D}}(h) \leq g(\beta) = 3\beta^2 - 2\beta^3$.

We can bound

$$error_{\mathcal{D}}(h) \leq \mathbf{P}_{\mathcal{D}}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] + \beta \mathbf{P}_{\mathcal{D}}[h_1(x) \neq h_2(x)]$$

Moreover, distribution $\mathcal{D}_2$ can be partitioned into

$$\begin{aligned} \gamma_1 &= \mathbf{P}_{\mathcal{D}_2}[h_1(x) = c(x) \wedge h_2(x) \neq c(x)] \\ \gamma_2 &= \mathbf{P}_{\mathcal{D}_2}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] \end{aligned}$$

where $\gamma_1 + \gamma_2 = \beta_2$. (Remainder of derivation omitted.)

4.2 Recursive Boosting

(Omitted.)

4.3 AdaBoost

AdaBoost increases the weight of examples L got wrong to have exactly 1/2 weight via the α_t updates. The new distribution makes the old h_t useless, forcing L to learn something useful. Moreover, since the α_t determine the final voting weights, a hypothesis with empirical error gets high voting weight.

def AdaBoost(L, T, examples):
    $\mathcal{D}_1(i) \leftarrow \frac{1}{m}$
    for t **in** range(T):
        $h_t \leftarrow L(\mathcal{D}_t, \text{examples})$
        $\epsilon_t \leftarrow$ empirical error of h_t
        $\alpha_t \leftarrow \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
        $\mathcal{D}_{t+1}(i) \leftarrow \frac{1}{Z_t}\mathcal{D}_i(t)\exp(-\alpha_t y_i h_t(x_i))$
    return $x \mapsto \text{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right)$

AdaBoost Theorem *After T iterations, each with advantage $\gamma_t = \frac{1}{2} - \epsilon_t$, the final hypothesis makes at most $\exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$ mistakes on the* training *set.*

Corollary: If each $\gamma_t \geq \gamma$, then running AdaBoost for $T = \frac{1}{2}\frac{1}{\gamma^2}\ln\frac{1}{\epsilon}$ suffices for the final error $\leq \epsilon$.

Proof of theorem: glue together these lemmas:

1. *(Exponential loss bounds discrete loss)* $\frac{1}{m}\left|i : H(x_i) \neq y_i\right| \leq \frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i f(x_i)\right)$. Proof: draw a picture.
2. *(Exponential loss bounded by product of normalizers)* $\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i f(x_i)\right) = \prod_{t=1}^{T} Z_t$. Proof: unfold the recursion on $\mathcal{D}$. Gather the powers of sums.
3. *(Square root bound)* $Z_t = \sqrt{1-4\gamma_t^2}$, and therefore $Z_t \leq \exp\left(-2\gamma_t^2\right)$ by the BIE. Proof: by definition,

$$\begin{aligned} Z_t &= \sum_{i=1}^{m}\mathcal{D}_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right) \\ &= \sum_{i:h_t(x_i)=y_i}\mathcal{D}_t(i)\exp\left(-\alpha_t\right) + \sum_{i:h_t(x_i)\neq y_i}\mathcal{D}_t(i)\exp\left(\alpha_t\right) \\ &= 2\sqrt{\epsilon_t\left(1-\epsilon_t\right)} \\ &= \sqrt{1-4\gamma_t^2} \end{aligned}$$

In practice, weak learners do not exist; that is, $\gamma_t \to 0$. Also, AdaBoost self-corrects for $\epsilon_t > 1/2$ due to the symmetry of $\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.

5 Noisy PAC

In the random classification onise model, the oracle $EX_{CN}^{\eta}(c,\mathcal{D})$ draws an example $x \sim \mathcal{D}$. With probability $1-\eta$, it returns $(x, c(x))$, while with probability η, it returns $(x, \neg c(x))$. An algorithm is said to *PAC learn in the presence of noise* if it fulfills all the PAC guarantees with this noisy oracle. In particular, its final error rate is measured with respect to $\mathcal{D}$, not with respect to the noisy distribution.

Under RCN,

$$\mathbf{P}_{EX^{\eta}(c,\mathcal{D})}\left[h(x) \neq c(x)\right] = \eta + \tau\left(1-2\eta\right)$$

If we don't know η but we do know an upper bound η_0, we can estimate the noise rate [KV pp. 115–6]. There exists some step size Δ, and run the SQ simulator for $i\Delta$. One guess will be close to the right value.

5.1 Statistical Queries

A *statistical query* is a computable binary function of a labelled example, i.e. $\chi : X \times \{0,1\} \to \{0,1\}$. A *statistical query oracle* is takes a query χ and a confidence parameter τ and returns some value in $\{p_\chi \pm \tau\}$ where $p_\chi = \mathbf{P}_{\mathcal{D}}\left[\chi\left(x, c(x)\right) = 1\right]$. In other words, a statistical query gives you a population estimate of your predicate χ, guaranteed to lie within τ of the true proportion.

We say a concept class $\mathcal{C}$ is *efficiently SQ-learnable* if there exists an algorithm L running in time $\text{poly}\left(\frac{1}{\epsilon}, n, size(c)\right)$, where each call to $STAT(c,\mathcal{D})$ takes 1 timestep, and each query (χ,τ) satisfies $\tau \geq \frac{1}{\text{poly}\left(n, \frac{1}{\epsilon}, size(c)\right)}$ (that is, we cannot cheat and ask the oracle for unreasonably accuracy), and $\chi(x, c(x))$ can be evaluated in time $\text{poly}\left(n, \frac{1}{\epsilon}, size(c)\right)$.

(Clean) PAC Learnability implies SQ Learnability This is trivial, because if the oracle is not noisy, we can just take samples from the oracle and use Chernoff bounds to determine when we are confident enough to return an answer.

Note that as in the homeworks before, the error tolerances $(1/\delta)$ must decrease geometrically. That's fine, because the log of a geometric sequence is a polynomial sequence, *so long as we can bound the total number of samples,* which is always the case in these problems. WAIT, DISREGARD THIS; THIS SOUNDS LIKE BULLSHIT!

SQ Learnability implies Noisy PAC Learnability. We simulate an SQ oracle using a Noisy-PAC oracle. Therefore, any SQ algorithm also works on a Noisy-PAC oracle.

The math gets messy pretty quickly. But the idea is to partition $X = X_1 \cup X_2$ where $x \in X_1$ if $\chi(x,0) \neq \chi(x,1)$ (the label matters) and $x \in X_2$ if $\chi(x,0) = \chi(x,1)$. Let their measures be p_1 and p_2.

Parity Functions are Noisy PAC-learnable, But Not SQ Learnable. $PAR_S(x) = \sum_{i \in S} x_i \mod 2$ can be learned by solving linear equations (Gaussian elimination) in $\mathbb{Z}_2$, and using CHF. But parity functions are not SQ learnable.

We say concepts c_1 and c_2 are *perfectly uncorrelated* or *orthogonal* if $\mathbf{P}_{\mathcal{D}}[c_1(x) = c_2(x)] = \frac{1}{2}$. If a concept class $\mathcal{C}$ is pairwise perfectly uncorrelated, then any SQ algorithm to learn $\mathcal{C}$ must either make more than $\sqrt{N}$ queries or make some query with $\tau \leq 1/\sqrt{N}$. (The $\sqrt{\cdot}$ is similar to random walk behavior.)

5.2 Example: Monotone Disjunctions Noisy PAC

TO FILL IN FROM BOOK AND NOTES.

5.3 Lower Bounds

5.3.1 Sample Complexity

Any PAC algorithms requires at least $\Omega\left(\frac{1}{1-2\eta}\right)$ examples [HW5.5]. We will instantiate a different, but equivalent noisy oracle:

- With probability η,
- Otherwise, return an uncorrupted sample.

Then with fewer than $\frac{1}{1-2\eta}$ examples, all we have is random noise, so we cannot learn anything.

5.3.2 Noise Rate and Adversarial Noise

With malicious noise with $\tau = \mathbf{P}_{\mathcal{D}}[h(x) \neq c(x)]$, if the noise rate $\eta \geq \frac{\tau}{1-\tau}$, then no PAC algorithm can achieve $\epsilon < \frac{\tau}{2}$ error rate. (Construct an adversarial distribution.)

In general, to deal with adversarial noise [HW5.6]

6 Representation-Independent (Cryptographic) Hardness

We want to show that a concept class $\mathcal{C}$ cannot be efficiently PAC-learned, even though they have finite VC dimension, and thus have no information-theoretic barrier to learning. Previous hardness results for 3DNF and for monotone disjunctions of length at most k only applies to *proper learning;* i.e. if we restrict the hypothesis. These results will show that for ANY representation of hypotheses, some problems remain hard to learn because they solving them is equvialent to solving a hard cryptographic problem.

Moreover, complexity-theoretic hardness bounds the worst case: the worst 3-colorability problem cannot be solved in polynomial time, but many practical problems can be. Cryptography bounds the average case (see the DCR).

6.1 Pseudorandom Functions are Hard to Learn

6.2 Discrete Cube Roots are Hard to Learn

The elementary number-theoretic result: $f_N(x) = x^3 \mod N$ is a permutation of $\mathbb{Z}_N^*$, because $f_N^{-1}(y) = y^d \mod N$ is an inverse map.

Discrete Cube Root Assumption Let $p(n)$ be any polynomial and $N = pq$ and N has n bits. No $p(n)$-time algorithm, when given N and $y = f_N(x)$, finds x with probability $\geq 1/p(n)$.

Learning Discrete Cube Roots Suppose we have some examples $\left(y_1, f_N^{-1}(y_1), \ldots, y_k, f_N^{-1}(y_k)\right)$. Does the problem of computing x become any easier? No, because the examples give us nothing we couldn't compute for ourselves in polynomial times. Since f is easily computable in the forward direction, we need only draw some samples $\{x_i\}$ and compute $\{f(x_i)\}$ to get our synthetic training set.

The remaining problem is that f outputs multiple bits. So just break f_N^{-1} into $f_{N,i}^{-1}$ functions for each bit i. Let $\mathcal{C} = \bigcup_{i=1}^{\lg N} f_{N,i}^{-1}$. Now suppose $\mathcal{C}$ were efficiently PAC-learnable. Then we can solve DCR by generating our training samples, project each boolean dimension at a time, and learn each dimension's binary function separately, and them together to get a DCR-inverter (with high success probability).