1 Last time and today

Previously:

- Administrative basics, introduction and high-level overview
- Concept classes and the relationships among them: DNF formulas, decision trees, decision lists, linear and polynomial threshold functions.
- The Probably Approximately Correct (PAC) learning model.
- PAC learning linear threshold functions in poly$(n, 1/\epsilon, \log 1/\delta)$ time
- PAC learning polynomial threshold functions.

Today:

- PAC algorithm to learn degree-$d$ PTFs in $O(n^d)$ time
- PTFs for decision trees and DTFs

Relevant Readings:

- Avrim Blum, “Rank-$r$ decision lists are a subclass of $r$-decision lists”, Information Processing Letters, 42, pp. 183-185 (1992) (as the title implies, gives the relation between rank-$r$ decision trees and $r$-decision lists)
2 Polynomial Threshold Functions

Let’s begin by recalling the definition of a PTF. Consider some boolean function $f : \{0,1\}^n \rightarrow \{-1,1\}$. If $p(x)$ is a degree $d$ polynomial such that $\text{sign}(p(x)) = f(x) \forall x \in \{0,1\}^n$, then $p$ gives a PTF for $f$. We say the PTF degree of $f$ is the minimum degree of any such polynomial $p$. Recall also that it’s easy to see that any $k$-DNF has a degree-$k$ PTF. For example, $x_1 \land \overline{x_2} \land x_4$ can be represented by $p(x) = x_1(1 - x_2)x_4 - \frac{1}{2}$.

A couple of notes on PTFs:

• We can view $f$’s inputs as in $\{0,1\}^n$ or in $\{-1,1\}^n$ depending on the situation. The choice won’t change the PTF degree, which is a good indicator that PTFs are a good construction to work with, since they are robust to changes in representation form. Why doesn’t the choice between $\{0,1\}$ and $\{-1,1\}$ not matter? If $p$ is a degree-$d$ PTF for $f$ over $\{0,1\}^n$, then let $q(x_1, \ldots, x_n) = p((x_1 + 1)/2, \ldots, (x_n + 1)/2)$,

where $q$ takes $\{-1,1\}^n$ inputs and clearly still has degree $d$. It is possible to go from $\{-1,1\}$ to $\{0,1\}$ as well, since the transformation applied to each variable is a bijection.

• It is only necessary to consider multilinear polynomials. Why? If we work over $\{0,1\}^n$, then inputs are either 0 or 1 so $x_i^2 = x_i$ for all $i$. Therefore, since the maximum number of possible monomials in a degree-$d$ PTF is

$$\sum_{i=0}^{d} \binom{n}{i} \leq (en/d)^d \leq n^d,$$

a degree-$d$ PTF is a $O(n^d)$ meta-variable LTF over meta-variables that are monomials of degree $\leq d$ over $x_1, \ldots, x_n$.

The latter note leads to the following
Theorem 2.1. The LP-based algorithm can PAC learn \( C = \{ \text{all degree-}d \text{ PTFs over} \{0, 1\}^n \} \) in time \( \text{poly}(n^d, 1/\epsilon, \log 1/\delta) \).

Proof. The proof is essentially the same as the one outlined in the proof of Theorem 3 in lecture 1, except now there are \( O(n^d) \) meta-variables instead of \( n \) variables as in the LTF case. Hence the running time is \( \text{poly}(n^d) \). \( \square \)

This theorem leads to the very useful

Corollary 2.2. Let \( C \) be a concept class over \( \{0, 1\}^n \) such that every concept \( c \in C \) has PTF degree at most \( d \). Then we can PAC learn \( C \) in time \( \text{poly}(n^d, 1/\epsilon, \log 1/\delta) \).

Proof. The validity of the corollary is obvious, since if we can learn \( C_d = \text{all degree-}d \text{ PTFs over} \{0, 1\}^n \) in \( \text{poly}(n^d) \) time, we can certainly learn \( C \subset \bigcup_{i<d} C_i \) in \( \text{poly}(n^d) \) as well. \( \square \)

Remark 2.3. As in the preceding proof, from now on we will omit the implicit (and standard) dependences on \( \epsilon \) and \( \delta \).

Our goal for the next few lectures will be to see how to apply this corollary by showing interesting concept classes satisfy the corollary’s hypotheses. As a warmup (but useful) example, let’s consider how we can PAC learn \( C = \{ k-\text{DLs} \} \) in \( \text{poly}(n^k) \) time. Consider some \( k\)-DL \( D = (((c_i)_{1 \leq i \leq m}, (b_i)_{1 \leq i \leq m}, b)) \), where \( c_i \) is the \( i \)-th conjunction with at most \( k \) literals, \( b_i \) is its corresponding output bit, and \( b \) is the final output bit. Then \( D \) has a corresponding degree-\( k \) PTF with polynomial function

\[
p(x) = \sum_{i=1}^{m} 2^{m-i+1} \cdot Q_i b_i + b,
\]

where \( Q_i \) is an arithmetization of \( c_i \) constructed in the manner described at the beginning of this section. For example, if \( c_i = x_1 \land \bar{x}_2 \land x_4 \land \bar{x}_9 \), then

\[
Q_i = x_1(1 - x_2)x_4(1 - x_9).
\]

So we conclude

Theorem 2.4. The concept class \( C = \{ m-\text{DLs with} \ m \leq k \} \) can be PAC learned in \( \text{poly}(n^k) \) time.
3 Decision Trees

For the first significant, non-trivial example of PAC learning using PTFs, we will consider decision trees. Recall that the size of a DT is equal to the number of leaves. Our first learning result for DTs is the corollary to the following

**Theorem 3.1** (Blum). Any size-$s$ DT has a $(\log s)$-DL of length at most $s$.

**Corollary 3.2.** There is a $\text{poly}(n \log s)$-time PAC learning for size-$s$ DTs.

**Proof.** This follows from the previous theorem and Theorem 2.4.

**Remark 3.3.** This is the fastest known algorithm for a learning size-$s$ DT. Even a small improvement to the runtime in a very special case would earn you $1000 from A. Blum.

**Proof of theorem.** Let $T$ be a size-$s$ DT. $T$ must have at least one leaf of depth at most $\log s$, since otherwise $T$ would have at least $2s$ leaves. Call this leaf $\ell$ with bit $b_\ell = \pm 1$. Let $S$ be the sibling of $\ell$ and let $C$ be the conjunction that corresponds to the path from the root to $\ell$. Now we modify $T$ to obtain a new tree $T'$ by replacing $\ell$'s parent with $S$. So $\ell$ has been removed, as has $\ell$'s parent. Thus, $T'$ no longer computes the correct input if $C$ is true. But this is the only case in which $T'$ disagrees with $T$. Now create a “DL” where if $C$ is true, output $b_\ell$, and otherwise compute the output using $T'$. Based on the previous observation, it is clear that this construction agrees with $T$ for all inputs. $T'$ has size $s - 1$, since $\ell$ has been removed. Now we can repeat the procedure inductively (on $T'$ instead of on $T$), each time guaranteed that the conjunction has size at most $\log \text{size}(T') \leq \log s$ (since the path to some leaf gives the length of the conjunction). Each time the size of the tree decreases by 1 and the DL’s length increases by 1. So after $s$ steps, the tree at the end of the “DL” will consist of a bit, making the scare quotes unnecessary. Thus we have a constructed a length-$s$ DL, proving the theorem.

**Open question:** Is there an $n^{o(\log s)}$-time algorithm to PAC learn size-$s$ DTs?

**Non-open question:** What is the smallest possible PTF degree upper bounds for size-$s$ DTs? Answer: there are size-$s$ DTs for which any PTF must have degree at least $\log s$. (Proving this is an official HW question.)

A small (but not insignificant) improvement over the previous result is possible by using a different measure of a DT’s size.
Definition 3.4. The rank of a rooted binary tree $T$ with left subtree $T_1$ and right subtree $T_2$ is specified as follows

- if $T$ is a single node, then $\text{rank}(T) = 0$
- if $T$ has at least one child, then
  - if $\text{rank}(T_1) \neq \text{rank}(T_2)$, $\text{rank}(T) = \max\{\text{rank}(T_1), \text{rank}(T_2)\}$
  - if $\text{rank}(T_1) = \text{rank}(T_2)$, $\text{rank}(T) = \text{rank}(T_1) + 1 = \text{rank}(T_2) + 1$

We can now prove a better version of Theorem 3.1 using the following

Lemma 3.5. Any rank-$r$ DT has at least one leaf with depth at most $r$ from the root.

Proof. By definition, if $T$ has rank $r$, then either both $T_1$ and $T_2$ have rank $r-1$ or one has rank $r$ and the other of rank $< r$. Move to a subtree with rank $< r$, and proceed to move down the tree inductively. With each step the rank of the tree decreases by at least 1, so after at most $r$ steps you will arrive at a subtree with rank 0, i.e. a leaf.

Theorem 3.6. Any rank-$r$ DT of size at most $s$ has an $r$-DL of length at most $s$

Proof. The proof is identical to that for Theorem 3.1 except we choose which leaf to remove by invoking the previous lemma (so instead of finding one with depth at most $\log s$ we find one with depth at most $r$).

This theorem combined with Theorem 2.4 gives

Corollary 3.7. The concept class $C = \{ \text{rank-$r'$ DTs with $r' \leq r$} \}$ can be PAC learned in $\text{poly}(n^{r'})$ time.

The following proposition explains why it is better to consider the rank of DT a instead of its size.

Proposition 3.8. Any rank-$r$ DT has at least $2^r$ leaves.

Thus, the concept class of rank-$r$ DTs is a strict superset of size-$s$ DTs. So Corollary 3.7 is stronger than Corollary 3.2. As a concrete example, consider a DL of length $r$. It is a DT of rank 1 but size $r$, so Corollary 3.7 tells us how to learn DLs much more quickly.
Proof of proposition. Let $\psi(r)$ be the minimum size of a rank-$r$ DT. In the base case

$$\psi(0) = 1$$

since a tree of rank 0 has a single leaf at its root. If the tree $T$ has rank $r$, either both $T_1$ and $T_2$ have rank $r - 1$ or one has rank $r$ and the other of rank less than $r$ (WLOG, say $T_1$ has rank $r$). This gives two possible relations. First

$$\text{size}(T) = \text{size}(T_1) + \text{size}(T_2) \geq 2\psi(r - 1)$$

so

$$\psi(r) = 2\psi(r - 1). \quad (\ast)$$

And second

$$\text{size}(T) \geq \text{size}(T_1)$$

so trivially we get

$$\psi(r) = \psi(r).$$

Thus, we only need to consider $(\ast)$, which is a simple recurrence relation that, when solved, gives $\psi(r) = \psi(0)2^r = 2^r$, proving the result. \qed

4 DNFs

Our next goal (which won’t be completed in this lecture) is to show that every $s$-term DNF (over $n$ vars) has a PTF with degree $O(n^{1/3} \log s)$. There is an old result [MP ’68] that there is an $n^{1/3}$-term DNF $f$ such that any PTF for $f$ must have degree $\Omega(n^{1/3})$. This indicates that heuristically, $n^{1/3}$ is somehow the right term to have.

The high-level plan for proving this results is as follows:

- Show (roughly) that there are two different $\sqrt{n \log s}$-degree PTF constructions for an $s$-term DNF.
- Combine these two constructions to get the desired PTF

Today, we give the first construction.

**Theorem 4.1** (Bshouty, ’96). *Any n-DL (and hence any s-term DNF, by having all but the last output bit be 1) of length $s$ over $\{0, 1\}^n$ can be expressed as an $O(\sqrt{n \log s \log n})$-DL (and hence has an $O(\sqrt{n \log s \log n})$-degree PTF).*
To prove this result we will need two lemmas.

**Lemma 4.2.** Let \( h = O(\sqrt{n \log s \log n}) \). Any \( n \)-DL of length \( s \) is equivalent to a DT where (i) the internal nodes are single variables \( x_1, \ldots, x_n \); (ii) the tree has \( \leq 2^h \) leaves; and (iii) each leaf is an \( h \)-DL.

**Proof.** Let \( C_1, \ldots, C_s \) be the conjunctions at the nodes of the \( n \)-DL \( f = ((C_i)_{1 \leq i \leq s}, (b_i)_{1 \leq i \leq s}, b) \). The conjunction \( C_i \) is said to be **long** if it has more than \( h \) literals.

The idea of the proof is to build a DT by picking variables in the DL that occur the most often in long terms and then using them to create a DT.

Let \( t \) equal the number of long terms in \( C_1, \ldots, C_s \). Each long term has length greater than \( h \), so there more than \( th \) occurrences of literals in all the long terms. So some literal \( x_i \) or \( \overline{x_i} \) occurs in at least \( th/(2n) \) long terms. Call such a literal **popular**.

Put \( x_i \) at the root of the DT. Let the left child be \( f|_{x_i=0} \) and the right child be \( f|_{x_i=1} \). Next, make the obvious simplifications to the two children: when \( x_i = 0 \), remove any terms with \( x_i \) and when \( x_i = 1 \) remove \( x_i \) from any term in which it is present. Now the sub-functions \( f|_{x_i=0} \) and \( f|_{x_i=1} \) have \( n-1 \) literals. One of the sub-functions has at most \( t - th/(2n) = t(1 - h/2n) \) long terms, since \( th/(2n) \) terms are removed from the DL. Recurse on each sub-function (each of which is a DL) unless that sub-function has no long terms. Conditions (i) and (iii) follow directly from the construction.

To show condition (ii) holds, define \( \phi(n, t) \) to be the maximum possible number of leaves in any DT that can result from doing the process just outlined starting on an \( n \)-DL with \( \leq t \) long terms. Then we have the recurrence relation

\[
\phi(n, t) \leq \phi(n - 1, t) + \phi(n - 1, t(1 - h/(2n)))
\]

since each child has one less literal and one child as at most \( t(1 - h/2n) \) long terms. Applying this relation repeatedly to its own first term we get

\[
\phi(n, t) \leq \phi(n - 2, t) + \phi(n - 2, t(1 - h/(2(n - 1)))) + \phi(n - 1, t(1 - h/(2n)))
\]

\[
\vdots
\]

\[
\phi(n, t) \leq \sum_{i=0}^{n-1} \phi(i, t(1 - h/(2(n - i)))).
\]

Next, approximate the terms in the sum by the largest term

\[
\phi(n, t) \leq n\phi(n, t(1 - h/(2n)))
\]
and then apply this relation repeatedly

\[ \phi(n, t) \leq n^2 \phi(n, t(1 - h/(2n))^2) \]

\[ \ldots \]

\[ \phi(n, t) \leq n^k \phi(n, t(1 - h/(2n))^k). \]

In the base cases

\[ \phi(1, \cdot) \leq 2 \quad \text{and} \quad \phi(\cdot, 0) = 1. \]

Thus, we want to choose \( k \) such that \( t(1 - h/(2n))^k < 1 \) so that the base case holds (giving \( \phi(n, t(1 - h/(2n))^k) = 1 \)). This condition is satisfied when \( k = 2n/h \cdot \ln t \), giving

\[ \phi(n, t) \leq n^k \cdot 1. \]

For a length \( s \) DL, \( t \leq s \), so our tree has at most \( n^k = n^{2n/h \cdot \log s} \) leaves. Thus, the tree's size is at most

\[ n^{2n/h \cdot \log s} = n^{2n \cdot \log s/\sqrt{\log n \log s}} = 2^{O(h)}. \]

So condition (ii) is in fact satisfied.

Lemma 4.3. Any decision tree as described in the previous lemma is equivalent to a \((2h)\)-DL.

Proof. The proof is very similar to the one for Theorem 3.1. Since the tree has at most \( 2^h \) nodes, there exists a leaf with depth at most \( h \). Let \( C \) be the conjunction reaching that leaf and let \( L = (C_i, b_i, b) \) be the \( h \)-DL at that leaf. Create a new DL \(( C \land C_i, b_i, T') \), where \( T' \) is modified in the same way as the previous result (replace \( L \)'s parent by \( L \)'s sibling). Each \( C \land C_i \) has at most \( 2h \) variables per node. Recursively apply the procedure to \( T' \) until the remaining tree is empty.

Proof of theorem. Follows directly from apply the second lemma to the decision tree guaranteed by the first lemma.