CIS 625 PROBLEM SET 1

Professor Kearns
Due Wednesday September 29, 2021 on Gradescope (code ERP5VV)

Problem 1.

Algorithm. Define a learning algorithm L such that L takes in inputs of the form $\langle (x_1, x_2, \ldots, x_n), c(x_1, x_2, \ldots, x_n)\rangle$, which we denote as $\langle x, c(x)\rangle$ and parameters ϵ, δ. We wish to output the tightest fit rectangle of n dimensions.
To do this, we examine all m inputs and select the smallest and largest value for each of n dimensions that is positively labeled. We return the rectangle defined by the $2n$ values that determine the min and max value (edges) of the n dimensions of the rectangle.
Runtime. We see m samples. For each sample, we iterate over each dimension to calculate the tightest fit on that dimension, giving us a $O(mn)$ runtime.
Error Analysis. The error analysis proceeds in a similar manner as the error analysis of PAC learning axis aligned rectangles of 2 dimensions. Consider if the points are selected to some distribution $\mathcal{D}$.
First observe that the hypothesis we return is indeed an axis aligned rectangle and that the rectangle of tightest fit is always contained within the target rectangle R. We define our empirical rectangle as R' and the error between the two rectangles as $R\Delta R' = R - R'$. Notice that we can split the error region $R\Delta R'$ into $2n$ strips. In the 2 dimensional case, it is the top, left, right, and bottom strips. We do this for the rectangle of n dimension. We want to guarantee that the weight of all strips under $\mathcal{D}$ is at most ϵ. Consider the strip with weight at most $\frac{\epsilon}{2n}$. If each strip has weight greater than $\frac{\epsilon}{2n}$, then our total error will exceed ϵ. But a strip will only have weight greater than $\frac{\epsilon}{2n}$ if there is no point in that region.
The probability that there is no such point landing in the region of a strip with weight $\frac{\epsilon}{2n}$ is $\left(1 - \frac{\epsilon}{2n}\right)$. Since every point is selected independently via m draws, the probability that a single strip does not have any point landing in that region is $\left(1 - \frac{\epsilon}{2n}\right)^m$. Using union bound, we can upper bound the probability that no point lands in *any* of the strips as:

$$\sum_{i=1}^{2n} \left(1 - \frac{\epsilon}{2n}\right)^m = 2n\left(1 - \frac{\epsilon}{2n}\right)^m$$

So the probability that our error is greater than epsilon is at least $2n\left(1 - \frac{\epsilon}{2n}\right)^m$. We want this to be less than δ. Now we solve for m:

$$\begin{aligned} 2n\left(1 - \frac{\epsilon}{2n}\right)^m &\leq \delta \\ -\frac{\epsilon m}{2n} &\leq \ln\left(\frac{\delta}{2n}\right) \\ m &\geq \frac{2n}{\epsilon}\ln\left(\frac{2n}{\delta}\right) \end{aligned}$$

Since $\mathcal{D}$ was arbitrary, this analysis must work for any distribution.

Problem 2.

Hypothesis/learning algorithm:

Our algorithm is simply to fit the smallest interval around each contiguous set of positive points. Let h be the hypothesis we output at the end of our algorithm. We construct h in $O(m \log m)$ time as follows:

(1) Sort the points given by the sample in increasing order. Since we are given a finite number of points, there must be a smallest and largest point value.
(2) Iterating from smallest point to right, keep track of the first positive example and then the last positive example that occurs before a negative example.
(3) Repeat on the next positive example.
(4) Add this interval to h so that at the end, $h = \bigcup_{i=1}^{n^*} [a_i, b_i]$

Analysis:

There are two places where our hypothesis can make mistakes: It can mislabel the ends of intervals as negative, and it can also mislabel some negative regions as positive; namely the negative intervals (between positive intervals) from which we don't see any examples. Note that there are $3d-1$ such bad regions ($2d$ ends of positive intervals, $d-1$ sandwiched negative intervals).

We can perform a similar geometric analysis as in the case of rectangles. Consider regions of weight $\frac{\epsilon}{3d}$ from the left and right ends of each of the positive intervals; there can be (at most) $2d$ of these. Also, consider any sandwiched negative intervals with weight $\geq \frac{\epsilon}{3d}$; there can be at most $d-1$ of these. As long as our sample hits each of these $3d-1$ target intervals at least once, each of our error regions will have probability mass $< \frac{\epsilon}{3d}$, so the total error region will have probability mass $< \frac{(3d-1)\epsilon}{3d} < \epsilon$. Our hypothesis is good as long as we hit each of these target intervals at least once.

Define the following events:

$$\begin{aligned} B_i (i \in [3d-1]) &:= \text{the sample misses the } i\text{th target interval} \\ B_i^j (j \in [m]) &:= \text{the } j\text{th example misses the } i\text{th target interval} \\ B &:= \text{the sample misses at least one of the target intervals} \\ G &:= \text{the good event, where we hit all target intervals} \end{aligned}$$

We can then compose these events as follows:

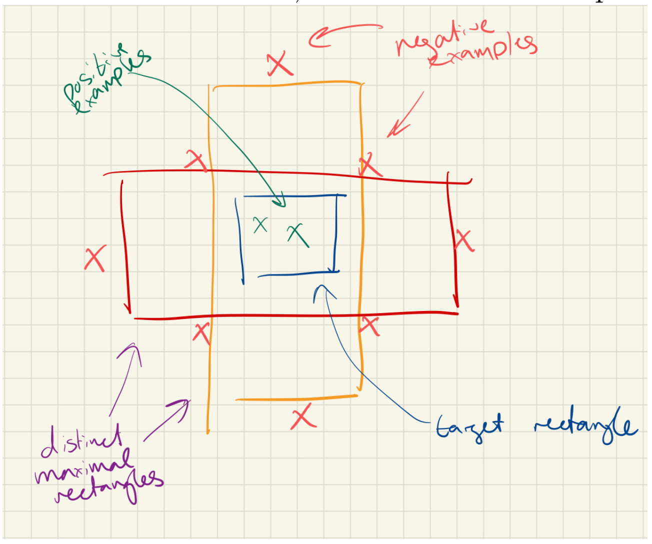
$$\begin{aligned} \forall i, j : B_i^j &\leq 1 - \frac{\epsilon}{3d} \\ \forall i : B_i &= \prod_{j \in [m]} B_i^j \leq \left(1 - \frac{\epsilon}{3d}\right)^m \\ \implies B &\leq \sum_{i \in [3d-1]} B_i \leq 3d \left(1 - \frac{\epsilon}{3d}\right)^m \\ \implies G &= 1 - B \geq 1 - 3d\left(1 - \frac{\epsilon}{3d}\right)^m \end{aligned}$$

Finally, we set $1 - \delta > G$ and find that it suffices for

$$m > \frac{3d}{\epsilon} \log \frac{3d}{\delta}.$$

Problem 3.

There are many answers, but here is one such example. Notice that the rectangles are distinct and maximal, so there is not a unique target rectangle that we can analyze.



Problem 4.

a) We wish to show that if monotone m-term DNF is PAC learnable by some learning algorithm that runs polynomial in $\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$, then general m-term DNF is PAC learnable. We proceed with a reduction from general m-term DNF to monotone m-term DNF.

Suppose that monotone m-term DNF is PAC learnable with some learning algorithm L_m. Define the following mapping

$$f : \{0,1\}^n \to \{0,1\}^{2n}$$

Such that f maps input $(x_1, x_2, \ldots, x_n) \to (x_1, x_2 \ldots, x_n, x_{n+1}, \ldots, x_{2n})$ where the x_{n+i} term represents the negated literal of the x_i term $(\neg x_i)$ in the input.

For example, if $n = 2$, then $f(1,0) = (1,0,0,1)$.

Now we define a mapping g from the set of m-term DNFs over n variables to the set of monotone m-term DNFs over $2n$ variables, which replaces any $\neg x_i$ with x_{n+i}. For example, if $n = 3$, $g(x_1 \neg x_2 \vee \neg x_3 \vee x_1 x_2) = x_1 x_5 \vee x_6 \vee x_1 x_2$. Clearly g is a bijection as for any monotone DNF, we can map it back to a general DNF via g^{-1} that sends any instance of a literal to itself or the negated version (i.e it replaces x_j with $\neg x_{j-n}$ for $j > n$ given the size of n). Notice that these mappings are linear in $O(mn)$.

Let L be a learning algorithm for m-term DNF, which consists of running an instance of L_m, and for each sample point, $\{x, c(x)\}$, L supplies L_m with $\{f(x), c(x)\}$. After processing the entire sample, L outputs $g^{-1}(h)$, where h is the hypothesis outputted by L_m and g^{-1} is the inverse of g as defined above. We claim that L is a PAC learning algorithm for m-term DNF, by m-term DNF.

To see why, suppose that we are given some target m-term DNF $c : \{0,1\}^n \to \{0,1\}$, some sampling distribution D over $\{0,1\}^n$, and some δ, ϵ. We claim that for any $x \in \{0,1\}^n$ and m-term DNF, h, $h(x) = (g(h))\,(f(x))$. To see why, suppose

$x = (z_1, z_2, ..., z_n)$ and let $f(x) = (y_1, y_2, ..., y_n, ...y_{2n})$.

$$\begin{aligned}
h &= x_{a_1}...x_{a_k}\neg x_{a_{k+1}}...\neg x_{a_{k+j}} \vee x_{b_1}...x_{b_l}\neg x_{b_{l+1}}...\neg x_{b_{l+m}} \vee x_{c_1}...x_{c_p}\neg x_{c_{p+1}}...\neg x_{c_{p+q}} \\
h(x) = 1 &\iff h(z_1, ..., z_n) = 1 \\
&\iff z_{a_1}...z_{a_k}\neg z_{a_{k+1}}...\neg z_{a_{k+j}} \vee z_{b_1}...z_{b_l}\neg z_{b_{l+1}}...\neg z_{b_{l+m}} \vee z_{c_1}...z_{c_p}\neg z_{c_{p+1}}...\neg z_{c_{p+q}} = 1 \\
&\iff y_{a_1}...y_{a_k}y_{a_{k+1}+n}...y_{a_{k+j}+n} \vee y_{b_1}...y_{b_l}y_{b_{l+1}+n}...y_{b_{l+m}+n} \vee y_{c_1}...y_{c_p}y_{c_{p+1}+n}...y_{c_{p+q}+n} = 1 \\
&\iff (g(h))(y_1, ..., y_{2n}) = 1 \\
&\iff (g(h))(f(x)) = 1
\end{aligned}$$

From the observation that $h(x) = (g(h))(f(x))$, we observe that running L with target c and distribution D is equivalent to running L_m with target $g(c)$ and some distribution D' over $\{0,1\}^{2n}$ (which is simply the D under f). Therefore by the PAC property of L_m, we may choose sample size m which polynomialy varies with $\frac{1}{\delta}, \frac{1}{\epsilon}$, and see that with probability at least $1 - \delta$,

$$\begin{aligned}
\varepsilon_D(g^{-1}(h)) &= \Pr_{x\sim D}[c(x) \neq g^{-1}(h)(x)] \\
&= \Pr_{x\sim D}[(g(c))(f(x)] \neq (h)(f(x))) \\
&= \Pr_{x\sim D'}[(g(c))(x) \neq (h)(x)] < \epsilon
\end{aligned}$$

We now may observe that this above argument generalizes to other classes of boolean formulae and their monotone counterpart. In particular, we see that if $\mathcal{C}$ is some boolean class of (finite) functions, then the g described above, is a map from $\mathcal{C}$ to its monotone-counterpart. By replicating the above analysis, we may conclude that PAC-learnability of the monotone counterpart by itself, implies PAC-learnablility of $\mathcal{C}$ by itself.

b) We wish to show that if read-once m-term DNF is PAC learnable by itself, then m-term DNF is PAC learnable by itself. Suppose that read-once m-term DNF is PAC learnable by itself with some learning algorithm L_r. Define a mapping

$$f : \{0,1\}^n \to \{0,1\}^{mn}$$

By sending $(x_1, ..., x_n) \to (x_1, ..., x_n, x_1, ..., x_n, x_1, ..., x_n)$. For example, if $n = 3$, then $f(1,1,0) = (1,1,0,1,1,0,1,1,0)$. Now define a mapping g from the set of m-term DNF's over to n variables to the set of read-once m-term DNF's over mn variables, which replaces any x_j in the second conjunction with x_{j+n} and any x_k in the third conjunction with x_{k+2n}. For example, if $n = 3$, $g(x_1\neg x_2 \vee \neg x_3 \vee x_1x_2) = x_1\neg x_2 \vee \neg x_6 \vee x_7x_8$. Notice that these reductions are linear in $O(mn)$.

Finally, let L be a learning algorithm for m-term DNF, which consists of running an instance of L_r, and for each sample point,$\{x, c(x)\}$, L supplies L_r with $\{f(x), c(x)\}$. After processing the entire sample, L outputs $g^{-1}(h)$, where h is the output model of L_r and g^{-1} is the inverse of g (i.e it replaces each x_j with $x_{j \bmod n}$). We claim that L is a PAC learning algorithm for m-term DNF, by m-term DNF.

To see why, suppose that we are given some target m-term DNF $c : \{0,1\}^n \to \{0,1\}$, some sampling distribution D over $\{0,1\}^n$, and δ, ϵ. We claim that for any $x \in \{0,1\}^n$ and m-term DNF,h, $h(x) = (g \circ h)(f(x))$. To see why, suppose $x = (z_1, z_2, ..., z_n)$ and $f(x) = (y_1, y_2, ..., y_{mn})$.

$$
\begin{aligned}
h &= x_{a_1}...x_{a_k}\neg x_{a_{k+1}}...\neg x_{a_{k+j}} \vee x_{b_1}...x_{b_l}\neg x_{b_{l+1}}...\neg x_{b_{l+m}} \vee x_{c_1}...x_{c_p}\neg x_{c_{p+1}}...\neg x_{c_{p+q}} \\
h(x) = 1 &\iff h(z_1,...,z_n) = 1 \\
&\iff z_{a_1}...z_{a_k}\neg z_{a_{k+1}}...\neg z_{a_{k+j}} \vee z_{b_1}...z_{b_l}\neg z_{b_{l+1}}...\neg z_{b_{l+m}} \vee z_{c_1}...z_{c_p}\neg z_{c_{p+1}}...\neg z_{c_{p+q}} = 1 \\
&\iff y_{a_1}...y_{a_k}\neg y_{a_{k+1}}...\neg y_{a_{k+j}} \vee \cdots \vee y_{c_1+2n}...y_{c_p+2n}\neg y_{c_{p+1}+2n}...\neg y_{c_{p+q}+2n} = 1 \\
&\iff (g(h))(y_1,...,y_{mn}) = 1 \\
&\iff (g(h))(f(x)) = 1
\end{aligned}
$$

From the observation that $h(x) = (g(h))(f(x))$, we observe that running L with target c and distribution D is equivalent to running L_m with target $g(c)$ and some distribution D' over $\{0,1\}^{mn}$ (which is simply the D under f). Therefore by the PAC property of L_m, we may choose sample size m which polynomially varies with $\frac{1}{\delta}, \frac{1}{\epsilon}$, and see that with probability at least $1-\delta$,

$$
\begin{aligned}
\varepsilon_D(g^{-1}(h)) &= \Pr_{x\sim D}(c(x) \neq g^{-1}(h)(x)) \\
&= \Pr_{x\sim D}((g(c))(f(x)) \neq (h)(f(x))) \\
&= \Pr_{x\sim D'}((g(c))(x) \neq (h)(x)) < \epsilon
\end{aligned}
$$

We now make a brief remark on the above analysis. We have thus far assumed that any variable x_j will not appear in the a single conjunction more than once. If we relax this assumption and allow x_j and $\neg x_j$ to appear in a single term of a m-term DNF, then we can trivially modify our f and g to preserve the analysis. In particular we may instead make f map into $\{0,1\}^{2mn}$, by $f(x_1,...,x_n) = (x_1,..,x_n,x_1,...,x_n)$. Accordingly, define g to map x_j and $\neg x_j$ to x_{j+n} and x_{j+2n} in the second term, take x_j and $\neg x_j$ to x_{j+3n} and x_{j+4n} in the third term and map $\neg x_j$ to x_{j+5n} in the first term.

Finally, we can generalize our entire analysis to other boolean classes. In particular, suppose $\mathcal{C}$ is some boolean class for any single variables x_j occurs at most some k times. If the read-once counterpart of $\mathcal{C}$ is PAC learnable, we may take some g that maps repeated instances of x_j in a formula in $\mathcal{C}$ to distinct $x_{j+\alpha n}$ and $x_{j+\beta n}$. Then by choosing the appropriate f, we can replicate the above analysis and conclude that PAC learnability of the read-once counterpart implies PAC learnability of $\mathcal{C}$.

Problem 5.

Recall that if L is a PAC learning algorithm, then for any distribution $\mathcal{D}$ and inputs ϵ, δ, m, L returns a hypothesis h such that $\varepsilon(h) < \epsilon$ with probability $1-\delta$ using $O(poly(1/\epsilon, 1/\delta, m, n)$ i.i.d. samples from $\mathcal{D}$ in $O(poly(1/\epsilon, 1/\delta, m, n)$ time.

Note that this above definition actually isn't complete in this specific case. We need to add the caveat that the examples in our samples are labeled by (i.e. consistent with) a DNF with m terms. When this is not the case, we do not get any guarantees from PAC. L could either crash and burn, or worse, output an arbitrarily (bad) hypothesis.

Note also that every m-term DNF can be written as an equivalent m' DNF for all $m' > m$. There are several ways to do this. For example, we could pad by taking some term in the original DNF and repeat it multiple times (since $T_1 \vee \cdots \vee T_1 \iff T_1$). As long as we pass in some m' to L which is larger than m (which we do not know a priori), we have PAC guarantees.

One trivial approach is to just guess $m' = 3^n$ (since every DNF is equivalent to a DNF with only distinct terms, and there are 3^n possible distinct terms). Then L is guaranteed to output a consistent hypothesis. But m' could be exponentially larger than m, so our algorithm would run in exponential time with respect to m as L depends polynomially on m'. This fails the PAC runtime efficiency conditions.

It remains to show how we can be smarter about choosing our m' to still guarantee polynomial runtime in m. This is where the "doubling" trick comes in. We can search for m' by guessing it in powers of two, and running L on each $m' = 2^i$ until we're satisfied with its hypothesis. Note that, if we are able to perfectly detect when L gives us a "good" hypothesis or always return a hypothesis with error $< \epsilon$, then we simply stop at $m' = \min_i\{2^i : 2^i > m\}$; so that $m' < 2 * m$, satisfying the "polynomial in m" requirement.
However, because we only have a PAC algorithm, we are only able to produce a hypothesis with error $< \epsilon$ with probability $1 - \delta$. So the goal now is to find a way to detect if the hypothesis produced by L is good with high probability.
To do this, we use Chernoff-Hoefding bounds! The idea will be to *test* every hypothesis we get with a newly drawn sample and then stop at the first hypothesis which gives us a "good enough" test error.
We implement this idea in the following algorithm L.

```
**L**$'(\epsilon, \delta, \mathcal{D})$:
    (1) Initialize $i = 0$.
    (2) **Training:** Run $L(\epsilon/3, \frac{1}{3*2^c}, m = 2^i)$ to get hypothesis $h_i$. If $2^i > 3^n$, **return** $h_i$.
    (3) **Testing:**
        (a) Draw a sample $S_i \sim D$ of size $p = \frac{5}{\epsilon^2}\ln(\max\{3 * 2^c, \frac{2n}{\delta}\})$ (see analysis for why).
        (b) If $\hat{\epsilon}_{S_i}(h_i) < \frac{2\epsilon}{3}$, **return** $h_i$. Otherwise, repeat from (2) with $i = i + 1$
```

c is defined to be the largest dependence that L has on m, across sample size and runtime. For example if the runtime of L is $O(m^3)$ and sample complexity is $O(m^4)$, then $c = 4$.

Correctness:
We'll first show that L' is indeed probabilistically approximately correct. Let h' denote the hypothesis returned. And let B denote the bad event:

$$B := [\epsilon(h') > \epsilon]$$

When can this event occur? Well, it's the union of B_i, defined as the event that h_i is bad, and $h' = h_i$ (i.e. we accepted h_i). Therefore, we can upperbound $Pr[B]$ by the union bound.

$$Pr[B] \leq \sum_{i=1}^{n \log 3} Pr[B_i]$$

Therefore, it suffices to upper bound every $Pr[B_i]$ by $Pr[B_i] \leq \frac{\delta}{n \log 3} < \frac{\delta}{2n}$ Now, $Pr[B_i]$ we can (loosely) upper bound by $Pr[\text{accept } h_i \mid \epsilon(h_i) > \epsilon]$. Formally, we see that

$$Pr[B_i] \leq Pr_{S_i}[\epsilon - \hat{\epsilon}_{S_i}(h_i) > \epsilon/2 \mid \epsilon(h_i) > \epsilon] \leq Pr_{S_i}[\epsilon(h_i) - \hat{\epsilon}_{S_i}(h_i) > \epsilon/3]$$

By Chernoff-Hoefding bounds, we have:

$$Pr_{S_i}[\epsilon(h_i) - \hat{\epsilon}_{S_i}(h_i) > \epsilon/2] \le e^{-2p(\epsilon/3)^2}$$

Plugging in our value for p results in $e^{-2p(\epsilon/3)^2} \le \frac{\delta}{2n}$

Sample and runtime complexity

Lastly, we need to make sure that this algorithm uses time and sample size expected polynomial in all relevant parameters: ϵ, δ, n, m.

First, some definitions:

(1) Let T_i be the runningtime of L' on round i. Note that each T_i can be different, since L' has time complexity which can depend on the number of terms m.
(2) Let P_i be the probability that we reach round i
(3) Let k be the smallest natural number for which $2^k \ge m$. I.e. $k = \lceil \log(m) \rceil$

First, we observe that the time it takes to complete the runs for which $m'_i < m$ (where $m'_i = 2^i$ is the m we pass to L at round i) is polynomial.

$$\begin{aligned}\sum_{i=0}^{k-1} P_i T_i &\le \sum_{i=0}^{k-1} T_i \\ &\le k \max_i(T_i) \\ &\le \lceil \log(m) \rceil \text{ poly}(\frac{4}{\epsilon}, 3 * 2^c, n, m'_{i^*}) \\ &= \text{poly}(\frac{1}{\epsilon}, \frac{1}{\delta}, n, m)\end{aligned}$$

In the second to last line, $i^* = \arg\max_i(T_i)$. And we obtain the final expression by absorbing the constants and the $\log(m)$ factor, and recalling that $m'^*_i < m$ by definition.

Second, we want to upperbound the expected amount of time (samples) we spend on rounds with $m'_i > m$. Note that above, we showed that the runtime is *guaranteed* polynomial, since in our analysis, we completely ignored the P_i's. However, now the P_i's play an important role. Since there's always a chance that we keep getting bad test sets, we might try all m' up to 3^n. We need to ensure that the probability of reaching large m'_i (relative to the true m) is "small enough." More formally:

Define $U_j = T_{j+k}$ be the running time for the jth round for which $m' > m$. Observe:

$$\begin{aligned}U_j &= T_{j+k} \\ &= T_k * \frac{T_{j+k}}{T_k} \\ &\le T_k * \left(\frac{m'_{j+k}}{m'_k}\right)^{c*j} \\ &= T_k * 2^{c*j}\end{aligned}$$

Recalling that T_k is polynomial in all problem parameters, it suffices if the probability to reach the jth extra round is bounded by $\frac{1}{2^{cj}}$.

Define $Q_j = P_{j+k}$ to be the probability we reach the jth round after $m' > m$. And define R_i to be the probability that, given that we've reach round i, we reject the ith hypothesis and keep going. Note that:

$$Q_j = \prod_{i=0}^{j+k-1} R_i \le \prod_{i=k}^{j+k-1} R_i$$

The idea is that, when $m'_i > m$, the probability that we accept is high (and so the probability that we reject is low). The only times we reject are if: (a) L fails to produce an $\epsilon/4$ accurate hypothesis, or (b) L produces an accurate hypothesis, but the sample we draw is unrepresentative. Define R_i^a, R_i^b to be the probabilities of these respective events, where from now on, we assume $i > k$.
R_i^a follows from the δ parameter we passed to L, and the fact that L is a PAC learning algo:

$$R_i^a \leq \delta_i = \frac{1}{3 * 2^c}$$

R_i^b follows from the exact same Chernoff bound analysis as for the correctness analysis from before, with the desired probability bound of $\frac{\delta}{2n}$ replaced with $\frac{1}{3*2^c}$.

$$R_i^b \leq \delta_i = \frac{1}{3 * 2^c}$$

Alright... we can finally start piecing things back together. Applying union bound, we get:

$$R_i \leq R_i^a + R_i^b = \frac{2}{3 * 2^c} < \frac{1}{2^c}$$

Plugging back into the Q_js:

$$Q_j \leq \prod_{i=k}^{j+k-1} R_i < \frac{1}{2^{cj}}$$

And finally, we can use this to compute the expected runningtime above m:

$$\begin{align}
\sum_{i=k}^{n\log 3} P_i T_i &= \sum_{j=0}^{n\log 3-k} Q_j U_j \tag{1}\\
&< \sum_{j=0}^{2n} T_k * 2^{cj} * \frac{1}{2^{cj}} \tag{2}\\
&= 2nT_k \tag{3}\\
&= \text{poly}(\frac{1}{\epsilon}, \frac{1}{\delta}, n, m) \tag{4}
\end{align}$$

Finally... we can conclude that the *expected* time complexity of this process is polynomial in all relevant parameters. Note that we only showed this for time complexity, but you can derive the same result for sample complexity by following the above analysis.

Notice that we didn't use much about the fact that we were specifically learning DNFs. The only part we somewhat used that fact was in stating that m was bounded by 3^n. Because of the generality of the proof, we can observe that the same overall algorithm can be used to turn any PAC learning algorithm which depends on a bounded integral parameter into an (expected polynomial) PAC learning algorithm which does not require the parameter as input.