Machine Learning HW1 - Theory

1. **Question:** Let D be a distribution on the domain $\{-1,1\}^n$ and let $f, g : \{-1,1\}^n \to \{-1,1\}$ be two Boolean functions. Prove that

$$\mathbb{P}_{x \sim D}[f(x) \neq g(x)] = \frac{1 - \mathbb{E}_{x \sim D}[f(x)g(x)]}{2}$$

Answer:

Notice that we have 2 complementary events: $A_= = \{x | f(x) = g(x)\}$ and $A_{\neq} = \{x | f(x) \neq g(x)\}$. Thus $\mathbb{P}_{x \sim D}[A_=] + \mathbb{P}_{x \sim D}[A_{\neq}] = 1$.

By definition,

$$\begin{aligned}\mathbb{E}_{x \sim D}[f(x)g(x)] &= \sum_x f(x)g(x)D(x) \\ &= \sum_{x \in A_=} f(x)g(x)D(x) + \sum_{x \in A_{\neq}} f(x)g(x)D(x)\end{aligned}$$

Using the fact that $g(x_=) = f(x_=)$ for $x_= \in A_=$ and $g(x_{\neq}) = -f(x_{\neq})$ for $x_{\neq} \in A_{\neq}$ (since $f(x_{\neq}) = 1$ implies $g(x_{\neq}) = -1$ and $f(x_{\neq}) = -1$ implies $g(x_{\neq}) = 1$) then

$$\begin{aligned}f(x_=)g(x_=) &= f^2(x_=) = (1 \text{ or } -1)^2 = 1 \\ f(x_{\neq})g(x_{\neq}) &= f(x_{\neq})(-f(x_{\neq})) = -f^2(x_{\neq}) = -1\end{aligned}$$

Therefore

$$\begin{aligned}\mathbb{E}_{x \sim D}[f(x)g(x)] &= \sum_{x \in A_=} 1 * D(x) + \sum_{x \in A_{\neq}} (-1) * D(x) \\ &= \mathbb{P}_D[A_=] - \mathbb{P}_D[A_{\neq}] \\ &= (1 - \mathbb{P}_D[A_{\neq}]) - \mathbb{P}_D[A_{\neq}] \\ &= 1 - 2\mathbb{P}_D[A_{\neq}] \iff \\ \mathbb{P}_{x \sim D}[f(x) \neq g(x)] &= \frac{1 - \mathbb{E}_{x \sim D}[f(x)g(x)]}{2}\end{aligned}$$

Q.E.D.

Question: Would this still be true if the domain were some other domain, such as $\mathbb{R}^n$?

Answer: Not necessarily, depending on some fundamental assumptions. If we are working in ZFC, then by using the axiom of choice we can construct a wacky non-measurable set W by the Vitali method on the pre-image of our distribution D so that $W \subset D^{-1}(\mathbb{R}^n)$. Then we can pick f as

$$f(x) = \begin{cases} \text{-1 if } x \in W \\ \text{1 if } x \notin W \end{cases}$$
$$g(x) = 1$$

Then $\mathbb{P}[f(x) \neq g(x)] = \int_W D(x)dx$ which is not well-defined.

Assuming that the sets $A_=$ and $A_{\neq}$ are measurable, then the result holds by a similar argument to our previous one:

$$\begin{aligned} \mathbb{E}_{x \sim A}[f(x)g(x)] &= \int_{A_=} f(x)g(x)D(x)dx + \int_{A_{\neq}} f(x)g(x)D(x)dx \\ &= \int_{A_=} 1 * D(x)dx + \int_{A_{\neq}} (-1) * D(x)dx \\ &= \int_{A_=} D(x)dx - \int_{A_{\neq}} D(x)dx \\ &= \mathbb{P}_= - \mathbb{P}_{\neq} \\ &= 1 - 2\mathbb{P}_{\neq} \iff \\ \mathbb{P}_{x \sim D}[f(x) \neq g(x)] &= \frac{1 - \mathbb{E}_{x \sim D}[f(x)g(x)]}{2} \end{aligned}$$

2. **Question:** Let f be a decision tree with t leaves over the variables $x = (x_1, ..., x_n) \in \{-1, 1\}^n$. Explain how to write f as a multivariate polynomial $p(x_1, ..., x_n)$ such that for ever input $x \in \{-1, 1\}^n, f(x) = p(x)$.

Answer:

I initially started with a more complicated proof, but I'm hoping a simpler one that can be extensible will be clearer.

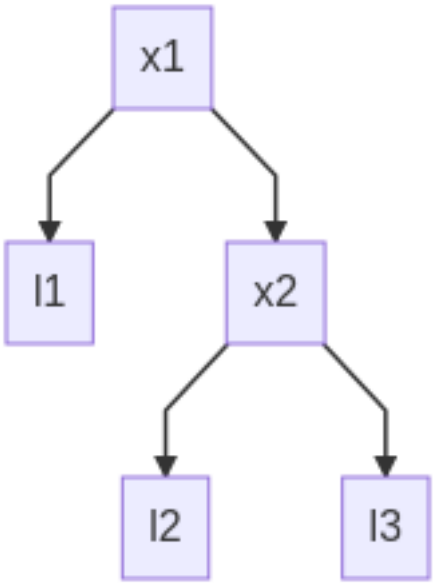
Assume that the tree's leaf values are $l_1, l_2 ... l_t \in \{-1, 1\}$. We can define f as

$$f(x) = \begin{cases} l_1 \text{ if } \vee_j \wedge_{k=1}^n x_k = v_{1jk} \\ l_2 \text{ if } \vee_j \wedge_{k=1}^n x_k = v_{2jk} \\ ... \\ l_t \text{ if } \vee_j \wedge_{k=1}^n x_k = v_{tjk} \end{cases}$$

Doesn't look much like a tree, but you can read it as this:

The function will return the value of leaf l_i if each of the values in the input matches a particular path in the tree determined by values v_{ijk} for some i and j.

For example:



$$f_{\text{eg}}(x) = \begin{cases} l_1 \text{ if } (x_1 = -1 \wedge x_2 = -1) \vee (x_1 = -1 \wedge x_2 = 1) \\ l_2 \text{ if } (x_1 = 1 \wedge x_2 = -1) \\ l_3 \text{ if } (x_1 = 1 \wedge x_2 = 1) \end{cases}$$

Note that the or implies that multiple input values can lead to the same path; this is because some x_i values don't count towards some particular paths.

Let's look at the inverse images of f:

$$f^{-1}(1) = N_1 = \{v = (v_1, ..., v_n) \in \{-1, 1\}^n | f(v) = 1\}$$
$$f^{-1}(-1) = N_{-1} = \{v = (v_1, ..., v_n) \in \{-1, 1\}^n | f(v) = -1\}$$

N_1 and N_{-1} are disjoint; if a value was to be in both sets, then $f(v) = 1 = -1$, which is a contradiction.

Define

$$I_v(x) = \prod_{k=1}^{n} \frac{(x_k + v_k)^2}{2^2}$$
$$p(x) = (1) \sum_{v \in N_1} I_v(x) + (-1) \sum_{v \in N_{-1}} I_v(x)$$

Pick $x = a = (a_1, ..., a_n) \in \{-1, 1\}^n$.

Then $\forall v \neq a \exists k$ s.t. $v_k \neq a_k$, otherwise v_k would agree with a_k on all values k, so v would have to equal a (contradiction). This implies that $v_k = -a_k$ (via binarity of the space). So $I_v(a) = 0$ since at least one of the terms of the product will be $v_k + a_k = 0$.

When $v = a$ then

$$I_v(a) = \prod_{k=1}^{n} \frac{(a_k + v_k)^2}{2^2} = \prod_{k=1}^{n} \frac{(2a_k)^2}{2^2} = \prod_{k=1}^{n} (1 \text{ or } -1)^2 = 1$$

If $f(a) = 1$, then $a \in N_1$ so $p(a) = (1) \cdot (I_a(a) + \sum_{v \in N_1, v \neq a} I_v(a)) + (-1) \cdot (\sum_{v \in N_{-1}, v \neq a} I_v(a)) = (1)(1+0) + (-1)0 = 1$.

If $f(a) = -1$, then, by a similar argument, $p(a) = -1$.

Thus $f(x) = p(x)$ for any x.

Q.E.D.

Note 1: you could drop the "ors" and thus some of the redundant terms in the polynomial to only cover the relevant paths to a leaf. It would make the proof more complicated, but that's probably the more compact form.

Note 2: You can apply the same proof for any label domain, not just the binary one.

3. **Question:** Compute a depth-two decision tree for the training data in table 1 using the Gini function, C(a) = 2a(1 − a) as described in class. What is the overall accuracy on the training data of the tree?

x_1	x_2	x_3	Number of positive examples	Number of negative examples
0	0	0	10	20
0	0	1	25	5
0	1	0	35	15
0	1	1	35	5
1	0	0	5	15
1	0	1	30	10
1	1	0	10	10
1	1	1	15	5

Answer:

Let's use −1 for a negative outcome and 1 for a positive outcome.

$\mathbb{P}(y = -1) = \frac{85}{250} = \frac{17}{50}$

Let's pick a root.

$$\begin{aligned}
\text{gain}(x_1) &= C(\mathbb{P}[y=-1]) - (\mathbb{P}[x_1=0]C(\mathbb{P}[y=-1|x_1=0]) + \mathbb{P}[x_1=1]C(\mathbb{P}[y=-1|x_1=1])) \\
&= \frac{561}{1250} - (\frac{3}{5} \cdot \frac{21}{50} + \frac{2}{5} \cdot \frac{12}{25}) = \frac{3}{625} \\
\text{gain}(x_2) &= C(\mathbb{P}[y=-1]) - (\mathbb{P}[x_2=0]C(\mathbb{P}[y=-1|x_2=0]) + \mathbb{P}[x_2=1]C(\mathbb{P}[y=-1|x_2=1])) \\
&= \frac{561}{1250} - (\frac{12}{25} \cdot \frac{35}{72} + \frac{13}{25} \cdot \frac{133}{338}) = \frac{529}{48750} \\
\text{gain}(x_3) &= C(\mathbb{P}[y=-1]) - (\mathbb{P}[x_3=0]C(\mathbb{P}[y=-1|x_3=0]) + \mathbb{P}[x_3=1]C(\mathbb{P}[y=-1|x_3=1])) \\
&= \frac{561}{1250} - (\frac{12}{25} \cdot \frac{1}{2} + \frac{13}{25} \cdot \frac{105}{338}) = \frac{384}{8125}
\end{aligned}$$

So x_3 is the winner with $\frac{384}{8125}$. Let's look at the next level

$$
\begin{aligned}
\text{gain}(x_1|x_3=0) &= C(\mathbb{P}[y=-1|x_3=0]) \\
&\quad - (\mathbb{P}[x_1=0|x_3=0]C(\mathbb{P}[y=-1|x_1=0\wedge x_3=0]) \\
&\quad + \mathbb{P}[x_1=1|x_3=0]C(\mathbb{P}[y=-1|x_1=1\wedge x_3=0])) \\
&= \frac{1}{2} - (\frac{2}{3}\cdot\frac{63}{128} + \frac{1}{3}\cdot\frac{15}{32}) = \frac{1}{64} \\
\text{gain}(x_2|x_3=0) &= C(\mathbb{P}[y=-1|x_3=0]) \\
&\quad - (\mathbb{P}[x_2=0|x_3=0]C(\mathbb{P}[y=-1|x_2=0\wedge x_3=0]) \\
&\quad + \mathbb{P}[x_2=1|x_3=0]C(\mathbb{P}[y=-1|x_2=1\wedge x_3=0])) \\
&= \frac{1}{2} - (\frac{5}{12}\cdot\frac{21}{50} + \frac{7}{12}\cdot\frac{45}{98}) = \frac{2}{35} \\
\text{gain}(x_1|x_3=1) &= C(\mathbb{P}[y=-1|x_3=1]) \\
&\quad - (\mathbb{P}[x_1=0|x_3=1]C(\mathbb{P}[y=-1|x_1=0\wedge x_3=1]) \\
&\quad + \mathbb{P}[x_1=1|x_3=1]C(\mathbb{P}[y=-1|x_1=1\wedge x_3=1])) \\
&= \frac{105}{338} - (\frac{7}{13}\cdot\frac{12}{49} + \frac{6}{13}\cdot\frac{3}{8}) = \frac{27}{4732} \\
\text{gain}(x_2|x_3=1) &= C(\mathbb{P}[y=-1|x_3=1]) \\
&\quad - (\mathbb{P}[x_2=0|x_3=1]C(\mathbb{P}[y=-1|x_2=0\wedge x_3=1]) \\
&\quad + \mathbb{P}[x_2=1|x_3=1]C(\mathbb{P}[y=-1|x_2=1\wedge x_3=1])) \\
&= \frac{105}{338} - (\frac{7}{13}\cdot\frac{33}{98} + \frac{6}{13}\cdot\frac{5}{18}) = \frac{4}{3549}
\end{aligned}
$$

So the local maxima are - $\text{gain}(x_2|x_3=0) > \text{gain}(x_1|x_3=0)$ and $\text{gain}(x_1|x_3=1) > \text{gain}(x_2|x_3=1)$.

Side-stepping ID3, we can also look at the decision between branches - what gain overall maximized?

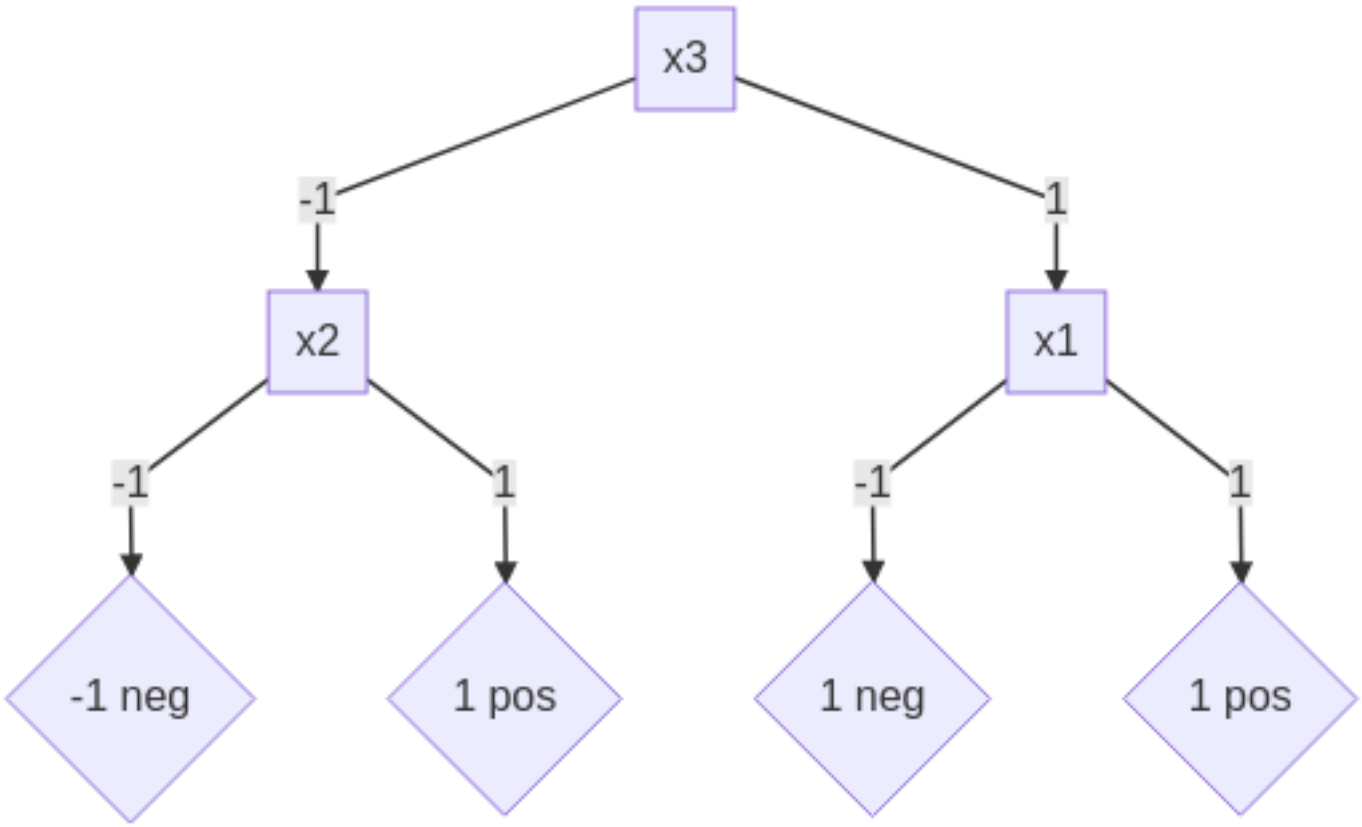
$$\text{gain}(x_2|x_3=0) + \text{gain}(x_1|x_3=1) = \frac{1487}{23660} > \frac{3805}{227136} = \text{gain}(x_1|x_3=0) + \text{gain}(x_2|x_3=1)$$

So the choice of algorithm doesn't impact our tree. (Interesting Q: is this always the case?).

Now to compute the leaves:

$$
\begin{aligned}
l(x_3=0, x_2=0) &= -1(35 \text{ outcomes out of } 50) \\
l(x_3=0, x_2=1) &= 1(45 \text{ outcomes out of } 70) \\
l(x_3=1, x_1=0) &= 1(60 \text{ outcomes out of } 70) \\
l(x_3=1, x_1=1) &= 1(45 \text{ outcomes out of } 60)
\end{aligned}
$$

Our hard-worked tree:



I'll include the code I used to compute some of the results at the end of the doc.

Question: What is the overall accuracy on the training data of the tree?

Answer: We can look at the numbers we used to compute the leaves to figure out the number of values the tree agrees with the training data. Accuracy $= \frac{35+45+60+45}{50+70+70+60} = \frac{185}{250} = .74$.

4. **Question:** Suppose the domain X is the real line, $\mathbb{R}$, and the labels lie in $Y = \{-1, 1\}$, let C be the concept class consisting of simple threshold functions of the form h_θ for some $\theta \in \mathbb{R}$, where $h_\theta(x) = -1$ for all $x \leq \theta$ and $h_\theta(x) = 1$ otherwise. Give a simple and efficient PAC learning algorithm for C that uses only $m = O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$ training examples to output a classifier with error at most ϵ with probability at least $1 - \delta$.

Answer:

Let

$$a = \max(x|S(x) = -1)$$

W.l.o.g. assume a exists. If $\{x|S(x) = -1\} = \emptyset$ then use the negative bijection $f(x) = -x$ to 'flip' the set, and have a value for a (assuming that $S \neq \emptyset$ from the get-go). You can also alternatively pick $a = \min(x|S(x) = 1)$ in the case where it doesn't exist; it makes the proof more complicated.

Let algorithm A output classifier

$$h_S(x) = \begin{cases} -1 \text{ if } x \leq a \\ 1 \text{ if } x > a \end{cases}$$

Good to note that $a \leq \theta$; if $\theta < a$, then our training set would have values that don't agree with h_θ (which is a contradiction, and an offense to our perfectly realizable oracle).

Pick $\theta_a < \theta$ such that $\mathbb{P}[x \in (\theta_a, \theta)] = \epsilon$.

If $\theta_a < a$ then using the fact that $err_{x \leq a}(h_S) = 0$ because $h_S(x) = h_\theta(x)$ on all values of $x \leq a$, $err_{x>\theta}(h_S) = 0$ because $h_S(x) = h_\theta(x)$ on all values $x > \theta$.

$$\begin{aligned} err(h_S) &= err_{x \leq a}(h_S) + err_{a<x}(h_S) \\ &= 0 + err_{a<x}(h_S) \\ &\leq err_{\theta_a<x}(h_S) \\ &= err_{\theta_a<x\leq\theta}(h_S) + err_{\theta\leq x}(h_S) \\ &= \epsilon + 0 = \epsilon \end{aligned}$$

So if $\theta_a < a$ then $err(h_S) \leq \epsilon$. Then

$$\begin{aligned} \mathbb{P}[err(h_S) > \epsilon] &= \mathbb{P}[\theta_a \geq a] \\ &= \mathbb{P}[S \cap (\theta_a, \theta) = \emptyset] \\ &= (1-\epsilon)^m \\ &\leq e^{-\epsilon m} \leq \delta \iff \\ -\epsilon m &\leq \ln(\delta) \iff \\ m &\geq -\frac{1}{\epsilon}\ln(\delta) \iff \\ m &\geq \frac{1}{\epsilon}\ln(\frac{1}{\delta}) \end{aligned}$$

Thus algorithm A satisfies outputting a classifier h_S with error at most ϵ with probability at least $1-\delta$ conditioned on $|S| = m = O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$.

5. a. **Question:** Fix a classifier $h : X \to Y$. If $err(h) > \epsilon$, what is the probability that h gets k random examples all correct? How large does k need to be for this probability to be at most δ'?

Answer:

$$\begin{aligned} \mathbb{P}_{x \sim S_k}[h(x) = c(x)] &= (1 - \mathbb{P}_{x \sim S_k}[h(x) \neq c(x)])^k \\ &< (1-\epsilon)^k \\ &\leq e^{-\epsilon k} \leq \delta' \iff \\ k &\geq \frac{1}{\epsilon} ln(\frac{1}{\delta'}) \end{aligned}$$

So for $k \geq \frac{1}{\epsilon} ln(\frac{1}{\delta'})$ we know the probability of getting k examples all correct is less than δ'.

b. **Question:** How many times can A possibly update its state? That is, how many different classifiers can it possibly go through? And therefore, how many examples do we need to see before we can be sure of getting a block of k examples all correct?

Answer: A can update its state t times, since we assume it updates its state only on a mistake, and it can make at most t of those. If A makes k mistakes on inputs $x_i, 0 \leq i \leq k$ and updates its state k times, then we know there is at least one block of size k where $h(x) = c(x)$, namely $\{x_i\}$.

So let's assume we see k blocks but don't get k mistakes. That means, by the pigeon hole principle, at least one of the blocks doesn't have a mistake, so we have a guarantee of an errorless k block.

So after seeing k^2 examples we know we either have encountered a k-sized block with no error, or have a guarantee of encountering one.

c. **Question:** Let E_i be the event that A in its ith state, call it h_i, has error greater than ϵ, yet we output h_i. (When would this happen?) Argue that the "failure event" of our overall PAC learner is $E = \cup_i E_i$.

We would output an insufficient h_i that has $err(h_i) > \epsilon$ if we haven't covered enough mistakes to progress A to a state $i+1$. In other words, we would output h_i if we are "unlucky" enough to have less than i blocks with mistakes, and more than $|S|/i - i$ blocks with $h_i(x) = c(x)$.

Note that there exists a k such that $err(h_k) \leq \epsilon$. This k exists because the error decreases monotonically $err(h_i) > err(h_{i+1})$ and $err(h_t) = 0$. Error decreases monotonically because each update decreases the number of values $h(x) and c(x)$ disagree on for at least one value of x. Thus if $err(h_i) > \epsilon, \exists k$ s.t. $err(h_i) > \epsilon \geq err(h_k) \geq err(h_t) = 0$.

The "failure event" $E = \cup_i E_i$ represents the event of outputting any hypotheses h_i having error greater than ϵ. Implicitly, if E does not happen, then we output any hypothesis h_k not in any E_i, then $err(h_k) \leq \epsilon$. If E happens, then we know $err(h_S) > \epsilon$. Thus E is equivalent with $err(h_S) > \epsilon$.

d. **Question:** Put everything together and fully describe a PAC learner that is able, with probability of failure at most δ, to output a classifier with error at most ϵ. How many examples does the learner need to use?

Answer: Let $h_S = h_k$ be the hypothesis returned by algorithm A run on sample S such that we return current A hypothesis h_i if we process a block of size i with no errors. If for any h_i intermediate hypotheses $err(h_i) \leq \epsilon$, then from point c. we know that $err(h_s) < \epsilon$. Let's find a bound for the failure event.

Using c:

$$\begin{aligned} \mathbb{P}[err(h_S) > \epsilon] = \mathbb{P}[E] &= \mathbb{P}[\cup_i E_i] \\ &\leq \sum_i \mathbb{P}[E_i] \end{aligned}$$

Using b. we know that $\mathbb{P}[E_i] = \mathbb{P}_{x \sim \text{ block of size} i}[h_i(x) = c(x)]^{\frac{|S|}{i} - (i+1)}$.

Using a. we know that $\mathbb{P}[h_i(x) = c(x)] < e^{-\epsilon k}$, and we know we're summing less than t terms, and from b we know that $k < \sqrt{(|S|)}$, so all together now:

$$\begin{aligned}\mathbb{P}[err(h_S) > \epsilon] &< \sum_i e^{-\epsilon i^2} \\ &< te^{-\epsilon|S|} \leq \delta \iff \\ |S| &\geq \frac{1}{\epsilon}(ln(\frac{1}{\delta}) - ln(t))\end{aligned}$$

Which implies that mistake-bounded learning is stronger in the PAC model.

Q3 Code Annex ruby helper script:

```
info = "
0 & 0 & 0 & 10 & 20\\
0 & 0 & 1 & 25 & 5\\
0 & 1 & 0 & 35 & 15\\
0 & 1 & 1 & 35 & 5\\
1 & 0 & 0 & 5 & 15\\
1 & 0 & 1 & 30 & 10\\
1 & 1 & 0 & 10 & 10\\
1 & 1 & 1 & 15 & 5\\
"

t = info.gsub("&", "").split("\\\n").map { |x| x.split("  ").map(&:to_i) }

def gain(t, i)
  gini = -> (x) do
    2 * x * (1 - x)
  end

  all = -> (x) do
    x.map { _1[3] + _1[4] }.sum
  end

  neg = -> (x) do
    x.map { _1[4] }.sum
  end

  c_init = Rational(t.then(&neg), t.then(&all)).then(&gini)

  p_0 = Rational(t.select { _1[i] == 0}.then(&all),
    t.then(&all))

  c_0 = Rational(t.select { _1[i] == 0 }.then(&neg),
    t.select { _1[i] == 0 }.then(&all)).then(&gini)

  p_1 = Rational(t.select { _1[i] == 1}.then(&all),
    t.then(&all))

  c_1 = Rational(t.select { _1[i] == 1 }.then(&neg),
    t.select { _1[i] == 1 }.then(&all)).then(&gini)

  return c_init, p_0, c_0, p_1, c_1, c_init - (p_0 * c_0 + p_1 * c_1)
end

# for leaf computation
t.select { _1[2] == 1 && _1[0] == 1 }.then { "-1 #{neg.(_1)}, 1 #{all.(_1) - neg.(_1)}, #{all
```