$\boxed{\text{Boosting}}$ = algorithm used to improve accuracy of weak learners. wrapper around other classifiers.

Recall PAC learning:

- δ = probability of failure.
- ϵ = accuracy parameter. want to make small.

PAC learning requirement:

For any choice of ϵ, δ, A should output with probability $\geq 1-\delta$, an ϵ-accurate classifier.

A is allowed to run in time poly$\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$, Take # of samples poly$\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$.

Question: what if we have an algorithm A with probability 5% outputs an ϵ-accurate classifier. How can we use A to obtain a standard PAC learner?

we want to increase that 5% probability of success to $1-\delta$.

Solution: Run A a large # of times, say, t.

$$p\left[A \text{ fails to output an } \epsilon\text{-accurate classifier}\right] \leq \underbrace{(0.95)^t}$$

probability it fails every time for t trials.

we can make $(0.95)^t$ very small by choosing t to be approximately $O\left(\log \frac{1}{\delta}\right)$, then we can "test" each classifier generated during these t trials to see if any of them are good classifiers.

<u>Bottom Line</u>: Amplifying the probability of success is not too difficult. $5\% \longrightarrow 1-\delta$ by running the algorithm many times.

$$p(\text{at least 1 success}) = 1 - p(\text{all failures}) = 1 - (\text{failure prob})^{\#\text{ of trial}}$$

<u>Trickier Question</u>: What if ϵ is fixed to, say, 0.49?

Imagine A with probability $\geq 1-\delta$ outputs a classifier with $\epsilon = 0.49 \Rightarrow$ accuracy 51%.

<u>Natural Question</u>: How do we amplify / improve the accuracy parameter?

$\boxed{A} \longrightarrow h$ s.t. $err(h) = 0.49$

want $\boxed{A'} \longrightarrow h'$ s.t. $err(h') \leq \epsilon$

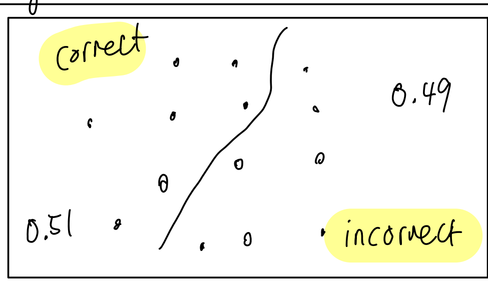
$\boxed{A}$ subroutine

Solution attempt: Try running A many times, $h_1, \dots, h_t$. Take the majority $(h_1, \dots, h_t)$.

Doesn't make sense, since ϵ is the same for all.

Question was posed by Valiant on PAC learning.
Solved by R. Schapire.
Solution by Freund and Schapire.
Algorithm presented: "Adaboost".

High level idea of adaboost:



← training set.

Run A on uniform distribution on points in the training set.

Assumes A outputs classifier h w/ $\epsilon = 0.49$.

Fraction of correct points is 0.51.

New distribution:

Put a little less weight on those we got right.
" " " more " wrong.

Divide by sum of total weights → new probability dist on points that we got wrong.

core idea:

- re-weight points we get wrong: have more weight.
- re-weight points we got right: have less weight.
- Run algorithm A again, to obtain classifier wrt the new weighting.
- At the end, take majority of classifiers generated during the process.

Adaboost (simplified) Algorithm: Ada = adapt

- training set of size m.

1) Initially D_0 = uniform distribution corresponds to $w_i = 1 \ \forall i$: Each point has weight 1.

   Dist is obtained by dividing W = sum of weights.
   capital W

2) Let:

error rate $\equiv E$. At each iteration, assume: $E = \frac{1}{2} - \gamma$

accuracy $A \equiv 1 - E$.

update factor $\beta \equiv E/A = \dfrac{E}{1-E} = \dfrac{\frac{1}{2}-\gamma}{1-E} = \dfrac{\frac{1}{2}-\gamma}{1-\left(\frac{1}{2}-\gamma\right)}$

$$= \boxed{\frac{\frac{1}{2}-\gamma}{\frac{1}{2}+\gamma}} \quad \leq 1 \ (?)$$

3) How to update the weights:

At iteration t, run A to obtain h_t.

- For each x_i s.t. $h_t(x_i)$ is correct:

  update $w_i^{new} = \beta \cdot w_i^{old}$

- For each x_i s.t. " incorrect: do nothing.

4) Repeat for T steps, output $MAJ(h_1, \dots, h_t)$.

$\boxed{\text{Claim: After } T \text{ iterations, error } h_{final} = MAJ(h_1, \cdots, h_T) \le e^{-2T\gamma^2} \Rightarrow \text{choose } T \approx \frac{1}{\gamma^2}\log\left(\frac{1}{\epsilon}\right) \text{ then error of } h_{final} \le \epsilon.}$

Total weight after an iteration:

- W is the weight of all points before iteration t.
- What is the weight of correct points after iteration t?

$$W_t^+ = \underset{\text{accuracy}}{A} \cdot \underset{\text{update factor}}{\beta} \cdot W = \left(\frac{1}{2} + \gamma\right) \cdot \beta \cdot W$$

- What is weight of the incorrect points after iteration t?

$W_t^- = \underset{\text{error rate}}{E} \cdot W = \left(\frac{1}{2} - \gamma\right) \cdot W$ since we don't update their weights.

- New sum of all weights (add W_t^+ and W_t^-).

$$W_t^+ + W_t^- = W\left(\frac{1}{2}\beta + \gamma\beta + \frac{1}{2} - \gamma\right)$$

$$\Rightarrow W\left(\left(\frac{1}{2} + \gamma\right) \cdot \beta + \frac{1}{2} - \gamma\right) = W\left(\cancel{\left(\frac{1}{2} + \gamma\right)} \frac{\left(\frac{1}{2} - \gamma\right)}{\cancel{\left(\frac{1}{2} + \gamma\right)}} + \frac{1}{2} - \gamma\right)$$

$= W(1-2\gamma) = W(2\cdot(\frac{1}{2}-\gamma))$

After i^{th} iterations, the sum of the weights =

$W_i = W_0(2(\frac{1}{2}-\gamma))^i$

After T iterations, the sum of all weights

$\le (2(\frac{1}{2}-\gamma))^T \cdot W_0$ ← initial weight.

Consider a point X_i that h_{final} gets wrong: that means

$W_T(X_i) \ge \beta^{T/2}$ (lower bound).

If h_{final} gets it wrong then it was wrong for at least $T/2$ iterations. So for at least $T/2$ iterations we didn't multiply by β. That means its weight has to be at least $\beta^{T/2}$.

$\Rightarrow$ If h_{final} has error ϵ, then weight of points h_{final} misclassifies $\ge \epsilon m \beta^{T/2}$ (lower bound).

$$\underbrace{\epsilon m \beta^{T/2}}_{\text{sum of weights of those classified incorrectly. Lower Bound.}} \le \underbrace{(2\cdot(\underbrace{\tfrac{1}{2}-\gamma}_{E}))^T \cdot m}_{\text{sum of all the weights. upper bound.}} \xRightarrow[m]{\text{cancel}} \epsilon \beta^{T/2} \le (2E)^T \xRightarrow[\beta^{T/2}]{\text{divide by}}$$

$$\epsilon \le \frac{(2E)^T}{\beta^{T/2}} \Rightarrow \epsilon \le \frac{(2E)^{2T/2}}{\beta^{T/2}} \Rightarrow \epsilon \le \left(\frac{4E^2}{\beta}\right)^{T/2}$$

$$\xRightarrow[\beta = \frac{\frac{1}{2}-\gamma}{\frac{1}{2}+\gamma}]{\text{using}} \epsilon \le (1-4\gamma^2)^{T/2} \le e^{-2\gamma^2 T} \quad (1+x \approx e^x)$$

$$\Rightarrow \boxed{\epsilon \le e^{-2T\gamma^2}}$$

The previous algorithm is simplified bc we assumed the accuracy was exactly $\frac{1}{2} + \gamma$. Can make small modifications such that if in some iterations you get a very accurate diagnosis, this algorithm will adapt and converge to a small error and quicker # of iterations.

Adaboost Modifications

In adaboost, $h_t \in \{-1, +1\}$

$$\beta_t = \frac{\varepsilon_t}{A_t}$$

weighing factor of each subclassifier.

Output: $\operatorname{sign}\left(\sum_t \alpha_t h_t - \frac{1}{2}\right)$, where $\alpha_t = \dfrac{\log\left(\frac{1}{\beta_t}\right)}{\sum_t \log\left(\frac{1}{\beta_t}\right)}$

subclassifier

How do we guarantee that h_{final} generalizes?

We need to make sure that my # of training points is sufficiently large.

If γ (accuracy) is independent from m (size of training set), then we can choose m to be sufficiently large.

Adaboost is a special case of an algorithm due to Freund and Schapire called "hedge" algorithm.

Hedge : "Best Expert" set up :

$C_1, \ldots, C_m$ at each iteration t, expert

C_i suffers a loss $l_i^t \in [0,1]$

l^t = a vector of losses suffered by all experts at the t^{th} iteration.

Intuition : we want to have a mixed strategy of experts, a weight average of experts.

<u>Goal</u> : the sum of our losses after T iterations should be "close" to the best expert in hindsight.

At each iteration, we maintain a set of weights :

$w_1, \ldots, w_m$ weighted average $= \dfrac{w_i}{\sum_i w_i} = P_i$

$w_1, \ldots, w_m \longrightarrow$ probability distribution p^t

$$\underbrace{p_i^t}_{\text{weighted average}} = \frac{w_i^t}{\sum_i w_i}$$

dot product.

☆ loss we suffer at the t iteration is $\underbrace{p^t \odot l^t}_{\text{weighted average of loss of experts.}}$

☆ Total loss we suffer after T iterations :

$$= \sum_{t=1}^{T} p^t \odot l^t$$

raised to power l_i^t

t^{th} iteration

<u>Hedge Algorithm</u> : $$w_i^{new} = w_i^{old} \cdot \beta^{l_i^t}$$

<u>Claim</u> : $\text{your loss} \le \underbrace{\min_i \sum_{t=1}^{T} l_i^t}_{\text{loss of best expert}} + O(\sqrt{T} \log n)$.

After T iterations, the sum of all of those average losses, is actually going to ~~be~~ the loss of the best expert (since we take a min over l), plus $O(\sqrt{T} \log n)$.

$$\text{Your average loss (total loss divided by T)} = \frac{\text{Your Loss}}{T} \le \frac{\min_i \sum_{t=1}^{T} l_i^t}{T} + \underbrace{\frac{O(\sqrt{T} \log n)}{T}}_{\text{goes to 0 quickly if T is large.}}$$

Hedge Algorithm = multiplicative weight update algorithm.

In contrast to additive update algorithms (GD or perceptron).