# CS 598: Theoretical Machine Learning

Lecturer: Pranjal Awasthi
Scribe: Pritish Sahu

Lecture #7

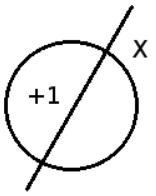
---

We have already seen in previous lectures that a concept class $\mathcal{C}$ is PAC-learnable by the hypothesis class $\mathcal{H}$ if there exists an algorithm A such that for all $c \in \mathcal{C}$, for all distributions D over X, for all $\epsilon > 0$, for all $\delta > 0$, A takes m = finite number of examples given in the form $\mathcal{S} = \langle (x_1, c(x_1)), ..., (x_m, c(x_m)) \rangle$, where each $x_i$ chosen from the space X at random according to the target distribution D, and produces a hypothesis $h \in \mathcal{H}$ such that $Pr[err_D(h) \leq \epsilon] \geq 1 - \delta$

After going through the definition, there are few questions which can be asked, such as what if we don't have an algorithm that works for every $\epsilon$ and every $\delta$. For instance, what if we have an algorithm that only works for $\delta = \frac{1}{3}$. This algorithm guarantees a success probability of $\frac{2}{3}$. Technically this algorithm is not a PAC-learning algorithm because one cannot give any success probability as input and expect to achieve success with that probability. Can we use this algorithm which has a success probability of $\frac{2}{3}$ to get arbitrary success probability?. One possible way is to run the algorithm many times, and we know from Chernoff bounds that the algorithm will succeed with probability $\geq 1-\delta$. In particular, if it is run for $log(\frac{1}{\delta}$ times , it can be guaranteed that in one of the tries, it could output a good hypothesis. Lets look at another question, suppose there is an algorithm that doesn't achieve any value of $\epsilon$(error of the algorithm) i.e it cannot achieve any arbitrary error. For instance, say we have an algorithm that only gets $\epsilon = 0.49$. In binary classification, if one were to just randomly guess then the error achieved would be half. Now, this algorithm performs slightly better than random guessing with error rate 0.49. Can we use this algorithm to achieve arbitrary accuracy?.

This algorithm described above is known as Weak PAC-learning. For all distributions $D \sim x$, for all $\delta > 0$ the algorithm A uses finite data and produces a hypothesis $h \in \mathcal{H}$ such that $Pr[err_D(h) \leq \frac{1}{2} - \gamma] \geq 1 - \delta$, for some fixed $\gamma > 0$. This learning does better than random guessing and is a easy to design compared to some algorithm that does very well on entire instance space(X) which gets error as close to 0(harder problem to design). It is clear from previous argument, strong PAC learning implies weak PAC learning. But can we say weak learning implies strong learning?. In fact, we will see in the following section, it is true. We will show that by combining these weak learning in some specific way we can achieve a strong learner.

Lets see an example,where the hypothesis class($\mathcal{H}$) is a class of boolean disjunctions.We have n boolean variables($x_1, x_2, ..., x_n$) and the target function is some disjunction $f^* = (x_1 \vee x_2 \vee ... \vee x_k)$. And we are trying to learn this disjunction.



Probability Mass Function

Possible solution : Lets start by designing a weak learner for this problem. Take a look at the entire space and find the ones where the function is +1. We can say now some of the

variables in the disjunction are +1. And lets assume the probability mass in "+1" region is $p$. In this space atleast one of these variables will be 1 all the time, which means one these variables will be 1 on atleast $\frac{p}{n}$ fraction.

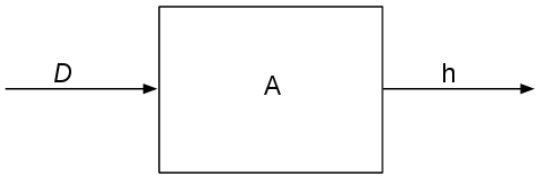



Figure 1: Illustration of a Weak Learner

Lets say $x_1$, is always on for $\frac{p}{n}$ times. if $x_1 = 1$, then predict +1, otherwise predict randomly.

$$Err(f) = \begin{cases} 0, & \text{over } \frac{p}{n} \text{ space} \\ \frac{1}{2}, & \text{remaining space} \end{cases}$$

$$Err(f) = \tfrac{1}{2} - \tfrac{p}{2n}.$$

As long as $p$ is not 0, this performs better than random guessing. If $p$ is 0, than all the output are -1, so we can predict -1. The method to use these weak learners iteratively in sort of a clever fashion to achieve a strong learner is known as **Boosting**. [1]

**Theorem :** Given $\mathcal{S} = \langle (x_1, y_1), ..., (x_m, y_m) \rangle$ drawn from the true distribution $D \sim X$, where $x_i \in X$, $y_i \in (-1, 1)$ and access to an algorithm A that can weak learn $\mathcal{H}$ for all $\mathcal{D}$ with $\frac{1}{2} - \gamma$ accuracy and with probability $\geq 1 - \delta$, then we can use A to get a strong learner.

*Proof*: The main idea behind boosting is to run the weak learning algorithm several times and combine the hypotheses from each run.This algorithm guarantees for all distribution $D$, it achieves $err_D(h) \leq \frac{1}{2} - \gamma$. So, there is atleast $\frac{1}{2} - \gamma$ of the entire space($\mathcal{X}$) where $h$ makes an error. We run the algorithm again in the region where $h$ was wrong and get hypothesis($h'$). Similarly, we continue to choose another distribution where previous hypothesis have failed and the new hypothesis will give better accuracy on the mistake region of previous hypothesis. We can consider this algorithm as a black box, and we will run it different times on multiple distributions [2] to finally obtain error $\epsilon$. Lets start by denoting accuracy of the hypothesis as $p$ which is $\frac{1}{2} - \gamma$. Say we have an algorithm which reduces error probability from $p$ to $p^2$ [3]. We can use this algorithm multiple times to reduce the error to $\epsilon$.

[1] A weak learner that works for a specific distribution cannot be boosted to get a strong learner for same distribution.But if a weak learner that works for every distribution, then it can be made a strong learner for every distribution.

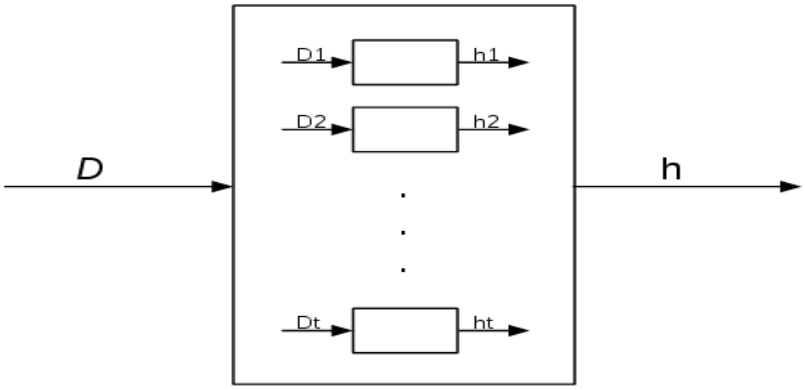



Figure 2: Illustration of Boosting Algorithm

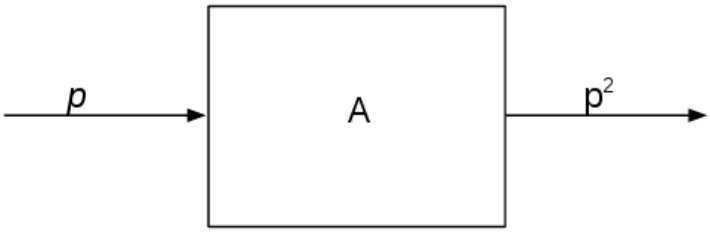



Figure 3: Illustration of a Weak PAC Learner

Lets look at the steps to achieve $p^2$ from $p$:
**Step 1:** Run the algorithm A on D(original distribution) to get $h_1$ and the $Err_D(h_1) = p$. The figure[4] shows the result of hypothesis on the instance space. We can call the region where the hypothesis predicts correctly to be A and B to be the one where it gives error. Alternatively, we can represent $B$ as $D|_{h_1 \neq f^*}$. As described earlier, we execute the algorithm on the region where previous hypothesis disagree with each other or all predict incorrectly. We use rejection sampling to sample points from this distribution by starting with the original distribution and finding points in the space where hypothesis gives error. Repeating this step for every iteration.
**Step 2:** Next we execute algorithm A on $D|_{h_1 \neq f^*}$ and output hypothesis $h_2$. The worst case for the algorithm is to predict $h_2 = -h_1$. The motto is to restrict the algorithm from providing hypothesis $-h_1$ (as we have not learned anything). So, the new chosen distribution has a half of old distribution where $h_1$ was successful and the other half from the distribution where $h_1$ failed.

$$D' = \frac{1}{2}D|_{h_1 = f} + \frac{1}{2}D|_{h_1 \neq f^*}$$

**Step 3:** We look at the region where both $h_1$ and $h_2$ disagree with each other. The ideal

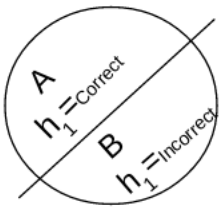



Figure 4: Correct and Incorrect region in Instance Space based on the putput by hypothesis($h_1$).

way is to get a few functions and take their majority vote. If the majority says +1, predict +1 and vice versa. The low confidence area is where both $h_1$ and $h_2$ disagree with each other. We run the algorithm again on $D'' = D|_{h_1 \neq h_2}$ and output hypothesis $h_3$. With these three hypothesis, we will use the majority vote to predict.

**Claim :** $Err(MAJ(h_1, h_2, h_3)) \leq 3p^2 - 2p^3$

We denote $p = \frac{1}{2} - \gamma$, also divide the instance space into four regions, depending on what $h_1$ and $h_2$ predict.

$$S_{CC} : h_1 = f^* and h_2 = f^*$$
$$S_{II} : h_1 \neq f^* and h_2 \neq f^*$$
$$S_{CI} : h_1 = f^* and h_2 \neq f^*$$
$$S_{IC} : h_1 \neq f^* and h_2 = f^*$$

The error of the majority function can be written in terms of the probability mass of the four regions,

$$Err(MAJ(h_1, h_2, h_3)) = S_{II} + p(S_{CI} + S_{IC})$$

Both $h_1$ and $h_2$ will make error on $S_{II}$ and where they disagree($S_{CI}$ and $S_{IC}$), $h_3$ will be referred. And $h_3$ will make error on p($S_{CI}$ + $S_{IC}$). Next, we will relate it with $h_2$. We know that $h_2$ was run on a distribution combining half the space was from $h_1$ where it made a mistake and other half where it was correct[6]. So if we multiple $\frac{1}{2}$ to every point in the region where $h_1$ makes a mistake the probability mass is $\frac{1}{2p}$, similarly for the other region we multiply the entire region by $\frac{1}{2(1-p)}$ the probability mass will be$\frac{1}{2(1-p)}$.

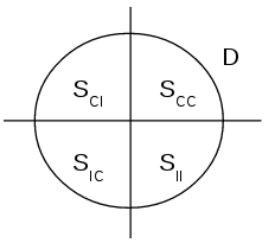



Figure 5: Instance Space divided into four regions based on prediction by hypothesis($h_1$ and $h_2$).

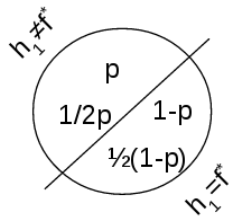



Figure 6: Illustration of hypothesis($h_1$) on the Instance Space

We can now write the regions in the distribution $D'$ in terms of $p$.

$$D'(S_{II}) = \frac{1}{2p} S_{II}$$
$$D'(S_{CI}) = \frac{1}{2(1-p)} S_{CI}$$
$$D'(S_{IC}) = \frac{1}{2p} S_{IC}$$

$$\begin{aligned} D'(S_{IC}) - D'(S_{CI}) &= \big(D'(S_{IC}) + D'(S_{II})\big) - \big(D'(S_{CI}) + D'(S_{II})\big) \\ &= \frac{1}{2} - p \end{aligned}$$

$$\begin{aligned} Err(MAJ(h_1, h_2, h_3)) &= 2pD'(S_{II}) + 2p(1-p)D'(S_{CI}) + 2p^2 D'(S_{IC}) \\ &= 2p(D'(S_{II}) + D'(S_{CI})) + 2p^2(D'(S_{IC}) - D'(S_{CI})) \\ &= 2p^2 + 2p^2(D'(S_{IC}) - D'(S_{CI})) \\ &= 2p^2 + 2p^2(\frac{1}{2} - p) \\ &= 3p^2 - 2p^3 \square \end{aligned}$$

**Resources**

- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.8630rep=rep1type=pdf. The original Boosting paper.
- http://www.cs.princeton.edu/courses/archive/spr07/cos424/papers/boosting-survey.pdf. An excellent survey on Boosting and Adaboost.
- http://rob.schapire.net/papers/explaining-adaboost.pdf. More on AdaBoost.