**Strong vs Weak PAC-Learning:**

So far in this class, we have shown that if we can PAC-learn a concept class $C$, then we can achieve a particular level of accuracy ($\epsilon$) at a particular level of confidence ($\delta$) by only requiring a polynomial amount of data with respect to the levels of accuracy and confidence. However, in real life, we do not have an infinite amount of data to achieve 100% accuracy with 100% confidence. In particular, a lot of the times, our algorithms can reliably achieve only a low level of accuracy with high confidence. Therefore, a natural question to ask is, if we manage to develop a weak-learner, i.e. an algorithm that can learn $C$ (i.e. reliably perform better than unbiased coin flips), but with a low level of accuracy, how can we automatically and reliably generate a strong learner, an algorithm that can achieve high accuracy with high confidence?

The algorithm that achieves this is called Boosting. To understand how this algorithm work, we have to distinguish between two types of PAC-learning:

**Definition:** A concept class $C$ is strongly PAC-learnable if $\exists$ algorithm $A$ s.t., $\forall c \in C$, $\forall D$, $\forall \epsilon, \forall \delta$, if $A$ is given $m = poly(\frac{1}{\delta}, \frac{1}{\epsilon})$ examples, then it outputs $h$ s.t.:

$$P(err_D(h) \leq \epsilon) \geq 1 - \delta$$

**Definition:** A concept class $C$ is weakly PAC-learnable if $\exists \gamma$ s.t., $\exists$ algorithm $A$ s.t., $\forall c \in C$, $\forall D$, $\forall \delta$, if $A$ is given $m = poly(\frac{1}{\delta})$ examples, then it outputs $h$ s.t.:

$$P(err_D(h) \leq 0.5 - \gamma) \geq 1 - \delta$$

The big difference between the two definitions are the terms $\gamma$ and $\epsilon$ that measure error. So the link we are trying to achieve is whether an algorithm that can achieve a particularly high level of accuracy reliably for all concepts classes and distributions $D$ can be "boosted" into an algorithm that can do the same for all levels of accuracy when given enough samples. Note that algorithmic run-time efficiency is also a consideration, but we will not be focusing on it in this discussion.
Boosting - The Theorem:

**Theorem:** A concept class $C$ is weakly PAC-learnable if and only if it is strongly PAC-learnable.

This theorem implies that learning is an all or nothing phenomenon. In other words, if you can find an algorithm that achieves a low level of accuracy in learning $C$, then there exists an algorithm that can do the same with a high level of accuracy.

The goal is to now prove the Theorem while working under the distribution-free model. In other words, we will show that, assuming the existence of a weak learning algorithm, we can develop a strong learning algorithm, without making any assumptions on $D$. The reverse direction trivially follows.

**Boosting - Proof Setup:**

Assume we are given random sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, where $\forall i, x_i \in X$, $y_i \in \{-1, +1\}$ and $(x_i, y_i) \sim D$. Assume we also know a weak-learner $A$ (i.e. an algorithm that can weakly PAC-learn the target class $C$, according to the definition above).

A good way of thinking about $A$ is as an algorithm, such that, $\forall D, \forall \delta$, when given $m = poly(1/\delta)$ examples from $D$, it computes a hypothesis $h$ s.t., given a fixed $\gamma$:

$$P(\text{err}_D(h) \leq 0.5 - \gamma) \geq 1 - \delta$$

What we need to do now is identify a boosting algorithm that, using $A$, finds a boosted hypothesis $H$ s.t. $\text{err}_D(H) \leq \epsilon$. The trick we will use is the fact $A$ generates a weak hypothesis $h$ for any distribution $D$, so we can create different distributions $D_j$ such that we generate multiple weak $h_j$ to ultimately combine them into a stronger hypothesis $H$.

**The AdaBoost Algorithm:**

The boosting algorithm we will consider and prove the Theorem for is AdaBoost. Here are the steps of AdaBoost:

0. Take in input sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. Define distribution $D_1$ over $S$ by initializing probability measure $P((x, y) = (x_i, y_i)) = D_1(i) = \frac{1}{m}, \forall i$. Pick the number of iterations $T$. Then, for $t = 1, \ldots, T$ repeat the following steps:

1. Run $A$ on $D_t$ to get weak hypothesis $h_t : X \rightarrow \{-1, 1\}$. We can achieve this because of the distribution-free assumption of weak PAC-learning.

2. Compute the generalization error on $S$: $\epsilon_t = \text{err}_{D_t}(h_t) = \sum_{1_{h_t(x_i) \neq y_i}} D_t(i)$. Let $\text{err}_{D_t}(h_t) = 0.5 - \gamma_t$. We call the term $\gamma_t$ edge, as it measures how far away the error is from the average error of an unbiased coin-hypothesis.

3. Compute $a_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$. 
4. Update probability distribution over \( S \):

\[
D_{t+1}(i) = \frac{D_t(i) \cdot e^{-a_th_t(x_i)y_i}}{Z_t},
\]

where \( Z_t \) is the normalization factor \( Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-a_th_t(x_i)y_i} \). Dividing by \( Z_t \) gives us back a probability distribution over \( S \).

Once the algorithm terminates, compute the combined hypothesis \( H(x) = \text{sign}(\sum_{t=1}^{T} a_th_t(x)) \).

The reasoning behind the choice of \( a_t \) becomes evident in the proof to follow. The intuition behind step 4 is that we want to get penalized more if we misclassify examples that we have misclassified in the past. Therefore, we weigh more heavily errors on ”hard” examples, while we weigh less errors on ”easy” examples, which makes the algorithm better at learning how to classify correctly ”hard” examples, which account for most errors.

Graphically, one can think of the boosting algorithm as a composition of two functions: \( f \) and \( g \). \( f \) takes in \( A, T \) and \( S \sim D \) and returns a set of weak hypotheses \( \{h_1, ..., h_T\} \), which are then fed into \( g \) to give back the combined hypothesis \( H \). Thus:

\[
\text{AdaBoost}(A, T, S) = g(f(A, T, S)) = g(\{h_1, ..., h_T\}) = H
\]

**AdaBoost - Training Error Bound:**

To complete the proof of the initial Theorem, we need to show that for the hypothesis \( H \) produced by AdaBoost, \( err_D(H) \leq \epsilon \). To do that, we will first prove the following auxiliary theorem.

**Theorem 2:** For AdaBoost, \( \hat{err}(H) \leq \prod_{t=1}^{T} (2\sqrt{\epsilon_t(1-\epsilon_t)}) \).

**Proof:**

Step 1: Consider the final distribution \( D_{T+1} \) defined on \( S \):

\[
\forall i, D_{T+1}(i) = D_1(i) \cdot \frac{e^{-a_1h_1(x_i)y_i}}{Z_1} \cdot \ldots \cdot \frac{e^{-a_T h_T(x_i)y_i}}{Z_T} = \frac{e^{-y_i \sum_{t=1}^{T} a_th_t(x_i)}}{m \prod_{t=1}^{T} Z_t} = \frac{e^{-y_iF(x_i)}}{m \prod_{t=1}^{T} Z_t}
\]

, where \( F(x_i) = \sum_{t=1}^{T} a_th_t(x_i) \). Thus, we have just shown that:

\[
\forall i : D_{T+1}(i) = \frac{e^{-y_iF(x_i)}}{m \prod_{t=1}^{T} Z_t}
\]

Step 2: Consider now \( \hat{err}_S(H) \), the training error of \( H \) on \( S \):

\[
\hat{err}_S(H) = \frac{1}{m} \sum_{i=1}^{m} 1_{H(x_i) \neq y_i} \leq \frac{1}{m} \sum_{i=1}^{m} e^{-y_iF(x_i)}
\]

The inequality follows from the fact \( 1_{H(x_i) \neq y_i} = 1_{y_iF(x_i) \leq 0} \leq e^{-y_iF(x_i)} \). It helps to
consider how each side of the inequality behaves when \( y_i F(x_i) \) is positive and negative.

Thus, by step 1, it follows that:

\[
e^\hat{err}_S(H) = \frac{1}{m} \sum_{i=1}^{m} (D_{T+1}(i) \cdot m \prod_{t=1}^{T} Z_t) = \prod_{t=1}^{T} Z_t
\]

Step 3: All we need to do now is compute \( Z_t \):

\[
Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-a_t h_t(x_i) y_i} = \sum_{i: y_i \neq h_t(x_i)} D_t(i) \cdot e^{a_t} + \sum_{i: y_i = h_t(x_i)} D_t(i) \cdot e^{-a_t} = e^{a_t} \epsilon_t + e^{-a_t} (1 - \epsilon_t)
\]

Now our goal is to minimize the right hand side of the equality. In fact, this minimization yields \( a_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \), which is the \( a_t \) term used in the AdaBoost algorithm. This yields:

\[
Z_t = 2 \sqrt{\epsilon_t (1 - \epsilon_t)}
\]

The results of steps 2 and 3 complete the proof. ■

**AdaBoost - Generalization Bound:**

Now that we have bounded the training error of AdaBoost, we can use what we have already proven in class these past weeks to prove a bound for the generalization error to complete the proof of the initial Theorem. It is payoff time!

Since all of the generalization bounds we have proven require some measure of the complexity of the hypothesis space, we need to measure the complexity of \( H(x) \). To do so, we will rewrite \( H(x) \) as follows:

\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} a_t h_t(x) \right) = g(h_1(x), ..., h_T(x))
\]

Note that \( g \) is a linear threshold function with dimension \( T \). Let \( G \) be the class of all linear threshold functions in \( T \) dimensions and let \( F \) be the hypothesis space of all \( h_i, i = 1, ..., T \), since all hypothesis generated from algorithm A come from the same hypothesis space. Let \( K \) be the hypothesis space of \( H \). Then, from the theorem proven in HW2, Problem 2, it follows:

\[
\Pi_K(m) \leq \Pi_G(m) \cdot (\Pi_F(m))^T
\]

Note that \( VCdim(G) = T \) and let \( VCdim(F) = d \). Thus, using the VC-dimension bound for the growth function we proved in a previous lecture, we get:

\[
\Pi_K(m) \leq \left( \frac{em}{T} \right)^T \cdot \left( \frac{em}{d} \right)^d
\]

This in turn yields:

\[
err_D(H) \leq \hat{err}_D(H) + O \left( \sqrt{\frac{ln(\Pi_K(m)) + ln(1/\delta)}{m}} \right) \leq \hat{err}_D(H) + \hat{O} \left( \sqrt{\frac{Td + ln(1/\delta)}{m}} \right)
\]
, where \( \hat{O} \) stands for a soft O-bound that ignores log terms multiplied by polynomial terms. This completes the proof of the initial Theorem. ■

**Conclusions:**

The generalization bound we just proved implies that, for not very large \( T \), the generalization error decreases. However, once \( T \) becomes sufficiently big, the O-term grows faster than the training error decreases, indicating that the generalization error increases. This is consistent with Ocamm’s razor that states that simpler hypotheses attain lower generalization error. This behavior is exhibited graphically in the figure below.

What is very interesting though is that this is not always the case in practice. In fact, in many application of Boosting, we see the generalization error decreasing even for very large \( T \). Examples of this odd behavior are included in the slides accompanying these notes.

So is Ocamm’s razor flawed? Or are all these applications of Boosting misguided? Answers about these questions and the paradox seemingly contradicting Ocamm’s razor are given in the next lecture.