1 Relating the Mistake Bound and PAC models

We can relate the mistake bound and PAC models as follows:

**Theorem 1** If algorithm $A$ learns a concept class $C$ in the mistake bound model, then $A$ also learns $C$ in the probably approximately correct model.

*Proof:* In proving this we would like to assume that an algorithm is conservative (or lazy), meaning that the algorithm only changes its state (its state is its current hypothesis) when it makes a mistake. We can insure that it does this by taking a snapshot of its state before giving it an example; if the algorithm predicts correctly, we will restore the state it had before the prediction. In this case the algorithm will behave as it would were it only given examples on which it makes mistakes, so the number of mistakes is still bounded. Thus, given $A$ learning $C$ in the mistake bound model we can construct $A'$ also learning $C$ in the mistake bound model but only changing its hypothesis when it makes a mistake.

Transforming the algorithm to a lazy one is not strictly necessary, but it makes the proof simpler.

From $A'$ we can construct the following algorithm $A_{PAC}$ (where $M$ is the mistake bound, which is bounded by a polynomial in $n$ and the size of $c$):

Run $A'$ on the data seen, halting if any hypothesis $h$ survives for more than $\frac{1}{\epsilon} \ln(M/\delta)$ subsequent examples. Return $h$ as the hypothesis of the algorithm.

Note that since the number of mistakes made is bounded by $M$, the algorithm will terminate within $\frac{M}{\epsilon} \ln(M/\delta)$ examples.

The probability that the algorithm accepts a hypothesis with error greater than $\epsilon$ is at most

$$M(1 - \epsilon)^{\frac{1}{2} \ln(M/\delta)} < Me^{-\frac{1}{2} \ln(M/\delta)} = M \frac{\delta}{M} = \delta$$

So the chance that an algorithm accepts a good hypothesis (of error at most $\epsilon$) is at least $1 - \delta$; hence $A_{PAC}$ learns in the PAC model.  ■
2 The Halving Algorithm

If we do not care about computation time or the hypothesis space $H$, and if $C$ is a finite set, then we can achieve a mistake bound of $\log |C|$. This is achieved by the following method called the “halving algorithm”:

1. Initialize $V$ to $C$. ($V$ is called the version space.)
2. Given example $x$, predict according to the majority of the concepts in $V$.
3. Remove from $V$ all the concepts in $C$ that predicted wrongly.
4. Return to step 2.

Since with each mistake at least half of the version space $V$ is removed, the number of mistakes is bounded by $\log |C|$. Clearly this is a very nice general algorithm to learn an arbitrary concept space in both the mistake bound model and the PAC model. Unfortunately, though, storing $V$ and predicting according to the majority of the concepts in $V$ is likely to be hard.

3 The Winnow Algorithm

We now turn to an algorithm called the Winnow Algorithm developed by Littlestone that performs especially well when many of the features given to the learner turn out to be irrelevant. Like the Perceptron Training Procedure discussed in the previous lectures, Winnow uses linear threshold functions as hypotheses and performs incremental updates to its current hypothesis. Unlike the Perceptron Training Procedure, Winnow uses multiplicative rather than additive updates. Winnow was developed with the goal of providing a significant Mistake Bound improvement when $r \ll n$.

We will analyze the Winnow algorithm for learning the class of $C$ of \{monotone disjunctions of $r$ variables\}. Note that Winnow or generalizations of Winnow can handle other specific concept classes (e.g. non-monotone disjunctions, majority functions, linear spearators), but the analysis is simplest in the case of monotone disjunctions.

3.1 The Algorithm

Both Winnow and Perceptron Algorithms use the same classification scheme:

- $\mathbf{w} \cdot \mathbf{x} \geq \theta$ ⇒ positive classification
- $\mathbf{w} \cdot \mathbf{x} < \theta$ ⇒ negative classification
For convenience, we assume that θ = n and we initialize w to the “all ones” vector, i.e., 
\[ w_i = 1 \] for all \( i \).

The Winnow Algorithm differs from the Perceptron Algorithm in its update scheme. When 
misclassifying a positive training example \( x \) (i.e. the prediction was negative because \( w \cdot x \) 
was too small):

\[ \forall x_i = 1 : w_i \leftarrow 2w_i. \]

When misclassifying a negative training example \( x \) (i.e. prediction was positive because \( w \cdot x \) 
was too large):

\[ \forall x_i = 1 : w_i \leftarrow w_i/2. \]

Notice that because we are updating multiplicatively, all weights remain positive; so, to 
handle a non-monotone target concept we would need to transform the input space to have 
\( \bar{x}_i \) be a new variable. Intuitively, Winnow does more with the training information than the 
traditional list-and-cross-off scheme, because now in order to predict positive there must be 
“enough” evidence; thus we make more progress when we make a mistake. As we will see 
later, this is why Winnow guarantees a better performance bound for learning disjunctions 
when \( r \) is small.

### 3.2 The Mistake Bound Analysis

We can show the following guarantee:

**Theorem 2** The Winnow Algorithm learns the class of monotone disjunctions in the Mis-
take Bound model, making at most \( 2 + \frac{3}{2}r(1 + \log n) \) mistakes when the target concept is an 
OR of \( r \) variables.

**Proof:** Let \( X_r = \{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\} \) be the \( r \) relevant variables in our target concept. Let 
\( W_r = \{w_{i_1}, w_{i_2}, \ldots, w_{i_r}\} \) be the weights of the relevant variables. Let \( w(t) \) denote the value 
of weight \( w \) at time \( t \) and let \( TW(t) \) be the Total Weight of the LTF (including both relevant 
and irrelevant variables) at time \( t \).

We will first bound the number of mistakes that will be made on positive examples. Note 
first that any mistake made on a positive example must double at least one of the weights in 
the target function (the relevant weights). So if at time \( t \) we misclassify a positive example 
we have:

\[ \exists w \in W_r \text{ such that } w(t + 1) = 2w(t) \]  \( 1 \)

\( ^{1} \)Note that Perceptron used a threshold of 0 but here we use a threshold of \( n \).
Moreover a mistake made on a negative example will not halve any of the relevant weights, by definition of a disjunction; so for all times $t$, we have:

$$\forall w \in W_r, w(t + 1) \geq w(t)$$  \hspace{1cm} (2)

Moreover, each of these weights can be doubled at most $1 + \log(n)$ times, since only weights that are less than $n$ can ever be doubled. Combining this together with (1) and (2), we get that Winnow makes at most $M_+ \leq r(1 + \log(n))$ mistakes on positive examples.

We now bound the number of mistakes made on negative examples. Note first that a mistake made on a positive example increases the total weight by at most $n$. To see this assume that we made mistake on the positive example $x$ at time $t$. We must have:

$$w_1(t)x_1 + \ldots + w_n(t)x_n < n$$

Since

$$TW(t + 1) = TW(t) + (w_1(t)x_1 + \ldots + w_n(t)x_n),$$

we get

$$TW(t + 1) \leq TW(t) + n. \hspace{1cm} (3)$$

Similarly, we can show that each mistake made on a negative example decreases the total weight by at least $n/2$. To see this assume that we made mistake on the negative example $x$ at time $t$. We must have:

$$w_1(t)x_1 + \ldots + w_n(t)x_n \geq n.$$ 

Since

$$TW(t + 1) = TW(t) - (w_1(t)x_1 + \ldots + w_n(t)x_n)/2,$$

we get

$$TW(t + 1) \leq TW(t) - n/2. \hspace{1cm} (4)$$

Finally, the total weight never drops below zero, i.e., at all times:

$$TW(t) > 0 \hspace{1cm} (5)$$

Combining equations (4), (3), and (5) we get:

$$0 < TW(t) \leq TW(0) + nM_+ - (n/2)M_- \hspace{1cm} (6)$$

The total weight summed over all the variables is initially $n$ since $w(0) = 1$. Solving (6) we get $M_- \leq 2 + 2M_+$. Combining both the negative and positive mistakes, we get that Winnow obtains makes at most $2 + 3r(1 + \log n)$ mistakes when the target concept is an OR of $r$ variables.

**Interesting Open Question:** Is there a computationally efficient algorithm for learning decision lists in the mistake bound model with a mistake bound $\text{poly}(L, \log n)$, where $L$ is length of target decision list? Note that the halving algorithm achieves this bound, but it is not a computationally efficient algorithm.