Mistake bound learning additional notes

The Idea: "Each Mistake Eliminates Half the Wrong Rules"

Imagine you're trying to guess a secret number between 1 and 16. At the start, you have 16 possible numbers (rules) that could be the secret number. Your strategy is to always guess the middle number of the remaining possibilities.

Step-by-Step Example:

1. **Initial Possibilities:**
   The secret number could be any number from 1 to 16.
   Total possibilities = 16.
2. **First Guess:**
   You guess 8 (the middle of 1 and 16).
   - **Outcome:** Suppose you are told your guess is too high.
   - **Elimination:** This means the secret number must be less than 8.
   - **New Possibilities:** Now the possible numbers are 1 through 7.
   - **Result:** You have eliminated the 8 numbers (8 through 16) that are too high. In this case, more than half were eliminated, but the idea is similar: a mistake gives you a lot of information and cuts down your choices significantly.

3. **Second Guess:**
   Now, with numbers 1 to 7 remaining, you guess the middle number—let's say 4.
   - **Outcome:** Suppose you're told 4 is too low.
   - **Elimination:** This tells you the secret number must be greater than 4.
   - **New Possibilities:** The possible numbers are now 5, 6, or 7.
   - **Result:** You've again removed a large portion of the remaining possibilities.
4. **Continuing This Process:**
   Each mistake (being told "too high" or "too low") helps you eliminate roughly half of the remaining numbers (rules).

How This Relates to Mistake-Bounded Learning:

- **In Learning:**
  Instead of numbers, think of having many possible "rules" or hypotheses about how to classify data (for example, what threshold to use or what feature determines spam emails).
- **With Each Mistake:**
  When the algorithm makes a mistake on an example, it can eliminate all the hypotheses that would have made that same mistake. If the hypotheses are well-structured (like our ordered numbers), then typically at least half of them will be wrong on that example.
- **Mathematically:**
  If you start with $|H|$ possible hypotheses, and every mistake cuts the number in half, after M mistakes you have at most:

$$\frac{|H|}{2^M}$$

When only one hypothesis remains (the correct rule), you have:

$$\frac{|H|}{2^M} = 1 \quad \Longrightarrow \quad M = \log_2 |H|$$

This is the basis for the mistake bound:

$$M(A, c) \leq \log_2 |H|$$

In Summary:

- **"Each mistake eliminates half the wrong rules"** means that every time the algorithm makes a mistake, it uses that mistake to rule out about half of the current candidate rules (or hypotheses).
- **Practical Impact:** This efficient elimination process ensures that even if there are many possible rules at the start, the algorithm only needs a logarithmic number of mistakes (in relation to the total number of rules) to narrow down to the correct one.

Example: Threshold-Based Spam Classification

Setup:

Imagine we have an AI that classifies emails as **spam** or **not spam** based on the number of "suspicious" words (like "free," "win," "offer") in the email. We assume there is a threshold k such that:

$$\text{Email is spam} \iff (\text{number of suspicious words}) \geq k$$

However, the AI doesn't know the correct threshold k at first. Instead, it considers a set of candidate hypotheses:

$$H = \{h_1, h_2, \ldots, h_{16}\}$$

where each h_k represents the rule: "Email is spam if and only if it contains at least k suspicious words."
So, there are 16 possible rules.

Learning Process Using Mistake-Bounded Learning

1. **Initial Hypothesis Space:**

   The AI starts with all 16 hypotheses. We denote the number of hypotheses by $|H| = 16$.

2. **Making a Prediction:**

   The AI uses a strategy (like the halving algorithm) to decide on a prediction. For instance, it might use the median rule among the remaining candidates.

3. **Receiving an Email and Checking the Outcome:**

   - **Email Example:** An email arrives with 7 suspicious words.
   - **True Rule (unknown to the AI):** Suppose the correct threshold is 7, meaning emails with 7 or more suspicious words are spam.
   - **AI's Initial Guess:** The AI might initially choose h_8 (i.e., it predicts an email is spam only if it has 8 or more suspicious words).
     - For this email, since $7 < 8$, h_8 would classify it as **not spam**.
   - **Mistake:** Since the true rule says that an email with 7 suspicious words is spam, the AI's prediction is wrong.

4. **Eliminating Wrong Hypotheses:**

   The mistake provides information:

   - The AI now knows that any hypothesis which would have predicted **not spam** for an email with 7 suspicious words is incorrect.
   - This means it can eliminate all h_k with $k > 7$ because those rules would classify an email with 7 suspicious words as not spam.
   - If the candidate space initially had 16 hypotheses, this error might cut the number of remaining candidates roughly in half.

5. **Repeating the Process:**

   The AI now uses the reduced hypothesis space and makes another prediction on the next email. Each mistake eliminates many wrong hypotheses. In the halving algorithm, every mistake eliminates at least half of the remaining candidates.

   Therefore, after at most:

$$M(A, c) \leq \log_2 |H| = \log_2 16 = 4 \quad \text{mistakes}$$

   the AI is left with one hypothesis, which is the correct rule for classifying emails as spam or not spam.

1. What Is a Monotone Disjunction?

A **monotone disjunction** is a Boolean function that is an OR of a subset of Boolean variables, with **no negations**. In other words, it's a function of the form:

$$f(x) = x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k}$$

where each x_{i_j} is a Boolean variable (taking values 0 or 1).

Key Points:

- **Monotonicity:**
  It's called *monotone* because if you change any input from 0 to 1 (i.e., "turn a variable on"), the output of the function can only stay the same or switch from 0 to 1—it will never flip from 1 to 0.
- **Example:**
  Consider three Boolean variables x_1, x_2, x_3. A monotone disjunction might be:

  $$f(x_1, x_2, x_3) = x_1 \vee x_3.$$

  This function outputs 1 (true) if either x_1 is 1 or x_3 is 1 (or both), and outputs 0 only if both x_1 and x_3 are 0.

2. Why Is It Applied in Mistake-Bounded Learning?

Mistake-bounded learning is a framework where a learning algorithm is allowed to make mistakes while learning, but the total number of mistakes is bounded by some function of the concept class (and not the number of examples).

Learning Monotone Disjunctions:

- **Hypothesis Space:**
  In learning monotone disjunctions, the algorithm's hypothesis space might be all possible disjunctions over n Boolean variables. This space is structured and well-behaved.
- **A Simple Algorithm:**
  A common algorithm for learning a monotone disjunction starts with the **most general hypothesis**—one that includes all variables. For example, if you have n features, the algorithm might initially assume:

  $$h(x) = x_1 \vee x_2 \vee \cdots \vee x_n.$$

How It Learns from Mistakes:
Suppose the true function is $f(x) = x_1 \vee x_3$ (i.e., only x_1 and x_3 are truly relevant). Here's what happens:

1. **Receiving an Example:**
   The algorithm predicts using $h(x)$.
2. **Making a Mistake on a Negative Example:**
   If a negative example (where $f(x) = 0$) is misclassified as positive by $h(x)$, it means that at least one variable in the current hypothesis was "on" even though the true concept says the output should be 0.
3. **Eliminating Irrelevant Variables:**
   The algorithm then eliminates the variables that were active (set to 1) in that negative example. Each variable can be removed only once. For instance, if an example shows $x_2 = 1$ but the label is 0, then x_2 must not be part of the true disjunction, so the algorithm removes x_2 from its hypothesis.

- **Mistake Bound:**
  Since each mistake can remove at least one variable from the hypothesis and there are at most n variables, the total number of mistakes the algorithm makes is bounded by n.

Why Use Monotone Disjunctions?

- **Simplicity:**
  Their simple structure makes them a great candidate for illustrating mistake-bounded learning principles.
- **Bounded Mistakes:**
  The elimination process (removing one variable per mistake) guarantees that the learning algorithm makes only a finite and predictable number of mistakes (at most one per variable).
- **Efficient Learning:**
  They show how a learner can converge to the correct hypothesis quickly, which is a central idea in mistake-bounded learning.

Learning Process Using Mistake-Bounded Learning

1. **Initial Guess:**
   The learning algorithm might use a strategy like the halving algorithm. It starts by choosing the median threshold from its candidate set.
   For 16 possibilities, the median is around $T = 8$.
2. **Making a Prediction:**
   Suppose an email arrives with **10 suspicious words**.
   - **Using h_8:** Since 10 is greater than or equal to 8, the algorithm predicts **spam**.
   - **True Label:** However, because the true threshold is 12, an email with 10 suspicious words should be **not spam**.
3. **Receiving Feedback and Making a Mistake:**
   The algorithm realizes its prediction was wrong (a mistake).
4. **Eliminating Candidates:**
   The mistake tells the algorithm that any candidate threshold that would have classified an email with 10 suspicious words as spam is incorrect.
   - This means all hypotheses h_k with $k \leq 10$ are ruled out.
   - The candidate set shrinks from 16 possibilities to those with thresholds $\{11, 12, 13, \ldots, 16\}$.

5. **Mistake Bound $B(C)$:**
   In the halving algorithm, every mistake eliminates about half of the remaining candidates.
   - If we start with 16 possibilities, after one mistake we might have around 8 left.
   - After a second mistake, about 4 remain, and so on.
   - In the worst-case scenario, the maximum number of mistakes made is:
   $$B(C) = \log_2(16) = 4.$$

   This tells us that no matter how many emails arrive, the algorithm will make at most 4 mistakes before it converges to the correct threshold (in our case, 12).

Why $B(C)$ Is Chosen This Way

- **Based on Hypothesis Space Size:**
  The mistake bound $B(C)$ is tied to the number of candidate hypotheses—in this case, 16 thresholds. The halving algorithm guarantees that with each mistake, roughly half the candidates are eliminated, which is why $B(C) = \log_2 |H|$.
- **Guaranteeing Convergence:**
  The bound is chosen to ensure that even if the algorithm processes an unlimited stream of emails, it will only make a finite number of mistakes (here, at most 4) before it learns the correct rule.
- **Efficiency:**
  By eliminating large portions of the candidate set with each mistake, the algorithm quickly zeroes in on the correct threshold, making learning efficient and predictable.