CS 2416 - Machine Learning Theory

Lecture #2: 22 September 2005

Lecturer: Toniann Pitassi

Scribe Notes by: Toniann Pitassi

1 The Mistake Bound Model

In the mistake bound model, learning is online. At each round the learner gets an unlabelled example and must predict the classification of this example. Then the learner is told the correct answer. The goal is to bound the total number of mistakes made. More formally we have the following definition.

Definition Let C be a concept class. Assume for now that C is boolean. An algorithm A learns C in the mistake bound model if for every sequence of examples and for every concept c in C, the total number of mistakes made by A is bounded by a polynomial in n and s, where n is the size of each example and s is the description length of c.

Definition C is efficiently learnable in the mistake bound model if there is an A as described above and furthermore the running time of A per stage is also polynomial in n and s.

Example. Consider learning a conjunction of boolean variables, such as $x_1 \wedge \neg x_3 \wedge x_7$ in the mistake bound model. The learning algorithm is as follows. Initially the hypothesis consists of the conjunction of all $2n$ literals: $x_1 \neg x_1 x_2 \neg x_2 \ldots x_n \neg x_n$. Whenever we make a mistake on a positive example, remove from h all literals that are not satisfied by the example. If we ever make a mistake on a negative example, halt and output "no consistent conjunction." The total number of mistakes made can never be more than $n + 1$ since exactly n literals are removed from h after the first example, and for each subsequent mistake, at least 1 literal is removed. It is left to show that the algorithm is correct. The reasoning is very similar to our earlier reasoning for learning conjunctions in the consistency model; we only removed literals from h when absolutely necessary, so at any point in time, the current hypothesis is the largest possible conjunction consistent with all positive examples so far. Thus if any conjunction is consistent with both the positive and the negative examples so far, then so is our hypothesis.

1.1 The halving algorithm

Assume for now that the concept class, C is finite. If we ignore the efficiency issues then there is a natural algorithm for learning any concept class C. Let $E = e_1, \cdots, e_q$ be a sequence of examples. We will describe a tree T associated with C, E, and the learner's decisions along the way. The root of T is labelled by C, the set of all concepts. The first example e_1 partitions C into two classes: C_1 are those concepts in C that are positive on e_1 and C_0 are those concepts in C that are negative on e_1. The left child of the root is labelled by C_0 and the right child by C_1. If the correct label for e_1 is "positive" then the learner can ignore the left child, and continue on the right side, now trying to output some concept in C_1 so as to be consistent with future examples. Similarly if the correct label for e_1 is "negative" then the learner can ignore the right child, and go left, not trying to output some concept in C_0 consistent with future examples. Note that whenever the set of concepts associated with the current node contains only one concept, then the learner can output that concept and never make any further mistakes on any valid sequence of labelled examples.

The halving algorithm is as follows. Starting with e_1, the learner predicts "positive" if and only if $|C_1| \geq |C_0|$. More generally, at stage i, if C is the current class of concepts associated with stage/node i, then the learner predicts "positive" if and only if $|C_1| \geq |C_0|$, where C_1 are the concepts in C that are positive on e_i and C_0 are the concepts in C that are negative on e_i.

Now consider a particular concept $c \in C$ and a particular set of labelled examples E, consistent with c. We want give an upper bound on the number of mistakes that the halving algorithm makes on c under E.

Let i_1 be the first node/stage where the learner makes a mistake, and let D be the set of concepts associated with i. Suppose the learner predicted positive when the answer was negative. Then we know that $|D_1| \geq |D_0|$, and thus after stage i_1, the size of the current set of possible concepts is at most half the size of D. Thus each time the halving algorithm makes a mistake, we are guaranteed that the number of concepts still under consideration has gone down by at least half. Thus the total number of mistakes made on any concept c on any set of examples E consistent with c is at most $\log |C|$.

1.2 An optimal algorithm

The upper bound on the number of mistakes given above for the halving algorithm is not always tight. First, sometimes halving does better than $\log |C|$. For example, consider $C =< 0001, 0010, 0100, 1000 >$. Then the halving algorithm makes only one mistake rather than 2.

Moreover, sometimes a different strategy can beat the halving strategy. To understand this situation, consider two extreme situations. Suppose that E is a set of examples such that the tree T induced by E and C is completely balanced. (Note that any example that leads to a trivial partition of C can essentially be ignored since the learner never makes a mistake on this example and therefore when talking about a balanced versus unbalanced tree, we are only interested in the degree 2 nodes of T.) Then the halving algorithm will give mistake bound $\log |C|$ on every concept $c \in C$. On the other hand, suppose that E' is another set of examples such that the tree T induced by E' and C is completely unbalanced. In this case, the halving algorithm will never make more than one mistake. Thus the more unbalanced the tree T is, the less mistakes the halving algorithm makes, and the more balanced the tree is, the more mistakes the algorithm makes, never exceeding $\log |C|$ many mistakes.

Now consider the following example. The concepts in C consist of ten boolean functions, over

$n = 3$ underlying variables. They are as follows:

00000000
00100000
01000000
01100000
11110000
11101000
11100100
11100010
11100001
11100000

Consider concept $c = 000000000$, and the example ordering e_1, e_2, e_3, where $e_1 = 000$, $e_2 = 001$ and $e_3 = 010$. The halving algorithm would make three mistakes on this concept under this sequence of examples. To see this, note that e_1 partitions C into C_0 and C_1, where C_0 has size 4, and C_1 has size 6. After making a mistake here, we would then partition C_0 into C_{00} and C_{01} upon seeing e_2, each of size 2. Then we would make a second mistake, and finally C_{00} would be partitioned into C_{000} and C_{001} upon seeing e_3 and then we would make our final, third mistake.

However, a better strategy in this case would be to realize after seeing e_1 that although C_1 is larger, any ordering of the remaining variables leads to highly imbalanced partitioning of C_1. If we change the strategy to predict "negative" on e_1 and then make the rest of the predictions as the halving algorithm did, then we can achieve only 2 mistakes on this sequence of examples. Further, this new strategy will always achieve at most 2 mistakes!

Thus an optimal algorithm thus has to take into consideration the worst-case scenario that is actually possible for a given set of concepts C. We can think of learning in the MB model as a 2-person game (ignoring computational efficiency). The game proceeds as follows.

- First the opponent selects some example x which splits C into $C_0(x)$ and $C_1(x)$.
- The algorithm/learner then gets to pick one of $C_0(x)$ or $C_1(x)$ to throw away.(By predicting positive and making a mistake it throws out $C_1(x)$; by predicting positive and not making a mistake it throws out $C_0(x)$. Similarly, by predicting negative and making a mistake it throws out $C_0(x)$, and by predicting negative and not making a mistake it throws out $C_1(x)$.
- If only one concept left then we're done, and otherwise, go back to step 1.

For each concept class C, the optimal mistake bound for C, $OPT(C)$, can be computed inductively. Clearly for C of size 1 the mistake bound $OPT(C)$ is 0. For larger sets C, $OPT(C) = 1 + \max_x(\min(OPT(C_0(x)), OPT(C_1(x))))$.

Calculating this value on the above example, we obtain a mistake bound of 2, whereas the halving algorithm gives a mistake bound of 3.

2 Predicting from expert advice and the Weighted Majority Algorithm

We can think of the above problem more abstractly as follows. There are n experts (these correspond to each of the $n = |C|$ concepts) and m examples. We want an algorithm that will come up with a prediction that does as well as the best expert. There are m trials, one for each of the m examples. Each trial is a sequence of events where the algorithm: (1) receives the answers from each of the n experts; (2) makes its own prediction ; and (3) gets the correct answer. The goal is to construct an algorithm that minimizes the total number of mistakes made, as a function of the number of mistakes made by the best expert, and the number of examples.

If we know in advance that one of the experts in the collection predicts perfectly on each of the m examples then the halving algorithm that we described previously (we well as the optimal algorithm) work. However, how well can we do if we are not guaranteed that one of the experts is always correct? We can modify the halving algorithm as follows.

There are n weights, $w_1, \cdots, w_n$ one corresponding to each of the n experts. Initially all weights are set to 1. We can assume without loss of generality that all examples are distinct, since if we ever see the same example a second time, we will always output the correct answer. Given a set of predictions $x_1, \cdots, x_n$ by the experts for the current example, output the prediction with the highest total weight. That is, output 1 iff the sum of the weights of the experts predicting 1 is greater than the sum of the weights of the experts predicting 0. When the correct answer is received, update each weight as follows. For a given weight w_i, if this expert made a mistake then the new weight is $w_i/2$; otherwise leave the weight unchanged.

Theorem 1 *The number of mistakes made by the weighted majority algorithm described above is never more than* $2.4(e_{\min} + \log n)$ *where* $e_{\min}$ *is the number of mistakes made by the best expert so far.*

Proof When the algorithm makes a mistake, this means that at least half of the total weight of experts predicted incorrectly and therefore the total weight is reduced from W to $3/4W$. Therefore if the algorithm makes M mistakes, the weight W_{inf} at the end is at most $n(3/4)^M$. On the other hand, if the best expert has made $e_{\min}$ mistakes, then its weight at the end is $(1/2)^{e_{\min}}$. Thus we have

$$(\frac{1}{2})^{e_{\min}} \leq W_{\text{inf}} \leq n(\frac{3}{4})^M.$$

Which implies

$$(\frac{4}{3})^M \leq n2^{e_{\min}}$$

and thus $M \leq 2.4(\log n + e_{\min})$, as desired.

In class people asked several interesting questions. First, someone asked how this algorithm compares to the halving algorithm. In particular, if in the case where e_min is zero, are there cases where the halving algorithm makes less mistakes than the weighted majority algorithm? The answer is yes! It is not hard to come up with an example–here is one. Say that there are 8 experts, $E_1, \ldots E_8$, and the correct expert is E_8 who answers 1 on all 6 examples $e_1, e_2, e_3, e_4, e_5, e_6$. Assume that $E_1, \ldots, E_6$ answer 0 on each of the first 3 examples, and their answers on the remaining 3 examples are 001, 010, 011, 100, 101, 110. Assume that E_7 answers 1 on e_1, and 0 on the remaining 5 examples. Now consider the halving algorithm versus the weighted majority algorithm on the sequence of examples $e_1, \ldots, e_6$. The halving algorithm will make 2 mistakes, whereas the weighted majority algorithm will make 3 mistakes. The point is that it takes time for the weighted majority algorithm to "catch up" to the halving algorithm, since the weights are adjusted locally.

Someone else asked whether the algorithm is tight. That is, can we obtain an algorithm that is guaranteed to get less than $2.4(\log n + e_{min})$ mistakes? In partial answer to this question, there is a fancier randomized version of this algorithm that gets closer to $(e_{min} + \log n)$. It is called RWM for randomized weighted majority algorithm. The idea is as follows. In the randomized version of this algorithm, rather than taking a weighted majority vote, we instead use the weights as probabilities. For example, if 70 percent of the weighted experts say 0, and 30 percent of the weighted experts say 1, then we choose 0 with probability .7. Now the expected number of mistakes is $\ln n$ rather than $\log n$.

3 The Winnow Algorithm

In this section we describe Littlestone's Winnow algorithm. This is an algorithm for learning when there are many irrelevant variables, in the MB model.

Say we want to learn the class of monotone disjunctions. We saw that the simple list-and-cross-off method achieves this in mistake bound model with only $O(n)$ mistakes. But what if most of the variables are irrelevant–that is suppose that the target concept is an OR of only r variables. Can we do better than n mistakes? The Winnow algorithm solves this problem with $O(r \log n)$ mistakes.

The algorithm is as follows. Associated with each variable is a weight, $w_1, \cdots, w_n$. Initially all weights are set to 1. Given an example, α, output 1 if $\sum w_i \alpha_i \geq n$ and otherwise output 0. Now modify the weights as follows. If the algorithm predicts 0 on a positive example, then for each $\alpha_i = 1$, double the value of w_i. (And leave the other weights unchanged). If the algorithm predicts 1 on a negative example, then it must have zeros corresponding to all the variables in the target disjunction, so for each $\alpha_i = 1$, set w_i to 0. (Another version[1] of the Winnow algorithm halves the w_i in the negative example, which is more tolerant of cases where a consistent target disjunction may not exist).

Theorem 2 *The Winnow Algorithm learns the class of monotone disjunctions in the mistake bound model, making at most $1 + 2r(\log n)$ mistakes when the target concept is an OR of r variables. (The corresponding bound when the w_i are halved is $2 + 3r(\log n)$).*

Proof We will first bound the number of mistakes on the positive examples. Any mistake made on a positive example must double at least one of the weights in the target function (one of the relevant weights), and a mistake made on a negative example will not change any of these weights, by definition of a disjunction. Further, each of these weights can be doubled at most $\log n$ times since only weights that are less than n can ever be doubled. Therefore, Winnow makes at most $r(\log n)$ mistakes on positive examples.

Now we bound the number of mistakes on negative examples. The total weight summed over all variables is initially n. Each mistake made on a positive example increases the total weight by at most n (since before doubling we must have had $w_1\alpha_1 + \cdots + w_n\alpha_n < n$, and only the w_i corresponding to the non-zero α_i contributed to this sum and are doubled). On the other hand, each mistake made on a negative example decreases the total weight by at least n, since before eliminating the w_i that corresponded to non-zero α_i we must have had $w_1x_1 + \ldots + w_nx_n \geq n$. The total weight never drops below zero, so:

$$\begin{aligned} 0 &\leq n + (\text{errors on positive examples})n - (\text{errors on negative examples})n \\ &= n(1 + (\text{errors on positive examples}) - (\text{errors on negative examples})). \end{aligned}$$

[1] Done in class.

Therefore the number of mistakes made on negative examples is at most the number of mistakes made on positive examples plus 1, or $1 + r(\log n)$. Adding these two quantities together gives the theorem.