X The Mistake-Bound Model for On-line Learning

1. Introduction
2. Some Models of Learning
3. The Mistake-Bound Model
4. Mistake Bound for the Monomial Learning Algorithm (FIND-S)
5. Mistake Bound for the Halving Algorithm
6. Optimal Mistake Bounds
7. Mistake Bound for the Weighted-Majority Algorithm
8. Summary of Lecture

1 Introduction

We have studied PAC (and Probably *Exactly* Correct) Learning.

We have obtained an upper bound on sample complexity using $|H|$ and upper and lower bounds using $VC(H)$.

Throughout we have assumed the teacher drew examples from an unknown distribution, labelled them, and gave them to the learner.

However, Computational Learning Theory covers many other models of learning (Anthony and Biggs only introduces the basics).

Introduction

In this lecture we'll look at on-line learning.

- Instead of asking how many examples we need ...
- we'll ask how many mistakes the learner will make before learning a concept.

This lecture is based on Mitchell, chapter 7.

2 Some Models of Learning

Off-line Learning:

- Train first, test later.
- We divide data into training data and test data.
- Used when: we have the data in advance.
- Objective: good performance on test data.

On-line Learning:

- Train and test at the same time.
- We use the same data for training and testing.
- Used when: we don't have data in advance.
- Objective: good performance all the time.

Off-line Learning

We've used the off-line model since lecture 1.

```
Teacher: Example 001100 is positive.
Teacher: Example 010110 is positive.
Teacher: Example 011101 is negative.
Teacher: Example 001011 is positive.
Teacher: Example 111100 is negative.
⋮
```


On-line Learning

In on-line learning, the learner classifies the example before the teacher labels it.

```
Teacher: Classify 001100.
Learner: Positive.
Teacher: Yes, it is positive.

Teacher: Classify 110100.
Learner: Positive.
Teacher: No, it is negative.
⋮
```


Active Learning

There are many other models of learning. In Active Learning, the learner can ask the teacher questions:

```
Learner: Is example 001100 positive?
Teacher: No, it is negative.

Learner: Is example 111001 positive?
Teacher: Yes, it is positive.
⋮
```

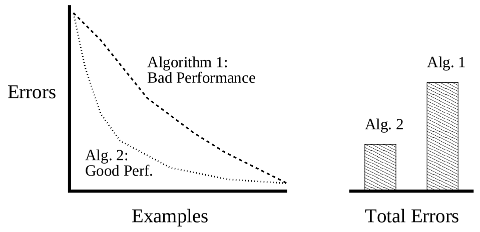
Here the learner chooses examples instead of passively waiting to see what the teacher shows it.

The learner has to do more work (choose examples), but can learn faster. E.g. the ray learning task (lecture 4) is much easier if the learner asks questions and avoids examples which are bigger than its current hypothesis.

3 The Mistake-Bound Model

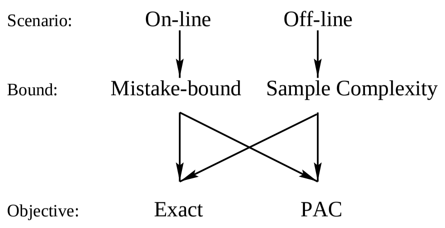
When learning on-line, we are interested in how many mistakes the learner makes before it learns the concept.

This is what the *mistake-bound model* is about.



On-line and Off-line Learning

We use the mistake-bound model for on-line learning. We can model either exact or PAC learning. (We consider exact learning today.)



Worst-case Mistake Bounds

We are specifically interested in the *worst-case* performance for algorithm L in C:

- Find the target concept $t \in C$ and training sequence on which L can produce the most errors.
- What is the maximum number of errors in this worst case?

4 Mistake Bound for the Monomial Learning Algorithm (FIND-S)

Here is a shorter (but equivalent) version of the standard algorithm for learning monomials from lecture 2 (page 11 of Anthony and Biggs):

1. Initialise h to the most specific hypothesis $\langle u_1\bar{u}_1u_2\bar{u}_2\ldots u_n\bar{u}_n\rangle$
2. For each positive example i
3. Delete from h any literal that is not satisfied by i

Note: Mitchell calls this the *FIND-S* algorithm (p. 26, 220).

What is the maximum number of mistakes this algorithm can make before learning the target concept exactly?

An Example of the Monomial Algorithm

Suppose there are $n = 3$ literals representing the following properties:

1. Tail
2. Whiskers
3. Black

and that the target concept "cat" is $\langle u_1u_2\rangle$, i.e., that cats have tails and whiskers. Suppose the training sample is:

$$((101, 0), (111, 1), (110, 1)).$$

The initial hypothesis is $\langle u_1\bar{u}_1u_2\bar{u}_2u_3\bar{u}_3\rangle$, and the subsequent hypotheses are:

$$\langle u_1\bar{u}_1u_2\bar{u}_2u_3\bar{u}_3\rangle, \langle u_1u_2u_3\rangle, \langle u_1u_2\rangle$$

Analysing the Algorithm

The algorithm converges to the target concept in the limit, provided $C \subseteq H$ and the training data is noise-free.

- The initial hypothesis $\langle u_1\bar{u}_1u_2\bar{u}_2\ldots u_n\bar{u}_n\rangle$ classifies everything negative. Remember: $\langle u_1\bar{u}_1u_2\bar{u}_2\ldots u_n\bar{u}_n\rangle = \langle u_1 \wedge \bar{u}_1 \wedge u_2 \wedge \bar{u}_2 \wedge \ldots \wedge u_n \wedge \bar{u}_n\rangle$ ...so it can never be satisfied (e.g., u_1 and $\bar{u}_1$ can never both be satisfied).
- We can think of the initial hypothesis as being fully specific. Each time it sees a positive example which contradicts some literal(s), it generalises the hypothesis by removing them.
- It can never classify negative examples as positive, since it makes minimal generalisations of the hypothesis to cover observed data.
- Each misclassified positive example deletes at least 1 literal.

Mistake Bound for the Algorithm

- The initial hypothesis contains $2n$ literals.
- The first mistake deletes n literals, leaving n literals.
- Each subsequent mistake deletes at least 1 literal.
- Total mistakes: at most $n + 1$.
  - That is, $n + 1$ mistakes will be made on the worst concept and training sequence.

How to make $n + 1$ mistakes.

- $n + 1$ mistakes occur only when the target concept is $\forall x\ t(x) = 1$, and the sequence of examples maximises errors. This is *very unlikely*.

5 Mistake-Bound for the Halving Algorithm

1. Version space = every hypothesis in H.
2. For each example x:
3. Classify the example using a majority vote in the version space.
4. Delete from version space any hypothesis h for which $h(x) \neq t(x)$.

Note 1: We break ties in the majority vote at random.

Note 2: This is very similar to the unbiased generalisation procedure from lecture 3.

Example of the Halving Algorithm

Let H consist of the following hypotheses:

y_1	y_2	$h^{(1)}$	$h^{(2)}$	$h^{(3)}$	$h^{(4)}$	$h^{(5)}$	$h^{(6)}$	$h^{(7)}$	$h^{(8)}$
0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	1	0	1	1
1	1	0	1	0	1	0	1	0	1

Compute the version space and predicted label for each example in:

$$\big((00,1),(10,1),(11,0),(01,1)\big)$$

How many mistakes are made?

Mistake Bound for the Halving Algorithm

The only time the algorithm makes a mistake is when the majority of the version space is inconsistent with the example (votes incorrectly).

For each mistake, all inconsistent hypotheses are deleted, so, for each mistake at least half the version space must be deleted.

So the maximum number of mistakes is: $\log_2|H|$.

(In fact, it can be shown that it is $\lfloor\log_2|H|\rfloor$.)

Notes on the Halving Algorithm

$\lfloor\log_2|H|\rfloor$ is a worst-case bound.

The halving algorithm can learn the target concept without making any mistakes.

Why? Because functions are deleted from the version space even when the majority vote is correct.

Questions:

1. Find an ordering of the training sequence in the last example which minimises the mistakes made by the halving algorithm.
2. How many mistakes will the algorithm make with this ordering?
3. Find an ordering which maximises the mistakes made.

6 Optimal Mistake Bounds

We've seen the mistake bounds for two algorithms.

For a given $C = H$, what is the *optimal mistake bound*, i.e., the lowest worst-case bound for any learning algorithm in C?

In other words, how badly can the best algorithm do in C?

- Hint: We've seen a bound of $\lfloor\log_2|H|\rfloor$ for the halving algorithm.

Littlestone (1987) shows that for any C:

$$VC(C) \quad\le\quad \text{Optimal mistake bound}(C) \quad\le\quad M_{Halving}(C) \quad\le\quad \log_2(|C|)$$

where $M_{Halving}(C)$ is the mistake bound for the halving algorithm.

In lecture 9 we saw that $VC(C)$ was related to the number of examples needed. Now we've seen it is related to the number of mistakes made.

7 Mistake-Bound for the Weighted-Majority Algorithm

The weighted-majority algorithm is a generalisation of the halving algorithm.

Instead of each hypothesis getting an equal vote, we weight votes according to the success of the voter.

- Each voter initially gets a weight of 1.
- For each incorrect prediction we decrease weight by multiplying it by $0 \le \beta < 1$.

Interesting Property: The number of mistakes made is related to the number of mistakes made by the best voter and the number of voters.

Notes on Weighted-Majority

- Voters can be hypotheses (within the same learning algorithm), or voters can be different learning algorithms.
- If $\beta > 0$ this algorithm *can* handle noisy training data, since it never completely eliminates voters – it only adjusts their weights.
- If $\beta = 0$ we have the halving algorithm.

The Weighted-Majority Algorithm

Let a_i index the voters (algorithms or hypotheses), w_i index the weights, and let $t \in C$ be the target concept.

```
1.  For all i initialise w_i ← 1
2.  For each example x
3.      Initialise q_0 and q_1 to 0
4.      For each voter a_i                        //Sum the weights
5.          If a_i(x) = 0 then q_0 ← q_0 + w_i
6.          If a_i(x) = 1 then q_1 ← q_1 + w_i
7.      If q_1 > q_0 then predict 1
8.      If q_0 > q_1 then predict 0
9.      If q_1 = q_0 then predict 0 or 1 at random
10.     For each voter a_i
11.         If a_i(x) ≠ t(x) then w_i ← βw_i      //Reduce weight if incorrect
```

Mistake-Bound for Weighted-Majority

Theorem.

Let:

- k be the minimum number of mistakes made by any voter.
- n be the number of voters.

The number of mistakes made by Weighted-Majority with $\beta = 1/2$ is at most:

$$2.4(k + \log_2 n)$$

Proof.

Let a_j be an algorithm which makes the optimal number of mistakes k.

- The final weight of a_j will be $(1/2)^k$ since its initial weight is 1 and is multiplied by $\beta = 1/2$ for each mistake.

Let $W = \sum_{i=1}^{n} w_i$ be the sum of all weights.

- W is initially n.

For each mistake W is reduced to at most $3/4W$.

- To make a mistake, at least $1/2W$ must vote incorrectly. These voters then lose $\beta = 1/2$ of their weight. The remaining weight is at most:

$$\text{incorrect half} \times \beta + \text{correct half} = 3/4W$$
$$(1/2)W \times (1/2) + 1/2W = 3/4W$$

Let M be the total mistakes made by weighted-majority. The final total weight is at most $n(3/4)^M$.

The final weight of a_j cannot be greater than the final total weight, so:

$$\left(\frac{1}{2}\right)^k \leq n\left(\frac{3}{4}\right)^M$$

which we can rewrite as:

$$M \quad \leq \quad \frac{(k + \log_2 n)}{-\log_2(3/4)} \quad \leq \quad 2.4(k + \log_2 n)$$

which proves the theorem. □

Further Notes on Weighted-Majority

- If we combine the predictions of a number of hypotheses or algorithms using the weighted-majority algorithm, the number of mistakes made depends on the performance of the best algorithm and the number of voters.
- Littlestone and Warmuth (1991) generalised the theorem to show that for any $0 \leq \beta < 1$ the bound is:

$$\frac{k \log_2(1/\beta) + \log_2 n}{\log_2\left(\frac{2}{1+\beta}\right)}$$

8 Summary of Lecture

We've seen two learning scenarios:

- Off-line learning: train first, test later.
- On-line learning: train and test at the same time.

We've seen two kinds of bounds on concept learning:

- Sample Complexity: how many examples do we need?
- Mistake Bound: how many mistakes will we make?

We calculate mistake bounds for on-line learning, and sample complexity bounds for off-line learning.

Summary of Lecture

To find the mistake-bound of an algorithm L in space C:

- Find the hardest $t \in C$ and ordering of training sequence which produces the most errors.
- What is the maximum number of mistakes in this case?

This gives us a *worst-case* bound on mistakes.

Summary of Lecture

The optimal mistake bound (mistake bound for the best algorithm) is related to $VC(C)$, the mistake bound for halving, and $\log_2(|C|)$.

If we combine the predictions of a number of hypotheses or algorithms using the weighted-majority algorithm, the number of mistakes made depends on the performance of the best algorithm and the number of voters.

9 References

N. Littlestone (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine Learning, 2, 285–318.

N. Littlestone and M. Warmuth (1991). The weighted majority algorithm (Technical report UCSC-CRL-91-28). University of California Santa Cruz, Computer Engineering and Information Sciences Dept., Santa Cruz, CA.

This lecture is based on part of chapter 7 of:

T. Mitchell (1997). Machine Learning. McGraw Hill.