

Intro. to machine learning (CSI 5325)

Lecture 17: Learning theory

Greg Hamerly

Some content from Tom Mitchell.

- 1 Mistake bounds
- 2 Tightness of bounds
- 3 Summary

Mistake Bounds

So far: how many examples needed to learn?

What about: how many mistakes before convergence?

Let's consider similar setting to PAC learning:

- Instances drawn at random from X according to distribution \mathcal{D}
- Learner must classify each instance before receiving correct classification from teacher.
- Can we bound the number of mistakes learner makes before converging?

Mistake Bounds: FIND-S

Consider FIND-S when $H =$ conjunction of boolean literals

FIND-S:

- Initialize h to the most specific hypothesis
 $l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \dots l_n \wedge \neg l_n$
- For each positive training instance x
 - Remove from h any literal that is not satisfied by x
- Output hypothesis h .

How many mistakes before converging to correct h ?

Mistake Bounds: HALVING Algorithm

Consider the HALVING Algorithm:

- Learn concept using version space CANDIDATE-ELIMINATION algorithm
- Classify new instances by majority vote of version space members

How many mistakes before converging to correct h ?

- ... in worst case?
- ... in best case?

Optimal Mistake Bounds

Let $M_A(C)$ be the max number of mistakes made by algorithm A to learn concepts in C . (maximum over all possible $c \in C$, and all possible training sequences)

$$M_A(C) \equiv \max_{c \in C} M_A(c)$$

Definition: Let C be an arbitrary non-empty concept class. The **optimal mistake bound** for C , denoted $Opt(C)$, is the minimum over all possible learning algorithms A of $M_A(C)$.

$$Opt(C) \equiv \min_{A \in \text{learning algorithms}} M_A(C)$$

$$VC(C) \leq Opt(C) \leq M_{Halving}(C) \leq \log_2(|C|).$$

WEIGHTED-MAJORITY algorithm

A generalization of the HALVING algorithm.

- uses multiple learning algorithms (or hypotheses)
- never completely eliminates any learner
- each algorithm a_i has an associated weight w_i
- use a weighted majority vote of the learners to make predictions
- instead of eliminating an algorithm which makes a mistake, just reduce its weight

Has nice bound on number of mistakes!

Connected to 'boosting' which we'll look at later.

WEIGHTED-MAJORITY algorithm (2)

Start with a group of learners $A = \{a_i\}$ and an initial weight $w_i = 1$ for each algorithm.

Then for each training example $\langle x, c(x) \rangle$,

- predict $+$ or $-$ for x depending on the weighted majority vote of the algorithms (randomly breaking ties)
- for each algorithm a_i that predicted incorrectly,
 - $w_i \leftarrow w_i \beta$

where $0 < \beta < 1$ is a user-chosen parameter.

Mistakes in the WEIGHTED-MAJORITY algorithm

Note that each learner (a_i) makes a mistake when it predicts $c(x)$ incorrectly.

However, the ensemble of learners only makes a mistake when the weighted majority of them makes a mistake.

We want a bound on how many mistakes the ensemble will make.

Mistake bound for WEIGHTED-MAJORITY

The WEIGHTED-MAJORITY algorithm makes at most

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

mistakes, where

- n is the number of learners
- k is the number of mistakes for the best learner
- β is assumed to be $1/2$
- the bound is over all possible training sequences

This is nice since it is close to the best learner!

Interesting question – what is the tradeoff between this bound and n , if we randomly choose the set of learning algorithms?

Tightness of bounds

Learning theory gives guarantees of performance (complexity, mistakes, etc.) for a concept class and a learner over any distribution of examples.

Of note:

- the learner's hypothesis space H could be huge (infinite, even)
- the distribution of examples could be anything

Because of these accommodations, PAC bounds, even ones based on VC dimension, tend to be very conservative.

Example: linear classifier

Linear classifier can only learn (shatter) 3 examples in 2 dimensions.

This means that theoretically, we can't learn anything more than 3 examples with a linear classifier in 2 dimensions.

In general, in d dimensions, only samples of size $d + 1$ can be learned perfectly.

Example: neural network

VC dimension of simplified neural network shape classifier:

- $n = 15 \times 15 = 225$ input nodes
- max number of inputs to any node is $r \leq 225$
- with 2 outputs and 10 hidden nodes, $s = 12$
- linear threshold perceptron as squashing function:
 $VC(\text{perceptron}) = r + 1$
- therefore, $VC(\text{net}) \leq 2(r + 1)s \log(es) \approx 18902$

So the sample complexity for $\epsilon = 0.1$ and $\delta = 0.05$ is:

$$m \geq \frac{1}{\epsilon} (4 \log_2(2/\delta) + 8VC(\text{net}) \log_2(13/\epsilon)) \approx 10^7$$

Rather large; but not as large as $|X| \ll 2^{225}$.

Reality versus theory

Theoretical bounds which allow any distribution over examples and very large hypothesis spaces fail to meet the reality of concepts which are relatively well-behaved.

Because of the conservative nature behind these bounds, it's not clear how useful they are in practice.

Thus, ongoing work is tightening bounds based on better analysis and domain-specific assumptions.

PAC-Bayes bounds

For example, the PAC-Bayes framework (McAllester and others) combines PAC analysis with Bayesian-style priors on the functions that will be learned.

This can tighten bounds by assuming that really complex functions which require lots of training are less likely to occur.

PAC-Bayes bounds (McAllester 1998)

Assume a prior probability distribution over the hypothesis space, where $P(h)$ is the probability of hypothesis h , and $H = \mathcal{C}$.

For a consistent learner and any $\delta > 0$, the following holds with probability at least $1 - \delta$ for all hypotheses consistent with a sample of size m :

$$\text{error}_{\mathcal{D}}(h) \leq \frac{\log(1/P(h)) + \log(1/\delta)}{m}$$

Thus the sample complexity is

$$m \geq \frac{1}{\epsilon} (\log(1/P(h)) + \log(1/\delta))$$

If h has high probability, then the sample complexity will be much smaller than the bound based on $|H|$ or $VC(H)$.

Thoughts on learning theory

At a very high level, they often boil down to: amount of training is related to the complexity of the functions you are training.

More complex functions (e.g. large $|H|$ or $VC(H)$) require more training (i.e. more examples).

Remember that learning theory results are typically very generic.

As such, they are guidelines but very loose. Making them more specific requires incorporating specifics about problems you're working on.