Learning Theory Part 2: Mistake Bound Model

Yingyu Liang
Computer Sciences 760
Fall 2017

http://pages.cs.wisc.edu/~yliang/cs760/

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Matt Gormley, Elad Hazan, Tom Dietterich, and Pedro Domingos.

Goals for the lecture

you should understand the following concepts

- the on-line learning setting
- the mistake bound model of learnability
- the Halving algorithm
- the Weighted Majority algorithm

Learning setting #2: on-line learning

Now let's consider learning in the *on-line* learning setting:

for $t = 1 \ldots$

learner receives instance $\boldsymbol{x}^{(t)}$

learner predicts $h(\boldsymbol{x}^{(t)})$

learner receives label $c(\boldsymbol{x}^{(t)})$ and updates model h

The *mistake bound* model of learning

How many mistakes will an on-line learner make in its predictions before it learns the target concept?

the *mistake bound model* of learning addresses this question

Mistake bound example: learning conjunctions with FIND-S

consider the learning task

- training instances are represented by n Boolean features
- target concept is conjunction of up to n Boolean (negated) literals

FIND-S:

initialize h to the most specific hypothesis $x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \ldots\ x_n \wedge \neg x_n$

for each positive training instance $\boldsymbol{x}$

  remove from h any literal that is not satisfied by $\boldsymbol{x}$

output hypothesis h

Example: using FIND-S to learn conjunctions

- suppose we're learning a concept representing the sports someone likes
- instances are represented using Boolean features that characterize the sport

Snow	(is it done on snow?)
Water	
Road	
Mountain	
Skis	
Board	
Ball	(does it involve a ball?)

Example: using FIND-S to learn conjunctions

$t = 0$ $\quad h$: $snow \wedge \neg snow \wedge water \wedge \neg water \wedge road \wedge \neg road \wedge mountain \wedge \neg mountain \wedge skis \wedge \neg skis \wedge board \wedge \neg board \wedge ball \wedge \neg ball$

$t = 1$ $\quad \boldsymbol{x}$: $snow, \neg water, \neg road, mountain, skis, \neg board, \neg ball$

$h(\boldsymbol{x}) =$ false $\quad c(\boldsymbol{x}) =$ true

h: $snow \wedge \neg water \wedge \neg road \wedge mountain \wedge skis \wedge \neg board \wedge \neg ball$

$t = 2$ $\quad \boldsymbol{x}$: $snow, \neg water, \neg road, \neg mountain, skis, \neg board, \neg ball$

$h(\boldsymbol{x}) =$ false $\quad c(\boldsymbol{x}) =$ false

$t = 3$ $\quad \boldsymbol{x}$: $snow, \neg water, \neg road, mountain, \neg skis, board, \neg ball$

$h(\boldsymbol{x}) =$ false $\quad c(\boldsymbol{x}) =$ true

h: $snow \wedge \neg water \wedge \neg road \wedge mountain \wedge \neg ball$

⋮

Mistake bound example: learning conjunctions with FIND-S

the maximum # of mistakes FIND-S will make $= n + 1$

Proof:

- FIND-S will never mistakenly classify a negative (h is always at least as specific as the target concept)
- initial h has $2n$ literals
- the first mistake on a positive instance will reduce the initial hypothesis to n literals
- each successive mistake will remove at least one literal from h

Halving algorithm

// initialize the version space to contain all $h \in H$

$VS_0 \leftarrow H$

for $t \leftarrow 1$ to T do

given training instance $\boldsymbol{x}^{(t)}$

// make prediction for $\boldsymbol{x}$

$h'(\boldsymbol{x}^{(t)}) =$ MajorityVote(VS_t, $\boldsymbol{x}^{(t)}$)

given label $c(\boldsymbol{x}^{(t)})$

// eliminate all wrong h from version space (reduce the size of the VS by at least half on mistakes)

$VS_{t+1} \leftarrow \{h \in VS_t : h(\boldsymbol{x}^{(t)}) = c(\boldsymbol{x}^{(t)}) \}$

return VS_{t+1}

Mistake bound for the Halving algorithm

the maximum # of mistakes the Halving algorithm will make $= \lfloor \log_2 |H| \rfloor$

Proof:

- initial version space contains $|H|$ hypotheses
- each mistake reduces version space by at least half

$\lfloor a \rfloor$ is the largest integer not greater than a

Optimal mistake bound

[Littlestone, *Machine Learning* 1987]

let C be an arbitrary concept class

$$VC(C) \le M_{opt}(C) \le M_{Halving}(C) \le \log_2(|C|)$$

mistakes by best algorithm (for hardest $c \in C$, and hardest training sequence)

mistakes by Halving algorithm

The Weighted Majority algorithm

given: a set of predictors $A = \{a_1 \ ... \ a_n\}$, learning rate $0 \leq \beta < 1$

for all i initialize $w_i \leftarrow 1$
for $t \leftarrow 1$ to T do
  given training instance $\boldsymbol{x}^{(t)}$
  // make prediction for $\boldsymbol{x}$
  initialize q_0 and q_1 to 0
  for each predictor a_i
    if $a_i(\boldsymbol{x}^{(t)}) = 0$ then $q_0 \leftarrow q_0 + w_i$
    if $a_i(\boldsymbol{x}^{(t)}) = 1$ then $q_1 \leftarrow q_1 + w_i$
  if $q_1 > q_0$ then $h(\boldsymbol{x}^{(t)}) = 1$
  else if $q_0 > q_1$ then $h(\boldsymbol{x}^{(t)}) \leftarrow 0$
  else if $q_0 = q_1$ then $h(\boldsymbol{x}^{(t)}) \leftarrow 0$ or 1 randomly chosen

  given label $c(\boldsymbol{x}^{(t)})$
  // update hypothesis
  for each predictor a_i do
    if $a_i(\boldsymbol{x}^{(t)}) \neq c(\boldsymbol{x}^{(t)})$ then $w_i \leftarrow \beta \, w_i$

The Weighted Majority algorithm

- predictors can be individual features or hypotheses or learning algorithms
- if the predictors are all $h \in H$, then WM is like a weighted voting version of the Halving algorithm
- WM learns a linear separator, like a perceptron
- weight updates are multiplicative instead of additive (as in perceptron/neural net training)
  - multiplicative is better when there are many features (predictors) but few are relevant
  - additive is better when many features are relevant
- approach can handle noisy training data

Relative mistake bound for Weighted Majority

Let

- D be any sequence of training instances
- A be any set of n predictors
- k be minimum number of mistakes made by best predictor in A for training sequence D

- the number of mistakes over D made by Weighted Majority using $\beta = 1/2$ is at most

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

Comments on mistake bound learning

- we've considered mistake bounds for learning the target concept exactly
- there are also analyses that consider the number of mistakes until a concept is PAC learned
- some of the algorithms developed in this line of research have had practical impact (e.g. Weighted Majority, Winnow)
  [Blum, *Machine Learning* 1997]