

PAC Learning

Hal Daumé III

CS5350: Machine Learning

21 October 2008

1. Why learning theory?
2. The PAC learning model
3. PAC learning by example:
 - 3.1 Axis aligned rectangles
 - 3.2 Boolean conjunctions
4. Occam's razor
5. Summary and outlook

Why learning theory?

We've put a lot of effort into **models** of learning and **algorithms** for solving those problems.

How can we tell if we've done a good job?

- ▶ Experimental results
- ▶ Theoretical analysis

Why theory?

- ▶ I can only run so many experiments
- ▶ Experiments rarely tell me what will go wrong
- ▶ I want to be able to deploy my algorithm on Mars

Why learning theory?

What can we hope for in terms of an algorithm? (Lets even allow ourselves infinite data)

- ▶ Always is optimal?
- ▶ Frequently is optimal?
- ▶ Always is almost-optimal?
- ▶ Frequently is almost-optimal? \Leftarrow **Our Goal**

No Free Lunch Theorem (Wolpert, 2001) states:

*In a **noise-free scenario** where the loss function is the **misclassification rate**, if one is interested in **off-training-set error**, then **all algorithms are equivalent, on average**.*

Why?

Formal (classification) problem:

Some terminology:

- ▶ **Hypothesis** (h) – the thing we learn
- ▶ **Hypothesis class** (\mathcal{H}) – the space of all h
- ▶ **Instance space** (\mathcal{X}) – the inputs
- ▶ **Target distribution** (\mathcal{D}) – any probability distribution over \mathcal{X}
- ▶ **Target concept** (c) – a mapping from \mathcal{X} to $\{0, 1\}$, what we learn

Given: $x_1, \dots, x_N \sim \mathcal{D}$, $y_1 = c(x_1), \dots, y_N = c(x_N)$ and \mathcal{H}

Find: $h \in \mathcal{H}$ to minimize error:

$$\begin{aligned} \overbrace{\mathbb{E}_{x \sim \mathcal{D}} [\mathbf{1}(c(x) \neq h(x))]}^{\text{expected loss}} &= \int_{\mathcal{X}} \mathbf{1}(c(x) \neq h(x)) d\mathcal{D}(x) \\ &\neq \underbrace{\sum_n \mathbf{1}(c(x_n) \neq h(x_n))}_{\text{training loss}} \end{aligned}$$

Probably approximately correct learning

Idea is that I want a learning algorithm that:

- ▶ Finds a solution on most data sets \Leftarrow "probably"
- ▶ Finds a solution that's close to optimal \Leftarrow "approximately"

A concept class \mathcal{C} is **PAC learnable** using \mathcal{H} if there exists an algorithm \mathcal{L} with the following property. Choose:

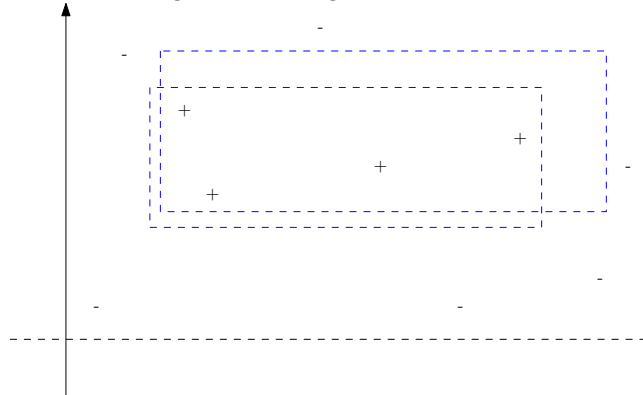
- ▶ Some concept $c \in \mathcal{C}$
- ▶ Some distribution over inputs, \mathcal{D} over \mathcal{X}
- ▶ An error parameter $0 < \epsilon < \frac{1}{2}$
- ▶ A confidence parameter $0 < \delta < \frac{1}{2}$

Then \mathcal{L} , given access to labeled examples, must with probability at least $1 - \delta$ output a hypothesis with error at most ϵ .

\mathcal{C} is **efficiently PAC learnable** if \mathcal{L} needs time polynomial in ϵ^{-1} , δ^{-1} .

Learning axis-aligned rectangles

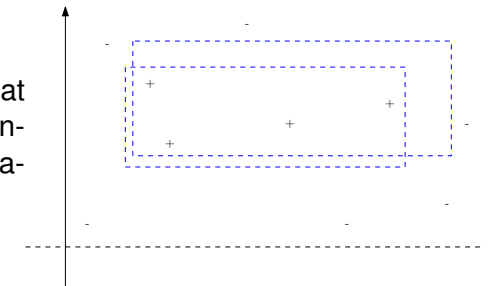
Data points are drawn from 2-dimensional real space (\mathcal{X}), concept (c) is an axis aligned rectangle:



Hypothesis space \mathcal{H} is a rectangle. Error is the probability that a point falls in $c\Delta h = (c - h) \cup (h - c)$

Learning axis-aligned rectangles

Suppose we have an algorithm that always returns the smallest rectangle that contains all the positively labeled points.



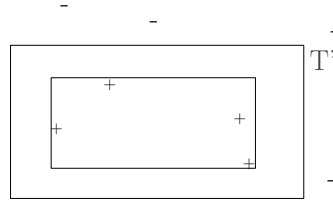
The algorithm shows that this problem is (efficiently) PAC learnable if the number of examples required to meet the ϵ and δ requirement is sufficiently small.

N , the minimum number of required examples, is the **sample complexity** of the algorithm.

Learning axis-aligned rectangles

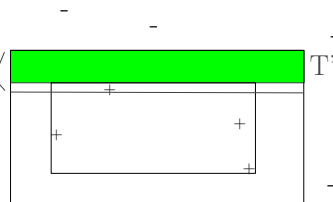
Observations:

- ▶ The hypothesis is always contained in the concept
- ▶ Thus, $c\Delta h = c - h$



Let T' be the "top missing" part of the concept rectangle, T be the top $\epsilon/4$ part of the concept rectangle.

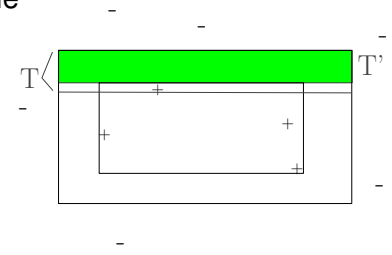
T' includes T if and only if no point in T appears in the sample.



Learning axis-aligned rectangles

Recall: T' is "top missing" part, T be the top $\epsilon/4$ part.

- ▶ Probability that any draw from \mathcal{D} misses T is $1 - \epsilon/4$
- ▶ Probability all N draws miss T is $(1 - \epsilon/4)^N$



Repeating on bottom, left and right, the probability that *any* of the four strips has weight greater than $\epsilon/4$ is at most $4(1 - \epsilon/4)^N$.

So we need to choose N big enough that $4(1 - \epsilon/4)^N \leq \delta$.

Use the fact that $(1 - x) \leq e^{-x}$ and require $4e^{-\epsilon N/4} \leq \delta$

Solving, we get $N \geq (4/\epsilon) \log(4/\delta)$; $\epsilon = \delta = 0.05 \Rightarrow N \geq 350$

Learning boolean conjunctions

Input space is a sequence of bits. Concept is a conjunction of these bits or negations thereof.

Example:

- ▶ $\mathcal{X} = \{0, 1\}^4$
- ▶ $c(x_1, x_2, x_3, x_4) = x_1 \wedge \neg x_3 \wedge x_4$
- ▶ $c(0, 0, 0, 0) = 0 \wedge 1 \wedge 0 = 0$
- ▶ $c(1, 1, 0, 1) = 1 \wedge 1 \wedge 1 = 1$
- ▶ $c(1, 1, 1, 1) = 1 \wedge 0 \wedge 1 = 0$

Algorithm:

- ▶ Initialize $h = x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \dots \wedge x_D \wedge \neg x_D$
- ▶ Ignore negative examples (those with $c(x) = 0$)
- ▶ On positive example with $x_d = 1$, remove $\neg x_d$; with $x_d = 0$, remove x_d

Learning boolean conjunction

Note that h always contains at least as many literals as c .

Take some literal z :

- ▶ z causes h to err only on positive examples with $z = 0$
- ▶ these are *exactly* those examples that would "fix" this error
- ▶ let $p(z)$ be the probability of these happening
- ▶ then the error is $\leq \sum_{z \in h} p(z)$
- ▶ define a literal as "bad" if $p(z) \geq \epsilon/(2D)$
- ▶ no bad literals implies success

Goal: bound the probability that a bad literal appears.

Learning boolean conjunctions

Recall: z is bad if $p(z) \geq \epsilon/(2D)$, where $p(z)$ is the probability that we draw a positive example with $z = 0$.

- ▶ The probability that **some** bad z is not deleted is at most $(1 - \epsilon/(2D))^N$
- ▶ The probability that **any** bad z is not deleted is at most $2D(1 - \epsilon/(2D))^N$
- ▶ Bound this by δ and apply $1 - x \leq e^{-x}$

$$N \geq \frac{2D}{\epsilon} \left(\log(2D) + \log \frac{1}{\delta} \right)$$

- ▶ For $\epsilon = \delta = 0.05$, $D = 20$, we get $N \geq 5348$

On to Occam's razor...

What and Why of Occam's razor?

- ▶ Named after William of Occam
- ▶ Idea: prefer simple hypotheses that explain the data
- ▶ In learning: want a model that fits the data but has a small representation
- ▶ Main result: Occam algorithms are also PAC algorithms

Occam's razor and PAC learning

Given:

- ▶ A concept class \mathcal{C}
- ▶ A hypothesis space \mathcal{H}
- ▶ A distribution \mathcal{D}
- ▶ A fixed but unknown concept $c \in \mathcal{C}$
- ▶ N labeled examples sampled from \mathcal{D}
- ▶ A polytime algorithm \mathcal{L} that outputs $h \in \mathcal{H}$ **consistent with the sample**

Then: \mathcal{L} PAC-learns the concept class so long as:

$$\log |\mathcal{H}| \leq bN\epsilon - \log \frac{1}{\delta}$$

for some constant b .

In other words:

$$N \geq \frac{1}{b\epsilon} \left[\log |\mathcal{H}| + \log \frac{1}{\delta} \right]$$

Learning boolean conjunctions, revisited

Remember our Algorithm:

- ▶ Initialize $h = x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \dots \wedge x_D \wedge \neg x_D$
- ▶ Ignore negative examples (those with $c(x) = 0$)
- ▶ On positive with $x_d = 1$, remove $\neg x_d$; with $x_d = 0$, remove x_d

We obtained $N \geq (2D/\epsilon)[\log(2D) + \log(1/\delta)]$

Clearly this algorithm is always consistent. Occam says:

$$\begin{aligned} N &\geq \frac{1}{b\epsilon} \left[\log |\mathcal{H}| + \log \frac{1}{\delta} \right] \\ &= \frac{1}{b\epsilon} \left[\log(3^D) + \log \frac{1}{\delta} \right] \\ &= \frac{1}{b\epsilon} \left[D \log 3 + \log \frac{1}{\delta} \right] \end{aligned}$$

What if we're inconsistent?

Another way of seeing the Occam bound is as a bound on the expected loss of a consistent hypothesis h :

$$\text{ExpectedLoss}(h) \leq \sqrt{\frac{\log |\mathcal{H}| + \log \frac{1}{\delta}}{2N}}$$

What happens if we don't have a consistent hypothesis? I.e., if $\text{TrainLoss}(h) > 0$?

The obvious thing:

$$\text{ExpectedLoss}(h) \leq \text{TrainLoss}(h) + \sqrt{\frac{\log |\mathcal{H}| + \log \frac{1}{\delta}}{2N}}$$

Summary and outlook

Summary:

- ▶ No universally good learning algorithm exists
- ▶ We care about *expected error*, not *training error*
- ▶ The measure of importance is *sample complexity*
- ▶ We can often bound expected error by using the fact that “if it were important, we would have seen it already”
- ▶ Occam's razor tells us that **consistency** and **small \mathcal{H}** is sufficient for generalization – we need $\mathcal{O}((1/\epsilon)[\log |\mathcal{H}| + \log(1/\delta)])$ examples

Outlook:

- ▶ What happens when there's noise?
- ▶ What happens when we have infinite hypothesis classes?