PAC Learning

prof. dr Arno Siebes

Algorithmic Data Analysis Group
Department of Information and Computing Sciences
Universiteit Utrecht

Recall: PAC Learning (Version 1)

A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm A with the following property:

- for every $\epsilon, \delta \in (0,1)$
- for every distribution $\mathcal{D}$ over $\mathcal{X}$
- for every labelling function $f : \mathcal{X} \to \{0,1\}$

If the realizability assumption holds wrt $\mathcal{H}, \mathcal{D}, f$, then

- when running A on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples generated by $\mathcal{D}$ labelled by f
- A returns a hypothesis $h \in \mathcal{H}$ such that with probability at least $1 - \delta$

$$L_{(\mathcal{D},f)}(h) \leq \epsilon$$

Recall: Finite Hypothesis Sets

And we had a theorem about the PAC learnability:

> *Every finite hypothesis class $\mathcal{H}$ is PAC learnable with sample complexity*

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil$$

And, we even know an algorithm that does the trick: the halving algorithm.

Before we continue, however, it is worthwhile to consider some concrete classes. In this case, boolean expressions, starting with conjunctions of boolean literals

Boolean Literals

Let $x_1, \ldots, x_n$ be boolean variables. A literal is

1. a boolean variable x_i
2. or its negation $\neg x_i$

The concept class (aka hypothesis class) C_n

- consists of conjunctions of upto n literals
- in which each variable occurs at most once

For example, for $n = 4$, $x_1 \wedge \neg x_2 \wedge x_4 \in C_4$

- $(1, 0, 0, 1)$ is a positive example of this concept
- while $(0, 0, 0, 1)$ is a negative example

(as usual, we equate 1 with *true* and 0 with *false*)

Similarly, $\neg x_1 \wedge x_3 \wedge \neg x_4$ has

- has $(0, 0, 1, 0)$ and $(0, 1, 1, 0)$ as positive examples
- and $(1, 0, 0, 0)$, $(1, 0, 1, 0)$, and $(0, 1, 1, 1)$ as negative examples.

C_n is PAC

Clearly, C_n is finite, so C_n is PAC learnable. In fact

$$|C_n| = 3^n$$

(either x_i is in the expression or $\neg x_i$ is in, or neither is in).

Hence we have sample complexity:

$$m_{C_n} \leq \left\lceil \frac{\log(3^n/\delta)}{\epsilon} \right\rceil = \left\lceil \frac{n\log(3) + \log(1/\delta)}{\epsilon} \right\rceil$$

For example, for $\delta = 0.02$, $\epsilon = 0.01$ and $n = 10$

- we need at least 149 examples
- with these 149 examples the bound guarantees (at least) 99% accuracy with (at least) 98% confidence

Learning C_n

Let $b = (b_1, \ldots b_n)$ be an example to learn an expression in C_n, if

- b is a positive example and $b_i = 1$, then $\neg x_i$ is ruled out
- b is a positive example and $b_i = 0$, then x_i is ruled out
- if b is a negative example, we can conclude nothing (we do not know which of the conjuncts is false)

A simple algorithm is, thus,

- start with the set of all possible literals
- with each positive example delete the literals per above
- when all positive examples have been processed return the conjunction of all remaining literals
- if all examples were negative, you have learned nothing

Note that this is obviously polynomial.

Conjunctive Normal Form

A conjunctive normal form formula is conjunction of disjunctions, more in particular, a k-CNF formula T is

- an expression of the form $T = T_1 \wedge \cdots \wedge T_j$
- in which each T_i is the disjunction of at most k literals

Note that even if we do not specify a maximal $j \in \mathbb{N}$

- something we don't do on purpose

k-CNF is obviously finite. For there are only

$$\sum_{i=1}^{k} \binom{2n}{i}$$

disjunctions of at most k literals. Without specifying a maximal j, computing an expression for the sample complexity is rather cumbersome and is, thus, left as an exercise.

Learning k-CNF

For each possible disjunction $u_1 \vee \ldots \vee u_l$ of at most k literals from $\{x_1, \ldots, x_n\}$ we introduce a new variable $w_{(u_1,\ldots,u_l)}$, where the truth value of this new variable is determined by

$$w_{(u_1,\ldots,u_l)} = u_1 \vee \ldots \vee u_l$$

If we now transform our examples to these new variables, learning k-CNF reduces to learning C_m, which is polynomial.

Note that by transforming the examples, we transform the distribution. This doesn't matter as PAC learning is agnostic to the underlying distribution.

Disjunctive Normal Form

Similarly to the conjunctive normal form, we can consider disjunctive normal form formula, i.e., disjunctions of conjunctions. More in particular, we consider k-DNF formula, which consist of

- the disjunction of at most k terms
- where each term is the conjunction of at most n literals

(note that more than n literals lead to an always false term anyway).

There are 3^{nk} such disjunctions, and hence, the sample complexity is given by

$$m_{k\text{-DNF}} \leq \left\lceil \frac{nk \log(3) + \log(1/\delta)}{\epsilon} \right\rceil$$

Learning k-DNF

Given that we need a modest (polynomial) sample size and learning, e.g., k-CNF is polynomial, you may expect that learning k-DNF is polynomial as well.

- unfortunately, it is *not*

Well, more precisely, *we do not know*

- the reduction from the graph k-colouring problem (which is **NP** hard)
- to k-DNF learning (i.e., this is in **NP**)
  - turn the graph into a sample and show that there exists a corresponding k-DNF formula iff the graph is k-colourable)
- shows that there is *no* efficient (polynomial) algorithm for k-DNF learning
- *unless* **RP** $=$ **NP**
  - if $\mathcal{H}$ is (polynomially) PAC learnable, $ERM_{\mathcal{H}}$ is in **RP**
- which is considered unlikely

Randomised Polynomial

The class **RP** consists of those decision problems (Yes/No problems) for which there exists a probabilistic algorithm (it is allowed to flip coins while running), such that

1. the algorithm is polynomial in its input size
2. if the correct answer is NO, the algorithm answers NO
3. there exists a constant $a \in (0, 1)$ such that if the correct answer is YES, the algorithm will answer YES with probability a and NO with probability $1 - a$.

note that often $a \geq 1/2$ is assumed, but that is not necessary.

Obviously, $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$

- for both relations it is, however, unknown whether or not equality holds

Why Randomised is Good

The importance of **RP** algorithms is that

- if the answer is YES, you are done
- if the answer is NO, you simply run again, after all,

$$\forall a \in (0,1) : lim_{n\to\infty}(1-a)^n = 0$$

- hence, if after a 1000 trials you only have seen NO, the answer probably is NO

Note that since the algorithm is polynomial, one run takes polynomial time, this loop is a viable approach

The best known problem in **RP** (well, the one I learned when I was a student) is probably primality testing

- although there is a polynomial algorithm since 2002.

But, first a simple example of a randomised algorithms

Searching for a

We have array A of size n

- half the entries are a
- the other half is b

and we want to find an entry a

The simple solution is to

- to iterate over $A[i]$ until an a is encountered

This is an $\mathcal{O}(n)$ algorithm

- can we do it faster?

By randomization we can, probably

Randomised Search

The idea is simple

- choose k elements from A
- inspect the k entries
  - no a among them: probability $\left(\frac{1}{2}\right)^k$
  - at least one a: probability $1 - \left(\frac{1}{2}\right)^k$

Since k is a constant this is $\Theta(1)$ algorithm

You can always run it multiple times

Miller Rabin

From elementary number theory: is n prime:

- n is presumed prime, hence $n-1$ is even (you are not going to test whether or not 2 is prime, are you?)
- hence, $n-1 = 2^r d$, for an odd d

If there is an $a \in \{0, 1, \ldots, n\}$ such that

1. $a^d \not\equiv 1 \bmod n$ and
2. $\forall s \in 0, \ldots, r-1 : a^{2^s d} \not\equiv -1 \bmod n$

then n is *not* prime

The $\mathbf{co-RP}$ algorithm is, thus:

- choose a random $a \in \{0, 1, \ldots, n\}$
- perform the tests above
  - if the answer is NO, you are done ($\geq 3/4$ of the possible a values will witness NO if n is composite)
  - otherwise, repeat as often as you want.

Wait A Second!

Both CNF and DNF are normal forms for Boolean expressions

- this means in particular that every CNF formula can be rewritten in a DNF formula and vice versa
  - note that k disjuncts do not necessarily translate to k conjuncts or vice versa
  - hence our liberal view of k-CNF

So, how can it be that

- PAC learning CNF is polynomial
- while PAC learning DNF is not?

The reason is simple rewriting is (probably) harder than you think:

- learning a CNF and then rewriting it to a DNF is in **NP**
- learning a CNF is easy, so rewriting is hard

In learning:

representation matters

A Larger Class

Each k-CNF formula, each k-DNF formula, and each C_n formula determines a subset of $\mathbb{B}^n$ (the set of boolean n-tuples)

- the subset of $\mathbb{B}^n$ that renders the formula true

What if we want to learn arbitrary subset of $\mathbb{B}^n$? This is known as the universal concept class U_n.

Note that $|U_n| = 2^{2^n}$, finite, but very large. Per our theorem, the sample complexity is in the order of

$$\left\lceil \frac{2^n \log(2) + \log(1/\delta)}{\epsilon} \right\rceil$$

Which is exponential in n, the number of variables. So, yes U_n is finite and hence PAC learnable, but we will need exponential time (to inspect exponentially many examples).

- it is *not* PAC learnable in any practical sense

In General

PAC learning is obviously in **NP**

- if I claim a solution, you can easily check it

In fact, a similar claim can be made for all learning algorithms

In other words, learning is easy if

$$\mathbf{P} = \mathbf{NP}$$

For some people this is a reason to believe that

- $\mathbf{P} \neq \mathbf{NP}$

I think that such arguments are silly

PAC Learning Revisited

The examples k-DNF and U_n tell us that we should be more restrictive. Leslie Valiant (one of the inventors of PAC learning for which he got the 2010 ACM Turing award) requires:

- that the mapping $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ is polynomial
  - more precisely $poly(1/\epsilon, 1/\delta, n, |\mathcal{H}|)$
- as well as that the learning algorithm A runs in polynomial time.
  - more precisely $poly(1/\epsilon, 1/\delta, n, |\mathcal{H}|)$

Recall that the n in $poly(1/\epsilon, 1/\delta, n, |\mathcal{H}|)$ is the dimensionality of our data. So, e.g., U_n is *not* PAC learnable in the sense of Valiant and k-DNF probably isn't either

For the moment, we will *not* require these conditions,

- we will return to the first one later in the course

Loosening Up

Up until now, we have assumed that

- $\mathcal{H}$ contains a hypothesis that is consistent with the examples

Loosely speaking (a bit too loose):

- we have assumed that $\mathcal{H}$ contains the true hypothesis

This is in practice not very often a realistic assumption

- would human behaviour really be governed by a polynomial?

That is, it is reasonable to assume that $L_{\mathcal{D}}(h) > 0$ for all $h \in \mathcal{H}$

A reasonable question is then:

- how much worse does our estimate of the loss get by using a sample?
- i.e., how big is $|L_{\mathcal{D}}(h) - L_D(h)|$ for a finite $\mathcal{H}$?

According to Hoeffding

Hoeffdings inequality tells that:

For any $\epsilon > 0$, for any D an i.i.d. sample of size m, and for any hypothesis $h : X \to \{0, 1\}$:

$$\mathbb{P}_{D \sim \mathcal{D}^m} \left(|L_{\mathcal{D}}(h) - L_D(h)| \geq \epsilon \right) \leq 2e^{-2m\epsilon^2}$$

A little algebra then tells us that:

For a given hypothesis $h : X \to \{0, 1\}$ and for any $\delta > 0$ we now with probability ate least $1 - \delta$:

$$L_{\mathcal{D}}(h) \leq L_D(h) + \sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

Now, all we have to do is go from a single hypothesis to $\mathcal{H}$

Learning, the Finite, Inconsistent, Case

For a finite hypothesis set $\mathcal{H}$, an i.i.d. sample of size m, and any $\delta > 0$, we know with probability at least $1 - \delta$ that:

$$\forall h \in \mathcal{H} : L_{\mathcal{D}}(h) \leq L_D(h) + \sqrt{\frac{\log |\mathcal{H}| + \log \frac{2}{\delta}}{2m}}$$

Proof:

$$\begin{aligned}
&\mathbb{P}\left(\exists h \in \mathcal{H} : |L_D(h) - L_{\mathcal{D}}(h)| > \epsilon\right) \\
&\quad = \mathbb{P}\left(\left(|L_D(h_1) - L_{\mathcal{D}}(h_1)| > \epsilon\right) \vee \cdots \vee \left(|L_D(h_k) - L_{\mathcal{D}}(h_k)| > \epsilon\right)\right) \\
&\quad \leq \sum_{h \in \mathcal{H}} \mathbb{P}\left(|L_D(h) - L_{\mathcal{D}}(h)| > \epsilon\right) \\
&\quad \leq 2|\mathcal{H}|e^{-2m\epsilon^2}
\end{aligned}$$

Solving $\delta = 2|\mathcal{H}|e^{-2m\epsilon^2}$ finishes the proof.

At What Cost?

In the consistent case, our bound is in terms of

- $\log |\mathcal{H}|$, $\log \frac{1}{\delta}$, and $\frac{1}{m}$

In the inconsistent case

- it is in the square root of such terms

More in particular this means that

- we need the square of the number of examples in the inconsistent case vs the consistent case for the same accuracy

In other words

- assuming the true hypothesis is in $\mathcal{H}$ gives you a square root advantage
  - your algorithm will finish far quicker
- if you give up this assumption, you need a far bigger sample
  - your algorithm will become considerably slower

Does this mean that one should simply assume consistency?

- *No*, of course not, one should only make realistic assumptions when modelling data

Loosening Up Further

There are two further assumptions in our definition of PAC learning that could do with relaxation:

- we assume that our data labelling is governed by a function
  - not very realistic, it doesn't allow for noise or errors
- it is completely tailored to *one* learning task
  - concept learning, i.e., binary classification

We loosen the first by *agnostic* PAC learning and the second by allowing arbitrary loss-functions.

Classification Reconsidered

As noted on the previous slide, assuming that the the data is labelled by a function is not very realistic

- losing this assumption means that we go back to a probability distribution on $X \times Y$, i.e.,

$$\mathcal{D} = \mathcal{D}|_X \times \mathcal{D}|_{Y|X} = \mathcal{X} \times \mathcal{Y}$$

For the (usual) two-class classification problem we thus have:

$$X \times \{0, 1\} \sim \mathcal{D}$$

In this setting the best label prediction is

$$f_{\mathcal{D}} = \begin{cases} 1 & \text{if } \mathbb{P}(y = 1 \mid x) \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

which is known as the Bayes optimal prediction. Since we do not know $\mathcal{D}$ we cannot hope to do as good as this predictor. What we can hope is that we do not do much worse.

Agnostic PAC Learning

A hypothesis class $\mathcal{H}$ is agnostic PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm A with the following property:

- for every $\epsilon, \delta \in (0,1)$
- for every distribution $\mathcal{D}$ over $X \times Y$
- when running A on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples generated by $\mathcal{D}$
- A returns a hypothesis $h \in \mathcal{H}$ such that with probability at least $1 - \delta$

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon$$

In other words, A yields a result which is probably close to the best possible result.

Note that if the realizability assumption holds, agnostic PAC learning reduces to (the previous definition of) PAC learning; after all, that assumption tells us that $\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') = 0$

Loss Functions

Until know we focussed on one definition of loss

- our aim was to minimize the number of errors

While this may seem reasonable for a two-class classification it isn't always

- suppose that a simple knock on the head saves your life if you suffer from X
- clearly misclassifying you as *not* suffering from X is much worse than the other way around

The general setting is: we have a stochastic variable Z with distribution $\mathcal{D}$ and a set of hypothesis $\mathcal{H}$. A loss function l is a function

$$l : Z \times \mathcal{H} \to \mathbb{R}_+$$

The loss of a hypothesis $h \in \mathcal{H}$ is then defined by

$$L_{\mathcal{D}}(h) = E_{z \sim \mathcal{D}} l(z, h)$$

PAC Learning, The General Case

A hypothesis class $\mathcal{H}$ is agnostic PAC learnable with respect to a set Z and a loss function $l : Z \times \mathcal{H} \to \mathbb{R}_+$ if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm A with the following property:

- for every $\epsilon, \delta \in (0,1)$
- for every distribution $\mathcal{D}$ over Z
- when running A on $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. samples generated by $\mathcal{D}$
- A returns a hypothesis $h \in \mathcal{H}$ such that with probability at least $1 - \delta$

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon$$

Note that this general formulation is due to Vapnik and Chervonenkis (1971, much earlier than Valiant). Vapnik is the founder of statistical learning theory based on structural risk minimization rather than empirical risk minimization

Concluding Remarks

Now we know what PAC learning is

- it is basically a framework that tells you when you can expect reasonable results from a learning algorithm
- that is, it tells you when you can learn

So, the big question is:

which hypothesis classes are PAC learnable?

Next time we will prove that finite hypothesis classes are also PAC learnable in this more general setting

- the realizability assumption is not necessary for finite classes

Moreover, we will make a start with a precise characterization of all PAC learnable hypothesis classes

- by introducing the VC dimension of a class of hypotheses.