Part 02 - Computational Learning Theory

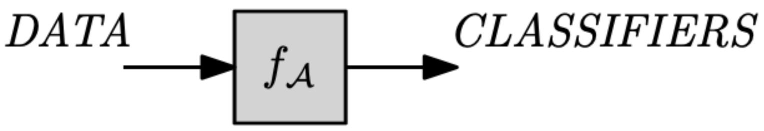
A Glimpse into Computational Learning Theory

@April 4, 2023

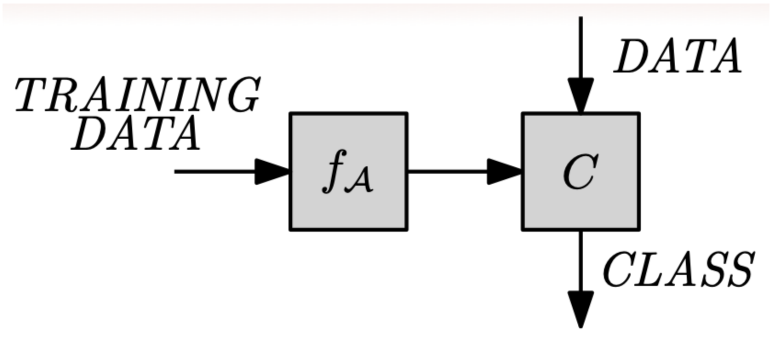
Computational Learning Theory is an attempt to use the tools from Theory of Computation for the study of learning problems (tasks underlying the Machine Learning tools).

For us, a computational task has up to now been a function or a language. But learning problems are computational problems anyway. A learning problem is kind of a two-phased computational task: from training data you get classifiers, and a classifier is an algorithm that solves a different problem.

A learning algorithm $\mathcal{A}$ is an algorithm that computes a function $f_{\mathcal{A}}$ from an input of labeled data and whose output can be seen as a classifier.

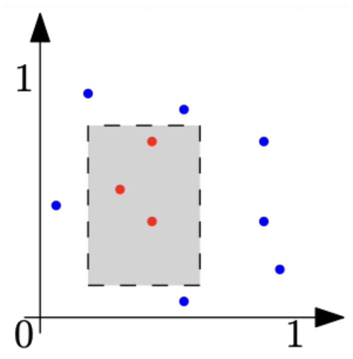


How can we define if $\mathcal{A}$ correctly solves a given learning task? The answer it "to some extent, yes". CLT allows us to obtain a definite answer.



Two-Phased Process: Learning Problem seen as a Computational Process

This is the overall idea. From the labelled data, $f_{\mathcal{A}}$ is capable of understanding the features of the training data that justify the labels. What is C? It's itself a computational task, but is even more appropriate to define it as a process. The output of $f_{\mathcal{A}}$ is a string that is meant to describe an algorithm, typically taken from a restricted set of classifiers (it is not arbitrary).



I have in input points in a cartesian plane $(x, y) \in \mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$, labeled as positive or negative depending on they being inside or outside a rectangle

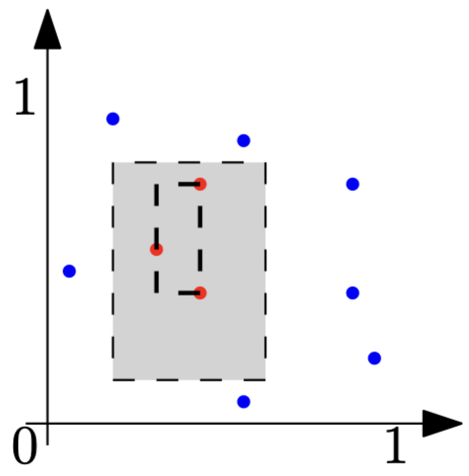
The learning problem is to learn the rectangle (the classifier is the rectangle). So the algorithm $\mathcal{A}$ should be able to guess the rectangle, based on the labelled data received in input. The two classes here are inside and outside. The algorithm $\mathcal{A}$ knows the data are labelled according to a rectangle R, but does not know the rectangle used. It cannot guess the rectangle with perfect accuracy (we cannot achieve perfect accuracy in the finite case). When the data grow in number, we expect, however, that $f_{\mathcal{A}}(D)$ converges to R (in the analytical sense).

$\mathcal{A}$ knows that the input is labeled by way of a rectangle ($\mathcal{A}$ knows the language in which we express the classifier, it knows we're interested in rectangles). We assume that the algorithm does not know how the label data are generated, the distribution D from which the points are generated. Strong requirements: we want of course to get a robust classifier that works well independently of the distribution. $\mathcal{A}$ is an ordinary algorithm: we can ultimately see it as a TM, and thus assume that real numbers can be approximated as binary strings. In some cases, might be good to have a randomized algorithm.

One possible solution, we call it $\mathcal{A}_{BFP}$: an algorithm that from any given sequence of points, each of them labelled $((x_i, y_i), p_i)$, it needs to be able to determine the rectangle $\rightarrow$ produce in output the smallest rectangle R including the positive instances.

In this case, the output would be an inner rectangle, rather different from the target.

Could we say that the algorithm $\mathcal{A}_{BFP}$ doesn't solve the problem? You can't ask the algorithm to return the real rectangle when the cardinality of the training set is poor, you must look for what happens at convergence. The algorithm is correct, we can give a correctness statement. Notion of approximately correct algorithm.



For the given rectangle R and the target rectangle T, the probability of error in using R as a replacement of T, when the distribution is $\mathbf{D}$, is:
$error_{\mathbf{D},T}(R) = \Pr_{x\sim\mathbf{D}}[x \in (R-T) \cup (T-R)]$

It is the probability that R makes an error in classifying. This is not a loss function. The more the two rectangles agree, the lower the probability.

Theorem

For every distribution $\mathbf{D}$, for every $0 < \varepsilon < 1/2$ and for every $0 < \delta < 1/2$, if $m \geq \frac{4}{\varepsilon}\ln(\frac{4}{\delta})$ - quasi-linear -, then

$$\Pr_{D\sim\mathbf{D}^m}[error_{\mathbf{D},T}(\mathcal{A}_{BFP}(T(D)) < \varepsilon] > 1-\delta$$

m is a natural number bounded by a polynomial which is the inverse of both δ and ε, which are two different kinds of error. The probability that the error produced by the algorithm is smaller than ε when you pass to the algorithm training data consisting of m point is greater than $1-\delta$. However you choose both ϵ and δ you have a threshold number of labeled points to obtain informative data.

This is a definition of approximate correctness, it only works at the limit.

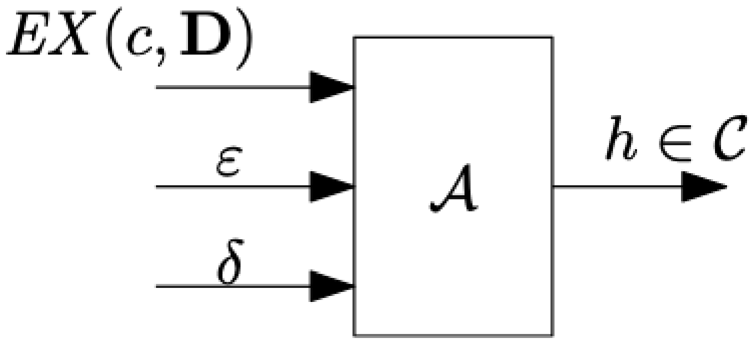
The General Model

We assume to work within an instance space X, i.e. space from which the data is drawn according to a distribution, unknown to the learner - after all, the theorem says that it has to hold for every distribution $\mathbf{D}$.

Concepts are an abstraction of the language in which you write the classifiers (we can call them also models). Subsets of X(they're the classifiers), with no further constraints, thought of as properties of objects (in the example, arbitrary subsets of $X = \mathbb{R}^2_{[0,1]}$).

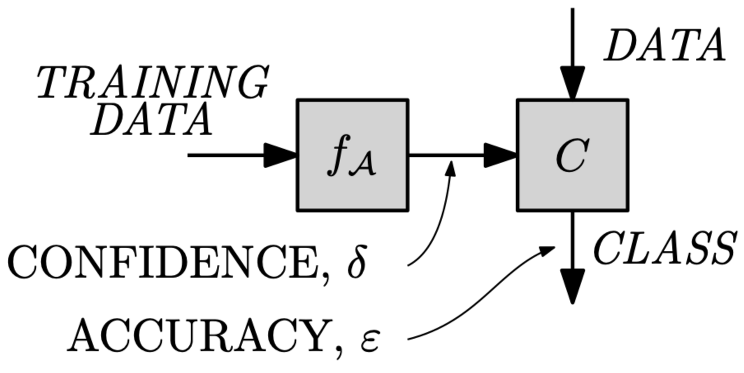
Concept class $\mathcal{C}$ is a collection of concepts (subset of $\mathcal{P}(X)$) in which you consider only some regions, some of the classifiers (e.g. in the example, we implicitly chose the class of all the rectangles). The target concept is an element of $\mathcal{C}$ that the learner wants to build a classifier for (task of the learning algorithm).

So, every learning algorithm is designed to learn concepts from a concept class $\mathcal{C}$, without knowing the target concept $c \in \mathcal{C}$, nor the associated distribution $\mathbf{D}$.



Basic interface for every learning algorithm $\mathcal{A}$

$EX(c, \mathbf{D})$ is a sort of fancy library function that the algorithm call whenever it needs a labelled set of data, it's an "oracle" which returns a correctly labeled (according to c) piece of data. ε is the error parameter/target accuracy, δ is the confidence of the algorithm. The error of any $h \in \mathcal{C}$ is $error_{\mathbf{D},c} = \Pr_{x \sim \mathbf{D}}[h(x) \neq c(x)]$



PAC Concept Classes

Let $\mathcal{C}$ be a concept class (the concept of function is played by the concept class) over the instance space X. We say that $\mathcal{C}$ is PAC-learnable iff there is an algorithm $\mathcal{A}$ s.t. for every $c \in \mathcal{C}$, for every distribution $\mathbf{D}$, for every $0 < \varepsilon < 1/2$ and for every $0 < \delta < 1/2$, then

$$Pr(error_{\mathbf{D},c}(\mathcal{A}(EX(c, \mathbf{D}), \varepsilon, \delta)) < \varepsilon] > 1 - \delta$$

where the error is computed over the outcome of the algorithm when ε, δ, and the oracle $EX(c, \mathbf{D})$ are passed.

Not really what we want, because the algorithm takes high accuracy with high confidence, but could call the oracle an unrealistic amount of times. We talk about efficient PAC learning if the time complexity of $\mathcal{A}$ is bounded by a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$.

Corollary of the Theorem about learning rectangles
The concept-class of axis-aligned rectangle over $\mathbb{R}^2_{[0,1]}$ is efficiently PAC-learnable.

So, the rectangle concept class is good, as it can be learned in a reasonable amount of time.

On the contrary, you cannot learn formulas, but you can learn some classes of Neural Networks.

@May 2, 2023

We cannot realistically ask for the algorithm to work efficiently in polynomial time regardless of the size of the concept class, so we introduce the concept of **Representation Class**, on which we parametrize the concept with the notion of its size. Representation classes are a way to stratify a concept class into concept classes of increasing size/complexity → instill structures, so we can maintain the efficiency (keep polynomial time)

Boolean Functions as Representation Class

Instance class is $X = \bigcup_{n \in \mathbb{N}} X_n$ stratified in bigger and bigger classes of size n, with $X_n = \{0,1\}^n$. First example of a representation class for X_n would be $\mathbf{CL}_n$, class of all conjunctions of literals on the variables $x_1, ..., x_n$. Not all subsets can be captured (the set of vectors which are mapped to one by this conjunction) this way, it is not universal. This, though, is perfectly okay.

Another example on the same X_n is the class of $\mathbf{CNF}_n$, consisting on the conjunctions of disjunctions of literals over $x_1, ..., x_n$. This can capture all subsets (any truth table is sufficient to capture a boolean function and you can always get a CNF from a truth table). This is powerful as you don't have to leave anything behind. You can even consider $k\mathbf{CNF}$, but you would lose universality.

Suppose your target concept is a conjunction of literals c on n variables $x_1, \dots, x_n$, and you have data in the form (s, b), where $s \in \{0,1\}^n$ is a boolean number and $b \in \{0,1\}$ a label that says if the $s \in c$ or not. The idea is to keep track of the over-approximation (which at the beginning includes both positive and negative versions of each variable) and update it using the data we have.

👩🏻‍💻 **Theorem**
The representation class of boolean conjunctions of literals is efficiently PAC-learnable

Intractability of Learning Disjunctive Normal Forms

Conjunctive Normal Forms are actually highly incomplete as a way to represent boolean functions. We then look at Disjunctive Normal Forms. A 3-Term DNF formula over n bits is a propositional formula in the form $T_1 \vee T_2 \vee T_3$, where each T_i is a conjunction of literals over $x_1, \dots, x_n$. This way, you are more expressive, but not universal. Actually good sign against the breaking of some crypto-systems. Machine Learning cannot learn everything.

👩🏻‍💻 **Theorem**
If $\mathbf{NP} \neq \mathbf{RP}$, then the representation class of 3-term DNF formulas is not efficiently PAC-learnable