Languages and Algorithms for Artificial Intelligence (Third Module)

A Glimpse into Computational Learning Theory

Ugo Dal Lago

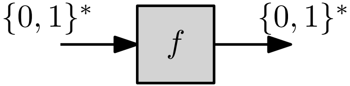
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

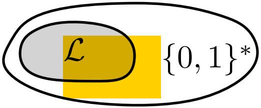
Inria informatiques mathématiques

University of Bologna, Academic Year 2019/2020

Learning Problems as Computational Problems

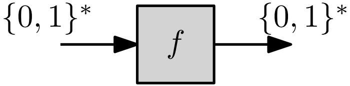
- We have so far taken functions and languages as our notions of **computational tasks**:

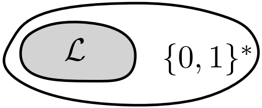




Learning Problems as Computational Problems

- We have so far taken functions and languages as our notions of **computational tasks**:

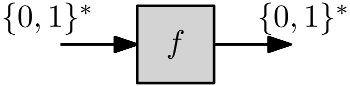


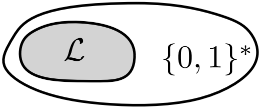


- Is it that **learning problemes**, among the most crucial tasks in AI, can be seen as computational problems?

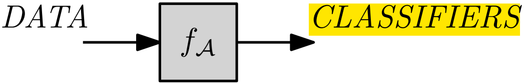
Learning Problems as Computational Problems

- We have so far taken functions and languages as our notions of **computational tasks**:



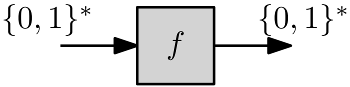


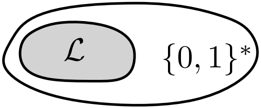
- Is it that **learning problemes**, among the most crucial tasks in AI, can be seen as computational problems?
- The answer is positive: any learning algorithm $\mathcal{A}$ actually computes a function $f_{\mathcal{A}}$ whose input is a finite sequence of *labelled data* and whose output can be seen as a *classifier*:



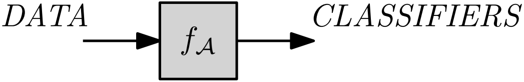
Learning Problems as Computational Problems

- We have so far taken functions and languages as our notions of **computational tasks**:





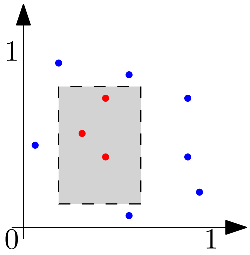
- Is it that **learning problemes**, among the most crucial tasks in AI, can be seen as computational problems?
- The answer is positive: any learning algorithm $\mathcal{A}$ actually computes a function $f_{\mathcal{A}}$ whose input is a finite sequence of *labelled data* and whose output can be seen as a *classifier*:



  - Could data and classifiers be encoded as strings, thus turning $f_{\mathcal{A}}$ as a function of the kind we know?
  - How could we formalize the fact that $\mathcal{A}$ correctly solves a given learning task?

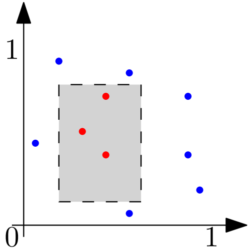
An Example Problem

Suppose that the data the algorithm $\mathcal{A}$ takes in input are points $(x, y) \in \mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$ (where $\mathbb{R}_{[0,1]}$ is the set of real numbers between 0 and 1). These are labelled as positive or negative depending on they being inside a rectangle



An Example Problem

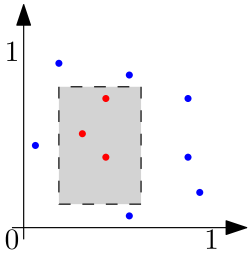
Suppose that the data the algorithm $\mathcal{A}$ takes in input are points $(x, y) \in \mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$ (where $\mathbb{R}_{[0,1]}$ is the set of real numbers between 0 and 1). These are labelled as positive or negative depending on they being inside a rectangle



- The algorithm $\mathcal{A}$ should be able to guess a classifier, namely a *rectangle*, based on the labelled data it received in input.

An Example Problem

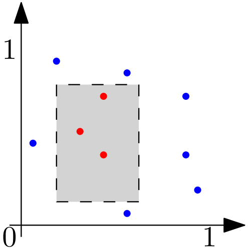
Suppose that the data the algorithm $\mathcal{A}$ takes in input are points $(x, y) \in \mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$ (where $\mathbb{R}_{[0,1]}$ is the set of real numbers between 0 and 1). These are labelled as positive or negative depending on they being inside a rectangle



- The algorithm $\mathcal{A}$ should be able to guess a classifier, namely a *rectangle*, based on the labelled data it received in input.
- It knows that the data are labelled according to a rectangle, R but it does not know *which rectangle* is being used.

An Example Problem

Suppose that the data the algorithm $\mathcal{A}$ takes in input are points $(x, y) \in \mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$ (where $\mathbb{R}_{[0,1]}$ is the set of real numbers between 0 and 1). These are labelled as positive or negative depending on they being inside a rectangle



- The algorithm $\mathcal{A}$ should be able to guess a classifier, namely a *rectangle*, based on the labelled data it received in input.
- It knows that the data are labelled according to a rectangle, R but it does not know *which rectangle* is being used.
- The algorithm $\mathcal{A}$ cannot guess the rectangle R with perfect accuracy if the data it receives in input are too few. As the data in D grow in number, we would expect the rectangle $f_{\mathcal{A}}(D)$ to converge to R, wouldn't we?

The Rules of the Game

- Again, $\mathcal{A}$ *knows* that the the way input data are labelled is by way of a rectangle (whose sides are parallel to the axes).
  - But it *does not know* which one!

The Rules of the Game

- Again, $\mathcal{A}$ *knows* that the the way input data are labelled is by way of a rectangle (whose sides are parallel to the axes).
  - But it *does not know* which one!
- $\mathcal{A}$ *does not know* the distribution $\mathbf{D}$ from which the points (x, y) are drawn.
  - It is supposed to "do the job" for each possible distribution $\mathbf{D}$.

The Rules of the Game

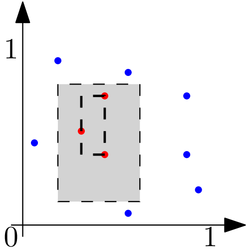
- Again, $\mathcal{A}$ *knows* that the the way input data are labelled is by way of a rectangle (whose sides are parallel to the axes).
  - But it *does not know* which one!
- $\mathcal{A}$ *does not know* the distribution $\mathbf{D}$ from which the points (x, y) are drawn.
  - It is supposed to "do the job" for each possible distribution $\mathbf{D}$.
- $\mathcal{A}$ is an ordinary algorithm.
  - Ultimately, it can be seen as a TM.
  - We thus assume that real numbers can be appropriately approximated as binary strings.
  - In some cases, it is useful to assume $\mathcal{A}$ to have the possibility to "flip a coin", i.e., to be a randomized algorithm.

The Algorithm $\mathcal{A}_{\mathsf{BFP}}$

- We could define an Algorithm $\mathcal{A}_{\mathsf{BFP}}$ as follows:
  1. Given the data $((x_1, y_1), p_1), \ldots, ((x_n, y_n), p_n)$;
  2. Determine the smallest rectangle R including all the positive instances;
  3. Return R.

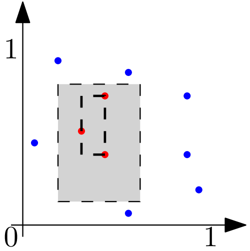
The Algorithm $\mathcal{A}_{\mathsf{BFP}}$

- We could define an Algorithm $\mathcal{A}_{\mathsf{BFP}}$ as follows:
  1. Given the data $((x_1, y_1), p_1), \ldots, ((x_n, y_n), p_n)$;
  2. Determine the smallest rectangle R including all the positive instances;
  3. Return R.
- In the probelm instance from the previous slides, one would get, when running $\mathcal{A}_{\mathsf{BFP}}$, the little bold rectangle. Of course, the result is always a sub-rectangle of the target rectangle.



The Algorithm $\mathcal{A}_{\mathsf{BFP}}$

- We could define an Algorithm $\mathcal{A}_{\mathsf{BFP}}$ as follows:
    1. Given the data $((x_1, y_1), p_1), \ldots, ((x_n, y_n), p_n)$;
    2. Determine the smallest rectangle R including all the positive instances;
    3. Return R.
- In the probelm instance from the previous slides, one would get, when running $\mathcal{A}_{\mathsf{BFP}}$, the little bold rectangle. Of course, the result is always a sub-rectangle of the target rectangle.



- The output classifier is a rectangle, which can be easily represented as a pair of coordinates.

Is $\mathcal{A}_{\mathsf{BFP}}$ (Approximately) Correct?

- The output of $\mathcal{A}_{\mathsf{BFP}}$ can be very different from the target rectangle.
  - Is the algorithm balantantly incorrect, then?

Is $\mathcal{A}_{\mathsf{BFP}}$ (Approximately) Correct?

- The output of $\mathcal{A}_{\mathsf{BFP}}$ can be very different from the target rectangle.
  - Is the algorithm balantantly incorrect, then?
- The answer is **negative**.

Is $\mathcal{A}_{\mathsf{BFP}}$ (Approximately) Correct?

- The output of $\mathcal{A}_{\mathsf{BFP}}$ can be very different from the target rectangle.
  - Is the algorithm balantantly incorrect, then?
- The answer is **negative**.
- For a given rectangle R and a target rectangle T, the *probability of error* in using R as a replacement of T (when the distribution is $\mathbf{D}$) is $\mathit{error}_{\mathbf{D},T}(R) = \Pr_{x \sim \mathbf{D}}[x \in (R - T) \cup (T - R)]$.

Is $\mathcal{A}_{\textsf{BFP}}$ (Approximately) Correct?

- The output of $\mathcal{A}_{\textsf{BFP}}$ can be very different from the target rectangle.
  - Is the algorithm balantantly incorrect, then?
- The answer is **negative**.
- For a given rectangle R and a target rectangle T, the *probability of error* in using R as a replacement of T (when the distribution is $\mathbf{D}$) is $error_{\mathbf{D},T}(R) = \Pr_{x\sim\mathbf{D}}[x \in (R - T) \cup (T - R)]$.
- As the number of samples in D grows, the result $\mathcal{A}_{\textsf{BFP}}(D)$ does *not* necessarily approach the target rectangle, but its probability of error approaches zero.

Theorem

For every distribution $\mathbf{D}$, *for every* $0 < \varepsilon < \frac{1}{2}$ *and for every* $0 < \delta < \frac{1}{2}$, *if* $m \geq \frac{4}{\varepsilon} \ln\left(\frac{4}{\delta}\right)$, *then*

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

The General Model — Terminology

- We assume to work within an **instance space** X.
  - X is the set of (encodings) of instances of objects the learner wants to classify.
  - Data from the instance spaces are generated through a distribution $\mathbf{D}$, unknown to the learner.
  - In the example, $X = \mathbb{R}^2_{[0,1]}$.

The General Model — Terminology

- We assume to work within an **instance space** X.
  - X is the set of (encodings) of instances of objects the learner wants to classify.
  - Data from the instance spaces are generated through a distribution $\mathbf{D}$, unknown to the learner.
  - In the example, $X = \mathbb{R}^2_{[0,1]}$.
- **Concepts** are subsets of X, i.e. collections of objects. These should be thought of as properties of objects.
  - In the example, concepts are arbitrary subsets of $X = \mathbb{R}^2_{[0,1]}$, i.e. arbitrary regions within $\mathbb{R}^2_{[0,1]}$.

The General Model — Terminology

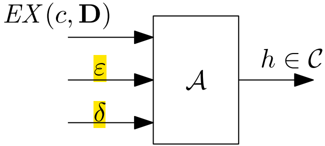
- We assume to work within an **instance space** X.
  - X is the set of (encodings) of instances of objects the learner wants to classify.
  - Data from the instance spaces are generated through a distribution $\mathbf{D}$, unknown to the learner.
  - In the example, $X = \mathbb{R}^2_{[0,1]}$.
- **Concepts** are subsets of X, i.e. collections of objects. These should be thought of as properties of objects.
  - In the example, concepts are arbitrary subsets of $X = \mathbb{R}^2_{[0,1]}$, i.e. arbitrary regions within $\mathbb{R}^2_{[0,1]}$.
- A **concept class** $\mathcal{C}$ is a collection of concepts, namely a subset of $\mathcal{P}(X)$. These are the concepts which are sufficiently simple to describe, and that algorithms can handle.
  - The concept class $\mathcal{C}$ we work with in the example is the one of rectangles whose sides are parallel to the axes.
  - The **target concept** $c \in \mathcal{C}$ is the concept the learner wants to build a classifier for.

The General Model — The Learning Algorithm $\mathcal{A}$

- Every learning algorithm is designed to learn concepts from a concept class $\mathcal{C}$ but it does not know the target concept $c \in \mathcal{C}$, nor the associated distribution $\mathbf{D}$.

The General Model — The Learning Algorithm $\mathcal{A}$

- Every learning algorithm is designed to learn concepts from a concept class $\mathcal{C}$ but it does not know the target concept $c \in \mathcal{C}$, nor the associated distribution $\mathbf{D}$.
- The interface of any learning algorithm $\mathcal{A}$ can be described as follows:

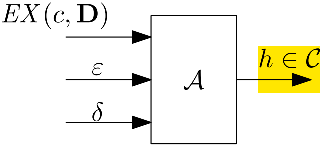


where:

  - ε is **error parameter**, while δ is the **confidence parameter**;
  - $EX(c, \mathbf{D})$ should be though as an *oracle*, a procedure that $\mathcal{A}$ can call as many times she wants, and which returns an element $x \sim \mathbf{D}$ from X, labelled according to whether it is in c or not.

The General Model — The Learning Algorithm $\mathcal{A}$

- Every learning algorithm is designed to learn concepts from a concept class $\mathcal{C}$ but it does not know the target concept $c \in \mathcal{C}$, nor the associated distribution $\mathbf{D}$.
- The interface of any learning algorithm $\mathcal{A}$ can be described as follows:



  where:
  - ε is **error parameter**, while δ is the **confidence parameter**;
  - $EX(c, \mathbf{D})$ should be though as an *oracle*, a procedure that $\mathcal{A}$ can call as many times she wants, and which returns an element $x \sim \mathbf{D}$ from X, labelled according to whether it is in c or not.
- The **error** of any $h \in \mathcal{C}$ is defined as $error_{\mathbf{D},c} = \Pr_{x \sim \mathbf{D}}[h(x) \neq c(x)]$.

The General Model — PAC Concept Classes

- Let $\mathcal{C}$ be a concept class over the instance space X. We say that $\mathcal{C}$ is **PAC learnable** iff there is an algorithm $\mathcal{A}$ such that for every $c \in \mathcal{C}$, for every distribution $\mathbf{D}$, for every $0 < \varepsilon < \frac{1}{2}$ and for every $0 < \delta < \frac{1}{2}$, then

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

  where the probability is taken over the calls to $EX(c,\mathbf{D})$

The General Model — PAC Concept Classes

- Let $\mathcal{C}$ be a concept class over the instance space X. We say that $\mathcal{C}$ is **PAC learnable** iff there is an algorithm $\mathcal{A}$ such that for every $c \in \mathcal{C}$, for every distribution $\mathbf{D}$, for every $0 < \varepsilon < \frac{1}{2}$ and for every $0 < \delta < \frac{1}{2}$, then

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

  where the probability is taken over the calls to $EX(c,\mathbf{D})$
- If the time complexity of $\mathcal{A}$ is bounded by a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$, we say that $\mathcal{C}$ is **efficiently PAC learnable**.
  - The complexity of $\mathcal{A}$ is measured taking into account the number of calls to $EX(c,\mathbf{D})$.

- Let $\mathcal{C}$ be a concept class over the instance space X. We say that $\mathcal{C}$ is **PAC learnable** iff there is an algorithm $\mathcal{A}$ such that for every $c \in \mathcal{C}$, for every distribution $\mathbf{D}$, for every $0 < \varepsilon < \frac{1}{2}$ and for every $0 < \delta < \frac{1}{2}$, then

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

  where the probability is taken over the calls to $EX(c,\mathbf{D})$
- If the time complexity of $\mathcal{A}$ is bounded by a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$, we say that $\mathcal{C}$ is **efficiently PAC learnable**.
  - The complexity of $\mathcal{A}$ is measured taking into account the number of calls to $EX(c,\mathbf{D})$.

Corollary

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

Thank You!

Questions?