Languages and Algorithms for Artificial Intelligence

Module 3

Matteo Donati

September 11, 2021

Contents

Chapter 1

Introduction

Nowadays artificial intelligence and machine learning are everywhere. This module will try to present intrinsic limits of artificial intelligence and what it is possible to do, in terms of accuracy and efficiency, with algorithms which solve a given problem. In particular, this module will be structured as follows:

- Historical, conceptual and mathematical preliminaries.
- The computational model.
- The P class.
- The NP class and NP-completeness.
- Optimization problems and their hardness.

Chapter 2

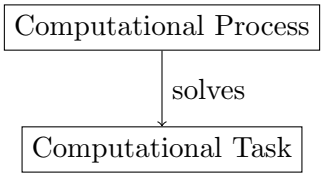
Historical, Conceptual and Mathematical Preliminaries

2.1 Historical Preliminaries

Since the pioneering works by Hartmanis, Stearns and Cobham (all from the late 60s), a new branch of theory of computation (ToC), which deals with efficiency, has emerged. This new branch of ToC has been named **computational complexity theory**.

2.2 Conceptual Preliminaries

Considering the following figure:



- Processes and tasks are different.
- In some cases, there could be many distinct processes solving the same task.
- In some other cases, the task is so hard that no process can solve it.

In particular:

- In the ToC, a process is taken to be an algorithm.
- Any algorithm must satisfy the following constraints:

  - It must be a finite description of a series of computation steps.
  - Each step must be elementary.
  - The way the next step is determined must be deterministic.

- Any program can be seen as a description of an algorithm.
- In order to classify tasks, it is necessary to prove that tasks are not solvable, beyond a certain level of efficiency, by processes. The problem is that, there exist infinitely many processes solving a certain task, so one cannot exhaustively examine them (i.e. one is very rarely able to prove the non-existence of an efficient algorithm). Rather than proving the non-existence of certain algorithms, complexity theory interrelates different tasks.

2.3 Mathematical Preliminaries

Below are listed some useful mathematical notions which will be used in this module:

- The cardinality of any set X is indicated as $|X|$ and it can be either finite or infinite.
- $\mathbb{N}$ is the set of natural numbers, while $\mathbb{Z}$ is the set of integers.
- A condition $P(n)$ depending on $n \in \mathbb{N}$ holds for sufficiently large n is there is $N \in \mathbb{N}$ such that $P(n)$ holds for every $n > N$.
- For a given real number x, $\lceil x \rceil$ is the smallest element of $\mathbb{Z}$ such that $\lceil x \rceil > x$. Whenever a real number x is in use in place of a natural number, $\lceil x \rceil$ in considered.
- For a natural number n, $[n]$ is the set $\{1, \ldots, n\}$.
- $\log x$ is the base 2 logarithm of x.
- If S is a finite set, then a string over the alphabet S is a finite, ordered, possibly empty, tuple of elements from S. Most often $S = \{0, 1\}$.
- The set of all strings over S of length exactly $n \in \mathbb{N}$ is indicated as S^n (where S^0 is the set containing only the empty string ϵ). – e.g. $\{0,1\}^2 = \{00, 01, 10, 11\}$
- The set of all strings over S is $\bigcup_{n=0}^{\infty} S^n$ and is indicated as S^*. Any subset of S^* is usually called a **language**.
- The concatenation of two strings x and y over S is indicated as xy. The string over S obtained by concatenating x with itself $k \in \mathbb{N}$ times is indicated as x^k.
- The length of a string x is indicated as $|x|$.
- Any task of interest consists in computing a function, $f : A \to B$, from $\{0, 1\}^*$ to itself. If the input and/or the output of a given function are not strings, but they are taken from a discrete set, they can be represented as strings, following some encoding.

- The encoding of any element x of A as a string is often indicated as $\llcorner x \lrcorner$ or simply as x.
- An important class of functions from $\{0,1\}^*$ to $\{0,1\}^*$ are those whose range are strings of length exactly equal to one. These are called **boolean functions** and are identified with the subset $\mathcal{L}_f$ of $\{0,1\}^*$ defined as follows: └ such functions return only a zero or a one

$$\text{language} \quad \mathcal{L}_f = \{x \in \{0,1\}^* | f(x) = 1\} \tag{2.1}$$

  Moreover, a **decision problem** for a given language $\mathcal{M}$ (i.e. ~~does~~ is $x \in \{0,1\}^*$ ~~is~~ in $\mathcal{M}$?) can be seen as the task of computing f such that $\mathcal{M} = \mathcal{L}_f$.
- A function $f : \mathbb{N} \to \mathbb{N}$ is $O(g)$ is there is a positive real constant c such that $f(n) \leq c \cdot g(n)$ for sufficiently large n.
- A function $f : \mathbb{N} \to \mathbb{N}$ is $\Omega(g)$ is there is a positive real constant c such that $f(n) \geq c \cdot g(n)$ for sufficiently large n.
- A function $f : \mathbb{N} \to \mathbb{N}$ is $\Theta(g)$ if f is both $O(g)$ and $\Omega(g)$.
- Studying in which relation two functions f and g are can be done by studying the limit of $\frac{f(n)}{g(n)}$ for n tending to infinity.

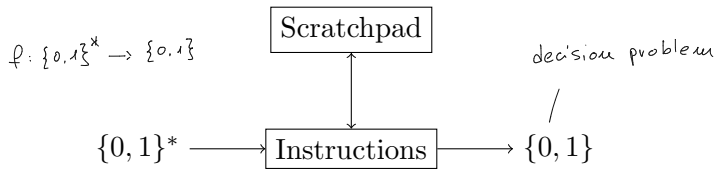
Chapter 3

The Computational Model

There exists a universally accepted model of computation, namely the **Turing Machine**.

3.1 The Model, Informally

Considering the following figure:



- The set of instructions to be followed is fixed and finite, and should work for every input x.
- The same instruction can be potentially used many times.
- Every instruction proceeds by:

  1. Reading a bit of the input.
  2. Reading a symbol from the scratchpad.

  Based on these values the model:

  - Either write the symbol to the scratchpad and proceed to another instruction.
  - Or declare the computation finished by stopping it and outputting either 0 or 1.

- The running time of this machine on x is simply the number of basic instructions which are executed on a certain input x. In particular, the machine runs in time $T(n)$ if it performs at most $T(n)$ instructions on input string of length n.

any string represents a machine

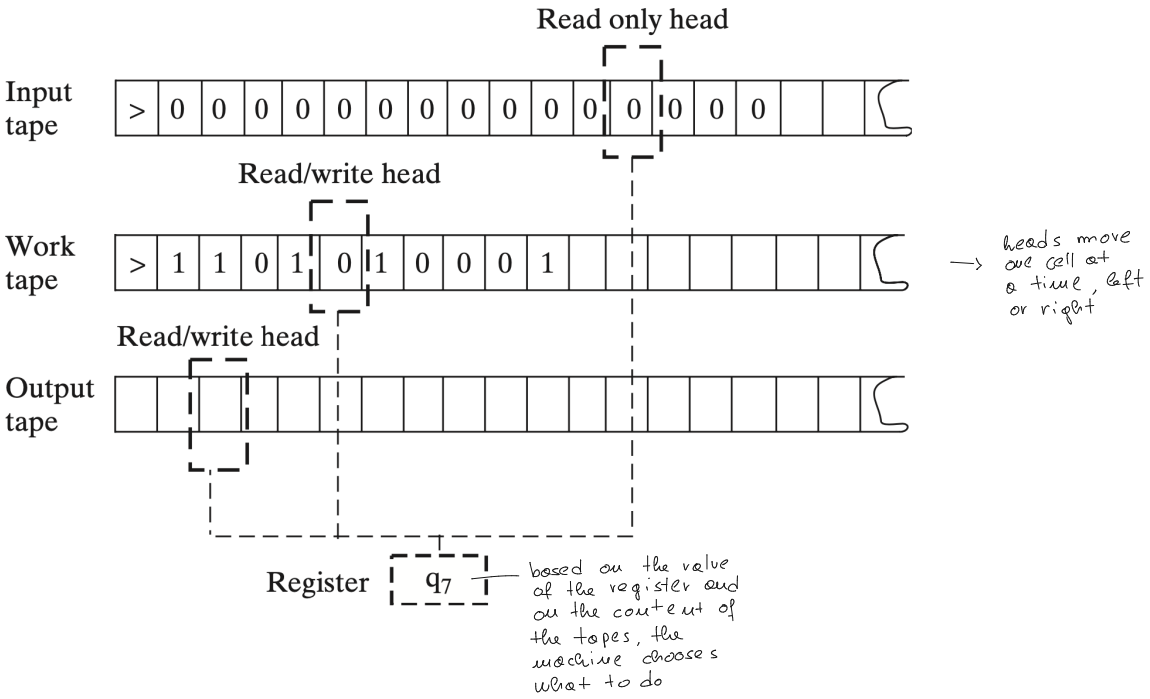
program description

programs can be seen as data

- Since there exist finitely many instructions, machine descriptions can be encoded as binary strings themselves.
- Given a string α, $\mathcal{M}_\alpha$ is the Turing Machine which α encodes. Moreover, there exists a **Universal Turing Machine**, $\mathcal{U}$, which simulates any other Turing Machine given its string representation: from a pair of strings (x, α), the machine $\mathcal{U}$ simulates the behaviour of $\mathcal{M}_\alpha$ on x. This simulation is very efficient: if the running time of $\mathcal{M}_\alpha$ is $T(|x|)$, then $\mathcal{U}$ would take time $O(T(|x|) \log T(|x|))$.

  overhead of simulation, due to the number of instructions of $\mathcal{M}_\alpha$

3.2 The Model, Formally



In particular:

- The scratchpad consists of k **tapes**, where a tape ia an infinite one-directional line of cells, each of which can hold a symbol from a finite alphabet Γ, the alphabet of the machine. In particular, the first tape is designated as **input tape** and is a read-only tape, while the last tape is the **output tape** and contains the result of the computation.
- Each tape is equipped with a **tape head** which can read or write symbols from or to the tape.
- The machine has a finite set of **states**, called Q, which determine the action to be taken at the next step. At each step, the machine:

  see q_7

  1. Reads the symbols under the k tape heads.

2. For the $k-1$ read-write tapes, replaces the symbol with a new one, or leaves it unchanged.
3. Changes its state to a new one.
4. Either moves each of the k tape heads to the left or to the right or keep the heads in place.

More formally, a **Turing Machine** (TM) working on k tapes is described as a triple (Γ, Q, δ) containing:

- A finite set Γ of **tape symbols**, which one assumes that contains the blank symbol $\square$, the start symbol $\triangleright$ and the binary digits 0 and 1.
- A finite set Q of **states** which includes a designated initial state $q_{\texttt{init}}$ and a designated final state $q_{\texttt{halt}}$.
- A transition function δ:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k \tag{3.1}$$

  which describes the instructions regulating the functioning of the machine at each step.

Given a TM, $\mathcal{M} = (\Gamma, Q, \delta)$, working on k tapes:

- A configuration of such machine is indicated as C and consists of:
  - The current state q.
  - The contents of the k tapes.
  - The positions of the k tape heads.
- The initial configuration for the input $x \in \{0,1\}^*$ is the configuration $\mathcal{I}_x$ in which:
  - The current state is $q_{\texttt{init}}$.
  - The first tape contains $\triangleright x$ followed by blank symbols, while the other tapes contain $\triangleright$ followed by blank symbols.
  - The tape heads are positioned on the first symbol of the k tapes.
- A final configuration for the output $y \in \{0,1\}^*$ is any configuration whose state is $q_{\texttt{halt}}$ and in which the content of the output tape is $\triangleright y$ followed by blank symbols.
- Given any configuration C, the transition function δ determines in a natural way the next configuration D: $C \xrightarrow{\delta} D$.
- $\mathcal{M}$ returns $y \in \{0,1\}^*$ on input $x \in \{0,1\}^*$ in t steps if:

$$\mathcal{I}_x \xrightarrow{\delta} C_1 \xrightarrow{\delta} C_2 \xrightarrow{\delta} \ldots \xrightarrow{\delta} C_t \tag{3.2}$$

  where C_t is a final configuration for y. One can write $\mathcal{M}(x)$ for y if this condition holds.

- $\mathcal{M}$ computes a function $f : \{0,1\}^* \to \{0,1\}^*$ if and only if $\mathcal{M}(x) = f(x)$ for every $x \in \{0,1\}^*$. In this case, f is said to be **computable**.
- $\mathcal{M}$ computes f in time $T : \mathbb{N} \to \mathbb{N}$ if and only if $\mathcal{M}$ returns $f(x)$ on input x in a number of steps smaller or equal to $T(|x|)$ for every $x \in \{0,1\}^*$. In this case, f is said to be **computable in time** T.
- A language $\mathcal{L}_f \subseteq \{0,1\}^*$ is **decidable** in time T if and only if f is computable in time T.
- A function $T : \mathbb{N} \to \mathbb{N}$ is **time-constructible** if and only if the function on $\{0,1\}^*$, defined as $x \to \llcorner T(|x|) \lrcorner$, is computable in time $O(T(|x|))$.

3.2.1 The Universal Turing Machine

There exists a Turing Machine $\mathcal{U}$ such that for every $x, \alpha \in \{0,1\}^*$, it holds that $\mathcal{U}(x,\alpha) = \mathcal{M}_\alpha(x)$, where $\mathcal{M}_\alpha$ denotes the Turing Machine represented by α. Moreover, if $\mathcal{M}_\alpha$ halts on input x within T steps then $\mathcal{U}(x,\alpha)$ halts within $CT\log(T)$ steps, where C is independent of $|x|$ and depends only on $\mathcal{M}_\alpha$.

3.2.2 Uncomputability

There exist a function $uc : \{0,1\}^* \to \{0,1\}^*$ that is not computable by any Turing Machine. For example, if one considers the following function:

$$uc(\alpha) = \begin{cases} 0 & \text{if } \mathcal{M}_\alpha(\alpha) = 1 \\ 1 & \text{otherwise} \end{cases} \tag{3.3}$$

if uc were computable, there would exist a Turing Machine $\mathcal{M}$ such that $\mathcal{M}(\alpha) = uc(\alpha)$ for every α, and in particular when $\alpha = \llcorner \mathcal{M} \lrcorner$. This would be a contradiction, because by definition:

$$uc(\llcorner \mathcal{M} \lrcorner) = 1 \Leftrightarrow \mathcal{M}(\llcorner \mathcal{M} \lrcorner) \neq 1 \Leftrightarrow uc(\llcorner \mathcal{M} \lrcorner) = 0$$

Considering another function, *halt*, defined as follows:

$$halt(\llcorner (\alpha, x) \lrcorner) = \begin{cases} 1 & \text{if } \mathcal{M}_\alpha \text{ halts on input } x \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

being able to compute such function would mean being able to check algorithms for termination. In particular, the function *halt* is not computable by any Turing Machine. More in general, one can prove that a problem under consideration, $\mathcal{L}$, is at least as hard as a problem $\mathcal{G}$, one already knows to be undecidable. This is done by showing that there exists a computable way ϕ of turning strings in $\{0,1\}^*$ into strings in $\{0,1\}^*$ in such a way that:

$$s \in \mathcal{G} \Leftrightarrow \phi(s) \in \mathcal{L} \tag{3.5}$$

By doing so, any hypothetical algorithm for $\mathcal{L}$ would be turned into one for $\mathcal{G}$, which, however, cannot exist.

Chapter 4

Polynomial Time Computable Problems

A **complexity class** is a set of tasks which can be computed within some prescribed resource bounds. Typically, the tasks one is interested in are decision problems, or equivalently languages (i.e. subsets of $\{0,1\}^*$). In particular, considering the function $T : \mathbb{N} \to \mathbb{N}$:

- A language $\mathcal{L}$ is in the class $\mathbf{DTIME}(T(n))$ if and only if there is a Turing Machine deciding $\mathcal{L}$ and running in time $n \mapsto c \cdot T(n)$ for come constant c.
- The letter "D" in $\mathbf{DTIME}(\cdot)$ refers to determinism: the machines on which the class is based work deterministically.

4.1 The Class P

The class **P** is defined as follows:

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c) \tag{4.1}$$

In other words, the class **P** includes all those languages $\mathcal{L}$ which can be decided by a Turing Machine working in time P, where P is a polynomial. In particular, **P** is considered as the class of efficiently decidable languages.

4.2 The Class FP

Sometimes, one would like to classify functions rather than languages. Considering the function $T : \mathbb{N} \to \mathbb{N}$:

- A function f is in the class $\mathbf{FDTIME}(T(n))$ if and only if there is a Turing Machine computing f and running in time $n \mapsto c \cdot T(n)$ for some constant c.

The class **FP** is defined as follows:

$$\mathbf{FP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(n^c) \tag{4.2}$$

In particular:

- For every $\mathcal{L} \in \mathbf{P}$, the characteristic function f of $\mathcal{L}$ is trivially in **FP**.
- However, it is not true that $f \in \mathbf{FP}$ implies $\mathcal{L}_f \in \mathbf{P}$.

4.3 The Classes EXP and FEXP

The classes **EXP** and **FEXP** are defined as follows:

class of languages — $$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \tag{4.3}$$

class of functions — $$\mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c}) \tag{4.4}$$

In particular:

- The tasks in these classes can be solved mechanically, but possibly cannot be solved efficiently.
- $\mathbf{P} \subseteq \mathbf{EXP}$, where the inclusion is strict. — if one proves that something is in P, then this something is also in EXP.
- $\mathbf{FP} \subseteq \mathbf{FEXP}$, where the inclusion is strict.

Chapter 5

Between the Feasible and the Unfeasible

Between **P** and **EXP** one can define many other classes, i.e. there are many ways of defining a class **A** such that:

$$\mathbf{P} \subseteq \mathbf{A} \subseteq \mathbf{EXP}$$

Very often, the language one would like to classify can be written as follows:

$$\mathcal{L} = \{x \in \{0,1\}^* | \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in \mathcal{A}\} \quad (5.1)$$

y certificates that x is in $\mathcal{L}$

where $p : \mathbb{N} \to \mathbb{N}$ and $\mathcal{A}$ is a set of pairs of strings. In other words, the elements of $\mathcal{L}$ are those strings for which one can find a "certificate" y (of polynomial length) such that the pari (x, y) passes the test $\mathcal{A} \subseteq \{0,1\}^* \times \{0,1\}^*$. The fact that $\mathcal{A}$ is decidable in polynomial time does not imply that also $\mathcal{L}$ is decidable in polynomial time. For example, given a certain string x, one can check whether $x \in \mathcal{L}$ by checking whether $(x, y) \in \mathcal{A}$ for all possible y such that $|y| \leq p(|x|)$, of which, however, there are exponentially many (i.e. given x, $\mathcal{L}$ is decidable in exponential time). Thus, crafting a solution for the problem (i.e. finding y) can potentially be more difficult than just checking y to be a solution to x. — e.g. sudoku problems

5.1 The Class NP

A language $\mathcal{L} \subseteq \{0,1\}^*$ is in the class **NP** if and only if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time Turing Machine $\mathcal{M}$ such that:

algorithm to verify the certificate on the given input

$$\mathcal{L} = \{x \in \{0,1\}^* | \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\} \quad (5.2)$$

In particular:

- $\mathcal{M}$ is said to be the **verifier** for $\mathcal{L}$.
- Any $y \in \{0,1\}^{p(|x|)}$ such that $\mathcal{M}(\llcorner x, y \lrcorner) = 1$ is said to be a **certificate** for x.

- $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$.

5.1.1 Nondeterministic Turing Machines

The class **NP** can also be defined using a veriant of Turing Machines, called **nondeterministic Turing Machines** (NDTM). In particular, a nondeterministic Turing Machine $\mathcal{M}$:

- Has two transition functions, δ_0 and δ_1 rather than just one. At every step, the machine chooses in a nondeterministic way one between the two transition functions and proceed according to it.
- Has a special state $q_{\texttt{accept}}$.
- Accepts the input $x \in \{0,1\}^*$ if and only if there exists one among the many possible evolutions of the machine $\mathcal{M}$, when fed with x, which makes it reaching $q_{\texttt{accept}}$.
- Rejects the input $x \in \{0,1\}^*$ if and only if none of the aforementioned evolutions leads to $q_{\texttt{accept}}$.
- Runs in time $T : \mathbb{N} \to \mathbb{N}$ if and only if, for every $x \in \{0,1\}^*$ and for every possible nondeterministic evolution, $\mathcal{M}$ reaches either $q_{\texttt{halt}}$ or $q_{\texttt{accept}}$ within $c \cdot T(|x|)$ steps, where $c > 0$.

Moreover:

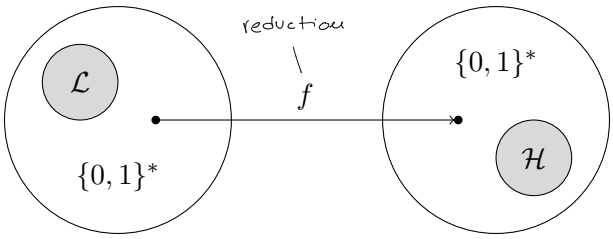
- For every function $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L} \subseteq \{0,1\}^*$, one can say that $\mathcal{L} \in \mathbf{NDTIME}(T(n))$ if and only if there is a nondeterministic Turing Machine $\mathcal{M}$ working in time T and such that $\mathcal{M}(x) = 1$ if and only if $x \in \mathcal{L}$.

Another equivalent definition of the **NP** class is the following:

$$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NDTIME}(n^c) \tag{5.3}$$

5.2 Reductions

Are all the problems in **NP** equivalent? In order to respond to this question one needs to define a pre-order relation between languages. Given two different languages being in relation, the pre-order function allows one to establish the relative difficulty of deciding these two languages. In particular, considering the following figure:



* If $\mathcal{L}$ is NP-hard and $\mathcal{L} \in P$, then $P = NP$.
If $\mathcal{L}$ is NP-complete, then $\mathcal{L} \in P$ iff $P = NP$.

† transform an instance of $\mathcal{L}$ into an instance of $\mathcal{H}$

$\mathcal{L}$ not more difficult than $\mathcal{H}$

The language $\mathcal{L}$ is said to be **polynomial-time reducible** to another language $\mathcal{H}$ ($\mathcal{L} \leq_p \mathcal{H}$) if and only if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $x \in \mathcal{L}$ if and only if $f(x) \in \mathcal{H}$. Moreover:

- If $\mathcal{L} \leq_p \mathcal{H}$, then $\mathcal{H}$ is at least as difficult as $\mathcal{L}$.
- A language $\mathcal{H} \subseteq \{0,1\}^*$ is said to be:
  - **NP**-hard if $\mathcal{L} \leq_p \mathcal{H}$ for every $\mathcal{L} \in \mathbf{NP}$. (even if HALT is undecidable, HALT is NP-hard, but is not NP-complete (not in NP))
  - **NP**-complete if $\mathcal{H}$ is **NP**-hard and $\mathcal{H} \in \mathbf{NP}$. An example of **NP**-complete language is the following:

*

$$\texttt{SAT} = \{\llcorner F \lrcorner | F \text{ is a satisfiable CNF}\} \tag{5.4}$$

where F is a propositional logic formula in conjunctive normal form.

*2

5.2.1 Proving a Problem NP-complete

If one wants to prove $\mathcal{L}$ to be **NP**-complete, one has to prove two statements:

- That $\mathcal{L}$ is in **NP**. This is done by showing that there are p polynomial and $\mathcal{M}$ polytime Turing Machine such that $\mathcal{L}$ can be written as: (looking for a certificate and verifier)

$$\mathcal{L} = \{x \in \{0,1\}^* | \exists y \in \{0,1\}^{p(|x|)}.\mathcal{M}(\llcorner x, y \lrcorner) = 1\}$$

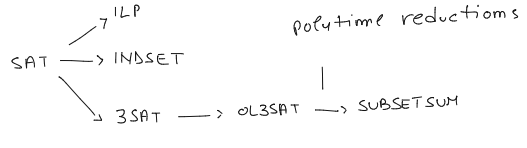
- That any other language $\mathcal{H} \in \mathbf{NP}$ is such that $\mathcal{H} \leq_p \mathcal{L}$. This is usually don by first proving that $\mathcal{J} \leq_p \mathcal{L}$ for a language $\mathcal{J}$ which is already known to be **NP**-complete. This process is correct, simply because $\leq_p$ is transitive (i.e. $\mathcal{H} \xrightarrow{\leq_p} \mathcal{J} \xrightarrow{\leq_p} \mathcal{L}$).

*2

- TMSAT = $\{(d, x, 1^m, 1^t) \mid \exists u \in \{0,1\}^m$ M_d outputs 1 on input (x,u) within t steps$\}$ is NP-complete.
- 3SAT = $\{\llcorner F \lrcorner \mid F$ is a satisfiable 3CNF$\}$ is NP-complete.
  (3CNF: CNF whose clauses contains at most 3 literals)

- INDSET is NP-complete. There is a polytime reduction from SAT to INDSET.
- SUBSETSUM is ". There is a polytime reduction from 3SAT to OL3SAT, and from OL3SAT to SUBSETSUM.
  ILP is ". There is a polytime reduction from SAT to ILP.

- The graph of NP-complete Problems
  For any $\mathcal{L}, \mathcal{H}$ of NP-complete problems, one has that $\mathcal{L} \leq_p \mathcal{H}$, $\mathcal{H} \leq_p \mathcal{L}$ (the two are equivalent)



Chapter 6

A Glimpse into Computational Learning Theory

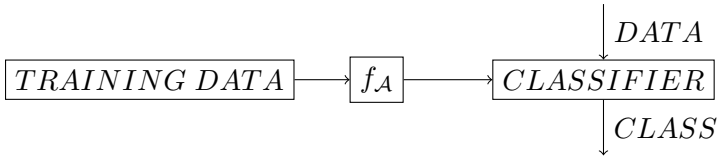
this can be seem as computational problems

Any learning algorithm $\mathcal{A}$ computes a function $f_{\mathcal{A}}$ whose input is a finite sequence of labelled data and whose output can be seen as a classifier:

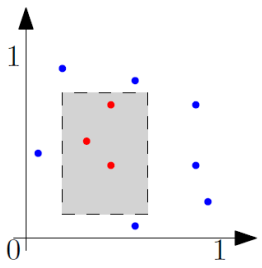
(for example (this can also be a regression problem)

In particular:

- Training data are labelled, so as to let $f_{\mathcal{A}}$ learn what relationship exists between data and labels.
- After the learning process, the classifier C takes as input not labelled data and its task is the one of finding the appropriate label for any of these data.



For example, suppose that the data the algorithm $\mathcal{A}$ takes in input are points $(x, y) \in \mathbb{R}_{[0,1]} \times \mathbb{R}_{[0,1]}$. These are labelled as positive or negative depending on they being inside a rectangle:

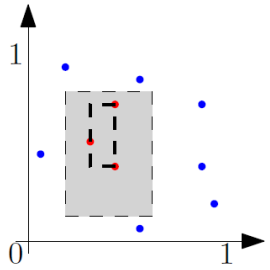


- The algorithm $\mathcal{A}$ should be able to guess a classifier, namely a rectangle, based on the labelled data it received in input.
- It knows that the data are labelled according to a rectangle, R, but it does not know which rectangle is being used.
- It does not known the distribution $\mathbf{D}$ from which the input points (x, y) are drawn.
- As the data in input, D, grows in number, one would expect the rectangle $f_{\mathcal{A}}(D)$ to converge to R.

One could defined an algorithm, $\mathcal{A}_{\text{BFP}}$, 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.

By applying this algorithm to the proposed problem the result would always be a sub-rectangle of the target rectangle:



In particular, 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 $\mathcal{A}_{\text{BFP}}(D)$ does not necessarily approach the target rectangle, but its probability of error approaches zero. As a matter of fact, for every distribution $\mathbf{D}$, for every $0 < \epsilon < \frac{1}{2}$ and for every $0 < \delta < \frac{1}{2}$, if $m \geq \frac{4}{\epsilon} \ln(\frac{4}{\delta})$, then:

$$\Pr_{D \sim \mathbf{D}^m}[error_{\mathbf{D},T}(\mathcal{A}_{\text{BFP}}(T(D))) < \epsilon] > 1 - \delta \tag{6.1}$$

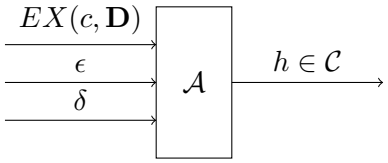
6.1 The General Model

The general model is built upon the following terminology:

- X is the **instance space** and contains the instances of objects the learner wants to classify. In the previous example, $X = \mathbb{R}^2_{[0,1]}$. Data from the instance spaces are generated through a distribution $\mathbf{D}$, unknown to the learner.
- **Concepts** are subsets of X. These should be thought of as properties of objects. In the previous example, concepts are arbitrary regions within $\mathbb{R}^2_{[0,1]}$. any kind of figure in $\mathbb{R}^2_{[0,1]}$, e.g. circles, triangles etc.
- 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. - e.g. the class of rectangles

The learning algorithm $\mathcal{A}$ has the following properties:

- Is designed to learn concepts from a concepts class $\mathcal{C}$ but it does not know the target concept $c \in \mathcal{C}$, nor the associated distribution $\mathbf{D}$.
- Its interface can be described as follows:



  where:

  - ϵ is the **error parameter**.
  - δ is the **confidence parameter**.
  - $EX(c, \mathbf{D})$ should be thought as an oracle, i.e. a procedure that $\mathcal{A}$ can call as many times as wanted, 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:

probabilistic approximated correct

$$error_{\mathbf{D},c} = \Pr_{x \sim \mathbf{D}}[h(x) \neq c(x)] \tag{6.2}$$

6.1.1 PAC Concept Classes

Let $\mathcal{C}$ be a concept class over the instance space X. $\mathcal{C}$ is said to be **PAC learnable** if and only if there is an algorithm $\mathcal{A}$ such that for every $c \in \mathcal{C}$, for every distribution $\mathbf{D}$, for every $0 < \epsilon < \frac{1}{2}$ and for every $0 < \delta < \frac{1}{2}$:

$$\Pr[error_{\mathbf{D},c}(\mathcal{A}(EX(c, \mathbf{D}), \epsilon, \delta)) < \epsilon] > 1 - \delta \tag{6.3}$$

where the probability is taken over the calls to $EX(c, \mathbf{D})$. Moreover, if the time complexity of $\mathcal{A}$ is bounded by a polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$, then $\mathcal{C}$ is **efficiently PAC learnable**.

* Representation class

CHAPTER 2

EXERCISE 1 Prove the undecidability of the halting problem, namely the fact that the function:

$$halt(d, x) = \begin{cases} 1 & \text{if } M_d(x) \text{ terminates} \\ 0 & \text{otherwise} \end{cases}$$

is uncomputable

SOLUTION we show that an hypothetical machine computing the function halt, call it M_{halt}, we can get another Turing machine, M_{UC}, which computes the function UC we know from the slides as uncomputable function, and as a consequence which cannot exist.

Let us now construct M_{UC} out of M_{halt}. On input d, M_{UC} proceeds by calling M_{halt} on input (d,d) and:

- in case M_{halt} returns zero, M_{UC} returns 1
- otherwise, M_{UC} knows that it can safely
  ← call U on input (d,d) and that U will termite its execution on that input, returning an output b.

U is the universal Turing machine

Knowing b, we can proceed by:

- returning 0 if b = 1
- returning 1 otherwise

M_{UC} is indeed a TM correctly computing UC as a function. Since we know that UC

is uncomputable, there is a contradiction.
Therefore the only hypothesis we made, namely the existence of M_{halt} must be false.

□

EXERCISE 2. Show that the function $inc: \mathbb{N} \to \mathbb{N}$ such that $inc(m) = m+1$ can be computed in linear time, by giving an explicit construction of a TM computing inc.

SOLUTION. inc works on the natural numbers, but TMs work on strings, so we have to adopt an encoding of the former into the latter. There is a standard encoding of natural number into strings:

12 -> 1 1 0 0

=> ▷ 1100 $\xmapsto{inc}$ 1101

▷ 1101 $\xmapsto{inc}$ 1110

▷ 1111 $\xmapsto{inc}$ 10000
↳ impossible to obtain

the encoding we use reverses the digits (12 is 0011).
If we use this new encoding, giving a TM with one tape and working in linear time is easy:
it suffices to make one pass from left to right on the input string. The transition function would look like the following:

$$(q_{init}, \triangleright) \xrightarrow{\delta} (q_0, \triangleright, R)$$

$$(q_0, 0) \xrightarrow{\delta} (q_1, 1, L)$$

$$(q_0, 1) \xrightarrow{\delta} (q_0, 0, R)$$

$$(q_0, \square) \xrightarrow{\delta} (q_1, 1, L)$$

$$(q_1, 1) \xrightarrow{\delta} (q_1, 1, L)$$

$$(q_1, 0) \xrightarrow{\delta} (q_1, 0, L)$$

$$(q_1, \triangleright) \xrightarrow{\delta} (q_{halt}, \triangleright, S)$$

$(eg.\ 111 \overset{inc}{\longmapsto} 1000)$

on all the other pairs (q, s) the behaviour of δ is left unspecified and can actually be anything.

EXERCISE 3. (Big O notation) $f(n) = n^2$, $g(n) = 4n + 100 \log(n)$

SOLUTION. $\lim_{n \to \infty} f(n)/g(n) = \infty$, so $f(n) = \Omega(g(n))$

EXERCISE 4. (Big O notation) $f(n) = 2^n n^2$, $g(n) = 3^n$

SOLUTION. $\lim_{n \to \infty} f(n)/g(n) = \dfrac{2^n n^2}{3^n} = \left(\dfrac{2}{3}\right)^n n^2 = 0$

so $f(n) = O(g(n))$

EXERCISE 5. (Big O notation) $f(n) = \min(n, 10^6)$, find g s.t. $f = \Theta(g)$

SOLUTION. $f = \Theta(1)$ ← g

EXERCISE 6. (Turing Machine) Define a TM which produce 1 if the number of ones is odd, and 0 if it is even:

$$f(w) = \begin{cases} 0 \text{ if } |w|_1 \text{ even} \\ 1 \text{ if } |w|_1 \text{ odd} \end{cases}$$

SOLUTION. two tapes are needed (input and output)

$\Gamma = \{0, 1, \square, \triangleright\}$

$Q = \{q_{init}, q_{halt}, q_0, q_1\}$ ← input tape, output tape

$\delta(q_{init}, (\triangleright, \triangleright)) = (q_0, \triangleright, (R, R))$

$\delta(q_0, (0, \square)) = (q_0, \square, (R, S))$

$\delta(q_0, (1, \square)) = (q_1, \square, (R, S))$

$\delta(q_0, (\square, \square)) = (q_{halt}, 0, (S, S))$

same for $\delta(q_1, \dots)$

$\delta(q_{halt}, (\sigma_1, \sigma_2)) = (q_{halt}, \sigma_2, (S, S))$ ← given by the definition of Turing Machine

EXERCISE 7. (Turing Machine)

$$f(w) = \begin{cases} 1 & \text{if } w \text{ is a polymdrome} \\ 0 & \text{otherwise} \end{cases}$$

SOLUTION. using a TM with three tapes (input, work, output).
The Machine will copy the input to the work tape, then the input head is moved to the far left. Lastly one can move simultaneously from left-right (input) and from right-left (work). If one sees two different values, one stops and returns zero, otherwise one will return one.

$\Gamma = \{0, 1, \square, \triangleright\}$

$Q = \{q_{init}, q_{halt}, q_{copy}, q_{left}, q_{test}\}$

$\delta(q_{init}, (\triangleright, \triangleright, \triangleright)) = (q_{copy}, (\triangleright, \triangleright), (R, R, R))$

$\delta(q_{copy}, (\sigma, \square, \square)) = (q_{copy}, (\sigma, \square), (R, R, S))$

$\sigma \in \{0, 1\}$

$\delta(q_{copy}, (\square, \square, \square)) = (q_{left}, (\square, \square), (L, S, S))$

$\delta(q_{left}, (\sigma, \square, \square)) = (q_{left}, (\square, \square), (L, S, S))$

$\delta(q_{left}, (\triangleright, \square, \square)) = (q_{test}, (\square, \square), (R, L, S))$

$\delta(q_{test}, (\sigma, \sigma, \square)) = (q_{test}, (\sigma, \square), (R, L, S))$

$\delta(q_{test}, (\sigma, \sigma', \square)) = (q_{halt}, (\sigma', 0), (S, S, S))$

$\delta(q_{test}, (\square, \triangleright, \square)) = (q_{halt}, (\triangleright, 1), (S, S, S))$

$\delta(q_{halt}, \ldots) = \ldots$

This TM takes $\underbrace{1}_{\text{initial step}} + \underbrace{m+1}_{\text{copy}} + \underbrace{m+1}_{\text{go left}} + \underbrace{m+1}_{\text{test}}$ steps to compute $f(w)$.

$$\Downarrow$$

$$T(m) = 3m + 4$$

Proof of termination: ...

EXERCISE 8. (Turing Machines) compute the mirror function:

$$mirror(w_1 \cdots w_m) = w_m \cdots w_1$$

SOLUTION

1. Go to the far right of input
2. Go back to left and copy into the output meanwhile
3. Halt when "Δ" is encountered

EXERCISE 9. (Decidability) Rice theorem

Any decidable and semantic language $\mathcal{L}$ is trivial ($\mathcal{L} = \phi$ or $\mathcal{L} = \{0,1\}^*$) — Rice theorem

$\Updownarrow$

If $\mathcal{L}$ is semantic and non-trivial, the $\mathcal{L}$ is undecidable.
(no TM can solve the problem)

Determine if $\mathcal{L} = \{ \lfloor M \rfloor \mid M(111) = 111 \}$ is semantic.

SOLUTION. - $\mathcal{L}$ is a language of encodings
- If M and N compute the same function } $\mathcal{L}$ is semantic

$\mathcal{L}$ is non-trivial, so $\mathcal{L}$ is undecidable.

CHAPTER 3

EXERCISE 10. Prove that function MINMAX which given the encoding of a list of natural numbers returns both the min and the max between $a_1, \dots, a_n$ in FP.

SOLUTION. We can describe a solution to the problem by way of pseudo-code:

INPUT: $(a_1, \dots, a_2)$, $a_i \in \mathbb{N}$, appropriately encoded
OUTPUT: (b, c), b is the min and c is the max

```
min ← a1                          ]
max ← a1                          ] 2 instructions
for i ← 2 to n do                 ] O(n) instructions
    if a_i < min then             ]
        min ← a_i                 ]
    if a_i > max then             ] at most 4 instructions
        max ← a_i                 ]
return (min, max)                 ] 1 instruction
```

- since we exaustively look for the min and the max, all the elements of the input list are considered, and the algorithm is thus correct.
  About the total number of instructions, they are:
  $3 + O(n) \cdot 4 \in O(n)$.
- About the size of intermediate results, besides the input itself. we just need the auxiliary min and max variables,

both of them store a sub-string of the input, so they are linear on the size of the input.

- Each instruction can itself be executed in polynomial time, since there are only:
  - assignments
  - comparisons between natural numbers

  } polynomial time

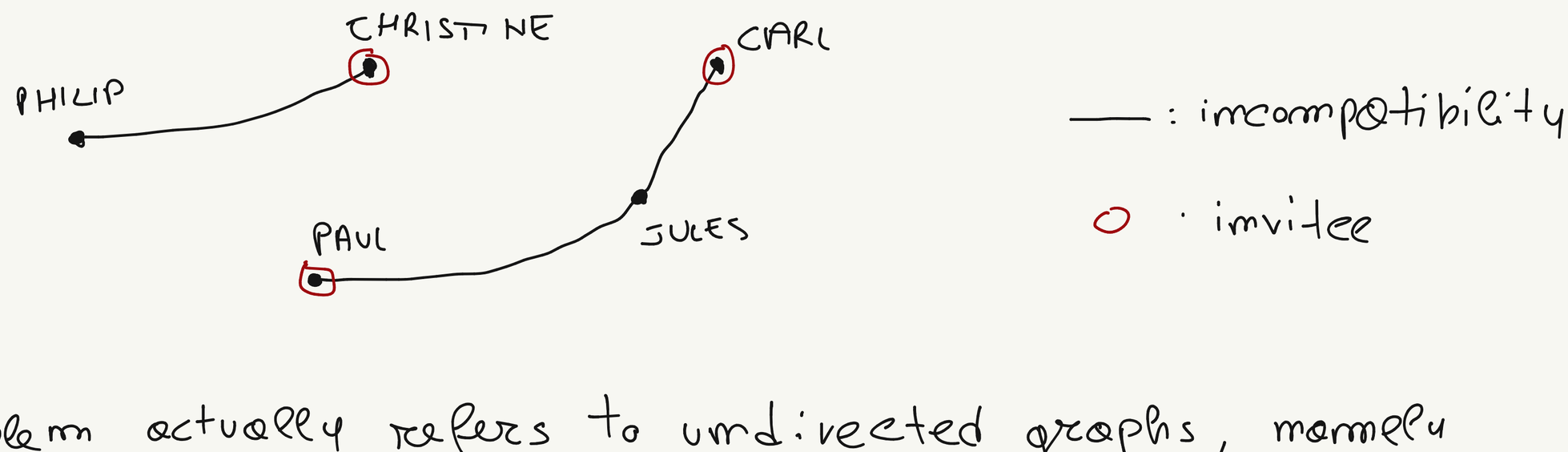
Therefore, MINMAX $\in$ FP.

EXERCISE 11. Prove that the <u>wedding seat problem</u> (see slides) is in the class FEXP.

↓

Given a list of invitees, and a list of incompatibilities constraints one wants to build a sub-list of invitees which are compatible with each other, and which has maximal length.

SOLUTION. We can see this problem as a problem on graphs:



The problem actually refers to undirected graphs, namely pairs (V, E), V set of nodes and E is a subset of: $\{\{c, m\} \mid c, m \in V, c \neq m\}$.

If an undirected graph $\phi = (V, E)$, if $v \in V$, then $N(v)$ is the subset of V defined as $N(v) := \{w \mid \{v, w\} \in E\}$

the problem can be spelled out as follows: given graph $G = (V, E)$ determine a subset W of V s.t.:

A. $\forall v \in W.\ N(v) \cap W = \emptyset$, where N is the neighbour function

B. $|W|$ is maximum among that of all subsets satisfying A
(cardinality)

The function we want to prove in FEXP, is as follows:

FINDSET: GRAPHS $\longrightarrow$ FINITE-SETS

The algorithm we are looking for looks as follows:

INPUT: graph $G = (V, E)$

OUTPUT: $W \subseteq V$ satisfying A and B above

```
Z ← ϕ                                          ] 1 instruction
foreach W ⊆ V do                               ] O(2^|V|)   ∗
    ind ← True                                 ] 1          ∗
    foreach w ∈ W do                           ] O(|V|)     ∗
        if N(w) ∩ W ≠ 0 then                   ]
            ind ← False                        ] ≤ 2        ∗
    if ind then                                ]
        if |Z| < |W| then                      ] ≤ 3        ∗
            Z ← W                              ]
return Z                                       ] 1          ∗
```

- correctness is evident: all possible solutions are considered.
- The number of instructions is: $z + O(z^{|v|}) \cdot (4 + O(|v|) \cdot z) \in O(z^{|v|}) \cdot O(|v|) \in O(z^{|v|} \cdot |v|)$.
- The size of intermidiate results, one can say that:
  - ind is constant-size
  - z, w are bitstrings of linear length (lenght $|v|$)
- All the instructions, except for the computation of $N(w) \cap w$, trivially take polynomial time (assignments, comparisons). Computing $N(w)$ requires just linear time in $|v| + |E|$, while computing $\cap$ can be seen as the process of computing the bitwise-and between two binary strings (linear time)

Therefore, FINDSET $\in$ FEXP, because the number of instructions it executes is $O(z^{|v|} \cdot |v|) \in O(z^{|v|} \cdot z^{|v|}) \in O(z^{z|v|}) \in O(z^{|v|^2}) \in O(\underbrace{z^{m^2}}_{z^{m^2}})$

precisely the bound of FEXP

EXERCISE 12. Prove that $P \subsetneq EXP$ (strictly included)

SOLUTION. The fact that $P \subseteq EXP$ is trivial, because any machine working in time m^c work also in time 2^{m^c}

The challenge is to prove that there is one problem, $\mathcal{L}$, s.t. $\mathcal{L} \in EXP$ and $\mathcal{L} \notin P$.

$\mathcal{L}$ is described by means of the following definition:

if $f, g : \mathbb{N} \to \mathbb{N}$, then $f(n) \in o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

(o: small o)

for example:

$m \in O(m^2)$, $m \in o(m^2)$

$m^3 \in O(2^m)$, $m^3 \in o(2^m)$

$3m \in O(\frac{1}{4}m)$, $3m \notin o(\frac{1}{4}m)$

Hierarchy Theorem

If f, g are time-constructible and $f(m) \log(f(m)) \in o(g(m))$,
then $\mathrm{DTIME}(f(m)) \subsetneq \mathrm{DTIME}(g(m))$

If one has f, g satisfying the hierarchy theorem and s.t.
$P \subseteq \mathrm{DTIME}(f(m)) \subsetneq \mathrm{DTIME}(g(m)) \subseteq EXP$ then $P \subsetneq EXP$.
Indeed, with $f(m) = 2^m$ and $g(m) = 2^{2m}$:

$$f(m)\log(f(m)) = 2^m \log 2^m = m\,2^m \in o\left(2^{2m}\right)$$

and $P \subseteq \mathrm{DTIME}[2^m]$ is trivial, $\mathrm{DTIME}[2^{2m}]$ is also trivial
because $EXP = \bigcup_{c \geq 1} \mathrm{DTIME}[2^{m^c}]$.

EXERCISE 13. $P \subseteq NP \subseteq EXP$ (theorem)

PROOF Let us consider $P \subseteq NP$. We want to prove that if
$\mathcal{L} \in P$, then $\mathcal{L}$ can be written as

$$\mathcal{L} = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)} \,.\, M(x,y) = 1\}$$

Polynomial

We want to find p and M.
(det polytime TM

If $L \in P$, there is nothing to certify and we can take $p(x) = 0$. We can get M out of a polytime TM N which decides L and which exists because $L \in P$. M on input (x, ε) simply returns $N(x)$. Indeed, if we do so, we get that:

$$\begin{aligned} L &= \{ x \in \{0,1\}^* \mid N(x) = 1 \} \\ &\| \\ &= \{ x \in \{0,1\}^* \mid M(x, \varepsilon) = 1 \} \\ &\| \\ &= \{ x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(|x|)} . \; M(x, y) = 1 \} \end{aligned}$$

We now want to prove that $NP \subseteq EXP$. By hypothesis, $L \in NP$ and out of it we want to get $L \in EXP$. From the fact that $L \in NP$ there are a TM M, working in polytime, and a polynomial p s.t.

$$L = \dots$$

out of p, M, we want to obtain an exponential time algo. for L. We can give this algorithm as pseudo-code:

```
INPUT: x ∈ {0,1}*
OUTPUT: element of {0,1}
Foreach y ∈ {0,1}^p(|x|)
    Simulate M(x,y) until it produces result b
    If b = 1 then
        return 1
return 0
```

The fact that this algorithm is correct is self-evident, however we have to check that its complexity is exponentially bounded:

- The for loop is executed at most $2^{p(|x|)}$ times.
- Each iteration takes time $q(|x|)$. ← polynomial

The number of instructions is bounded by $2^{p(|x|)} \cdot q(|x|) \leq 2^{p(|x|)} \cdot 2^{\log(q(|x|))} = 2^{p(|x|)+\log(q(|x|))} \leq 2^{p(|x|)+q(|x|)} \leq 2^{m^c}$ with c big enough.

EXERCISE 14. Suppose CLIQUE is the following.

$$\text{CLIQUE} = \{ \overset{\text{encoding}}{\overset{\downarrow}{\ulcorner}} (G, k) \urcorner \mid G \text{ is an undirected graph containing a clique of size at least } k \}$$

where a clique is a subset $W \subseteq V$ s.t. any pair $v, w \in W$ of distinct nodes is s.t. $\{v, w\} \in E$.

Prove that CLIQUE is NP-complete by showing that

$$3SAT \leq_p \text{CLIQUE}$$

SOLUTION. About containment in NP:

- A certificate is an encoding of a subset W of V s.t. W is CLIQUE of cardinality at least k.
- The verifier can be easily built s.t. its runtime is polynomially bounded.

INPUT: (G, k), W

If $|w| < k$

return 0

else → polynomially bounded

if w is a CLIQUE

return 1

else

return 0

About NP-hardness (necessary to prove CLIQUE $\in$ NP).

We want to define a function $f: \{0,1\}^* \longrightarrow \{0,1\}^*$

$\lfloor 3CNF \rfloor$ $\quad$ $\lfloor (G, k) \rfloor$

we want to define f in such a way that $g(\lfloor F \rfloor)$ is in CLIQUE iff F is satisfiable.

Given a 3CHF F which can be written as: $(e_1^1 \vee e_2^1 \vee e_3^1) \wedge \dots (e_1^m \vee e_2^m \vee e_3^m)$

clause

We can build G as follows:

e_1^1 • e_2^1 • e_1^m •

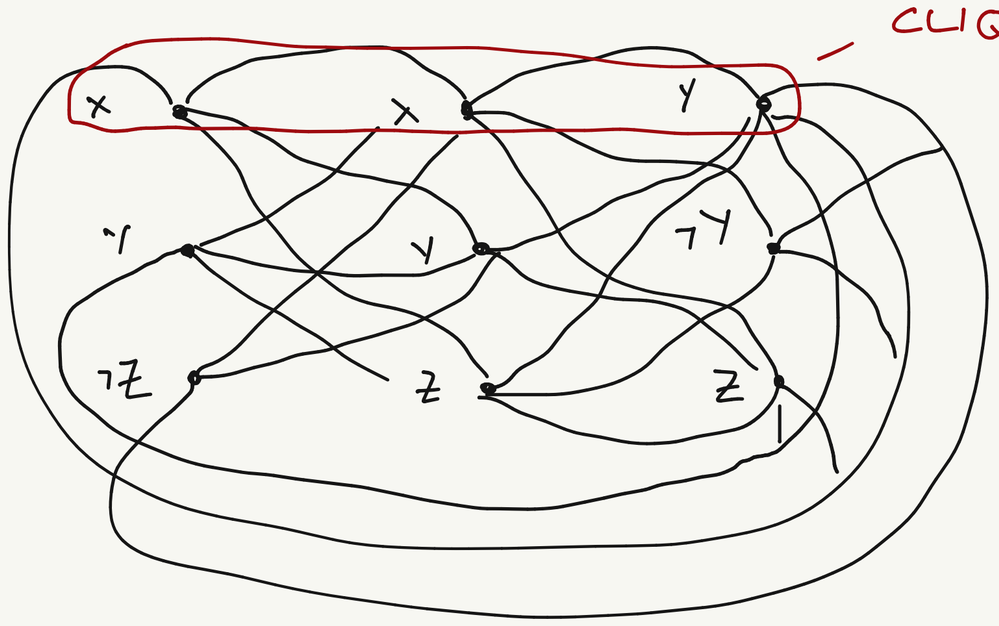
e_2^1 • e_2^2 • ... e_2^m • } nodes

e_3^1 • e_3^2 • e_3^m •

About edges: e_i^k and e_j^s are linked iff:

- $k \neq s$
- e_i^k and e_j^s are consistent, i.e. the former should not be the neg. of the latter or viceversa.

e.g. $F := (x \vee y \vee \neg z) \wedge (x \vee y \vee z) \wedge (x \vee \neg y \vee z)$



We have to prove two things about g:

- g can be computed in polynomial time one should write an algorithm computing g but this is quite easy.
- $g(F) \in$ CLIQUE iff $F \in$ 3SAT:
  - right-to-left impl. Suppose F is satis. then there is an assignment P of truth values s.t. all clauses in F are made true by P. The fact that this holds for a single P means that all literals are comsistent. By construction, they are different clauses. As a consequence, $\{e^{s_1}_1, \ldots, e^{s_m}_m\}$ forms a CLIQUE in $g(F)$.

  - left-to-right implication. Suppose that $g(F)$ has a CLIQUE of size m. This means that all "columns" of $g(F)$ are involved in the CLIQUE. In other words there is exactly 1 node for each columns which is part of the CLIQUE. We can chose one literal from each clause in F s.t one these literals are pair-wise comsistent. By definition of an assignment, this gives us P s.t. F is true:

$\forall x$ $P(x)$ is True if x is part of the CLIQUE. $P(x)$ is false if $\neg x$ is part of the CLIQUE, $P(x)$ can be anything if x and $\neg x$ are not of the CLIQUE. As a coms. F is satisfiable.

EXERCISE. Let G be an undirected graph, then:

- a CLIQUE in G is a subset W of V s.t.
  if $v, w \in W$ and $v \neq w$ then $\{v, w\} \in E$.
- an INDSET in G is a subset W of V s.t.
  if $v, w \in W$ then $\{v, w\} \notin E$.
- a COVER of G is a subset W of V s.t.
  whenever $\{v, w\} \in E$ either $v \in W$ or $w \in W$.

LEMMA. For every graph $G = (V, E)$ and every $W \subseteq V$, the following is equivalent:

iff
1. W is a COVER of G
2. $V - W$ is a INDSET for G
3. $V - W$ is a CLIQUE in the complement G^c of G:
   $G^c = (V, E^c)$, $E^c = \{\{u, v\} \mid u, v \in V, \{u, v\} \notin E\}$.

We can conclude that COVER, INDSET and CLIQUE are all NP-complete just going through the Lemma above.

EXERCISE 15. Is this algorithm in polynomial time?

INPUT: string $s \in \{0,1\}^*$

$p \leftarrow s$

$e \leftarrow |s|$

$i \leftarrow 1$

while $i < e$ do

$p := p :: s$, concat , $i := i+1$

end

return p

SOLUTION. 1. Input can be encoded as binary string.

2. The total number of instructions is polynomially bounded:

The total number of instruction is $2 \cdot e = 2|s|$

3. The intermediate results are also polynomially bounded:

p is equal to $\underbrace{s :: s :: \dots :: s}_{|s| \text{ times}}$, so $|p| \leq |s|^2$

4. All instructions can be simulated in a TM in polynomial time.

The algorithm can be simulated on a poly-time TM.

If one considers the following modified algorithm.

$\vdots$

while $i < e$ do

$i := i+1$

$p := p :: p$

$\vdots$

then, the size of p is $|p| = 2^{\overbrace{i}^{\text{iteration}}} |s|$, at the end $|p| = 2^{|s|} |s|$ (exp).

EXERCISE 16. Prove that this problem is in P:

INPUT: graph $G = (V, E)$

FUNCTION 1 if the graph is triangle-free, 0 otherwise

SOLUTION.

INPUT: graph $G = (V, E)$

Forall $u \in V$

Forall $(v, w) \in E$

if $(u \neq v, u \neq w, (u, v) \in E, (u, w) \in E)$

return 0

return 1

1. A graph can be encoded in a binary string
2. Number of instruction is $|E| \times |V| \times 4$
   └ always $\leq |V|^2$
3. There are no intermediate resets
4. Every instruction can be simulated in a polytime TM.

EXERCISE 17. ∨ ∨ ∨ FP:

INPUT: list of integers

OUTPUT: $f(e) = e'$ which is the sorted list

SOLUTION. one can give any sorting algorithm.

EXERCISE 18. Prove the following is in NP:

INPUT: $G = (V, E)$

$G \in L$ iff G has an Hamiltonian path (goes through all $v \in V$ exactly once)

SOLUTION. The certificate is an Hamiltonian path: $[v_1, v_2, \dots, v_k]$ (polynomial)

In order to verify such path:

1. Each $v_i \in V$
2. $(v_i, v_{i+1}) \in E$
3. All v_i are seen only once

→

INPUT: $G = (V, E)$, p (path)

Initialize an array: $t[v] = 0 \quad \forall v \in V$ (boolean)

$v_0 \leftarrow$ head(p)

$q \leftarrow$ tail(p)

$v_1 \leftarrow$ head(p)

$q \leftarrow$ tail(p)

$t[v_1] \leftarrow 1$, $t[v_0] \leftarrow 1$

if ($v_0 = v_1$ OR $(v_0, v_1) \notin E$) then return 0

while $q \neq []$

$v_0 \leftarrow v_1$

$v_1 \leftarrow$ head(q)

$q \leftarrow$ tail(q)

if ($t[v_1] = 1$ OR $(v_0, v_1) \notin E$) then return 0

end

Forall $v \in V$

if $t[v] = 0$ return 0

return 1

EXERCISE 19. If $L_1 \in NP$ and $L_2 \in NP$, does $L_1 \cap L_2 \in NP$?

SOLUTION. $L_1 \in NP$ so V_1 and M_1

$L_2 \in NP$ so V_2 and M_2

$x \in \{0,1\}^*$, $x \in L_1 \cap L_2$ iff $x \in L_1$ and $x \in L_2$

iff y_1 and y_2 are s.t. $M_1(\langle x, y_1 \rangle) = 1 \wedge$
'certificate for $x \in L_1$ $M_2(\langle x, y_2 \rangle) = 1$

y is the certificate $\langle y_1, y_2 \rangle$, and M is the TM s.t on

input x, y_1, y_2:

1. Simulate $M_1(\langle x, y_1 \rangle)$

   - if $M_1(\langle x, y_1 \rangle) = 0$, return 0

   else

2. Simulate $M_2(\langle x, y_2 \rangle)$

   - return $M_2(\langle x, y_2 \rangle)$

EXERCISE 20. What about $L_1 \cup L_2$?

SOLUTION. Let $x \in \{0,1\}^*$.

Certificate: either 1 as string y_1, or 2 as string y_2

Verifier: on an input x, c : (certificate)

- if $c = \langle 1, y_1 \rangle$ then return $M_1(\langle x, y_1 \rangle)$
- else $c = \langle 2, y_2 \rangle$ " " $M_2(\langle x, y_2 \rangle)$

EXERCISE 20. what about $\mathcal{L}_1$ concat $\mathcal{L}_2$?

SOLUTION. $\mathcal{L}_1$ CONCAT $\mathcal{L}_2 = \{ x \in \{0,1\}^* \mid \exists x_1, x_2 , x_1 \in \mathcal{L}_1, x_2 \in \mathcal{L}_2 , x = x_1 :: x_2 \}$

Let $x \in \{0,1\}^*$.

Certificate: $\langle m, y_1, y_2 |$

m: number of characters in x_1

Verifier: on input $x, \langle m, y_1, y_2 |$:

1. Separate x into x_1 and x_2 using m ($x_1 = w_1 \cdots w_m$, $x_2 = w_{m+1} \cdots w_m$ |

2. Simulate $M_1(\lfloor x_1, y_1 \rfloor)$

   - if $=0$ then return 0

3. Simulate $M_2(\lfloor x_2, y_2 \rfloor)$

EXERCISE 21. $\mathcal{L} \leq_p H$ iff there is a polytime function

$f : \{0,1\}^* \to \{0,1\}^*$ s.t. $x \in \mathcal{L} \longleftrightarrow f(x) \in H$

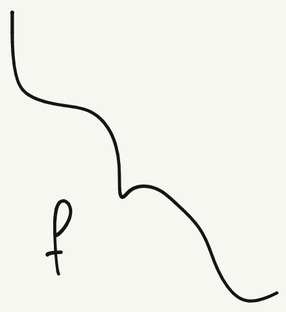
Prove that SAT $\leq_p$ 3SAT.

└ can clauses have at most 3 literals

SOLUTION. we need to find f s.t. $f(x) = y$ and $x \in \text{SAT} \leftrightarrow y \in \text{3SAT}$.

take a clause $c_1 \vee c_2 \vee \dots \vee c_k$ with $k > 3$. we introduce new variables $z_1, \dots, z_{k-3}$, $(c_1 \vee c_2 \vee z_1) \wedge (\overline{z_1} \vee c_3 \vee z_2) \wedge (\overline{z_2} \vee c_4 \vee z_3) \wedge \dots \wedge (\overline{z_{k-4}} \vee c_{k-2} \vee z_{k-3}) \wedge (\overline{z_{k-3}} \vee c_{k-1} \vee c_k)$

see notes for the entire solution



EXERCISE 22. Suppose that INDSET is NP-complete.

NODECOVER:

input: $G=(V,E)$ and $b \geq 2$.

output: return 1 if there is a set $C \subseteq V$ s.t

- $|C| \leq b$
- for all $(u,v) \in E$, $u \in C$ or $v \in C$.

Prove that NODECOVER is NP-complete

SOLUTION. NODECOVER $\in$ NP ?

Certificate: A subset C of V, which is polysize.

Verifier: verify that C is a covering and that $|C| \leq b$.

↳ this is polytime

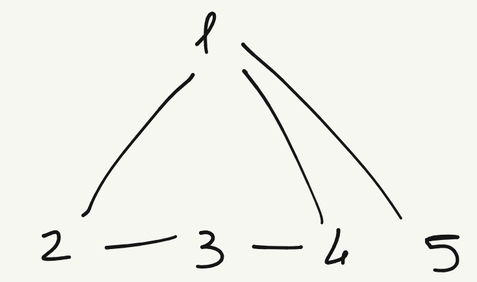
↳ the problem is in NP

NODECOVER $\in$ NP-HARD?

Prove that INDSET $\leq_p$ NODECOVER. Find f s.t.

$$\langle G, k \rangle \in \text{INDSET} \iff f(\langle G, k \rangle) \in \text{NODECOVER}$$
$$= \langle G', b \rangle$$

Considering a random graph:



The maximal indset is of size 3: $\{2,4,5\}$.

Node covering set of size 2: $\{1,3\}$.

Let's take $G=(V,E)$. I is indset of $G \iff \forall u,v \in I, (u,v) \notin E \iff \forall (u,v) \in E, u \notin I$ or $v \notin I \iff \forall (u,v) \in E$.

$u \in I^c$ or $v \in I^c$.

$\iff$ I^c is a node covering of G.

We prove $f(\langle G, k\rangle) = \langle G, |V| - k\rangle$.

$\langle G, k\rangle \in$ INDSET $\iff$ there exist I, indset of G

$\iff$ there exist $C = I^c$, C is a node cov of G

$|C| \leq |V| - k$

$\iff$ $\langle G, |V| - k\rangle \in$ NODECOVERING

Therefore INDSET $\leq_p$ NODECOVER and NODECOVER $\in$ NP-COMPLETE.

EXERCISE 23. CLIQUE $\in$ NP-COMPLETE?

- Certificate: a subset $C \subseteq V$ of polynomial size.
- Verifier: $|C| \geq k$ and forall $(u,v) \in C$, $u \neq v$, $(u,v) \in E$
  ↳ polytime

Moreover INDSET $\leq_p$ CLIQUE. We need to find f s.t.

$\langle G, k\rangle \in$ INDSET $\iff$ $f(\langle G, k\rangle) \in$ CLIQUE

$\langle G', k'\rangle$

Considering the same graph of the previous exercise:

- INDSET: $\{2, 4, 5\}$.
- CLIQUE: $\{2, 4, 5\}$ for G^c

Suppose I is indset of G $\iff$ $\forall u, v \in I$, $(u,v) \notin E$

$\iff$ $\forall u, v \in I$, $(u,v) \in E^c$, $u = v$

$\iff$ $\forall u, v \in I$, $u \neq v$, $(u,v) \in E^c$

$\iff$ I is a clique in G^c

Reduction: we pose $f(\langle G, k\rangle) = \langle G^c, k\rangle$.

$\langle G, k\rangle \in$ INDSET $\Leftrightarrow \exists I$, indset of G

$\Leftrightarrow \exists I$, clique of G^c, $|I| \geq k$

$\Leftrightarrow \langle G, k\rangle \in$ CLIQUE.

Therefore, CLIQUE $\in$ NP-COMPLETE.

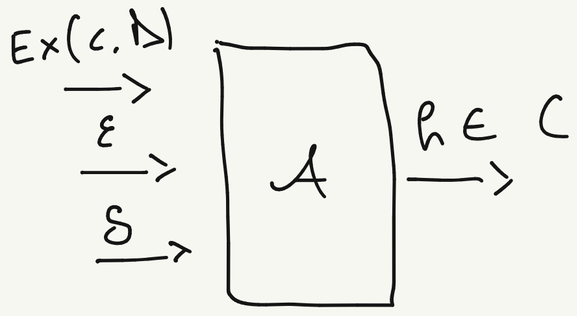
EXERCISE 24. 5-CLIQUE?

Input: $G = (V, E)$

Output: 1 if G has a clique of size 5.

5-CLIQUE is in P! (if one gives size of clique then it is in P)

EXERCISE 25. Instance space X, concept class $C \subseteq P(X)$ (subsets of)



A class is PAC-learnable iff there exist an algorithm

s.t. $\forall c \in C$, D, $0 < \varepsilon < 1/2$, $0 < \delta < 1/2$ we have that:

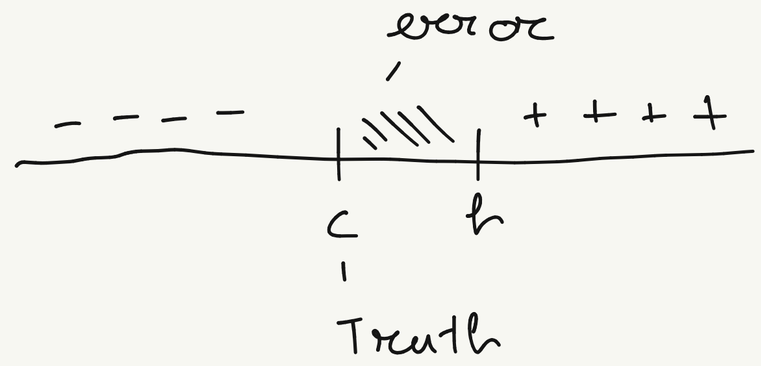
$$Pr_{c,D}(error(A(EX(c,D), \varepsilon, \delta)) < \varepsilon) > 1 - \delta$$

↑ if A polynomial PAC efficient

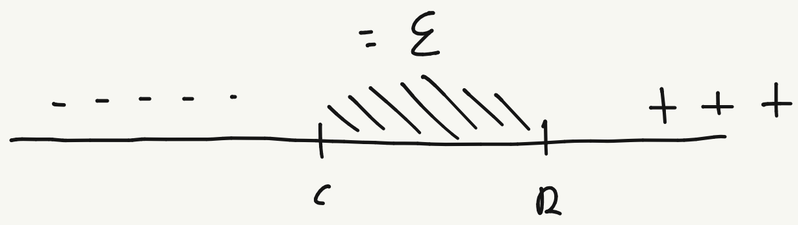
- Lear positive half line: ----0++++ $x = \mathbb{R}$

Algorithm:

one makes an hypothesis h :
(according to training data



Bad case : $Pr \geq \varepsilon$. In order to avoid this, one can define R:



if a data falls into $[c, R]$ then the $Pr[err(h)] < \varepsilon$.

So, $Pr[err \geq \varepsilon] \leq Pr$ (no training data falls in $[c, R]$)

$$Pr[err \geq \varepsilon] \leq Pr(x_1 \notin [c,R], \ldots, x_m \notin [c,R])$$

$x_1 \ldots x_m$ i.i.d.

$$\leq Pr(x_1 \notin [c,R]) \cdot \ldots \cdot Pr(x_m \in [c,R])$$

$$\leq (1-\varepsilon) \cdots (1-\varepsilon) \rightarrow (1-\varepsilon)^m$$

we have $1 + x \leq e^x$ for all x : $Pr(err \geq \varepsilon) \leq e^{-mx}$

we want $Pr(err \geq \varepsilon) < \delta$: it is sufficient to have $e^{-m\varepsilon} < \delta$,

$-m\varepsilon < \ln(\delta)$. $-m < \frac{1}{\varepsilon} \ln(\delta)$, $m > \frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)$.

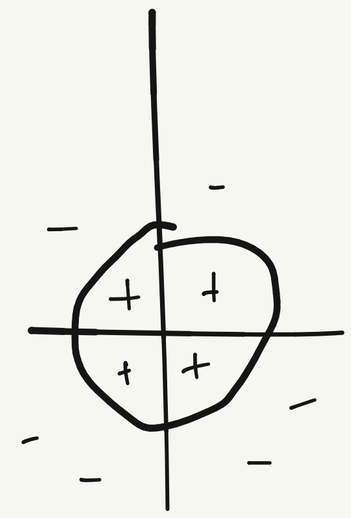
If m is greater than $\frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)$, the probab. of error of the algorithm is $> 1 - \delta$:

$$Pr(err(A(EX(c,D), \varepsilon, \delta)) < \varepsilon] > 1 - \delta$$

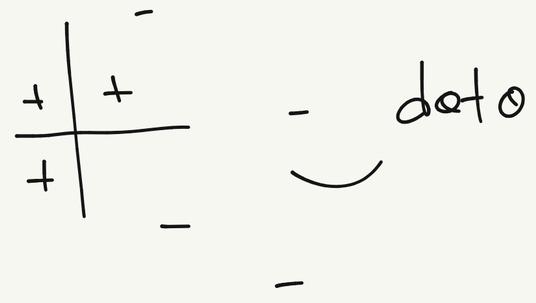
the algorithm is polynomial, the number of samples is polynomial, then half lines is efficiently PAC learnable (in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$

EXERCISE 26. Learning concentric circles.

$X = \mathbb{R}^2$

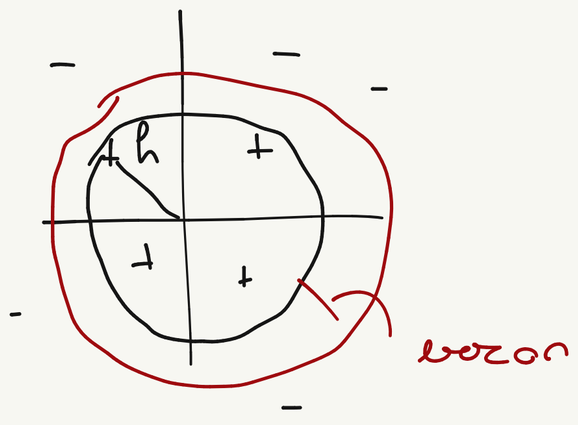


Concept class: circles centered at origin

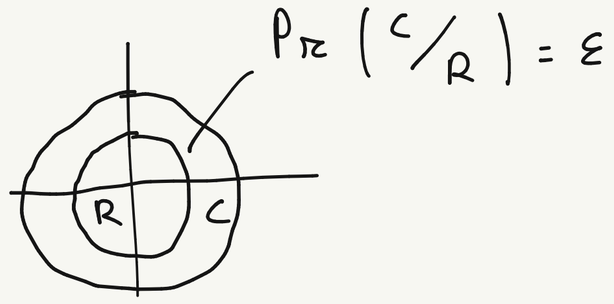
Algorithm: 

data

For each point (x, y), take the radius and consider the max to be h, so to build a circle.



we define R such that



so, $Pr(error(h) \geq \varepsilon) \leq Pr$ (no training data falls into C/R)

$\leq Pr(x_1 \notin C/R, \ldots, x_m \notin C/R)$

$\leq Pr(x_1 \in C/R) \cdots Pr(x_m \notin C/R)$

$\leq (1-\varepsilon)^m$

as previous exercise.

EXERCISE 27. We say that an algorithm A chose hypothesis in a set H if for any input $EX(c, D), \varepsilon, \delta$ the output $h \in H$.

$H \subseteq C$.

Let us consider H finite.

Theorem: if there exist a learning algorithm $\mathcal{A}$ that finds an hypothesis $h_{\mathcal{A}} \in H$ consistent with m samples, for $m \geq \frac{1}{\varepsilon}\left(\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right)$, then

$$P_n\left(\underset{\text{c.d}}{\text{error}}(h_{\mathcal{A}}) \geq \varepsilon\right) \leq \delta$$

Fix $\varepsilon > 0$, we do not know which consistent hypothesis $h_{\mathcal{A}}$ is selected by $\mathcal{A}$. So, the bound should work for all consistent h.

$$P_n\left(\exists h \in H \mid h \text{ consistent with } m \text{ samples}, \text{err}(h) \geq \varepsilon\right)$$

$= P_n(h_1 \text{ consistent} \vee \text{or } h_2 \text{ consistent} \vee \text{or } h_{|H|} \text{ consistent} \vee)$

$|H|$ finite

$$P_n[\vee] \leq \sum_{h \in |H|} P_n(h_i \text{ consistent with } \text{err}(h) \geq \varepsilon)$$

conditional prob

$$\leq \sum_{h \in |H|} P_n(h_i \text{ consistent} \mid \text{err}(h) \geq \varepsilon)\, P_n(\text{err}(h) \geq \varepsilon)$$

$$\leq \sum_{h \in |H|} P_n(h_i \text{ consistent} \mid \text{err}(h') \geq \varepsilon)$$

$$\leq \vee \quad (1-\varepsilon)^m$$

$$\leq |H|(1-\varepsilon)^m \leq |H|\, e^{-m\varepsilon}$$

it is sufficient to take $|H|\, e^{-m\varepsilon} \leq \delta$

so $e^{-m\varepsilon} \leq \delta/|H| \rightarrow m \geq 1/\varepsilon \ln\left(\frac{|H|}{\delta}\right) \geq \frac{1}{3}\left(\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right)$

EXERCISE 28 comsider the imstauce spoce $\{0,1\}^m$,

comopt closs: comjuctiom of eitoroes.

A subset $c \subseteq P(x)$ is roprosouted by: $e_1 \wedge e_2 \wedge e_3 \cdots$

where $e_i = x_k$ on $\bar{x}_k$. $1 \leq k \leq m$.

corrisponding

The V subset is the subset of assigments satisfying the farmuco.

Algonithm: m assigmouts with a truth volue.

You stort with $x_1 \wedge \bar{x}_1 \wedge x_2 \wedge \bar{x}_2 \cdots$

L ∀ assigmont this is folco.

vector

whom givom o positive troining somplo $(\vec{v}, +1)$,

$v \in \{0,1\}^m$, wo wout this assigmont to verify ouv hipothosys fonmulo, so wo romovo the litoral $\bar{x}_i$

if $v(i)=1$ on x_i if $v(i)=0$.

This olgonithm olwoys roturms o consistont hypothesis.

- The constructed formula is made to be true on all positive troining sample

And we remove the literals we don't need.

According to the target c, if an assigment v is such that v verify c. them it is meassory that ∀ $1 \leq i \leq m$.

if $v(i)=1$ them $x_i \notin c$

$v(i)=0$ them $x_i \in c$

so, by construction, $c \subseteq h$, it means that negative examples for c will be negative for h.

so h is a consistent hypothesis

$|H| = |\{$ all passible conjuctions of literals with n variables $\}|$

$= 4^n$

↳ 4 choices for each variable

so, considering the previous theorem, if $m \geq \frac{1}{\varepsilon}\left(\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right)$ then $P_n(err(h_A) \geq \varepsilon) \leq \delta$.

so it means that $m \geq \frac{1}{\varepsilon}\left(n\ln(4) + \ln\left(\frac{1}{\delta}\right)\right)$.

↳ polynomial in $1/\varepsilon$ and $1/\delta$ and n

conjuction of literals is efficiently PAC learnable.