

Languages and Algorithms for Artificial Intelligence (Module 3)

Last update: 06 May 2024

Academic Year 2023 – 2024
Alma Mater Studiorum · University of Bologna

Contents

1	Introduction	1
1.1	Notations	1
1.1.1	Strings	1
1.1.2	Tasks encoding	1
1.1.3	Asymptotic notation	2
2	Turing Machine	3
2.1	k -tape Turing Machine	3
2.2	Computation	4
2.3	Universal Turing Machine	4
2.4	Computability	5
2.4.1	Undecidable functions	5
2.4.2	Rice's theorem	6
3	Complexity	7
3.1	Polynomial time	7
3.2	Exponential time	7
3.3	NP class	8
4	Computational learning theory	11
4.1	Boolean functions as representation class	12
4.1.1	Conjunctions of literals	12
4.1.2	3DNF	13
4.2	Axes-aligned rectangles over $\mathbb{R}_{[0,1]}^2$	13
5	VC dimension	16

1 Introduction

Computational task Description of a problem.

Computational task

Computational process Algorithm to solve a task.

Computational
process

Algorithm (informal) A finite description of elementary and deterministic computation steps.

1.1 Notations

Set of the first n natural numbers Given $n \in \mathbb{N}$, we have that $[n] = \{1, \dots, n\}$.

1.1.1 Strings

Alphabet Finite set of symbols.

Alphabet

String Finite, ordered, and possibly empty tuple of elements of an alphabet.

String

The empty string is denoted as ε .

Strings of given length Given an alphabet S and $n \in \mathbb{N}$, we denote with S^n the set of all the strings over S of length n .

Kleene star Given an alphabet S , we denote with $S^* = \bigcup_{n=0}^{\infty} S^n$ the set of all the strings over S .

Kleene star

Language Given an alphabet S , a language \mathcal{L} is a subset of S^* .

Language

1.1.2 Tasks encoding

Encoding Given a set A , any element $x \in A$ can be encoded into a string of the language $\{0, 1\}^*$. The encoding of x is denoted as $\lfloor x \rfloor$ or simply x .

Encoding

Task function Given two countable sets A and B representing the domain, a task can be represented as a function $f : A \rightarrow B$.

Task

When not stated, A and B are implicitly encoded into $\{0, 1\}^*$.

Characteristic function Boolean function of form $f : \{0, 1\}^* \rightarrow \{0, 1\}$.

Characteristic
function

Given a characteristic function f , the language $\mathcal{L}_f = \{x \in \{0, 1\}^* \mid f(x) = 1\}$ can be defined.

Decision problem Given a language \mathcal{M} , a decision problem is the task of computing a boolean function f able to determine if a string belongs to \mathcal{M} (i.e. $\mathcal{L}_f = \mathcal{M}$).

Decision problem

1.1.3 Asymptotic notation

Big O A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is $O(g)$ if g is an upper bound of f .

Big O

$$f \in O(g) \iff \exists \bar{n} \in \mathbb{N} \text{ such that } \forall n > \bar{n}, \exists c \in \mathbb{R}^+ : f(n) \leq c \cdot g(n)$$

Big Omega A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is $\Omega(g)$ if g is a lower bound of f .

Big Omega

$$f \in \Omega(g) \iff \exists \bar{n} \in \mathbb{N} \text{ such that } \forall n > \bar{n}, \exists c \in \mathbb{R}^+ : f(n) \geq c \cdot g(n)$$

Big Theta A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is $\Theta(g)$ if g is both an upper and lower bound of f .

Big Theta

$$f \in \Theta(g) \iff f \in O(g) \text{ and } f \in \Omega(g)$$

2 Turing Machine

2.1 k -tape Turing Machine

Tape Infinite one-directional line of cells. Each cell can hold a symbol from a finite alphabet Γ . Tape

Tape head A tape head reads or writes one symbol at a time and can move left or right on the tape.

Input tape Read-only tape where the input will be loaded.

Work tape Read-write auxiliary tape used during computation.

Output tape Read-write tape that will contain the output of the computation.

Remark. Sometimes the output tape is not necessary and the final state of the computation can be used to determine a boolean outcome.

Instructions Given a finite set of states Q , at each step, a machine can: Instructions

Read from the k tape heads.

Replace the symbols under the writable tape heads, or leave them unchanged.

Change state.

Move each of the k tape heads to the left or right, or leave unchanged.

k -tape Turing Machine (TM) A Turing Machine working on k tapes (one of which is the input tape) is a triple (Γ, Q, δ) : k -tape Turing Machine (TM)

- Γ is a finite set of tape symbols. We assume that it contains a blank symbol (\square), a start symbol (\triangleright), and the digits 0, 1.
- Q is a finite set of states. The initial state is q_{init} and the final state is q_{halt} .
- δ is the transition function that describes the instructions allowed at each step. It is defined as:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{\text{L}, \text{S}, \text{R}\}^k$$

By convention, when the state is q_{halt} , the machine is stuck (i.e. it cannot change state or operate on the tapes):

$$\delta(q_{\text{halt}}, \{\sigma_1, \dots, \sigma_k\}) = (q_{\text{halt}}, \{\sigma_1, \dots, \sigma_k\}, (\text{S}, \dots, \text{S}))$$

Theorem 2.1.1 (Turing Machine equivalence). The following computational models have, with at most a polynomial overhead, the same expressive power: 1-tape TMs, k -tape TMs, non-deterministic TMs, random access machines, λ -calculus, unlimited register machines, programming languages (Böhm-Jacopini theorem), ...

2.2 Computation

Configuration Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$, a configuration C is described by:

Configuration

- The current state q .
- The content of the tapes.
- The position of the tape heads.

Initial configuration Given the input $x \in \{0, 1\}^*$, the initial configuration \mathcal{I}_x is described as follows:

- The current state is q_{init} .
- The first (input) tape contains $\triangleright x \square \dots$. The other tapes contain $\triangleright \square \dots$.
- The tape heads are positioned on the first symbol of each tape.

Final configuration Given an output $y \in \{0, 1\}^*$, the final configuration is described as follows:

- The current state is q_{halt} .
- The output tape contains $\triangleright y \square \dots$.

Computation (string) Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$, \mathcal{M} returns $y \in \{0, 1\}^*$ on input $x \in \{0, 1\}^*$ (i.e. $\mathcal{M}(x) = y$) in t steps if:

Computation
(string)

$$\mathcal{I}_x \xrightarrow{\delta} C_1 \xrightarrow{\delta} \dots \xrightarrow{\delta} C_t$$

where C_t is a final configuration for y .

Computation (function) Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$ and a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, \mathcal{M} computes f iff:

Computation
(function)

$$\forall x \in \{0, 1\}^* : \mathcal{M}(x) = f(x)$$

If this holds, f is a computable function.

Computation in time T Given a TM \mathcal{M} and the functions $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ and $T : \mathbb{N} \rightarrow \mathbb{N}$, \mathcal{M} computes f in time T iff:

Computation in time
 T

$$\forall x \in \{0, 1\}^* : \mathcal{M}(x) \text{ returns } f(x) \text{ in at most } T(|x|) \text{ steps}$$

Decidability in time T Given a function $f : \{0, 1\}^* \rightarrow \{0, 1\}$, the language \mathcal{L}_f is decidable in time T iff f is computable in time T .

Decidability in time
 T

2.3 Universal Turing Machine

Turing Machine encoding Given a TM $\mathcal{M} = (\Gamma, Q, \delta)$, the entire machine can be described by δ through tuples of form:

$$Q \times \Gamma^k \times Q \times \Gamma^{k-1} \times \{L, S, R\}^k$$

It is therefore possible to encode δ into a binary string and consequently create an encoding $\sqcup \mathcal{M} \sqcup$ of \mathcal{M} .

The encoding should satisfy the following conditions:

1. For every $x \in \{0, 1\}^*$, there exists a TM \mathcal{M} such that $x = \sqcup \mathcal{M} \sqcup$.

2. Every TM is represented by an infinite number of strings. One of them is the canonical representation.

Theorem 2.3.1 (Universal Turing Machine (UTM)). There exists a TM \mathcal{U} such that, for every binary strings x and α , it emulates the TM defined by α on input x :

Universal Turing Machine (UTM)

$$\mathcal{U}(x, \alpha) = \mathcal{M}_\alpha(x)$$

where \mathcal{M}_α is the TM defined by α .

Moreover, \mathcal{U} simulates \mathcal{M}_α with at most $CT \log(T)$ time overhead, where C only depends on \mathcal{M}_α .

2.4 Computability

2.4.1 Undecidable functions

Theorem 2.4.1 (Existence of uncomputable functions). There exists a function $uc : \{0, 1\}^* \rightarrow \{0, 1\}^*$ that is not computable by any TM.

Uncomputable functions

Proof. Consider the following function:

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

If uc was computable, there would be a TM \mathcal{M} that computes it (i.e. $\forall \alpha \in \{0, 1\}^* : \mathcal{M}(\alpha) = uc(\alpha)$). This will result in a contradiction:

$$uc(\perp \mathcal{M} \perp) = 0 \iff \mathcal{M}(\perp \mathcal{M} \perp) = 1 \iff uc(\perp \mathcal{M} \perp) = 1$$

Therefore, uc cannot be computed. □

Halting problem Given an encoded TM α and a string x , the halting problem aims to determine if \mathcal{M}_α terminates on input x . In other words:

Halting problem

$$\text{halt}(\perp(\alpha, x) \perp) = \begin{cases} 1 & \text{if } \mathcal{M}_\alpha \text{ stops on input } x \\ 0 & \text{otherwise} \end{cases}$$

Theorem 2.4.2. The halting problem is undecidable.

Proof. Note: this proof is slightly different from the traditional proof of the halting problem.

Assume that **halt** is decidable. Therefore, there exists a TM $\mathcal{M}_{\text{halt}}$ that decides it.

We can define a new TM \mathcal{M}_{uc} that uses $\mathcal{M}_{\text{halt}}$ such that:

$$\mathcal{M}_{uc}(\alpha) = \begin{cases} 1 & \text{if } \mathcal{M}_{\text{halt}}(\alpha, \alpha) = 0 \text{ (i.e. } \mathcal{M}_\alpha(\alpha) \text{ diverges)} \\ \begin{cases} 0 & \text{if } \mathcal{M}_\alpha(\alpha) = 1 \\ 1 & \text{if } \mathcal{M}_\alpha(\alpha) \neq 1 \end{cases} & \text{if } \mathcal{M}_{\text{halt}}(\alpha, \alpha) = 1 \text{ (i.e. } \mathcal{M}_\alpha(\alpha) \text{ converges)} \end{cases}$$

This results in a contradiction:

- $\mathcal{M}_{uc}(\perp \mathcal{M}_{uc} \perp) = 1 \iff \mathcal{M}_{\text{halt}}(\perp \mathcal{M}_{uc} \perp, \perp \mathcal{M}_{uc} \perp) = 0 \iff \mathcal{M}_{uc}(\perp \mathcal{M}_{uc} \perp) \text{ diverges}$
- $\mathcal{M}_{\text{halt}}(\perp \mathcal{M}_{uc} \perp, \perp \mathcal{M}_{uc} \perp) = 1 \Rightarrow \mathcal{M}_{uc} \text{ is not computable by Theorem 2.4.1.}$

□

Diophantine equation Polynomial equality with integer coefficients and a finite number of unknowns.

Diophantine equation

Theorem 2.4.3 (MDPR). Determining if an arbitrary diophantine equation has a solution is undecidable.

2.4.2 Rice's theorem

Semantic language Given a language $\mathcal{L} \subseteq \{0, 1\}^*$, \mathcal{L} is semantic if:

Semantic language

- Any string in \mathcal{L} is an encoding of a TM.
- If $\llbracket \mathcal{M} \rrbracket \in \mathcal{L}$ and the TM \mathcal{N} computes the same function of \mathcal{M} , then $\llbracket \mathcal{N} \rrbracket \in \mathcal{L}$.

A semantic language can be seen as a set of TMs that have the same property.

Trivial language A language \mathcal{L} is trivial iff $\mathcal{L} = \emptyset$ or $\mathcal{L} = \{0, 1\}^*$

Theorem 2.4.4 (Rice's theorem). If a semantic language is non-trivial, then it is undecidable (i.e. any decidable semantic language is trivial).

Rice's theorem

Proof idea. Assuming that there exists a non-trivial decidable semantic language \mathcal{L} , it is possible to prove that the halting problem is decidable. Therefore, \mathcal{L} is undecidable. □

3 Complexity

Complexity class Set of tasks that can be computed within some fixed resource bounds. Complexity class

3.1 Polynomial time

Deterministic time (DTIME) Let $T : \mathbb{N} \rightarrow \mathbb{N}$ and \mathcal{L} be a language. \mathcal{L} is in **DTIME**($T(n)$) iff there exists a TM that decides \mathcal{L} in time $O(T(n))$. Deterministic time (**DTIME**)

Polynomial time (P) The class **P** contains all the tasks computable in polynomial time: Polynomial time (**P**)

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

Remark. **P** is closed to various operations on programs (e.g. composition of programs)

Remark. In practice, the exponent is often small.

Remark. **P** considers the worst case and is not always realistic. Other alternative computational models exist.

Church-Turing thesis Any physically realizable computer can be simulated by a TM with an arbitrary time overhead. Church-Turing thesis

Strong Church-Turing thesis Any physically realizable computer can be simulated by a TM with a polynomial time overhead. Strong Church-Turing thesis

Remark. If this thesis holds, the class **P** is robust (i.e. does not depend on the computational device) and is therefore the smallest class of bounds.

Deterministic time for functions (FDTIME) Let $T : \mathbb{N} \rightarrow \mathbb{N}$ and $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$. f is in **FDTIME**($T(n)$) iff there exists a TM that computes it in time $O(T(n))$. Deterministic time for functions (**FDTIME**)

Polynomial time for functions (FP) The class **FP** is defined as: Polynomial time for functions (**FP**)

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

Remark. It holds that $\forall \mathcal{L} \in \mathbf{P} \Rightarrow f_{\mathcal{L}} \in \mathbf{FP}$, where $f_{\mathcal{L}}$ is the characteristic function of \mathcal{L} . Generally, the contrary does not hold.

3.2 Exponential time

Exponential time (EXP/FEXP) The **EXP** and **FEXP** classes are defined as: Exponential time (**EXP/FEXP**)

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c}) \quad \mathbf{FEXP} = \bigcup_{c \geq 1} \mathbf{FDTIME}(2^{n^c})$$

Theorem 3.2.1. The following hold:

$$\mathbf{P} \subset \mathbf{EXP} \quad \mathbf{FP} \subset \mathbf{FEXP}$$

3.3 NP class

Certificate Given a set of pairs $\mathcal{C}_{\mathcal{L}}$ and a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$, we can define the language \mathcal{L} such that:

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

Given a string w and a certificate y , we can exploit $\mathcal{C}_{\mathcal{L}}$ as a test to check whether y is a certificate for w :

$$w \in \mathcal{L} \iff (w, y) \in \mathcal{C}_{\mathcal{L}}$$

Nondeterministic TM (NDTM) TM that has two transition functions δ_0, δ_1 and, at each step, non-deterministically chooses which one to follow. A state q_{accept} is always present:

- A NDTM accepts a string iff one of the possible computations reaches q_{accept} .
- A NDTM rejects a string iff none of the possible computations reach q_{accept} .

Nondeterministic time (NDTIME) Let $T : \mathbb{N} \rightarrow \mathbb{N}$ and \mathcal{L} be a language. \mathcal{L} is in $\text{NDTIME}(T(n))$ iff there exists a NDTM that decides \mathcal{L} in time $O(T(n))$.

Remark. A NDTM \mathcal{M} runs in time $T : \mathbb{N} \rightarrow \mathbb{N}$ iff for every input, any possible computation terminates in time $O(T(n))$.

Complexity class NP

NDTM formulation The class **NP** contains all the tasks computable in polynomial time by a nondeterministic TM:

$$\mathbf{NP} = \bigcup_{c \geq 1} \text{NDTIME}(n^c)$$

Verifier formulation Let $\mathcal{L} \subseteq \{0, 1\}^*$ be a language. \mathcal{L} is in **NP** iff there exists a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial TM \mathcal{M} (verifier) such that:

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

In other words, \mathcal{L} is the language of the strings that can be verified by \mathcal{M} in polynomial time using a certificate y of polynomial length.

Theorem 3.3.1. $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$.

Proof. We have to prove that $\mathbf{P} \subseteq \mathbf{NP}$ and $\mathbf{NP} \subseteq \mathbf{EXP}$:

$\mathbf{P} \subseteq \mathbf{NP}$ Given a language $\mathcal{L} \in \mathbf{P}$, we want to prove that $\mathcal{L} \in \mathbf{NP}$.

By hypothesis, there is a polynomial time TM \mathcal{N} that decides \mathcal{L} . To prove that \mathcal{L} is in **NP**, we show that there is a polynomial verifier \mathcal{M} that certifies \mathcal{L} with a polynomial certificate. We can use any constant certificate (e.g. of length 1) and use \mathcal{N} as the verifier \mathcal{M} :

$$\mathcal{M}(x, y) = \begin{cases} 1 & \text{if } \mathcal{N}(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

\mathcal{M} can ignore the polynomial certificate and it "verifies" a string in polynomial time through \mathcal{N} .

NP \subseteq EXP) Given a language $\mathcal{L} \in \mathbf{NP}$, we want to prove that $\mathcal{L} \in \mathbf{EXP}$.

By hypothesis, there is a polynomial time TM \mathcal{N} that is able to certify any string in \mathcal{L} with a polynomial certificate. Given a polynomial p , can define the following algorithm:

```
def np_to_exp(x ∈ {0,1}*):
    foreach y ∈ {0,1}^{p(|x|)}:
        if M(x,y) == 1:
            return 1
    return 0
```

The algorithm has complexity $O(2^{p(|x|)}) \cdot O(q(|x| + |y|)) = O(2^{p(|x|) + \log(q(|x| + |y|))})$, where q is a polynomial. Therefore, the complexity is exponential.

□

Polynomial-time reducibility A language \mathcal{L} is poly-time reducible to \mathcal{H} ($\mathcal{L} \leq_p \mathcal{H}$) iff:

Polynomial-time
reducibility

$\exists f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that $(x \in \mathcal{L} \iff f(x) \in \mathcal{H})$ and
 f is computable in poly-time

f can be seen as a mapping function.

Remark. Intuitively, when $\mathcal{L} \leq_p \mathcal{H}$, \mathcal{H} is at least as difficult as \mathcal{L} .

Theorem 3.3.2. The relation \leq_p is a pre-order (i.e. reflexive and transitive).

Proof. We want to prove that \leq_p is reflexive and transitive:

Reflexive) Given a language \mathcal{L} , we want to prove that $\mathcal{L} \leq_p \mathcal{L}$.

We have to find a poly-time function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that:

$$x \in \mathcal{L} \iff f(x) \in \mathcal{L}$$

We can choose f as the identity function.

Transitive) Given the languages $\mathcal{L}, \mathcal{H}, \mathcal{J}$, we want to prove that:

$$(\mathcal{L} \leq_p \mathcal{H}) \wedge (\mathcal{H} \leq_p \mathcal{J}) \Rightarrow (\mathcal{L} \leq_p \mathcal{J})$$

By hypothesis, it holds that $\mathcal{L} \leq_p \mathcal{H}$ and $\mathcal{H} \leq_p \mathcal{J}$. Therefore, there are two poly-time functions $f, g : \{0,1\}^* \rightarrow \{0,1\}^*$ such that:

$$x \in \mathcal{L} \iff f(x) \in \mathcal{H} \text{ and } y \in \mathcal{H} \iff f(y) \in \mathcal{J}$$

We want to find a poly-time mapping from \mathcal{L} to \mathcal{J} . This function can be the composition $(g \circ f)(z) = g(f(z))$. $(g \circ f)$ is poly-time as f and g are poly-time.

□

NP-hard Given a language $\mathcal{H} \in \{0,1\}^*$, \mathcal{H} is **NP-hard** iff:

NP-hard

$$\forall \mathcal{L} \in \mathbf{NP} : \mathcal{L} \leq_p \mathcal{H}$$

NP-complete Given a language $\mathcal{H} \in \{0,1\}^*$, \mathcal{H} is **NP-complete** iff:

NP-complete

$$\mathcal{H} \in \mathbf{NP} \text{ and } \mathcal{H} \text{ is NP-hard}$$

Theorem 3.3.3.

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

Proof.

1. Let \mathcal{L} be **NP**-hard and $\mathcal{L} \in \mathbf{P}$. We want to prove that $\mathbf{P} = \mathbf{NP}$:

P \subseteq **NP**) Proved in Theorem 3.3.1.

NP \subseteq **P**) Let \mathcal{H} be a language in **NP**. As \mathcal{L} is **NP**-hard, by definition it holds that $\mathcal{H} \leq_p \mathcal{L}$. Moreover, by hypothesis, it holds that $\mathcal{L} \in \mathbf{P}$. Therefore, we can conclude that $\mathcal{H} \in \mathbf{P}$ as it can be reduced to a language in **P**.

2. Let \mathcal{L} be **NP**-complete. We want to prove that $\mathcal{L} \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$:

$(\mathcal{L} \in \mathbf{P}) \Rightarrow (\mathbf{P} = \mathbf{NP})$ Trivial for Point 1 as \mathcal{L} is also **NP**-hard.

$(\mathcal{L} \in \mathbf{P}) \Leftarrow (\mathbf{P} = \mathbf{NP})$ Let $\mathbf{P} = \mathbf{NP}$. As \mathcal{L} is **NP**-complete, it holds that $\mathcal{L} \in \mathbf{NP} = \mathbf{P}$.

□

Theorem 3.3.4. The problem **TMSAT** of simulating any TM is **NP**-complete:

$$\mathbf{TMSAT} = \{(\alpha, x, 1^n, 1^t) \mid \exists u \in \{0, 1\}^n : \mathcal{M}_\alpha(x, u) = 1 \text{ within } t \text{ steps}\}$$

Theorem 3.3.5 (Cook-Levin). The following languages are **NP**-complete:

Cook-Levin theorem

$$\begin{aligned} \mathbf{SAT} &= \{\ulcorner F \urcorner \mid F \text{ is a satisfiable CNF}\} \\ \mathbf{3SAT} &= \{\ulcorner F \urcorner \mid F \text{ is a satisfiable 3CNF}\} \end{aligned}$$

4 Computational learning theory

Instance space Set X of (encoded) instances of objects that a learner wants to classify. Instance space
 Data from the instance space is drawn from a distribution \mathcal{D} unknown to the learner.

Concept Subset $c \subseteq X$ of the instance space which can be intended as properties of objects (i.e. a way to classify the instance space). Concept

Concept class Collection $\mathcal{C} \subseteq \mathbb{P}(X)$ of concepts. Concept class
 It represents the concepts that are sufficiently simple for the algorithm to handle (i.e. the space of learnable concepts).

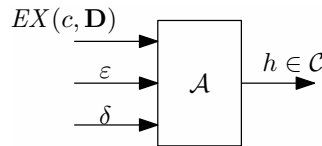
Target concept Concept $c \in \mathcal{C}$ that the learner wants to learn.

Remark. A learning algorithm is designed to learn concepts from a concept class neither knowing the target concept nor its data distribution.

Learning algorithm Given a concept class \mathcal{C} and a target concept $c \in \mathcal{C}$ with unknown distribution \mathcal{D} , a learning algorithm \mathcal{A} takes as input: Learning algorithm

- ε , the error parameter (or accuracy if seen as $(1 - \varepsilon)$),
- δ , the confidence parameter,
- $EX(c, \mathcal{D})$, an oracle that \mathcal{A} can call to retrieve a data point $x \sim \mathcal{D}$ with a label to indicate whether it is in the target concept c or not (i.e. training data),

and outputs a concept $h \in \mathcal{C}$.



Probability of error Given a concept class \mathcal{C} , a target concept $c \in \mathcal{C}$ with unknown distribution \mathcal{D} and a learning algorithm \mathcal{A} , the probability of error (i.e. misclassifications) for any output $h \in \mathcal{C}$ of \mathcal{A} is defined as: Probability of error

$$\text{error}_{\mathcal{D},c} = \mathcal{P}_{x \sim \mathcal{D}}[h(x) \neq c(x)]$$

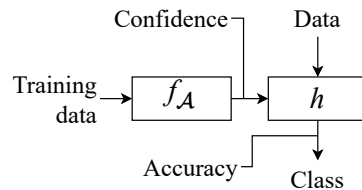


Figure 4.1: General idea of a learning algorithm \mathcal{A} computed as a function $f_{\mathcal{A}}$

PAC learnability A concept class \mathcal{C} over the instance space X is probably approximately correct (PAC) learnable iff there is an algorithm \mathcal{A} such that: PAC learnability

- For each target concept $c \in \mathcal{C}$,
- For each distribution \mathcal{D} ,
- For each error $0 < \varepsilon < \frac{1}{2}$,
- For each confidence $0 < \delta < \frac{1}{2}$,

it holds that:

$$\mathcal{P} \left[\text{error}_{\mathcal{D},c}(\mathcal{A}(EX(c, \mathcal{D}), \varepsilon, \delta)) < \varepsilon \right] > 1 - \delta$$

where the probability is computed by sampling data points from $EX(c, \mathcal{D})$.

In other words, the probability that \mathcal{A} has an error rate lower than ε (or an accuracy higher than $(1 - \varepsilon)$) is greater than $(1 - \delta)$.

Efficient PAC learnability A concept class \mathcal{C} is efficiently PAC learnable iff it is PAC learnable and the algorithm \mathcal{A} that learns it has a time complexity bound to a polynomial in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$. Efficient PAC learnability

Remark. The complexity of \mathcal{A} is measured taking into account the number of calls to $EX(c, \mathcal{D})$.

Representation class A concept class \mathcal{C} is a representation class if each concept $c \in \mathcal{C}$ can be represented as a binary string of $\text{size}(c)$ bits. Representation class

Remark. Let X^n be an instance space (e.g. $\{0, 1\}^n$) and \mathcal{C} be a representation class. If a single learning algorithm \mathcal{A} is designed to work for every n of X^n , then its efficient PAC learnability definition is extended to allow a polynomial in n , $\text{size}(c)$, $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$.

4.1 Boolean functions as representation class

4.1.1 Conjunctions of literals

Consider the instance space $X^n = \{0, 1\}^n$ where the target concept c is a conjunction of literals on n variables x_1, \dots, x_n . The training data is in the form (s, b) where $s \in \{0, 1\}^n$ and $b \in \{0, 1\}$ such that $(b = 1) \Rightarrow (s \in c)$ and $(b = 0) \Rightarrow (s \notin c)$.

A learning algorithm that wants to learn c can proceed as follows:

1. Start with an initial literal h defined as:

$$(x_1 \wedge \neg x_1) \wedge \dots \wedge (x_n \wedge \neg x_n)$$

2. For each training entry $(s, 0)$, ignore it.
3. For each training entry $(s, 1)$, update h by removing literals that contradicts s .

Example. For $n = 3$, assume that the current state h is $x_1 \wedge x_2 \wedge \neg x_2 \wedge \neg x_3$. If the algorithm receives $(101, 1)$, it updates h as $x_1 \wedge \neg x_2$.

Theorem 4.1.1. The representation class of boolean conjunctions of literals is efficiently PAC learnable.

Remark. Conjunctions of literals are highly incomplete.

4.1.2 3DNF

Consider the instance space $X^n = \{0, 1\}^n$ where the target concept c is a 3-term disjunctive normal form formula over n variables x_1, \dots, x_n .

Remark. 3DNF is more expressive than conjunctions of literals but is still not universal.

Remark. 3DNF is the dual of 3CNF.

Theorem 4.1.2. If $\mathbf{NP} \neq \mathbf{Randomized-P}$, then the representation class of 3DNF is not efficiently PAC learnable.

Proof. We have to show that there exists a polytime reduction such that:

$$\begin{array}{ccc} \alpha \in \{0, 1\}^* & \xrightarrow{f} & \mathcal{S}_\alpha \\ \text{Instance of an } \mathbf{NP}\text{-complete} & & \text{Training set for 3DNF} \\ \text{problem (e.g. graph 3-coloring)} & & \end{array}$$

□

4.2 Axes-aligned rectangles over $\mathbb{R}_{[0,1]}^2$

Consider the instance space $X = \mathbb{R}_{[0,1]}^2$ and the concept class \mathcal{C} of concepts represented by all the points contained within a rectangle parallel to the axes of arbitrary size.

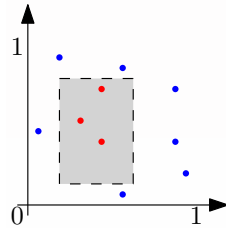


Figure 4.2: Example of problem instance. The gray rectangle is the target concept, red dots are positive data points and blue dots are negative data points.

An algorithm has to guess a classifier (i.e. a rectangle) without knowing the target concept and the distribution of its training data. Let an algorithm \mathcal{A}_{BFP} be defined as follows:

- Take as input some data $\{((x_1, y_1), p_1), \dots, ((x_n, y_n), p_n)\}$ where (x_i, y_i) are the coordinates of the point and p_i indicates if the point is within the target rectangle.
- Return the smallest rectangle that includes all the positive instances.

Given the rectangle R predicted by \mathcal{A}_{BFP} and the target rectangle T , the probability of error in using R in place of T is:

$$\text{error}_{\mathcal{D}, T}(R) = \mathcal{P}_{x \sim \mathcal{D}}[x \in (R \setminus T) \cup (T \setminus R)]$$

In other words, a point is misclassified if it is in R but not in T or vice versa.

Remark. By definition of \mathcal{A}_{BFP} , it always holds that $R \subseteq T$. Therefore, $(R \setminus T) = \emptyset$ and the error can be rewritten as:

$$\text{error}_{\mathcal{D}, T}(R) = \mathcal{P}_{x \sim \mathcal{D}}[x \in (T \setminus R)]$$

Theorem 4.2.1 (Axes-aligned rectangles over $\mathbb{R}_{[0,1]}^2$ PAC learnability). It holds that:

- For every distribution \mathcal{D} ,
- For every error $0 < \varepsilon < \frac{1}{2}$,
- For every confidence $0 < \delta < \frac{1}{2}$,

if $m \geq \frac{4}{\varepsilon} \ln\left(\frac{4}{\delta}\right)$, then:

$$\mathcal{P}_{D \sim \mathcal{D}^m} \left[\text{error}_{\mathcal{D},T}(\mathcal{A}_{\text{BFP}}(T(D))) \leq \varepsilon \right] > 1 - \delta$$

where $D \sim \mathcal{D}^m$ is a sample of m data points (i.e. training data) and $T(\cdot)$ labels the input data wrt to the target rectangle T .

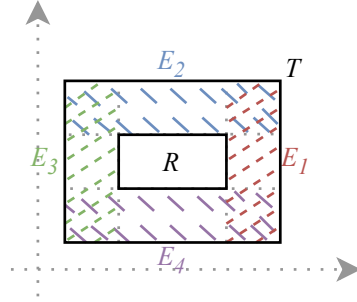
Proof. By definition, the error of \mathcal{A}_{BFP} is defined as:

$$\text{error}_{\mathcal{D},T}(R) = \mathcal{P}_{x \sim \mathcal{D}}[x \in (T \setminus R)]$$

where R is the predicted rectangle and T is the target rectangle.

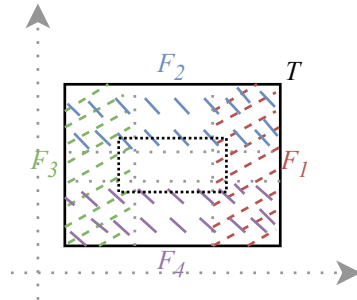
First, we need to prove some auxiliary lemmas:

1. Consider the space defined by $(T \setminus R)$ divided in four sections $E_1 \cup \dots \cup E_4 = (T \setminus R)$:



Consider the probabilistic event " $x \in E_i$ ". For the training data $x \sim \mathcal{D}$ this holds iff none of those points end up in E_i as, if a training point is in E_i , R would be bigger to include it and E_i would be smaller.

Now consider four other regions F_1, \dots, F_4 of the plane related to E_i but defined differently in such a way that $\mathcal{P}_{x \sim \mathcal{D}}[x \in F_i] = \frac{\varepsilon}{4}$. This can be achieved by expanding the E_i regions to take some area of the rectangle R .



Then, as E_i are smaller than F_i , it holds that:

$$\begin{aligned}\mathcal{P}_{x \sim \mathcal{D}}[x \in E_i] &\leq \frac{\varepsilon}{4} \Rightarrow \mathcal{P}_{x \sim \mathcal{D}}[x \in (T \setminus R)] \leq \varepsilon \\ &\Rightarrow \text{error}_{\mathcal{D},T}(R) \leq \varepsilon\end{aligned}\tag{4.1}$$

2. We want to prove that:

$$\begin{aligned}\forall i : \left(\begin{array}{l} \text{Some red points} \\ \text{in the training} \\ \text{data is in } F_i \end{array} \right) &\Rightarrow E_i \subseteq F_i \\ &\Rightarrow \mathcal{P}_{x \sim \mathcal{D}}[x \in E_i] \leq \mathcal{P}_{x \sim \mathcal{D}}[x \in F_i] \\ &\Rightarrow \mathcal{P}_{x \sim \mathcal{D}}[x \in E_i] \leq \frac{\varepsilon}{4} \quad \text{Def. of } \mathcal{P}_{x \sim \mathcal{D}}[x \in F_i] \\ &\Rightarrow \text{error}_{\mathcal{D},T}(R) \leq \varepsilon \quad \text{By Equation (4.1)}\end{aligned}\tag{4.2}$$

Now, we can prove the theorem:

$$\begin{aligned}m &\geq \frac{4}{\varepsilon} \ln \left(\frac{4}{\delta} \right) \Rightarrow \frac{\varepsilon \cdot m}{4} \geq \ln \left(\frac{4}{\delta} \right) \\ &\Rightarrow \ln(4) + \ln(e^{-\varepsilon/4})^m \leq \ln(\delta) \\ &\Rightarrow 4 \cdot (e^{-\varepsilon/4})^m \leq \delta \\ &\Rightarrow 4 \cdot \left(1 - \frac{\varepsilon}{4} \right)^m \leq \delta \quad e^x \text{ Taylor series} \\ &\Rightarrow \mathcal{P} \left[\exists i : \left(\begin{array}{l} \text{None of the points in the} \\ \text{training data occur in } F_i \end{array} \right) \right] \leq \delta \quad \begin{array}{l} \text{Since } \mathcal{P}_{x \sim \mathcal{D}}[x \in F_i] = \frac{\varepsilon}{4}, \\ \text{then } \mathcal{P}_{x \sim \mathcal{D}}[x \notin F_i] = 1 - \frac{\varepsilon}{4} \end{array} \\ &\Rightarrow \mathcal{P} \left[\forall i : \left(\begin{array}{l} \text{Some points in the train-} \\ \text{ing data occur in } F_i \end{array} \right) \right] > 1 - \delta \quad \text{Invert event} \\ &\Rightarrow \mathcal{P}[\text{error}_{\mathcal{D},T}(R) \leq \varepsilon] > 1 - \delta \quad \text{By Equation (4.2)}\end{aligned}$$

□

Corollary 4.2.1.1. The concept class of axis-aligned rectangles over $\mathbb{R}_{[0,1]}^2$ is efficiently PAC learnable.

5 VC dimension

Consistent learner Learning algorithm capable of perfectly predicting a concept.

Consistent learner

A learning algorithm \mathcal{L} is a consistent learner for \mathcal{C} if for all $n \geq 1$, for all $c \in \mathcal{C}_n$, \mathcal{L} outputs on input $((x_1, y_1), \dots, (x_m, y_m))$ a concept $h \in \mathcal{C}_n$ such that $h(x_i) = y_i$.

Remark. We are not interested in models that grow with the size of the training data (e.g. a model with a chain of if-else for each example).

Theorem 5.0.1 (Occam's razor). Let \mathcal{C} be a concept class and \mathcal{L} a consistent learner for \mathcal{C} . For all:

- $n \geq 1$,
- $c \in \mathcal{C}_n$
- Distributions \mathcal{D} over X_n ,
- $0 < \varepsilon < \frac{1}{2}$,
- $0 < \delta < \frac{1}{2}$,

if \mathcal{L} is given a sample of size m drawn from \mathcal{D} such that:

$$m \geq \frac{1}{\varepsilon} (\log |\mathcal{C}_n| + \log \frac{1}{\delta})$$

Then, \mathcal{L} is guaranteed to output h that with probability at least $1 - \delta$ satisfies $\mathbf{err}(h) \leq \varepsilon$. Furthermore, if \mathcal{L} is efficient and $\log |\mathcal{C}_n|$ is polynomial in n and the $\mathbf{size}(c)$, then \mathcal{C} is efficiently PAC learnable through \mathcal{L} .

Remark. This theorem considers the cardinality of the concept class.

Let $S \subseteq X$ be a finite set of instances. For a concept $c : X \rightarrow \{0, 1\}$, we can consider the restriction $c|_S : S \rightarrow \{0, 1\}$ of c to S where $c|_S(x) = c(x)$ for every $x \in S$.

Let's define the following:

$$\Pi(S) = \{c|_S \mid c \in \mathcal{C}\}$$

The set Π_S is in bijective correspondence with the set of vectors in $\{0, 1\}$ of length $|S|$. What matters of $c|_S$ is $(c(x_1), \dots, c(x_{|S|}))$ where $S = \{x_1, \dots, x_{|S|}\}$.

It holds that:

$$|\Pi_S| \leq |\{(b_1, \dots, b_{|S|}) \mid b_i \in \{0, 1\}\}| = 2^{|S|}$$

Shattered set A finite set $S \subseteq X$ is shattered by \mathcal{C} if:

$$|\Pi_{\mathcal{C}}(S)| = 2^{|S|}$$

In other words, all possible dichotomies over S can be realized by \mathcal{C} .

Vapnik-Chervonenkis dimension The VC dimension of \mathcal{C} (denoted $\text{VCD}(\mathcal{C})$) is the cardinality d of the largest finite set S which is shattered by \mathcal{C} .

Vapnik-
Chervonenkis
dimension

If \mathcal{C} shatters arbitrarily big sets, then the VC dimension of \mathcal{C} is $+\infty$.

Example (Intervals). Let $X = \mathbb{R}$ and $\mathcal{C} = \{c : X \rightarrow \{0, 1\} \mid c \text{ is a closed interval}\}$.

It holds that $\text{VCD}(\mathcal{C}) = 2$. A set $S \subset \mathbb{R}$ such that $|S| = 2$ can always be shattered.

A set S such that $|S| = 3$ can never be shattered.

Example (Rectangles). If $|S| = 4$, it is possible to capture all the possible dichotomies for at least an S . If $|S| = 5$, this is not possible.

It holds that $\text{VCD}(\mathcal{C}) = 4$.

Theorem 5.0.2 (Upper bound). Let \mathcal{C} be a concept class with $\text{VCD}(\mathcal{C}) = d$ where $1 \leq d < +\infty$, \mathcal{L} be a consistent learner for \mathcal{C} . Then, for every $0 < \varepsilon < \frac{1}{2}$ and $0 < \delta < \frac{1}{2}$, \mathcal{L} is a PAC learning algorithm for \mathcal{C} if:

$$m \geq k_0 \left(\frac{1}{\varepsilon} \log \frac{1}{\delta} + \frac{d}{\varepsilon} \log \frac{1}{\varepsilon} \right)$$

Theorem 5.0.3 (Lower bound). Let \mathcal{C} be a concept class with $\text{VCD}(\mathcal{C}) \geq d$ where $d \geq 25$ (it was proved on neural networks). Then, every PAC learning algorithm for \mathcal{C} requires a number of samples of at least:

$$\max \left\{ \frac{d-1}{32\varepsilon}, \frac{1}{4\varepsilon}, \log \frac{1}{4\delta} \right\}$$

Remark. Note that this value depends on the VC dimension d . In theory, it should be very difficult to learn neural networks with large VCD.

In practice, it usually happens that the dataset has particular distributions