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 12

4.2 Axes-aligned rectangles over $\mathbb{R}^2_{[0,1]}$ 13

5 Computational learning theory extras **16**

5.1 Occam's razor 16

5.2 VC dimension 17

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 $\llcorner x \lrcorner$ or simply x.

Encoding

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

Task

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

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

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 $f : \mathbb{N} \to \mathbb{N}$ 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 $f : \mathbb{N} \to \mathbb{N}$ 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 $f : \mathbb{N} \to \mathbb{N}$ 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

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

- **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

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

- **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)

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):

- 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:

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

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

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

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

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

- 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)

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:

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

where C_t is a final configuration for y.

Computation (function)

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

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

If this holds, f is a computable function.

Computation in time T

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

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

Decidability in time T

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

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

It is therefore possible to encode δ into a binary string and consequently create an encoding $\llcorner \mathcal{M} \lrcorner$ 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 $x = \llcorner \mathcal{M} \lrcorner$.

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

Universal Turing Machine (UTM)

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:

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

where $\mathcal{M}_\alpha$ is the TM defined by α.

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

2.4 Computability

2.4.1 Undecidable functions

Uncomputable functions

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

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:

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

Therefore, uc cannot be computed. □

Halting problem

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

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

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 $\mathcal{M}_{\texttt{halt}}$ that decides it.

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

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

This results in a contradiction:

- $\mathcal{M}_{uc}(\llcorner \mathcal{M}_{uc} \lrcorner) = 1 \Leftarrow \mathcal{M}_{\texttt{halt}}(\llcorner \mathcal{M}_{uc} \lrcorner, \llcorner \mathcal{M}_{uc} \lrcorner) = 0 \iff \mathcal{M}_{uc}(\llcorner \mathcal{M}_{uc} \lrcorner)$ diverges

- $\mathcal{M}_{\texttt{halt}}(\llcorner \mathcal{M}_{uc} \lrcorner, \llcorner \mathcal{M}_{uc} \lrcorner) = 1 \Rightarrow \mathcal{M}_{uc}$ 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 $\llcorner \mathcal{M} \lrcorner \in \mathcal{L}$ and the TM $\mathcal{N}$ computes the same function of $\mathcal{M}$, then $\llcorner \mathcal{N} \lrcorner \in \mathcal{L}$.

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 $\mathcal{L} = \varnothing$ 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 $T : \mathbb{N} \to \mathbb{N}$ and $\mathcal{L}$ be a language. $\mathcal{L}$ is in $\mathbf{DTIME}(T(n))$ iff there exists a TM that decides $\mathcal{L}$ in time $O(T(n))$.

Deterministic time (**DTIME**)

Polynomial time (P) The class $\mathbf{P}$ contains all the tasks computable in polynomial time:

Polynomial time (**P**)

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

> **Remark.** $\mathbf{P}$ is closed to various operations on programs (e.g. composition of programs)

> **Remark.** In practice, the exponent is often small.

> **Remark.** $\mathbf{P}$ 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 $\mathbf{P}$ 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 $T : \mathbb{N} \to \mathbb{N}$ and $f : \{0,1\}^* \to \{0,1\}^*$. f is in $\mathbf{FDTIME}(T(n))$ 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 $\mathbf{FP}$ 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 $\mathbf{EXP}$ and $\mathbf{FEXP}$ classes are defined as:

Exponential time (**EXP**/**FEXP**)

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

Theorem 3.2.1. The following hold:

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

3.3 NP class

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

Certificate

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

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:

Nondeterministic TM (NDTM)

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

Nondeterministic time (**NDTIME**)

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

Complexity class NP

Complexity class **NP**

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

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

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

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

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($x \in \{0,1\}^*$):
    foreach $y \in \{0,1\}^{p(|x|)}$:
        if $\mathcal{M}(x,y) == 1$:
            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

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

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 $f : \{0,1\}^* \to \{0,1\}^*$ 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 $f, g : \{0,1\}^* \to \{0,1\}^*$ 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

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

Theorem 3.3.3.

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

Proof.

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

   $\mathsf{P} \subseteq \mathsf{NP}$) Proved in Theorem 3.3.1.

   $\mathsf{NP} \subseteq \mathsf{P}$) Let $\mathcal{H}$ be a language in $\mathbf{NP}$. As $\mathcal{L}$ is $\mathbf{NP}$-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 $\mathbf{P}$.

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

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

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

□

Theorem 3.3.4. The problem `TMSAT` of simulating any TM is $\mathbf{NP}$-complete:

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

Theorem 3.3.5 (Cook-Levin)**.** The following languages are $\mathbf{NP}$-complete:

Cook-Levin theorem

$$\texttt{SAT} = \{\llcorner F \lrcorner \mid F \text{ is a satisfiable CNF}\}$$
$$\texttt{3SAT} = \{\llcorner F \lrcorner \mid F \text{ is a satisfiable 3CNF}\}$$

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.

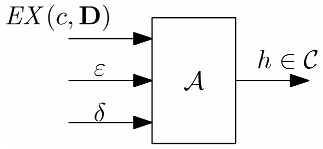
> **Remark.** A learning algorithm is designed to learn concepts from a concept class neither knowing the target concept nor its data distribution.

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

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

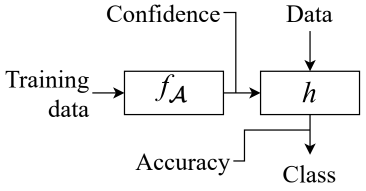


Figure 4.1: General idea of a learning algorithm $\mathcal{A}$ computed as a function $f_\mathcal{A}$

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:

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

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

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 $\frac{1}{\delta}$.

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

Representation class

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 $\texttt{size}(c)$ bits.

> **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, $\texttt{size}(c)$, $\frac{1}{\varepsilon}$ 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 **NP** $\neq$ **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:

$$\underset{\substack{\text{Instance of an }\textbf{NP}\text{-complete} \\ \text{problem (e.g. graph 3-coloring)}}}{\alpha \in \{0,1\}^*} \overset{f}{\mapsto} \underset{\text{Training set for 3DNF}}{\mathcal{S}_\alpha}$$

□

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

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

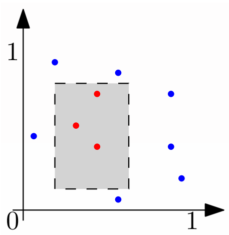


Figure 4.2: Example of problem instance. The gray rectangle is the target concept, red dots are positive data points and blue dots are negative data points.

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

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

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

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

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

Theorem 4.2.1 (Axes-aligned rectangles over $\mathbb{R}^2_{[0,1]}$ 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:

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

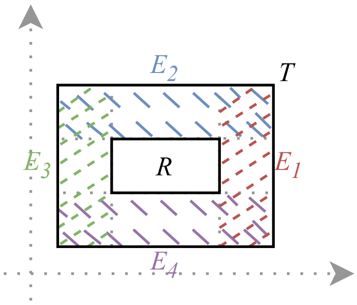
where $D \sim \mathcal{D}^m$ is a sample of m data points (i.e. training data) and $T(\cdot)$ labels the input data wrt to the target rectangle T.

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

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

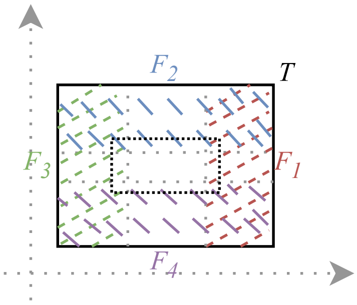
where R is the predicted rectangle and T is the target rectangle.
First, we need to prove some auxiliary lemmas:

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



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 $F_1, \ldots, F_4$ 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:

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

2. We want to prove that:

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

Now, we can prove the theorem:

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

□

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

5 Computational learning theory extras

5.1 Occam's razor

Consistent learner Given a concept class $\mathcal{C}$, a learning algorithm $\mathcal{L}$ is a consistent learner for $\mathcal{C}$ if for all:

Consistent learner

- $n \geq 1$,
- $m \geq 1$,
- $c \in \mathcal{C}^n$ ($\mathcal{C}^n$ is the concept class over the instance space X^n),

$\mathcal{L}$ outputs on input $((x_1, y_1), \dots, (x_m, y_m))$ a concept $h \in \mathcal{C}^n$ such that $h(x_i) = y_i$. In other words, $\mathcal{L}$ is capable of perfectly predicting the training concept.

> **Remark.** A simple but not interesting family of learning algorithms is that of the models that learn the training data as a chain of `if-else`.
> We are not interested in models that grow with the size of the training data.

Efficient consistent learner A learning algorithm $\mathcal{L}$ is an efficient consistent learner for $\mathcal{C}$ if it is a consistent learner for $\mathcal{C}$ and works in polynomial time in n, $\texttt{size}(c)$ and m.

Efficient consistent learner

Theorem 5.1.1 (Occam's razor)**.** Let $\mathcal{C}$ be a concept class and $\mathcal{L}$ a consistent learner for $\mathcal{C}$. It holds that for all:

Occam's razor

- $n \geq 1$,
- $c \in \mathcal{C}^n$
- 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:

$$m \geq \frac{1}{\varepsilon}\left(\log(|\mathcal{C}^n|) + \log\left(\frac{1}{\delta}\right)\right)$$

then $\mathcal{L}$ is guaranteed to output a concept h that satisfies $\texttt{err}(h) \leq \varepsilon$ with a probability of at least $(1 - \delta)$.

Corollary 5.1.1.1. If $\mathcal{L}$ is an efficient consistent learner and $\log(|\mathcal{C}^n|)$ is polynomial in n and $\texttt{size}(c)$, then $\mathcal{C}$ is efficiently PAC learnable through $\mathcal{L}$.

> **Remark.** This implies that small (i.e. polynomial) concept classes can be efficiently learned.

Not required for the exam

5.2 VC dimension

Concept class restriction Given an instance space X, a finite subset $S \subseteq X$ and a concept $c : X \to \{0, 1\}$, the restriction of c to S is the concept $c|_S : S \to \{0, 1\}$ such that:

Concept class restriction

$$\forall x \in S : c|_S(x) = c(x)$$

The set of all the restrictions of a concept class $\mathcal{C}$ to S is denoted as:

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

> **Remark.** It holds that:
>
> $$\left|\Pi_{\mathcal{C}}(S)\right| \leq \left|\{(b_1, \dots, b_{|S|}) \mid b_i \in \{0, 1\}\}\right| = 2^{|S|}$$

Shattered set Given an instance space X and a concept class $\mathcal{C}$, a finite set $S \subseteq X$ is shattered by $\mathcal{C}$ if:

Shattered set

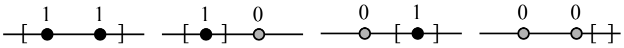
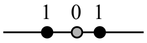
$$|\Pi_{\mathcal{C}}(S)| = 2^{|S|}$$

In other words, all possible dichotomies (division into two partitions) over S can be done by $\mathcal{C}$.

Vapnik-Chervonenkis dimension The VC dimension of a concept class $\mathcal{C}$ (denoted as $\texttt{VCD}(\mathcal{C})$) is the cardinality d of the largest finite set S which is shattered by $\mathcal{C}$. In other words, $\texttt{VCD}(\mathcal{C}) = d$ if there exists a set S such that $|S| = d$ and it 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 $\mathcal{C} = \{c : X \to \{0, 1\}\}$. Learning a concept $c \in \mathcal{C}$ consists of learning an interval that includes the reals labeled with 1 and excludes the ones labeled with 0.
> It holds that $\texttt{VCD}(\mathcal{C}) = 2$ as a set $S \subseteq \mathbb{R}$ such that $|S| = 2$ can always be shattered.
>
> 
>
>
> Figure 5.1: Solutions for all the possible cases with $|S| = 2$
>
> A set S such that $|S| = 3$ can never be shattered.
>
> 
>
>
> Figure 5.2: Instance with $|S| = 3$ in which an interval cannot be found

> **Example** (Rectangles)**.** Let $X = \mathbb{R}^2$ and $\mathcal{C} = \{c : X \to \{0, 1\}\}$. Learning a concept $c \in \mathcal{C}$ consists of learning an axis-aligned rectangle that includes the points labeled with 1 and excludes the ones labeled with 0.
> It holds that $\texttt{VCD}(\mathcal{C}) = 4$ as it is possible to capture all the possible dichotomies for at least an S such that $|S| = 4$.

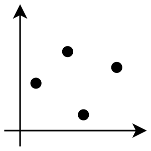


Figure 5.3: Case with $|S| = 4$ in which any subset of the points can be enclosed within a rectangle that includes only those points

For $|S| = 5$, there is at least an instance for any S in which a rectangle is not possible.

Sample complexity upper bound

Theorem 5.2.1 (Sample complexity upper bound)**.** Let $\mathcal{C}$ be a concept class with $\texttt{VCD}(\mathcal{C}) = d$ where $1 \leq d < +\infty$ and $\mathcal{L}$ a consistent learner for $\mathcal{C}$. Then, for every $0 < \varepsilon < \frac{1}{2}$ and $\delta \leq \frac{1}{2}$, $\mathcal{L}$ is a PAC learning algorithm for $\mathcal{C}$ if it is provided with an amount m of samples such that:

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

for some constant k_0.

Sample complexity lower bound

Theorem 5.2.2 (Sample complexity lower bound)**.** Let $\mathcal{C}$ be a concept class with $\texttt{VCD}(\mathcal{C}) \geq d$ where $d \geq 25$[1]. Then, every PAC learning algorithm for $\mathcal{C}$ requires an amount m of samples such that:

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

Remark. Note that this value depends on the VC dimension d which, in theory, implies that it should be very difficult to learn neural networks with large `VCD` as it would require lots of data.
In practice, it usually happens that the dataset has particular distributions or properties.

`<end of course>`

[1] This comes from the original proof that was done on neural networks. With a slightly modified argument, the theorem holds for $d \geq 2$.