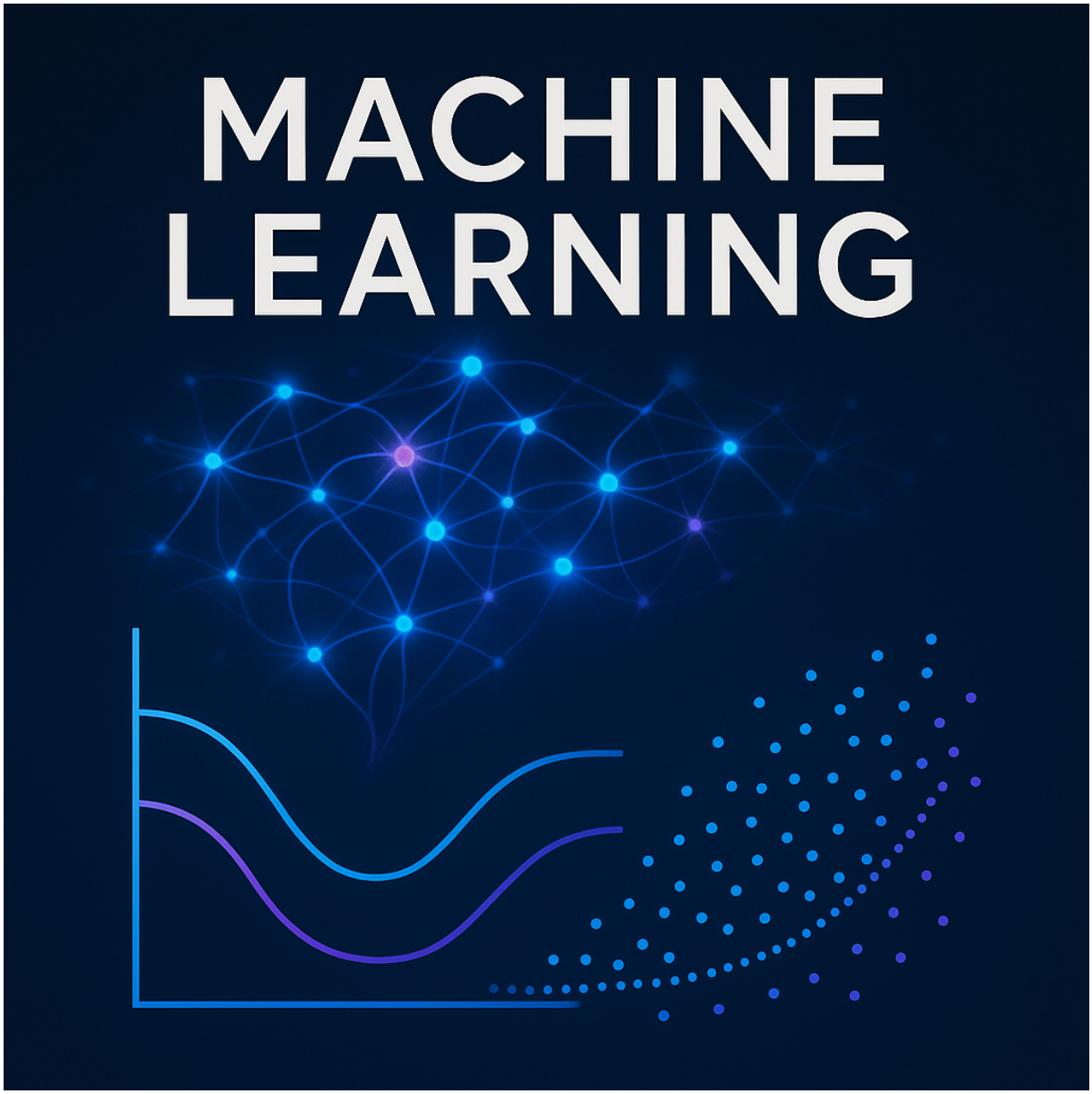
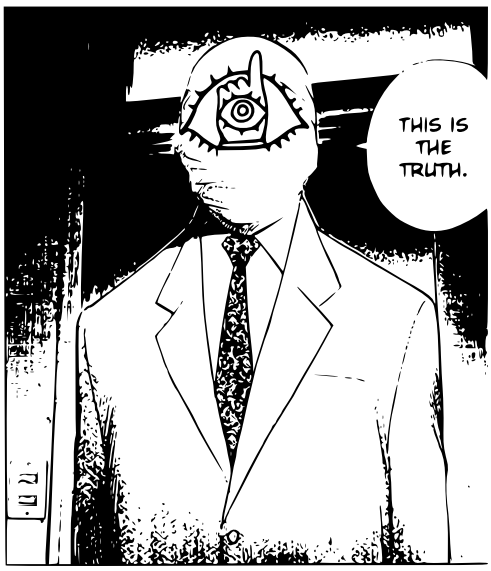
Marco Casu



SAPIENZA
UNIVERSITÀ DI ROMA

Faculty of Information Engineering, Computer Science and Statistics
Department of Computer, Control and Management Engineering
Master's degree in Artificial Intelligence and Robotics

This document summarizes and presents the topics for the Machine Learning course for the Master's degree in Artificial Intelligence and Robotics at Sapienza University of Rome. The document is free for any use. If the reader notices any typos, they are kindly requested to report them to the author.



CONTENTS

1 Introduction **3**
1.1 Basic Definition of a ML Problem 3
1.2 Types of Machine Learning Problems 5

CHAPTER

1

INTRODUCTION

1.1 Basic Definition of a ML Problem

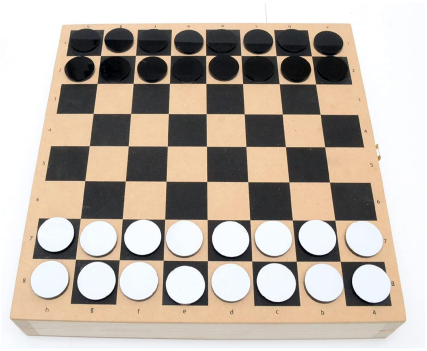
In this chapter we will introduce the basics of what is a machine learning problem, giving a mathematical definition. informally, with machine learning we define the use of knowledge (data) to improve the performance of a given program, using past experiences.

Generally, we use machine learning to solve problems with no deterministic solutions, trying to find an approximate one (such as recognizing what animal is represented in a given photo).

Usually, a machine learning problem consists in three main component:

- T : the given task
- P : a performance metric
- E : the past experiences (the data)

Let's see an example, we want to model a program capable of playing *Checkers*.



The task T is to play the game, the performance metric P measure the ratio of win in a tournament, and the experiences is given by the past match. We can create this matches by letting the computer play against himself, or by making it play against a human. We can consider two types of target functions that models the behavior of the model:

- $ChooseMove : Board \to Move$
- $V : Board \to \mathbb{R}$

Board is the set of the possible board configurations, move is the set of feasible moves. The image of the function V should represents the validity of a board in the following sense:

- if $b \in Boards$ is a configuration that represents the win of the player, then $V(b) = 1$
- else if $b \in Boards$ is a configuration that represents the win of the opponent, then $V(b) = -1$
- else if $b \in Boards$ is a configuration that represents a draw, then $V(b) = 0$
- else, $V(b) = V(b')$ where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

The function V can be used to predict the next move by considering all possible boards that can be obtained from the feasible moves. We focus on the function V, this should be an optimal model, but is not computable, because we cannot tell if a play is "optimal" (this is the goal of the model), so we want to consider a function that approximate the behavior of V.

We consider a function $\hat{V} : Board \to \mathbb{R}$ defined as follows:

$$\hat{V}(b) = \sum_{i=0}^{6} w_i f_i(b) \tag{1.1}$$

where $\mathbf{w} = (w_i)$ is real coefficients, and the functions f_i represents some features of the given board:

- $f_1(b)$: number of black pieces on b
- $f_2(b)$: number of red pieces on b
- ecc...

We don't know if some features are useful to predict which move is optimal, is important that this features model the knowledge of the *domain* (in this case, the game of Checkers).

with *learning the function* $\hat{V}$, we intend to finds the coefficients $\mathbf{w}$ which make the function $\hat{V}$ more similar to V as possible. There are various method to find these coefficients.

Let's introduce some notation:

- with V we define the **target function** (always unknown and uncomputable).
- with $\hat{V}$ we define the **learned function**, the approximation of V that we want to find.
- with $V_{train}(b)$ we define the value of V obtained at b, where b is a part of a data set that is given. We will use the values of V_{train} on the data set to synthesize $\hat{V}$.
- with D we define the given data set:

$$D = \bigcup_{i=1}^{n} \{b_i, V_{train}(b_i)\} \tag{1.2}$$

  n is the number of the available data.

The iterative method given in the Algorithm 1 is an informal example of how we can find the values for $\mathbf{w}$. In this case c is a small constant, usually in $(0, 1]$, to moderate the rate of learning.

The function $error(b)$ is computable only on the sample b given in the dataset B, the goal of the method is to converge to a local minimum for the function

$$\sum_{b_i \in D} error(b_i). \tag{1.3}$$

Once we synthesize $\hat{V}$ with this method, we can make the model play against human and track the result to use it as an additional dataset. The diagram in image 1.1 represents the process of designing an artificial intelligence agent.

Algorithm 1 LMS weight update rule

Require: D, V_{train}, k
  initialize **w** with small random values
  for k times **do**
    select a sample b from D
    $error(b) = V_{train}(b) - \hat{V}(b)$
    for each feature i **do**
      $w_i \leftarrow w_i + c \cdot f_i(b) \cdot error(b)$
    end for
  end for

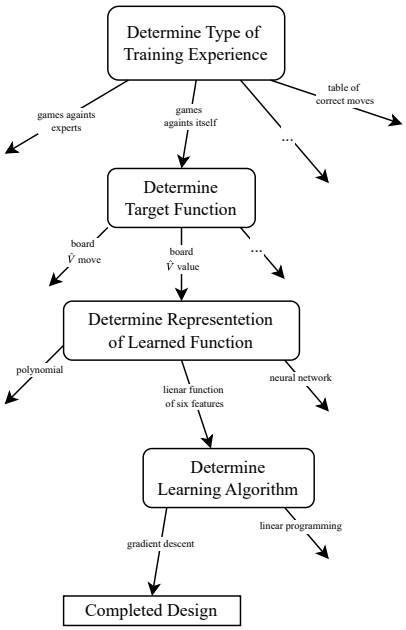


Figure 1.1: Design Choices

1.2 Types of Machine Learning Problems

There are different types of machine learning problems:

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
- Reinforcement Learning

Definition 1 *Given a function $f : X \to Y$, and a training set $X_D \subset X$ containing information about f, **learning** the function f means computing an approximated function $\hat{f}$ such that is much close as possible to f on X*

$$\hat{f}(x) \simeq f(x), \quad x \in X \tag{1.4}$$

more formally, we want to find $\hat{f}$ that minimize the following integral:

$$\int_X |\hat{f}(x) - f(x)| dx \tag{1.5}$$

if $X \subseteq \mathbb{R}^n$ and is uncountable, and

$$\sum_{x \in X} |\hat{f}(x) - f(x)| \tag{1.6}$$

if X is countable.

This is not a simple problem, f is not computable, so the difference $f - \hat{f}$, and X is usually uncountable ora big set, way bigger than the training set X_D.

Machine learning problems can be classified in terms of the input data set D, given a target function $f : X \to Y$ a problem is

- a **supervised learning** problem if $D = \bigcup_{i=1}^{n} \{(x_i, y_i)\} \subset X \times Y$.
- an **unsupervised learning** problem if $D = \bigcup_{i=1}^{n} \{(x_i)\} \subset X$.
- a **reinforcement learning** problem, the condition on the input dataset will be discussed later.

The problems can also be classified in terms of the target function $f : X \to Y$

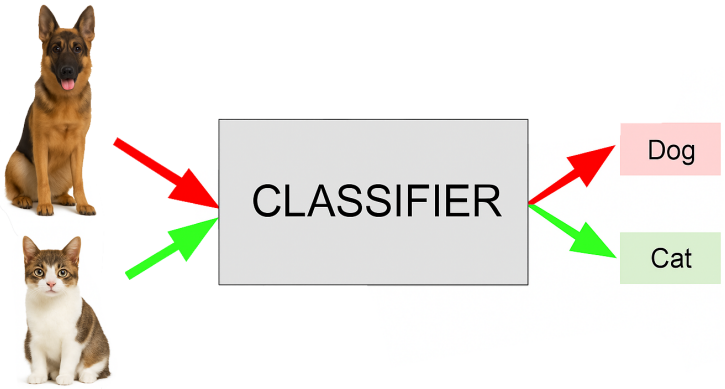
$$X = \begin{cases} A_1 \times \cdots \times A_m, \ A_i \text{ finite set} & \textbf{(Finite Discrete Problem)} \\ \mathbb{R}^n & \textbf{(Continuous)} \end{cases}$$

$$Y = \begin{cases} \mathbb{R}^k & \textbf{(Regression)} \\ \{C_1, C_2 \times C_k\} & \textbf{(Classification)} \end{cases}$$

special case (**Concept Learning**):

$$X = A_1 \times \cdots \times A_m, \ A_i \text{ finite set}$$
$$Y = \{0, 1\}$$

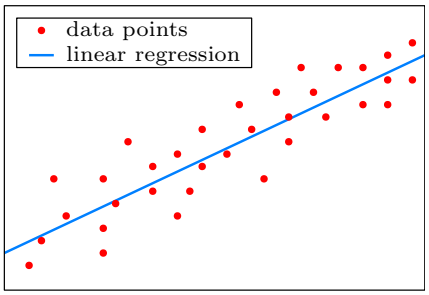
Classification problems is also known as *Pattern Recognition Problems*, the goal is to return the class to which a specific instance belong.



Some examples are:

- face/object/character recognition
- speech/sound recognition
- medical diagnosis
- document classification.

Regression problems consists in approximating real valued functions.

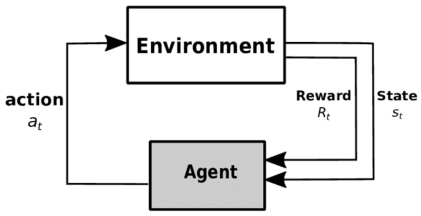


Unsupervised learning discovers data patterns without a specific output. The main goal is to understand what's normal in a dataset. *Clustering* is a key technique that groups similar data points. Applications include customer segmentation, image compression, and bioinformatics motif learning.

Reinforcement learning consists in learning a policy, which is a strategy that tells the agent what action to take in any given situation (or state) to maximize its long-term reward. This is often represented as a function that maps a state to an action. Unlike supervised learning, where the model is trained with labeled examples, in reinforcement learning agents learn through a process of trial and error.

They don't have a correct answer to guide them. Instead, they receive sparse and time-delayed rewards. This means the feedback (the reward) for a good action might not come immediately, and it might be a simple, numerical value (like +1 for winning a game or -1 for losing). Some examples are:

- Game playing: Think of programs that have learned to play chess, Go, or video games better than humans.
- Robotic tasks: A robot learning to navigate a room, pick up an object, or perform a specific manufacturing task.
- Any dynamical system with an unknown or partially known model: This is a broad category that includes things like optimizing traffic flow, managing a power grid, or controlling financial trading strategies.



In concept learning, the output consists in only two classes, the target function $c : X \to \{0, 1\}$ maps any kind of input in one of two distinct values. The following example model a program that predict if is a good day to play Tennis, the input set is

$$X = Day \times Outlook \times Temperature \times Humidity \times Wind$$

the output set is $PlayTennis = \{\text{Yes, No}\}$. An example of data samples:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Usually, we call **Hypothesis** a possible learned function h, and we define H as the **Hypothesis space**, such space contains all possible function that can be learnt (all possible approximation of the target function). In this terms, a learning problem is described as a search in the hypothesis space using the given dataset D, that aims to find the best possible approximation h^*:

$$h^* = \arg\max_{h \in H} Performance(h, D) \tag{1.7}$$