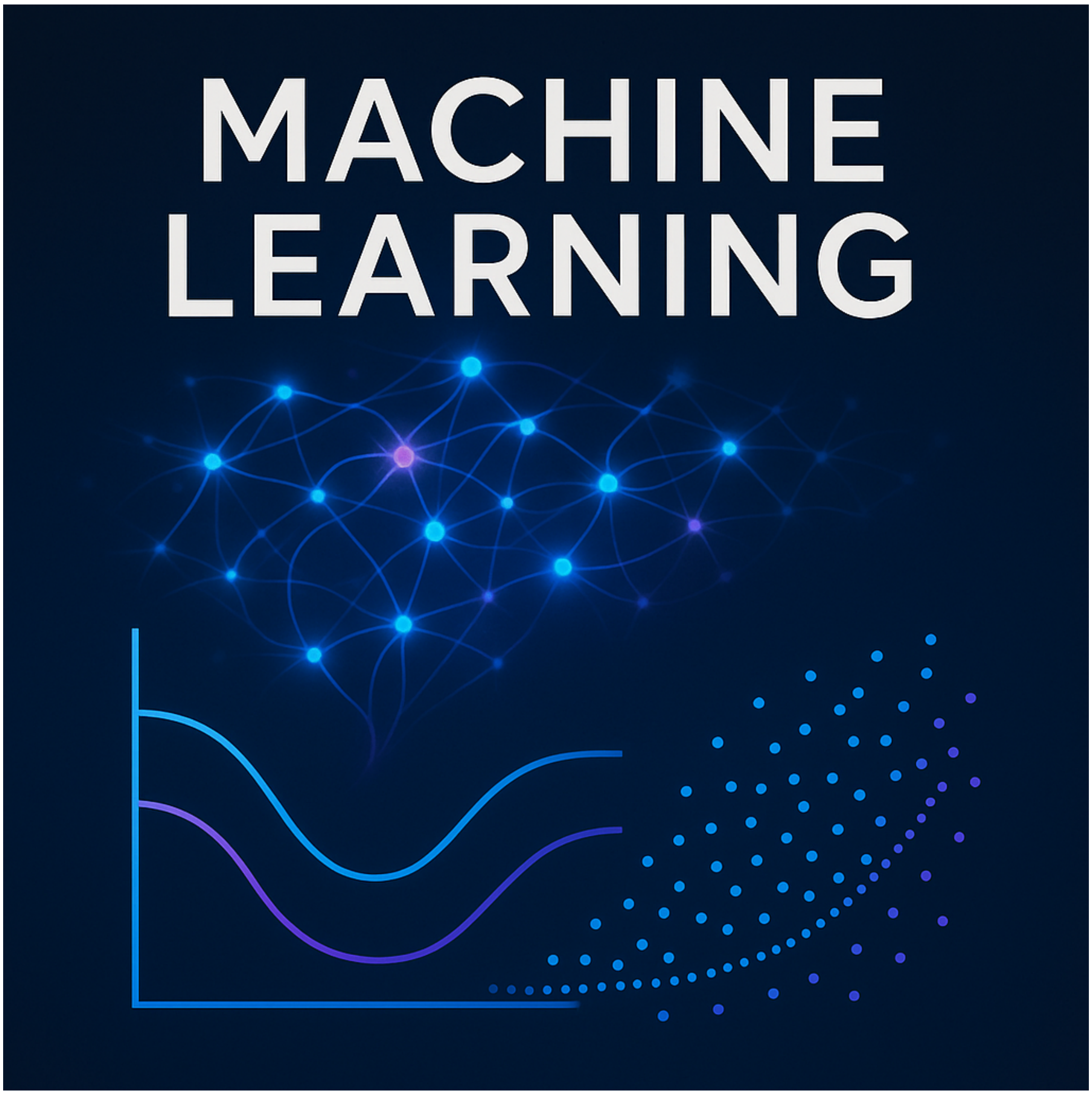
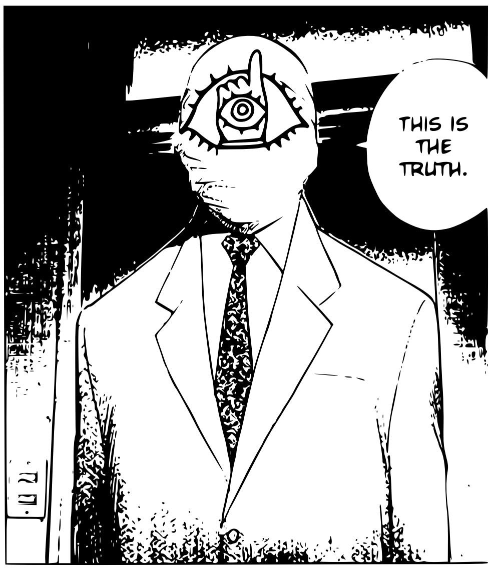
Marco Casu



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

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 $\mathbf{w}$ 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
```

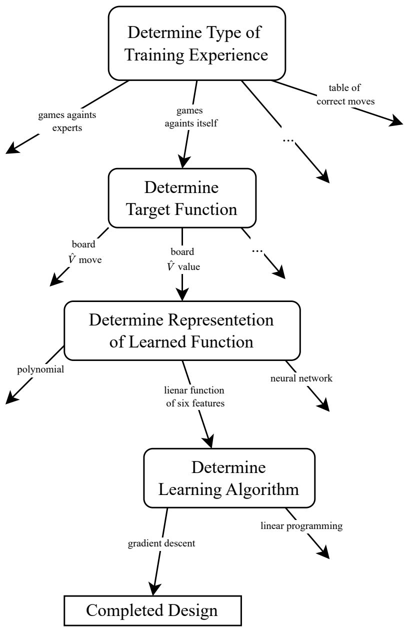


Figure 1.1: Design Choices

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*

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

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 $f : X \to Y$

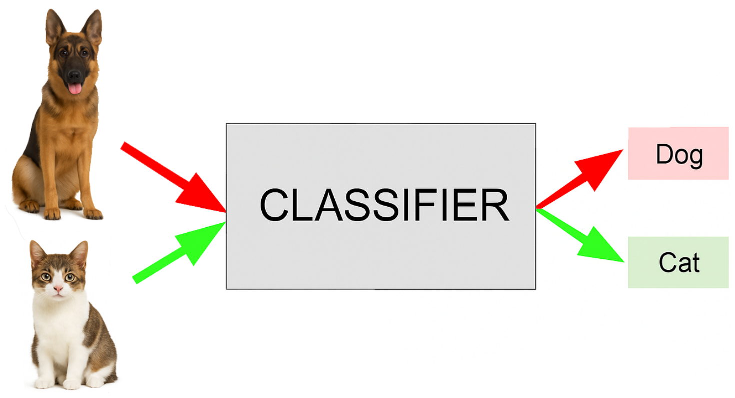
$$X = \begin{cases} A_1 \times \cdots \times A_m,\ A_i \text{ finite set} & \textbf{(Finite Discrete Problem)} \\ \mathbb{R}^n & \textbf{(Continuous)} \end{cases}$$

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

special case (**Concept Learning**):

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

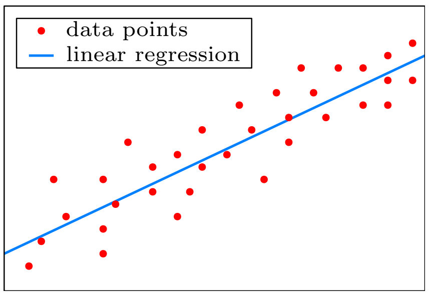
Classification problems is also known as *Pattern Recognition Problems*, the goal is to return the class to which a specific instance belong.



Some examples are:

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

Regression problems consists in approximating real valued functions.

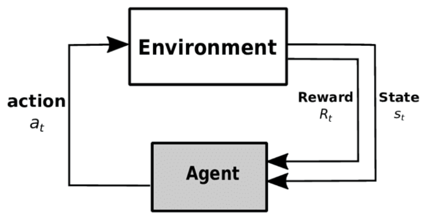


Unsupervised learning discovers data patterns without a specific output. The main goal is to understand what's normal in a dataset. *Clustering* is a key technique that groups similar data points. Applications include customer segmentation, image compression, and bioinformatics motif learning.

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

1.3 Performance Evaluation

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^*:

$$h^* = \arg\max_{h \in H} Performance(h, D) \tag{1.5}$$

1.3.1 Concept Learning

We focus now on the concept learning (binary classification), let's consider a target function

$$c : X \to \{0, 1\} \tag{1.6}$$

let $D \subset \{X, Y\}$ to be the sample set

$$D = \bigcup_{i=1}^{n} \{(x_i, c(x_i))\} \tag{1.7}$$

we denote X_D the point of X in the sample set. For now, we assume that the sample set does not have noise:

- noise dataset $D = \bigcup_{i=1}^{n} \{(x_i, c(x_i) + \varepsilon_i)\}, \quad \varepsilon_i \in \mathbb{R}$
- perfect dataset $D = \bigcup_{i=1}^{n} \{(x_i, c(x_i))\}$

in general, the dataset is noisy. We want to find a function $\hat{f}$ that approximate c, as we just said, the hypothesis space H is the set of all hypothesis h.Each hypothesis is a possible solution to the problem

(1.5), it is possible to compare an hypothesis h with c on a sample in X_D.

Definition 2 *Given a target function c and a set of sample point X_D, an hypothesis h is* ***consistent*** *if*

$$h(x) = c(x), \quad \forall x \in X_D. \tag{1.8}$$

The **Version Space** $VS_{H,D}$ is the set of all consistent hypothesis:

$$VS_{H,D} = \{h \ : \ h(x) = c(x), \quad \forall x \in X_D\} \subset H. \tag{1.9}$$

A solution that does not lie in the version space is probably not a good solution.

Let's consider an example, let c to be the following target function

$$c : \mathbb{N} \to \{-, +\} \tag{1.10}$$

the dataset is

$$D = \{(1, +), (3, +), (5, +), (6, -), (8, -), (10, -)\} \tag{1.11}$$

given the hypothesis space H, we consider four different hypothesis

$$\begin{aligned} h_1(n) = + &\iff n \text{ is odd} \\ h_2(n) = + &\iff n \leq 5 \\ h_3(n) = + &\iff n \text{ is either 1 or prime} \\ h_4(n) = + &\iff n \in \{1, 3, 5\} \end{aligned}$$

if $n = 11$ we have

$$\begin{aligned} h_1(11) &= + \\ h_2(11) &= - \\ h_3(11) &= + \\ h_4(11) &= - \end{aligned}$$

we can't tell which of the four hypothesis is better than the others. Let's now consider a new hypothesis space H', defined as the power set of H

$$H' = \mathcal{P}(H) = \{I \ : \ I \subseteq H\} \tag{1.12}$$

The set H' contains more information than H, let $\theta \in H'$, we define the value of θ on x in a different way

$$\theta = \bigcup_i \{h_i\} \tag{1.13}$$

$$\theta(x) = \text{maj} \bigcup_i \{h_i(x)\} \tag{1.14}$$

where the function maj returns the element that is more frequent in a set. It's important to know that, if you have a set where two different items appear the exact same number of times, and no other item appears more often than either of them, then the majority element can't be identified, and the value of the function maj is undefined.

With the hypothesis space H', the point $n = 11$ can't be classified

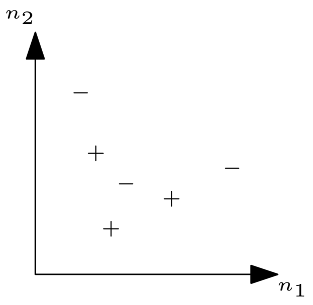
$$\theta(11) = \text{maj}\{h_1(11), h_2(11), h_3(11), h_4(11)\} = \text{maj}\{+, -, +, -\}. \tag{1.15}$$

Even if H' is more powerful than H, this hypothesis space can't classify an element that can be classified in H, this problem is called overfitting.

Now let's consider the following target function

$$c : \mathbb{N}^2 \to \{+, -\} \tag{1.16}$$

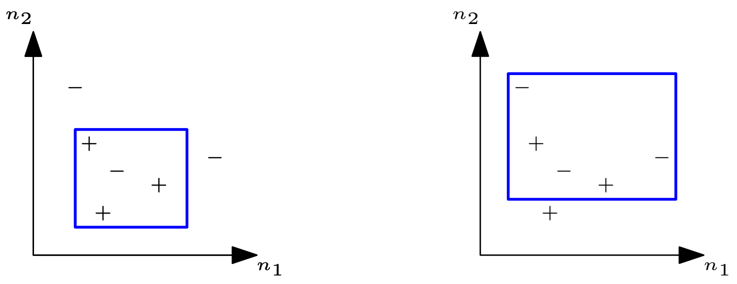
with the following dataset D that can be plotted on a 2D plane:



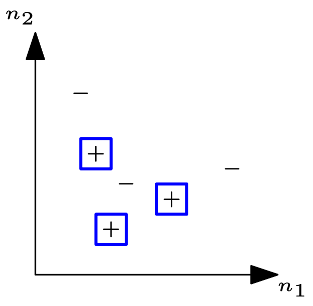
We consider H as the set of the functions that, assigns at all the points in a specific rectangle the $+$ value, and the $-$ value for all the points outside.

$$h \in H \iff \Big(\exists R := \{(x,y) \ : \ a \le x \le b \wedge c \le y \le d\} : h(x,y) = + \iff (x,y) \in R\Big)$$

For that data samples, does not exists a consistent hypothesis, because does not exists a rectangle that contains all the $+$ value point, letting outside the $-$ value points.



If we consider the hypothesis space $H' = \mathcal{P}(H)$, geometrically, we can represent a function by a finite number of rectangles, in this case consistent hypothesis are allowed, but no $-$ value points can be predicted.



Even in this example, a powerful representation of the hypothesis space lead to less generalization. This is known (as previously said) as **overfitting**, when the hypothesis space is not powerful enough, the problem could not be represented well, this is known as **underfitting**.

If n is a number that determines how big an hypothesis space, we can observe the following trend about the performance of a model.

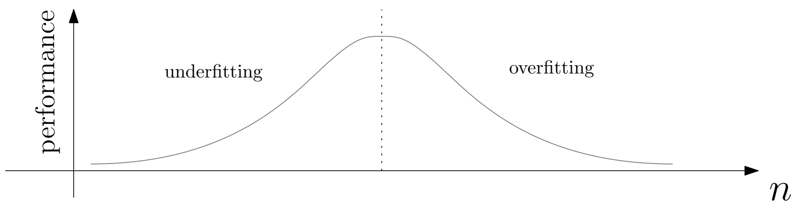


Figure 1.2: overfitting and underfitting trend

1.3.2 Error Estimation and Performance Metric

We want to define a metric for the performance of a model, to estimate how a given hypothesis h is good respect to the others. We have to introduce some statistical methods, let's consider a target function

$$f : X \to Y \tag{1.17}$$

where Y is countable (classification problem). Let $\mathcal{D}$ to be a probability distribution over X.

Definition 3 *If X is a countable subset of $\mathbb{R}^n$, $\mathcal{D}$ is a probability distribution on X if*

$$\forall x \in X, \quad \mathcal{D}(x) \in [0,1] \tag{1.18}$$

and

$$\sum_{x\in X} \mathcal{D}(x) = 1$$

if X is an uncountable subset of $\mathbb{R}^n$, $\mathcal{D}$ is a probability distribution on X if

$$\forall x \in X, \quad \mathcal{D}(x) \geq 0 \tag{1.19}$$

$$\int_X \mathcal{D}(x)dx = 1$$

Definition 4 *Given a target function $f : X \to Y$, an hypothesis h and a probability distribution $\mathcal{D}$ on X, the **true error** is the probability that h will misclassify an instance $x \in X$ drawn according to the distribution $\mathcal{D}$:*

$$\text{error}_{\mathcal{D}}(h) = \mathbb{P}_{x\sim\mathcal{D}}(f(x) \neq h(x)) \tag{1.20}$$

Note: With $x \sim \mathcal{D}$, we mean that x was extracted from X according to the probability distribution $\mathcal{D}$. We remind that, if $\mathcal{D}(x) = p$, then, the probability of extracting x from X by taking a random element is p.

Since we can't compute f for all $x \in X$, the true error can't be computed. We need to define an estimation of the error given by the sample set extracted from X.

Definition 5 *Given a target function $f : X \to Y$, an hypothesis h and a sample (finite) set $\mathcal{S} \subset X$, the **sample error** is defined as follows:*

$$\text{error}_{\mathcal{S}}(h) = \frac{1}{|\mathcal{S}|} \sum_{x\in\mathcal{S}} \delta(f(x) \neq h(x)) \tag{1.21}$$

where

- $\delta(\phi) = 1$ if ϕ is true
- $\delta(\phi) = 0$ if ϕ is false.

The sample error can be computed since, for each $x \in \mathcal{S}$, we know the value of $f(x)$. Is an approximation of the true error that depends from the sample set $\mathcal{S}$.

Since $\text{error}_{\mathcal{S}}(h)$ depends from the choice of $\mathcal{S}$, is not fixed, but we can model it as a random process, by extracting n random values from X according to the distribution $\mathcal{D}$ to construct $\mathcal{S}$, and we can evaluate the expected value of $\text{error}_{\mathcal{S}}(h)$ denoted

$$\mathbb{E}(\text{error}_{\mathcal{S}}(h)). \tag{1.22}$$

We want to formalize this expected value, in this case the sample error is a random variable that assign to each subset $\mathcal{S}$ of X of size n a number between 0 and 1:

$$\text{error}_{\mathcal{S}}(h) : \Omega \to [0,1] \tag{1.23}$$
$$\Omega = \{\mathcal{S} \ : \ \mathcal{S} \subset X \wedge |\mathcal{S}| = n\} \tag{1.24}$$

in this context, n is fixed. The probability of getting a certain value γ from this random variable, is the probability to extract from X a subset $\mathcal{S}$ of n items such that, the sample error is γ, and this depends from the probability distribution $\mathcal{D}$ on X. The expected value $\mathbb{E}(\text{error}_{\mathcal{S}}(h))$ is now well defined.

1.3.3 Unbiased Estimation

Definition 6 *The **bias** is defined as the expected value of the difference between the sample error and the true error:*

$$\mathbb{E}(\text{error}_{\mathcal{S}}(h)) - \text{error}_{\mathcal{D}}(h). \tag{1.25}$$

We want to find an unbiased sample error, in such case:

$$\text{bias} = 0 \implies \mathbb{E}(\text{error}_{\mathcal{S}}(h)) = \text{error}_{\mathcal{D}}(h) \tag{1.26}$$

To compute an unbiased estimation, we need to train an evaluate an hypothesis h on different set, let D to be the dataset, we have to split D in two disjoint set

$$D = T \cup S \tag{1.27}$$
$$T \cap S = \emptyset \tag{1.28}$$

Usually, $\frac{|T|}{|D|} \simeq \frac{2}{3}$. We use T to train our learning function to get h, and then, we calculate the sample error on S

$$\text{error}_{S}(h) = \frac{1}{|S|}\sum_{x \in S} \delta(f(x) \neq h(x)) \tag{1.29}$$

is ideal to choose T and S such that they have similar probability distribution over the features, in this case the random variable error_{S} is an unbiased estimator for the true error $\text{error}_{\mathcal{D}}$.

1.3.4 The Cross Validation Algorithm and others Performance Metrics

There exists an algorithm to estimate the expected value of the sample error, more the dataset D is large, more the estimation will be precise. The algorithm 2 estimate the expected value of the error, with L is denoted a fixed learning algorithm:

- $h = L(T)$, is the result of the learning algorithm L applied on the training set T

Algorithm 2 K-Fold Cross Validation

```
Require: $D$, $k$, $h$, $L$
  partition $D$ in $k$ disjoint sets $S_1, S_2 \dots, S_k$
  for $i = 1, 2 \dots, k$ do
    $T_i \leftarrow D \setminus S_i$
    $h_i \leftarrow L(T_i)$
    $\delta_i = \text{error}_{S_i}(h_i)$
  end for
  return $\text{error}_{L,D} = \frac{1}{k}\sum_{i=1}^{k} \delta_i$
```

Algorithm 3 Accuracy Comparator

```
Require: $D$, $k$, $h$, $L$
  partition $D$ in $k$ disjoint sets $S_1, S_2 \dots, S_k$
  for $i = 1, 2 \dots, k$ do
      $T_i \leftarrow D \backslash S_i$
      $h_a \leftarrow L_a(T_i)$
      $h_b \leftarrow L_b(T_i)$
      $\delta_i = \text{error}_{S_i}(h_a) - \text{error}_{S_i}(h_b)$
  end for
  return $\bar{\delta} = \frac{1}{k}\sum_{i=1}^{k} \delta_i$
```

We define the accuracy of a learning algorithm L as

$$\text{accuracy} = 1 - \text{error}_{L,D}. \tag{1.30}$$

The cross-validation algorithm can be used to compare the accuracy of two different learning methods L_a and L_b, as shown in algorithm 3.

Now that we defined the sample error, we can give a formal definition of overfitting. Let h to be an hypothesis for a model, h is overfitting is exists an hypothesis h' such that

$$\text{error}_S(h) < \text{error}_S(h') \tag{1.31}$$
$$\wedge \tag{1.32}$$
$$\text{error}_{\mathcal{D}}(h) > \text{error}_{\mathcal{D}} \tag{1.33}$$

Let's consider other performance metrics in binary classification. Let $f : X \to \{-, +\}$ to be the target function ad let D to be a sample set such that, 90% of elements in D is of class $+$. An hypothesis that always returns $+$ will have an accuracy of 90%, in this scenario the dataset is unbalanced, so the accuracy is not a good performance metric for the model. We can consider a table that counts the number of points in the sample set that are well classified or misclassified by an hypothesis:

	predicted class	
true class	+	-
+	true positive	false negative
-	false positive	true negative

we can define two additional metrics that is useful when we deal with binary classification:

- the **recall** is the ability of the hypothesis to avoid false negatives and is defined as follows
$$\frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{1.34}$$
- the **precision** is the ability of the hypothesis to avoid false positives and is defined as follows
$$\frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{1.35}$$

the importance of these metrics depend on the application.

For the classification problems we can define an extension of the previous table, called **confusion matrix**, and report how many instances of class C_i are classified in class C_j, the main diagonal contains the accuracy for each class. An example is shown in figure 1.3.

We consider now some performance metrics for the regression problems. Let $f : X \to \mathbb{R}^d$ to be the target function, and let $\hat{f}$ to be the learned function, for each sample $(x_i, f(x_i))$ in the dataset, we can compute the euclidian distance

$$|\hat{f}(x_i) - f(x_i)|. \tag{1.36}$$

Let n to be the number of samples, the three main metrics are the following:

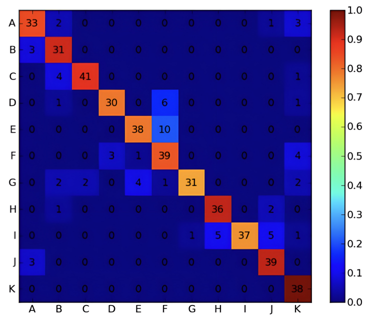


Figure 1.3: an example of a confusion matrix

- Mean Absolute Error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{f}(x_i) - f(x_i)| \tag{1.37}$$

- Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}(x_i) - f(x_i))^2 \tag{1.38}$$

- Root Mean Squared Error

$$RMSE = \sqrt{MSE} \tag{1.39}$$

The cross validation algorithm can be extended for the regression problems as shown in algorithm 4.

Algorithm 4 K-Fold Cross Validation for Regression

```
Require: $D$, $k$, $h$, $L$
  partition $D$ in $k$ disjoint sets $S_1, S_2 \dots, S_k$
  for $i = 1, 2 \dots, k$ do
      $T_i \leftarrow D \backslash S_i$
      $h_i \leftarrow L(T_i)$
      $\delta_i = MAE_{S_i}(h_i)$
  end for
  return $\text{error}_{L,D} = \frac{1}{k}\sum_{i=1}^{k}\delta_i$
```

CHAPTER

2 DECISION TREES

Let's consider the following classification problem, we have a target function

$$c : X \to C \tag{2.1}$$

and a data set

$$D = \bigcup_{i=1}^{n} \{(x_i, y_i)\} \subset X \times C \tag{2.2}$$

our approach is to define the hypothesis space H and search among all the possible solutions. Clearly we would like to find a solution $h^* \in D$ such that

$$h^*(x') \simeq f(x'), \quad \forall x' \notin D. \tag{2.3}$$

Recall: About the notation abuse, we say that $x' \in D$ if $\exists (x_i, y_i) \in D$ such that $x' = x_i$.

In this chapter we will see how to define the hypothesis space H with *decision trees*, let's assume that we are dealing with a finite problem, where

$$X = A_1 \times \cdots \times A_m \text{ with } A_i \text{ finite} \tag{2.4}$$

and clearly C is a finite set of classes.

Definition 7 *In the context of classification problems, a* ***decision tree*** *is a tree such that, each node is labeled with an attribute A_i, each edge is labeled with a value of an attribute $a_{i,j} \in A_i$, and each leaf assigns a classification value $c \in C$.*

A decision tree decides the class for each input $x \in X$, so we define the hypothesis space H as the set of all possible decision trees. The tree h' showed in figure 2.1, defines the following value

$$h'(a_{1,4}, a_{2,2}, a_{3,2}) = c_1 \tag{2.5}$$

Let's consider the following example about tennis, the input space X is

$$\begin{align} X &= \mathit{Outlook} \times \mathit{Temperature} \times \mathit{Humidity} \times \mathit{Wind} \tag{2.6}\\ \mathit{Outlook} &= \{\mathit{sunny}, \mathit{overcast}, \mathit{rain}\} \tag{2.7}\\ \mathit{Temperature} &= \{\mathit{hot}, \mathit{mild}, \mathit{cold}\} \tag{2.8}\\ \mathit{Humidity} &= \{\mathit{normal}, \mathit{high}\} \tag{2.9}\\ \mathit{Wind} &= \{\mathit{weak}, \mathit{strong}\} \tag{2.10} \end{align}$$

the target function $\mathit{PlayTennis} : X \to \{Yes, No\}$ should decides if the conditions are feasible to play tennis outside. The dataset is the onw shown in table 2.1.

Table 2.1: Dataset for the tennis problem

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

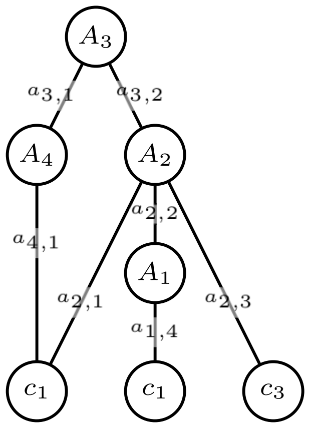
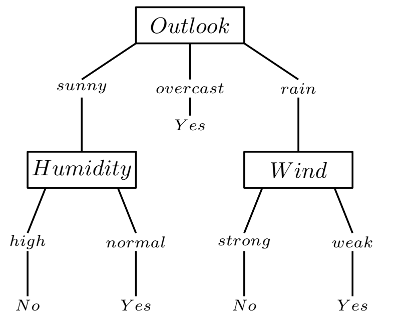


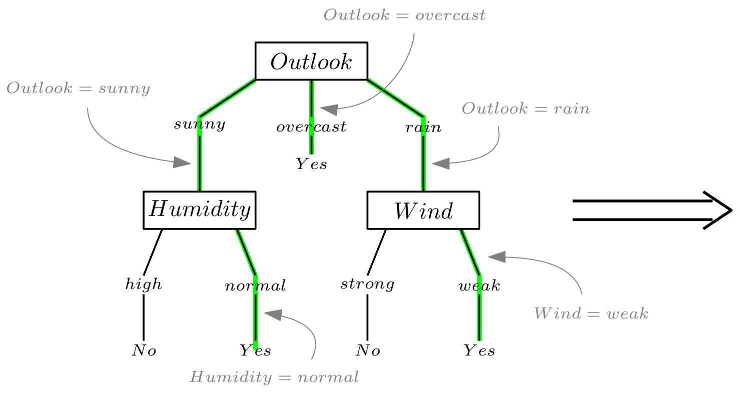
Figure 2.1: Example of a decision tree



Here's an example of a decision tree for the tennis problem. This representation is easy to classify a new instance that are not in the dataset, this tree defines rules such as: If the outlook is *overcast*, the output is Yes. The policy and the decisions are *explicit*, and provides an explanation for the outcomes.

In the hypothesis space H we have all possible decision trees, that are finite for this kind of problems. Decision trees represent a disjunction of conjunctions of constraints, that can be easily described by looking at the tree, traveling the traces of the tree backward starting from the outcomes. An example is shown in figure 2.2. The recursive algorithm 5 works to compute one decision tree given the dataset D.

We denote $N(+)$ a leaf node with the positive value, and $N(-)$ a leaf node with the negative value. If A_i is an attribute, we denote $N(A_i)$ an intermediate node with the attribute A_i. We denote $N(\ )$ a node for which a value has not yet been assigned.



$$(Outlook = sunny \land Humidity = normal) \lor$$
$$(Outlook = overcast) \lor$$
$$(Outlook = rain \land Wind = weak)$$

Figure 2.2: How paths in the tree transform into a logical formula

2.1 The Information Gain

The function `criteria`(Att) choose one attribute from the remaining ones in the set Att, and defines decision policy. What is the best attribute to choose during the algorithm? That procedure defines the behavior of the algorithm. Let's consider the following example for a reduced version of the tennis problem, with the dataset shown in table 2.2.

Table 2.2: Dataset for the reduced tennis problem

Day	*Outlook*	*Temperature*	*PlayTennis*
D1	Sunny	Hot	Yes
D2	Sunny	Cold	Yes
D3	Rain	Hot	No
D4	Rain	Cold	No

If we apply the algorithm on this sample, there will be two possible solutions, one for the case where we choose *Outlook* as the first choice for an attribute, one if we choose *Temperature*, the resulting trees as shown in figure 2.3.

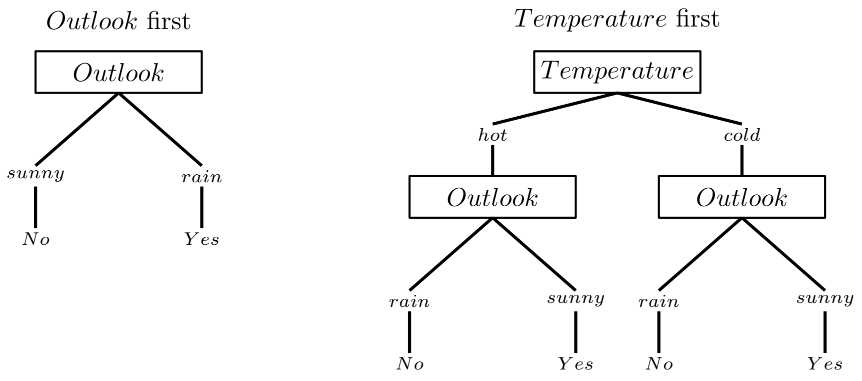


Figure 2.3: Different decision trees

In general, not making the correct choice in the attributes can lead to a tree that incapsulate less information.

Definition 8 *Given an example dataset S, let $p_\oplus$ to be the proportion of positive examples in S, and $p_\ominus = 1 - p_\oplus$ the proportion of negative examples in S, the* ***entropy*** *of S is the following value*

$$Entropy(S) = -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus. \tag{2.11}$$

The curve of the *Entropy* function is shown in figure 2.4.

Algorithm 5 Computing Decision Tree for Binary Classification

```
Require: $D = \{(x_i, c_i)_i\}$, $Att = \{A_1 \dots, A_m\}$
  if $\forall (x_i, c_i) \in D, \;\; c_i = +$ then
      return $N(+)$
  end if
  if $\forall (x_i, c_i) \in D, \;\; c_i = -$ then
      return $N(-)$
  end if
  if $Att = \emptyset$ then
      $D_+ = \{(x_i, c_i) \in D \;:\; c_i = +\}$
      $D_- = D \backslash D_+$
      if $|D_+| > |D_-|$ then
          return $N(+)$
      end if
      if $|D_-| > |D_+|$ then
          return $N(-)$
      end if
      return a random choice between $N(+)$ and $N(-)$
  end if
  $A_i = \texttt{criteria}(Att) \in Att$
  we create a node $N(A_i)$
  for each $a_{i,j} \in A_i$ do
      create an edge $N(A_i) \longrightarrow_{a_{i,j}} N(\;)$
      $D' = \{(x_i, c_i) \in D \;:\; x_i(A_i) = a_{i,j}\}$
      $N(\;) =$RecursiveCall$(D', Att \backslash \{A_i\})$
  end for
  return the tree with root $N(A_i)$
```

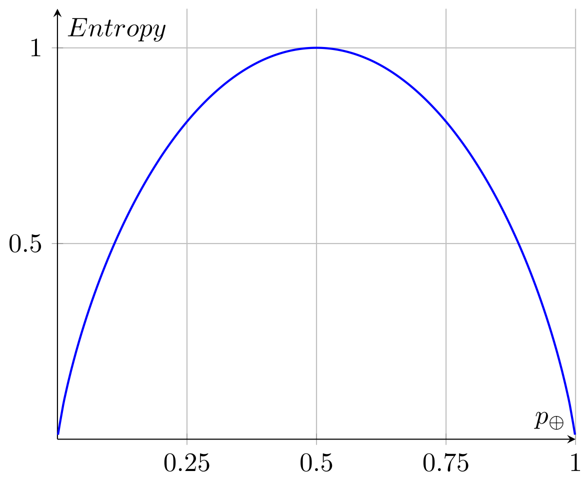


Figure 2.4: The curve of the *Entropy* function

The entropy is minimum if we have all positive samples or all negative samples. The goal is to reduce the entropy of a dataset. We can partition the dataset in function of the different attributes, and then calculate the entropy for all the partitions.

Given the table 2.2, we can create two partitions, one by selecting *Outlook*, and one by selecting *Temperature*, as shown in figure 2.5.

Splitting the dataset with outlook will generate two subsets with all positive/negative examples, so this two subsets will have an entropy of zero. In the other case, with temperature, the entropy will be 1.

In the case of multi-valued target function (classification problem that is not binary), we can define the

Day	Outlook	Temperature	PlayTennis
D1	**Sunny**	Hot	**Yes**
D2	**Sunny**	Cold	**Yes**
D3	**Rain**	Hot	**No**
D4	**Rain**	Cold	**No**

Day	Outlook	Temperature	PlayTennis
D1	Sunny	**Hot**	**Yes**
D3	Rain	**Hot**	**No**
D2	Sunny	**Cold**	**Yes**
D4	Rain	**Cold**	**No**

Figure 2.5: Partitions of the dataset

proportion of each class, denoted p_i, with n classes, the entropy is

$$Entropy(S) = \sum_{i=1}^{n} -p_i \log_2 p_i. \tag{2.12}$$

Note: We define $0 \log_2 0 = 0$.

Definition 9 *Given an example set S and an attribute A, we define the **information gain** as the expected reduction in entropy of S caused by knowing the value of the attribute A*

$$Gain(S, A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.13}$$

where

$$S_v = \{s \in S \ : \ s(A) = v\} \tag{2.14}$$

If we take in example the dataset in the table 2.1, we see that it have 9 positive samples and 5 negative samples, the entropy of the set is

$$-\frac{9}{14} \log_2 \left(\frac{9}{14}\right) - \frac{5}{14} \log_2 \left(\frac{5}{14}\right) = 0.94 \tag{2.15}$$

We want to calculate the information gain for the attribute $Wind$, that can be $Weak$ or $Strong$. If we partition the sub-dataset for this attribute, we get one dataset with 6 positive values and 2 negative values, and a sub-dataset with 3 positive values and 3 negative values.

Table 2.3: Split Dataset for the tennis problem

Outlook	*Temperature*	*Humidity*	**Wind**	*PlayTennis*
Sunny	Hot	High	Weak	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Overcast	Hot	Normal	Weak	Yes

Outlook	*Temperature*	*Humidity*	**Wind**	*PlayTennis*
Sunny	Hot	High	Strong	No
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Rain	Mild	High	Strong	No

we have that

$$|S_{weak}| = 8 \tag{2.16}$$
$$|S_{strong}| = 6 \tag{2.17}$$
$$|S| = 14 \tag{2.18}$$
$$Entropy(S_{weak}) = 0.811 \tag{2.19}$$
$$Entropy(S_{strong}) = 1 \tag{2.20}$$

The information gain is

$$Gain(S, Wind) = Entropy(S) - \frac{8}{14}Entropy(S_{weak}) - \frac{6}{14}Entropy(S_{strong}) \tag{2.21}$$
$$= 0.94 - \frac{8}{14}0.811 - \frac{6}{14} = 0.048. \tag{2.22}$$

This is a little information gain. A possible criteria for the choice of the next attribute in the algorithm 5, is to select the attribute with the highest information gain, in the case of the dataset 2.1, we have that

$$Gain(S, Outlook) = 0.245 \tag{2.23}$$
$$Gain(S, Humidity) = 0.151 \tag{2.24}$$
$$Gain(S, Wind) = 0.048 \tag{2.25}$$
$$Gain(S, Temperature) = 0.029 \tag{2.26}$$

We call `ID3` the decision tree algorithm 5 with the information gain criteria. This algorithm can be seen as an algorithm that makes a search in the space of all possible hypothesis/decision trees.

The hypothesis space of decision trees is *complete*, and we can represent all possible subsets of the input space. The algorithm returns one single hypothesis that is consistent with the dataset, this algorithm cannot find the global optimum function but a local minima.

This algorithm is not incremental and uses all the training set at each step.

2.2 Issues in Decision Trees

We have to consider the following problems:

- determining how deeply to grow the decision tree
- handling continuous attributes
- choosing appropriate attribute selection rule
- handling training data with missing attribute values
- handling attribute with different ”cost”.

Let D to be the dataset shown in table 2.1, we consider

$$D' = D \cup \langle sunny, hot, normal, strong, PlayTennis = No\rangle \tag{2.27}$$

`ID3` will generate two different decision trees T, T' for the different dataset D, D'. The question is:

is T' in general a better solution for this learning problem?

CHAPTER

3 PROBABILISTIC MODELS

3.1 Probability Recap

Consider the following action

$$A_t = \text{ leave the house } t \text{ minutes before the flight} \tag{3.1}$$

If we apply this action, what will be the answer to the question

The action A_t will it get me there on time?

The answer can't be a simple *Yes* or *No*, because depends from a series of casual factors that are unpredictable (such as the traffic reports, or a possible flat tire). The probability is needed to represents the uncertainties:

- Given the available evidence, A_{25} will get me at the airport on time with probability 0.04
- Given the available evidence, A_{60} will get me at the airport on time with probability 0.84
- Given the available evidence, A_{1440} will get me at the airport on time with probability 0.999.

In this context, the set of *all possible elementary events* that can occurs is called **Sample Space**, and is usually denoted Ω.

Definition 10 *A* ***Discrete Probability Space*** *is a tuple* $(\Omega, \mathcal{A}, \mathbb{P})$ *where*

- Ω *is the sample set of the elementary events.*
- $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ *is a* σ*-algebra, and contains all the events that can occurs, for which we can calculate a probability.* $\mathcal{A}$ *satisfies the following properties:*
  - $\Omega \in \mathcal{A}$
  - $A \in \mathcal{A} \implies A^C = \Omega \backslash A \in \mathcal{A}$
  - *if* $\{A_i\} \subset \mathcal{A}$ *is a countable sequence of events, then* $\bigcup_i \{A_i\} \in \mathcal{A}$.
- $\mathbb{P}$ *is the* ***probability measure****, is a function* $\mathbb{P} : \mathcal{A} \to [0, 1]$ *that assigns a probability to each event in* $\mathcal{A}$ *and satisfies the following properties:*
  - $\mathbb{P}(\Omega) = \sum_{\omega \in \Omega} \mathbb{P}(\{\omega\}) = 1$

– *if $\{A_i\}$ is a countable collection of disjoint sets in $\mathcal{A}$, then*

$$\mathbb{P}\left(\bigcup_i A_i\right) = \sum_i \mathbb{P}(A_i). \tag{3.2}$$

Definition 11 *Given a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, a* ***random variable*** *is a function $X : \Omega \to Y$.*

The probability $\mathbb{P}$ induces a *probability distribution* on a random variable X:

$$\mathbb{P}(X = x_i) = \sum_{\omega \in \Omega \ : \ X(\omega)=x_i} \mathbb{P}(\omega). \tag{3.3}$$

Definition 12 *Given a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ a* ***proposition*** *a is an event in $\mathcal{A}$, associated with a random variable $A : \Omega \to \{True, False\}$ such that*

$$a := A \ is \ True := \{\omega \in \Omega \ : \ A(\omega) = True\}. \tag{3.4}$$

We can use the first order logic to combine and operates on propositions

$$\neg a := A \text{ is } False := \{\omega \in \Omega \ : \ A(\omega) = False\} \tag{3.5}$$
$$a \wedge b := A \text{ and } B \text{ is } True := \{\omega \in \Omega \ : \ A(\omega) = B(\omega) = True\} \tag{3.6}$$
$$\neg a \vee b := A \text{ is } False \text{ or } B \text{ is } True := \{\omega \in \Omega \ : \ A(\omega) = False \vee B(\omega) = True\} \tag{3.7}$$

Since a proposition is an element of $\mathcal{A}$, we can calculate the probability:

$$\mathbb{P}(\neg a \vee b) = \sum_{\omega \in \Omega \ : \ A(\omega)=False \vee B(\omega)=True} \mathbb{P}(\omega) \tag{3.8}$$

The Definition 3 refers to a probability distribution, if $X : \Omega \to Y$ is a random variable, the probability distribution $\mathcal{D}$ assigns a probability to each element in $\mathcal{P}(Y)$.

We will give now an example of a probability space. Let's consider the rolling of a dice, in this case we have

- the sample set $\Omega = \{1, 2, 3, 4, 5, 6\}$
- the σ-algebra $\mathcal{A} = \{\{1\}, \{1, 2\} \dots\}$
- the probability $\mathbb{P}$ defined as follows

$$\forall \omega \in \Omega, \ \mathbb{P}(\{\omega\}) = \frac{1}{6}. \tag{3.9}$$

The probability of getting an odd number is

$$\mathbb{P}(\{1, 3, 5\}) = \mathbb{P}(\{1\}) + \mathbb{P}(\{3\}) + \mathbb{P}(\{5\}) = \frac{1}{2} \tag{3.10}$$

Let's consider the following random variable

$$X : \Omega \to \{-20, 30\} \tag{3.11}$$
$$X(\omega) = 30 \iff \omega \in \{1, 2\} \tag{3.12}$$

X represents a gamble on the dice, if the result is less than 3, the player wins 30 euros, else, he loses 20 euros. The probability of winning is

$$\mathbb{P}(\{1, 2\}) = \frac{1}{3}. \tag{3.13}$$

If we have n random variables $\{X_i\}$, it is possible to consider the **joint probability distribution**, inducted by the joint probability function.

Let X_1, X_2 to be two distinct random variables on two distinct probability spaces

$$X_1 : \Omega_1 \to Y_1 \text{ for the space } (\Omega_1, \mathcal{A}_1, \mathbb{P}_1) \tag{3.14}$$
$$X_2 : \Omega_2 \to Y_2 \text{ for the space } (\Omega_2, \mathcal{A}_2, \mathbb{P}_2) \tag{3.15}$$

where $\mathcal{D}_1$ is the distribution on X_1 and $\mathcal{D}_2$ is the distribution on X_2, the joint distribution $\mathcal{D}$ is defined as follows:

$$\text{let } x_1 \in Y_1, \quad x_2 \in Y_2 \tag{3.16}$$
$$\mathcal{D}(X_1 = x_1,\ X_2 = x_2) = \mathbb{P}(\{(\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 \ : \ X_1(\omega_1) = x_1 \wedge X_2(\omega_2) = x_2\}) \tag{3.17}$$

where $\mathbb{P} : \mathcal{A}_1 \times \mathcal{A}_2 \to [0, 1]$ is the joint probability and defines the probability that two events from the different probability space occurs.

If a and b are two events/propositions, we define the **conditional probability** as follows:

$$\mathbb{P}(a|b) = \frac{\mathbb{P}(a \wedge b)}{\mathbb{P}(b)}, \quad \text{if } \mathbb{P}(b) \neq 0 \tag{3.18}$$

it represents the probability that a occurs knowing that b is true. The **product rule** holds:

$$\mathbb{P}(a, b) = \mathbb{P}(a \wedge b) = \mathbb{P}(a|b)\mathbb{P}(b) = \mathbb{P}(b|a)\mathbb{P}(a) \tag{3.19}$$

The following equality is called **chain rule** and is needed to calculate the probability that multiple events $A_1, A_2 \dots A_n$ occurs:

$$\mathbb{P}(A_1, A_2 \dots A_n) = \prod_{i=1}^{n} \mathbb{P}(A_i | A_1, \dots A_{i-1}) = \tag{3.20}$$
$$\prod_{i=1}^{n} \left(A_i | \bigcap_{j=1}^{i-1} A_j \right). \tag{3.21}$$

Note: We will use the following abuse of notation, if $\mathbb{P}$ is the probability function for a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, and X is a random variable on Ω, we use the same symbol $\mathbb{P}$ to denote a probability distribution on X

$$\mathbb{P}(A),\ A \in \mathcal{A} \ \text{ in this context } \mathbb{P} \text{ is the probability function} \tag{3.22}$$
$$\mathbb{P}(X = x_i) \ \text{ in this context } \mathbb{P} \text{ is the probability distribution.} \tag{3.23}$$

Definition 13 *Given a probability space* $(\Omega, \mathcal{A}, \mathbb{P})$*, and two events* $A, B \in \mathcal{A}$*, we say that* A *and* B *are* ***independent*** *if and only if*

$$\mathbb{P}(A, B) = \mathbb{P}(A \wedge B) = \mathbb{P}(A)\mathbb{P}(B). \tag{3.24}$$