Model Selection for Learning Boolean Hypothesis

Joel Edu Sánchez Castro

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Ronaldo Fumio Hashimoto
Coorientador: Prof. Dr. Junior Barrera

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CNPq.

São Paulo, junho de 2018

Model Selection for Learning Boolean Hypothesis

Esta é a versão original da tese elaborada pelo
candidato Joel Edu Sánchez Castro, tal como
submetida à Comissão Julgadora.

Model Selection for Learning Boolean Hypothesis

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 10/08/2018. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Ronaldo Fumio Hashimoto (orientador) - IME-USP
- Prof. Dr. Junior Barrera - IME-USP
- Prof. Dr. Ulisses de Mendonça Braga Neto - TAMU
- Prof. Dr. David Correa Martins Junior - UFABC
- Prof. Dr. Marcelo Ris - Ernest & Young

Acknowledgments

First and foremost, I would like to express my sincere gratitude and appreciation to my advisors Professor Dr. Ronaldo Hashimoto and Professor Dr. Junior Barrera for all the guidance, support, patience and encouragement through these years. Their enthusiasm, passion for research and also discussions we had, sometimes heated ones, taught me a lot and they have made me a better scientist and person. I am also thank the thesis committee members for their helpful insights and revision of this work.

I am so grateful to the Instituto de Matemática e Estatística da Universidade de São Paulo for having awesome professors and making it possible for me to study here.

I express my gratitude to special people I met through these years. First, I would like to thank my peruvian family here in São Paulo for all the support and comprehension: Miguel, Reynaldo, Mariela, Rusbert, Julio, Raul, Christian, Hans, Grover among others. To the Melgarcito team: Leandro, Frank, chorri, Carlos, chiquito, Mittani, Bruno among others. To the wonderful friends I met in São Paulo in special to Jefferson, Adriano. To the E-science lab colleagues for the moments we share all these years. Finally, to my friends I met in my internship at Google. Without all of you I could not have surpassed the difficulty of having my family away from here.

I also dedicate this Ph.D. thesis specially to my parents, brothers, my little sister and all my family in Peru, without your support I would not have finished this work.

This thesis was financed by the National Council for Scientific and Technological Development (CNPq), thanks for supporting this work.

Abstract

Castro, J. E. S. **Model Selection for Learning Boolean Hypothesis**. 2018. Thesis (Phd) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

The state of the art in machine learning of Boolean functions is to learn a hypothesis h, which is similar to a target hypothesis f, using a training sample of size N and a family of a priori models in a given hypothesis set $\mathcal{H}$, such that h must belong to some model in this family. An important characteristic in learning is that h should also predict outcome values of f for previously unseen data, so the learning algorithm should minimize the generalization error which is the discrepancy measure between outcome values of f and h. The method proposed in this thesis learns family of models compatible with training samples of size N.

Taking into account that generalizations are performed through equivalence classes in the Boolean function domain, the search space for finding the correct model is the projection of $\mathcal{H}$ in all possible partitions of the domain. This projection can be seen as a model lattice which is anti-isomorphic to the partition lattice and also has the property that for every chain in the lattice there exists a relation order given by the VC dimension of the models. Hence, we propose a model selector that uses the model lattice for selecting the best model with VC dimension compatible to a training sample of size N, which is closely related to the classical sample complexity theorem.

Moreover, this model selector generalizes a set of learning methods in the literature (i.e, it unifies methods such as: the feature selection problem, multiresolution representation and decision tree representation) using models generated from a subset of partitions of the partition space. Furthermore, considering as measure associated to the models the estimated error of the learned hypothesis, the chains in the lattice present the so-called U-curve phenomenon. Therefore, we can use U-curve search algorithms in the model lattice to select the best models and, consequently, the corresponding VC dimension.

However, this new generation of learning algorithms requires an increment of computational power. In order to face this problem, we introduce a stochastic U-curve algorithm to work on bigger lattices. Stochastic search algorithms do not guarantee finding optimal solutions, but maximize the mean quality of the solution for a given amount of computational power. The contribution of this thesis advances both the state

of the art in machine learning theory and in practical problem solutions in learning.
Keywords: machine learning, Boolean hypothesis, domain partitions, learning feasibility, VC-dimension.

Resumo

Castro, J. E. S. **Seleção de Modelos para o Aprendizado de Hipóteses Booleanas**. 2018. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

O estado da arte em aprendizado de funções Booleanas é aprender uma hipótese h, que é similar a uma hipótese objetivo f, a partir de uma amostra de tamanho N e uma família de modelos a priori em um dado conjunto de hipóteses $\mathcal{H}$, tal que h deve pertencer a algum modelo nesta família. Uma característica importante no aprendizado é que h deve também predizer resultados de f para elementos que não aparecem no conjunto de treinamento, então o algoritmo de aprendizado deve minimizar o erro de generalização, o qual mede a discrepância entre os resultados de f e h. O método proposto nesta tese aprende uma família de modelos compatíveis com um conjunto de treinamento de tamanho N.

Tomando em consideração que as generalizações são realizadas através de classes de equivalência no domínio da função Booleana, o espaço de busca para encontrar um modelo apropriado é a projeção de $\mathcal{H}$ em todas as possíveis partições do domínio. Esta projeção pode ser vista como um reticulado de modelos que é anti-isomórfica ao reticulado de partições e também tem a propriedade que para cada cadeia no reticulado existe uma relação de ordem dada pela dimensão VC dos modelos. Portanto, propomos um seletor de modelos que usa o reticulado de modelos para selecionar o melhor modelo com dimensão VC compatível ao conjunto de treinamento de tamanho N, o qual é intimamente relacionado ao teorema clássico de complexidade da amostra.

Além disso, este seletor de modelos generaliza um conjunto de métodos de aprendizado na literatura (i.e, ele unifica métodos tais como: o problema de seleção de características, a representação multiresolução e a representação por árvores de decisão) usando modelos gerados por um subconjunto de partições do espaço de partições. Ademais, considerando como medida associada aos modelos o erro de estimação da hipótese aprendida, as cadeias no reticulado apresentam o fenômeno chamado U-curve. Portanto, podemos usar algoritmos de busca U-curve no reticulado de modelos para selecionar os melhores modelos, consequentemente, a correspondente dimensão VC.

No entanto, esta nova geração de algoritmos de aprendizado requerem um incremento de poder computacional. Para enfrentar este problema, introduzimos o algoritmo

Stochastic U-curve para trabalhar em reticulados maiores. Algoritmos de busca estocásticos não garantem encontrar soluções ótimas, mas maximizam a qualidade média das soluções para uma determinada quantidade de poder computacional. A contribuição desta tese avança ambos o estado da arte na teoria de aprendizado de máquina e soluções a problemas práticos em aprendizado.
Palavras chave: aprendizado de máquina, hipóteses Booleanas, partições do domínio, viabilidade de aprendizado, dimensão VC.

Contents

List of Abbreviations

BDCDT	Boolean Division Criteria Decision Trees
VC dimension	Vapnik-Chervonenkis dimension
GMS	Generic Model Selector
UCS	U-Curve Search
SFS	Sequential Forward Selection algorithm
SFFS	Sequential Forward Floating Selection

List of Symbols

$\|A\|$	Size of the set A
∞	Infinity
$\lg(a)$	Logarithm of the number a
$\mathcal{A}$	Learning algorithm
$(A, \leq)$	Poset composed by the set A taken together with a partial order $\leq$
A^X	Set of all vectors of the form $(x_1, x_2, \ldots, x_n)$ where $X = \{x_1, x_2, \ldots, x_n\}$ and each x_i takes values from A.
$B(k, p)$	Binomial distribution with parameters k and p
B_n	Bell number n
$c(A)$	Cost function used for search algorithm
$\mathcal{D}$	Set of pairs $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
$\mathcal{D}_{train}$	Training set
$\mathcal{D}_{val}$	Validation set
$d_{VC}(\mathcal{H})$	VC dimension of the hypothesis set $\mathcal{H}$
δ	Confidence parameter
ϵ	Error tolerance
exp	Exponential function
$E_{in}(h)$	In-sample error of hypothesis h
$E_{out}(h)$	Out-of-sample error of hypothesis h
$E_{val}(h)$	Validation error of hypothesis h
$\mathbb{E}(X)$	Expectation of the random variable X
$\mathcal{F}$	Set of events
h^-	Hypothesis learned using an incomplete dataset
$\mathcal{H}$	Hypothesis set
$\mathcal{H}_\pi$	Hypothesis set induced by partition π
$\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$	Dichotomies generated by the points $\mathbf{x}_1, \ldots, \mathbf{x}_N$
$L()$	Loss function used in machine learning
$m_{\mathcal{H}}()$	Growth function of $\mathcal{H}$
$\mathcal{P}(A)$	Power set of A
$\equiv_\pi$	Equivalence class of elements in the same block in π
$\hat{p}$	Estimated value of p

π	Partition of a set
Π_A	Set of all partitions of the set A
$\Pi(\mathcal{P}(A))$	Set of all partitions generated by subsets of A
Π_u	Partition set given by the user
$(\Pi_A, \leq)$	Partition lattice of the set A
Ω	Sample space
$\vee$	Disjunction
$\wedge$	Conjunction
$\bar{\cdot}$	Complementation
ψ	Characteristic function of a W-operator
Ψ	W-operator
$\Pr()$	Probability Function
$\Pr(X)$	Probability mass function of random variable X
$\Pr_{\mathcal{D}}(X)$	Estimated probability mass function of random variable X using the dataset $\mathcal{D}$
$\Pr(X, Y)$	Joint probability mass function of random variables X and Y
$\Pr(A\|B)$	Conditional probability
$\mathcal{L}(X)$	Set of all elements in $\mathcal{S}$ with cost less than X
$\mathcal{M}$	Set of all elements in $\mathcal{S}$ that are local minima in a lattice
$\mathcal{N}(X)$	Set of all elements in $\mathcal{S}$ that are neighbors of X in a lattice
$\mathcal{S}$	Set of all subsets of X in the U-curve problem
$O()$	Big O notation
ρ	Down-sampling
$\sigma(W)$	Association of the elements of a Boolean variable set W of size n to an n-tuple of Boolean variables
Θ	Set of formulas for representation
θ	Formula for representation
$\mathbf{x}$	Point (vector) in the space $\{0,1\}^n$.
$\mathcal{X}$	Set of points in the space $\{0,1\}^n$.
$\mathcal{X}_t$	Subspace of the domain that node t represents
y_t	Output label of node t
$\mathbb{U}$	Stochastic U-curve algorithm
$\mathbb{U}^*$	Basic Stochastic U-curve algorithm
$\mathbb{R}$	Set of all real numbers
$\mathbb{Z}$	Set of all integers
Z_c	$1 - \frac{1}{2}c$ quantile of a standard normal distribution corresponding to the target error rate c

List of Figures

List of Tables

Chapter 1

Introduction

In many areas of science as in medicine, finance and industry, *machine learning* has become a successful instrument to predict future data or other outcomes of interest based on past observations or measurements [Murphy, 2012].

In this work, we focus only in *Boolean hypothesis learning* because it is one of the most simple and educational applications in machine learning [Anthony and Biggs, 1992]. In a typical scenario, we want to predict the outcomes of an unknown target Boolean hypothesis based on a set of features (Boolean variables). To do this, we have a training set of data, in which we observe the outcome and feature values of past observations. Using this data, the objective of a learning method is to build a prediction model that will enable us to predict the outcomes for new unseen elements [Hastie *et al.*, 2009].

Over the years, many works in the literature have addressed the topic of Boolean hypothesis learning which is a very rich topic due to the developed theory and the variety of different learning models proposed as summarized in [Anthony and Biggs, 1992], [Natarajan, 1989] and [Anthony, 2005]. In the last two decades, binary W-operators learning (an application for Boolean hypothesis learning) has been widely studied. Consequently, many different approaches were proposed for solving this problem, as can be found in [Barrera *et al.*, 1997], [Dougherty *et al.*, 2001], [Kim, 2001], [Hirata Junior *et al.*, 2002], [Martins *et al.*, 2006], [Hirata, 2009], [Santos *et al.*, 2010], [Montagner *et al.*, 2017], just to mention a few of them. However, most of these proposed methods are more oriented in their efficiency and accuracy solving problems and little related to the study of their learning capabilities and properties.

On real applications, one of the most important steps in the learning process is *model selection*. In particular, we define as *model* a family of hypothesis with certain common properties. Thereby, the goal of model selection is selecting the best model and output the best hypothesis for that model [Abu-Mostafa *et al.*, 2012]. Therefore, model selection usually evaluates the performance of different model candidates in the problem in order to choose the best model achievable [Hastie *et al.*, 2009]. However, as the famous aphorism says: "Essentially, all models are wrong, but some are useful"[Box, 1976], some model candidates can be more appropriate than others for a specific learning problem. In typical model selectors, the model candidates are specified by the user in order to select the best one [Abu-Mostafa *et al.*, 2012]. Consequently, specifying the candidates can be a difficult task because of the necessity to have some machine learning background for understanding the requirements of the problem to choose a set of models according to the available data.

In general, it is a fact that the true error of the selected hypothesis in a model is

bounded above by the in-sample error of the hypothesis plus the generalization error of the model. Therefore, for a training sample of a fixed size, as the VC-dimension of the models grows, the in-sample error decreases, however the generalization error increases. This event is known as peaking phenomenon [Hastie *et al.*, 2009] (also known as U-curve phenomenon [Sima and Dougherty, 2008]). Although, we can find some learning methods in the literature as [Martins *et al.*, 2006], [Ris *et al.*, 2010], [Atashpaz-Gargari *et al.*, 2013], [Reis, 2013], [Atashpaz-Gargari *et al.*, 2018] that exploit the U-curve phenomenon to find appropriate hypothesis according to a training sample. We can improve the understanding on this subject in order to find more applications of U-curve algorithms.

1.1 Objectives and scope

In this thesis, we aim to study some aspects of the design of a new family of Boolean models and its importance in the design of new model selection methods. More specifically, the objectives that we want to achieve are:

- Study learning properties and capabilities of a new family of Boolean models based on partitions of the domain space.
- Study the properties between nested models based on partitions, and how these models along with a partial order form a lattice.
- Development and design of new model selectors that use models based on partitions as components.
- Improvements in the design of model selectors using poset search algorithms to improve their efficiency.
- Study the generalization properties of these model selectors unifying different learning methods in the literature that can be seen as instances of the proposed model selectors.
- Development of a stochastic algorithm to solve the U-curve problem and how it can be applied as improvement of the proposed model selectors.

1.2 Outline of the thesis

After presenting this introduction, this work is organized as follows:

- Chapter 2 (State of the Art): we make a revision of concepts needed to introduce the contributions of this work. We divide these concepts in three main parts: algebraic fundamentals where we present basic definitions about lattices, Boolean algebra, Boolean function representations and binary morphological operators; basic probability theory; and finally machine learning theory where we present important concepts for creating and evaluating learning models. Additionally, we present some learning applications and other important concepts in learning such as: ensemble methods, model selection, feature selection and the U-curve phenomenon.
- Chapter 3 (Design of Boolean Models): we present some contributions of this work. Initially, we design a new family of Boolean models built from partitions of the

domain space, then based on this new family we propose a new model selector called *Generic model selector* (GMS). Moreover, we present the generalization properties of the GMS studying different learning methods such as the feature selection problem, multiresolution representation and decision tree representation. We show that those learning methods generate a partition set and consequently can be seen as instances of the GMS. Furthermore, we propose some efficiency improvements in GMS design to deal with bigger search spaces. Finally, we show real simulations of the GMS to corroborate the theory presented in this chapter.

- Chapter 4 (Stochastic U-curve): we present another contribution of this work: a search algorithm in lattices that present the U-curve phenomenon called Stochastic U-curve. We show that this algorithm is not optimal but it has a bounded computational complexity, as well as having stochastic properties to guarantee good results. Finally, we present experiments of this algorithm that show its efficiency.

- Chapter 5 (Conclusions and Future Work): we review the principal contributions of this thesis along with some discussions about the presented results. Additionally, we present some guidelines for the Generic model selector as future work.

Chapter 2

State of the Art

In this chapter, we introduce fundamental concepts and definitions on which this work builds upon. We divided this chapter in three main sections. In Section 2.1, we see algebraic fundamentals such as Boolean Algebra and its representation, basic lattice theory and binary morphological operators. In Section 2.2, we see basic probability theory. Finally, in Section 2.3, we study elemental machine learning theory which includes its components, performance evaluation and its applications.

2.1 Algebraic Fundamentals

In this section we present basic algebraic fundamentals, most of this theory will be based on Boolean algebra. We start this section introducing the lattice algebraic structure which is used in many sections of this work. Next, we define Boolean algebras and some popular Boolean function representations, followed by more complex methods for representation in the following sections. Finally, we present binary morphological operators that can be seen as Boolean function applications.

2.1.1 Lattices

Lattices are abstract structures that have many different applications, they are used to define important concepts in this work and also they help us to represent different elements in a structured way. Full details of lattice theory presented in this section can be found in [Grätzer and Davey, 2003].

First, we define some previous concepts before introducing lattices. Formally, a relation $\leq$ is a *partial order* in the set A if satisfies: for all a,b and c in A;

1. $a \leq a$ (reflexivity).
2. if $a \leq b$ and $b \leq a$, then $a = b$ (antisymmetry).
3. if $a \leq b$ and $b \leq c$, then $a \leq c$ (transitivity).

A *partially ordered set* (or *poset*) is a set A taken together with a partial order $\leq$ on it, and it is denoted by $(A, \leq)$.

An *upper bound* of $a \in A$ is any element $u \in A$ such that $a \leq u$, similarly, a *lower bound* of $a \in A$ is any element $l \in A$ such that $l \leq a$.

A *supremum* of two elements $a, b \in A$ is the upper bound $u \in A$ of both a, b such that for any upper bound $z \in A$ of both a, b, we have $u \leq z$. An *infimum* of two elements

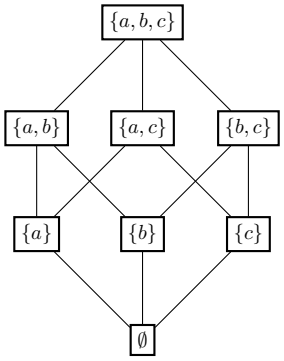


Figure 2.1: *Hasse diagram of the lattice* $(\mathcal{P}(A), \subseteq)$.

$a, b \in A$ is the lower bound $l \in A$ of both a, b such that for any lower bound $z \in A$ of both a, b, we have $z \leq l$.

A *lattice* is a poset $(A, \leq)$ in which every two elements $a, b \in A$ have a supremum and an infimum. Moreover, a *complete lattice* is a poset $(A, \leq)$ in which every subset S of A has both a supremum and an infimum.

In this work, we will use Hasse diagrams for illustrating posets and lattices. Concretely, in a *Hasse diagram* for a poset $(A, \leq)$, a point is drawn for each element in A and a line segment that goes upward from a point a to a point b is drawn whenever there are two points a, b in the poset such that $a < b$ and there is no c such that $a < c < b$.

The following example shows the lattice generated by the collection of all subsets of a set and it is illustrated using a Hasse diagram.

Example 2.1.1. For any set A, the collection of all subsets of A also called *power set* of A (denoted by $\mathcal{P}(A)$) along with the partial order of set inclusion is a lattice, that is, $(\mathcal{P}(A), \subseteq)$ is a lattice [Grätzer and Davey, 2003].

As example, given a set $A = \{a, b, c\}$, the lattice $(\mathcal{P}(A), \subseteq)$ is illustrated in the Hasse diagram of Figure 2.1. As example we observe that the upper bounds of $\{a\}$ are the sets $\{a, b\}$, $\{a, c\}$ and $\{a, b, c\}$, similarly, the lower bounds of $\{a\}$ is only the set $\emptyset$. Also, we observe that the supremum of $\{a\}$ and $\{c\}$ is the set $\{a, c\}$ and their infimum is the set $\emptyset$.

Partition Lattice

The partition lattice is a special type of lattice and is one of the most important concepts in this work. This lattice is based on all partitions of a set and a partial order between them. Formally, a partition of a set can be defined as follows:

Definition 2.1.1. A *partition* of a set X is a set π of nonempty subsets of X such that every element $x \in X$ is in exactly one of these subsets. The members of π are called *blocks* of π. In other words, we have a partition π of X if and only if all the following conditions hold:

1. Partition π does not contain the empty set.
2. The union of the blocks in π is equal to X (that is $\cup_{A \in \pi} A = X$).
3. Any two blocks are either equal or pairwise disjoint (that is, $\forall A, B \in \pi, A \neq B \Rightarrow A \cap B = \emptyset$).

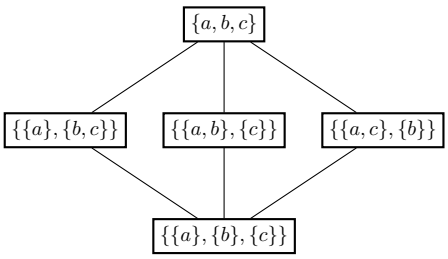


Figure 2.2: *Hasse diagram of the partition lattice of the set* $\{a, b, c\}$*, i.e.* $(\Pi_{\{a,b,c\}}, \leq)$.

We say that two elements $x, y \in X$ are equivalent on π if they belong to the same block of π, we denote their equivalence as $x \equiv_\pi y$.

More interestingly, the set of all partitions of X with the partial order $\leq$ defined as:

$$\pi_1 \leq \pi_2 \Leftrightarrow x \equiv_{\pi_1} y \text{ implies that } x \equiv_{\pi_2} y,$$

is a lattice [Grätzer and Davey, 2003], and it is known as *partition lattice* of X. So, if we denote as Π_X the set of all partitions of X, then $(\Pi_X, \leq)$ is the partition lattice of X. For instance, Figure 2.2 illustrates a Hasse diagram of the partition lattice of the set $\{a, b, c\}$, i.e. $(\Pi_{\{a,b,c\}}, \leq)$.

Furthermore, the partition lattice $(\Pi_X, \leq)$ is a complete lattice and the number of the elements in the lattice grows exponentially as the number of elements in X grows. The number of elements in Π_X is given by the Bell number $B_{|X|}$ [Hedmark, 2017], where $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$, and $B_0 = B_1 = 1$. Using the definition, the first Bell numbers are: $B_0 = 1, B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15, B_5 = 52$, $B_6 = 203, \ldots, B_{20} = 583.274.220.505$.

2.1.2 Boolean Algebra

A *Boolean algebra* is a tuple $\langle B, \vee, \wedge, \bar{\cdot}, 0, 1\rangle$, where B is a set of elements with two binary operations $\vee$ (disjunction), $\wedge$ (conjunction) and a unary operation $\bar{\cdot}$ (complementation), satisfying the following axioms:

1. Axiom 1: Operators $\vee, \wedge$ are commutative.
   $a \vee b = b \vee a, \forall a, b \in B$ and
   $a \wedge b = a \wedge b, \forall a, b \in B$.

2. Axiom 2: There exist two identity elements 0 and 1 such that
   $a \vee 0 = a, \forall a \in B$ and
   $a \wedge 1 = a, \forall a \in B$.

3. Axiom 3: Each operation is distributive with respect to the other, that is:
   $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c), \forall a, b, c \in B$ and
   $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c), \forall a, b, c \in B$.

4. Axiom 4: For each element $a \in B$, exists an inverse element $\overline{a} \in B$ such that $a \wedge \overline{a} = 0$ and $a \vee \overline{a} = 1$.

One of the most common examples of Boolean algebra is the tuple $\langle \{0, 1\}, \vee, \wedge, \bar{\cdot}, 0, 1\rangle$ satisfying the following rules:

$$0 \vee 0 = 0, 0 \vee 1 = 1, 1 \vee 0 = 1, 1 \vee 1 = 1;$$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Figure 2.3: *Truth table of a function with three Boolean variables.*

$$0 \wedge 0 = 0, 0 \wedge 1 = 0, 1 \wedge 0 = 0, 1 \wedge 1 = 1;$$
$$\overline{0} = 1, \overline{1} = 0.$$

In the next definitions, we will only consider the Boolean algebra $\langle \{0,1\}, \vee, \wedge, \bar{}, 0, 1 \rangle$. A *Boolean variable* is a variable that may take values from the set $B = \{0,1\}$, In some applications the set $\{0,1\}$ is interpreted by other two-element set such as {Yes, No}, {True, False}, {ON, OFF} or {Success, Failure}.

A *Boolean function* f of n Boolean variables is a function $f : \{0,1\}^n \to \{0,1\}$ where n is a positive integer and $\{0,1\}^n$ denotes the n-fold Cartesian product of the set $\{0,1\}$ with itself. A point in the space $\{0,1\}^n$ also called *vector* is denoted by $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, for instance, a specific point in the space $\{0,1\}^4$ could be $\mathbf{x} = (1,0,1,1)$ or simply denoted by $\mathbf{x} = 1011$. If $f(x) = 1$ (resp., 0), then x is called a true point (resp., false point) of f. The set of all true points (resp., false points) of f is denoted by $T(f)$ (resp., $F(f)$). In the same way, a set of points in the space $\{0,1\}^n$ is denoted by $\mathcal{X} \subseteq \{0,1\}^n$. For example, let $\mathbf{x}, \mathbf{y} \in \{0,1\}^3$, $\mathbf{x} = 101$ and $\mathbf{y} = 110$, the set of points $\mathcal{X}$ composed by $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\mathcal{X} = \{\mathbf{x}, \mathbf{y}\} = \{101, 110\}.$$

Given a variable set $X = \{x_1, x_2, \ldots, x_n\}$ such that each variable may take values from the set B, the set of all vectors of the form $(x_1, x_2, \ldots, x_n)$ is denoted by B^X. For instance, if $E = \{x_1, x_2\}$ is a set of Boolean variables, then we have that

$$\{0,1\}^E = \{(0,0); (0,1); (1,0); (1,1)\} = \{00, 01, 10, 11\}.$$

For more details on these definitions, see [Crama and Hammer, 2011].

2.1.3 Boolean Function Representations

In the literature, as we can see in [Crama and Hammer, 2011], there are many methods for representing Boolean functions. In this work, we study some of these representations for later introducing learning of Boolean functions.

One of the easiest ways to represent a Boolean function is through a *truth table*. A truth table is a table that uses one column for each Boolean variable and one final column to show the output of the function as we illustrate in Figure 2.3. However, truth tables are difficult to represent because the number of entries of the Boolean function grows exponentially with n.

Another popular representation of Boolean functions is through Boolean expressions.

A *Boolean expression* in the variables $x_1, x_2, \ldots x_n$ is any expression that can be constructed by those variables and applying a finite number of operations AND, OR and NOT; e.g. $x_2 \vee (x_1 \wedge \overline{x_3})$. It is common to ignore the symbol $\wedge$ in Boolean expressions when is understood ; e.g. $x_2 \vee (x_1 \wedge \overline{x_3}) = x_2 \vee x_1\overline{x_3}$. More elaborated representation methods of Boolean functions will be presented in following sections.

2.1.4 Multiresolution representation for Boolean functions

Multiresolution representation for Boolean functions is a method that, if used properly, it can use only a small number of elements for representing Boolean functions. As presented in [Dougherty *et al.*, 2001] and [Hirata Junior *et al.*, 2002], multiresolution representation was used specifically for learning W-operators (see later in Section 2.1.7). Nevertheless, in this section we present how this approach can be adapted for representing Boolean functions in general.

Multiresolution representation of Boolean functions is based on an important concept called *down-sampling* (also called projection), it is used for dividing the Boolean domain into element groups. After the division, a representative of each group is used to assign the same output value to all the elements in the group. This fact implies that the number of mappings needed to cover all the domain elements is usually small enough to represent them efficiently in a computer. The down-sampling function is defined as follows:

Definition 2.1.2. Let X, Z be two sets of Boolean variables such that $|X| > |Z|$ and let $\mathcal{X}_0 = \{0,1\}^X$ and $\mathcal{X}_1 = \{0,1\}^Z$ be the set of variables realizations of X and Z respectively. A *down-sampling* $\rho : \mathcal{X}_0 \to \mathcal{X}_1$ is a mapping that assigns to each configuration $\mathbf{x} \in \mathcal{X}_0$ a configuration $\mathbf{z} = \rho(\mathbf{x}), \mathbf{z} \in \mathcal{X}_1$.

More interestingly, a down-sampling $\rho : \mathcal{X}_0 \to \mathcal{X}_1$ induces a partition π of the space $\mathcal{X}_0$ and determines an equivalence relation on $\mathcal{X}_0$ given by

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}_0, \mathbf{x}_1 \equiv_\pi \mathbf{x}_2 \Leftrightarrow \rho(\mathbf{x}_1) = \rho(\mathbf{x}_2).$$

In the next example is presented a simple down-sampling of the space $\mathcal{X}_0$.

Example 2.1.2. Let $X = \{x_1, x_2, x_3\}$ and $Z = \{z_1, z_2\}$ be two sets of Boolean variables, and let $\mathcal{X}_0 = \{0,1\}^X$ and $\mathcal{X}_1 = \{0,1\}^Z$, i.e. $\mathcal{X}_0 = \{000, 001, 010, 011, 100, 101, 110, 111\}$ and $\mathcal{X}_1 = \{00, 01, 10, 11\}$. An example of down-sampling of $\mathcal{X}_0$ is the function $\rho : \mathcal{X}_0 \to \mathcal{X}_1$ defined as

$$\rho(x_1, x_2, x_3) = (z_1 = x_2, z_2 = x_3).$$

For instance, two examples of the application of ρ are $\rho(101) = 01$ and $\rho(010) = 10$. Moreover, it is observed that ρ induces a partition π in $\mathcal{X}_0$, that is $\pi = \{\{000, 100\}, \{001, 101\}, \{010, 110\}, \{011, 111\}\}$.

It is important to mention that in this work we define the multiresolution representation of Boolean functions in a more restrictive way than defined in [Dougherty *et al.*, 2001] and [Hirata Junior *et al.*, 2002] in order to cover only the more common application of the multiresolution representation. This is when we have Boolean variable sets $W_0, W_1, \ldots, W_n$ such that $W_{i+1} \subset W_i$, for $0 \leq i \leq n-1$, and these sets are used to automatically define a sequence of down-samplings also called multiresolution down-samplings. To facilitate next definitions we define the function σ as:

Definition 2.1.3. Let $W = \{w_1, w_2, \ldots, w_n\}$ be a set of Boolean variables with $n = |W|$, we define as $\sigma(W)$ the association of the elements in W to an n-tuple of Boolean variables, that is,

$$\sigma(W) = \sigma(\{w_1, w_2, \ldots, w_n\}) = (w_1, w_2, \ldots, w_n).$$

Now, using Definition 2.1.3 the multiresolution down-samplings are defined as follows:

Definition 2.1.4. Let $W_0, W_1, \ldots, W_n$ be sets of Boolean variables such that $W_{i+1} \subset W_i$, for $0 \leq i \leq n-1$, and let $\mathcal{X}_0 = \{0,1\}^{W_0}, \mathcal{X}_1 = \{0,1\}^{W_1}, \ldots, \mathcal{X}_n = \{0,1\}^{W_n}$. *Multiresolution down-samplings* are defined as a down-sampling sequence $\rho_1 : \mathcal{X}_0 \to \mathcal{X}_1, \rho_2 : \mathcal{X}_1 \to \mathcal{X}_2, \ldots, \rho_n : \mathcal{X}_{n-1} \to \mathcal{X}_n$ such that

$$\rho_1(\mathbf{w}_0) = \mathbf{w}_1, \rho_2(\mathbf{w}_1) = \mathbf{w}_2, \ldots, \rho_n(\mathbf{w}_{n-1}) = \mathbf{w}_n.$$

where $\mathbf{w}_i = \sigma(W_i)$, for $0 \leq i \leq n$.

Furthermore, using multiresolution down-samplings we can induce a sequence of partitions $\pi_1, \pi_2, \ldots, \pi_n$ of the space $\mathcal{X}_0$ as follows:

$$\begin{aligned}
&\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}_0, \mathbf{x}_1 \equiv_{\pi_1} \mathbf{x}_2 \Leftrightarrow \rho_1(\mathbf{x}_1) = \rho_1(\mathbf{x}_2);\\
&\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}_0, \mathbf{x}_1 \equiv_{\pi_2} \mathbf{x}_2 \Leftrightarrow \rho_2(\rho_1(\mathbf{x}_1)) = \rho_2(\rho_1(\mathbf{x}_2));\\
&\qquad\qquad\vdots\\
&\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}_0, \mathbf{x}_1 \equiv_{\pi_n} \mathbf{x}_2 \Leftrightarrow \rho_n(\ldots \rho_2(\rho_1(\mathbf{x}_1)) \ldots) = \rho_n(\ldots \rho_2(\rho_1(\mathbf{x}_1)) \ldots).
\end{aligned}$$

Hence, using Definition 2.1.4, we can imply that from given nested sets of Boolean variables $W_0, W_1, \ldots, W_n$, we can induce the spaces $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_n$, multiresolution down-samplings $\rho_1, \rho_2, \ldots, \rho_n$ and partitions $\pi_1, \pi_2, \ldots, \pi_n$.

An example of multiresolution down-samplings is presented below.

Example 2.1.3. Let $W_0 = \{w_1, w_2, w_3\}, W_1 = \{w_2, w_3\}$ and $W_2 = \{w_3\}$ be sets of Boolean variables, we induce the spaces $\mathcal{X}_0 = \{0,1\}^{W_0}, \mathcal{X}_1 = \{0,1\}^{W_1}$ and $\mathcal{X}_2 = \{0,1\}^{W_2}$, this is, $\mathcal{X}_0 = \{000, 001, 010, 011, 100, 101, 110, 111\}$, $\mathcal{X}_1 = \{00, 01, 10, 11\}$ and $\mathcal{X}_2 = \{0,1\}$. Besides, we induce the multiresolution down-samplings $\rho_1 : \mathcal{X}_0 \to \mathcal{X}_1$ and $\rho_2 : \mathcal{X}_1 \to \mathcal{X}_2$ given by

$$\rho_1(w_1, w_2, w_3) = (w_2, w_3) \text{ and } \rho_2(w_2, w_3) = (w_3).$$

For instance, $\rho_1(010) = 10$, $\rho_1(110) = 10$ and $\rho_2(\rho_1(110)) = \rho_2(10) = 0$. Consequently, ρ_1 and ρ_2 induce the partitions $\pi_1, \pi_2 \in \mathcal{X}_0$ respectively, that is $\pi_1 = \{\{000, 100\}, \{001, 101\}, \{010, 110\}, \{011, 111\}\}$ and $\pi_2 = \{\{000, 100, 010, 110\}, \{001, 101, 011, 111\}\}$ as illustrated in Figure 2.4.

Using multiresolution down-samplings, we define the multiresolution representation of Boolean functions as:

Definition 2.1.5. For defining a Boolean function with *multiresolution representation* we need a set of nested Boolean variables sets $W_0, W_1, \ldots, W_n$ and element sets $\mathcal{X}_{00}, \mathcal{X}_{01}, \mathcal{X}_{10}, \mathcal{X}_{11}, \ldots, \mathcal{X}_{n0}, \mathcal{X}_{n1}$ satisfying some properties presented below.

First, we induce the domain spaces $\mathcal{X}_0 = \{0,1\}^{W_0}, \mathcal{X}_1 = \{0,1\}^{W_1}, \ldots, \mathcal{X}_n = \{0,1\}^{W_n}$ and their respective multiresolution down-samplings $\rho_1 : \mathcal{X}_0 \to \mathcal{X}_1$, $\rho_2 : \mathcal{X}_1 \to \mathcal{X}_2, \ldots,$ $\rho_n : \mathcal{X}_{n-1} \to \mathcal{X}_n$. The element sets $\mathcal{X}_{00}, \mathcal{X}_{01}, \mathcal{X}_{10}, \mathcal{X}_{11}, \ldots, \mathcal{X}_{n0}, \mathcal{X}_{n1}$ must satisfy the

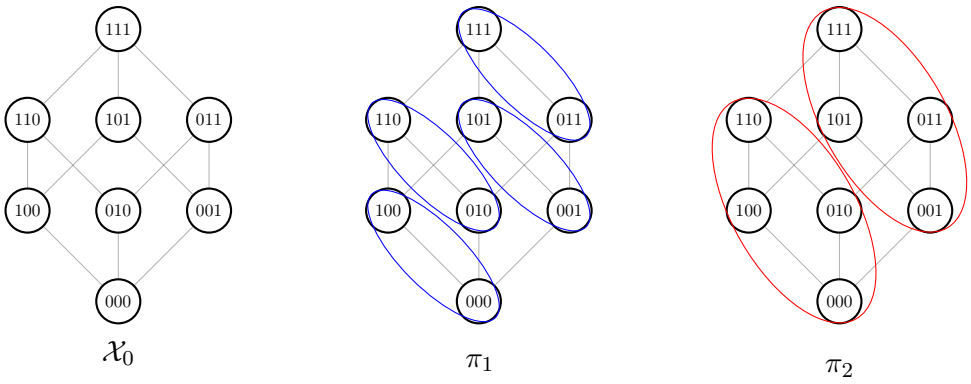


Figure 2.4: *Partitions of the Boolean domain in Example 2.1.3.*

following property:

$$\mathcal{X}_{i0}, \mathcal{X}_{i1} \subseteq \mathcal{X}_i \text{ and } \mathcal{X}_{i0} \cap \mathcal{X}_{i1} = \emptyset, \text{ for } 0 \leq i \leq n.$$

Thereby, the Boolean function with multiresolution representation $f : \mathcal{X}_0 \to \{0, 1\}$ is given by:

$$f(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in \mathcal{X}_{00} \\ 1, & \text{else if } \mathbf{x} \in \mathcal{X}_{01} \\ 0, & \text{else if } \rho_1(\mathbf{x}) \in \mathcal{X}_{10} \\ 1, & \text{else if } \rho_1(\mathbf{x}) \in \mathcal{X}_{11} \\ \vdots & \\ 0, & \text{else if } \rho_n(\dots(\rho_2(\rho_1(\mathbf{x})))\dots) \in \mathcal{X}_{n0} \\ 1, & \text{else if } \rho_n(\dots(\rho_2(\rho_1(\mathbf{x})))\dots) \in \mathcal{X}_{n1} \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Most importantly, a multiresolution representation also determines a partition of the domain space $\mathcal{X}_0$, this is given by the following algorithm:

In the next example, we present a multiresolution representation of a Boolean function and its generated partition.

Example 2.1.4. This example shows the multiresolution representation of a Boolean function, we have the nested boolean sets $W_0 = \{w_1, w_2, w_3\}, W_1 = \{w_2, w_3\}$ and $W_2 = \{w_3\}$ and the element sets $\mathcal{X}_{00} = \{000, 001\}$, $\mathcal{X}_{01} = \{100\}$, $\mathcal{X}_{10} = \{00\}$, $\mathcal{X}_{11} = \{10\}$, $\mathcal{X}_{20} = \{1\}, \mathcal{X}_{21} = \{0\}$ that satisfy all properties explained in Definition 2.1.5.

Now, we will show the Boolean function f that these sets represent. First, as shown in Example 2.1.3, we induce the sets $\mathcal{X}_0 = \{0, 1\}^{W_0}, \mathcal{X}_1 = \{0, 1\}^{W_1}$ and $\mathcal{X}_2 = \{0, 1\}^{W_2}$ and their respective multiresolution down-samplings $\rho_1 : \mathcal{X}_0 \to \mathcal{X}_1$, $\rho_2 : \mathcal{X}_1 \to \mathcal{X}_2$. Therefore, the Boolean function $f : \mathcal{X}_0 \to \{0, 1\}$ represented is given by evaluating $f(\mathbf{x})$ for each element in $\mathcal{X}_0$ using the algorithm in Definition 2.1.5.

For example, we evaluate the function output for $\mathbf{x} = 111$. It is observed that $\mathbf{x} = 111$ does not belong neither to $\mathcal{X}_{00}$ nor to $\mathcal{X}_{01}$. Subsequently, we observe that $\rho_1(111) = 11$ does not belong neither to $\mathcal{X}_{10}$ nor to $\mathcal{X}_{11}$, but $\rho_2(\rho_1(111)) = 1$ and 1 belongs to $\mathcal{X}_{20}$. As a result, $f(\mathbf{x} = 111) = 0$. Thereby, evaluating all the elements $\mathcal{X}_0$ we obtain the Boolean function represented by the truth table in Figure 2.5(a). Furthermore, using Algorithm 1 we obtain the partition $\pi = \{\{000\}, \{001\}, \{100\}, \{010, 110\}, \{101, 011, 111\}\}$ illustrated

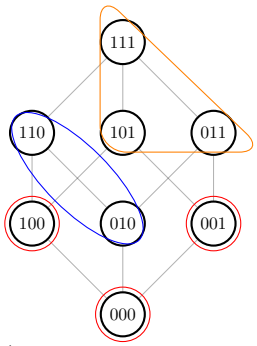
Algorithm 1: Partition generated by the multiresolution representation.

Output: Partition π of $\mathcal{X}_0$.

```
π = {};
for i = 0 to n do
    for j = 0 to 1 do
        if i = 0 then
            Add a block in π for each element in X_0j consisting of
                exactly that element.
        else
            Add a block in π for each element in X_ij with elements
                x ∈ X_0 such that ρ_i(...(ρ_1(x))...) ∈ X_ij and x was
                not previously defined in any partition.
        end
    end
end
Add a block in π consisting of all elements x ∈ X_0 that was not previously
    defined in any partition;
return π;
```

$\mathbf{x}$	$f(\mathbf{x})$
000	0
001	0
010	1
011	0
100	1
101	0
110	1
111	0

(a) Truth table.



(b) Generated partition.

Figure 2.5: *Truth table of the Boolean function in Example 2.1.4 and its generated partition.*

in Figure 2.5(b).

The next example shows the multiresolution representation ability for defining Boolean functions of many variables.

Example 2.1.5. In this example we show how the multiresolution representation for Boolean functions uses a small number of elements to completely define a Boolean function of 6 Boolean variables. Given the Boolean variable sets W_0, W_1, W_2, such that $W_0 = \{w_1, w_2, w_3, w_4, w_5, w_6\}$, $W_1 = \{w_1, w_2, w_3\}$, $W_2 = \{w_1\}$, so we have $W_2 \subset W_1 \subset W_0$. Besides, we have the following element sets satisfying properties in Definition 2.1.5, $\mathcal{X}_{00} = \{010010, 110011, 111111\}$, $\mathcal{X}_{01} = \{011110\}$, $\mathcal{X}_{10} = \{011, 010\}$, $\mathcal{X}_{11} = \{111, 000, 110\}$ $\mathcal{X}_{20} = \{0\}$, $\mathcal{X}_{21} = \{1\}$.

Now, we proceed to show the Boolean function represented by these sets. First, we induce the respective domain spaces $\mathcal{X}_0 = \{0,1\}^{W_0}, \mathcal{X}_1 = \{0,1\}^{W_1}, \mathcal{X}_2 = \{0,1\}^{W_2}$ and their respective multiresolution down-samplings $\rho_1 : \mathcal{X}_0 \to \mathcal{X}_1$, $\rho_2 : \mathcal{X}_1 \to \mathcal{X}_2$. Thereby, we obtain a Boolean function $f : \mathcal{X}_0 \to \{0,1\}$ such that $f(010010) = f(110011) = f(111111) = 0$, $f(011110) = 1$ (using $\mathcal{X}_{00}$ and $\mathcal{X}_{01}$). Using $011 \in \mathcal{X}_{10}$ we have that

all the elements in $\mathcal{X}_0$ not previously defined that have a configuration compatible with the pattern 011XXX (where X means any value 0 or 1) are mapped to 0, that is $f(011XXX) = 0$, notice that the element 011110 is compatible with 011XXX but $f(011110) = 1$ because it was defined before. In a similar way, $f(010XXX) = 0$ for elements not previously defined. Using the set $\mathcal{X}_{11}$, we have $f(111XXX) = f(000XXX) = f(110XXX) = 1$ for elements not previously defined. Using the sets $\mathcal{X}_{20}$ and $\mathcal{X}_{21}$ we have $f(0XXXXX) = 0$ and $f(1XXXXX) = 1$, for elements not previously defined. Observe that these two patterns covers all the points in the domain. As we have seen, the Boolean function f of 6 Boolean variables is completely defined using only 6 element sets, if we would represent this function entry by entry we would need to define $2^6 = 64$ entries.

However, the multiresolution representation seems difficult to be used in practice, but there are applications that use this representation successfully such as the Multiresolution W-operators (seen later in Section 2.3.4).

2.1.5 Decision tree representation for Boolean Functions

Another representation for Boolean functions that is used in many parts of this work is the decision tree representation. At first, we briefly introduce some graph and tree definitions, for more details on these concepts, see [Louppe, 2014].

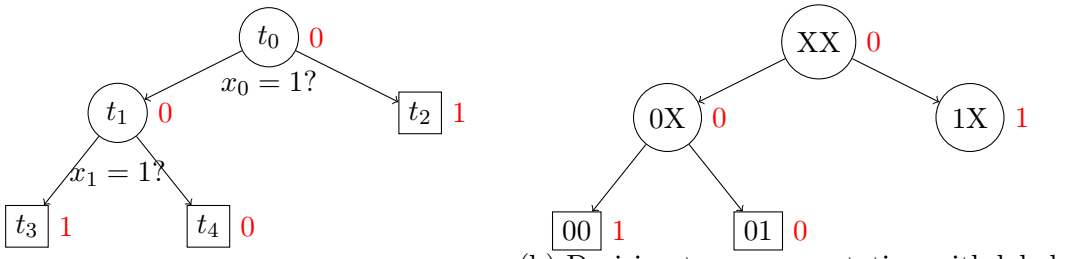
- A *graph* is an ordered pair $G = (V, E)$ comprising a set V of nodes and a set E of edges where each edge associates two vertices as an ordered pair.
- A *tree* is a graph $G = (V, E)$ in which any two nodes are connected by exactly one path.
- A *rooted tree* is a tree in which one of the nodes is designated as the root. We also considered the rooted tree as a directed graph where all edges are directed away from the root.
- For an edge from t_1 to t_2 in the tree, the node t_1 is said to be *parent* of node t_2, and t_2 is said to be child of node t_1.
- In a rooted tree, a node is said to be *internal* if it has at least one child, a node is said to be *terminal* or *leaf* if it has no children.

The following decision tree definition presents slight modifications from typical definitions with the purpose of representing more families of Boolean functions.

Definition 2.1.6. Let $\mathcal{X} = \{0, 1\}^n$ be a Boolean domain, a *decision tree* representation of a Boolean function $f : \mathcal{X} \to \{0, 1\}$ is given by a rooted tree where every node satisfies the following properties:

- A node t in the tree represents a subspace $\mathcal{X}_t \subseteq \mathcal{X}$ of the Boolean domain, in special the tree's root represents the whole domain $\mathcal{X}$, for illustration consider the decision tree in Figure 2.6(a).
- A node in the tree is internal or terminal, they are defined as follows:

  Internal nodes: an internal node t has at least one child and receives two labels, the first label indicates the division criteria of the subspace $\mathcal{X}_t$ into disjoint



(a) Decision tree representation.

(b) Decision tree representation with labels indicating the domain subspace.

Figure 2.6: *Decision trees representing the same Boolean function.*

subspaces $\mathcal{X}_{ti}, \ldots \mathcal{X}_{tj}$ (children nodes of t). And the second label indicates the output label $y_t \in \{0, 1\}$ of elements in $\mathcal{X}_t$ that are not present in none of the children of t, that is, for all elements $\mathbf{x} \in \mathcal{X}_t$ such that $\mathbf{x} \notin \mathcal{X}_{ti}, \ldots, \mathbf{x} \notin \mathcal{X}_{tj}$, we have $f(\mathbf{x}) = y_t$.

Terminal nodes or leaves: a terminal node t has only one label that indicates the output label $y_t \in \{0, 1\}$ of elements in $\mathcal{X}_t$, that is, for all elements $\mathbf{x} \in \mathcal{X}_t$, we have $f(\mathbf{x}) = y_t$.

Thereby, using this definition we can know the output of the function represented by a decision tree using Algorithm 2. The next example shows two decision tree representations of the same Boolean function.

Algorithm 2: Output of the function $f : \mathcal{X} \to \{0, 1\}$ represented by a decision tree.

Input: Input to be evaluated $\mathbf{x}$.
Output: Output $y = f(\mathbf{x})$.

```
1   /* t_0 is the tree's root */
2   t = t_0;
3   while t is not a terminal node do
4       if exists a child node t' of t such that x ∈ X_t' then
5           t = t';
6       else
7           break;
8       end
9   end
10  return y_t;
```

Example 2.1.6. In Figure 2.6 are illustrated two decision trees that represent the same Boolean function $f : \{0, 1\}^2 \to \{0, 1\}$. In the figure, internal nodes are represented by circles and terminal nodes by rectangles. Figure 2.6(a) shows the configuration described in Definition 2.1.6 where the two Boolean variables in the function are x_1 and x_2. We observe that the subspaces represented by nodes t_0, t_1, t_2, t_3 and t_4 are given by $\mathcal{X}_{t_0} = \{00, 01, 10, 11\}, \mathcal{X}_{t_1} = \{00, 01\}, \mathcal{X}_{t_2} = \{10, 11\}, \mathcal{X}_{t_3} = \{00\}$ and $\mathcal{X}_{t_4} = \{01\}$. The labels below the internal nodes describe the division criteria, if the statement is true, then we take the right path if it exists, otherwise, we take the left path if it exists. Besides, red labels in the image represent the function output value for each node.

Thus, using Algorithm 2 we know that the outputs of Boolean function f is given by $f(00) = 1, f(01) = 0, f(10) = 1, f(11) = 1$. Notice that $f(11) = 1$ because we stop the algorithm process when $t = t_2$, so the output label of node t_2 is 1.

Figure 2.6(b) shows another way to illustrate the same tree. We let the labels inside tree nodes represent the set of elements comprised by the node, where character X in the label means that the Boolean variable can take either the value 0 or 1, that is, XX = {00, 01, 10, 11}, 0X = {00, 01}, 1X = {10, 11}, 00 = {00}, 01 = {01}. This way, labels indicating the division criteria are not necessary because the mentioned labels inside the nodes explain the division criteria by themselves.

More interestingly, using Definition 2.1.6 we observe that every node of a decision tree represents a subset of elements of the domain space $\mathcal{X}$, i.e. a node t in the decision tree is associated to a subset $\mathcal{X}_t$ such that $\mathcal{X}_t \subseteq \mathcal{X}$. Most importantly, notice that any subtree of the tree composed of node t and its descendants creates a partition of $\mathcal{X}_t$ independent from their node output labels. Thus, it is evident that any decision tree as defined in this work creates a partition of the domain space. For example, Figure 2.7 illustrates a decision tree and its corresponding partition.

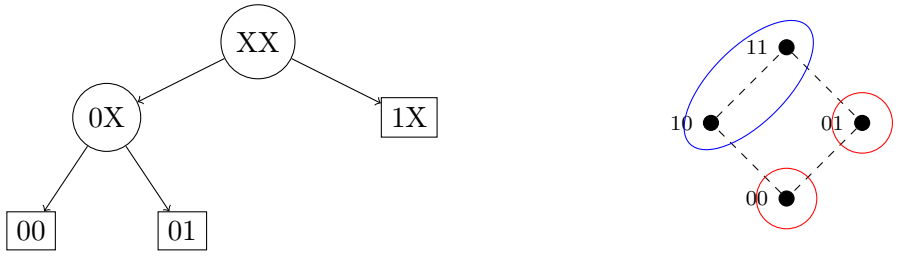


Figure 2.7: *Decision tree and its corresponding domain space partition.*

However, an important fact of the mentioned decision tree definition is that it allows the existence of more than one tree representing the same Boolean function. In the next section, we present a well-known problem in Boolean functions.

2.1.6 Partially defined Boolean functions

A common problem in the context of Boolean functions is when we the behaviour of a Boolean function in certain points are known, i.e. we know the output values for some points in $\{0,1\}^n$. A *partially defined Boolean function* is defined by a pair of sets T and F, denoted by pdBf(T, F), such that $T, F \subseteq \{0,1\}^n$, where T (resp., F) denotes a set of true points (resp., false points). A Boolean function f is called an *extension* of the pdBf(T, F) if $T \subseteq T(f)$ and $F \subseteq F(f)$ [Crama and Hammer, 2011]. Clearly, the disjointness of T and F is the only condition for the existence of an extension in the class of all Boolean functions. The next example shows some extensions of a partially defined Boolean function for the class of all Boolean functions.

Example 2.1.7. Let us consider $T = \{001, 100, 101\}$, $F = \{000, 010, 111\}$ and the pdBf(T, F). Then, all possible extensions of the pdBf(T, F) in the class of all Boolean functions are the Boolean functions f_1, f_2, f_3 and f_4 defined as:

$$T(f_1) = \{001, 100, 101, 011, 100\}; F(f_1) = \{000, 010, 111\}.$$
$$T(f_2) = \{001, 100, 101, 011\}; F(f_2) = \{000, 010, 111, 100\}.$$
$$T(f_3) = \{001, 100, 101, 100\}; F(f_3) = \{000, 010, 111, 011\}.$$

$$T(f_4) = \{001, 100, 101\}; F(f_4) = \{000, 010, 111, 011, 100\}.$$

However, finding out whether a pdBf(T, F) has an extension in a specific class of Boolean functions $\mathcal{C}$ is more complicated. In the case where there is no extension for a pdBf(T, F) in the class $\mathcal{C}$, we would like to find an extension which makes the smallest number of errors. Some approaches try to minimize a specific cost function such as the *best-fit extension problem* [Crama and Hammer, 2011] that is a well-known approach for finding extensions in a class $\mathcal{C}$ and it is explained below.

Definition 2.1.7. Assume a *weighting function* $w(x)$ defined for every point in $x \in T \cup F$, i.e. $w : T \cup F \to \mathbb{R}^+$, and for a subset $S \subseteq T \cup F$, we let $w(S) = \sum_{x \in S} w(x)$.

Definition 2.1.8. Given a pdBf(T, F), where $T, F \subseteq \{0, 1\}^n$ and a weighting function $w : T \cup F \to \mathbb{R}^+$, the objective of the *best-fit extension problem* is to find subsets T^* and F^* such that $T^* \cap F^* = \emptyset$ and $T^* \cup F^* = T \cup F$, for which pdBf(T, F) has an extension in $\mathcal{C}$, and $w(T^* \cap F) + w(F^* \cap T)$ is minimum.

The next example shows a best fit-extension of a pdBf(T, F).

Example 2.1.8. Let us consider $T = \{001, 100, 101\}$, $F = \{000, 010, 101\}$ and the pdBf(T, F). The weighting function w is given by $w(x) = 1$ for all $x \in T \cup F$. Since $T \cap F = \{101\}$, a best-fit extension in the class of all Boolean functions is obtained from the pdBf(T^*, F^*) defined by

$$T^* = T \setminus \{101\},$$
$$F^* = F.$$

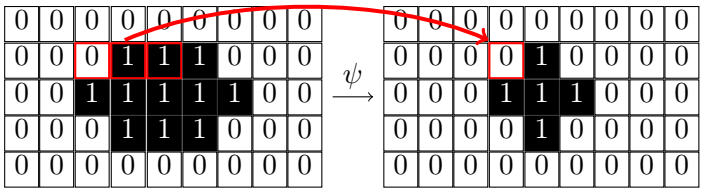
Later, in Section 2.3.1 we describe a real application of this problem. In the next section, we present binary morphological operators which are applications of Boolean functions.

2.1.7 Binary Morphological Operators

Mathematical morphology is a theory and technique for analysis of shapes based on the geometric and topological properties of an image or form, it was originally introduced by Matheron [1975] and Serra [1982]. In image processing, morphology operators are seen as mappings or transformations between images, some commonly used morphological operators are: erosion, dilation, opening, closing among others. Now, we define binary morphological operators using notations adopted from Hirata [2011].

Let $E = \mathbb{Z}^2$ be the Cartesian plane, the domain of the images where each point of E represents a pixel in a image. A binary image defined on E can be expressed as a function $f : E \to \{0, 1\}$. Informally, an image is a set of points (pixels) that assume the value 0 when it is a background pixel and the value 1 when is a foreground pixel. Thus, an image can be represented by a set $S = \{x \in E : f(x) = 1\}$. A mapping between two binary images most known as *binary morphological operators* is denoted by $\Psi : \{0, 1\}^E \to \{0, 1\}^E$, where $\{0, 1\}^E$ denotes the set of all possible images defined on E, or equivalently, $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$. For the sake of simplicity, we refer as *operators* any binary morphological operator.

Given a subset $X \subset E$, $X_h = \{x + h : x \in X\}$ defines the translate of X by h, where $x + y$ denotes the addition between two points $x, y \in E$.



x	$\psi(\mathbf{x})$
000	0
001	0
010	0
011	0
100	0
101	0
110	0
111	1

Figure 2.8: *Transformation using a W-operator.*

An operator $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ is translation-invariant if and only if, for any $h \in E$ and $X \in \mathcal{P}(E), \Psi(X_h) = \Psi(X)_h$.

Given a non-empty set $W \subseteq E$, most known as window, an operator $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ is locally defined within W if and only if, for any $h \in E$ and $X \in \mathcal{P}(E)$, $[\Psi(X)](h) = [\Psi(X \cap W_h)](h)$. Using these previous definitions, we define a W-operator as follows [Heijmans, 1994]:

Definition 2.1.9. An operator Ψ is called a W-operator if and only if, Ψ is translation-invariant and locally defined within W.

In [Heijmans, 1994], it is observed an important property of the W-operators, an operator $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ is a W-operator if and only if, exists a function $\psi : \mathcal{P}(W) \to \{0, 1\}$ such that

$$\Psi(S) = \{x \in E : \psi(S_{-x} \cap W) = 1\}$$

for any $S \subseteq \mathcal{P}(E)$. Consequently, any function $\psi : \mathcal{P}(W) \to \{0, 1\}$ represents a W-operator and vice-versa, i.e. we can use any representation of Boolean functions to represent a W-operator. The function ψ is called the characteristic function of Ψ. For the sake of simplicity, the function ψ is going to be referred as a W-operator when is treated in this context.

In Figure 2.8 is illustrated an image transformation using a W-operator, the red boxes in the figure indicate the window W and its corresponding mapping in the transformed image. Moreover, in the figure we observe the characteristic function ψ represented by its truth table.

2.2 Probability Theory

In this work, we use probability theory to mathematically model situations depending on chance. Some basic concepts of probability are presented in this section, more details can be seen in [DeGroot and Schervish, 2013].

The main element in probability theory is the *probability space*. A probability space is associated to an experiment is a tuple $(\Omega, \mathcal{F}, \Pr)$ where

- Ω is a sample space, which is a non-empty set of all possible outcomes,
- $\mathcal{F}$ is a set of events consisting of zero or more outcomes,
- Pr is a function from $\mathcal{F}$ to $[0, 1]$ that associates each event $E \in \mathcal{F}$ to a probability $\Pr(E)$.

The event set $\mathcal{F}$ is a σ-algebra, that is:

- $\Omega \in \mathcal{F}$,
- if $E \in \mathcal{F}$, then $(\Omega \setminus E) \in \mathcal{F}$,
- if $E_i \in \mathcal{F}$ for $i = 1, 2, \ldots$, then $(\cup_{i=1}^{\infty} E_i) \in \mathcal{F}$.

The probability function Pr satisfies the following axioms:

- $\forall E \in \mathcal{F}, 0 \leq \Pr(E) \leq 1$,
- $\Pr(\Omega) = 1$,
- if $\{E_i\}_{i=1}^{\infty} \subseteq \mathcal{F}$ is a countable collection of pairwise disjoint sets, then $\Pr(\cup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \Pr(E_i)$.

Conditional probability

The *conditional probability* of the event A given that event B has occurred is defined by

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)}.$$

The conditional probability $\Pr(A|B)$ is not defined if $\Pr(B) = 0$.

Discrete random variables

We can extend the notion of events by defining a *discrete random variable* X, which can take on any value from a finite or countably infinite set $\mathcal{X}$. A *probability distribution* of a random variable X is a mathematical function that provides probabilities for the different outcomes of X that can be characterized by a *probability mass function* $\Pr()$, then we denote the probability of the event that $X = x$ by $\Pr(X = x)$. Hence

$$\sum_x \Pr(X = x) = 1$$

as x runs through all possible values of X.

The *joint probability distribution* of two discrete random variables X, Y is characterized by a joint probability mass function such that:

$$\Pr(X = x, Y = y) = \Pr(Y = y|X = x) \cdot \Pr(X = x) = \Pr(X = x|Y = y) \cdot \Pr(Y = y),$$

where $\Pr(Y = y|X = x)$ is the probability of $Y = y$ given that $X = x$. These probabilities satisfies

$$\sum_x \sum_y \Pr(X = x, Y = y) = 1$$

as x runs through all possible values of X and y runs through all possible values of Y.

Let X be a random variable with a finite number of outcomes $x_1, x_2, \ldots, x_n$ occurring with probabilities $\Pr(X = x_1), \Pr(X = x_2), \ldots, \Pr(X = x_n)$, respectively. The *expected value* of X is defined as

$$\mathbb{E}[X] = x_1 \Pr(X = x_1) + x_2 \Pr(X = x_2) + \cdots + x_n \Pr(X = x_n).$$

In the next section, we will use some of these definitions to model uncertainty on data.

2.3 Machine Learning

Machine learning can be described as a set of methods that find patterns in data, and then use those patterns to predict future data under uncertainty [Murphy, 2012]. For example, a well-known problem in machine learning consists in finding patterns in images containing either cats or dogs and then classify them, this example will help us to explain the main concepts of machine learning in this chapter.

Usually machine learning is divided in three main types [Murphy, 2012]. The first one is the *supervised learning* where the goal is to learn a function for inputs $\mathbf{x}$ to outputs y given a set of input-output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$. The second type of machine learning is the *unsupervised learning*, this learning only receives as input a set of points $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and the goal is to find "interesting patterns" in data. The third type is *reinforcement learning*, which is the type less used, this kind of learning uses data to determine actions to maximize some notion of cumulative reward, some applications of this learning is to train machines to perform human tasks for example determining the best moves for a chess game.

In this work, we only use supervised learning. In the following sections we describe its components, theory and some applications.

2.3.1 Components of Learning

For explaining the different components of the learning problem we use the problem of classifying cats and dogs images. Suppose we have a set of images with either cats or dogs for training our learning method, so we want to find a formula that receives as input any image and should be able to predict whether this image corresponds to a cat or a dog. We will use supervised learning for solving this problem.

Supervised learning has the following components[Abu-Mostafa *et al.*, 2012]: the unknown target function $f : \mathcal{X} \to \mathcal{Y}$ (ideal function to distinguish between cats and dogs), where $\mathcal{X}$ is the input space (set of all possible cats or dogs images), and $\mathcal{Y}$ is the output space (set of all possible outputs, in this case the values representing a cat and a dog). Thus, f receives as input an element $\mathbf{x} \in \mathcal{X}$ (dog or cat image) and returns as output an element $y \in \mathcal{Y}$ (value representing a cat or a dog). Moreover, another component is a dataset $\mathcal{D}$ of input-output examples $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ drawn from a joint probability distribution $\Pr(X, Y)$, where $y_i = f(\mathbf{x}_i)$ for $i = 1, \ldots, N$ (examples of already classified dogs and cats images). Finally, a learning algorithm $\mathcal{A}$ and a hypothesis set $\mathcal{H}$ are the last components. The learning algorithm should be able to find a function $h : \mathcal{X} \to \mathcal{Y}, h \in \mathcal{H}$ that approximates f using the dataset $\mathcal{D}$. For example, if we represent each cat or dog image as a numerical vector and we choose $\mathcal{H}$ being the set of all linear functions, then algorithm $\mathcal{A}$ will choose a linear function $h \in \mathcal{H}$ that approximates f. Figure 2.9 illustrates the components of the learning problem.

2.3.2 Performance Evaluation

The purpose of learning is to have a good predictor outside of $\mathcal{D}$, so we can infer something outside $\mathcal{D}$ under the assumption that the inputs in $\mathcal{D}$ are picked independently according to some distribution Pr on $\mathcal{X}$ and $\mathcal{Y}$. Let X and Y be random variables from $\mathcal{X}$ and $\mathcal{Y}$ with respect to the joint probability distribution $\Pr(X, Y)$.

As mentioned before, algorithm $\mathcal{A}$ will choose a hypothesis $h \in \mathcal{H}$ whose predictions are as good as possible. Thus, we will find a hypothesis which minimizes its *out-of-sample*

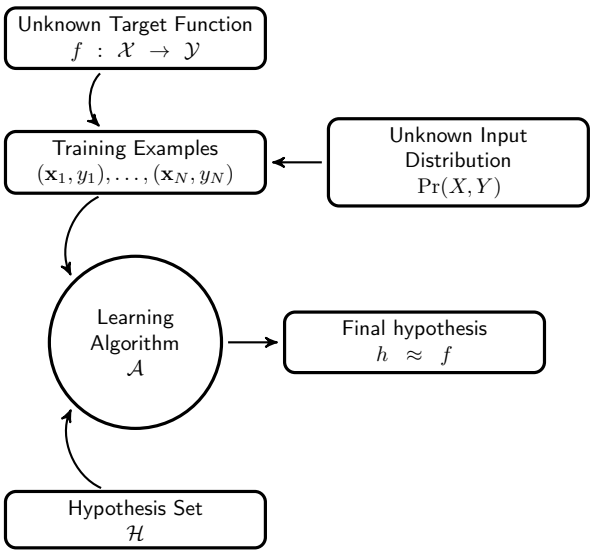


Figure 2.9: *Components of Learning, [Abu-Mostafa et al., 2012].*

error also known as *test error*, defined as follows:

$$E_{out}(h) = \mathbb{E}_{X,Y}[L(Y, h(X))] \tag{2.2}$$

where L is a *loss function* measuring the difference between its two arguments. Equation (2.2) measures the error of the hypothesis h evaluated in all the possible values of $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$. For classification, the most common loss function is the *zero-one loss function* defined as follows:

$$L(x, y) = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

Since the distribution $\Pr(X, Y)$ is usually unknown, Equation (2.2) cannot be used to evaluate the chosen hypothesis h. Instead, an estimation of the out-of-sample error usually known as *in-sample error* or *empirical risk* can be used, it is defined as follows:

$$E_{in}(h) = \frac{1}{N} \sum_{\mathbf{x}_i \in \mathcal{D}} L(y_i, h(\mathbf{x}_i)). \tag{2.4}$$

In the context of Boolean functions, minimizing this equation can be seen as obtaining a best-fit extension as seen in Section 2.1.6 when we consider a partially defined Boolean function pdBf(T, F) where T (resp., F) is the set of positive points (resp., false points) in the experiment and $w(x) = 1$ for every $x \in T \cup F$.

Unfortunately, the in-sample error could misinterpret the performance of the learning method because of many factors explained in the next section.

2.3.3 Feasibility of Learning

The main question about feasibility of learning is whether we can infer something outside the data using only $\mathcal{D}$. One way to solve this question is looking at the relationship between the out-of-sample error ($E_{out}(h)$) and the in-sample error ($E_{in}(h)$). To understand this relationship we introduce some important concepts in machine learning

such as the *growth function* and the *VC dimension*. For a better understanding, these concepts are explained using only binary target functions, i.e. $h \in \mathcal{H}, h : \mathcal{X} \to \{0, 1\}$.

For any function $h \in \mathcal{H}$, a *dichotomy* is an N-tuple that results when applying h to a finite sample of N points $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}$, that is, the dichotomy is the N-tuple $(h(\mathbf{x}_1), \ldots, h(\mathbf{x}_N))$, e.g. $(0, 1, 1, \ldots, 1, 0)$. Consequently, each $h \in \mathcal{H}$ generates one dichotomy on a sample, but there can be many functions $h \in \mathcal{H}$ that generates the same dichotomy on the same sample. The next definition [Abu-Mostafa *et al.*, 2012] shows the dichotomy set generated by a hypothesis set $\mathcal{H}$ on a specific sample.

Definition 2.3.1. Let $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}$. The dichotomies generated by $\mathcal{H}$ on these points are defined by

$$\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \{ (h(\mathbf{x}_1), h(\mathbf{x}_2), \ldots, h(\mathbf{x}_N)) \mid h \in \mathcal{H}\}. \tag{2.5}$$

The *growth function* (also known as shattering coefficient) measures the 'diversity' of a hypothesis set $\mathcal{H}$ and is defined as follows [Abu-Mostafa *et al.*, 2012]:

Definition 2.3.2. The growth function is defined for a hypothesis set $\mathcal{H}$ by

$$m_{\mathcal{H}}(N) = \max_{\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}} |\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)|, \tag{2.6}$$

where $|\cdot|$ denotes the cardinality of a set.

Thus, we say that the growth function measures diversity because it gives us the maximum number of dichotomies generated by the hypothesis set $\mathcal{H}$ on any N points in $\mathcal{X}$. Therefore, it is straightforward to show that $m_{\mathcal{H}}(N) \leq 2^N$ for any hypothesis set $\mathcal{H}$. We say that $\mathcal{H}$ can *shatter* the sample $\mathbf{x}_1, \ldots, \mathbf{x}_N$ when $\mathcal{H}$ can generate every possible configuration, that is $\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \{0, 1\}^N$.

The VC dimension is a very important concept in learning theory that is used to show some properties of any hypothesis set $\mathcal{H}$, it is defined as follows [Abu-Mostafa *et al.*, 2012]:

Definition 2.3.3. The Vapnik-Chervonenkis (VC) dimension of a hypothesis set $\mathcal{H}$, denoted by $d_{VC}(\mathcal{H})$ or simply d_{VC}, is the largest value of N for which $m_{\mathcal{H}}(N) = 2^N$. If $m_{\mathcal{H}}(N) = 2^N$ for all N, then $d_{VC}(\mathcal{H}) = \infty$.

As an example of these concepts we are going to find the formula of $m_{\mathcal{H}}(N)$ (growth function) and the $d_{VC}(\mathcal{H})$ (VC dimension) of the Positive rays hypothesis set.

Example 2.3.1. Let $\mathcal{H}$ be the *Positive rays* hypothesis set which consists of all functions $h : \mathbb{R} \to \{0, 1\}$ of the form

$$h(x) = \begin{cases} 0, & \text{if } x \leq a, a \in \mathbb{R}; \\ 1, & \text{otherwise.} \end{cases}$$

That is, all functions are defined in a one-dimensional space, they return 0 for every point to the left of a and return 1 for every point to the right of a as illustrated in Figure 2.10. Given N different points in a one-dimensional space, we notice that the line is split by the points in $N + 1$ regions and a dichotomy varies from others when the point a changes from one region to another, i.e. a can be in any position inside any region without modifying the result of the dichotomy, then we get at most $N + 1$ different dichotomies. Therefore, $m_{\mathcal{H}}(N) = N + 1$, and $d_{VC}(\mathcal{H}) = 1$ because this is the largest value of N for which $m_{\mathcal{H}}(N) = 2^N$.

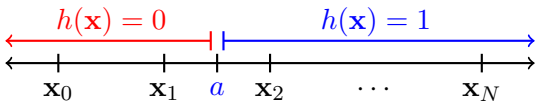


Figure 2.10: *Positive rays Hypothesis Set.*

The following example is going to be very useful in this work, it shows some properties of the hypothesis set composed of Boolean functions of n variables.

Example 2.3.2. Let $\mathcal{H}$ be the hypothesis set consisting of all Boolean functions of n variables, i.e. all functions of the form $h : \{0,1\}^n \to \{0,1\}$. First, notice that there are 2^{2^n} different Boolean functions of n variables, i.e. $|\mathcal{H}| = 2^{2^n}$. Now, for $N \leq 2^n$, we can select N different points in $\{0,1\}^n$, so we can obtain 2^N different dichotomies using functions in $\mathcal{H}$, then $m_{\mathcal{H}}(N) = 2^N$. On the other hand, for $N > 2^n$ we can select at most 2^n different points and select other $N - 2^n$ repeated points, then $m_{\mathcal{H}}(N) = 2^n$. Thus, $d_{VC}(\mathcal{H}) = 2^n$ because this is the largest value of N for which $m_{\mathcal{H}}(N) = 2^N$.

The following proposition [Anthony and Biggs, 1992] shows an upper bound for the VC dimension of finite hypothesis sets.

Proposition 2.3.1. *If $\mathcal{H}$ is a finite hypothesis space, then*

$$d_{VC}(\mathcal{H}) \leq lg|\mathcal{H}|.$$

As discussed in this chapter, the value of the in-sample error does not always generalize the value of the out-of-sample error. So we define as *generalization error* the discrepancy between these two errors. Now, we introduce the *generalization bound* [Devroye *et al.*, 1996], this bound gives us an important result in generalization theory, for any fixed tolerance δ, the bound shows whether the generalization error can be small enough for a sufficiently large sample. This bound is applied when the size of the hypothesis set is known and it applies to any hypothesis set $\mathcal{H}$, any learning algorithm $\mathcal{A}$, any probability distribution Pr, and any binary target function f. It is defined as follows:

Proposition 2.3.2. *The generalization bound is defined as: For any tolerance $\delta > 0$,*

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{1}{2N} \ln\left(\frac{2|\mathcal{H}|}{\delta}\right)}$$

with probability $\geq 1 - \delta$.

Similarly, the *VC generalization bound* [Abu-Mostafa *et al.*, 2012] can be obtained using the VC dimension and it is defined as follows:

Proposition 2.3.3. *The VC generalization bound is defined as: For any tolerance $\delta > 0$,*

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{8}{N} \ln\left(\frac{4((2N)^{d_{VC}} + 1)}{\delta}\right)}$$

with probability $\geq 1 - \delta$.

These bounds are really important as stated by Abu-Mostafa *et al.* [2012]: "The VC generalization bound is the most important result in the theory of learning. It

establishes the feasibility of learning with infinite hypothesis sets". Interpreting these bounds, we notice that if $d_{VC}(\mathcal{H}) = \infty$, then $m_{\mathcal{H}}(2N)$ is exponential, in that case, the generalization error does not converge to zero when N goes to infinity. Thus, as long as the VC dimension is finite, the generalization error will converge to zero for a sufficiently large sample.

Since the second term on the right hand side of Proposition 2.3.3 increases as the VC dimension increases, this term is also known as *penalty of the model complexity*. Because of the generality of the VC generalization bound, we can assume that this bound is not tight in most cases since the same bound has to cover all cases. Nevertheless, the VC analysis gives us an idea of how to compare the generalization performance for different learning models. In practice, it is observed that learning models with lower d_{VC} tend to generalize better than those with higher d_{VC} [Abu-Mostafa *et al.*, 2012].

Now, we show an interesting property about nested hypothesis sets that helps us to determine the most suitable hypothesis set for a learning problem [Abu-Mostafa *et al.*, 2012].

Proposition 2.3.4. *Let $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \mathcal{H}_3 \subseteq \cdots \subseteq \mathcal{H}_n$ be nested hypothesis sets, then we have that $d_{VC}(\mathcal{H}_1) \leq d_{VC}(\mathcal{H}_2) \leq d_{VC}(\mathcal{H}_3) \leq \cdots \leq d_{VC}(\mathcal{H}_n)$.*

Thereby, Figure 2.11 illustrates the curve generated by the out-of-sample error of nested hypothesis sets. Experimentally, it is observed that this curve usually decreases until a certain point, and increases afterwards, this phenomenon is known as the U-curve phenomenon that is explained in detail in Section 2.3.6.

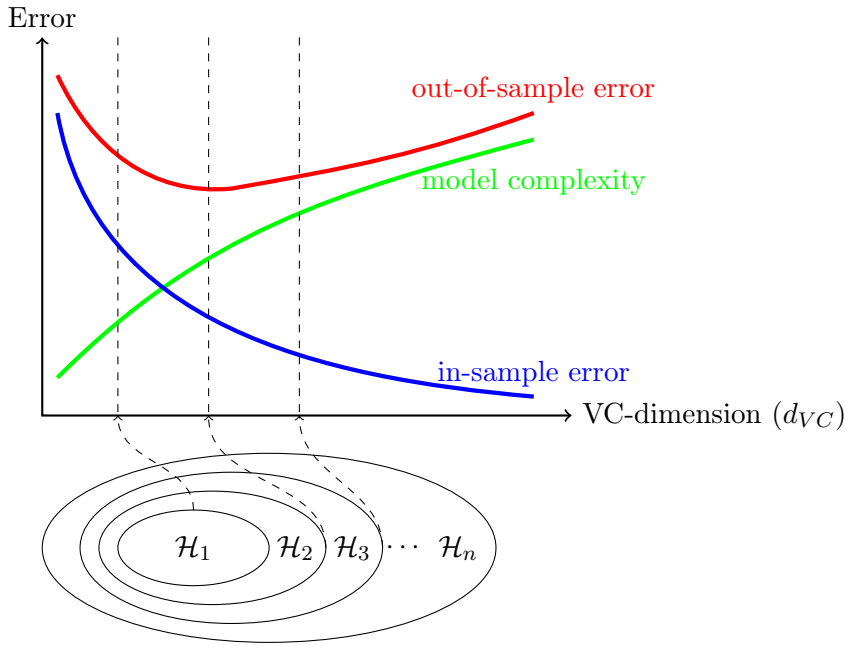
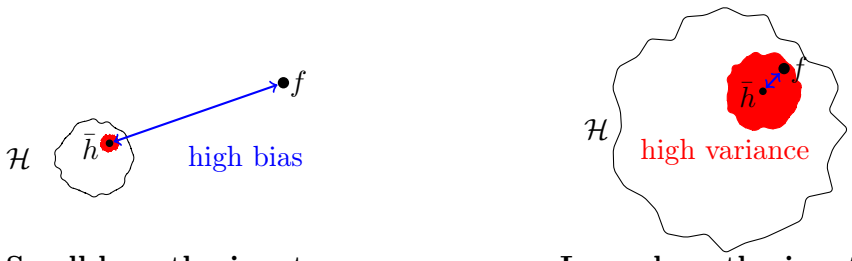


Figure 2.11: *Error curves of nested hypothesis sets, [Abu-Mostafa et al., 2012].*

Sample complexity

As a consequence of the VC generalization bound we have a similar result known as *sample complexity*. The sample complexity measures the number of examples N needed to achieve a good generalization performance. We will denote as ϵ the error



Small hypothesis set. **Large hypothesis set.**

Figure 2.12: *Bias and variance trade-off, in blue is represented the error due to bias and in red the error due to variance.*

tolerance, this is the allowed generalization error and we will denote as δ the confidence parameter, this is the probability of not getting the specified error tolerance. Using the generalization bound we have

$$N \geq \frac{1}{2\epsilon^2} \ln\left(\frac{2|\mathcal{H}|}{\delta}\right) \tag{2.7}$$

suffices to obtain generalization error at most ϵ with probability at least $1-\delta$. A similar result can be obtained using the VC dimension Abu-Mostafa *et al.* [2012]

$$N \geq \frac{8}{\epsilon^2} \ln\left(\frac{4((2N)^{d_{VC}}+1)}{\delta}\right) \tag{2.8}$$

suffices to obtain generalization error at most ϵ with probability at least $1-\delta$.

On real applications, these VC bounds are mostly used as a guideline for generalization rather than strict bounds to be used. For instance, Abu-Mostafa *et al.* [2012] point out that a popular rule of thumb is that the size of the sample N should be at least 10 times the VC dimension to obtain decent generalization.

The main objective of the VC bounds is to have a guarantee for minimizing the out-of-sample error, another way of seeing this issue is through the well-known problem called bias and variance trade-off which is presented in the next section.

Bias and variance trade-off

In learning, the out-of-sample error of our final hypothesis h can be decomposed into two components: error due to *bias* and error due to *variance*. In Figure 2.12 is illustrated the bias and variance trade-off, we briefly explain these concepts for understanding this phenomenon.

- **Error due to bias**
  This error measures how much the expected output function (average function $\bar{h}$) of the chosen hypothesis set deviates from the target function f. Smaller hypothesis sets have less chance to contain the target function, so the error due to bias is likely to be high.

- **Error due to variance**
  This error measures the variability of our final function, i.e. in average, how much our final hypothesis h deviates from the expected output function $\bar{h}$. Large hypothesis sets have higher chance to contain the target function, but the variability of our hypothesis set is likely to be high.

Unfortunately, in most cases the bias and variance cannot be calculated in practice because the target function and the expected output function of our hypothesis set are unknown. Nevertheless, an explicit trade-off exists between bias and variance, where decreasing one increases the other [Raschka, 2015].

Another common issue when designing learning methods is overfitting that will be presented in the next section.

Overfitting

As Abu-Mostafa *et al.* [2012] states, *overfitting* or *overtraining* is the phenomenon where the learning model fits the data more than is warranted, that is, our model does not generalize as expected, it picks a hypothesis with a decent in-sample error but a poor out-of-sample error. Overfitting usually occurs when the learning algorithm was performed for too long time in a large hypothesis set and/or there exists too much noise in the training data.

There are several techniques to decrease the chance of overfitting such as: cross-validation, regularization, pruning, early stopping, etc. The fundamentals of these techniques are either penalize the use of complex models or improve the generalization of the model testing the output models using data not used for training.

In summary, in this section we have discussed basic theory for understanding the process of designing machine learning methods. In the next section, we present some machine learning applications using this theory.

2.3.4 Machine Learning Applications

In this section, we study some known machine learning applications for learning Boolean functions, we start by presenting a basic learning method called monomial learning, then we present more complex methods such as W-operator learning, multiresolution W-operator learning and Boolean decision tree learning.

Monomial Learning

In learning theory, one of the most basic applications is the monomial concept learning [Anthony and Biggs, 1992]. A monomial concept can be described as the representation of the attributes of a concept using a group of Boolean variables. For example, suppose that we want to describe the concept 'keyboard' and we have to decide whether the following attributes corresponds to this concept.

- Electronic - yes
- Alive -no
- Coloured gray - irrelevant
- Transportable - yes
- Consumable - no

Then, the following Boolean function defines a keyboard : $\{0,1\}^5 \to \{0,1\}$ as follows.

$$\text{keyboard}(u_1u_2u_3u_4u_5) = \begin{cases} 1, & \text{if } u_1 = 1, u_2 = 0, u_4 = 1, u_5 = 0; \\ 0, & \text{otherwise.} \end{cases} \tag{2.9}$$

For convention, the function keyboard is represented as $u_1\overline{u_2}u_4\overline{u_5}$, where the value of the function is 1 if and only if the first bit is 1, the second is 0, the third is irrelevant, the fourth is 1 and the fifth is 0.

More formally, let $\{u_1, u_2, \ldots, u_n\}$ be a set of Boolean variables. A *literal* is a Boolean variable u_i or its negation $\overline{u_i}$. A *monomial* is a propositional Boolean formula that is expressed as a conjunction (AND) of literals, e.g. $u_1\overline{u_2}u_4\overline{u_5}$.

Now, we present a learning algorithm for monomials, but first let us identify the components of the problem using the definitions in Section 2.3.1. First, we know that the unknown target function is a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Second, we are given a training sample $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, where $\mathbf{x}_i \in \{0,1\}^n$ and $y_i \in \{0,1\}$, for $0 \leq i \leq N$. The hypothesis set $\mathcal{H}$ in this problem is the set of monomials using literals from the set $u_1, \overline{u_1}, u_2, \overline{u_2}, \ldots, u_n, \overline{u_n}$, so we have to find a function $h \in \mathcal{H}$ such that h approximates f.

Using this setup, Valiant [1984] presented Algorithm 3 for the monomial learning problem.

Algorithm 3: Monomial learning [Valiant, 1984].

Input: $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$.
Output: Monomial h.

```
1  set U = {u_1, ū_1, u_2, ū_2, ..., u_n, ū_n};
2  for i = 1 to N do
3      if y_i = 1 then
4          for j = 1 to n do
5              if (x_i)_j = 1 then
6                  delete ū_j if present in U;
7              else
8                  delete u_j if present in U;
9              end
10         end
11     end
12 end
13 let h_U denote the monomial formula containing the literals in the set U;
14 return h_U;
```

In Example 2.3.3 is presented an example of Algorithm 3 execution.

Example 2.3.3. We will present the execution of Algorithm 3 for the following training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}$, $= \{(110110, 1), (010101, 0), (100100, 1), (110101, 1)\}$, where $\mathbf{x}_i \in \{0,1\}^6$ and $y_i \in \{0,1\}$ for $0 \leq i \leq 4$. At the beginning of the algorithm, the initial set U is given by $U = \{u_1, \overline{u_1}, u_2, \overline{u_2}, \ldots, u_6, \overline{u_6}\}$. At the end of the main loop (line 12) in each iteration i we have:

- for $i = 1$, $U = \{u_1, u_2, \overline{u_3}, u_4, u_5, \overline{u_6}\}$.
- for $i = 2$, $U = \{u_1, u_2, \overline{u_3}, u_4, u_5, \overline{u_6}\}$.
- for $i = 3$, $U = \{u_1, \overline{u_3}, u_4, \overline{u_6}\}$.
- for $i = 4$, $U = \{u_1, \overline{u_3}, u_4\}$.

Thus, the final hypothesis obtained is $h_U = u_1\overline{u_3}u_4$, notice that the second example is not considered because $y_2 = 0$. It is interesting observing that the number of points in the dataset can change completely the final hypothesis, if we would consider as dataset only the first three points of $\mathcal{D}$, we would obtain as output the final hypothesis $h_U = u_1\overline{u_3}u_4\overline{u_6}$.

W-operator Learning

Another interesting application is the learning of binary W-operators which were presented in Section 2.1.7. This particular problem was introduced in earlier works such as [Dougherty and Loce, 1994] and [Barrera *et al.*, 1997], the formulation of this problem is presented as follows. At first, a set of image pairs is given $\{(I_1, O_1), \ldots, (I_M, O_M)\}$, each pair consists of an image to be processed (input image) and its corresponding transformed image (output image). The objective of the problem is estimating a W-operator Ψ such that $\Psi(I_i) \approx O_i$, for $0 \leq i \leq M$, or equivalently estimate its characteristic function ψ as discussed in Section 2.1.7.

The training data is collected as follows, a window W is slided for each input image, so for each pixel in each image, we have a realization of a random Boolean vector X representing an image patch observed by W and a realization of a random variable Y representing its corresponding binary output in the output image. All realizations of (X, Y), i.e. $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, become the training dataset $\mathcal{D}$ for the classifier ψ to be estimated. In Figure 2.13 is illustrated the training dataset extraction.

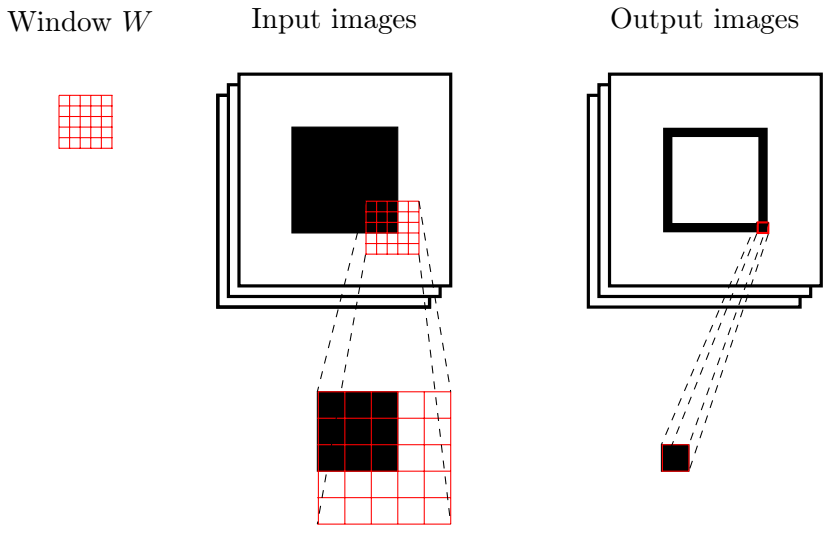


Figure 2.13: *Training set extraction for W-operators.*

To Barrera *et al.* [1997], estimating the W-operator ψ is formalized as follows. Assume that all image pairs (I, O) are jointly stationary, we want to estimate a W-operator Ψ characterized by ψ such that we minimize the out-of-sample error defined as:

$$E_{out}(\psi) = \mathbb{E}_{X,Y}[L(Y, \psi(X))] \tag{2.10}$$

assuming that (X, Y) is a local joint process where X is a random vector which realizations are in $\{0,1\}^{|W|}$ and Y is a random variable which realizations are in $\{0,1\}$.

Since the distribution $\Pr(X, Y)$ is unknown, it is common to minimize the in-sample error defined as:

$$E_{in}(\psi) = \frac{1}{N} \sum_{\mathbf{x}_i \in \mathcal{D}} L(y_i, \psi(\mathbf{x}_i)) \tag{2.11}$$

Minimizing this equation is trivial when estimated $\Pr(Y = 1|X = \mathbf{x})$ and $\Pr(Y = 0|X = \mathbf{x})$ from the training dataset $\mathcal{D}$, we just need to define the operator ψ as

$$\psi(\mathbf{x}) = \begin{cases} 1, \text{ if } \Pr(Y = 1|X = \mathbf{x}) \geq \Pr(Y = 0|X = \mathbf{x}); \\ 0, \text{ otherwise.} \end{cases} \tag{2.12}$$

In this way, it is minimized the in-sample error, notice that for samples $\mathbf{x} \in \{0,1\}^{|W|}$ that do not belong to the dataset $\mathcal{D}$ (the so-called *don't cares*) we have that $\Pr(Y = 1|X = \mathbf{x})$ and $\Pr(Y = 0|X = \mathbf{x})$ are not estimated, in consequence don't cares are labeled without any criteria, so the generalization power of ψ is mediocre.

Overall, there exist many algorithms proposed in the literature that solve this problem, mentioning just a few of the most important approaches we have Decision Trees algorithms [Quinlan, 1986], Support vector machine algorithms [Cortes and Vapnik, 1995] and Neural Networks algorithms [Haykin, 1998]. Specifically for W-operator learning, different approaches were presented such as [Barrera *et al.*, 1997], [Dougherty *et al.*, 2001], [Kim, 2001], [Hirata Junior *et al.*, 2002], [Martins *et al.*, 2006], [Hirata, 2009], [Santos *et al.*, 2010] and [Montagner *et al.*, 2017]. One of these approaches is the multiresolution W-operators explained in the following section.

Multiresolution W-operator learning

For the first time, multiresolution W-operators and their learning were presented in [Dougherty *et al.*, 2001], we can define a multiresolution W-operator as any W-operator which is represented by a Boolean function with multiresolution representation. As we have seen in Section 2.1.4, for defining such Boolean functions we need a set of nested Boolean variables sets $W_0, W_1, \ldots, W_n$ and element sets $\mathcal{X}_{00}, \mathcal{X}_{01}, \mathcal{X}_{10}, \mathcal{X}_{11}, \ldots, \mathcal{X}_{n0}, \mathcal{X}_{n1}$ satisfying some properties defined in Section 2.1.4.

The training data for the multiresolution W-operator is collected in a similar way compared to the W-operator training in the previous section. Similarly, a set of image pairs is given $\{(I_1, O_1), \ldots, (I_M, O_M)\}$, each pair consists of an image to be processed (input image) and its corresponding transformed image (output image). By contrast, in multiresolution W-operator learning a set of nested windows is given $W_0, W_1, \ldots, W_n$ which are slided for each image of the input set. So for each pixel in each image, we have realizations of random Boolean vectors $X_0, X_1, \ldots, X_n$ representing image patches observed by $W_0, W_1, \ldots, W_n$ and realizations of a random variables Y representing the binary output in the output image.

As mentioned in Section 2.1.4, we can induce the domain spaces $\mathcal{X}_0 = \{0,1\}^{W_0}, \mathcal{X}_1 = \{0,1\}^{W_1}, \ldots, \mathcal{X}_n = \{0,1\}^{W_n}$ and their respective multiresolution down-samplings $\rho_1 : \mathcal{X}_0 \to \mathcal{X}_1$, $\rho_2 : \mathcal{X}_1 \to \mathcal{X}_2, \ldots, \rho_n : \mathcal{X}_{n-1} \to \mathcal{X}_n$. Moreover, we consider the set $\mathcal{Y} = \{0,1\}$.

Now, we define the following sets: for $i \in \{0, \ldots, n\}$, let $\mathcal{D}_i \in (\mathcal{X}_i, \mathcal{Y})$ be the set of all realizations of the pair (X_i, Y). Using definitions in [Dougherty *et al.*, 2001], we define

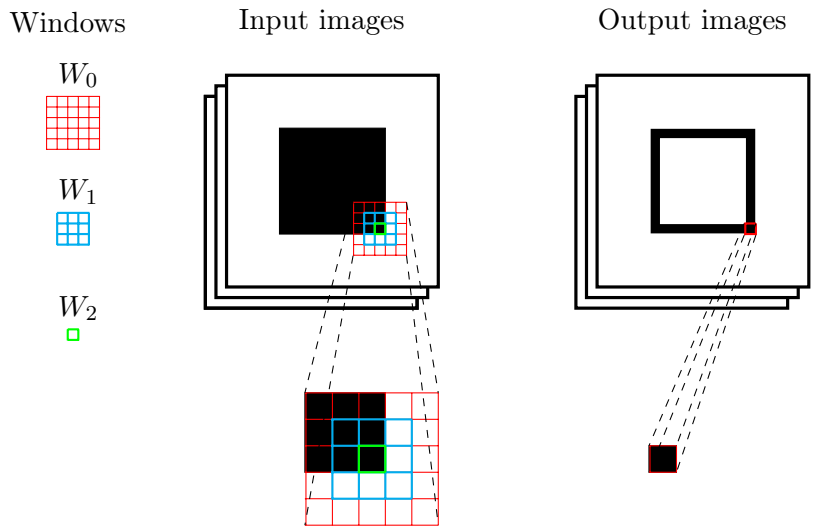
the element sets $\mathcal{X}_{i0}, \mathcal{X}_{i1}, 0 \leq i \leq n$ as follows: for each $i \in \{0, \ldots, n\}$,

$$\mathcal{X}_{i1} = \{\mathbf{x} \in \mathcal{X}_i : (\mathbf{x}, y) \in \mathcal{D}_i \text{ and } \Pr_{\mathcal{D}_i}(Y = 1 | X_i = \mathbf{x}) \geq \Pr_{\mathcal{D}_i}(Y = 0 | X_i = \mathbf{x})\},$$

$$\mathcal{X}_{i0} = \{\mathbf{x} \in \mathcal{X}_i : (\mathbf{x}, y) \in \mathcal{D}_i \text{ and } \Pr_{\mathcal{D}_i}(Y = 1 | X_i = \mathbf{x}) < \Pr_{\mathcal{D}_i}(Y = 0 | X_i = \mathbf{x})\},$$

where $\Pr_{\mathcal{D}_i}(Y = 1 | X_i = \mathbf{x})$ and $\Pr_{\mathcal{D}_i}(Y = 0 | X_i = \mathbf{x})$ are probability estimations using only the set $\mathcal{D}_i$.

In Figure 2.14 is illustrated the training data extraction of a multiresolution W-operator.



$$\mathbf{x}^0 = 111001110011100000000000, \qquad y = 1,$$
$$\mathbf{x}^1 = 110110000,$$
$$\mathbf{x}^2 = 1,$$
$$\mathcal{D}_0 = \{(\mathbf{x}^0_1, y_1), (\mathbf{x}^0_2, y_2), \ldots, (\mathbf{x}^0_{N_0}, y_{N_0})\},$$
$$\mathcal{D}_1 = \{(\mathbf{x}^1_1, y_1), (\mathbf{x}^1_2, y_2), \ldots, (\mathbf{x}^1_{N_1}, y_{N_1})\},$$
$$\mathcal{D}_2 = \{(\mathbf{x}^2_1, y_1), (\mathbf{x}^2_2, y_2), \ldots, (\mathbf{x}^2_{N_2}, y_{N_2})\},$$
$$\mathcal{X}_{01} = \{\mathbf{x}^0, \ldots\}, \mathcal{X}_{11} = \{\mathbf{x}^1, \ldots\}, \mathcal{X}_{21} = \{\mathbf{x}^2, \ldots\}.$$

Figure 2.14: *Training set extraction for multiresolution W-operators.*

It is worth to mention that in [Dougherty *et al.*, 2001] the element sets $\mathcal{X}_{i0}, \mathcal{X}_{i1}, 0 \leq i \leq n$ are not defined, instead a W-operator set $(\psi_0, \psi_1, \ldots, \psi_n)$ is defined using the realization sets $\mathcal{D}_i, 0 \leq i \leq n$. In this work, we preferred to use a notation compatible with the multiresolution representation for Boolean functions described in Section 2.1.4.

Now, having all element sets $\mathcal{X}_{i0}, \mathcal{X}_{i1}, 0 \leq i \leq n$, we define a multiresolution W-operator ψ similarly to the multiresolution representation seen in Section 2.1.4, that is:

$$\psi(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in \mathcal{X}_{00} \\ 1, & \text{else if } \mathbf{x} \in \mathcal{X}_{01} \\ 0, & \text{else if } \rho_1(\mathbf{x}) \in \mathcal{X}_{10} \\ 1, & \text{else if } \rho_1(\mathbf{x}) \in \mathcal{X}_{11} \\ \vdots & \\ 0, & \text{else if } \rho_n(\dots(\rho_2(\rho_1(\mathbf{x})))\dots) \in \mathcal{X}_{n0} \\ 1, & \text{else if } \rho_n(\dots(\rho_2(\rho_1(\mathbf{x})))\dots) \in \mathcal{X}_{n1} \\ 0, & \text{otherwise.} \end{cases} \tag{2.13}$$

On real applications, [Dougherty *et al.*, 2001] showed that multiresolution W-operators perform much better than single W-operators as the size of W gets larger, this occurs because of the good generalization ability of the multiresolution W-operators.

Boolean decision tree learning

In Section 2.1.5 were introduced the decision trees representing Boolean functions also known as Boolean decision trees. Now, we present the process of finding a Boolean function h that approximates an unknown target Boolean function $f : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} = \{0,1\}^n$ and $\mathcal{Y} = \{0,1\}$ such that h can be represented by a Boolean decision tree. This process consists in finding the best tree structure such that the space partition generated by the tree is the closest to the partition engendered by $\mathcal{Y}$ over $\mathcal{X}$. Since this partition is unknown, our objective is finding a tree structure which partitions $\mathcal{X}$ according to the dataset $\mathcal{D}$ [Louppe, 2014].

The common approach to solve this problem is finding a Boolean function represented by a decision tree that minimizes the in-sample error, however there may exist many of these functions with minimum in-sample error and also each of these functions may be represented by different decision trees. Nevertheless, as shown by Quinlan and Rivest [1989], finding the smallest tree with the minimum in-sample error is an NP-complete problem. This fact suggests the construction of non-optimal decision trees guaranteeing efficient computational complexity.

In this work, we consider two classes of Boolean decision trees. The first class are the Boolean decision trees with predetermined division criteria, that is, at the beginning of the algorithm the division criterion of all the internal nodes in the tree are already defined. For instance, a common division criterion used is the values of a predetermined set of Boolean variables as seen in Example 2.1.6.

The second class are the Boolean decision trees with automatic division criteria. For example, one of the most used criteria is the measure known as *impurity* introduced by [Breiman *et al.*, 1984]. The impurity evaluates the goodness of any node in the decision tree, the smaller is this measure, the purer the node, so better the predictions for that node. Formally, the impurity of a node is defined by different metrics, the most common metric is the impurity based on the Gini index defined as follows:

Definition 2.3.4. The impurity function $i_G(t)$ based on the Gini index [Gini, 1912] is:

$$i_G(t) = \sum_{c=0}^{1} p(c|\mathcal{X}_t)(1 - p(c|\mathcal{X}_t))$$

where $\mathcal{X}_t$ is the subspace represented by the node t.

Breiman *et al.* [1984] make the greedy assumption to grow a decision tree selecting the Boolean variable that maximize the decrease of impurity of the resulting child nodes, thereby seeking for good generalization. Formally, this decrease of impurity is defined as follows:

Definition 2.3.5. The *impurity decrease* of a split using the Boolean variable s that divide the node t into a left node t_L and a right node t_R is [Louppe, 2014]:

$$\Delta(s,t) = i_G(t) - \frac{N_{t_L}}{N_t} i_G(t_L) - \frac{N_{t_R}}{N_t} i_G(t_R)$$

where N_t is the size of the subset $\mathcal{X}_t$.

Furthermore, decision trees usually uses the so-called *stopping criteria* for deciding if a node should be terminal or not. There are different stopping criteria, some of them are described below:

- Depth of the node is more than some pre-specified limit.
- All the predictor values in $\mathcal{X}_t$ are equal, so the split is not necessary.
- Purity of the node is more than some pre-specified limit.
- Number of nodes in the tree is more than some pre-specified limit.

Using these preliminaries, Algorithm 4 is presented. In the algorithm we let the user decide if the split criteria will be predetermined or it will be calculated by the algorithm using a specified metric. Moreover, the user must define the stopping criteria beforehand.

In summary, as stated by [Louppe, 2014], since the visualization of the tree construction is straightforward, decision tree learning is simple to understand and interpret and we can see the whole process as a white box. On the other hand, decision trees are not accurate as other methods because they usually have low bias and high variance, and they have a high chance to overfit data. In the next section, we will present how the decision trees can improve their performance by using ensemble techniques.

2.3.5 Ensemble Methods

Ensemble methods are well-established techniques for combining several machine learning hypotheses into one predictive hypothesis which is called *ensemble*. Dietterich [2000] states that ensembles are often much more accurate than the individual hypothesis that make them up and he identifies three fundamental reasons why ensembles often work better than single hypothesis.

The first reason is statistical. As mentioned in Section 2.3.1, the learning algorithm $\mathcal{A}$ can find several functions in the hypothesis space $\mathcal{H}$ with the same performance, this problem usually arises when the training data is too small for the hypothesis set chosen. By constructing an ensemble of different hypotheses in $\mathcal{H}$, the algorithm can average their prediction reducing the risk of choosing the wrong hypothesis.

The second reason is computational. Most learning algorithms works by making greedy assumptions and performing some local search in the hypothesis set $\mathcal{H}$. For example as mentioned in Section 2.3.4, the decision tree learning employs a greedy splitting rule to grow decision trees. Therefore, an embedding constructed by individual

Algorithm 4: Learning a decision tree of function $f : \mathcal{X} \to \{0, 1\}$.

Input: $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$.
Output: Decision tree of a Boolean function.

```
Create a decision tree f with root node t_0;
Create en empty stack S of pairs (tree node, dataset);
S.PUSH((t_0, D));
while S is not empty do
    t, D_t = S.POP();
    y_t = the label determined for D_t;
    if the stopping criteria is met for t then
        continue;
    end
    if s* is not predetermined then
        Find a Boolean variable s* such that it maximizes the impurity
            decrease;
    end
    Partition D into D_1, ..., D_n according to s*;
    foreach partition D_i of D do
        Create a child node t_i of t S.PUSH((t_i, D_i));
    end
end
return f;
```

hypotheses obtained from searches with different starting points may provide a better approximation of the true unknown function than individual hypotheses.

The third reason is representational. In most machine learning applications the selected hypothesis set $\mathcal{H}$ does not contain the true unknown function. By combining individual hypotheses it may be possible to expand the hypothesis space of representable functions and find a better hypothesis.

These three issues are the most important reasons why most learning algorithms fail [Dietterich, 2000]. Therefore, ensemble methods can be really useful for improving the performance of machine learning algorithms.

There exist many methods for constructing ensembles in the literature, some common types of ensembles are briefly mentioned below:

- **Bootstrap aggregating (bagging)**
  In bagging, each hypothesis in the ensemble votes with equal weight. In order to reduce variance, bagging trains each hypothesis in the ensemble with a different training set consisting of a sample of N training elements drawn randomly with replacement from the original training set of N elements. As an example, the random forest method [Breiman, 2001] combines random decision trees and bagging as we will present in the next section.

- **Boosting**
  In boosting, each hypothesis in the ensemble is learned iteratively, in each iteration we learn a hypothesis using a modified training set which emphasizes the training instances misclassified by previous hypotheses. The most common application of boosting is AdaBoost [Freund and Schapire, 1997].

- **Stacking**
  In stacking (or stacked generalization), all hypotheses in the ensemble are used as input of another learning algorithm known as a *combiner*. The most common combiner used in practice is the logistic regression method [McCullagh and Nelder, 1989].

As an example, in the next section we present the random forest method that is one of the most popular ensemble methods.

Random Forests

Random forest is a popular and powerful ensemble method named by Breiman [2001], in this section we will describe briefly the most important aspects of this method. Random forest is an ensemble of several decision trees with two characteristics explained below.

- Since decision trees are known to have low bias and high variance, random forest uses bagging as ensemble method. Breiman [1996] shows theoretically and empirically the benefit of using bagging of decision trees in order to reduce the error due to variance.

- Introduced by [Amit *et al.*, 1997], random forest grows each decision tree searching the best split at each node over a random subset of variables.

Breiman [2001] shows that random forests are easy to understand and interpret, he presents results that are competitive with other successful approaches in the literature. It is verified that random forest runs efficiently on large data and most importantly they do not tend to overfit.

Another important process in a learning method construction is feature selection which is presented in the next section.

2.3.6 Feature Selection

In machine learning, feature selection is the process of identifying and selecting the most useful subset of features from data. It is usually used to deal with the excessive number of features in data to reduce computational complexity and reduce overfitting. There exist several algorithms to perform feature selection as we can see in detail in [Chandrashekar and Sahin, 2014].

Following the definitions in [Ris *et al.*, 2010], the feature selection problem can be represented by a complete lattice $(\mathcal{P}(X), \subseteq)$ associated to a cost function c, where X is a finite set of features, $\mathcal{P}(X)$ is the collection of all subsets of X, and $\subseteq$ represents the usual inclusion on sets. Hence, the search space is composed by $n = 2^{|X|}$ elements. The cost function $c : \mathcal{P}(X) \to \mathbb{R}$ is associated to every element in the lattice, this representation is illustrated in Figure 2.15.

Using these definitions, the objective of the feature selection problem is to find a subset in $\mathcal{P}(X)$ that minimizes the cost c. Unfortunately, in a huge search space this problem is computationally unfeasible and for most cost functions this problem is NP-hard. Thereby, most known approaches for this combinatorial problem are meta-heuristics, these heuristics range in complexity from simple local search algorithms to complex global search algorithms [Chandrashekar and Sahin, 2014].

Next, we present a common phenomenon presented when we perform feature selection.

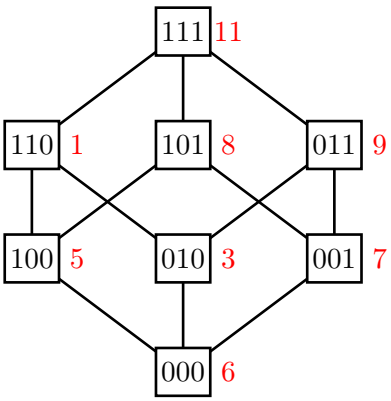


Figure 2.15: *Feature selection problem, a subset of features is represented by strings of ones and zeros where one (zero) indicates that the variable is present (not present). The costs associated to each node are shown in red.*

U-curve phenomenon

The *peaking phenomenon* or *U-curve phenomenon* arises when we estimate the out-of-sample error of a designed classifier using different subsets of features in the problem. It is observed that as the number of features considered in the model increases, the estimation error tends to decrease until a certain point and tends to increase afterwards. In general, most feature selection algorithms try to minimize a specified cost function to find the best subset of features. It was observed that several of these cost functions present the so-called *U-curve property*, they are functions defined as being monotone non-increasing and then monotone non-decreasing.

Formally, the *U-curve problem* is a search problem in a complete lattice $(\mathcal{P}(S), \subseteq)$ embedded with a cost function $c : \mathcal{P}(S) \to \mathbb{R}$, such that the cost function c has the U-curve property in any path from the bottom to the top of the lattice. The U-curve property states that for any elements $A, B, C \in \mathcal{P}(S)$ such that $A \subseteq B \subseteq C$ it holds that $\max(c(A), c(C)) \geq c(B)$. The goal in the U-curve problem is to find the element $X \in \mathcal{P}(S)$ with minimum cost, i.e. $c(X) \leq c(Y), \forall X, Y \in \mathcal{P}(S)$. This problem is proved to be NP-complete in [Reis, 2013].

In the literature, there are approaches that solve the U-curve problem and variations of this problem such as [Ris *et al.*, 2010], [Atashpaz-Gargari *et al.*, 2013], [Reis, 2013] and [Atashpaz-Gargari *et al.*, 2018]. Furthermore, Reis *et al.* [2017] present a practical framework for feature selection algorithms and cost functions, it was also used in this work as we will see later in Section 4.3.

In the next section, it is presented another important subject in learning theory, this is the representation of hypothesis spaces.

2.3.7 Representations of Hypothesis Spaces

As we studied in this chapter, the hypothesis space in the learning process is the set of all hypotheses that might possibly be returned by a machine learning algorithm. Since there can be infinite representations of a hypothesis space, we will formalize these representations as follows. Anthony and Biggs [1992] define a *representation* to be a surjection $\Theta \to \mathcal{H}$ where Θ is a set of formulas constructed by some specific rules of formation. We denote as $\theta \in \Theta$ a formula such that the surjection assigns θ to its corresponding function $h_\theta \in \mathcal{H}$. Thus, a set of formulas Θ can induce a hypothesis

set that we denote as $\mathcal{H}_\Theta$ (the hypothesis set corresponding to the surjection). In Example 2.3.4, we observe a hypothesis set represented by the representation set Θ.

Example 2.3.4. Let $\mathcal{H}_\Theta$ be the set $h : \{0,1\}^2 \to \{0,1\}$ such that each h must be represented by a monomial formula. Considering as x_1 and x_2 the two variables of the Boolean function, $\mathcal{H}_\Theta$ is composed by 9 Boolean functions that are represented by $\Theta = \{\emptyset, x_1, x_2, \overline{x_1}, \overline{x_2}, x_1x_2, x_1\overline{x_2}, \overline{x_1}x_2, \overline{x_1}\,\overline{x_2}\}$. In other words, $\mathcal{H}_\Theta$ is the set of functions corresponding to the surjection $\Theta \to \mathcal{H}$ and it is presented in Figure 2.16.

x	$h_\emptyset$	h_{x_1}	h_{x_2}	$h_{\overline{x_1}}$	$h_{\overline{x_2}}$	$h_{x_1x_2}$	$h_{x_1\overline{x_2}}$	$h_{\overline{x_1}x_2}$	$h_{\overline{x_1}\,\overline{x_2}}$
00	0	0	0	1	1	0	0	0	1
01	0	0	1	1	0	0	0	1	0
10	0	1	0	0	1	0	1	0	0
11	0	1	1	0	0	1	0	0	0

Figure 2.16: *Hypothesis set of Boolean functions of two variables represented only by monomials.*

In learning theory, the last subject we will cover in this work is the model selection problem which is presented in the next section.

2.3.8 Model Selection

We define as *model* a hypothesis set with certain common properties, for example, a model with low complexity is the hypothesis set composed of only linear functions. Model selection is the task of selecting a model between M candidate models $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$. One of the most important characteristics of a good model selector is the ability to select the model that contains a hypothesis with the lowest out-of-sample error. As we mention in the previous subsection, the VC bounds gives a bound of the out-of-sample error using the in-sample error of the hypothesis using a dataset $\mathcal{D}$.

Another way to estimate the out-of-sample error is calculating the *validation error*, to calculate the validation error we partition the dataset $\mathcal{D}$ into two separate datasets, i.e. a *training dataset* $\mathcal{D}_{train}$ of size $N-K$ and a *validation dataset* $\mathcal{D}_{val}$ of size K, then we use the learning algorithm using the training set $\mathcal{D}_{train}$ to obtain the hypothesis $h^- \in \mathcal{H}$ (the minus superscript indicates that we only used $N-K$ points of the training dataset), next we calculate the validation error of h as:

$$E_{val}(h^-) = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{D}_{val}} L(y_i, h^-(\mathbf{x}_i))$$

As presented in [Abu-Mostafa *et al.*, 2012], the validation error can be used to achieve model selection estimating the out-of-sample error of each candidate model $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$. To accomplish this, we use the training set $\mathcal{D}_{train}$ to learn a hypothesis h_m^- for each model, then we estimate the out-of-sample error using the validation errors $E_{val}(h_m^-)$ for $m = 1, \ldots, M$, so the selected model will be the one with the lowest validation error. Now, let m^* be the index of the model with lowest validation error, we know that $E_{val}(h_{m^*}^-) \leq E_{val}(h_m^-)$, for $m = 1, \ldots, M$. Then, using the selected model we can learn the final hypothesis $h_{m^*} \in \mathcal{H}_{m^*}$ using the complete dataset $\mathcal{D}$. This process is illustrated in Figure 2.17.

This model selector presents successful results in practice, however proving statistically that the validation error is closer to the out-of-sample error is not an easy task. In

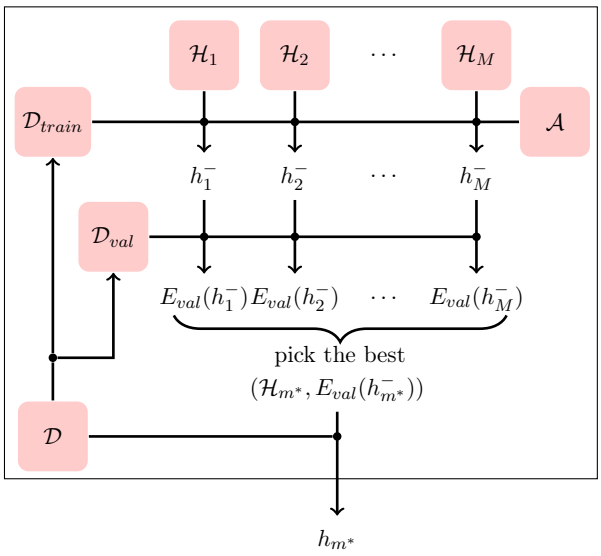


Figure 2.17: *Model selection process [Abu-Mostafa et al., 2012].*

[Abu-Mostafa *et al.*, 2012] we can see an interesting inequality to see the relationship between the out-of-sample error and the validation error.

$$E_{out}(h_{m^*}) \leq E_{out}(h_{m^*}^-) \leq E_{val}(h_{m^*}^-) + O\left(\sqrt{\frac{\ln M}{K}}\right).$$

The first inequality cannot be proved but in practice the out-of-sample error should be lower and more accurate using more data.

Chapter 3

Design of Boolean Models

A crucial aspect in the process of learning Boolean functions consists in selecting the best model that fits better our input data. Since every learning problem has its own characteristics, each instance must be treated differently. In fact, this aspect is usually solved by experts that, based on their knowledge and experience, select the best model for the problem or they use a model selector to pick one from a set of pre-designed models as shown in Section 2.3.8.

In this chapter, we introduce a new approach for designing Boolean function models based on partitions of the domain space; these models show interesting properties such as established size and fixed VC dimension. Furthermore, we propose a set of model selectors that exploit these mentioned properties to pick the best model according to our input data.

In summary, in Section 3.1 we present the design of hypothesis sets induced by partitions and their properties. Next, in Section 3.2 we present the aforementioned model selectors. Later, in Section 3.3 is presented an efficiency improvement of the proposed model selector using search algorithms in posets. Finally, in Section 3.4 we present real experiments of the model selectors.

3.1 Hypothesis sets induced by partitions

In this section, we discuss the importance of using partitions of the domain space in order to create learning models. So, we study how partitions can be used as restrictions on the definition of Boolean hypothesis sets obtaining some interesting properties that will help us to design model selectors.

We discussed in Section 2.3, that in the process of learning a Boolean function we need to find a Boolean function $h : \{0,1\}^n \to \{0,1\}$ in a hypothesis set $\mathcal{H}$ ($h \in \mathcal{H}$) given a sample $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \{0,1\}^n, y_i \in \{0,1\}, 0 \leq i \leq N$. In fact, for a fixed number of instances N, as the number of features n increases, the number of elements of the domain space $\{0,1\}^n$ that do not appear in the sample $\mathcal{D}$ increases. This fact implies that we do not have any previous labeling information of a big number of elements in $\{0,1\}^n$ when n is big, so our algorithm will have to predict the mapping of numerous unseen instances. For this reason, we will define hypothesis sets (or models) that group elements of $\{0,1\}^n$ such that all elements in the same group will be mapped to the same value.

Thereby, we study some theory of hypothesis sets and the element grouping in $\{0,1\}^n$. Suppose that our target function is defined in the space $h : \{0,1\}^2 \to \{0,1\}$,

then we have four elements to be labeled, these elements can be represented as a lattice as shown in Figure 3.1.

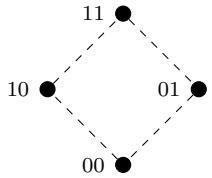


Figure 3.1: *Lattice representation of the set* $\{0,1\}^2$.

All the possible ways to group elements of $\{0,1\}^2$ can be seen as a partition lattice of $\{0,1\}^2$ illustrated in Figure 3.2.

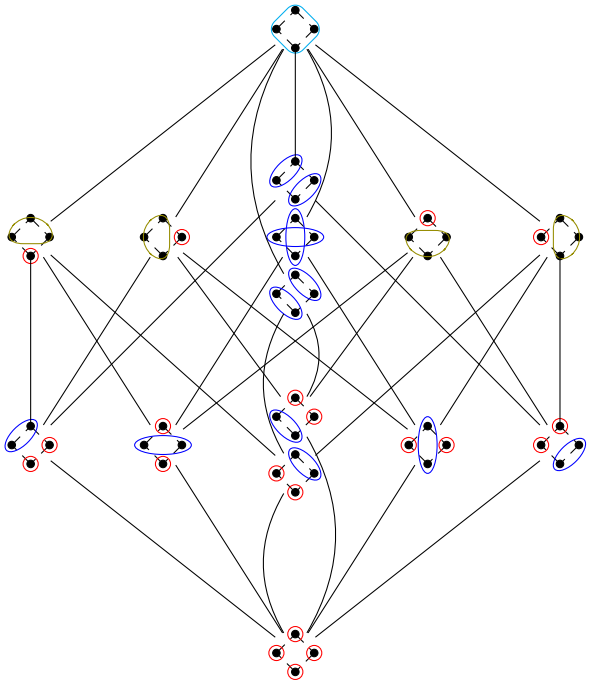


Figure 3.2: *Partition Lattice of the set* $\{0,1\}^2$.

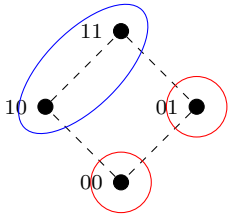
As we studied in Section 2.3.3, the size of the hypothesis sets composed of all the Boolean functions is exponential on the number of Boolean variables, one approach to reduce this size is using partition restrictions as we will explain in the next paragraphs. Given a partition π of the domain space $\{0,1\}^n$, we impose the following restriction: the same label must be given to all the elements in every partition block. Thereby, depending on the number of blocks in π we can reduce considerably the number of Boolean functions that satisfy the restriction and therefore reduce the size of the hypothesis set. Furthermore, in the following definition we show how each partition π can induce a hypothesis set of Boolean functions that we will denote as $\mathcal{H}_\pi$.

Definition 3.1.1. Let π be a partition of $\{0,1\}^n$. We define the hypothesis set (model) $\mathcal{H}_\pi$ induced by a partition π as

$$\mathcal{H}_\pi = \{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_\pi \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y})\}.$$

The next example shows a partition and its respective induced model.

Example 3.1.1. Let π be a partition of $\{0,1\}^2$ defined as $\pi = \{\{00\},\{01\},\{10,11\}\}$. Figure 3.3 illustrates π and its respective induced model $\mathcal{H}_\pi$.



x	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
00	0	0	0	0	1	1	1	1
01	0	0	1	1	0	0	1	1
10	0	1	0	1	0	1	0	1
11	0	1	0	1	0	1	0	1

Figure 3.3: *Partition* $\pi = \{\{00\},\{01\},\{10,11\}\}$ *and its respective induced hypothesis set* $\mathcal{H}_\pi$.

In the next propositions we present some properties of the models induced by partitions, the following proposition shows the size of any model induced by a partition.

Proposition 3.1.1. *Let* π *be a partition of* $\{0,1\}^n$*, then* $|\mathcal{H}_\pi| = 2^{|\pi|}$.

Proof. Because every function $h \in \mathcal{H}_\pi$ takes values either 0 or 1 as output for each block in π, so we obtain $2^{|\pi|}$ functions in $\mathcal{H}_\pi$. □

The next proposition is a consequence of Proposition 3.1.1 and the VC dimension definition.

Proposition 3.1.2. *Let* π *be a partition of* $\{0,1\}^n$*, then the VC dimension of* $\mathcal{H}_\pi$ *is* $|\pi|$.

Proof. By Proposition 3.1.1 we know that $|\mathcal{H}_\pi| = 2^{|\pi|}$, then using Proposition 2.3.1 we have that $d_{VC}(\mathcal{H}_\pi) \leq \lg|\mathcal{H}_\pi| = |\pi|$. On the other hand, if we have $N = |\pi|$ different elements of $\{0,1\}^n$ such that each element belongs to exactly one different partition of π, we could attribute the values either 0 or 1 to each element, then we have $m_{\mathcal{H}}(N) = 2^N$. Thus, using the VC dimension definition we have that $d_{VC}(\mathcal{H}_\pi) \geq |\pi|$. Hence $d_{VC}(\mathcal{H}_\pi) = |\pi|$. □

Notice that Proposition 3.1.2 is very important for understanding the complexity of the selected model, and also for deduce its sample complexity. In practice, we can use the sample size in the learning problem for defining a model with VC dimension according to our data. For instance, we can use the popular rule of thumb described in Section 2.3.3 to select a model with VC dimension of approximately $\frac{N}{10}$ to obtain decent generalization.

Moreover, we can see that there exists an order relation between models induced by partitions of $\{0,1\}^n$ as shown in the next proposition.

Proposition 3.1.3. *Let* $(\Pi_{\{0,1\}^n}, \leq)$ *be the partition lattice of* $\{0,1\}^n$*. We have that*

$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \text{if } \pi_1 \leq \pi_2 \Rightarrow \mathcal{H}_{\pi_2} \subseteq \mathcal{H}_{\pi_1}$$

Proof.

$$\mathcal{H}_{\pi_1} = \{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_{\pi_1} \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y})\}$$
(definition of $\mathcal{H}_{\pi_1}$)

$$\mathcal{H}_{\pi_1} = \{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_{\pi_1} \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y}), \mathbf{x} \equiv_{\pi_2} \mathbf{y}\} \cup$$
$$\{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_{\pi_1} \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y}), \mathbf{x} \not\equiv_{\pi_2} \mathbf{y}\}$$
(union of sets property)

$$\mathcal{H}_{\pi_1} = \{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_{\pi_2} \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y})\} \cup$$
$$\{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_{\pi_2} \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y}), \mathbf{x} \not\equiv_{\pi_2} \mathbf{y}\}$$
(because $\pi_1 \leq \pi_2$)

$$\mathcal{H}_{\pi_1} = \mathcal{H}_{\pi_2} \cup$$
$$\{h : \{0,1\}^n \to \{0,1\} \mid \forall \mathbf{x}, \mathbf{y} \in \{0,1\}^n, \mathbf{x} \equiv_{\pi_2} \mathbf{y}, h(\mathbf{x}) = h(\mathbf{y}), \mathbf{x} \not\equiv_{\pi_2} \mathbf{y}\}$$
(definition of $\mathcal{H}_{\pi_2}$)

Then $\mathcal{H}_{\pi_2} \subseteq \mathcal{H}_{\pi_1}$. □

Hence, using Proposition 2.3.4 we obtain the following result: let $\mathbb{H}$ be the set of all models induced by all partitions of $\{0,1\}^n$, then $(\mathbb{H}, \leq)$ is a lattice, which we will call *model lattice*. Furthermore, using the previous proposition we notice that the model lattice is anti-isomorphic to the partition lattice (the order between elements in the model lattice is maintained as in the partition lattice, but with opposite partial order). Similarly, as an immediate consequence of Proposition 2.3.4 we present the following corollary.

Corollary 3.1.4.

$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \textit{if } \pi_1 \leq \pi_2 \Rightarrow d_{VC}(\mathcal{H}_{\pi_2}) \leq d_{VC}(\mathcal{H}_{\pi_1})$$

Proof.

$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \text{if } \pi_1 \leq \pi_2 \Rightarrow \mathcal{H}_{\pi_2} \subseteq \mathcal{H}_{\pi_1} \quad \text{(for Proposition 3.1.3)}$$
$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \text{if } \pi_1 \leq \pi_2 \Rightarrow d_{VC}(\mathcal{H}_{\pi_2}) \leq d_{VC}(\mathcal{H}_{\pi_1}) \quad \text{(for Proposition 2.3.4)}$$

□

Notice that using Proposition 3.1.3 and Corollary 3.1.4 we have that: for any set of partitions such that $\pi_1 \leq \cdots \leq \pi_{n-1} \leq \pi_n$ we know that $\mathcal{H}_{\pi_n} \subseteq \mathcal{H}_{\pi_{n-1}} \subseteq \cdots \subseteq \mathcal{H}_{\pi_1}$ and also $d_{VC}(\mathcal{H}_{\pi_n}) \leq d_{VC}(\mathcal{H}_{\pi_{n-1}}) \leq \cdots \leq d_{VC}(\mathcal{H}_{\pi_1})$. As seen in Section 2.3.3, there exists a direct relation between nested hypothesis sets, the VC dimension and sample complexity as illustrated in Figure 2.11. Therefore, we have that out-of-sample errors of hypotheses taken from $\mathcal{H}_{\pi_n}, \mathcal{H}_{\pi_{n-1}}, \ldots, \mathcal{H}_{\pi_1}$, i.e. $E_{out}(h_{\pi_n})$, $E_{out}(h_{\pi_{n-1}})$, …, $E_{out}(h_{\pi_1})$ present the U-curve phenomenon as illustrated in Figure 3.4. Accordingly, the objective of a model selector of models induced by partitions should be finding the partition π^* or equivalently finding the model $\mathcal{H}_{\pi^*}$ such that $E_{out}(h_{\pi^*})$ is minimum as possible.

In summary, models induced by partitions have interesting properties that can be used for model selection methods to find a hypothesis set according to our data. However, there are learning problems where we want to find Boolean functions with more restrictions than models induced by partitions provide. Thus, in the next section is presented how we can combine models induced by partitions with other hypothesis restrictions.

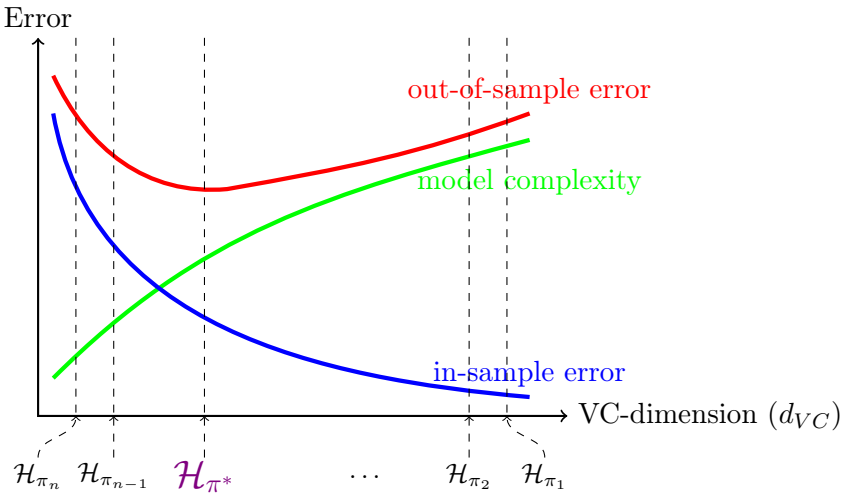
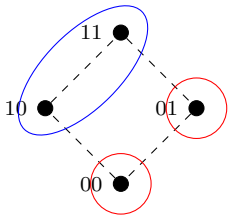


Figure 3.4: *Error curves of nested models induced by nested partitions. In purple is highlighted the hypothesis set with minimum out-of-sample error.*

3.1.1 Restrictions on models induced by partitions

In the process of learning Boolean functions it is usual to restrict hypothesis spaces to limit the search space and consequently have a lower VC dimension. For example, learning Boolean functions that can only be represented by monomials is a restriction in the hypothesis space as shown in Example 2.3.4. The next example shows a Boolean hypothesis set restricted to a partition π and also restricted by a monomial representation.

Example 3.1.2. We presented in Example 2.3.4 that the hypothesis set $\mathcal{H}_\Theta$ where $\Theta = \{\emptyset, x_1, x_2, \overline{x_1}, \overline{x_2}, x_1x_2, x_1\overline{x_2}, \overline{x_1}x_2, \overline{x_1}\,\overline{x_2}\}$ is the hypothesis set of Boolean functions of two variables represented by monomials. In this example, we want the mentioned hypothesis set also restricted to the partition $\pi = \{\{00\}, \{01\}, \{10, 11\}\}$, this is $\mathcal{H} = \mathcal{H}_\Theta \cap \mathcal{H}_\pi$. Figure 3.5 shows π and the resulting restricted hypothesis set.



x	$h_\emptyset$	h_{x_1}	$h_{\overline{x_1}}$	$h_{\overline{x_1}x_2}$	$h_{\overline{x_1}\,\overline{x_2}}$
00	0	0	1	0	1
01	0	0	1	1	0
10	0	1	0	0	0
11	0	1	0	0	0

Figure 3.5: *Partition $\pi = \{\{00\}, \{01\}, \{10, 11\}\}$ and the hypothesis set $\mathcal{H} = \mathcal{H}_\Theta \cap \mathcal{H}_\pi$*

Moreover, as we show in Proposition 3.1.3, there exists an order relation between models induced by partitions. In the next corollary, we show that this order relation is maintained when we impose more restrictions in the hypothesis set.

Corollary 3.1.5. *Let $(\Pi_{\{0,1\}^n}, \leq)$ be the partition lattice of $\{0,1\}^n$. If we apply a restriction $\mathcal{H}^*$ on models $\mathcal{H}_{\pi_1}, \mathcal{H}_{\pi_2}$ such that $\pi_1 \leq \pi_2$ we have*

$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \text{if } \pi_1 \leq \pi_2 \Rightarrow \mathcal{H}_{\pi_2} \cap \mathcal{H}^* \subseteq \mathcal{H}_{\pi_1} \cap \mathcal{H}^*$$

Proof.

$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \text{if } \pi_1 \leq \pi_2 \Rightarrow \mathcal{H}_{\pi_2} \subseteq \mathcal{H}_{\pi_1} \quad \text{(using Proposition 3.1.3)}$$
$$\forall \pi_1, \pi_2 \in \Pi_{\{0,1\}^n}, \text{if } \pi_1 \leq \pi_2 \Rightarrow \mathcal{H}_{\pi_2} \cap \mathcal{H}^* \subseteq \mathcal{H}_{\pi_1} \cap \mathcal{H}^* \quad \text{(property of intersection in sets)}$$

□

Therefore, if all models in the model lattice $(\mathbb{H}, \leq)$ are restricted by other hypothesis sets, then the restricted models continue maintaining their partial order in the lattice.

In summary, in this section we presented the model lattice which is an algebraic structure composed by models generated by partitions of the domain, we show that these models have interesting properties. In the next section, we will use these concepts for designing new model selectors.

3.2 Generic Model Selector for Boolean Functions

In this section, we present our first model selector of Boolean functions using partition concepts introduced in the previous section, we will call this model as *Model selector using partitions*. In contrast to the typical model selection process seen in Section 2.3.8, this new model selector generates its own models using partition restrictions in the domain space. Thereby, the generated models satisfy properties explained in the previous section such as having a defined size and a fixed VC dimension, then the model selector picks the best generated model according to the training data provided.

Model selector using partitions can be explained as a workflow illustrated in Figure 3.6 and we can see more details of this workflow in Algorithm 5.

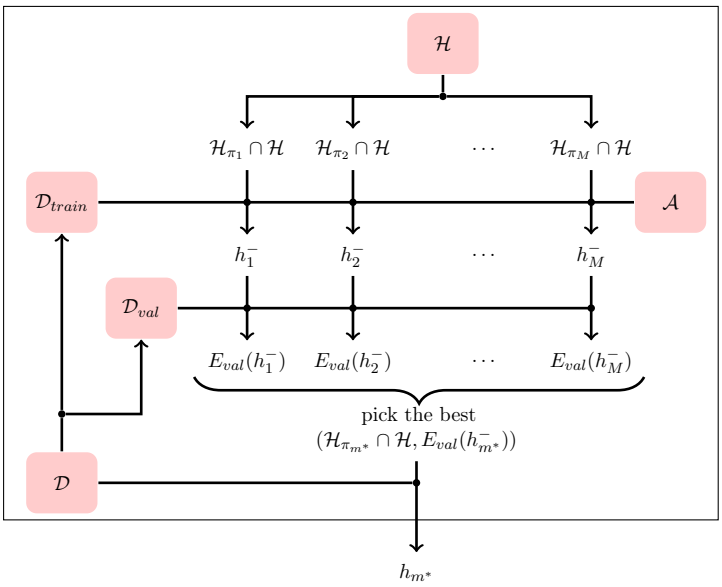


Figure 3.6: *Model selector using partitions.*

Some details of the Algorithm 5 are explained below:

- The procedure Generate-All-Partitions generates all possible partitions of $\{0,1\}^n$.

Algorithm 5: Model selector using partitions.

Input: Learning Algorithm $\mathcal{A}$, model $\mathcal{H}$ and a dataset $\mathcal{D}$.
Output: Output hypothesis h_{m^*} and the best model $\mathcal{H}_{\pi_{m^*}}$.

```
/* Generate all partitions from {0,1}^n */
Π_{{0,1}^n} ← Generate-All-Partitions(D);
/* Divide the dataset in two (training and validation set) */
(D_train, D_val) ← D;
/* Initiate variables */
π_{m*} ← ∅ ;
best_E ← ∞ ;
forall π ∈ Π_{{0,1}^n} do
    E_val(h^-) ← Get-Cost(A, H_π ∩ H, D_train, D_val);
    if E_val(h^-) < best_E then
        π_{m*} ← π;
        best_E ← E_val(h^-);
    end
end
/* Calculate best hypothesis using all dataset D */
h_{m*} ← Find-Hypothesis(A, H_{π_{m*}} ∩ H, D);
return (h_{m*}, H_{π_{m*}} ∩ H);
```

Algorithm 6: Get-Cost$(\mathcal{A}, \mathcal{H}, \mathcal{D}_{train}, \mathcal{D}_{val})$. Obtain the validation error of the best hypothesis on the model $\mathcal{H}$ using the learning algorithm $\mathcal{A}$.

Input: Learning Algorithm $\mathcal{A}$, model $\mathcal{H}$, training dataset $\mathcal{D}_{train}$ and validation dataset $\mathcal{D}_{val}$
Output: Cost $E_{val}(h^-)$ of best hypothesis found in the model $\mathcal{H}$ using the learning algorithm $\mathcal{A}$.

```
/* Get the best hypothesis on the model H using learning algorithm A */
h^- ← Find-Hypothesis(A, H, D_train);
/* Calculate the validation error */
E_val(h^-) ← (1/K) Σ_{x_i ∈ D_val} L(y_i, h^-(x_i));
return E_val(h^-);
```

- The procedure Find-Hypothesis$(\mathcal{A}, \mathcal{H}, \mathcal{D})$ executes the learning algorithm $\mathcal{A}$ using as input the hypothesis set $\mathcal{H}$ and the dataset $\mathcal{D}$. We can use any learning algorithm in this procedure, for example we could use known algorithms such as the perceptron learning algorithm or the SVM algorithm to find the best hypothesis in $\mathcal{H}$.
- The procedure Get-Cost is defined in Algorithm 6.
- Variable π_{m^*} stores the best partition found so far.
- Variable $best_E$ stores the validation error obtained using partition π_{m^*}.

In Example 3.2.1, we present an application to show how the model selector using partitions can be used.

Example 3.2.1. Suppose that we want to find a Boolean function with n variables that is represented by monomials given a training dataset. So we can use the Model selector using partitions for finding a suitable model and a Boolean function according to the data provided. Thus, the Model selector using partitions will use the following input arguments:

- The input hypothesis set will be the set of functions that can be represented by monomials $\mathcal{H}_\Theta$ as mentioned in Example 2.3.4.
- The learning algorithm might be any algorithm, for example we can use the monomials learning algorithm presented in Section 2.3.4.
- Suppose that our dataset $\mathcal{D}$ contains a set of N pairs, that is $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \{0,1\}^n, y_i \in \{0,1\}, 0 \leq i \leq N$.

Thereby, it is expected that the Model selector using partitions will output a suitable Boolean function that can be represented by monomials.

Unfortunately, the model selector using partitions is more a guide for creating a model selector rather than a practical model selector because of the computational complexity of generating all possible partitions of $\{0,1\}^n$ and the burden of executing the learning algorithm $\mathcal{A}$ so many times. Thus, this model selector is not computationally feasible if the set of possible models induced by partitions is not restricted. So, one solution to this problem is let the user specify the partition set as it has been doing implicitly in some common machine learning methods in the literature, as we will see later in the next sections.

Therefore, we propose a model selector that improves the Model selector using partitions letting the user specify the set of partitions that generate the models, it is called *Generic model selector* and is illustrated in Figure 3.7, more details are presented in Algorithm 7.

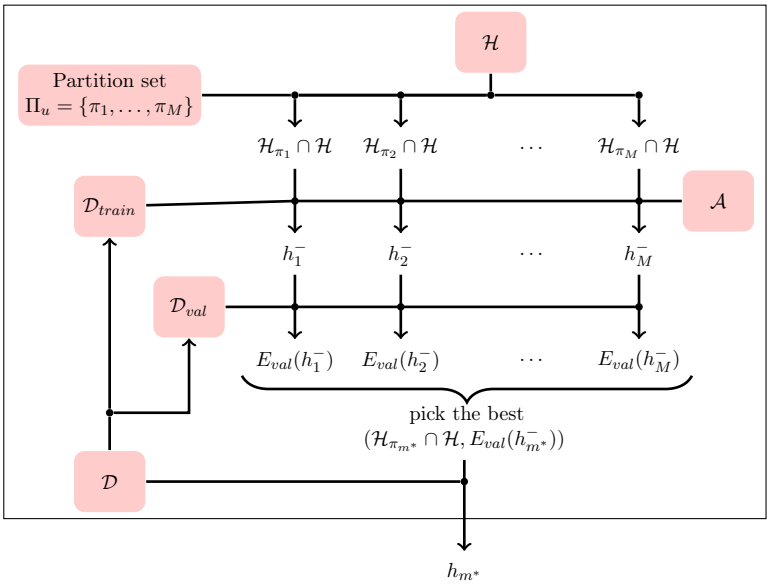


Figure 3.7: *Generic Model Selector.*

We call this method as Generic model selector because it generalizes the set of learning methods that use models induced by partition sets. On different learning

Algorithm 7: Generic Model Selection.

Input: Learning Algorithm $\mathcal{A}$, hypothesis set $\mathcal{H}$, partition set Π_u and a dataset $\mathcal{D}$.

Output: Output hypothesis h_{m^*} and the best model $\mathcal{H}_{\pi_{m^*}}$.

```
1  /* Divide the dataset in two (training and validation set) */
2  (D_train, D_val) ← D;
3  /* Initiate variables */
4  π_{m*} ← ∅ ;
5  best_E ← ∞ ;
6  forall π ∈ Π_u do
7      E_val(h⁻) ← Get-Cost(A, H_π ∩ H, D_train, D_val);
8      if E_val(h⁻) < best_E then
9          π_{m*} ← π;
10         best_E ← E_val(h⁻);
11     end
12 end
13 /* Calculate best hypothesis using all dataset D */
14 h_{m*} ← Find-Hypothesis(A, H_{π_{m*}} ∩ H, D);
15 return (h_{m*}, H_{π_{m*}} ∩ H);
```

methods, the specified partition set is not given by the user, but the partition set can be generated by some algorithm in order to restrict the partition search space. In the following subsections, we will show how partition sets are generated implicitly by different learning methods.

3.2.1 Feature selection problem and the Generic model selector

In Section 2.3.6, the feature selection problem and its representation were introduced; now we present how the feature selection problem can restrict the partition search space generating a partition set of the domain space and use this partition set as input for the Generic model selector.

Considering a learning problem, let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of features where $x_1, x_2, \ldots, x_n$ are Boolean variables, and let $\mathcal{X}_0$ be the domain of the elements, so $\mathcal{X}_0 = \{0,1\}^X$. Now, let us assume that a feature selection algorithm chooses a subset of features $Z \subseteq X$ as output. Then, the unique Boolean variables considered in the problem will be the ones contained in Z, this fact implies that groups of elements in $\mathcal{X}_0$ will be considered the same in the problem as we show in the next example.

Example 3.2.2. Suppose that the set of Boolean variables considered in the learning problem is $X = \{x_1, x_2\}$, then the domain of elements is the set $\mathcal{X}_0 = \{0,1\}^X = \{00, 01, 10, 11\}$. For example, if we perform feature selection and obtain as output a set of Boolean variables $Z = \{x_1\}$, then elements 00 and 01 will be considered the same in the learning problem because we only consider the Boolean variables in Z (we only consider x_1).

Thereby, we will show how the feature selection problem generates a partition in the domain space in the same way that partitions are generated by multiresolution down-samplings (see Section 2.1.4 for more details). Let $\mathcal{X}_1 = \{0,1\}^Z$ be the domain of elements when we only consider variables in Z, then the sets X and Z ($Z \subseteq X$)

induce the multiresolution down-sampling $\rho : \mathcal{X}_0 \to \mathcal{X}_1$ such that $\rho(\mathbf{x}) = \mathbf{z}$, where $\mathbf{x} = \sigma(X)$ and $\mathbf{z} = \sigma(Z)$. For instance, in Example 3.2.2 we have that X and Z induces the multiresolution down-sampling $\rho(x_1, x_2) = (x_1)$.

Besides, ρ induces a partition π of the space $\mathcal{X}_0$ given by

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}_0, \mathbf{x}_1 \equiv_\pi \mathbf{x}_2 \Leftrightarrow \rho(\mathbf{x}_1) = \rho(\mathbf{x}_2).$$

For instance, in Example 3.2.2 ρ induces the partition $\{\{00, 01\}, \{10, 11\}\}$.

Therefore, we observe that every subset $Z \subseteq X$ can generate a different partition of the domain space $\{0, 1\}^n$. Thereby, gathering all partitions generated by subsets of X we can obtain a partition set of $\{0, 1\}^n$. In fact, this partition set has 2^n elements and we will denote it by $\Pi(\mathcal{P}(X))$. Notice that this partition set is a subset of all partitions of $\{0, 1\}^n$, i.e. $\Pi(\mathcal{P}(X)) \subseteq \Pi_{\{0,1\}^n}$.

In the following example, we show how the feature selection problem generates a partition set of the domain space $\{0, 1\}^2$.

Example 3.2.3. Let $\mathcal{X} = \{0, 1\}^2$ be the domain space, the feature selection problem can be represented by the search in the lattice $(\mathcal{P}(X), \subseteq)$ where $X = \{x_1, x_2\}$ is a set Boolean variables. Thus, considering each subset of X as a possible solution of the problem, we have one multiresolution down-sampling for each subset of X (in total four down-samplings), they are $\rho_1, \rho_2, \rho_3, \rho_4$ and they are defined as $\rho_1(x_1, x_2) = ()$, $\rho_2(x_1, x_2) = (x_1)$, $\rho_3(x_1, x_2) = (x_2)$, $\rho_4(x_1, x_2) = (x_1, x_2)$.

Furthermore, we observe that ρ_1 induces the partition $\{\{00, 01, 10, 11\}\}$, ρ_2 induces the partition $\{\{00, 01\}, \{10, 11\}\}$, ρ_3 induces the partition $\{\{00, 10\}, \{01, 11\}\}$ and ρ_4 induces the partition $\{\{00\}, \{10\}, \{01\}, \{11\}\}$. Therefore, these four induced partitions form the partition set $\Pi(\mathcal{P}(X))$ which is presented as a sub-lattice of $\{0, 1\}^2$ in Figure 3.8.

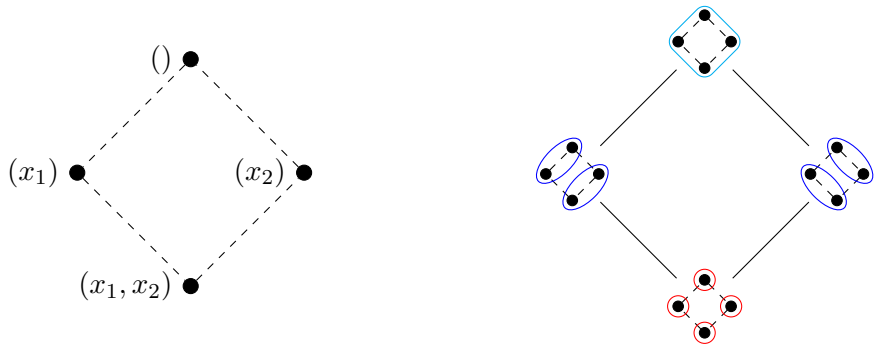


Figure 3.8: *Feature selection problem for the feature set* $X = \{x_1, x_2\}$ *and the partition set* $\Pi(\mathcal{P}(X))$ *presented as a sub-lattice of* $(\Pi_{\{0,1\}^n}, \leq)$.

In general, the feature selection problem generates a partition set $\Pi(\mathcal{P}(X))$ using multiresolution down-samplings induced by features subsets of X. Thus, $\Pi(\mathcal{P}(X))$ can be used as input for the Generic model selector (see Algorithm 7), this way we are creating a model selector that performs feature selection and reducing the model search space from $B_{|\mathcal{X}|} = B_{2^n}$ to 2^n models (B_n is defined in Section 2.1.1). Furthermore, we can take advantage of proposed feature selection methods in the literature [Chandrashekar and Sahin, 2014] for testing a small number of feature subsets reducing the search complexity. Therefore, using only the models generated by feature subsets in the Generic model selector we can reduce drastically the number of tested hypothesis in the model.

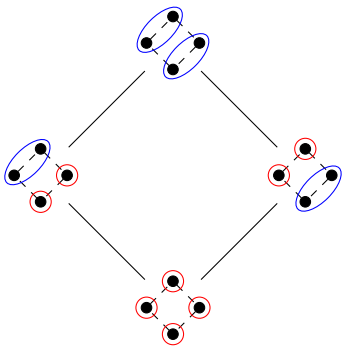
3.2.2 Multiresolution representation and the Generic model selector

In this section, we present how multiresolution down-samplings (see Section 2.1.4) can also generate partition sets restricting the partition search space, additionally they can be used as input for the Generic model selector. As seen in the previous section, the feature selection problem induces a partition set using a set of multiresolution down-samplings generated by feature subsets. In the following examples, we observe how defined multiresolution down-samplings also generate partition subsets of the domain.

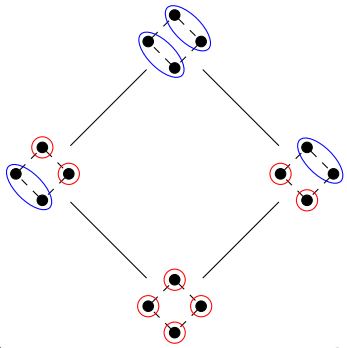
Example 3.2.4. Let $W_0 = \{w_1, w_2\}$ and $W_1 = \{w_1\}$ be sets of Boolean variables, and let $\mathcal{X}_0 = \{0,1\}^{W_0}$ and $\mathcal{X}_1 = \{0,1\}^{W_1}$ be domain spaces, i.e. $\mathcal{X}_0 = \{00, 01, 10, 11\}$ and $\mathcal{X}_1 = \{0, 1\}$. As seen in Section 2.1.4, from these sets we can induce a multiresolution down-sampling $\rho : \mathcal{X}_0 \to \mathcal{X}_1$ defined as $\rho(w_1, w_2) = (w_1)$. Thus, depending on the element sets $\mathcal{X}_{00}$, $\mathcal{X}_{01}, \mathcal{X}_{10}$ and $\mathcal{X}_{11}$, we can generate different partitions of the domain space, all these partitions can be seen as a partition poset illustrated in Figure 3.9(a).

The following example is very similar to the previous one, we want to show the difference between the generated partition posets when we modify the set W_1.

Example 3.2.5. Let $W_0 = \{w_1, w_2\}$ and $W_1 = \{w_2\}$ be sets of Boolean variables, and let $\mathcal{X}_0 = \{0,1\}^{W_0}$ and $\mathcal{X}_1 = \{0,1\}^{W_1}$ be domain spaces. Then, we induce a multiresolution down-sampling $\rho : \mathcal{X}_0 \to \mathcal{X}_1$ defined as $\rho(w_1, w_2) = (w_2)$. Similarly, depending on the element sets $\mathcal{X}_{00}$, $\mathcal{X}_{01}, \mathcal{X}_{10}$ and $\mathcal{X}_{11}$ we generate the partition poset illustrated in Figure 3.9(b).



(a) Partition poset generated in Example 3.2.4.



(b) Partition poset generated in Example 3.2.5.

Figure 3.9: *Partition sets generated by multiresolution down-samplings.*

Thus, as shown in the examples, we observe that any multiresolution down-sampling generates a partition poset that can be used as input for the Generic model selector.

3.2.3 Decision trees and the Generic model selector

As seen in previous sections with other learning methods, decision trees also can restrict the partition search space generating a partition set to be used as input for the Generic model selector.

The Boolean decision tree definition presented in Section 2.1.5 can be used to define a big number of different decision trees for representing Boolean functions because of its generality. In this section, we restrict this definition to only cover a decision tree family called Boolean division criteria decision trees and they are defined as follows:

Definition 3.2.1. The *Boolean division criteria decision trees* (BDCDT) is the family of decision trees representing Boolean functions with the property that each node division criterion in the tree is given by the values of a non empty set of Boolean variables.

In Example 3.2.6, we show two decision trees that generate the same partition and both belong to the BDCDT family.

Example 3.2.6. In Figure 3.10 two BDCDT are illustrated and they induce the same partition in the domain space. The division criterion of the left tree root are the values of the Boolean variable set $\{x_1, x_2\}$. Likewise, in the right tree the decision criterion of node XX are the values of the Boolean variable set $\{x_1\}$ and for nodes 1X and 0X are the values of the Boolean variable set $\{x_2\}$.

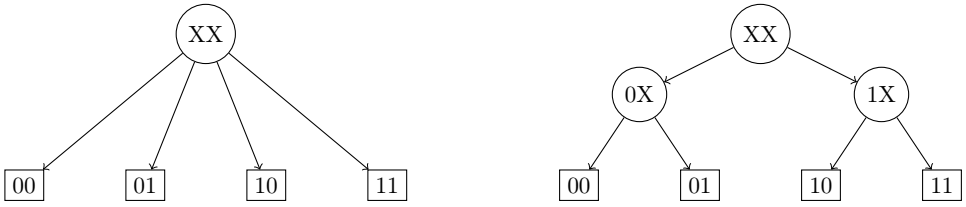


Figure 3.10: *Decision trees representing the same partition of the set* $\{0,1\}^2$.

As shown in Section 2.1.5, decision trees generate partitions in the domain space. Thus, in Example 3.2.7 we illustrate a BDCDT and the domain subsets that are represented by each node in the tree.

Example 3.2.7. In Figure 3.11 is illustrated a BDCDT representing a Boolean function of 4 variables. Besides, we illustrate the domain space $\{0,1\}^4$ as a Hasse diagram and we use colored ellipses for representing subsets of the domain represented by each node in the BDCDT. Notice that the node output labels are not considered in the figure because of their independence with the domain subsets.

Since any BDCDT induces a partition of the domain space, the BDCDT family generates a subset of the set of all partitions of the domain. For example, Figure 3.12 illustrates the partition subset generated by the BDCDT family of two Boolean variables organized as a sub-lattice of the partition lattice of $\{0,1\}^2$. This way, we can use the partition subset generated by BDCDT families to use as input for our Generic model selector.

As conclusion of this section, it was presented a Generic model selector that uses models generated by partitions of the domain space. Besides, it was explored that there exist different machine learning methods that create partition sets of the domain space, additionally, all of them can be used as input for our Generic model selector. The variety and size of the partition sets make some of them more appropriate than others depending of the learning problem. In the next section, we take advantage of the partition poset structure to improve the efficiency of the Generic model selector.

3.3 Generic model selector and lattice search algorithms

Until now, we have seen that the Generic model selector presented in Section 2.3.8 improves its efficiency when the model receives a partition set given by the user. Although the number of partitions generated by the different methods are considerably smaller

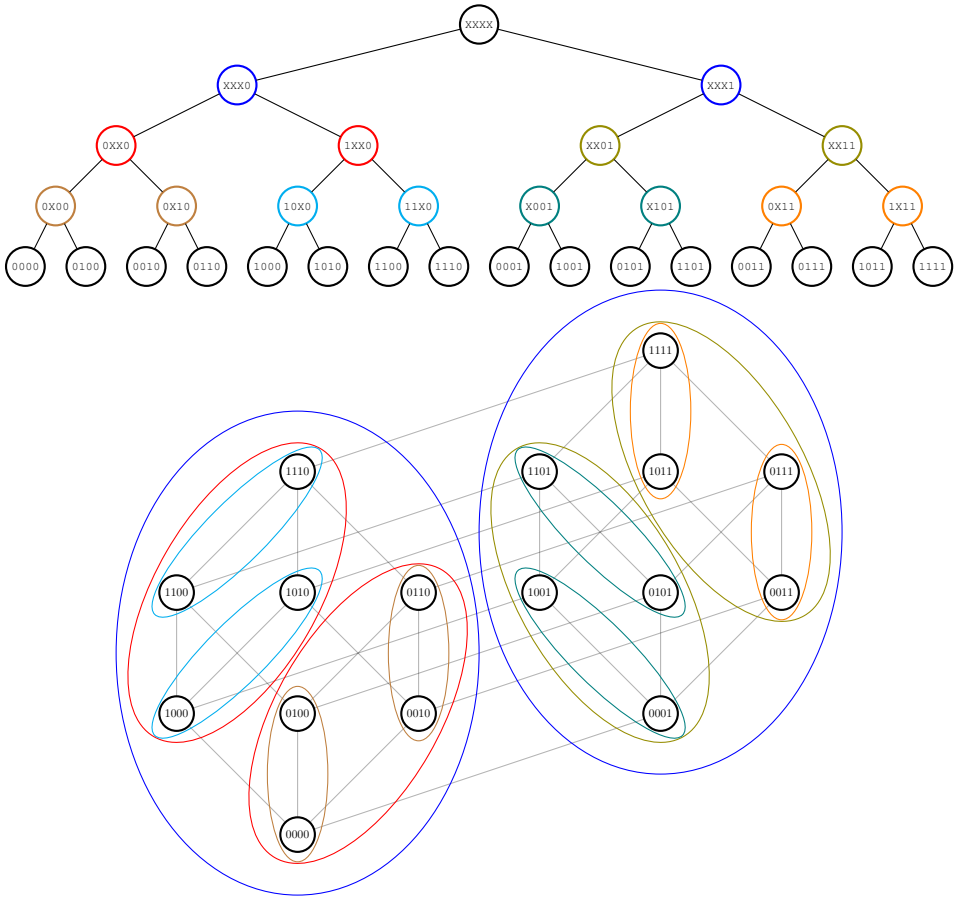


Figure 3.11: *Decision tree and its corresponding domain space partition.*

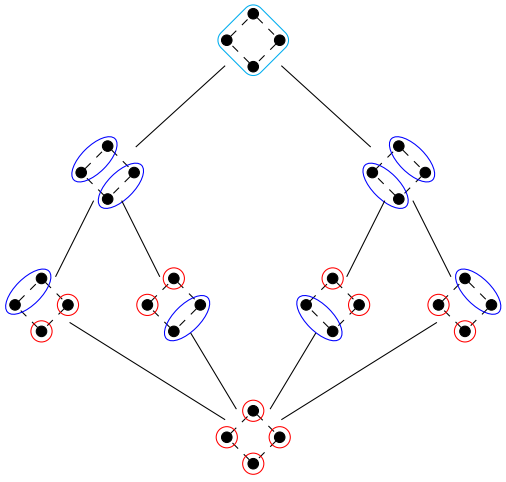


Figure 3.12: *Sub-lattice of the partition lattice of* $\{0,1\}^2$ *induced by the BDCDT family of two Boolean variables*

compared to the whole partition set of the domain, we still have to run the learning algorithm $\mathcal{A}$ on all the models generated by these partitions.

In this section, we present another extension of the Generic model selector to improve its performance. To achieve this, we will let users specify the search algorithm in posets they prefer in order to constrain the model search space, also this approach takes

advantage of a wide variety of search algorithms in the literature. Furthermore, search strategies can exploit the fact that out-of-sample errors taken from nested models present the U-curve phenomenon as shown earlier in this chapter. Some algorithms that deal exactly with this kind of problem can be seen in [Ris *et al.*, 2010], [Reis, 2013], [Atashpaz-Gargari *et al.*, 2013], [Atashpaz-Gargari *et al.*, 2018].

Therefore, we propose the *Generic model selector using search algorithms* and its workflow is illustrated in Figure 3.13. In this workflow, we select the Get-Cost function (see Algorithm 6) to estimate the out-of-sample error of the best function found in any model. Then, the poset search algorithm specified by the user adopts this cost function to evaluate the visited partitions and use them as criterion for the search process.

In Algorithm 8, we can see the two steps of this model selector, the first step calls algorithm Run-Search-Algorithm, which in turn calls the procedure Get-Cost whenever a partition π is visited, that is:

$$\text{Get-Cost}(\mathcal{A}, \mathcal{H}_\pi \cap \mathcal{H}, \mathcal{D}_{train}, \mathcal{D}_{val})$$

where $\mathcal{A}$, $\mathcal{H}$, $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ are parameters specified by the user. Finally, the procedure Find-Hypothesis is called for finding the best hypothesis in the model $\mathcal{H}_{\pi_{m^*}} \cap \mathcal{H}$ using all the available data. In the following examples, we present applications of the Generic model selector using search algorithms.

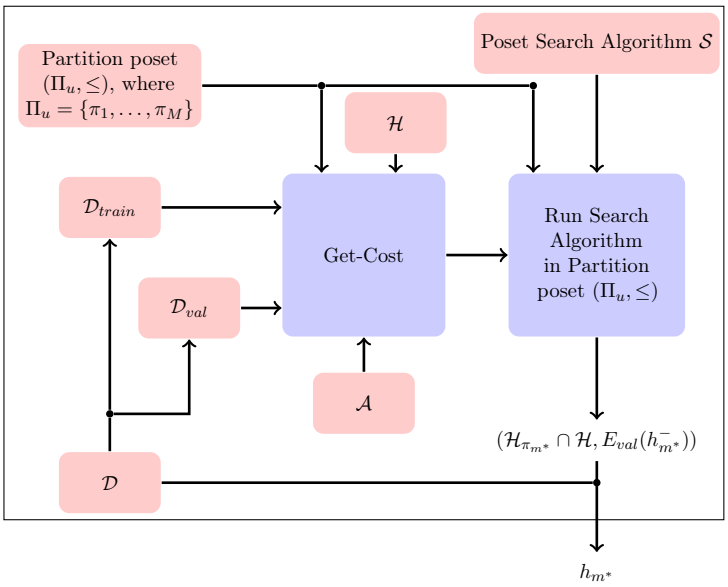


Figure 3.13: *Generic Model Selector using search algorithms.*

Example 3.3.1. In this example, we are going to use the Generic model selector using search algorithms to create a specific model selector using the following input arguments:

- The selected hypothesis set $\mathcal{H}$ for the model is the set of all the Boolean functions with two variables.
- The selected partition poset $(\Pi_u, \leq)$ is generated by the BDCDT family of two Boolean variables as presented in Figure 3.12.
- Suppose that our learning algorithm $\mathcal{A}$ could be any learning algorithm such that it receives as input an hypothesis set $\mathcal{H}_\pi$ and returns as output the Boolean

Algorithm 8: Generic Model Selector using search algorithms.

Input: Search Algorithm $\mathcal{S}$, Learning Algorithm $\mathcal{A}$, hypothesis set $\mathcal{H}$, partition poset $(\Pi_u, \leq)$ and a dataset $\mathcal{D}$.

Output: Output hypothesis h_{m^*} and the best model $\mathcal{H}_{\pi_{m^*}}$.

```
/* Obtain best partition set using the Search Algorithm S */
π_{m*} ← Run-Search-Algorithm(S, (Π_u, ≤), Get-Cost) ;
/* Find the best hypothesis using all dataset D */
h_{m*} ← Find-Hypothesis(A, H_{π_{m*}} ∩ H, D);
return (h_{m*}, H_{π_{m*}} ∩ H);
```

function f_π.

- Suppose that our dataset $\mathcal{D}$ contains a set of N pairs, that is $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \{0,1\}^2, y_i \in \{0,1\}, 0 \leq i \leq N$.
- The selected search algorithm in the poset $(\Pi_u, \leq)$ is a depth first search algorithm for posets with the U-curve property described in Algorithm 9.

Now, suppose that the out-of-sample errors of $\{f_\pi : \pi \in \Pi_u\}$, i.e. $\{E_{out}(f_\pi) : \pi \in \Pi_u\}$, satisfy the U-curve property. In Figure 3.14 is illustrated the sub-lattice induced by the BDCDT family of two Boolean variables, written in purple we have the $E_{out}(f_\pi)$ for each partition, in green we indicate the visited partitions for the DFS algorithm starting with the partition at top of the lattice.

Thus, the DFS algorithm will return the partition and the cost of the best partition found, in this example the best partition found is the partition $\pi_{m^*} = \{\{00, 01\}, \{10\}, \{11\}\}$. Finally, we use the second step of the model selector, that is, finding the best hypothesis using all dataset $\mathcal{D}$, this is $h_{m^*} = \text{Find-Hypothesis}(\mathcal{A}, \mathcal{H}_{\pi_{m^*}}, \mathcal{D})$.

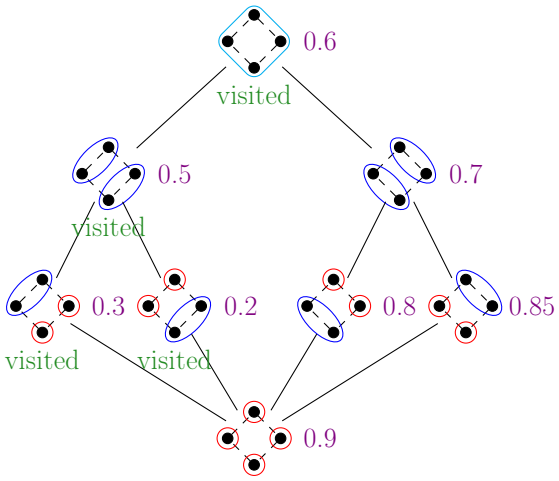


Figure 3.14: *Partition sub-lattice induced by the BDCDT family of two Boolean variables, $E_{out}(f_\pi), \pi \in \Pi_u$ in purple and the partitions labeled as visited by the algorithm are indicated.*

Example 3.3.2. In this example, we show the learning of a Boolean function with n Boolean variables represented by a BDCDT seen as an instance of the Generic model selector using search algorithms. In this way, the model selector is going to use the following input arguments:

Algorithm 9: DFS($(\Pi_u, \leq)$, π) Depth first search algorithm in poset with the U-curve property.

Input: Partition poset $(\Pi_u, \leq)$, cost function Get-Cost with its arguments
Output: Partition π_{m^*} with minimum cost and its cost.

```
/* Get cost for each partition */;
current_cost ← Get-Cost(A, H_π ∩ H, D_train, D_val);
Label π as visited;
best_cost ← current_cost;
π_{m*} ← π;
forall partitions π' adjacent to π in (Π_u, ≤) do
    if partition π' is not visited and
    Get-Cost(A, H_{π'}, D_train, D_val) < current_cost then
        /* Recursively call DFS */
        (π_{m'}, adjacent_cost) = DFS((Π_u, ≤), π');
        if adjacent_cost < best_cost then
            best_cost ← adjacent_cost;
            π_{m*} ← π_{m'};
        end
    end
end
return (π_{m*}, best_cost);
```

- The selected hypothesis set $\mathcal{H}$ for the model is the set of all the Boolean functions with n variables.
- The selected partition poset $(\Pi_u, \leq)$ is the poset generated by the BDCDT family of n Boolean variables.
- The learning algorithm $\mathcal{A}$ works as follows: $\mathcal{A}$ picks the function that minimize the in-sample error in the model induced by the partition.
- Suppose that our dataset $\mathcal{D}$ contains a set of N pairs, that is $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \{0,1\}^n, y_i \in \{0,1\}, 0 \leq i \leq N$.
- The selected search algorithm in the poset $(\Pi_u, \leq)$ will use the same idea of how we learn a BDCDT with automatic division criteria as presented in Algorithm 4. At first, notice that in the decision tree learning with automatic division criteria, every time a node does not meet the stopping criteria, the algorithm creates new nodes in the tree generating a new partition of the domain $\{0,1\}^n$. Interestingly, these generated partitions form a sequence of partitions that can be seen as a path in the partition lattice $(\Pi_u, \leq)$ starting at the top of the lattice. As an example, we illustrate in Figure 3.15 how the growing of a BDCDT can be seen as a path in the poset induced by the BDCDT family of two Boolean variables (see Figure 3.12).

Therefore, using these input arguments we created a model selector that uses a greedy search algorithm to pick an output model based on the learning of a BDCDT of n Boolean variables.

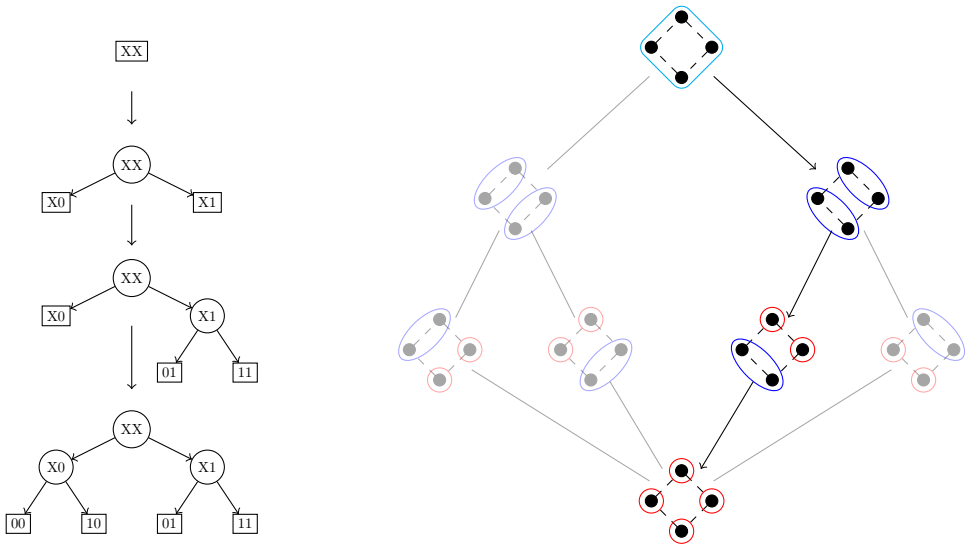


Figure 3.15: *The growing of a BDCDT and the path in the sub-lattice induced by the BDCDT family of two Boolean variables.*

3.4 Experiments

In this section, we present experiments of real applications of the Generic model selector. We show that the concepts presented in this chapter are applicable in real situations and they are not only theoretical results. For these experiments we do not use the Generic model selector using search algorithms because the size of the set of all partitions of the domain is small enough.

In the following experiments, our objective is to learn the best model for a Boolean function $h : \{0,1\}^2 \to \{0,1\}$ which is approximated to a target Boolean function $f : \{0,1\}^2 \to \{0,1\}$ determined by a given joint probability distribution $\Pr(X,Y)$ defined in Figure 3.16(a), so the target Boolean function is the one that minimizes the out-of-sample error which is illustrated in Figure 3.16(b).

Thereby, we will use the Generic model selector shown in Section 3.2 to find the best model and function using the following input components:

- An input dataset $\mathcal{D}$ consisting of N pairs, that is $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \{0,1\}^2, y_i \in \{0,1\}, 0 \leq i \leq N$.
- The selected hypothesis set $\mathcal{H}$ is the set of all Boolean functions of two Boolean variables.
- The partition set Π_u is the set of all partitions of the domain, i.e. $\Pi_u = \Pi_{\{0,1\}^2}$.
- The selected learning algorithm $\mathcal{A}$ for each model $\mathcal{H}_\pi$ is the one that takes the hypothesis $h \in \mathcal{H}_\pi$ that minimizes the in-sample error, this is:

  For all elements $\mathbf{x}$ in the same block $B \in \pi$, we have

  $$h(\mathbf{x}) = \begin{cases} 0, & \text{if } \sum_{\mathbf{x}\in B} \Pr_{\mathcal{D}}(Y=0|X=\mathbf{x}) \geq \sum_{\mathbf{x}\in B} \Pr_{\mathcal{D}}(Y=1|X=\mathbf{x}); \\ 1, & \text{otherwise.} \end{cases}$$

  where $\Pr_{\mathcal{D}}(Y=0|X=\mathbf{x})$ and $\Pr_{\mathcal{D}}(Y=1|X=\mathbf{x})$ are estimations of $\Pr(Y=0|X=\mathbf{x})$ and $\Pr(Y=1|X=\mathbf{x})$ respectively using the dataset $\mathcal{D}$.

	Y=0	Y=1
X=00	0.237	0.003
X=01	0.191	0.248
X=10	0.074	0.032
X=11	0.043	0.172

(a) Joint probability distribution.

$\mathbf{x}$	$f(\mathbf{x})$
00	0
01	1
10	0
11	1

(b) Target function.

Figure 3.16: *Joint probability distribution of random variables X and Y and the target function of the problem*

Then, the Generic model selector estimates the out-of-sample error of the best Boolean function for each model $\mathcal{H}_\pi$. Once we suppose that $\Pr(X, Y)$ is unknown and considering that the size of $\mathcal{D}$ is small, we will estimate the out-of-sample errors using the Leave-one-out cross-validation using $\mathcal{D}$ instead of calculating the validation error (for more details, see [Abu-Mostafa *et al.*, 2012]). Finally, the Generic model selector picks the model and Boolean function that minimizes the estimated out-of-sample error.

Experiment 1

As first experiment, we simulate the Generic model selector execution using a sample $\mathcal{D}$ of 10 elements obtained at random using $\Pr(X, Y)$, it is:

$$\mathcal{D} = \{(01, 1); (00, 0); (00, 0); (01, 1); (01, 0); (11, 0); (10, 1); (11, 1); (01, 1); (00, 0)\}.$$

Thereby, the Generic model selector calculates the in-sample error for every function in all models using the estimated joint probability distribution $\hat{\Pr}(X, Y)$ obtained from $\mathcal{D}$ (see Figure 3.17(a)). For each model, the model selector picks the function with minimum in-sample error, these values are illustrated in Figure 3.18(a), then it estimates the out-of-sample error (Leave-one-out cross-validation) of these functions which are illustrated in Figure 3.18(b).

For example, the Generic model selector could pick the model induced by the partition $\pi = \{\{00, 11\}, \{01, 10\}\}$ because there exists a function $f \in \mathcal{H}_\pi$ (defined in Figure 3.17(b)) such that it minimizes the in-sample error in the model $\mathcal{H}_\pi$, and also it has the minimum estimated out-of-sample error between all models.

For comparison, in Figure 3.19(a) we illustrate the true out-of-sample errors of the functions with minimum in-sample error for each model. Consequently, we observe that the estimated values are not close to the true values because a dataset with 10 elements is small for a good estimation. However, the ideal model that the Generic model selector should pick is the model induced by the partition $\pi = \{\{00, 10\}, \{01, 11\}\}$ because it is the one with minimum true out-of-sample error.

Observing the error values in Figure 3.18(a) and Figure 3.18(b), the in-sample error decreases when the VC dimension increases, in contrast, most of the chains in the model lattice when we estimate the out-of-sample error have the U-curve property. For instance, in Figure 3.19(b) we present the error curves in a chain of the model lattice.

Experiment 2

Similarly to Experiment 1, in this experiment we simulate the Generic model selector execution using a dataset $\mathcal{D}$ of size 20. By contrast, we estimate the average value of the different errors for each model, that is, for each model we calculate the in-sample,

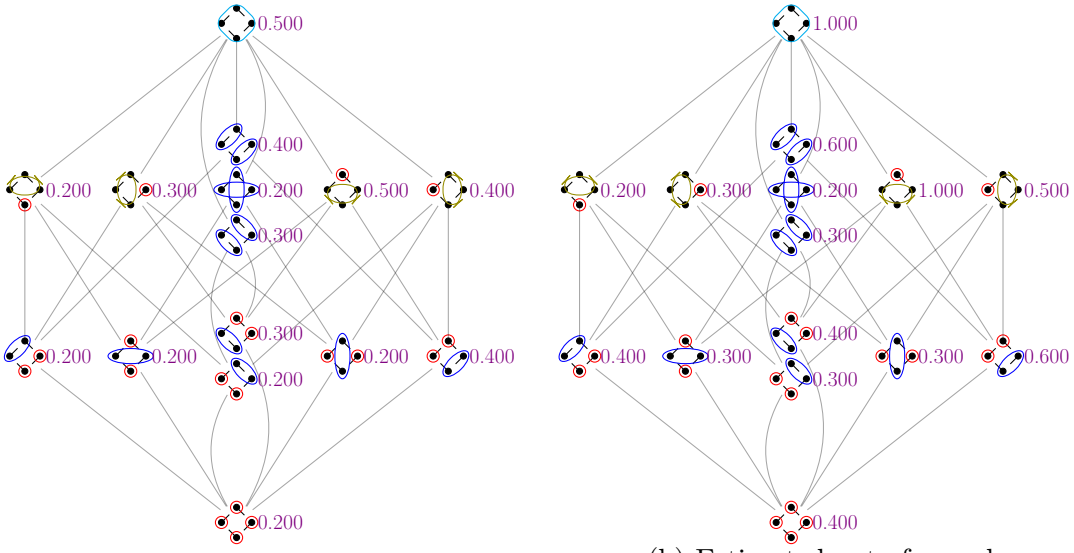
	Y=0	Y=1
X=00	0.300	0.000
X=01	0.100	0.300
X=10	0.000	0.100
X=11	0.100	0.100

(a) Estimated joint probability distribution.

$\mathbf{x}$	$f(\mathbf{x})$
00	0
01	1
10	1
11	0

(b) Boolean function with minimum in-sample error and minimum estimated out-of-sample error.

Figure 3.17: *Estimated joint probability distribution of random variables X and Y and the estimated function in Experiment 1*



(a) In-sample error of the best function in each model $\mathcal{H}_\pi$.

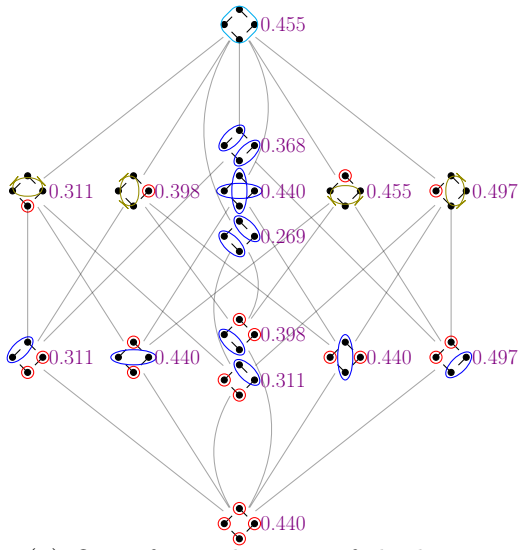
(b) Estimated out-of-sample (cross-validation) error of the best function in each model $\mathcal{H}_\pi$.

Figure 3.18: *In-sample and estimated out-of-sample errors in the model lattice using a defined dataset of 10 elements.*

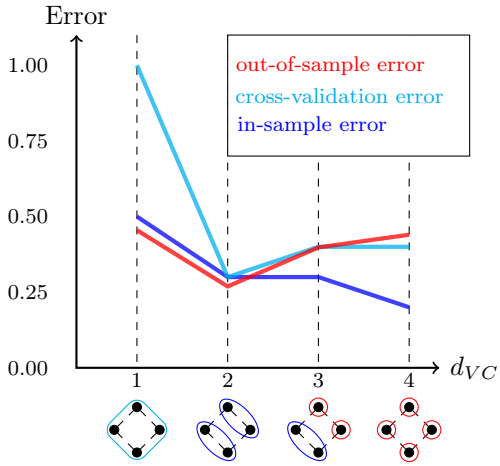
estimated out-of-sample and out-of-sample errors in average when the experiment is repeated 1000 times. For each repetition, we used a different dataset of size 20 obtained at random using $\Pr(X, Y)$. Thereby, these average errors are illustrated in Figure 3.20 and Figure 3.21.

For this experiment, the Generic model selector picks in average the model induced by the partition $\{\{00, 10\}, \{01, 11\}\}$ because it has the minimum estimated out-of-sample error, consequently, it also picks in average the target Boolean function.

Observing the results, as expected the average in-sample errors decreases when the VC dimension increases. Moreover, since we take the average of 1000 experiments it is expected that all estimated out-of-sample errors are close to the true out-of-sample errors. Additionally, most of the chains in the model lattice when estimating the out-of-sample error have the U-curve property. For instance, in Figure 3.19(b) we present the error curves in a chain of the model lattice.

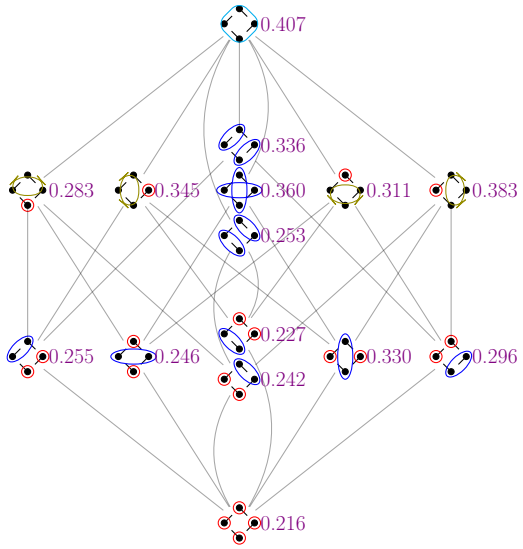


(a) Out-of-sample error of the best function in each model $\mathcal{H}_\pi$.

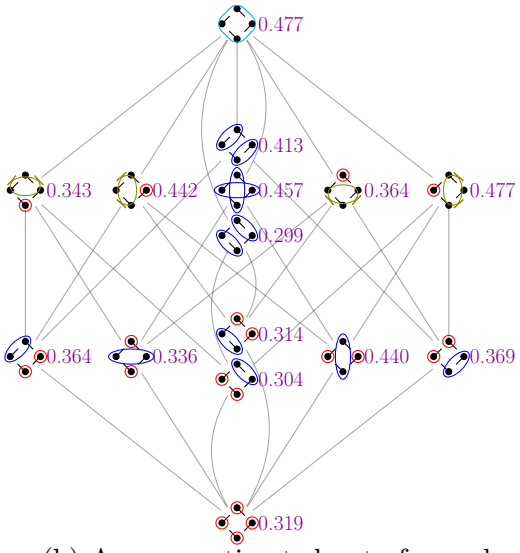


(b) Error curves in a chain of models in the model lattice.

Figure 3.19: *Out-of-sample errors in the model lattice and error curves in a chain using a defined dataset of 10 elements.*



(a) Average in-sample error of the best function in each model $\mathcal{H}_\pi$.



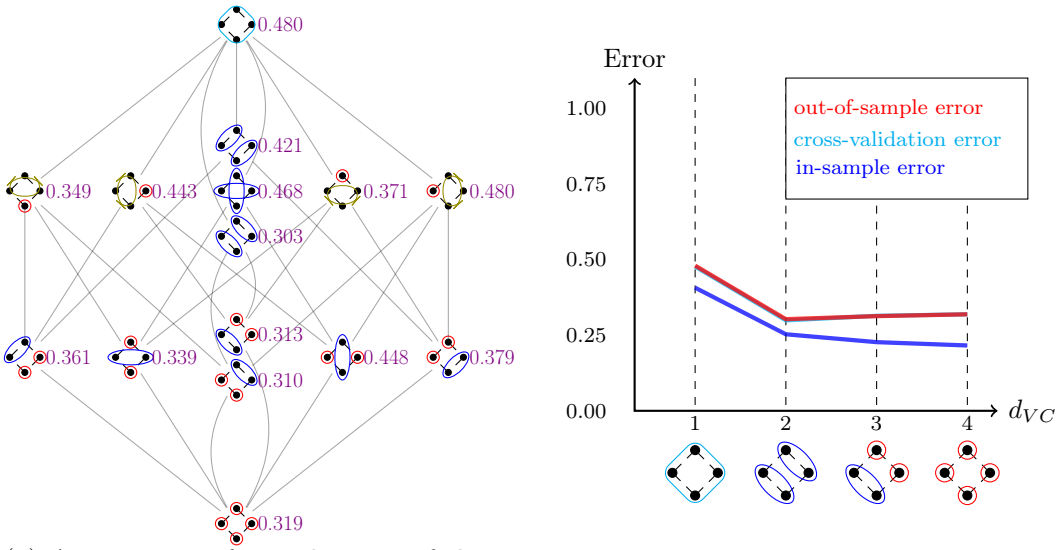
(b) Average estimated out-of-sample (cross-validation) error of the best function in each model $\mathcal{H}_\pi$.

Figure 3.20: *Average in-sample and estimated out-of-sample errors in the model lattice using 1000 datasets of 20 elements.*

Experiment 3

This experiment is very similar to Experiment 2, by contrast, we estimate the average errors using datasets of size 1000. Thereby, all these average errors are illustrated in Figure 3.22 and Figure 3.23.

Notice that, once we use a big dataset we can estimate correctly the joint probability distribution. Consequently, the in-sample, estimated out-of-sample and out-of-sample

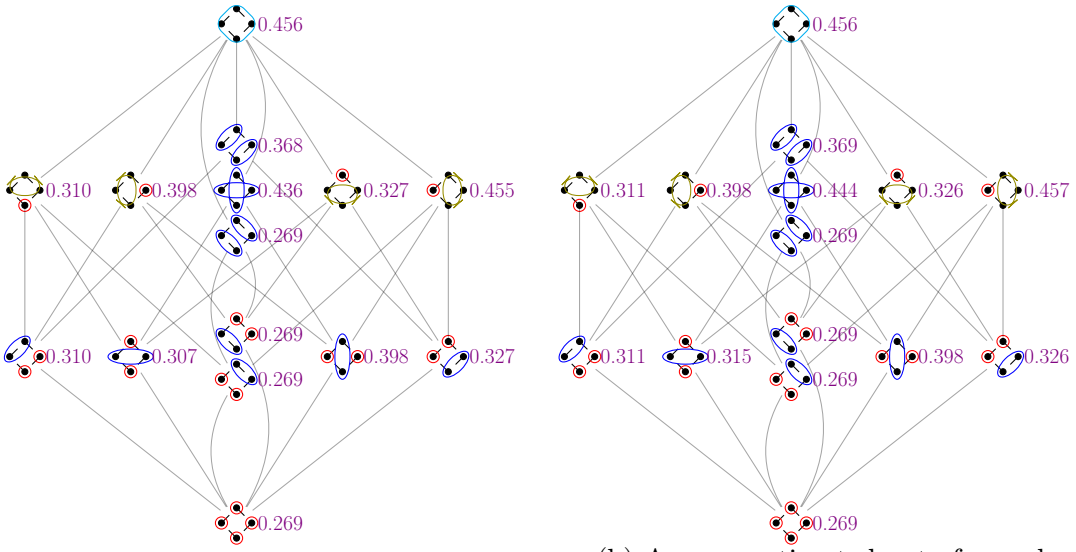


(a) Average out-of-sample error of the best function in each model $\mathcal{H}_\pi$.

(b) Average error curves in a chain of models in the model lattice.

Figure 3.21: *Average out-of-sample errors in the model lattice and error curves in a chain using 1000 datasets of 20 elements.*

errors are almost similar. Additionally, all curves tend to decrease in the model lattice.

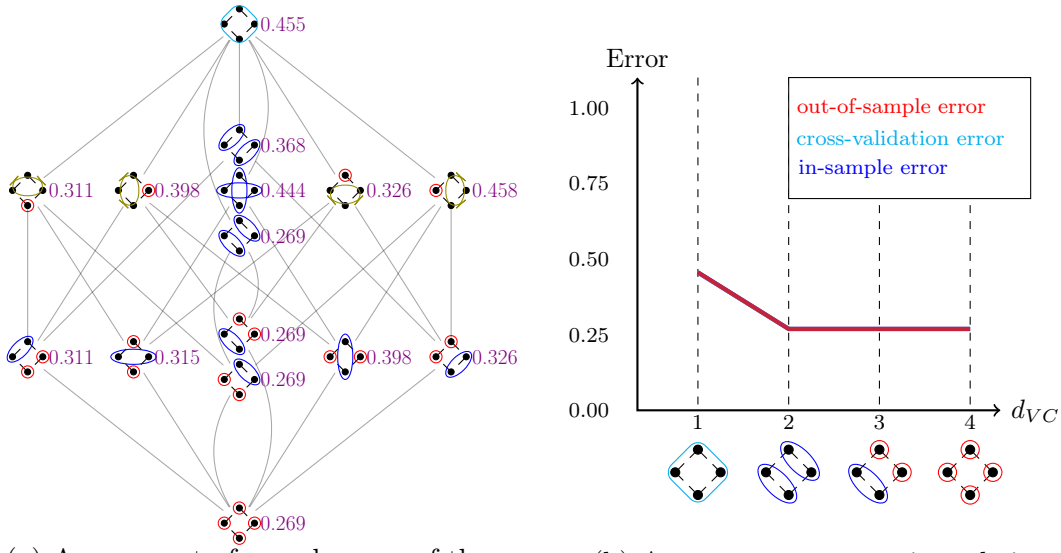


(a) Average in-sample error of the best function in each model $\mathcal{H}_\pi$.

(b) Average estimated out-of-sample (cross-validation) error of the best function in each model $\mathcal{H}_\pi$.

Figure 3.22: *Average in-sample and estimated out-of-sample errors in the model lattice using 1000 datasets of 1000 elements.*

In the next chapter, we present a search algorithm in lattices that can improve the efficiency of the proposed Generic model selector.



(a) Average out-of-sample error of the best function in each model $\mathcal{H}_\pi$.

(b) Average error curves in a chain of models in the model lattice.

Figure 3.23: *Average out-of-sample errors in the model lattice and error curves in a chain using 1000 datasets of 1000 elements.*

Chapter 4

Stochastic U-curve

In this chapter, we will introduce a stochastic algorithm to solve the U-curve problem called *Stochastic U-curve* algorithm, at first we will review some concepts and problems that motivate its design. As studied in Section 2.3.6, the *U-curve problem* is a search problem in a complete Boolean lattice $(\mathcal{P}(S), \subseteq)$ embedded with a cost function c with the U-curve property. Moreover, it is known that there exist proposed optimal solutions for this problem such as [Ris *et al.*, 2010], [Atashpaz-Gargari *et al.*, 2013], [Reis, 2013] and [Atashpaz-Gargari *et al.*, 2018]. Unfortunately, the U-curve problem is NP-hard so these approaches present exponential computational complexity. This issue suggests the formulation of suboptimal algorithms such as the SFS algorithm [Whitney, 1971] and the SFFS algorithm [Pudil *et al.*, 1994] that use heuristic greedy strategies to find decent solutions. The Stochastic U-curve algorithm is a suboptimal approximation algorithm but shows interesting probabilistic properties, so we introduce some probability concepts for defining this approach.

Following [DeGroot and Schervish, 2013], we define *probability* for a specific sample space according to our problem. In the U-curve problem, the sample space $\mathcal{S}$ is defined as all the possible subsets of a set S, this is $\mathcal{S} = \mathcal{P}(S)$. The set of events in the problem corresponds to all possible collections in $\mathcal{S}$, that is $\mathcal{P}(\mathcal{S})$. Finally, we define the probability Pr such that:

- For each event $\mathcal{E} \in \mathcal{P}(\mathcal{S})$, $\Pr(\mathcal{E}) \geq 0$.
- $\Pr(\mathcal{S}) = 1$.
- For any pairwise disjoint sets $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_n \in \mathcal{P}(\mathcal{S})$

$$\Pr(\bigcup_{i=1}^{n} \mathcal{E}_i) = \sum_{i=1}^{n} \Pr(\mathcal{E}_i).$$

Therefore, the tuple $(\mathcal{S}, \mathcal{P}(\mathcal{S}), \Pr)$ is a probability space (see Section 2.2).

Now, we define the event $\mathcal{L}(X) \subseteq \mathcal{S}$ being the subset of elements in $\mathcal{S}$ with costs less than X, i.e.

$$\mathcal{L}(X) = \{Y \in \mathcal{S} : c(Y) < c(X)\}.$$

In the U-curve problem, we will denote a specific instance of the problem as $\langle(\mathcal{S}, \subseteq), c\rangle$, then any algorithm should return as solution an element $X \in \mathcal{S}$. Therefore, suppose we obtain $X \in \mathcal{S}$ as output when we execute an algorithm $\mathbb{U}$ on an instance $\langle(\mathcal{S}, \subseteq), c\rangle$.

Then, the error of algorithm $\mathbb{U}$ is given by

$$error(\mathbb{U}\langle(\mathcal{S},\subseteq),c\rangle) = \Pr(\mathcal{L}(X)),$$

because all the elements in $\mathcal{L}(X)$ are better solutions than X.

4.1 Stochastic U-curve algorithms

In this section, we propose approximation algorithms for the U-curve problem satisfying the following restrictions: they return a unique element of $\mathcal{S}$ and the error of each algorithm is at most ϵ with probability at least $1-\delta$. As a basic result, we present Algorithm $\mathbb{U}^*$ (see Algorithm 10) that is a simple algorithm with interesting probabilistic properties. For instance, in Proposition 4.1.1 we restrict the value of variable m in the algorithm in order to satisfy a probability restriction.

Algorithm 10: Algorithm $\mathbb{U}^*$

Input: An instance $\langle(\mathcal{S},\subseteq),c\rangle$ of the U-curve problem and real numbers δ and ϵ.
Output: A subset $R \in \mathcal{S}$ such that $\Pr(error(\mathbb{U}^*\langle(\mathcal{S},\subseteq),c\rangle) \leq \epsilon) \geq 1-\delta$.

```
R = ∅;
m = ⌈(1/ε) ln(1/δ)⌉;
for i ← 1 to m do
    Select an element X ∈ S at random;
    if R = ∅ or c(X) < c(R) then
        R ← X;
    end
end
return R;
```

Proposition 4.1.1. *In Algorithm $\mathbb{U}^*$, m must be at least $\lceil \frac{1}{\epsilon}\ln\frac{1}{\delta}\rceil$ to satisfy*

$$\Pr(error(\mathbb{U}^*\langle(\mathcal{S},\subseteq),c\rangle) \leq \epsilon) \geq 1-\delta.$$

Proof. This demonstration is based on the proof that the Learning Rays algorithm is Probably Approximately Correct shown in [Anthony and Biggs, 1992].

We observe that Algorithm $\mathbb{U}^*$ picks m aleatory elements of $\mathcal{S}$ with repetition. Let $\mathcal{X}$ be the set of aleatory selected elements, i.e. $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$, so we return as solution the element R with minimum cost in $\mathcal{X}$. Let M be the element with minimum cost in $\mathcal{S}$ such that $\Pr(\mathcal{L}(M)) \geq \epsilon$. Then, the probability of each element of $\mathcal{X}$ has cost more or equal to M is at most $(1-\epsilon)$. Therefore, the probability of all elements of $\mathcal{X}$ have cost more or equal to M is at most $(1-\epsilon)^m$. Taking the complementary event, it follows that the probability that the set $\mathcal{X}$ does not contain any element X satisfying $\Pr(\mathcal{L}(X)) \leq \epsilon$ is at least $1-(1-\epsilon)^m$. Thus, we have

$$\Pr(error(\mathbb{U}^*\langle\mathcal{S},c\rangle)) \leq \epsilon) \geq 1-(1-\epsilon)^m.$$

In order to make the right-hand side greater than $1-\delta$ we can take

$$m \geq m_0 = \lceil \frac{1}{\epsilon}\ln\frac{1}{\delta}\rceil. \tag{4.1}$$

For then it follows that

$$(1-\epsilon)^m \leq (1-\epsilon)^{m_0} < \exp(-\epsilon m_0) < \exp(\ln \delta) = \delta.$$

□

Although, Algorithm $\mathbb{U}^*$ satisfies ϵ and δ restrictions, it is clear that the computational complexity of the algorithm is $O(\lceil \frac{1}{\epsilon} \ln \frac{1}{\delta} \rceil)$, consequently, for small values of ϵ and δ the complexity of the algorithm is too high and turn this algorithm impractical. In particular, Algorithm $\mathbb{U}^*$ does not take advantage of the lattice structure property nor the U-curve property. Thereby, we propose the Stochastic U-curve algorithm, or simply Algorithm $\mathbb{U}$ (see Algorithm 11) that satisfies ϵ and δ restrictions and exploits these properties reducing the computational complexity of the algorithm.

Algorithm 11: Stochastic U-curve algorithm (Algorithm $\mathbb{U}$).

Input: An instance $\langle(\mathcal{S}, \subseteq), c\rangle$ of the U-curve problem and real numbers δ and ϵ.
Output: A subset $R \in \mathcal{S}$ such that is expected that
$\Pr(error(\mathbb{U}\langle(\mathcal{S}, \subseteq), c\rangle) \leq \epsilon) \geq 1 - \delta$.

```
1  R = ∅;
2  m = Estimate-m(ε, δ);
3  for i ← 1 to m do
4      set V = ∅;
5      Select an element X ∈ S at random;
6      V = V ∪ {X};
7      N(X) = {Y ∈ S : Y neighbor of X and c(Y) ≤ c(X)};
8      while N(X) ≠ ∅ do
9          X ← select at random from N(X);
10         V = V ∪ {X};
11         N(X) = {Y ∈ S : Y neighbor of X, c(Y) ≤ c(X) and Y ∉ V};
12     end
13     if R = ∅ or c(X) < c(R) then
14         R ← X;
15     end
16 end
17 return R;
```

Some characteristics of Algorithm $\mathbb{U}$ are presented below:

1. For every iteration of the outer loop (line 3), the algorithm creates a sequence $X_1, X_2, \ldots, X_T$ such that $c(X_1) \geq c(X_2) \geq \cdots \geq c(X_T)$.
2. $X_2 \in \mathcal{N}(X_1), X_3 \in \mathcal{N}(X_2), \ldots, X_T \in \mathcal{N}(X_{T-1})$ and $\mathcal{N}(X_T) = \emptyset$.
3. X_T is local minimum, i.e. all its neighbors have cost more or equal than X_T.

The next example shows an instance of the Algorithm $\mathbb{U}$ execution.

Example 4.1.1. We illustrate in Figure 4.1, the execution of only one iteration of the inner loop of Algorithm $\mathbb{U}$ (line 8). The assignments of variable X can be seen as a sequence of elements in $\mathcal{S}$, in this example the sequence is $(01000, 01100, 01110, 01010, 00010)$, satisfying $c(01000) \geq c(01100) \geq c(01110) \geq c(01010) \geq c(00010)$. It is worth

noting that all the element neighbors satisfying the conditions had the same chances to be selected as next element, e.g. each element in $\mathcal{N}(01000) = \{01100, 01001, 01010, 11000\}$ had the same chance to be the second selected element. We observe that the element 01000 was the first element to be selected at random and the element 00010 was the last assignment of X because 00010 is a local minimum in the lattice.

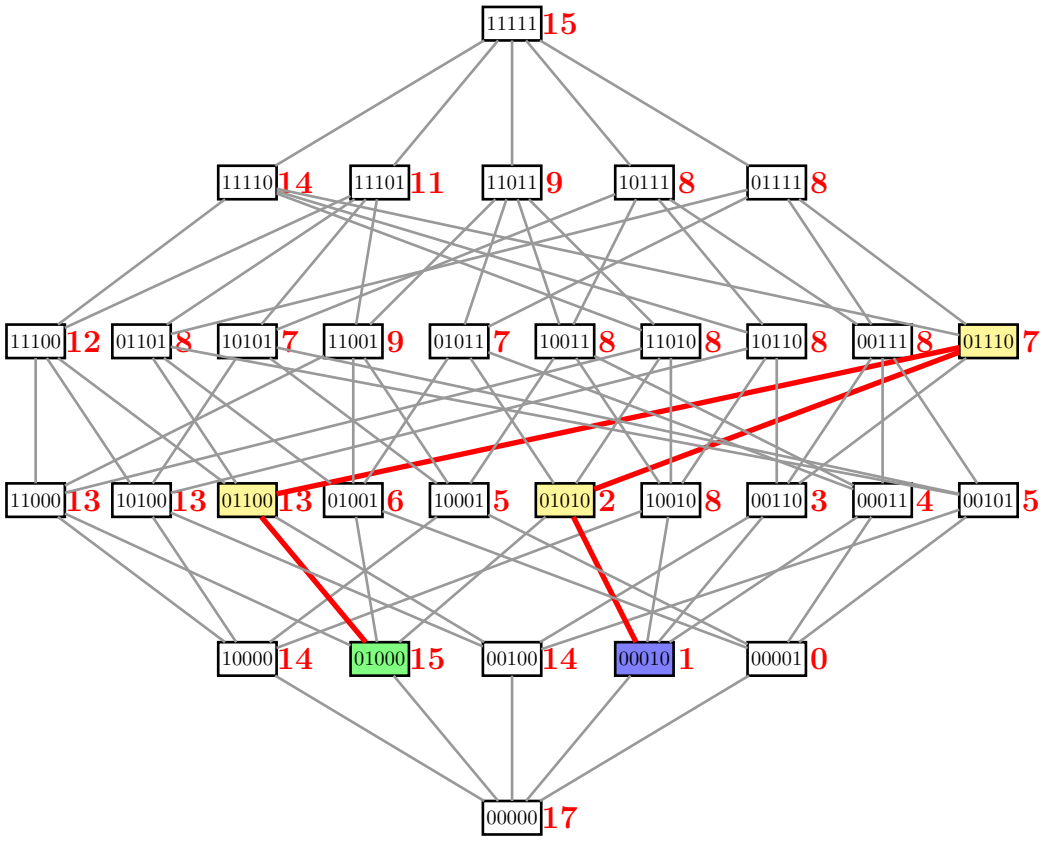


Figure 4.1: *Execution of one iteration of the inner loop in Algorithm* $\mathbb{U}$ *(line 8), credits:[Reis, 2013].*

Now, we will study how large m must be in Algorithm $\mathbb{U}$ to satisfy the inequality

$$\Pr(error(\mathbb{U}\langle\mathcal{S}, c\rangle)) \leq \epsilon) \geq 1 - \delta.$$

First, we describe the following two topological properties of the U-curve problem in order to find a estimative for m.

1. We define a local minimum to be an element X in the lattice such that none of its neighbors has cost less than $c(X)$. Therefore, every element that is not a local minimum has at least one neighbor with lower cost.

2. Since the unique restriction in c is that it must have the U-curve property, then finding the element with minimum cost on the lattice (global minimum) implies checking every local minimum in the lattice because of the independence of costs between local minima.

Thereby, Algorithm $\mathbb{U}$ (see Algorithm 11) uses these two properties as a search strategy to find local minima in the lattice, i.e. at the end of line 12 the variable X

is assigned to a local minimum. Furthermore, Algorithm $\mathbb{U}$ was designed to pick m aleatory elements with repetition from the subset of local minima in the lattice. However, we believe that there is no algorithm to achieve that property, so Algorithm $\mathbb{U}$ uses a heuristic to find a local minimum at random from the subset of local minima. Using these properties, we present the following proposition to estimate the size of m under certain conditions.

Proposition 4.1.2. *Let X be the element with minimum cost in the lattice such that $\Pr(\mathcal{L}(X)) \geq \epsilon$, and let $\mathcal{M} \subseteq \mathcal{S}$ be the subset of local minima in the problem. Under the following three conditions:*

1. $\Pr(\mathcal{M}) \geq \epsilon$.
2. *All elements in $\mathcal{L}(X)$ belong to $\mathcal{M}$.*
3. *Algorithm $\mathbb{U}$ picks m aleatory elements with repetition from $\mathcal{M}$.*

We have that m must be

$$m \geq m_0 = \left\lceil \frac{\Pr(\mathcal{M})}{\epsilon} \ln \frac{1}{\delta} \right\rceil$$

to satisfy $\Pr(error(\mathbb{U}\langle(\mathcal{S}, \subseteq), c\rangle)) \leq \epsilon) \geq 1 - \delta$.

Proof. If all those three conditions are satisfied, this proposition is similar to Proposition 4.1.1 evaluating Algorithm $\mathbb{U}^*$ with the only difference that the selection of the m aleatory elements is from $\mathcal{M}$ and not from $\mathcal{S}$. Then, from Proposition 4.1.1 and using the three conditions, we know that m must be at least $\lceil \frac{1}{\epsilon} \ln \frac{1}{\delta} \rceil$ to satisfy that the probability of picking m elements from $\mathcal{S}$ and none of these m elements has cost less or equal than X is $1 - \delta$. However, as we are picking m elements from $\mathcal{M}$ and not from $\mathcal{S}$, the mass of probability that represents ϵ in $\mathcal{S}$ is given by $\frac{\epsilon \cdot \Pr(\mathcal{S})}{\Pr(\mathcal{M})} = \frac{\epsilon}{\Pr(\mathcal{M})}$ (because $\Pr(\mathcal{S}) = 1$). Replacing ϵ in $\lceil \frac{1}{\epsilon} \ln \frac{1}{\delta} \rceil$ by the real mass of probability we obtain that m must be at least $\lceil \frac{\Pr(\mathcal{M})}{\epsilon} \ln \frac{1}{\delta} \rceil$ to satisfy $\Pr(error(\mathbb{U}\langle\mathcal{S}, c\rangle)) \leq \epsilon) \geq 1 - \delta$. □

Nevertheless, the assumptions in Proposition 4.1.2 are strong to prove the exact value of m, the presented result could be used as a lower bound for the real value m because Algorithm $\mathbb{U}$ may need more iterations than $\lceil \frac{\Pr(\mathcal{M})}{\epsilon} \ln \frac{1}{\delta} \rceil$ to satisfy the restrictions in the problem. Anyway, this lower bound cannot be calculated because we do not know the exact value of $\Pr(\mathcal{M})$. So, we are going to estimate this value as follows:

At first, we assume that all the elements in $\mathcal{S}$ have the same probability to be chosen. Now, let p be the proportion of local minima in the lattice, and let $\mathcal{D}$ be a sample of $\mathcal{S}$ with k elements, i.e. $\mathcal{D} \subseteq \mathcal{S}$, $|\mathcal{D}| = k$. We define an estimator of p as $\hat{p} = M/k$ where M is a random variable that counts the number of local minima in the sample $\mathcal{D}$. Therefore, the binomial distribution describes the behavior of M, so M is a binomial distributed random variable with parameters k and p ($M \sim B(k, p)$). Moreover, since it is expected that Algorithm $\mathbb{U}$ is used in practical problems, we assume that k is large enough. Therefore, we can use the Central Limit Theorem that states that for large values of k, the distribution of the random variable M is approximately normal [DeGroot and Schervish, 2013]. In fact, this approximation is not good for small values of k, so as a rule of thumb we use this approximations when $kp \geq 10$ and $k(1 - p) \geq 10$.

Therefore, for large values of k we can use the normal approximation to obtain a confidence interval for p as

$$\hat{p} \pm Z_c \sqrt{\frac{\hat{p}(1 - \hat{p})}{k}}$$

where Z_c is the $1 - \frac{1}{2}c$ quantile of a standard normal distribution corresponding to the target error rate c. However, we are interested in the value of k to obtain a small error of approximation. Thereby, we know that the error of approximation is given by $Error = Z_c\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}$, so the value of k is given by

$$k = \hat{p}(1-\hat{p})(\frac{Z_c}{Error})^2.$$

At the beginning of the estimation process we do not know the value of $\hat{p}$, but considering the worst case for k when $\hat{p} = \frac{1}{2}$, we obtain $k = \frac{1}{4}(\frac{Z_c}{Error})^2$. For example, in order to generate a confidence interval of 99% with maximum error 0.01 we have $k = \frac{1}{4}(\frac{Z_{0.99}}{0.01})^2 = 13572$. This means that, using a set $\mathcal{D}$ of size 13572, we can estimate p using the value $\hat{p} = M/k$ satisfying $\hat{p} - 0.01 \leq p \leq \hat{p} + 0.01$ with probability 99%. Thus, considering the worst case scenario we will use the estimative $\hat{p} + 0.01$ for $\Pr(\mathcal{M})$ in our experiments. Using this estimation we present the Algorithm Estimate-m (see Algorithm 12) in order to complement Algorithm $\mathbb{U}$.

Algorithm 12: Algorithm Estimate-$m(\delta, \epsilon)$

Input: Real numbers δ and ϵ.
Output: An estimate of the value of m.

```
k ← 13572;
M ← 0;
for i ← 1 to k do
    Select an element X ∈ S at random;
    N(X) = {Y ∈ S : Y neighbor of X and c(Y) < c(X)};
    if N(X) ≠ ∅ then
        M ← M + 1;
    end
end
P̂r(M) = M/k + 0.01;
m = ⌈(P̂r(M)/ε) ln(1/δ)⌉;
return m;
```

It is important to notice that when we use Algorithm $\mathbb{U}$ in instances with larger lattices ($|\mathcal{S}| = 2^n$ grows exponentially), the role of the variable $Error$ becomes crucial for estimating the correct value of m. In fact, when n is large it is expected that for most instances of the U-curve problem $\Pr(\mathcal{M})$ is much smaller than 0.01. So, the estimation for $\Pr(\mathcal{M})$ given by $\hat{p} + Error$ where $Error = 0.01$ is inaccurate in most cases when n is large because $\hat{p}$ can be much smaller than 0.01. Thus, in order to obtain a good estimate for $\hat{p}$ we can reduce the value of $Error$, that would imply an increment in the value of the variable k. For example, in order to generate a confidence interval of 99% with maximum error 0.001 we have $k = \frac{1}{4}(\frac{Z_{0.99}}{0.001})^2 = 1357225$, so the estimate for $\Pr(\mathcal{M})$ will be given by $\hat{p} + 0.001$. Therefore, there exist a trade-off between the accuracy of the estimate for $\Pr(\mathcal{M})$ and the maximum error. Thereby, in situations where m is really big because of the bad estimation of $\Pr(\mathcal{M})$ we can decrease the value of the variable $Error$, that will increment the value of k but we will obtain a better estimation of $\Pr(\mathcal{M})$ and in consequence a smaller value for m.

4.2 Computational complexity

In this section, we analyze the computational complexity of the Stochastic U-curve algorithm (Algorithm $\mathbb{U}$). We observe that Algorithm $\mathbb{U}$ (see Algorithm 11) depends on m and the number of times the inner loop (line 8) is executed. The worst-case complexity of Algorithm $\mathbb{U}$ is when the inner loop is executed an exponential number of times, e.g. when the algorithm visits each element of $\mathcal{S}$, considering $|\mathcal{S}| = 2^n$ the computational complexity is $O(2^n n)$. However, Algorithm $\mathbb{U}$ has a decent performance in the average case, in the next proposition we show an important property of Algorithm $\mathbb{U}$ when executed under certain conditions.

Proposition 4.2.1. *Having the following assumptions: Let $\langle(\mathcal{S}, \subseteq), c\rangle$ be a lattice embedded with any cost function $c : \mathcal{S} \to \mathbb{R}$ (not necessarily satisfying the U-curve property), and the costs of every element in $\mathcal{S}$ are different pairwise (i.e. $\forall X, Y \in \mathcal{S}, c(X) = c(Y) \Leftrightarrow X = Y$). Then, the expected number of times the inner loop in Algorithm $\mathbb{U}$ (line 8) is executed is $O(n)$.*

Proof. Let $\mathcal{S} = \{X_1, X_2, \ldots, X_{2^n}\}$ be the sample space. Since $\forall X, Y \in \mathcal{S}, c(X) = c(Y) \Leftrightarrow X = Y$, we could establish an order relation between all the elements in $\mathcal{S}$ in the following way: $c(X_{o(1)}) < c(X_{o(2)}) < \cdots < c(X_{o(2^n)})$ where $(o(1), o(2), \ldots, o(2^n))$ is a permutation of $(1, 2, \ldots, 2^n)$.

Notice that in each iteration of the outer loop (line 3) in Algorithm $\mathbb{U}$, the variable X receives different values of $\mathcal{S}$, this variable assignment of X can be seen as a Markov chain, this is a sequence of random variables $Y_1, Y_2, Y_3, \ldots, Y_T$ with the Markov property, where the realizations of each variable are in $\mathcal{S}$. Thus, we have the sequence $\mathcal{Y} = (Y_1, Y_2, \ldots, Y_T)$ with the property that for any pair of two consecutive elements in the sequence $\mathcal{Y}$, it holds that if (X_a, X_b) are the realizations of (Y_t, Y_{t+1}) respectively, then $c(X_a) > c(X_b)$. Thus, our problem can be reduced to obtain the expected value of the size of the sequence $\mathcal{Y}$.

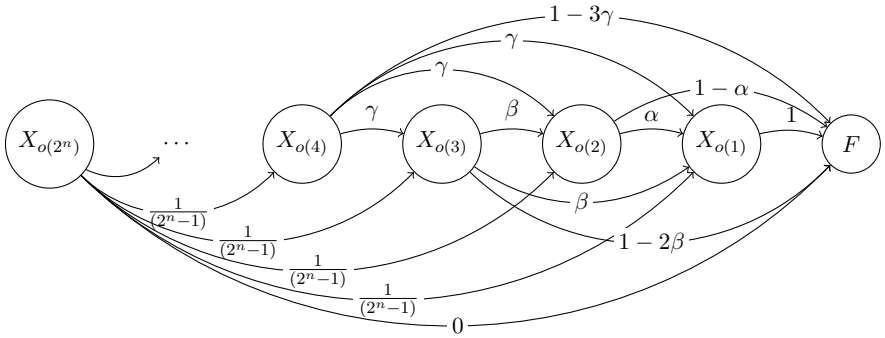
Since we are calculating the expected value of the size of any possible sequence, then every realization of Y_{t+1} has the same probability given the realization of Y_t. This is:

$$\Pr(Y_{t+1} = X_{o(i-1)} | Y_t = X_{o(i)}) = \Pr(Y_{t+1} = X_{o(i-2)} | Y_t = X_{o(i)}) = \cdots =$$
$$\Pr(Y_{t+1} = X_{o(2)} | Y_t = X_{o(i)}) = \Pr(Y_{t+1} = X_{o(1)} | Y_t = X_{o(i)}).$$

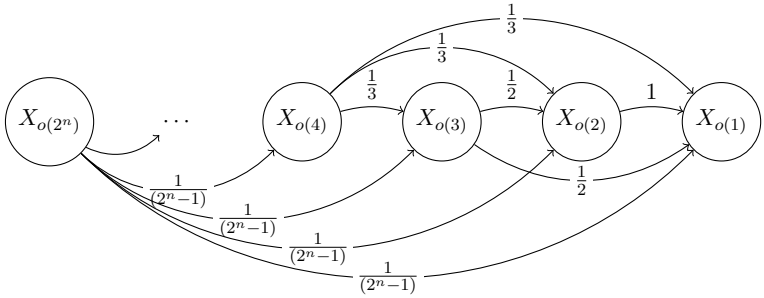
However, these conditional probabilities of the Markov chain are difficult to calculate since it depends on factors such as the lattice structure and the number of neighbors of each node. Contrarily, we know that the sequence $\mathcal{Y}$ stops when the selected node X does not have any neighbors with cost less than $c(X)$ or all the neighbors of X with cost less than $c(X)$ were previously visited. Thus, each node has a defined probability to stop, it can be modeled in the Markov chain creating a final state node F. The modeled Markov chain is illustrated on Figure 4.2(a), at the beginning of the problem all the states except the terminal state has the same probability to be the first element in the sequence, i.e. $\Pr(Y_1 = X_{o(i)}) = \frac{1}{2^n}, 0 \leq i \leq 2^n$.

Unfortunately, we could not determine the exact expected value of the size of the sequence $\mathcal{Y}$, but there exists a similar problem also modeled as a Markov chain called "Pure adaptive search for finite global optimization" [Zabinsky *et al.*, 1995] (illustrated in Figure 4.2(b)), where the expected value of the created sequence is bounded above by $1 + \log(2^n)$. Comparing both Markov chains in Figure 4.2, we observe that all the transitions in the second Markov chain are present in the first Markov chain, so it is

straightforward to prove that the conditional probability for any common transition in the first Markov chain is less or equal than the transition in the second Markov chain (i.e. in Figure 4.2 we have that $\alpha \leq 1$, $\beta \leq \frac{1}{2}$, $\gamma \leq \frac{1}{3}$, ...). In consequence, since we have one more state in the first Markov chain and the probability for each node to stop earlier in the first Markov chain is greater than in the second Markov chain, then the expected size of the sequence in our problem is at most one step more than the expected size of the created sequence by the second Markov chain. Thus, the expected size of the sequence $\mathcal{Y}$ is bounded above by $2 + \log(2^n)$, so the expected number of times the inner loop is executed in Algorithm $\mathbb{U}$ (line 8) is $O(n)$. □



(a) Markov chain of our problem.



(b) Markov chain of the Strong Pure adaptive search problem.

Figure 4.2: *Markov chains described in Proposition 4.2.1.*

Although the assumptions in Proposition 4.2.1 are a little different than our problem, we assume that the expected number of times the inner loop of Algorithm $\mathbb{U}$ is executed is approximately $O(n)$ also supported by experiments as we will show in Section 4.3. Thereby, in each iteration of the inner loop of Algorithm $\mathbb{U}$ we select an element X and then we check all the n neighbors of X, so the complexity of each iteration is $O(n)$. Hence, the expected complexity of Algorithm $\mathbb{U}$ is approximately $O(n^2 m)$. Since we prove that m is approximately $\lceil \frac{\Pr(\mathcal{M})}{\epsilon} \ln \frac{1}{\delta} \rceil$, then the expected computational complexity of Algorithm $\mathbb{U}$ is $O(n^2 m) = O(\frac{n^2 \Pr(\mathcal{M})}{\epsilon} \ln \frac{1}{\delta})$ for instances $\langle (\mathcal{S}, \subseteq), c \rangle$ satisfying the U-curve property.

4.3 Experiments

In this section, we present two experiments that show the efficiency of Algorithm $\mathbb{U}$, for the realization of these experiments we used a framework for benchmarking of

feature selection algorithms and cost functions called *featsel* [Reis *et al.*, 2017].

Experiment 1

As first experiment, we evaluate whether Algorithm $\mathbb{U}$ works as expected in different instances of the U-curve problem, i.e. we check whether Algorithm $\mathbb{U}$ satisfies

$$\Pr(error(\mathbb{U}\langle\mathcal{S}, c\rangle)) \leq \epsilon) \geq 1 - \delta.$$

Accordingly, we created 40 different random instances of the problem, these instances are divided in 4 groups of 10, the first group are lattices generated by 20 features (i.e. lattices with 2^{20} elements), the second group are lattices generated by 21 features, the third by 22 features, and the last group by 23 features. In Table 4.1, we show the diversity of the instances, presenting the number of local minima in each level of the lattice.

For each instance we created 25 examples, where every example uses a different input pair of values ϵ and δ. All pairs are obtained from the Cartesian product of sets $E \times D$ where $E = \{0.00001, 0.00002,\ 0.00005, 0.0001, 0.001\}$ and $D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$. Thus, we executed Algorithm $\mathbb{U}$ 1000 times, each of them with different input parameters and obtain the following results: In 99.4% of the examples, Algorithm $\mathbb{U}$ satisfies

$$\Pr(error(\mathbb{U}\langle\mathcal{S}, c\rangle)) \leq \epsilon) \geq 1 - \delta.$$

Moreover, the probabilities of success for every pair ϵ, δ are shown in Table 4.2. It is worth mentioning that all unsuccessful results were presented in only 2 instances.

Experiment 2

As a second experiment, we present a comparison between three search algorithms in lattices presented in the literature (U-curve Search (UCS) [Reis, 2013], SFS [Whitney, 1971], SFFS [Pudil *et al.*, 1994]) and Algorithm $\mathbb{U}$. In this experiment, the objective is to find the element with minimum cost in a lattice. However, we know that Algorithm $\mathbb{U}$ is a stochastic algorithm, so we will try to find the element with minimum cost using input parameters $\epsilon = 0.00001$ and $\delta = 0.001$. In the experiment, for each size of lattice we created ten random generated instances using featsel, and for each instance we performed the different algorithms.

In Table 4.3 and Table 4.4 are shown the results of this experiment. The following items show the label descriptions in these tables:

- n: considering this problem as a feature selection problem, n is the number of features in the problem.
- 2^n: number of elements in the search space.
- Total Time (sec): required time to run each algorithm (average of up to 10 executions).
- Cost Function Time (sec): required time for the computation of all calls to the cost function, during the execution of each algorithm (average of 10 executions).
- # Computed nodes: number of times the chosen cost function is computed by each algorithm (average of 10 executions).

Instance	log(size)	Number of local minima per level
1	20	[0 8 13 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
2	20	[0 0 0 0 0 0 54 468 1675 3522 4947 4289 2456 815 132 4 0 0 0 0 0]
3	20	[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
4	20	[0 0 0 4 203 871 2025 2488 1806 652 116 2 0 0 0 0 0 0 0 0 0]
5	20	[0 0 0 0 59 724 2861 5641 5196 2596 539 41 0 0 0 0 0 0 0 0 0]
6	20	[0 8 5 4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
7	20	[0 0 0 0 0 0 0 0 69 871 3679 7580 8305 4895 1453 159 0 0 0 0 0]
8	20	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 27 53 47 0 0]
9	20	[0 0 0 0 0 0 0 0 0 1 61 282 656 788 870 731 304 26 0 0 0]
10	20	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 33 52 40 19 3 0]
11	21	[0 3 27 81 88 17 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
12	21	[0 0 2 151 276 345 262 134 26 2 0 0 0 0 0 0 0 0 0 0 0]
13	21	[0 0 0 0 0 0 0 2 116 724 2231 3546 4204 3847 2410 854 169 0 0 0 0 0]
14	21	[0 0 1 164 473 673 456 115 9 0 0 0 0 0 0 0 0 0 0 0 0]
15	21	[0 0 0 0 0 0 0 1 159 1697 7626 17207 20274 12980 4217 711 18 0 0 0 0 0]
16	21	[0 0 0 33 626 1678 2112 1100 224 9 0 0 0 0 0 0 0 0 0 0 0]
17	21	[0 0 0 0 0 0 0 0 0 5 164 806 959 999 1298 941 235 2 0 0 0]
18	21	[0 0 0 0 0 0 0 0 0 0 14 123 288 493 496 458 315 89 0 0 0]
19	21	[0 0 0 0 0 0 0 0 0 0 0 0 5 147 638 1055 756 218 0 0 0]
20	21	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 8 4 0]
21	22	[0 0 0 0 0 0 0 0 0 0 0 54 280 782 1287 1391 967 515 64 0 0 0]
22	22	[0 0 0 34 1988 2770 1218 108 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
23	22	[0 0 0 0 0 0 0 0 0 1 48 467 1900 3844 4436 3124 1332 277 0 0 0 0]
24	22	[0 0 0 0 0 0 0 0 0 0 0 0 5 66 297 619 631 425 162 20 0 0]
25	22	[0 0 0 0 0 0 0 0 0 457 4830 12546 16025 16067 10860 2771 123 0 0 0 0 0]
26	22	[0 0 47 117 89 71 21 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
27	22	[0 0 0 0 0 0 116 1136 3600 5585 5350 5313 5382 3265 931 85 0 0 0 0 0 0]
28	22	[0 0 9 99 307 587 632 469 274 90 4 0 0 0 0 0 0 0 0 0 0]
29	22	[0 0 0 0 0 0 0 574 5351 9754 8795 9394 8588 2880 106 0 0 0 0 0 0 0]
30	22	[0 0 0 0 0 0 161 1364 4905 10847 15327 14086 8658 3401 730 47 0 0 0 0 0 0]
31	23	[0 0 0 0 0 0 0 0 30 1364 11509 39081 63747 59294 31198 6961 589 0 0 0 0 0 0]
32	23	[0 8 23 24 13 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
33	23	[0 0 0 0 0 0 0 0 0 0 13 1007 5691 11319 14550 13806 9358 2612 117 0 0 0 0]
34	23	[0 0 0 0 0 0 100 2555 14028 22240 21657 21090 12673 2503 119 0 0 0 0 0 0 0 0]
35	23	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 1 1 6 0]
36	23	[0 0 0 266 946 1064 715 206 6 0 0 0 0 0 0 0 0 0 0 0 0 0]
37	23	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 73 119 64 0 0]
38	23	[0 0 29 76 254 370 410 382 192 42 7 0 0 0 0 0 0 0 0 0 0 0]
39	23	[0 0 0 0 0 0 0 0 0 31 1762 6131 11691 14610 12293 7796 2240 10 0 0 0 0]
40	23	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 44 61 74 85 58 0 0]

Table 4.1: *Vector of number of local minima for each instance.*

ϵ \ δ	0.001	0.005	0.01	0.05	0.1
0.00001	1.00	1.00	1.00	1.00	1.00
0.00002	1.00	0.975	0.975	0.95	0.975
0.00005	1.00	1.00	1.00	1.00	0.975
0.0001	1.00	1.00	1.00	1.00	1.00
0.001	1.00	0.975	1.00	1.00	1.00

Table 4.2: *Probability of success for every pair ϵ, δ.*

- # Best solution found: number of times (out of 10) that each algorithm found the best solution (element with minimum cost).

		Total Time (sec)				Cost Function Time (sec)			
n	2^n	UCS	SFS	SFFS	$\mathbb{U}$	UCS	SFS	SFFS	$\mathbb{U}$
18	262144	1.32	0.01	0.02	6.56	0.77	0.00	0.01	4.01
19	524288	2.96	0.01	0.02	7.11	1.70	0.00	0.01	4.31
20	1048576	5.69	0.01	0.02	8.32	3.22	0.00	0.01	5.01
21	2097152	13.28	0.01	0.02	9.36	7.43	0.00	0.01	5.63
22	4194304	26.09	0.01	0.02	9.17	14.44	0.00	0.02	5.46
23	8388608	57.12	0.01	0.04	11.80	31.22	0.00	0.03	6.99
24	16777216	125.28	0.01	0.04	12.15	67.42	0.00	0.03	7.14
25	33554432	202.01	0.01	0.03	11.43	106.47	0.00	0.02	6.54

Table 4.3: *Statistics of finding the element with minimum cost in different lattices.*

		# Computed nodes				# Best solution found			
n	2^n	UCS	SFS	SFFS	$\mathbb{U}$	UCS	SFS	SFFS	$\mathbb{U}$
18	262144	127746.8	96.5	1482.6	379863.2	10	1	1	9
19	524288	281943.1	109.6	1443.9	393458.0	10	1	1	6
20	1048576	537628.1	115.1	1799.5	461405.7	10	2	2	9
21	2097152	1234867.2	138.3	2526.3	493024.4	10	0	0	7
22	4194304	2362784.8	150.5	2246.4	473084.7	10	0	0	4
23	8388608	5113903.5	176.0	3841.7	585904.8	10	0	0	6
24	16777216	11138126.9	200.5	4255.0	594240.0	10	0	0	6
25	33554432	17552359.7	180.1	2547.1	542825.4	10	0	0	5

Table 4.4: *Statistics of finding the element with minimum cost in different lattices.*

Chapter 5

Conclusions and Future Work

In recent years, machine learning has become an attractive field of research. However, it remains as an open field to be explored in which many questions are still unanswered. Thereby, we believe that the contributions of this thesis advance both the state of the art in machine learning theory and in practical problems solution. In this chapter, we review the principal contributions of this thesis along with some discussions about the presented results.

In Section 3.1, we studied a new family of Boolean models based on partitions of the domain space. We saw that each of these models has a defined size and also a fixed VC-dimension, these properties can be used to deduce their sample complexity which in practice are used for model selectors to find models with good generalization properties. Moreover, we proved that the set of the models generated by all partitions of the domain along with a partial order form the model lattice which is anti-isomorphic to the partition lattice. We have also shown that if all models in the model lattice are restricted by other hypothesis sets, then the restricted models continue maintaining their partial order in the lattice.

In Section 3.2, we proposed a new model selector that we call *Generic model selector* (GMS). In contrast to typical model selectors, GMS uses a partition set to restrict the partition space and generate its own models, then it picks the best model according to the training data provided. On real applications, the specified partition set is not given by the user, but the partition set is generated implicitly by different learning methods. We showed that GMS generalizes the set of learning methods that use models induced by partition sets, for example it unifies a set of known learning methods such as the feature selection problem, the multiresolution representation and the decision tree representation.

In Section 3.3, we proposed an improvement of the GMS using poset search algorithms in order to reduce the number of visited partitions in the poset. In this way, we can take advantage of the wide variety of search algorithms in the literature. Moreover, we studied that search strategies can exploit the fact that errors obtained from nested models present the U-curve phenomenon.

In Section 3.4, we presented experiments of real applications of the GMS showing that the theory presented in this thesis is congruent and also applicable. The experiments show that the U-curve phenomenon is present when we estimate the out-of-sample error of the best Boolean function in each model, so we could exploit this property using U-curve search algorithms. Moreover, we observe that the size of the sample influences the selection of the best model. This fact satisfies the property that the model selector picks a model with VC-dimension according to the size of the sample.

Finally, in Chapter 4, we presented a stochastic algorithm to solve the U-curve problem called *Stochastic U-curve algorithm*. We studied that this algorithm is suboptimal but it has stochastic properties to guarantee good results. This is, for an instance $\langle(\mathcal{S}, \subseteq), c\rangle$ of the U-curve problem and real numbers δ and ϵ, we obtain an element such that it is expected that

$$\Pr(error(\mathbb{U}\langle(\mathcal{S}, \subseteq), c\rangle) \leq \epsilon) \geq 1 - \delta.$$

Moreover, we have shown that the expected computational complexity of the Stochastic U-curve algorithm is $O(\frac{n^2 \Pr(\mathcal{M})}{\epsilon} \ln \frac{1}{\delta})$ where n is the logarithm of the number of elements in the lattice and $\Pr(\mathcal{M})$ is the fraction of elements in $\mathcal{S}$ that are local minima.

Overall, we believe that the contributions of this thesis show some learning properties that were not explored before. We expect that future works in this direction can increase even more the knowledge and understanding of the design of model selectors for learning Boolean functions.

5.1 Future Work

In this section, we present an idea that emerged from this work. Although the presented GMS seems promising, there are still some Boolean learning models that cannot be represented by our current model. Thereby, we show how we could use ensemble techniques to improve the representation of the GMS.

As presented in Section 2.3.5, ensemble methods provide well-established techniques for combining machine learning hypotheses. Thereby, we show how the GMS can be used as part of an ensemble method for improving its performance. Figure 5.1 illustrates an ensemble method built from several hypotheses obtained from different model selectors. For instance, we present in Example 5.1.1 how the random forest model can be seen as an instance of an ensemble of generic model selectors.

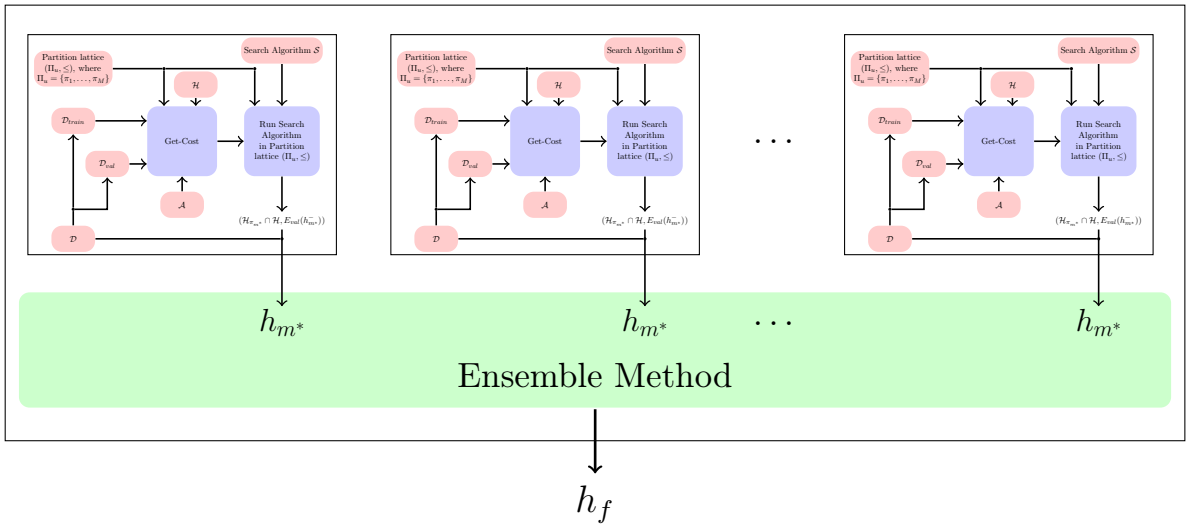


Figure 5.1: *Ensemble method using the Generic model selector.*

Example 5.1.1. In this example, we present the random forest model as an instance of the ensemble of generic model selectors using Example 3.3.2 for learning individual

hypotheses in the ensemble. So we can use individual model selectors to generate singular BDCDTs hypotheses with the following input arguments:

- For all model selectors, the selected hypothesis set $\mathcal{H}$ for the model is the set of all the Boolean functions with n variables.
- For all model selectors, the selected partition poset $(\Pi_u, \leq)$ is generated by the BDCDT family of n Boolean variables.
- The learning algorithm $\mathcal{A}$ works as follows: $\mathcal{A}$ picks the function that minimizes the in-sample error in the model induced by the partition.
- Each model selector uses an instance of the bagging process, i.e. each model selector uses a different training set consisting of a sample of N training elements drawn randomly with replacement from the original training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \{0,1\}^n, y_i \in \{0,1\}, 0 \leq i \leq N$.
- For all model selectors, the selected search algorithm in the lattice $(\Pi_u, \leq)$ is the same algorithm described by the generic model selector in Example 3.3.2. The unique difference is that we use a modified BDCDT learning with automatic criteria, that is, we force that the chosen Boolean variable as best split in each node will be selected from a random subset of Boolean variables.
- Each hypothesis obtained from individual model selectors has equal weight in the ensemble vote.

Hence, we observe that the ensemble of generic model selectors can be used successfully for design complex models such as the Random forest model.

Therefore, we observe that GMS along with ensemble methods can generate new families of models unlike only the GMS could generate. Nevertheless, we do not completely understand the resulting models from applying ensemble methods to a set of models generated by partitions. This subject seems an interesting topic to be studied in the future.

Bibliography

Abu-Mostafa *et al.***(2012)** Yaser S. Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin. *Learning From Data*. AMLBook. ISBN 1600490069, 9781600490064. cited in page 1, 19, 20, 21, 22, 23, 24, 25, 35, 36, 54

Amit *et al.***(1997)** Yali Amit, Donald Geman and Kenneth Wilder. Joint induction of shape features and tree classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(11): 1300–1305. ISSN 0162-8828. doi: 10.1109/34.632990. URL https://doi.org/10.1109/34.632990. cited in page 33

Anthony(2005) Martin Anthony. Learning boolean functions. cited in page 1

Anthony and Biggs(1992) Martin Anthony and Norman Biggs. *Computational Learning Theory: An Introduction*. Cambridge University Press, New York, NY, USA. ISBN 0-521-41603-5. cited in page 1, 22, 25, 34, 60

Atashpaz-Gargari *et al.***(2013)** E. Atashpaz-Gargari, U. M. Braga-Neto and E. R. Dougherty. Improved branch-and-bound algorithm for u-curve optimization. In *2013 IEEE International Workshop on Genomic Signal Processing and Statistics*, pages 100–101. doi: 10.1109/GENSIPS.2013.6735948. cited in page 2, 34, 50, 59

Atashpaz-Gargari *et al.***(2018)** Esmaeil Atashpaz-Gargari, Marcelo S. Reis, Ulisses M. Braga-Neto, Junior Barrera and Edward R. Dougherty. A fast branch-and-bound algorithm for u-curve feature selection. *Pattern Recognition*, 73:172 – 188. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.08.013. URL http://www.sciencedirect.com/science/article/pii/S0031320317303254. cited in page 2, 34, 50, 59

Barrera *et al.***(1997)** Junior Barrera, Edward R. Dougherty and Nina S. Tomita. Automatic programming of binary morphological machines by design of statistically optimal operators in the context of computational learning theory. *Journal of Electronic Imaging*, 6(1):54–67. cited in page 1, 27, 28

Box(1976) George E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799. doi: 10.1080/01621459.1976.10480949. cited in page 1

Breiman *et al.***(1984)** L. Breiman, J. Friedman, R. Olshen and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA. cited in page 30, 31

Breiman(1996) Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140. ISSN 1573-0565. doi: 10.1023/A:1018054314350. URL https://doi.org/10.1023/A:1018054314350. cited in page 33

Breiman(2001) Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324. cited in page 32, 33

Chandrashekar and Sahin(2014) Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16 – 28. ISSN 0045-7906. doi: https://doi.org/10.1016/j.compeleceng.2013.11.024. URL http://www.sciencedirect.com/science/article/pii/S0045790613003066. 40th-year commemorative issue. cited in page 33, 46

Cortes and Vapnik(1995) Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297. ISSN 1573-0565. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018. cited in page 28

Crama and Hammer(2011) Y. Crama and P.L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press. ISBN 9780521847513. URL https://books.google.com.br/books?id=5f8GnwEACAAJ. cited in page 8, 15, 16

DeGroot and Schervish(2013) M.H. DeGroot and M.J. Schervish. *Probability and Statistics*. Pearson custom library. Pearson Education. ISBN 9781292025049. URL https://books.google.com.br/books?id=hIPkngEACAAJ. cited in page 17, 59, 63

Devroye *et al.***(1996)** Luc Devroye, László Györfi and Gábor Lugosi. *A Probablistic Theory of Pattern Recognition*, volume 31. cited in page 22

Dietterich(2000) Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN 978-3-540-45014-6. cited in page 31, 32

Dougherty and Loce(1994) Edward R. Dougherty and Robert P. Loce. Precision of morphological-representation estimators for translation-invariant binary filters: Increasing and nonincreasing. *Signal Process.*, 40(2-3):129–154. cited in page 27

Dougherty *et al.***(2001)** Edward R. Dougherty, Junior Barrera, Gerard Mozelle, Seungchan Kim and Marcel Brun. Multiresolution analysis for optimal binary filters. *Journal of Mathematical Imaging and Vision*. cited in page 1, 9, 28, 29, 30

Freund and Schapire(1997) Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X. cited in page 32

Gini(1912) C. Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche. [/.]*. Studi economico-giuridici pubblicati per cura della facoltà di Giurisprudenza della R. Università di Cagliari. Tipogr. di P. Cuppini. URL https://books.google.com.br/books?id=fqjaBPMxB9kC. cited in page 30

Grätzer and Davey(2003) G. Grätzer and B.A. Davey. *General Lattice Theory*. Springer. ISBN 9783764369965. cited in page 5, 6, 7

Hastie *et al.***(2009)** Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/. cited in page 1, 2

Haykin(1998) Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition. ISBN 0132733501. cited in page 28

Hedmark(2017) D. Hedmark. *The Partition Lattice in Many Guises*. University of Kentucky Libraries. URL https://books.google.com.br/books?id=D9qxtAEACAAJ. cited in page 7

Heijmans(1994) H.J.A.M. Heijmans. *Morphological image operators*. Number v. 25 in Advances in electronics and electron physics: Supplement. Academic Press. ISBN 9780120145997. cited in page 17

Hirata(2009) N. S. T. Hirata. Multilevel training of binary morphological operators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):707–720. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.118. cited in page 1, 28

Hirata(2011) Nina S. T. Hirata. *Morphological Operator Design from Training Data*, pages 31–58. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-17934-1. doi: 10.1007/978-3-642-17934-1_3. URL https://doi.org/10.1007/978-3-642-17934-1_3. cited in page 16

Hirata Junior *et al.***(2002)** Roberto Hirata Junior, Marcel Brun, Junior Barrera and Edward R. Dougherty. Multiresolution design of aperture operators. *Journal of Mathematical Imaging and Vision*, 16(3):199–222. ISSN 09249907. doi: 10.1023/A:1020377610141. cited in page 1, 9, 28

Kim(2001) Hae Yong Kim. Binary operator design by k-nearest neighbor learning with application to image resolution increasing. *International Journal of Imaging Systems and Technology*, 11(5):331–339. doi: 10.1002/ima.1017. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ima.1017. cited in page 1, 28

Louppe(2014) G. Louppe. Understanding Random Forests: From Theory to Practice. *ArXiv e-prints*. cited in page 13, 30, 31

Martins *et al.***(2006)** David C. Martins, Roberto M. Cesar and Junior Barrera. W-operator window design by minimization of mean conditional entropy. *Pattern Analysis and Applications*, 9(2-3):139–153. ISSN 14337541. doi: 10.1007/s10044-006-0031-0. cited in page 1, 2, 28

Matheron(1975) G. Georges Matheron. *Random sets and integral geometry*. Wiley series in probability and mathematical statistics. Wiley, New York, London. cited in page 16

McCullagh and Nelder(1989) P. McCullagh and J.A. Nelder. *Generalized Linear Models, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis. ISBN 9780412317606. URL https://books.google.com.br/books?id=h9kFH2_FfBkC. cited in page 33

Montagner *et al.***(2017)** Igor S. Montagner, Nina S.T. Hirata and Roberto Hirata. Staff removal using image operator learning. *Pattern Recognition*, 63:310 – 320. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.10.002. URL http://www.sciencedirect.com/science/article/pii/S0031320316303181. cited in page 1, 28

Murphy(2012) Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press. ISBN 0262018020, 9780262018029. cited in page 1, 19

Natarajan(1989) B. K. Natarajan. On learning sets and functions. *Machine Learning*, 4(1):67–97. ISSN 1573-0565. doi: 10.1007/BF00114804. URL https://doi.org/10.1007/BF00114804. cited in page 1

Pudil *et al.***(1994)** P. Pudil, J. Novovičová and J. Kittler. Floating search methods in feature selection. *Pattern Recogn. Lett.*, 15(11):1119–1125. ISSN 0167-8655. doi: 10.1016/0167-8655(94)90127-9. URL https://doi.org/10.1016/0167-8655(94)90127-9. cited in page 59, 67

Quinlan(1986) J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106. ISSN 0885-6125. doi: 10.1023/A:1022643204877. URL http://dx.doi.org/10.1023/A:1022643204877. cited in page 28

Quinlan and Rivest(1989) J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description lenght principle. *Information and Computation*, 80(3):227 – 248. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(89)90010-2. URL http://www.sciencedirect.com/science/article/pii/0890540189900102. cited in page 30

Raschka(2015) Sebastian Raschka. *Python Machine Learning*. Packt Publishing. ISBN 1783555130, 9781783555130. cited in page 25

Reis(2013) Marcelo da Silva Reis. *Minimização de funções decomponíveis em curvas em U definidas sobre cadeias de posets algoritmos e aplicações*. PhD Thesis, "Universidade de São Paulo". cited in page 2, 34, 50, 59, 62, 67

Reis *et al.***(2017)** Marcelo S. Reis, Gustavo Estrela, Carlos Eduardo Ferreira and Junior Barrera. featsel: A framework for benchmarking of feature selection algorithms and cost functions. *SoftwareX*, 6:193 – 197. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2017.07.005. URL http://www.sciencedirect.com/science/article/pii/S2352711017300286. cited in page 34, 67

Ris *et al.***(2010)** Marcelo Ris, Junior Barrera and David C. Martins Jr. U-curve: A branch-and-bound optimization algorithm for u-shaped cost functions on boolean lattices applied to the feature selection problem. *Pattern Recognition*, 43(3):557 – 568. cited in page 2, 33, 34, 50, 59

Santos *et al.***(2010)** Carlos S. Santos, Nina S.T. Hirata and Roberto Hirata. An information theory framework for two-stage binary image operator design. *Pattern Recognition Letters*, 31(4):297 – 306. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2009.03.019. URL http://www.sciencedirect.com/science/article/pii/S0167865509001354. 20th SIBGRAPI: Advances in Image Processing and Computer Vision. cited in page 1, 28

Serra(1982) Jean Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press. cited in page 16

Sima and Dougherty(2008) Chao Sima and Edward R. Dougherty. The peaking phenomenon in the presence of feature-selection. *Pattern Recognition Letters*, 29(11):1667–1674. ISSN 01678655. cited in page 2

Valiant(1984) L. G. Valiant. Deductive learning. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 312 (1522):441–446. ISSN 0080-4614. doi: 10.1098/rsta.1984.0069. URL http://rsta.royalsocietypublishing.org/content/312/1522/441. cited in page 26

Whitney(1971) A. W. Whitney. A direct method of nonparametric measurement selection. *IEEE Trans. Comput.*, 20(9):1100–1103. ISSN 0018-9340. doi: 10.1109/T-C.1971.223410. URL http://dx.doi.org/10.1109/T-C.1971.223410. cited in page 59, 67

Zabinsky *et al.***(1995)** Z. B. Zabinsky, G. R. Wood, M. A. Steel and W. P. Baritompa. Pure adaptive search for finite global optimization. *Mathematical Programming*, 69 (1):443–448. ISSN 1436-4646. doi: 10.1007/BF01585570. URL http://dx.doi.org/10.1007/BF01585570. cited in page 65