Studies on Computational Learning via Discretization

Mahito Sugiyama

Doctoral dissertation, 2012

Department of Intelligence Science and Technology
Graduate School of Informatics
Kyoto University

A doctoral dissertation

submitted in partial fulfillment of the requirements
for the degree of Doctor of Informatics.

Department of Intelligence Science and Technology,
Graduate School of Informatics,
Kyoto University.

Typeset with
XƎTEX, Version 3.1415926-2.3-0.9997.5 (TeX Live 2011),
XY-pic, Version 3.8.5.

Abstract

This thesis presents cutting-edge studies on computational learning. The key issue throughout the thesis is amalgamation of two processes; *discretization* of continuous objects and *learning* from such objects provided by data.

Machine learning, or data mining and knowledge discovery, has been rapidly developed in recent years and is now becoming a huge topic in not only research communities but also businesses and industries. Discretization is essential for learning from continuous objects such as real-valued data, since every datum obtained by observation in the real world must be discretized and converted from analog (continuous) to digital (discrete) form to store in databases and manipulate on computers. However, most machine learning methods do not pay attention to the process: they use digital data in actual applications whereas assume analog data (usually real vectors) in theories. To bridge the gap, we cut into computational aspects of learning from theory to practice through three parts in this thesis.

Part I addresses theoretical analysis, which forms a disciplined foundation of the thesis. In particular, we analyze learning of *figures*, nonempty compact sets in Euclidean space, based on the *Gold-style learning model* aiming at a computational basis for binary classification of continuous data. We use *fractals* as a representation system, and reveal a learnability hierarchy under various learning criteria in the track of traditional analysis of learnability in the Gold-style learning model. We show a mathematical connection between machine learning and fractal geometry by measuring the complexity of learning using the *Hausdorff dimension* and the *VC dimension*. Moreover, we analyze computability aspects of learning of figures using the framework of Type-2 Theory of Effectivity (TTE).

Part II is a way from theory to practice. We start from designing a new measure in a computational manner, called *coding divergence*, which measures the difference between two sets of data, and go further by solving the typical machine learning tasks: classification and clustering. Specifically, we give two novel clustering algorithms, COOL (COding Oriented cLustering) and BOOL (Binary cOding Oriented cLustering). Experiments show that BOOL is faster than the *K*-means algorithm, and about two to three orders of magnitude faster than two state-of-the-art algorithms that can detect non-convex clusters of arbitrary shapes.

Part III treats more complex problems: *semi-supervised* and *preference* learning, by benefiting from *Formal Concept Analysis* (FCA). First we construct a SELF (SEmi-supervised Learning via FCA) algorithm, which performs classification and label ranking of *mixed-type data* containing both discrete and continuous variables. Finally, we investigate a biological application; we challenge to find *ligand* candidates of receptors from databases by formalizing the problem as *multi-label classification*, and develop an algorithm LIFT (Ligand FInding via Formal ConcepT Analysis) for the task. We experimentally show their competitive performance.

Acknowledgments

I am deeply grateful to all the people who have supported me along the way. First of all, I would like to sincerely thank to my supervisor Prof. Akihiro Yamamoto, who is my thesis committee chair. His comments and suggestions had inestimable value for my study. I would also like to thank to the other committee members, Prof. Tatsuya Akutsu and Prof. Toshiyuki Tanaka, for reviewing this thesis and for their meticulous comments.

Special thanks to my co-authors Prof. Hideki Tsuiki and Prof. Eiju Hirowatari, who have been greatly tolerant and supportive and gave insightful comments and suggestions. I am also indebted to Mr. Kentaro Imajo, Mr. Keisuke Otaki, and Mr. Tadashi Yoshioka, who are also my co-authors and colleagues in our laboratory.

My deepest appreciation goes to Prof. Shigeo Kobayashi who was my supervisor during my Master's course. It has been a unique chance for me to learn from his biological experience and his never-ending passion of scientific discovery.

I have had the support and encouragement of Prof. Takashi Washio, Prof. Shinichi Minato, Dr. Yoshinobu Kawahara, and Dr. Matthew de Brecht. I would like to express my gratitude to Dr. Marco Cuturi for his constant support in the English language throughout this thesis.

I would like to warmly thank all of the people who helped or encouraged me in various ways during my doctoral course: Dr. Koichiro Doi, Dr. Ryo Yoshinaka, and my colleagues in our laboratory.

Apart from individuals, I gratefully appreciate the financial support of the Japan Society for the Promotion of Science (JSPS) and Japan Student Services Organization that made it possible to complete my thesis.

Finally, I would like to thank to my family: my mother Hiroko, my father-in-law Masayuki Fujidai, and my wife Yumiko. In particular, Yumiko's support was indispensable to complete my doctoral course.

My father and my mother-in-law passed away during my doctoral course. I would like to devote this thesis to them.

Contents

List of Figures

List of Tables

List of Algorithms & Procedures

1

INTRODUCTION

LET US IMAGINE measuring the size of a cell. One of the most straightforward ways is to use a microscope equipped with a pair of micrometers; an *ocular micrometer* and a *stage micrometer*. The ocular micrometer is a glass disk with a ruled scale (like a ruler) located at a microscope eyepiece, which is used to measure the size of magnified objects. The stage micrometer is used for calibration, because the actual length of the marks on the scale of the ocular micrometer is determined by the degree of magnification. Here we consider only four objectives whose magnification is 1×, 2×, 4×, and 8×, for simplicity, and do not consider magnification of the eyepiece.

micrometer

Figure 1.1 shows an example of measurement of a cell. Let the length of the cell be x and marks represent 1 μm in length without any magnification. We obtain

$$2\ \mu\mathrm{m} \le x \le 3\ \mu\mathrm{m}$$

if we use the objective with 1× magnification. We call the width $3 - 2 = 1\ \mu$m the *error* of measurement. This is a very rough value, but the result can be *refined*, that is, the error can be reduced if we use a high-power objective. Then, we have

error

$$\begin{aligned} 2\ \mu\mathrm{m} \le x \le 2.5\ \mu\mathrm{m} &\qquad (2\times), \\ 2.25\ \mu\mathrm{m} \le x \le 2.5\ \mu\mathrm{m} &\qquad (4\times),\ \text{and} \\ 2.25\ \mu\mathrm{m} \le x \le 2.375\ \mu\mathrm{m} &\qquad (8\times), \end{aligned}$$

and errors are 0.5 μm, 0.25 μm, and 0.125 μm, respectively. Thus we can see that every *datum* in the real world obtained by a microscope has a numerical error which depends on the degree of magnification, and if we magnify $k\times$, the error becomes $k^{-1}\ \mu$m. This is not only the case for a microscope and is fundamental for every measurement: Any datum obtained by an experimental instrument, which is used for scientific activity such as proposing and testing a working hypothesis, must have some numerical error (cf. Baird, 1994).

datum

In the above discussion, we (implicitly) used a *real number* to represent the *true length x* of the cell, which is the standard way to treat objects in the real world mathematically. However, we cannot directly treat such real numbers on a *computer* — an infinite sequence is needed for exact encoding of a real number without

real number

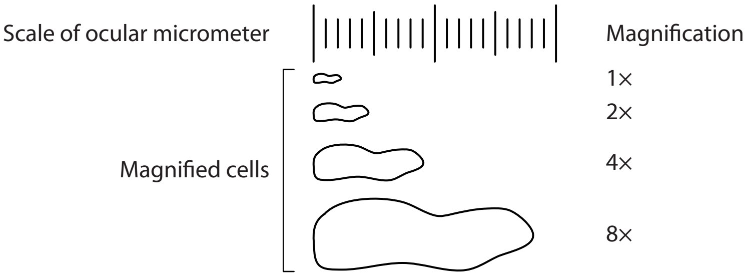


Figure 1.1 | Measurement of a cell by a microscope.

continuum

discretization

binary encoding

binary representation

any numerical error. This is why both of the cardinalities of the set of real numbers $\mathbb{R}$ and the set of infinite sequences Σ^{ω} are the *continuum*, whereas that of the set of finite sequences Σ^{*} is the same as $\aleph_0$, the cardinality of the set of natural numbers $\mathbb{N}$. Therefore, we cannot escape from *discretization* of real numbers to treat them on a computer in finite time.

In a typical computer, numbers are represented through the *binary encoding scheme*. For example, numbers 1, 2, 3, 4, 5, 6, 7, and 8 are represented as

$$\texttt{000}, \texttt{001}, \texttt{010}, \texttt{011}, \texttt{100}, \texttt{101}, \texttt{110}, \texttt{111},$$

respectively and, in the following, we focus on real numbers in $[0, 1]$ (the closed interval from 0 to 1) to go into the "real" world more deeply.

Mathematically, the *binary encoding*, or *binary representation*, of real numbers in $[0, 1]$ is realized as a surjective function ρ from Σ^{ω} to $\mathbb{R}$ with $\Sigma = \{0, 1\}$ such that

$$\rho(p) = \sum p_i \cdot 2^{-(i+1)}$$

for an infinite binary sequence $p = p_0 p_1 p_2 \dots$ (Figure 1.2). For instance,

$$\rho(\texttt{1000}\dots) = 0.5, \rho(\texttt{0100}\dots) = 0.25, \text{ and } \rho(\texttt{1111}\dots) = 1.$$

Thus, for an unknown real number x such that $x = \rho(p)$, if we observe the first bit p_0, we can determine to

$$p_0 \cdot 2^{-1} \le x \le p_0 \cdot 2^{-1} + 2^{-1}.$$

This means that this datum has an *error* $2^{-1} = 0.5$, which is the width of the interval. In the same way, if we observe the second, the third, and the fourth bits p_1, p_2, and p_3, we have

$$\sum_{i=0}^{1} p_i \cdot 2^{-(i+1)} \le x \le \sum_{i=0}^{1} p_i \cdot 2^{-(i+1)} + 2^{-2} \qquad (\text{for } p_0 p_1)\,,$$

$$\sum_{i=0}^{2} p_i \cdot 2^{-(i+1)} \le x \le \sum_{i=0}^{2} p_i \cdot 2^{-(i+1)} + 2^{-3} \qquad (\text{for } p_0 p_1 p_2)\,, \text{ and}$$

$$\sum_{i=0}^{3} p_i \cdot 2^{-(i+1)} \le x \le \sum_{i=0}^{3} p_i \cdot 2^{-(i+1)} + 2^{-4} \qquad (\text{for } p_0 p_1 p_2 p_3)\,.$$

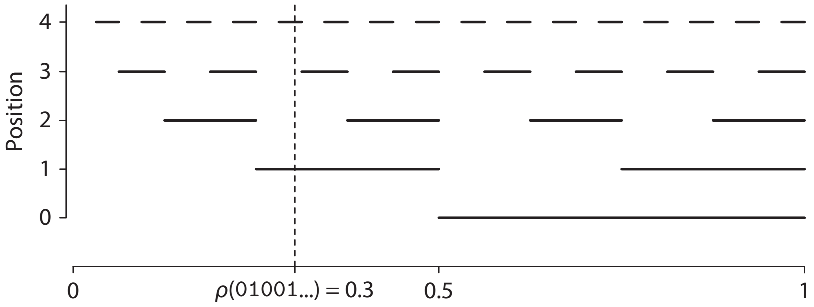


Figure 1.2 | Binary encoding of real numbers in [0, 1]. The position *i* is 1 if it is on the line, and 0 otherwise.

Thus a *prefix*, a *truncated finite binary sequence*, has a partial information about the true value x, which corresponds to a measured datum by a microscope. The error becomes $2^{-(k+1)}$ when we obtain the prefix whose length is k. Thus observing the successive bit corresponds to magnifying the object to double. In this way, we can *reduce* the error but, the important point is, we cannot know the exact *true value* of the object. In essentials, only such an *observable* information, specifically, a prefix of an infinite binary sequence encoding a real number, can be used on a computer, and all computational processings must be based on discrete structure manipulation on such approximate values.

Recently, computation for real numbers has been theoretically analyzed in the area of *computable analysis* (Weihrauch, 2000), where the framework of *Type-2 Theory of Effectivity* (TTE) has been introduced based on a *Type-2 machine*, which is an extended mathematical model of a *Turing machine*. This framework treats computation between infinite sequences; *i.e.*, treats manipulation for real numbers through their representations (infinite sequences). The key to realization of real number computation is to guarantee the computation between *streams* as follows: when a computer reads longer and longer prefixes of the input sequence, it produces longer and longer prefixes of the resulting sequence. Such procedure is called *effective computing*.

Type-2 Theory of Effectivity, TTE
Type-2 machine
Turing machine
stream
effective computing

Here we go to the central topic of the thesis: *machine learning*, which "is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data"[1]. Machine learning, including *data mining* and *knowledge discovery*, has been rapidly developed in recent years and is now becoming a huge topic in not only research communities but also businesses and industries.

Since the goal is to learn from empirical data obtained in the real world, basically, *discretization lies in any process in machine learning* for continuous objects. However, most machine learning methods do not pay attention to discretization as a principle for computation of real numbers. Although there are several discretization techniques (Elomaa and Rousu, 2003; Fayyad and Irani, 1993; Friedman *et al.*, 1998; Gama and Pinto, 2006; Kontkanen *et al.*, 1997; Lin *et al.*, 2003; Liu *et al.*, 2002; Skubacz and Hollmén, 2000), they treat discretization as just the data preprocessing for improving accuracy or efficiency, and the process discretization itself is not considered from computational point of view. Now, the mainstream in machine learning is an approach based on statistical data analysis techniques, so-called *statistical machine learning*, and they also (implicitly) use digital data in actual applications on computers whereas assume analog data (usually vectors of

[1] Reprinted from Wikipedia (`http://en.wikipedia.org/wiki/Machine_learning`)

real numbers) in theory. For example, methods originated from the perceptron are based on the idea of regulating analog wiring (Rosenblatt, 1958), hence they take no notice of discretization.

This gap is the motivation throughout this thesis. We cut into computational aspects of learning from theory to practice to bridge the gap. Roughly speaking, we build an "analog-to-digital (A/D) converter" into machine learning processes.

1.1 Main Contributions

This thesis consists of three parts. We list the main contributions for each part in the following with referring the publications by the author. We also summarize our contributions in Table 1.1 by categorizing them into learning types. See pp. 157 – 158 for the list of publications.

Part I: Theory

All results presented in this part have been published in [P1,P2].

Chapter 2: Learning Figures as Computable Classification

- We formalize learning of figures using fractals based on the Gold-style learning model towards fully computable binary classification (Section 2.2). We construct a representation system for learning using self-similar sets based on the binary representation of real numbers, and show desirable properties of it (Lemma 2.2, Lemma 2.3, and Lemma 2.4).
- We construct the learnability hierarchy under various learning criteria, summarized in Figure 2.3 (Section 2.3 and 2.4). We introduce four criteria for learning: explanatory learning (Subsection 2.3.1), consistent learning (Subsection 2.3.2), reliable and refutable learning (Subsection 2.3.3), and effective learning (Section 2.4).
- We show a mathematical connection between learning and fractal geometry by measuring the complexity of learning using the Hausdorff dimension and the VC dimension (Section 2.5). Specifically, we give the lower bound to the number of positive examples using the dimensions.
- We also show a connection between computability of figures and learnability of figures discussed in this chapter using TTE (Section 2.6). Learning can be viewed as computable realization of the identity from the set of figures to the same set equipped with the finer topology.

Part II: From Theory to Practice

All results presented in this part have been published in [B1,P3,P4,P6,P7,P8]. Chapter 3 is based on [B1,P3,P7], Chapter 4 on [P4,P6], and Chapter 5 on [P8].

Chapter 3: Coding Divergence

- We propose a measure of the difference between two sets of real-valued data, called *coding divergence*, to computationally unify two processes of discretization and learning (Definition 3.5).
- We construct a classifier using the divergence (Subsection 3.3.2), and experimentally illustrate its robust performance (Section 3.4).

Chapter 4: Minimum Code Length and Gray Code for Clustering

- We design a measure, called the *Minimum Code Length* (MCL), that can score the quality of a given clustering result under a *fixed* encoding scheme (Definition 4.1).
- We propose a general strategy to translate any encoding method into a cluster algorithm, called COOL (COding-Oriented cLustering) (Section 4.2). COOL has a low computational cost since it scales linearly with the data set size.
- We consider the *Gray Code* as the encoding scheme to present G-COOL (Section 4.3). G-COOL can find clusters of arbitrary shapes and remove noise.
- G-COOL is theoretically shown to achieve internal cohesion and external isolation and is experimentally shown to work well for both synthetic and real datasets (Section 4.4).

Chapter 5: Clustering Using Binary Discretization

- We present a new clustering algorithm, called BOOL (Binary cOding Oriented cLustering), for multivariate data using binary discretization (Section 5.1, 5.2). It can detect arbitrarily shaped clusters and is noise tolerant.
- Experiments show that BOOL is faster than the K-means algorithm, and about two to three orders of magnitude faster than two state-of-the-art algorithms that can detect non-convex clusters of arbitrary shapes (Section 5.3).
- We also show the robustness of BOOL to changes in parameters, whereas most algorithms for arbitrarily shaped clusters are known to be overly sensitive to such changes (Section 5.3).

Part III: With Formal Concept Analysis

All results presented in this part have been published in [J1,J2,P5,C1]. Chapter 6 is based on [J1,P5] and Chapter 7 on [J2,C1].

Chapter 6: Semi-supervised Classification and Ranking

- We present a new semi-supervised learning algorithm, called SELF (SEmi-supervised Learning via FCA), which performs multiclass classification and label ranking of *mixed-type data* containing both discrete and continuous variables (Section 6.2). SELF uses *closed set lattices*, which have been recently used for frequent pattern mining within the framework of the data analysis technique of *Formal Concept Analysis* (FCA).
- SELF can *weight* each classification rule using the lattice, which gives a partial order of preference over class labels (Section 6.2).
- We experimentally demonstrate competitive performance of SELF in classification and ranking compared to other learning algorithms (Section 6.3).

Table 1.1 | Contributions.

Supervised Learning	
Chapter 2	Theoretical Analysis of Learning Figures LLLL 2009 [P1], ALT 2010 [P2]
Chapter 3	Coding Divergence: Measuring the Similarity between Two Sets Book [B1], ACML 2010 [P3], ALSIP 2011 [P7]
Unsupervised Learning	
Chapter 4	(G-)COOL: Clustering with the MCL and the Gray Code LLLL 2011 [P4], ECML PKDD 2011 [P6]
Chapter 5	BOOL: Clustering Using Binary Discretization ICDM 2011 [P8]
Semi-supervised Learning	
Chapter 6	SELF: Semi-supervised Learning via FCA ICCS 2011 [P5], IDA [J1]
Chapter 7	LIFT: Ligand Finding via FCA ILP 2011 [C1], IPSJ TOM [J2]

Chapter 7: Ligand Finding by Multi-label Classification

- We mathematically model the problem of ligand finding, which is a crucial problem in biology and biochemistry, as *multi-label classification*.
- We develop a new algorithm LIFT (Ligand FInding via Formal ConcepT Analysis) for multi-label classification, which can treat ligand data in databases in the *semi-supervised* manner.
- We experimentally show that LIFT effectively solves our task compared to other machine learning algorithms using real data of ligands and receptors in the IUPHAR database.

Part I

Theory

“The symbol is defined as a set of points in this square, viz.
the set occupied by printer’s ink.”

— Alan Mathison Turing, *On Computable Numbers, with the Application to the Entscheidungsproblem*

2

LEARNING FIGURES AS COMPUTABLE CLASSIFICATION

discretization

DISCRETIZATION is a fundamental process in machine learning from analog data. For example, Fourier analysis is one of the most essential signal processing methods and its discrete version, *discrete Fourier analysis*, is used for learning or recognition on a computer from continuous signals. However, in the method, only the direction of the time axis is discretized, so each data point is not purely discretized. That is to say, continuous (electrical) waves are essentially treated as finite/infinite sequences of *real numbers*, hence each value is still continuous (analog). The gap between analog and digital data therefore remains.

This problem appears all over machine learning from observed multivariate data as mentioned in Introduction. The reason is that an infinite sequence is needed to encode a real vector exactly without any numerical error, since the cardinality of the set of real numbers, which is the same as that of infinite sequences, is much larger than that of the set of finite sequences. Thus to treat each data point on a computer, it has to be *discretized* and considered as an approximate value with some numerical error. However, to date, most machine learning algorithms ignore the gap between the original value and its discretized representation. This gap could result in some unexpected numerical errors[1]. Since now machine learning algorithms can be applied to massive datasets, it is urgent to give a theoretical foundation for learning, such as classification, regression, and clustering, from multivariate data, in a fully computational manner to guarantee the soundness of the results of learning.

Valiant-style learning model

In the field of computational learning theory, the *Valiant-style learning model* (also called *PAC, Probably Approximately Correct, learning model*), proposed by Valiant (1984), is used for theoretical analysis of machine learning algorithms. In this model, we can analyze the robustness of a learning algorithm in the face of noise or inaccurate data and the complexity of learning with respect to the rate of convergence or the size of the input using the concept of probability. Blumer *et al.* (1989) and Ehrenfeucht *et al.* (1989) provided the crucial conditions for

[1] Müller (2001) and Schröder (2002b) give some interesting examples in the study of computation for real numbers.

learnability, that is, the lower and upper bounds for the sample size, using the *VC* (*Vapnik-Chervonenkis*) *dimension* (Vapnik and Chervonenkis, 1971). These results can be applied to targets for continuous values, *e.g.*, the learning of neural networks (Baum and Haussler, 1989). However, this learning model does not fit to discrete and computational analysis of machine learning. We cannot know which class of continuous objects is exactly learnable and what kind of data are needed to learn from a finite expression of discretized multivariate data. Although Valiant-style learning from axis-parallel rectangles have already been investigated by Long and Tan (1998), which can be viewed as a variant of learning from multivariate data with numerical error, they are not applicable in the study since our goal is to investigate computational learning focusing on a common ground between "learning" and "computation" of real numbers based on the behavior of Turing machine without any probability distribution, and we need to distinguish abstract mathematical objects such as real numbers and their concrete representations, or codes, on a computer.

VC dimension

Instead, in this chapter we use the *Gold-style learning model* (also called *identification in the limit*), which is originally designed for learning of recursive functions (Gold, 1965) and languages (Gold, 1967). In the model, a learning machine is assumed to be a procedure, *i.e.*, a Turing machine (Turing, 1937) which never halts, that receives training data from time to time, and outputs representations (hypotheses) of the target from time to time. All data are usually assumed to be given in time. Starting from this learning model, learnability of classes of discrete objects, such as languages and recursive functions, has been analyzed in detail under various learning criteria (Jain *et al.*, 1999b). However, analysis of learning for continuous objects, such as classification, regression, and clustering for multivariate data, with the Gold-style learning model is still under development, despite such settings being typical in modern machine learning. To the best of our knowledge, the only line of studies by Hirowatari and Arikawa (1997); Apsītis *et al.* (1999); Hirowatari and Arikawa (2001); Hirowatari *et al.* (2003, 2005, 2006) devoted to learning of real-valued functions, where they addressed the analysis of learnable classes of real-valued functions using computable representations of real numbers. We therefore need a new theoretical and computational framework for modern machine learning based on the Gold-style learning model with discretization of numerical data.

Gold-style learning model

In this chapter we consider the problem of *binary classification* for multivariate data, which is one of the most fundamental problems in machine learning and pattern recognition. In this task, a training dataset consists of a set of pairs

binary classification

$$\{ (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \},$$

where $x_i \in \mathbb{R}^d$ is a *feature vector*, $y_i \in \{0, 1\}$ is a *label*, and the d-dimensional Euclidean space $\mathbb{R}^d$ is a *feature space*. The goal is to learn a *classifier* from the given training dataset, that is, to find a mapping $h : \mathbb{R}^d \to \{0, 1\}$ such that, for all $x \in \mathbb{R}^d$, $h(x)$ is expected to be the same as the true label of x. In other words, such a classifier h is the *characteristic function* of a subset

feature vector

classifier

characteristic function

$$L = \left\{ x \in \mathbb{R}^d \;\middle|\; h(x) = 1 \right\}$$

of $\mathbb{R}^d$, which has to be similar to the true set

$$K = \left\{ x \in \mathbb{R}^d \;\middle|\; \text{the true label of } x \text{ is } 1 \right\}$$

as far as possible. Throughout the chapter, we assume that each feature is normalized by some data preprocessing such as min-max normalization for simplicity, that is, the feature space is the unit interval (cube) $\mathcal{I}^d = [0,1] \times \ldots \times [0,1]$ in the d-dimensional Euclidean space $\mathbb{R}^d$. In many realistic scenarios, each target K is a closed and bounded subset of $\mathcal{I}^d$, *i.e.*, a nonempty compact subset of $\mathcal{I}^d$, called a *figure*. Thus here we address the problem of binary classification by treating it as "learning of figures".

figure

In this machine learning process, we implicitly treat any feature vector through its *representation*, or *code* on a computer, that is, each feature vector $x \in \mathcal{I}^d$ is represented by a sequence p over some alphabet Σ using an encoding scheme ρ. Here such a surjective mapping ρ is called a *representation* and should map the set of "infinite" sequences Σ^ω to $\mathcal{I}^d$ since there is no one-to-one correspondence between finite sequences and real numbers (or real vectors). In this chapter, we use the *binary representation* $\rho : \Sigma^\omega \to [0,1]$ with $\Sigma = \{0,1\}$, which is defined by

representation

binary representation

$$\rho(p) := \sum p_i \cdot 2^{-(i+1)}$$

for an infinite sequence $p = p_0 p_1 p_2 \ldots$. For example,

$$\rho(\mathtt{0100}\ldots) = 0.25,\ \rho(\mathtt{1000}\ldots) = 0.5,\ \text{and}\ \rho(\mathtt{0111}\ldots) = 0.5.$$

However, we cannot treat infinite sequences on a computer in finite time and, instead, we have to use *discretized* values, *i.e.*, *truncated finite sequences* in any actual machine learning process. Thus in learning of a classifier h for the target figure K, we cannot use an exact data point $x \in K$ but have to use a discretized finite sequence $w \in \Sigma^*$ which tells us that x takes one of the values in the set $\{\rho(p) \mid w \sqsubset p\}$ ($w \sqsubset p$ means that w is a *prefix* of p). For instance, if $w = \mathtt{01}$, then x should be in the interval $[0.25, 0.5]$. For a finite sequence $w \in \Sigma^*$, we define

$$\rho(w) := \left\{\, \rho(p) \;\middle|\; w \sqsubset p \text{ with } p \in \Sigma^\omega \,\right\}$$

using the same symbol ρ. From a geometric point of view, $\rho(w)$ means a hyper rectangle whose sides are parallel to the axes in the space $\mathcal{I}^d$. For example, for the binary representation ρ, we have

$$\rho(\mathtt{0}) = [0, 0.5],\ \rho(\mathtt{1}) = [0.5, 1],\ \rho(\mathtt{01}) = [0.25, 0.5],$$

and so on. Therefore in the actual learning process, while a target set K and each point $x \in K$ exist mathematically, a learning machine can only treat finite sequences as training data.

Here the problem of binary classification is stated in a computational manner as follows: given a training dataset

$$\left\{\, (w_1, y_1), (w_2, y_2), \ldots, (w_n, y_n) \,\right\}$$

($w_i \in \Sigma^*$ for each $i \in \{1, 2, \ldots, n\}$), where

$$y_i = \begin{cases} 1 & \text{if } \rho(w_i) \cap K \neq \emptyset \text{ for a target figure } K \subseteq \mathcal{I}^k \\ 0 & \text{otherwise,} \end{cases}$$

learn a classifier $h : \Sigma^* \to \{0,1\}$ for which $h(w)$ should be the same as the true label of w. Each training datum (w_i, y_i) is called a *positive example* if $y_i = 1$ and a

positive example

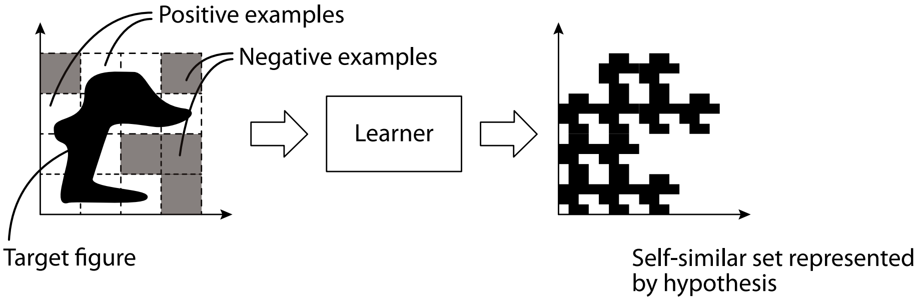


Figure 2.1 | Framework of learning figures.

negative example if $y_i = 0$.

negative example

Assume that a figure K is represented by a set P of infinite sequences, *i.e.*, $\{\rho(p) \mid p \in P\} = K$, using the binary representation ρ. Then learning the figure is different from learning the well-known *prefix closed set* $\mathrm{Pref}(P)$, defined as

prefix closed set

$$\mathrm{Pref}(P) := \left\{\, w \in \Sigma^* \;\middle|\; w \sqsubset p \text{ for some } p \in P \,\right\},$$

since generally $\mathrm{Pref}(P) \neq \{w \in \Sigma^* \mid \rho(w) \cap K \neq \emptyset\}$ holds. For example, if $P = \{p \in \Sigma^\omega \mid 1 \sqsubset p\}$, the corresponding figure K is the interval $[0.5, 1]$. The infinite sequence $0111\ldots$ is a positive example since $\rho(0111\ldots) = 0.5$ and $\rho(0111\ldots) \cap K \neq \emptyset$, but it is not contained in $\mathrm{Pref}(P)$. Solving this mismatch between objects of learning and their representations is one of the challenging problems of learning continuous objects based on their representation in a computational manner.

For finite expression of classifiers, we use *self-similar sets* known as a family of *fractals* (Mandelbrot, 1982) to exploit their simplicity and the power of expression theoretically provided by the field of fractal geometry. Specifically, we can approximate any figure by some self-similar set arbitrarily closely (derived from the Collage Theorem given by Falconer (2003)) and can compute it by a simple recursive algorithm, called an *IFS* (*Iterated Function System*) (Barnsley, 1993; Falconer, 2003). This approach can be viewed as the analog of the discrete Fourier analysis, where *FFT* (*Fast Fourier Transformation*) is used as the fundamental recursive algorithm. Moreover, in the process of sampling from analog data in discrete Fourier analysis, *scalability* is a desirable property. It requires that when the sample resolution increases, the accuracy of the result is monotonically refined. We formalize this property as *effective learning* of figures, which is inspired by *effective computing* in the framework of Type-2 Theory of Effectivity (TTE) studied in computable analysis (Schröder, 2002a; Weihrauch, 2000). This model guarantees that as a computer reads more and more precise information of the input, it produces more and more accurate approximations of the result. Here we interpret this model from computation to learning, where if a learner (learning machine) receives more and more accurate training data, it learns better and better classifiers (self-similar sets) approximating the target figure.

self-similar set

fractals

IFS (Iterated Function System)

To summarize, our framework of learning figures (shown in Figure 2.1) is as follows: Positive examples are axis-parallel rectangles intersecting the target figure, and negative examples are those disjoint with the target. A learner reads a presentation (infinite sequence of examples), and generates hypotheses. A hypothesis is a finite set of finite sequences (codes), which is a discrete expression of a self-similar set. To evaluate "goodness" of each classifier, we use the concept of *generalization error* and measure it by the *Hausdorff metric* since it induces the standard topology on the set of figures (Beer, 1993).

Hausdorff metric

The rest of the chapter is organized as follows: We review related work in comparison to the present work in Section 2.1. We formalize computable binary classification as learning of figures in Section 2.2 and analyze the learnability hierarchy induced by variants of our model in Section 2.3 and Section 2.4. The mathematical connection between fractal geometry and the Gold-style learning model with the Hausdorff and the VC dimensions is presented in Section 2.5 and between computability and learnability of figures in Section 2.6. Section 2.7 gives the summary of this chapter.

2.1 Related Work

Statistical approaches to machine learning are achieving great success (Bishop, 2007) since they are originally designed for analyzing observed multivariate data and, to date, many statistical methods have been proposed to treat continuous objects such as real-valued functions. However, most methods pay no attention to discretization and the finite representation of analog data on a computer. For example, multi-layer perceptrons are used to learn real-valued functions, since they can approximate every continuous function arbitrarily and accurately. However, a perceptron is based on the idea of regulating analog wiring (Rosenblatt, 1958), hence such learning is not purely computable, *i.e.*, it ignores the gap between analog raw data and digital discretized data. Furthermore, although several discretization techniques have been proposed (Elomaa and Rousu, 2003; Fayyad and Irani, 1993; Gama and Pinto, 2006; Kontkanen *et al.*, 1997; Li *et al.*, 2003; Lin *et al.*, 2003; Liu *et al.*, 2002; Skubacz and Hollmén, 2000), they treat discretization as data preprocessing for improving the accuracy or efficiency of machine learning algorithms. The process of discretization is not therefore considered from a computational point of view, and “computability” of machine learning algorithms is not discussed at sufficient depth.

There are several related work considering learning under various restrictions in the Gold-style learning model (Goldman *et al.*, 2003), the Valiant-style learning model (Ben-David and Dichterman, 1998; Decatur and Gennaro, 1995), and other learning context (Khardon and Roth, 1999). Moreover, recently learning from partial examples, or examples with missing information, has attracted much attention in the Valiant-style learning model (Michael, 2010, 2011). In this chapter we also consider learning from examples with missing information, which are truncated finite sequences. However, our model is different from them, since the “missing information” in this chapter corresponds to *measurement error* of real-valued data. As mentioned in Introduction (Chapter 1), our motivation comes from actual measurement/observation of a physical object, where every datum obtained by an experimental instrument must have some numerical error in principle (Baird, 1994). For example, if we measure the size of a cell by a microscope equipped with micrometers, we cannot know the true value of the size but an approximate value with numerical error, which depends on the degree of magnification by the micrometers. In this chapter we try to treat this process as learning from multivariate data, where an approximate value corresponds to a truncated finite sequence and error becomes small as the length of the sequence increases, intuitively. The asymmetry property of positive and negative examples is naturally derived from the motivation. The model of computation for real numbers within the framework of TTE fits to our motivation, which is unique in computational learning theory.

Self-similar sets can be viewed as a geometric interpretation of languages recognized by ω-*automata* (Perrin and Pin, 2004), first introduced by Büchi (1960), and learning of such languages has been investigated by De La Higuera and Janodet (2001); Jain *et al.* (2011). Both works focus on learning ω-languages from their prefixes, *i.e.* texts (positive data), and show several learnable classes. This approach is different from ours since our motivation is to address computability issues in the field of machine learning from numerical data, and hence there is a gap between prefixes of ω-languages and positive data for learning in our setting. Moreover, we consider learning from both positive and negative data, which is a new approach in the context of learning of infinite words.

ω-automaton

To treat values with numerical errors on computers, various effective methods have been proposed in the research area of numerical computation with result verification (Oishi, 2008). Originally, they also used an interval as a representation of an approximate value and, recently, some efficient techniques with floating-point numbers have been presented (Ogita *et al.*, 2005). While they focus on *computation* with numerical errors, we try to embed the concept of errors into *learning* based on the computation schema of TTE using interval representation of real numbers. Considering relationship between our model and such methods discussed in numerical computation with result verification and constructing efficient algorithms using the methods is an interesting future work.

2.2 Formalization of Learning

To analyze binary classification in a computable approach, we first formalize learning of figures based on the Gold-style learning model. Specifically, we define targets of learning, representations of classifiers produced by a learning machine, and a protocol for learning. In the following, let $\mathbb{N}$ be the set of natural numbers including 0, $\mathbb{Q}$ the set of rational numbers, and $\mathbb{R}$ the set of real numbers. The set $\mathbb{N}^+$ (resp. $\mathbb{R}^+$) is the set of positive natural (resp. real) numbers. The d-fold product of $\mathbb{R}$ is denoted by $\mathbb{R}^d$ and the set of nonempty compact subsets of $\mathbb{R}^d$ is denoted by $\mathcal{K}^*$.

Throughout this chapter, we use the *binary representation* $\rho^d : (\Sigma^d)^\omega \to \mathcal{I}^d$ as the canonical representation for real numbers. If $d = 1$, this is defined as follows: $\Sigma = \{0, 1\}$ and

binary representation

$$\rho^1(p) \coloneqq \sum_{i=0}^{\infty} p_i \cdot 2^{-(i+1)} \tag{2.1}$$

for an infinite sequence $p = p_0 p_1 p_2 \dots$. Note that Σ^d denotes the set $\{a_1 a_2 \dots a_d \mid a_i \in \Sigma\}$ and $\Sigma^1 = \Sigma$. For example, $\rho^1(0100\dots) = 0.25$, $\rho^1(1000\dots) = 0.5$, and so on. Moreover, by using the same symbol ρ, we introduce a representation $\rho^1 : \Sigma^* \to \mathcal{K}^*$ for finite sequences defined as follows:

$$\begin{aligned}\rho^1(w) \coloneqq \rho^1(\uparrow w) &= [\,\rho(w000\dots), \rho(w111\dots)\,] \\ &= \left[\sum w_i \cdot 2^{-(i+1)}, \sum w_i \cdot 2^{-(i+1)} + 2^{|w|}\right],\end{aligned} \tag{2.2}$$

where $\uparrow w = \{\, p \in \Sigma^\omega \mid w \sqsubset p \,\}$. For instance, $\rho^1(01) = [0.25, 0.5]$ and $\rho^1(10) = [0.5, 0.75]$.

In a d-dimensional space with $d > 1$, we use the *d-dimensional binary representation* $\rho^d : (\Sigma^d)^\omega \to \mathcal{I}^d$ defined in the following manner.

$$\rho^d(\langle p^1, p^2, \dots, p^d \rangle) := \left(\rho^1(p^1), \rho^1(p^2), \dots, \rho^1(p^d)\right), \tag{2.3}$$

where d infinite sequences p^1, p^2, …, and p^d are concatenated using the *tupling function* $\langle\cdot\rangle$ such that

$$\langle p^1, p^2, \dots, p^d \rangle := p_0^1 p_0^2 \dots p_0^d p_1^1 p_1^2 \dots p_1^d p_2^1 p_2^2 \dots p_2^d \dots . \tag{2.4}$$

Similarly, we define a representation $\rho^d : (\Sigma^d)^* \to \mathcal{K}^*$ by

$$\rho^d(\langle w^1, w^2, \dots, w^d \rangle) := \rho^d(\uparrow\langle w^1, w^2, \dots, w^d \rangle),$$

where

$$\langle w^1, w^2, \dots, w^d \rangle := w_0^1 w_0^2 \dots w_0^d w_1^1 w_1^2 \dots w_1^d \dots w_n^1 w_n^2 \dots w_n^d.$$

with $|w^1| = |w^2| = \dots = |w^d| = n$. Note that, for any $w = \langle w^1, \dots, w^d \rangle \in (\Sigma^d)^*$, $|w^1| = |w^2| = \dots = |w^d|$ always holds, and we denote the length by $|w|$ in this chapter. For a set of finite sequences, *i.e.*, a *language* $L \subset \Sigma^*$, we define

language

$$\rho^d(L) := \left\{ \rho^d(w) \mid w \in L \right\}.$$

We omit the superscript d of ρ^d if it is understood from the context.

A target set of learning is a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$ fixed *a priori*, and one of them is chosen as a target in each learning term. A learning machine uses *self-similar sets*, known as fractals and defined by finite sets of contractions. This approach is one of the key ideas in this chapter. Here, a *contraction* is a mapping $\mathrm{CT} : \mathbb{R}^d \to \mathbb{R}^d$ such that, for all $x, y \in X$, $d(\mathrm{CT}(x), \mathrm{CT}(y)) \le cd(x, y)$ for some real number c with $0 < c < 1$. For a finite set of contractions C, a nonempty compact set F satisfying

contraction

$$F = \bigcup_{\mathrm{CT} \in C} \mathrm{CT}(F)$$

is determined uniquely (see the book (Falconer, 2003) for its formal proof). The set F is called the *self-similar set* of C. Moreover, if we define a mapping $\mathbf{CT} : \mathcal{K}^* \to \mathcal{K}^*$ by

self-similar set

$$\mathbf{CT}(K) := \bigcup_{\mathrm{CT} \in C} \mathrm{CT}(K) \tag{2.5}$$

and define

$$\mathbf{CT}^0(K) := K \text{ and } \mathbf{CT}^{k+1}(K) := \mathbf{CT}(\mathbf{CT}^k(K)) \tag{2.6}$$

for each $k \in \mathbb{N}$ recursively, then

$$F = \bigcap_{k=0}^{\infty} \mathbf{CT}^k(K)$$

for every $K \in \mathcal{K}^*$ such that $\mathrm{CT}(K) \subset K$ for every $\mathrm{CT} \in C$. This means that we have a level-wise construction algorithm with **CT** to obtain the self-similar set F.

Actually, a learning machine produces *hypotheses*, each of which is a finite language and becomes a finite expression of a self-similar set that works as a classifier. Formally, for a finite language $H \subset (\Sigma^d)^*$, we consider $H^0, H^1, H^2, \dots$ such that H^k is recursively defined as follows:

$$\begin{cases} H^0 := \{\lambda\}, \\ H^k := \left\{ \langle\, w^1u^1, w^2u^2, \dots, w^du^d \,\rangle \;\middle|\; \begin{array}{l} \langle\, w^1, w^2, \dots, w^d \,\rangle \in H^{k-1} \text{ and} \\ \langle\, u^1, u^2, \dots, u^d \,\rangle \in H \end{array} \right\}. \end{cases}$$

We can easily construct a fixed program which generates $H^0, H^1, H^2, \dots$ when receiving a hypothesis H. We give the semantics of a hypothesis H by the following equation:

$$\kappa(H) := \bigcap_{k=0}^{\infty} \bigcup \rho(H^k). \tag{2.7}$$

Since $\bigcup \rho(H^k) \supset \bigcup \rho(H^{k+1})$ holds for all $k \in \mathbb{N}$, $\kappa(H) = \lim_{k\to\infty} \bigcup \rho(H^k)$. We denote the set of hypotheses $\{H \subset (\Sigma^d)^* \mid H \text{ is finite}\}$ by $\mathcal{H}$ and call it the *hypothesis space*. We use this hypothesis space throughout the chapter. Note that, for a pair of hypotheses H and L, $H = L$ implies $\kappa(H) = \kappa(L)$, but the converse may not hold.

hypothesis space

Example 2.1
Assume $d = 2$ and let a hypothesis H be the set $\{\langle\, 0, 0 \,\rangle, \langle\, 0, 1 \,\rangle, \langle\, 1, 1 \,\rangle\} = \{00, 01, 11\}$. We have

$$\begin{aligned} &H^0 = \emptyset, \quad H^1 = \{\langle\, 0, 0 \,\rangle, \langle\, 0, 1 \,\rangle, \langle\, 1, 1 \,\rangle\} = \{00, 01, 11\}, \\ &H^2 = \left\{ \begin{array}{l} \langle\, 00, 00 \,\rangle, \langle\, 00, 01 \,\rangle, \langle\, 01, 01 \,\rangle, \langle\, 00, 10 \,\rangle, \langle\, 00, 11 \,\rangle, \\ \langle\, 01, 11 \,\rangle, \langle\, 10, 10 \,\rangle, \langle\, 10, 11 \,\rangle, \langle\, 11, 11 \,\rangle \end{array} \right\} \\ &\quad\;\; = \{0000, 0001, 0011, 0100, 0101, 0111, 1100, 1101, 1111\}, \dots \end{aligned}$$

and the figure $\kappa(H)$ defined in the equation (2.7) is the *Sierpiński triangle* (Figure 2.2). If we consider the following three mappings:

Sierpiński triangle

$$\begin{aligned} \mathrm{CT}_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\ \mathrm{CT}_2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}, \\ \mathrm{CT}_3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}, \end{aligned}$$

the three squares $\mathrm{CT}_1(\mathcal{I}^d)$, $\mathrm{CT}_2(\mathcal{I}^d)$, and $\mathrm{CT}_3(\mathcal{I}^d)$ are exactly the same as $\rho(00)$, $\rho(01)$, and $\rho(11)$, respectively. Thus each sequence in a hypothesis can be viewed as a representation of one of these squares, which are called *generators* for a self-similar set since if we have the initial set $\mathcal{I}^d$ and generators $\mathrm{CT}_1(\mathcal{I}^d)$, $\mathrm{CT}_2(\mathcal{I}^d)$, and $\mathrm{CT}_3(\mathcal{I}^d)$, we can reproduce the three mappings CT_1, CT_2, and CT_3 and construct the self-similar set from them. Note that there exist infinitely many hypotheses L such that $\kappa(H) = \kappa(L)$ and $H \neq L$. For example, $L = \{\langle\, 0, 0 \,\rangle, \langle\, 1, 1 \,\rangle, \langle\, 00, 10 \,\rangle, \langle\, 00, 11 \,\rangle, \langle\, 01, 11 \,\rangle\}$. □

generator

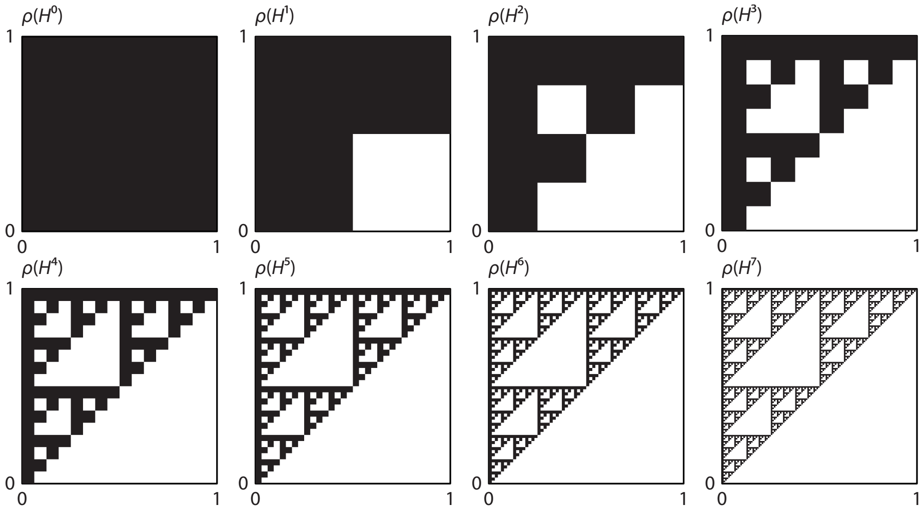


Figure 2.2 | Generation of the Sierpiński triangle from the hypothesis $H = \{\langle 0,1\rangle, \langle 0,1\rangle, \langle 1,1\rangle\}$ (Example 2.1).

Lemma 2.2: Soundness of hypotheses
For every hypothesis $H \in \mathcal{H}$, the set $\kappa(H)$ defined by the equation (2.7) is a self-similar set.

Proof. Let $H = \{w_1, w_2, \dots, w_n\}$. We can easily check that the set of rectangles $\rho(w_1), \dots, \rho(w_n)$ is a generator defined by the mappings $\mathrm{CT}_1, \dots, \mathrm{CT}_n$, where each CT_i maps the unit interval $\mathcal{I}^d$ to the figure $\rho(w_i)$. Define $\mathbf{CT}$ and $\mathbf{CT}^k$ in the same way as the equations (2.5) and (2.6). For each $k \in \mathbb{N}$,

$$\bigcup \rho(H^k) = \mathbf{CT}^k(\mathcal{I}^d)$$

holds. It follows that the set $\kappa(H)$ is exactly the same as the self-similar set defined by the mappings $\mathrm{CT}_1, \mathrm{CT}_2, \dots, \mathrm{CT}_n$, that is, $\kappa(H) = \bigcup \mathrm{CT}_i(\kappa(H))$ holds. □

generalization error

Hausdorff metric

To evaluate the “goodness” of each hypothesis, we use the concept of *generalization error*, which is usually used to score the quality of hypotheses in a machine learning context. The generalization error of a hypothesis H for a target figure K, written by $\mathrm{GE}(K, H)$, is defined by the *Hausdorff metric* d_{H} on the space of figures,

$$\mathrm{GE}(K, H) := d_{\mathrm{H}}(K, \kappa(H)) = \inf\left\{ \delta \;\middle|\; K \subseteq \kappa(H)_\delta \text{ and } \kappa(H) \subseteq K_\delta \right\},$$

δ-neighborhood

where K_δ is the *δ-neighborhood* of K defined by

$$K_\delta := \left\{ x \in \mathbb{R}^d \;\middle|\; d_{\mathrm{E}}(x, a) \le \delta \text{ for some } a \in K \right\}.$$

The metric d_{E} is the Euclidean metric such that

$$d_{\mathrm{E}}(x, a) = \sqrt{\sum_{i=1}^{d} (x^i - a^i)^2}$$

for $x = (x^1, \dots, x^d), a = (a^1, \dots, a^d) \in \mathbb{R}^d$. The Hausdorff metric is one of the standard metrics on the space since the metric space $(\mathcal{K}^*, d_{\mathrm{H}})$ is complete (in the sense of topology) and $\mathrm{GE}(K, H) = 0$ if and only if $K = \kappa(H)$ Beer (1993); Kechris (1995). The topology on $\mathcal{K}^*$ induced by the Hausdorff metric is called the *Vietoris topology*. Since the cardinality of the set of hypotheses $\mathcal{H}$ is smaller than that of the set of figures $\mathcal{K}^*$, we often cannot find the exact hypothesis H for a figure K such that $\mathrm{GE}(K, H) = 0$. However, following the Collage Theorem given by Falconer (2003), we show that the power of representation of hypotheses is still sufficient, that is, we always can approximate a given figure arbitrarily closely by some hypothesis.

Vietoris topology

> **Lemma 2.3: Representational power of hypotheses**
> *For any $\delta \in \mathbb{R}$ and for every figure $K \in \mathcal{K}^*$, there exists a hypothesis H such that $\mathrm{GE}(K, H) < \delta$.*

Proof. Fix a figure K and the parameter δ. Here we denote the diameter of the set $\rho(w)$ with $|w| = k$ by $\mathrm{diam}(k)$. Then we have

$$\mathrm{diam}(k) = \sqrt{d} \cdot 2^{-k}.$$

For example, $\mathrm{diam}(1) = 1/2$ and $\mathrm{diam}(2) = 1/4$ if $d = 1$, and $\mathrm{diam}(1) = 1/\sqrt{2}$ and $\mathrm{diam}(2) = 1/\sqrt{8}$ if $d = 2$. For k with $\mathrm{diam}(k) < \delta$, let

$$H = \left\{ w \in (\Sigma^d)^* \;\middle|\; |w| = k \text{ and } \rho(w) \cap K \neq \emptyset \right\}.$$

We can easily check that the $\mathrm{diam}(k)$-neighborhood of the figure K contains $\kappa(H)$ and $\mathrm{diam}(k)$-neighborhood of $\kappa(H)$ contains K. Thus we have $\mathrm{GE}(K, H) < \delta$. □

There are many other representation systems that meet the following condition. One of remarkable features of our system with self-similar sets will be shown in Lemma 2.37. Moreover, to work as a classifier, every hypothesis H has to be *computable*, that is, the function $h : (\Sigma^d)^* \to \{0, 1\}$ such that, for all $w \in (\Sigma^d)^*$,

computable

$$h(w) = \begin{cases} 1 & \text{if } \rho(w) \cap \kappa(H) \neq \emptyset, \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

should be computable. We say that such h is the *classifier* of H. The computability of h is not trivial, since for a finite sequence w, the two conditions $h(w) = 1$ and $w \in H^k$ are not equivalent. Intuitively, this is because each interval represented by a finite sequence is *closed*. For example, in the case of Example 2.1, $h(\mathtt{10}) = 1$ because $\rho(\mathtt{10}) = [0.5, 1] \times [0, 0.5]$ and $\rho(\mathtt{10}) \cap \kappa(H) = \{(0.5, 0.5)\} \neq \emptyset$ whereas $\mathtt{10} \notin H^k$ for any $k \in \mathbb{N}$. Here we guarantee this property of computability.

classifier

closed

> **Lemma 2.4: Computability of classifiers**
> *For every hypothesis $H \in \mathcal{H}$, the classifier h of H defined by the equation* (2.8) *is computable.*

Proof. First we consider whether or not the boundary of an interval is contained in $\kappa(H)$. Suppose $d = 1$ and let C be a finite set of contractions and F be the self-similar set of C. We have the following property: For every interval $[x, y] = \mathrm{CT}_1 \circ \mathrm{CT}_2 \circ \ldots \circ \mathrm{CT}_n(\mathcal{I}^1)$ such that $\mathrm{CT}_i \in C$ for all $i \in \{1, \ldots, n\}$ ($n \in \mathbb{N}$), we have $x \in F$ (resp. $y \in F$) if and only if $0 \in \mathrm{CT}(\mathcal{I}^1)$ (resp. $1 \in \mathrm{CT}(\mathcal{I}^1)$) for some $\mathrm{CT} \in C$. This means that if $[x, y] = \rho(v)$ with a sequence $v \in H^k$ ($k \in \mathbb{N}$) for a hypothesis

Algorithm 2.1: Classifier h of hypothesis H

```
Input: Finite sequence $w$ and hypothesis $H$
Output: Class label 1 or 0 of $w$
$k \leftarrow 0$
repeat
    $k \leftarrow k + 1$
until $\min_{v \in H^k} |v| > |w|$
for each $v \in H^k$
    if $w \sqsubseteq v$ then
        output 1 and halt
    else if CheckBoundary$(w, v, H) = 1$ then
        output 1 and halt
    end if
end for
output 0

function CheckBoundary$(w, v, H)$
$a \leftarrow a^1 a^2 \dots a^d$    // $a$ is a finite sequence whose length is $d$
for each $a^s$ in $\{a^1, a^2, \dots, a^d\}$
    if $w^s \sqsubseteq v^s$ then $a^s \leftarrow \perp$
    else
        if $w^s \sqsubseteq \underline{v^s}$ then $a^s \leftarrow 0$
        else if $w^s \sqsubseteq \overline{v^s}$ then $a^s \leftarrow 1$
        else return 0
        end if
    end if
end for
for each $u \in H$
    if $u = aaa \dots a$ then return 1
end for
return 0
```

H, we have $x \in \kappa(H)$ (resp. $y \in \kappa(H)$) if and only if $u \in \{0\}^+$ (resp. $u \in \{1\}^+$) for some $u \in H$.

We show a pseudo-code of the classifier h in Algorithm 2.1 and prove that the output of the algorithm is 1 if and only if $h(w) = 1$, *i.e.*, $\rho(w) \cap \kappa(H) \neq \emptyset$. In the algorithm, $\underline{v^s}$ and $\overline{v^s}$ denote the previous and subsequent binary sequences of v^s with $|v^s| = |\underline{v^s}| = |\overline{v^s}|$ in the lexicographic order, respectively. For example, if $v^s = 001$, $\underline{v^s} = 000$ and $\overline{v^s} = 010$. Moreover, we use the special symbol $\perp$ meaning undefinedness, that is, $v = w$ if and only if $v_i = w_i$ for all $i \in \{0, 1, \dots, |v| - 1\}$ with $v_i \neq \perp$ and $w_i \neq \perp$.

The "if" part: For an input of a finite sequence w and a hypothesis H, if $h(w) = 1$, there are two possibilities as follows:

1. For some $k \in \mathbb{N}$, there exists $v \in H^k$ such that $w \sqsubseteq v$. This is because $\rho(w) \supseteq \rho(v)$ and $\rho(w) \cap \kappa(H) \neq \emptyset$.

2. The above condition does not hold, but $\rho(w) \cap \kappa(H) \neq \emptyset$.

In the first case, the algorithm goes to line 7 and stops with outputting 1. The second case means that the algorithm uses the function CheckBoundary. Since $h(w) = 1$, there should exist a sequence $v \in H$ such that $u = a$ for some $u \in H$, where a is obtained in lines 1–10. CheckBoundary therefore returns 1.

The “only if” part: In Algorithm 2.1, if $v \in H^k$ satisfies conditions in line 6 or line 8, $h(w) \cap \kappa(H) \neq \emptyset$. Thus $h(w) = 1$ holds. □

The set $\{\, \kappa(H) \mid H \subset (\Sigma^d)^* \text{ and the classifier } h \text{ of } H \text{ is computable} \,\}$ exactly corresponds to an *indexed family of recursive concepts / languages* discussed in computational learning theory (Angluin, 1980), which is a common assumption for learning of languages. On the other hand, there exists some class of figures $\mathcal{F} \subseteq \mathcal{K}^*$ that is not an indexed family of recursive concepts. This means that, for some figure K, there is no *computable* classifier which classifies all data correctly. Therefore we address the problems of both exact and approximate learning of figures to obtain a computable classifier for any target figure.

indexed family of recursive concepts

We consider two types of input data streams, one includes both positive and negative data and the other includes only positive data, to analyze learning based on the Gold-style learning model. Formally, each training datum is called an *example* and is defined as a pair (w, l) of a finite sequence $w \in (\Sigma^d)^*$ and a label $l \in \{0, 1\}$. For a target figure K, we define

example

$$l := \begin{cases} 1 & \text{if } \rho(w) \cap K \neq \emptyset \ \ (\textit{positive example}), \\ 0 & \text{otherwise} \ \ (\textit{negative example}). \end{cases}$$

positive example

negative example

In the following, for a target figure K, we denote the set of finite sequences of positive examples $\{w \in (\Sigma^d)^* \mid \rho(w) \cap K \neq \emptyset\}$ by $\mathrm{Pos}(K)$ and that of negative examples by $\mathrm{Neg}(K)$. From the geometric nature of figures, we obtain the following *monotonicity* of examples:

monotonicity

> **Lemma 2.5: Monotonicity of examples**
> *If $(v, 1)$ is an example of K, then $(w, 1)$ is an example of K for all prefixes $w \sqsubseteq v$, and $(va, 1)$ is an example of K for some $a \in \Sigma^d$. If $(w, 0)$ is an example of K, then $(wv, 0)$ is an example of K for all $v \in (\Sigma^d)^*$.*

Proof. From the definition of the representation ρ in the equations (2.1) and (2.3), if $w \sqsubseteq v$, we have $\rho(w) \supseteq \rho(v)$, hence $(w, 1)$ is an example of K. Moreover,

$$\bigcup_{a \in \Sigma^d} \rho(va) = \rho(u)$$

holds. Thus there should exist an example $(va, 1)$ for some $a \in \Sigma^d$. Furthermore, for all $v \in \Sigma^*$, $\rho(wv) \subset \rho(w)$. Therefore if $K \cap \rho(w) = \emptyset$, then $K \cap \rho(wv) = \emptyset$ for all $v \in (\Sigma^d)^*$, and $(wv, 0)$ is an example of K. □

We say that an infinite sequence σ of examples of a figure K is a *presentation* of K. The ith example is denoted by $\sigma(i - 1)$, and the set of all examples occurring in σ is denoted by $\mathrm{range}(\sigma)$[2]. The initial segment of σ of length n, *i.e.*, the sequence

presentation

[2] The reason for this notation is that σ can be viewed as a mapping from $\mathbb{N}$ (including 0) to the set of examples.

Table 2.1 | Relationship between the conditions for each finite sequence $w \in \Sigma^*$ and the standard notation of binary classification.

		Target figure K	
		$w \in \mathrm{Pos}(K)$ $(\rho(w) \cap K \neq \emptyset)$	$w \in \mathrm{Neg}(K)$ $(\rho(w) \cap K = \emptyset)$
Hypothesis H	$h(w) = 1$ $(\rho(w) \cap \kappa(H) \neq \emptyset)$	True positive	False positive (Type I error)
	$h(w) = 0$ $(\rho(w) \cap \kappa(H) = \emptyset)$	False negative (Type II error)	True negative

$\sigma(0), \sigma(1), \dots, \sigma(n-1)$, is denoted by $\sigma[n-1]$. A *text* of a figure K is a presentation σ such that

$$\{\, w \mid (w, 1) \in \mathrm{range}(\sigma) \,\} = \mathrm{Pos}(K) \quad (= \{\, w \mid \rho(w) \cap K \neq \emptyset \,\}),$$

informant

and an *informant* is a presentation σ such that

$$\{\, w \mid (w, 1) \in \mathrm{range}(\sigma) \,\} = \mathrm{Pos}(K) \text{ and}$$
$$\{\, w \mid (w, 0) \in \mathrm{range}(\sigma) \,\} = \mathrm{Neg}(K).$$

Table 2.1 shows the relationship between the standard terminology in classification and our definitions. For a target figure K and the classifier h of a hypothesis H, the set

$$\{\, w \in \mathrm{Pos}(K) \mid h(w) = 1 \,\}$$

true positive

corresponds to *true positive*,

$$\{\, w \in \mathrm{Neg}(K) \mid h(w) = 1 \,\}$$

false positive (type I error)

false positive (type I error),

$$\{\, w \in \mathrm{Pos}(K) \mid h(w) = 0 \,\}$$

false negative (type II error)

false negative (type II error), and

$$\{\, w \in \mathrm{Neg}(K) \mid h(w) = 0 \,\}$$

true negative

true negative.

consistent

Let h be the classifier of a hypothesis H. We say that the hypothesis H is *consistent* with an example (w, l) if $l = 1$ implies $h(w) = 1$ and $l = 0$ implies $h(w) = 0$, and consistent with a set of examples E if H is consistent with all examples in E.

learner

A learning machine, called a *learner*, is a procedure, (*i.e.* a Turing machine that never halts) that reads a presentation of a target figure from time to time, and outputs hypotheses from time to time. In the following, we denote a learner by $\mathbf{M}$ and an infinite sequence of hypotheses produced by $\mathbf{M}$ on the input σ by $\mathbf{M}_\sigma$, and $\mathbf{M}_\sigma(i-1)$ denotes the ith hypothesis produced by $\mathbf{M}$. Assume that $\mathbf{M}$ receives j examples $\sigma(0), \sigma(1), \dots, \sigma(j-1)$ so far when it outputs the ith hypothesis $\mathbf{M}_\sigma(i-1)$. We do not require the condition $i = j$, that is, the inequation $i \leq j$ usually holds since $\mathbf{M}$ can "wait" until it receives enough examples. We say that an infinite sequence of hypotheses $\mathbf{M}_\sigma$ *converges* to a hypothesis H if there exists $n \in \mathbb{N}$ such that $\mathbf{M}_\sigma(i) = H$ for all $i \geq n$.

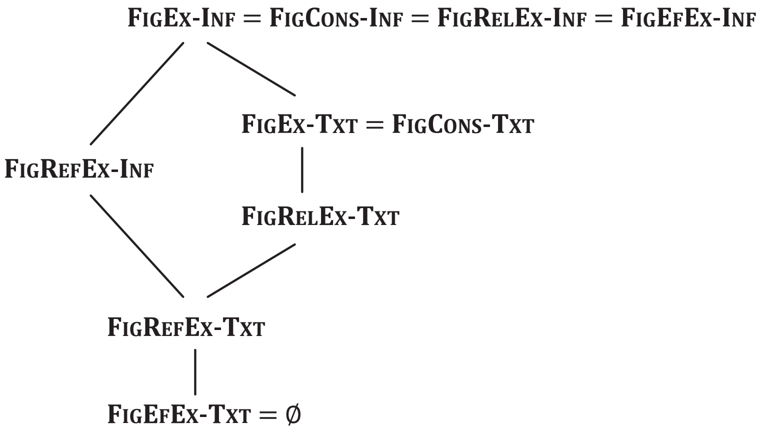


Figure 2.3 | Learnability hierarchy. In each line, the lower set is a proper subset of the upper set.

2.3 Exact Learning of Figures

We analyze "exact" learning of figures. This means that, for any target figure K, there should be a hypothesis H such that the generalization error is zero (*i.e.*, $K = \kappa(H)$), hence the classifier h of H can classify all data correctly with no error, that is, h satisfies the equation (2.8). The goal is to find such a hypothesis H from examples (training data) of K.

In the following two sections (Sections 2.3 and 2.4), we follow the standard path of studies in computational learning theory (Jain *et al.*, 1999b; Jain, 2011; Zeugmann and Zilles, 2008), that is, we define learning criteria to understand various learning situations and construct a learnability hierarchy under the criteria. We summarize our results in Figure 2.3.

2.3.1 Explanatory Learning

The most basic learning criterion in the Gold-style learning model is **Ex**-learning (EX means EXplain), or *learning in the limit*, proposed by Gold (1967). We call these criteria **FigEx-Inf**- (INF means an informant) and **FigEx-Txt**-learning (TXT means a text) for **Ex**-learning from informants and texts, respectively. We introduce these criteria into the learning of figures, and analyze the learnability.

learning in the limit

Definition 2.6: Explanatory learning
A learner **M** **FigEx-Inf**-*learns* (resp. **FigEx-Txt**-*learns*) a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$ if for all figures $K \in \mathcal{F}$ and all informants (resp. texts) σ of K, the outputs $\mathbf{M}_\sigma$ converge to a hypothesis H such that $\mathrm{GE}(K, H) = 0$.

FigEx-Inf-learning
FigEx-Txt-learning

For every learning criterion **CR** introduced in the following, we say that a set of figures $\mathcal{F}$ is **CR**-*learnable* if there exists a learner that **CR**-learns $\mathcal{F}$, and denote by **CR** the collection of **CR**-learnable sets of figures following the standard notation of this field (Jain *et al.*, 1999b).

First, we consider **FigEx-Inf**-learning. Informally, a learner can **FigEx-Inf**-learn a set of figures if it has an ability to enumerate all hypotheses and to judge whether or not each hypothesis is consistent with the received examples (Gold,

Procedure 2.2: Learning procedure that **FigEx-Inf**-learns $\kappa(\mathcal{H})$

```
Input: Informant $\sigma = (w_0, l_0), (w_1, l_1), \ldots$ of figure $K \in \kappa(\mathcal{H})$
Output: Infinite sequence of hypotheses $\mathbf{M}_\sigma(0), \mathbf{M}_\sigma(1), \ldots$
$i \leftarrow 0$
$E \leftarrow \emptyset$    // $E$ is a set of received examples
repeat
    read $\sigma(i)$ and add to $E$    // $\sigma(i) = (w_i, l_i)$
    search the first hypothesis $H$ consistent with $E$
      through a normal enumeration
    output $H$    // $\mathbf{M}_\sigma(i) = H$
    $i \leftarrow i + 1$
until forever
```

normal enumeration

1967). Here we introduce a convenient enumeration of hypotheses. An infinite sequence of hypotheses $H_0, H_1, \ldots$ is called a *normal enumeration* if $\{ H_i \mid i \in \mathbb{N} \} = \mathcal{H}$ and, for all $i, j \in \mathbb{N}$, $i < j$ implies

$$\max_{v \in H_i} |v| \le \max_{w \in H_j} |w|.$$

We can easily implement a procedure that enumerates $\mathcal{H}$ through a normal enumeration.

Theorem 2.7
The set of figures $\kappa(\mathcal{H}) = \{\kappa(H) \mid H \in \mathcal{H}\}$ is **FigEx-Inf***-learnable.*

Proof. This learning can be done by the well-known strategy of identification by enumeration. We show a pseudo-code of a learner **M** that **FigEx-Inf**-learns $\kappa(\mathcal{H})$ in Procedure 2.2. The learner **M** generates hypotheses through normal enumeration. If **M** outputs a wrong hypothesis H, there must exist a positive or negative example that is not consistent with the hypothesis since, for a target figure K_*,

$$\mathrm{Pos}(K_*) \ominus \mathrm{Pos}(\kappa(H)) \neq \emptyset$$

for every hypothesis H with $\kappa(H) \neq K_*$, where $X \ominus Y$ denotes the *symmetric difference*, *i.e.*, $X \ominus Y = (X \cup Y) \setminus (X \cap Y)$. Thus the learner **M** changes the wrong hypothesis and reaches to a correct hypothesis H_* such that $\kappa(H_*) = K_*$ in finite time. If **M** produces a correct hypothesis, it never changes it, since every example is consistent with it. Therefore **M FigEx-Inf**-learns $\kappa(\mathcal{H})$. □

Next, we consider **FigEx-Txt**-learning. In learning of languages from texts, the necessary and sufficient conditions for learning have been studied in detail (Angluin, 1980, 1982; Kobayashi, 1996; Lange *et al.*, 2008; Motoki *et al.*, 1991; Wright, 1989), and characterization of learnability using finite tell-tale sets is one of the crucial results. We interpret these results into the learning of figures and show the **FigEx-Txt**-learnability.

Definition 2.8: Finite tell-tale set (cf. Angluin, 1980)
Let $\mathcal{F}$ be a set of figures. For a figure $K \in \mathcal{F}$, a finite subset $\mathcal{T}$ of the set of positive examples $\mathrm{Pos}(K)$ is a *finite tell-tale set of K with respect to $\mathcal{F}$* if for all figures $L \in \mathcal{F}, \mathcal{T} \subset \mathrm{Pos}(L)$ implies $\mathrm{Pos}(L) \not\subset \mathrm{Pos}(K)$ (*i.e.*, $L \not\subset K$). If every $K \in \mathcal{F}$ has finite tell-tale sets with respect to $\mathcal{F}$, we say that $\mathcal{F}$ has a finite tell-tale set.

finite tell-tale set

Theorem 2.9
Let $\mathcal{F}$ be a subset of $\kappa(\mathcal{H})$. Then $\mathcal{F}$ is **FIGEX-TXT***-learnable if and only if there is a procedure that, for every figure $K \in \mathcal{F}$, enumerates a finite tell-tale set W of K with respect to $\mathcal{F}$.*

This theorem can be proved in exactly the same way as that for learning of languages (Angluin, 1980). Note that such procedure does not need to stop. Using this theorem, we show that the set $\kappa(\mathcal{H})$ is not **FIGEX-TXT**-learnable.

Theorem 2.10
The set $\kappa(\mathcal{H})$ does not have a finite tell-tale set.

Proof. Fix a figure $K = \kappa(H) \in \kappa(\mathcal{H})$ such that $\#H \geq 2$ and fix a finite set $T = \{w_1, w_2, \dots, w_n\}$ contained in $\mathrm{Pos}(K)$. For each finite sequence w_i, there exists $u_i \in \mathrm{Pos}(K)$ such that $w_i \sqsubset u_i$ with $w_i \neq u_i$. For the figure $L = \kappa(U)$ with $U = \{u_1, \dots, u_n\}$, $T \subset \mathrm{Pos}(L)$ and $\mathrm{Pos}(L) \subset \mathrm{Pos}(K)$ hold. Therefore K has no finite tell-tale set with respect to $\kappa(\mathcal{H})$. □

Corollary 2.11
The set of figures $\kappa(\mathcal{H})$ is not **FIGEX-TXT***-learnable.*

In any realistic situation of machine learning, however, this set $\kappa(\mathcal{H})$ is too large to search for the best hypothesis since we usually want to obtain a "compact" representation of a target figure. Thus we (implicitly) have an upper bound on the number of elements in a hypothesis. Here we give a fruitful result for the above situation, that is, if we fix the number of elements $\#H$ in each hypothesis H *a priori*, the resulting set of figures becomes **FIGEX-TXT**-learnable. Intuitively, this is because if we take k large enough, the set $\{w \in \mathrm{Pos}(K) \mid |w| \leq k\}$ becomes a finite tell-tale set of K. For a finite subset of natural numbers $N \subset \mathbb{N}$, we denote the set of hypotheses $\{H \in \mathcal{H} \mid \#H \in N\}$ by $\mathcal{H}_N$.

Theorem 2.12
There exists a procedure that, for all finite subsets $N \subset \mathbb{N}$ and all figures $K \in \kappa(\mathcal{H}_N)$, enumerates a finite tell-tale set of K with respect to $\kappa(\mathcal{H}_N)$.

Proof. First, we assume that $N = \{1\}$. It is trivial that there exists a procedure that, for an arbitrary figure $K \in \kappa(\mathcal{H}_N)$, enumerates a finite tell-tale set of K with respect to $\kappa(\mathcal{H}_N)$, since we always have $L \not\subset K$ for all pairs of figures $K, L \in \kappa(\mathcal{H}_N)$.

Next, fix a finite set $N \subset \mathbb{N}$ with $N \neq \{1\}$. Let us consider the procedure that enumerates elements of the sets

$$\mathrm{Pos}_1(K), \mathrm{Pos}_2(K), \mathrm{Pos}_3(K), \dots .$$

We show that this procedure enumerates a finite tell-tale set of K with respect to $\kappa(\mathcal{H}_N)$. Notice that the number of elements $\#\mathrm{Pos}_k(K)$ monotonically increases when k increases whenever $K \notin \kappa(\mathcal{H}_{\{1\}})$.

For each level k and for a figure $L \in \kappa(\mathcal{H})$,

$$L \subset K \text{ and } \operatorname{Pos}(L) \supseteq \bigcup_{i \in \{1,2,\dots,k\}} \operatorname{Pos}_i(K)$$

implies

$$\operatorname{Pos}(L) = \bigcup_{i \in \{1,2,\dots,k\}} \operatorname{Pos}_i(K). \tag{2.9}$$

Here we define the set

$$\mathcal{L}_k = \{\, L \in \kappa(\mathcal{H}_N) \mid L \subset K \text{ and } L \text{ satisfies the condition (2.9)} \,\}$$

for each level $k \in \mathbb{N}$. Then we can easily check that the minimum size of hypothesis $\min_{\kappa(H) \in \mathcal{L}_k} \#H$ monotonically increases as k increases. This means that there exists a natural number n such that $\mathcal{L}_k = \emptyset$ for every $k \geq n$, since for each hypothesis $H \in \mathcal{H}_N$ we must have $\#H \in N$. Therefore the set

$$\mathcal{T} = \bigcup_{i \in \{1,2,\dots,n\}} \operatorname{Pos}_i(K)$$

is a finite tell-tale set of K with respect to $\kappa(\mathcal{H}_N)$. □

Corollary 2.13
For all finite subsets of natural numbers $N \subset \mathbb{N}$, the set of figures $\kappa(\mathcal{H}_N)$ is **FIGEX-TXT**-*learnable.*

2.3.2 Consistent Learning

In a learning process, it is natural that every hypothesis generated by a learner is consistent with the examples received by it so far. Here we introduce **FIGCONS-INF**- and **FIGCONS-TXT**-learning (CONS means CONSistent). These criteria correspond to **CONS**-learning that was first introduced by Blum and Blum (1975)[3]. This model was also used (but implicitly) in the Model Inference System (MIS) proposed by Shapiro (1981, 1983), and studied in the computational learning of formal languages and recursive functions (Jain *et al.*, 1999b).

FIGCONS-INF-learning
FIGCONS-TXT-learning

Definition 2.14: Consistent learning
A learner **M** **FIGCONS-INF**-*learns* (resp. **FIGCONS-TXT**-*learns*) a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$ if **M** **FIGEX-INF**-learns (resp. **FIGEX-TXT**-learns) $\mathcal{F}$ and for all figures $K \in \mathcal{F}$ and all informants (resp. texts) σ of K, each hypothesis $\mathbf{M}_\sigma(i)$ is consistent with E_i that is the set of examples received by **M** until just before it generates the hypothesis $\mathbf{M}_\sigma(i)$.

Assume that a learner **M** achieves **FIGEX-INF**-learning of $\kappa(\mathcal{H})$ using Procedure 2.2. We can easily check that **M** always generates a hypothesis that is consistent with the received examples.

[3] Consistency was also studied in the same form by Barzdin (1974) in Russian.

Corollary 2.15
FigEx-Inf = **FigCons-Inf**.

Suppose that $\mathcal{F} \subset \kappa(\mathcal{H})$ is **FigEx-Txt**-learnable. We can construct a learner **M** in the same way as in the case of **Ex**-learning of languages from texts (Angluin, 1980), where **M** always outputs a hypothesis that is consistent with received examples.

Corollary 2.16
FigEx-Txt = **FigCons-Txt**.

2.3.3 Reliable and Refutable Learning

In this subsection, we consider target figures that might not be represented exactly by any hypothesis since there are infinitely many such figures, and if we have no background knowledge, there is no guarantee of the existence of an exact hypothesis. Thus in practice this approach is more convenient than the explanatory or consistent learning considered in the previous two subsections.

To realize the above case, we use two concepts, *reliability* and *refutability*. Reliable learning was introduced by Blum and Blum (1975) and Minicozzi (1976) and refutable learning by Mukouchi and Arikawa (1995) and Sakurai (1991) in computational learning of languages and recursive functions to introduce targets which cannot be exactly represented by any hypotheses, and developed in literatures (Jain *et al.*, 2001; Merkle and Stephan, 2003; Mukouchi and Sato, 2003). Here we introduce these concepts into the learning of figures and analyze learnability.

First, we treat reliable learning of figures. Intuitively, reliability requires that an infinite sequence of hypotheses only converges to a correct hypothesis.

Definition 2.17: Reliable learning
A learner **M FigRelEx-Inf**-*learns* (resp. **FigRelEx-Txt**-*learns*) a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$ if **M** satisfies the following conditions:

FigRelEx-Inf-learning
FigRelEx-Txt-learning

1. The learner **M FigEx-Inf**-learns (resp. **FigEx-Txt**-learns) $\mathcal{F}$.
2. For any target figure $K \in \mathcal{K}^*$ and its informants (resp. texts) σ, the infinite sequence of hypotheses $\mathbf{M}_\sigma$ does not converge to a wrong hypothesis H such that $\mathrm{GE}(K, \kappa(H)) \neq 0$.

We analyze reliable learning of figures from informants. Intuitively, if a learner can judge whether or not the current hypothesis H is consistent with the target figure, *i.e.*, $\kappa(H) = K$ or not in finite time, then the target figure is reliably learnable.

Theorem 2.18
FigEx-Inf = **FigRelEx-Inf**.

Proof. Since the statement **FigRelEx-Inf** $\subseteq$ **FigEx-Inf** is trivial, we prove the opposite **FigEx-Inf** $\subseteq$ **FigRelEx-Inf**. Fix a set of figures $\mathcal{F} \subseteq \kappa(\mathcal{H})$ with $\mathcal{F} \in$ **FigEx-Inf**, and suppose that a learner **M FigEx-Inf**-learns $\mathcal{F}$ using Procedure 2.2. The goal is to show that $\mathcal{F} \in$ **FigRelEx-Inf**. Assume that a target figure K belongs to $\mathcal{K}^* \setminus \mathcal{F}$. Here we have the following property: for all figures $L \in \mathcal{F}$, there must exist a finite sequences $w \in (\Sigma^d)^*$ such that

$$w \in \mathrm{Pos}(K) \ominus \mathrm{Pos}(L),$$

hence for any **M**'s current hypothesis H, **M** changes H if it receives a positive or negative example (w, l) such that $w \in \mathrm{Pos}(K) \ominus \mathrm{Pos}(\kappa(H))$. This means that an infinite sequence of hypotheses does not converge to any hypothesis. Thus we have $\mathcal{F} \in$ **FIGRELEX-INF**. □

In contrast, we have an interesting result in the reliable learning from texts. We show in the following that **FIGEX-TXT** $\neq$ **FIGRELEX-TXT** holds and that a set of figures $\mathcal{F}$ is reliably learnable from positive data only if any figure $K \in \mathcal{F}$ is a singleton. Remember that $\mathcal{H}_N$ denotes the set of hypotheses $\{H \in \mathcal{H} \mid \#H \in N\}$ for a subset $N \subset \mathbb{N}$ and, for simplicity, we denote $\mathcal{H}_{\{n\}}$ by $\mathcal{H}_n$ for $n \in \mathbb{N}$.

Theorem 2.19
The set of figures $\kappa(\mathcal{H}_N)$ is **FIGRELEX-TXT**-*learnable if and only if $N = \{1\}$.*

Proof. First we show that the set of figures $\kappa(\mathcal{H}_1)$ is **FIGRELEX-TXT**-learnable. From the self-similar sets property of hypotheses, we have the following: a figure $K \in \kappa(\mathcal{H})$ is a singleton if and only if $K \in \kappa(\mathcal{H}_1)$. Let $K \in \mathcal{K}^* \setminus \kappa(\mathcal{H}_1)$, and assume that a learner **M** **FIGEX-TXT**-learns $\kappa(\mathcal{H}_1)$. We can naturally suppose that **M** changes the current hypothesis H whenever it receives a positive example $(w, 1)$ such that $w \notin \mathrm{Pos}(\kappa(H))$ without loss of generality. For any hypothesis $H \in \mathcal{H}_1$, there exists $w \in (\Sigma^d)^*$ such that

$$w \in \mathrm{Pos}(K) \setminus \mathrm{Pos}(\kappa(H))$$

since K is not a singleton. Thus if the learner **M** receives such a positive example $(w, 1)$, it changes the hypothesis H. This means that an infinite sequence of hypotheses does not converge to any hypothesis. Therefore $\kappa(\mathcal{H}_1)$ is **FIGRELEX-TXT**-learnable.

Next, we prove that $\kappa(\mathcal{H}_n)$ is not **FIGRELEX-TXT**-learnable for any $n > 1$. Fix such $n \in \mathbb{N}$ with $n > 1$. We can easily check that, for a figure $K \in \kappa(\mathcal{H}_n)$ and any of its finite tell-tale sets $\mathcal{T}$ with respect to $\kappa(\mathcal{H}_n)$, there exists a figure $L \in \mathcal{K}^* \setminus \kappa(\mathcal{H}_n)$ such that $L \subset K$ and $\mathcal{T} \subset \mathrm{Pos}(L)$. This means that

$$\mathrm{Pos}(L) \subseteq \mathrm{Pos}(K) \text{ and } \mathcal{T} \subseteq \mathrm{Pos}(L)$$

hold. Thus if a learner **M** **FIGEX-TXT**-learns $\kappa(\mathcal{H}_n)$, $\mathbf{M}_\sigma$ for some presentation σ of some such L must converge to some hypothesis in $\mathcal{H}_n$. Consequently, we have $\kappa(\mathcal{H}_n) \notin$ **FIGRELEX-TXT**. □

Corollary 2.20
FIGRELEX-TXT $\subset$ **FIGEX-TXT**.

Sakurai (1991) proved that a set of concepts $\mathcal{C}$ is reliably **EX**-learnable from texts if and only if $\mathcal{C}$ contains no infinite concept (p. 182, Theorem 3.1)[4]. However, we have shown that the set $\kappa(\mathcal{H}_1)$ is **FIGRELEX-TXT**-learnable, though all figures $K \in \kappa(\mathcal{H}_1)$ correspond to infinite concepts since $\mathrm{Pos}(K)$ is infinite for all $K \in \kappa(\mathcal{H}_1)$. The monotonicity of the set $\mathrm{Pos}(K)$ (Lemma 2.5), which is a constraint naturally derived from the geometric property of examples, causes this difference.

Next, we extend **FIGEX-INF**- and **FIGEX-TXT**-learning by paying our attention to *refutability*. In refutable learning, a learner tries to learn figures in the limit, but it

[4] The literature (Sakurai, 1991) was written in Japanese. The same theorem was mentioned by Mukouchi and Arikawa (1995, p. 60, Theorem 3).

understands that it cannot find a correct hypothesis in finite time, that is, outputs the refutation symbol $\triangle$ and stops if the target figure is not in the considered space.

Definition 2.21: Refutable learning

FigRefEx-Inf-learning

FigRefEx-Txt-learning

A learner $\mathbf{M}$ **FigRefEx-Inf**-*learns* (resp. **FigRefEx-Txt**-*learns*) a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$ if $\mathbf{M}$ satisfies the followings. Here, $\triangle$ denotes the *refutation symbol*.

1. The learner $\mathbf{M}$ **FigEx-Inf**-learns (resp. **FigEx-Txt**-learns) $\mathcal{F}$.
2. If $K \in \mathcal{F}$, then for all informants (resp. texts) σ of K, $\mathbf{M}_\sigma(i) \neq \triangle$ for all $i \in \mathbb{N}$.
3. If $K \in \mathcal{K}^* \setminus \mathcal{F}$, then for all informants (resp. texts) σ of K, there exists $m \in \mathbb{N}$ such that $\mathbf{M}_\sigma(i) \neq \triangle$ for all $i < m$, and $\mathbf{M}_\sigma(i) = \triangle$ for all $i \geq m$.

Conditions 2 and 3 in the above definition mean that a learner $\mathbf{M}$ refutes the set $\mathcal{F}$ in finite time if and only if a target figure $K \in \mathcal{K}^* \setminus \mathcal{F}$. To characterize refutable learning, we prepare the following lemma, which is a translation of Mukouchi and Arikawa (1995, Lemma 4).

Lemma 2.22

Suppose a learner $\mathbf{M}$ **FigRefEx-Inf**-*learns* (*resp.* **FigRefEx-Txt**-*learns*) *a set of figures* $\mathcal{F}$*, and let* $K \in \mathcal{K}^* \setminus \mathcal{F}$*. For every informant* (*resp. text*) σ *of* K*, if* $\mathbf{M}$ *outputs* $\triangle$ *after receiving* $\sigma[n]$*, then for any* $L \in \mathcal{F}$*, the set of examples* $\mathrm{range}(\sigma[n])$ *is not consistent with* L*.*

Here we compare **FigRefEx-Inf**-learnability with other learning criteria.

Theorem 2.23

FigRefEx-Inf $\not\subseteq$ **FigEx-Txt** *and* **FigEx-Txt** $\not\subseteq$ **FigRefEx-Inf**.

Proof. First we consider **FigRefEx-Inf** $\not\subseteq$ **FigEx-Txt**. We show an example of a set of figures $\mathcal{F}$ with $\mathcal{F} \in$ **FigRefEx-Inf** and $\mathcal{F} \notin$ **FigEx-Txt** in the case of $d = 2$. Let $K_0 = \kappa(\{\langle 0, 0 \rangle, \langle 1, 1 \rangle\})$, $K_i = \kappa(\{\langle w, w \rangle \mid w \in \Sigma^i \setminus \{1^i\}\})$ for every $i \geq 1$, and $\mathcal{F} = \{K_i \mid i \in \mathbb{N}\}$. Note that K_0 is the line $y = x$ and $K_i \subset K_0$ for all $i \geq 1$.

We prove that $\mathcal{F} \in$ **FigRefEx-Inf**. If a target figure $K \supset K_0$, it is trivial that, for any K's informant σ, the set of examples $\mathrm{range}(\sigma[n])$ for some $n \in \mathbb{N}$ is not consistent with any $K_i \in \mathcal{F}$ (consider a positive example for a point $x \in K \setminus K_0$). Otherwise if $K \subset K_0$, there should exist a negative example $\langle v, v \rangle \in \mathrm{Neg}(K)$. Then we have $K \neq K_i$ for all $i > |v|$. Thus a leaner can refute candidates $\{K_1, K_2, \dots, K_{|v|}\}$ in finite time. Therefore from Lemma 2.22, $\mathcal{F} \in$ **FigRefEx-Inf** holds.

Next we show that $\mathcal{F} \notin$ **FigEx-Txt**. Let K_0 be the target figure. For any finite set of positive examples $\mathcal{T} \subset \mathrm{Pos}(K_0)$, there exists a figure $K_i \in \mathcal{F}$ such that $K_i \subset K_0$ and $\mathcal{T}$ is consistent with K_i. Therefore it has no finite tell-tale set with respect to $\mathcal{F}$ and hence $\mathcal{F} \notin$ **FigEx-Txt** from Theorem 2.9.

Second we check **FigEx-Txt** $\not\subseteq$ **FigRefEx-Inf**. Assume that $\mathcal{F} = \kappa(\mathcal{H}_{\{1\}})$ and a target figure K is a singleton $\{x\}$ with $K \notin \mathcal{F}$. It is clear that, for any K's informant σ and $n \in \mathbb{N}$, $\mathrm{range}(\sigma[n])$ is consistent with some figure $L \in \mathcal{F}$. Thus $\mathcal{F} \notin$ **FigRefEx-Inf** whereas $\mathcal{F} \in$ **FigEx-Txt**. □

Theorem 2.24

FigRelEx-Txt $\not\subseteq$ **FigRefEx-Inf** *and* **FigRefEx-Inf** $\not\subseteq$ **FigRelEx-Txt**.

Proof. It is trivial that **FigRelEx-Txt** $\not\subseteq$ **FigRefEx-Inf** since we have $\kappa(\mathcal{H}_{\{1\}}) \notin$ **FigRefEx-Inf** in the above proof and $\kappa(\mathcal{H}_{\{1\}}) \in$ **FigRelEx-Txt** from Theorem 2.19.

Next we consider **FigRefEx-Inf** $\not\subseteq$ **FigRelEx-Txt**. We show a counterexample in the case of $d = 2$. Let $\mathcal{F} = \{K\}$ with $K = \kappa(\{\langle 0, 0 \rangle, \langle 1, 1 \rangle\})$ and **M** be a **FigRelEx-Txt**-learner of $\mathcal{F}$. Notice that the figure K is the line $y = x$. For simplicity, in the following we assume that every text σ of a figure K is an infinite sequence of finite sequences $w_0, w_1, w_2, \dots$ such that $\{w_i\}_{i \in \mathbb{N}} = \mathrm{Pos}(K)$.

For a dense subset $x_0, x_1, \dots$ of the figure K, define $X_i = \{x_0, x_1, \dots, x_{i-1}\}$. Then, for each $n \in \mathbb{N}$, there exists a prefix $\sigma_{r(n)}$ of a text for $X_{r(n)}$ with some monotonically increasing function $r : \mathbb{N} \to \mathbb{N}$ satisfying the following three conditions:

1. The learner **M** makes a mind change (changes the current hypothesis) when it receives $\sigma_{r(n)}$,
2. $\mathrm{range}(\sigma_{r(n)}) \supseteq \{w \in \mathrm{Pos}(K) \mid |w| \le n\}$, and
3. $\sigma_{r(n-1)} \sqsubseteq \sigma_{r(n)}$ (with $n \ge 1$).

If n goes to infinite, $\sigma_{r(n)}$ becomes a text for K, but **M**'s output does not converge to any hypothesis. Thus $\mathcal{F} \notin$ **FigRelEx-Txt**.

On the other hand, we can easily check that $\mathcal{F} \in$ **FigRefEx-Inf**. □

These results mean that both **FigRefEx-Inf**- and **FigRelEx-Txt**-learning are difficult, but they are incomparable in terms of learnability. Furthermore, we have the following hierarchy.

Theorem 2.25
FigRefEx-Txt $\neq \emptyset$ *and* **FigRefEx-Txt** $\subset$ **FigRefEx-Inf**.

Proof. Let a set of figures $\mathcal{F}$ be a singleton $\{K\}$ such that $K = \kappa(w)$ for some $w \in (\Sigma^d)^*$. Then there exists a learner **M** that **FigRefEx-Txt**-learns $\mathcal{F}$, since all **M** has to do is to check whether or not, for a given positive example $(v, 1)$, $v \sqsubseteq u$ for some $u \in \mathrm{Pos}(K) = \{x \mid x \sqsubseteq www\dots\}$.

Next, let $\mathcal{F} = \{K\}$ such that $K = \kappa(H)$ with $\#H \ge 2$. We can easily check that $\mathcal{F} \notin$ **FigRefEx-Txt** because if a target figure L is a proper subset of K, no learner can refute $\mathcal{F}$ in finite time. Conversely, $\mathcal{F} \in$ **FigRefEx-Inf** since for all L with $L \neq K$, there exists an example with which the hypothesis H is not consistent. □

Corollary 2.26
FigRefEx-Txt $\subset$ **FigRelEx-Txt**.

2.4 Effective Learning of Figures

In learning under the proposed criteria, *i.e.*, explanatory, consistent, reliable, and refutable learning, each hypothesis is just considered as exactly "correct" or not, that is, for a target figure K and for a hypothesis H, H is correct if $\mathrm{GE}(K, H) = 0$ and is not correct if $\mathrm{GE}(K, H) \neq 0$. Thus we cannot know the rate of convergence to the target figure and how far it is from the recent hypothesis to the target. It is therefore more useful if we consider *approximate* hypotheses by taking various *generalization errors* into account in the learning process.

generalization error

We define novel learning criteria, **FigEfEx-Inf**- and **FigEfEx-Txt**-learning (EF means EFfective), to introduce into learning the concept of *effectivity*, which has been analyzed in computation of real numbers in the area of computable analysis (Weihrauch, 2000). Intuitively, these criteria guarantee that for any target figure, a generalization error becomes smaller and smaller monotonically and converges

effectivity

to zero. Thus we can know when the learner learns the target figure "well enough". Furthermore, if a target figure is learnable in the limit, then the generalization error goes to zero in finite time.

Definition 2.27: Effective learning
A learner **M** **FigEfEx-Inf**-*learns* (resp. **FigEfEx-Txt**-*learns*) a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$ if **M** satisfies the following conditions:

FigEfEx-Inf-learning
FigEfEx-Txt-learning

1. The learner **M** **FigEx-Inf**-learns (resp. **FigEx-Txt**-learns) $\mathcal{F}$.
2. For an arbitrary target figure $K \in \mathcal{K}^*$ and all informants (resp. texts) σ of K, for all $i \in \mathbb{N}$,

$$\mathrm{GE}(K, \mathbf{M}_\sigma(i)) \leq 2^{-i}.$$

This definition is inspired by the *Cauchy representation* (Weihrauch, 2000, Definition 4.1.5) of real numbers.

Effective learning is related to *monotonic* learning (Lange and Zeugmann, 1993, 1994; Kinber, 1994; Zeugmann *et al.*, 1995) originally introduced by Jantke (1991) and Wiehagen (1991), since both learning models consider monotonic convergence of hypotheses. In contrast to their approach, where various monotonicity over languages was considered, we geometrically measure the generalization error of a hypothesis by the Hausdorff metric. On the other hand, the effective learning is different from **Bc**-learning developed in the learning of languages and recursive functions (Jain *et al.*, 1999b) since **Bc**-learning only guarantees that generalization errors go to zero in finite time. This means that **Bc**-learning is *not* effective.

First we show that we can bound the generalization error of the hypothesis H using the diameter $\mathrm{diam}(k)$ of the set $\rho(w)$ with $|w| = k$. Recall that we have

$$\mathrm{diam}(k) = \sqrt{d} \cdot 2^{-k}$$

(see proof of Lemma 2.3). In the following, we denote by E^k the set of examples $\{(w, l) \mid |w| = k\}$ in σ and call each example in it a *level-k example*.

level-*k* example

Lemma 2.28
Let σ be an informant of a figure K and H be a hypothesis that is consistent with the set of examples $E^k = \{(w, l) \mid |w| = k\}$. We have the inequality

$$\mathrm{GE}(K, H) \leq \mathrm{diam}(k).$$

Proof. Since H is consistent with E^k,

$$\kappa(H) \cap \rho(w) \begin{cases} \neq \emptyset & \text{if } (w, 1) \in E^k, \\ = \emptyset & \text{if } (w, 0) \in E^k. \end{cases}$$

For $\delta = \mathrm{diam}(k)$, the δ-neighborhood of $\kappa(H)$ contains K and the δ-neighborhood of K contains $\kappa(H)$. Thus it follows that $\mathrm{GE}(K, H) = d_{\mathrm{H}}(K, \kappa(H)) \leq \mathrm{diam}(k)$. □

Theorem 2.29
The set of figures $\kappa(\mathcal{H})$ is **FigEfEx-Inf**-*learnable.*

Proof. We show the pseudo-code of **M** that **FigEfEx-Inf**-learns $\kappa(\mathcal{H})$ in Procedure 2.3. We use the function

$$g(k) = \lceil k + \log_2 \sqrt{d}\, \rceil.$$

Then for all $k \in \mathbb{N}$, we have

$$\mathrm{diam}(g(k)) = \sqrt{d} \cdot 2^{-g(k)} \leq 2^{-k}.$$

The learner **M** stores examples, and when it receives all examples at the level $g(k)$, it outputs a hypothesis. Every kth hypothesis $\mathbf{M}_\sigma(k)$ is consistent with the set of examples $E^{g(k)}$. Thus we have

$$\mathrm{GE}(K, \mathbf{M}_\sigma(k)) \leq \mathrm{diam}(g(k)) \leq 2^{-k}$$

for all $k \in \mathbb{N}$ from Lemma 2.28.

Assume that $K \in \kappa(\mathcal{H})$. If **M** outputs a wrong hypothesis, there must be a positive or negative example that is not consistent with the hypothesis, and it changes the wrong hypothesis. If it produces a correct hypothesis, then it never changes the correct hypothesis, since every example is consistent with the hypothesis. Thus there exists $n \in \mathbb{N}$ with $\mathrm{GE}(K, \mathbf{M}_\sigma(i)) = 0$ for all $i \geq n$. Therefore, **M** **FigEfEx-Inf**-learns $\kappa(\mathcal{H})$. □

Corollary 2.30
FigEfEx-Inf = **FigRelEx-Inf** = **FigEx-Inf**.

The learner with Procedure 2.3 can treat the set of *all* figures $\mathcal{K}^*$ as learning targets, since for any figure $K \in \mathcal{K}^*$, it can approximate the figure arbitrarily closely using only the figures represented by hypotheses in the hypothesis space $\mathcal{H}$.

In contrast to **FigEx-Txt**-learning, there is no set of figures that is **FigEfEx-Txt**-learnable.

Theorem 2.31
FigEfEx-Txt = $\emptyset$.

Proof. We show a counterexample of a target figure which no learner **M** can approximate effectively. Assume that $d = 2$ and a learner **M** **FigEfEx-Txt**-learns a set of figures $\mathcal{F} \subseteq \mathcal{K}^*$. Let us consider two target figures $K = \{(0,0),(1,1)\}$ and $L = \{(0,0)\}$. For L's text σ, for all examples $(w, 1) \in \mathrm{range}(\sigma)$, $w \in \{00\}^*$. Since **M** **FigEfEx-Txt**-learns $\mathcal{F}$, it should output the hypothesis H as $\mathbf{M}_\sigma(2)$ such that $\mathrm{GE}(L, H) < 1/4$. Suppose that **M** receives n examples before outputting the hypothesis H. Then there exists a presentation τ of the figure K such that $\tau[n-1] = \sigma[n-1]$, and **M** outputs the hypothesis H with receiving $\tau[n-1]$. However, $\mathrm{GE}(K, H) \geq \sqrt{2} - 1/4$ holds from the triangle inequality, contradicting our assumption that **M** **FigEfEx-Txt**-learns $\mathcal{F}$. This proof can be applied for any $\mathcal{F} \subseteq \mathcal{K}^*$, thereby we have **FigEfEx-Txt** = $\emptyset$. □

Since **FigRefEx-Txt** $\neq \emptyset$, we have the relation

FigEfEx-Txt $\subset$ **FigRefEx-Txt**.

This result means that we cannot learn any figures "effectively" by using only positive examples.

Procedure 2.3: Learning procedure that **FigEfEx-Inf**-learns $\kappa(\mathcal{H})$

```
Input: Informant $\sigma = (w_0, l_0), (w_1, l_1), \dots$ of figure $K \in \kappa(\mathcal{H})$
Output: Infinite sequence of hypotheses $\mathbf{M}_\sigma(0), \mathbf{M}_\sigma(1), \dots$

$i \leftarrow 0$
$k \leftarrow 0$
$E \leftarrow \emptyset$    // $E$ is a set of received examples
repeat
    read $\sigma(i)$ and add to $E$    // $\sigma(i) = (w_i, l_i)$
    if $E^{g(k)} \subseteq E$ then
        // $E^{g(k)} = \{(w, l) \in \mathrm{range}(\sigma) \mid |w| = g(k)\}$
        // $g(k) = \lceil k + \log_2 \sqrt{d}\, \rceil$
        search the first $H$ that is consistent with $E$
        through a normal enumeration
        output $H$    // $\mathbf{M}_\sigma(i) = H$
        $k \leftarrow k + 1$
    end if
    $i \leftarrow i + 1$
until forever
```

2.5 Evaluation of Learning Using Dimensions

Here we show a novel mathematical connection between fractal geometry and Gold-style learning under the proposed learning model described in Section 2.2. More precisely, we measure the number of positive examples, one of the complexities of learning, using the Hausdorff dimension and the VC dimension. The Hausdorff dimension is known as the central concept of fractal geometry, which measures the density of figures, and VC dimension is the central concept of the Valiant-style learning model (PAC learning model) (Kearns and Vazirani, 1994), which measures the complexity of classes of hypotheses.

2.5.1 Preliminaries for Dimensions

First we introduce the Hausdorff and related dimensions: the box-counting dimension, the similarity dimension, and the VC dimension.

For $X \subseteq \mathbb{R}^d$ and $s \in \mathbb{R}$ with $s > 0$, define

$$\mathfrak{H}^s_\delta(X) := \inf \left\{ \sum_{U \in \mathcal{U}} |U|^s \;\middle|\; \mathcal{U} \text{ is a } \delta\text{-cover of } X \right\}.$$

The *s-dimensional Hausdorff measure* of X is $\lim_{\delta \to 0} \mathfrak{H}^s_\delta(X)$, denoted by $\mathfrak{H}^s(X)$. We say that $\mathcal{U}$ is a δ-cover of X if $\mathcal{U}$ is countable, $X \subseteq \bigcup_{U \in \mathcal{U}} U$, and $|U| \le \delta$ for all $U \in \mathcal{U}$. When we fix a set X and view $\mathfrak{H}^s(X)$ as a function with respect to s, it has at most one value where the value $\mathfrak{H}^s(X)$ changes from ∞ to 0 (Federer, 1996). This value is called the *Hausdorff dimension* of X. Formally, the Hausdorff dimension of

s-dimensional Hausdorff measure

Hausdorff dimension

a set X, written as $\dim_{\mathrm{H}} X$, is defined by

$$\dim_{\mathrm{H}} X \coloneqq \sup\{\, s \mid \mathfrak{H}^s(X) = \infty \,\} = \inf\{\, s \geq 0 \mid \mathfrak{H}^s(X) = 0 \,\}.$$

The box-counting dimension, also known as the Minkowski-Bouligand dimension, is one of widely used dimensions since its mathematical calculation and empirical estimation are relatively easy compared to the Hausdorff dimension. Moreover, if we try to calculate the box-counting dimension, which is given as the limit of the following equation (2.10) by decreasing δ, the obtained values often converge to the Hausdorff dimension at the same time. Thus we can obtain an approximate value of the Hausdorff dimension by an empirical method. Let X be a nonempty bounded subset of $\mathbb{R}^d$ and $N_\delta(X)$ be the smallest cardinality of a δ-cover of X. The *box-counting dimension* $\dim_{\mathrm{B}} X$ of X is defined by

box-counting dimension

$$\dim_{\mathrm{B}} X \coloneqq \lim_{\delta \to 0} \frac{\log N_\delta(X)}{-\log \delta} \tag{2.10}$$

if this limit exists. We have

$$\dim_{\mathrm{H}} X \leq \dim_{\mathrm{B}} X$$

for all $X \subseteq \mathbb{R}^d$.

It is usually difficult to find the Hausdorff dimension of a given set. However, we can obtain the dimension of a certain class of self-similar sets in the following manner. Let C be a finite set of contractions, and F be the self-similar set of C. The *similarity dimension* of F, denoted by $\dim_{\mathrm{S}} F$, is defined by the equation

similarity dimension

$$\sum_{\mathrm{CT} \in C} L(\mathrm{CT})^{\dim_{\mathrm{S}} F} = 1,$$

contractivity factor

where $L(\mathrm{CT})$ is the *contractivity factor* of CT, which is defined by the infimum of all real numbers c with $0 < c < 1$ such that $d(\mathrm{CT}(x), \mathrm{CT}(y)) \leq cd(x, y)$ for all $x, y \in X$. We have

$$\dim_{\mathrm{H}} F \leq \dim_{\mathrm{S}} F$$

and if C satisfies the open set condition,

$$\dim_{\mathrm{H}} F = \dim_{\mathrm{B}} F = \dim_{\mathrm{S}} F$$

open set condition

(Falconer, 2003). Here, a finite set of contractions C satisfies the *open set condition* if there exists a nonempty bounded open set $O \subset \mathbb{R}^d$ such that $\mathrm{CT}(O) \subset O$ for all $\mathrm{CT} \in C$ and $\mathrm{CT}(O) \cap \mathrm{CT}'(O) = \emptyset$ for all $\mathrm{CT}, \mathrm{CT}' \in C$ with $\mathrm{CT} \neq \mathrm{CT}'$.

Intuitively, the VC dimension (Vapnik and Chervonenkis, 1971; Valiant, 1984) is a parameter of separability, which gives lower and upper bounds for the sample size in the Valiant-style, or PAC, learning model (Kearns and Vazirani, 1994). For all $\mathcal{R} \subseteq \mathcal{H}$ and $W \subseteq \Sigma^*$, define

$$\Pi_{\mathcal{R}}(W) \coloneqq \{\, \mathrm{Pos}(\kappa(H)) \cap W \mid H \in \mathcal{R} \,\}.$$

VC dimension

If $\Pi_{\mathcal{R}}(W) = 2^W$, we say that W is *shattered* by $\mathcal{R}$. Here the *VC dimension* of $\mathcal{R}$, denoted by $\dim_{\mathrm{VC}} \mathcal{R}$, is the cardinality of the largest set W shattered by $\mathcal{R}$.

2.5.2 Measuring the Complexity of Learning with Dimensions

We show that the Hausdorff dimension of a target figure gives a lower bound to the number of positive examples. Remember that $\mathrm{Pos}_k(K) = \{w \in \mathrm{Pos}(K) \mid |w| = k\}$ and the diameter $\mathrm{diam}(k)$ of the set $\rho(w)$ with $|w| = k$ is $\sqrt{d}2^{-k}$. Moreover, for all $k \in \mathbb{N}$, $\#\{w \in (\Sigma^d)^* \mid |w| = k\} = 2^{kd}$.

> **Theorem 2.32**
> *For every figure $K \in \mathcal{K}^*$ and for any $s < \dim_{\mathrm{H}} K$, if we take k large enough,*
>
> $$\#\mathrm{Pos}_k(K) \geq 2^{ks}.$$

Proof. Fix $s < \dim_{\mathrm{H}} K$. From the definition of the Hausdorff measure,

$$\mathfrak{H}^s_{\mathrm{diam}(k)}(K) \leq \#\mathrm{Pos}_k(K) \cdot (\sqrt{d}2^{-k})^s$$

since $\mathrm{diam}(k) = \sqrt{d}2^{-k}$. If we take k large enough,

$$\mathfrak{H}^s_{\mathrm{diam}(k)}(K) \geq (\sqrt{d})^s$$

because $\mathfrak{H}^s_\delta(K)$ is monotonically increasing with decreasing δ, and goes to ∞. Thus

$$\#\mathrm{Pos}_k(K) \geq \mathfrak{H}^s_{\mathrm{diam}(k)}(K)(\sqrt{d}2^{-k})^{-s} \geq (\sqrt{d})^s(\sqrt{d}2^{-k})^{-s} = 2^{ks}$$

holds. □

Moreover, if a target figure K can be represented by some hypothesis, that is, $K \in \kappa(\mathcal{H})$, we can use the exact dimension $\dim_{\mathrm{H}} K$ as a bound for the number $\#\mathrm{Pos}_k(K)$. We use the *mass distribution principle* (Falconer, 2003, Mass distribution principle 4.2): Let μ be a mass distribution, which is a measure on a bounded subset of $\mathbb{R}^d$ satisfying $0 < \mu(\mathbb{R}^d) < \infty$, over K. Assume that for some s there exists $c > 0$ and $\varepsilon > 0$ such that $\mu(U) \leq c|U|^s$ holds for all sets U with $|U| < \varepsilon$. Then we have $\mathfrak{H}^s(K) \geq \mu(K)/c$.

mass distribution principle

> **Theorem 2.33**
> *For every figure $K \in \kappa(\mathcal{H})$ with $K = \kappa(H)$, if we take k large enough,*
>
> $$\#\mathrm{Pos}_k(K) \geq 2^{k \dim_{\mathrm{H}} K}.$$

Proof. From the definition of the Hausdorff measure,

$$\mathfrak{H}^{\dim_{\mathrm{H}} K}_{\mathrm{diam}(k)}(K) \leq \#\mathrm{Pos}_k(K) \cdot (\sqrt{d}2^{-k})^{\dim_{\mathrm{H}} K}.$$

Let μ be a mass distribution such that $\mu(\rho(w)) = 2^{-kd}$ for each $w \in (\Sigma^d)^k$. Assume that

$$c = (\sqrt{d})^{-\dim_{\mathrm{H}} K}.$$

We have

$$\begin{aligned}\mu(\rho(w)) = 2^{-kd} \leq c \cdot \mathrm{diam}(k)^{\dim_{\mathrm{H}} K} &= \sqrt{d}^{-\dim_{\mathrm{H}} K} \cdot \left(\sqrt{d}2^{-k}\right)^{\dim_{\mathrm{H}} K} \\ &= 2^{-k\dim_{\mathrm{H}} K}.\end{aligned}$$

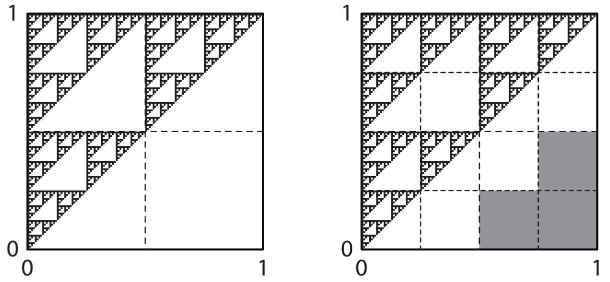


Figure 2.4 | Positive and negative examples for the Sierpiński triangle at level 1 and 2. White (resp. gray) squares mean positive (resp. negative) examples.

Thus from the mass distribution principle,

$$\mathfrak{H}^{\dim_{\mathrm{H}} K}(K) \geq 1/c = (\sqrt{d})^{\dim_{\mathrm{H}} K}.$$

Therefore if we take k large enough,

$$\begin{aligned}\#\mathrm{Pos}_k(K) \geq \mathfrak{H}^{\dim_{\mathrm{H}} K}_{\mathrm{diam}(k)}(K)(\sqrt{d}2^{-k})^{-\dim_{\mathrm{H}} K} &\geq (\sqrt{d})^{\dim_{\mathrm{H}} K}(\sqrt{d}2^{-k})^{-\dim_{\mathrm{H}} K}\\ &= 2^{k\dim_{\mathrm{H}} K}\end{aligned}$$

holds. □

Example 2.34
Let us consider the figure K in Example 2.1. It is known that $\dim_{\mathrm{H}} K = \log 3/\log 2 = 1.584\ldots$. From Theorem 2.33,

$$\mathrm{Pos}_1(K) \geq 2^{\dim_{\mathrm{H}} K} = 3$$

holds at level 1 and

$$\mathrm{Pos}_2(K) \geq 4^{\dim_{\mathrm{H}} K} = 9$$

holds at level 2. Actually, $\mathrm{Pos}_1(K) = 4$ and $\mathrm{Pos}_2(K) = 13$. Note that K is already covered by 3 and 9 intervals at level 1 and 2, respectively (Figure 2.4). □

The VC dimension can also be used to characterize the number of positive examples. Define

$$\mathcal{H}^k := \{\, H \in \mathcal{H} \mid |w| = k \text{ for all } w \in H \,\},$$

and call each hypothesis in the set a *level-k hypothesis*. We show that the VC dimension of the set of level k hypotheses $\mathcal{H}^k$ is equal to $\#\{w \in (\Sigma^d)^* \mid |w| = k\} = 2^{kd}$.

Lemma 2.35
At each level k, we have $\dim_{\mathrm{VC}} \mathcal{H}^k = 2^{kd}$.

Proof. First of all,

$$\dim_{\mathrm{VC}} \mathcal{H}^k \leq 2^{kd}$$

is trivial since $\#\mathcal{H}^k = 2^{2^{kd}}$. Let $\mathcal{H}^k_n$ denote the set $\{H \in \mathcal{H}^k \mid \#H = n\}$. For all $H \in \mathcal{H}^k_1$, there exists $w \in \mathrm{Pos}(\kappa(H))$ such that $w \notin \mathrm{Pos}(\kappa(G))$ for all $G \in \mathcal{H}^k_1$ with $H \neq G$. Thus if we assume $\mathcal{H}^k_1 = \{H_1, \ldots, H_{2^{kd}}\}$, there exists the set of finite sequences $W = \{w_1, \dots, w_{2^{kd}}\}$ such that for all $i \in \{1, \dots, 2^{kd}\}$, $w_i \in \mathrm{Pos}(\kappa(H_i))$ and $w_i \notin \mathrm{Pos}(\kappa(H_j))$ for all $j \in \{1, \dots, 2^{kd}\}$ with $i \neq j$. For every pair $V, W \subset (\Sigma^d)^*$, $V \subset W$ implies $\kappa(V) \subset \kappa(W)$. Therefore the set W is shattered by $\mathcal{H}^k$, meaning that we have $\dim_{\mathrm{VC}} \mathcal{H}^k = 2^{kd}$. □

Therefore we can rewrite Theorems 2.32 and 2.33 as follows.

Theorem 2.36
For every figure $K \in \mathcal{K}^*$ and for any $s < \dim_{\mathrm{H}} K$, if we take k large enough,

$$\#\mathrm{Pos}_k(K) \geq (\dim_{\mathrm{VC}} \mathcal{H}^k)^{s/d}.$$

Moreover, if $K \in \kappa(\mathcal{H})$ with $K = \kappa(W)$, if we take k large enough,

$$\#\mathrm{Pos}_k(K) \geq (\dim_{\mathrm{VC}} \mathcal{H}^k)^{\dim_{\mathrm{H}} K/d}.$$

These results demonstrate a relationship among the complexities of learning figures (numbers of positive examples), classes of hypotheses (VC dimension), and target figures (Hausdorff dimension).

2.5.3 Learning the Box-Counting Dimension Effectively

One may think that **FIGEFEX-INF**-learning can be achieved without the proposed hypothesis space, since if a learner just outputs figures represented by a set of received positive examples, the generalization error becomes smaller and smaller. Here we show that one "quality" of a target figure, the box-counting dimension, is also learned in **FIGEFEX-INF**-learning, whereas if a learner outputs figures represented by a set of received positive examples, the box-counting dimension (and also the Hausdorff dimension) of any figure is always d.

Recall that for all hypotheses $H \in \mathcal{H}$, $\dim_{\mathrm{H}} \kappa(H) = \dim_{\mathrm{B}} \kappa(H) = \dim_{\mathrm{S}} \kappa(H)$, since the set of contractions encoded by H meets the open set condition.

Theorem 2.37
Assume that a learner **M** **FIGEFEX-INF**-*learns $\kappa(\mathcal{H})$. For all target figures $K \in \mathcal{K}^*$,*

$$\lim_{k \to \infty} \dim_{\mathrm{B}} \kappa(\mathbf{M}_\sigma(k)) = \dim_{\mathrm{B}} K.$$

Proof. First, we assume that a target figure $K \in \kappa(\mathcal{H})$. For every informant σ of K, $\mathbf{M}_\sigma$ converges to a hypothesis H with $\kappa(H) = K$. Thus

$$\lim_{k \to \infty} \dim_{\mathrm{B}} \kappa(\mathbf{M}_\sigma(k)) = \dim_{\mathrm{B}} K = \dim_{\mathrm{H}} K.$$

Next, we assume $K \in \mathcal{K}^* \setminus \kappa(\mathcal{H})$. Since $\mathrm{GE}(K, \mathbf{M}_\sigma(i)) \leq 2^{-i}$ holds for every $i \in \mathbb{N}$, for each $k \in \mathbb{N}$, we have some $i \geq k$ such that the hypothesis $\mathbf{M}_\sigma(i)$ is consistent with the set of level-k examples $E^k = \{(w, l) \in \mathrm{range}(\sigma) \mid |w| = k\}$. Thus

$$\dim_{\mathrm{S}} \kappa(\mathbf{M}_\sigma(i)) = \dim_{\mathrm{B}} \kappa(\mathbf{M}_\sigma(i)) = \frac{\log \#\mathrm{Pos}_k(K)}{-\log 2^{-k}}.$$

Falconer (2003) shows that the box-counting dimension $\dim_{\mathrm{B}} K$ is defined equivalently by

$$\dim_{\mathrm{B}} K = \lim_{k\to\infty} \frac{\log \#\mathrm{Pos}_k(K)}{-\log 2^{-k}},$$

where 2^{-k} is the length of one side of an interval $\rho(w)$ with $|w| = k$. Therefore from the definition of the box-counting dimension,

$$\lim_{i\to\infty} \dim_{\mathrm{H}} \kappa(\mathbf{M}_\sigma(i)) = \lim_{k\to\infty} \frac{\log \#\mathrm{Pos}_k(K)}{-\log 2^{-k}} = \dim_{\mathrm{B}} K$$

holds. □

2.6 Computational Interpretation of Learning

Recently, the notion of "computability" for continuous objects has been introduced (Schröder, 2002a; Weihrauch, 2000, 2008; Weihrauch and Grubba, 2009; Tavana and Weihrauch, 2011) in the framework of Type-2 Theory of Effectivity (TTE), where we treat an uncountable set X as objects for computing through infinite sequences over a given alphabet Σ. Using the framework, we analyze our learning model from the computational point of view. Some studies by de Brecht and Yamamoto (2009); de Brecht (2010) have already demonstrated a close connection between TTE and the Gold-style learning model, and our analysis becomes an instance and extension of their analysis.

2.6.1 Preliminaries for Type-2 Theory of Effectivity

First, we prepare mathematical notations for TTE. In the following in this section, we assume $\Sigma = \{0, 1, [,], \|\}$. A partial (resp. total) function g from a set A to a set representation B is denoted by $g :\subseteq A \to B$ (resp. $g : A \to B$). A *representation* of a set X is a surjection $\xi :\subseteq C \to X$, where C is Σ^* or Σ^ω. We see $p \in \mathrm{dom}(\xi)$ as a name of the encoded element $\xi(p)$.

Computability of string functions $f :\subseteq X \to Y$, where X and Y are Σ^* or Σ^ω, is Type-2 machine defined via a *Type-2 machine*, which is a usual Turing machine with one-way input tapes, some work tapes, and a one-way output tape (Weihrauch, 2000). The function $f_M :\subseteq X \to Y$ computed by a Type-2 machine M is defined as follows: When Y is Σ^*, $f_M(p) := q$ if M with input p halts with q on the output tape, and when Y is Σ^ω, $f_M(p) := q$ if M with input p writes step by step q onto the output tape. computability We say that a function $f :\subseteq C \to D$ is *computable* if there is a Type-2 machine that computes f, and a finite or infinite sequence p is computable if the constant function f which outputs p is computable. A Type-2 machine never changes symbols that have already been written onto the output tape, thus each prefix of the output depends only on a prefix of the input.

By treating a Type-2 machine as a translator between names of some objects, a hierarchy of representations is introduced. A representation ξ is *reducible* to ζ, denoted by $\xi \le \zeta$, if there exists a computable function f such that $\xi(p) = \zeta(f(p))$ equivalent for all $p \in \mathrm{dom}(\xi)$. Two representations ξ and ζ are *equivalent*, denoted by $\xi \equiv \zeta$, if both $\xi \le \zeta$ and $\zeta \le \xi$ hold. As usual, $\xi < \zeta$ means $\xi \le \zeta$ and not $\zeta \le \xi$.

Computability for functions is defined through representations and computability of string functions.

Definition 2.38: Computability and realization
Let ξ and ζ be representations of X and Y, respectively. An element $x \in X$ is ξ-*computable* if there is some computable p such that $\xi(p) = x$. A function $f :\subseteq X \to Y$ is (ξ, ζ)-*computable* if there is some computable g such that

ξ-computable

(ξ, ζ)-computable

$$f \circ \xi(p) = \zeta \circ g(p)$$

for all $p \in \operatorname{dom}(\xi)$. This g is called a (ξ, ζ)-*realization* of f.

(ξ, ζ)-realization

Thus the abstract function f is "realized" by the concrete function (Type-2 machine) g through the two representations ξ and ζ.

Various representations of the set of nonempty compact sets $\mathcal{K}^*$ are well studied by Brattka and Weihrauch (1999); Brattka and Presser (2003). Let

$$Q = \left\{ A \subset \mathbb{Q}^d \;\middle|\; A \text{ is finite and nonempty} \right\}$$

and define a representation $\nu_Q :\subseteq \Sigma^* \to Q$ by

$$\nu_Q([w_0 \| w_1 \| \dots \| w_n]) := \{\nu_{\mathbb{Q}^d}(w_0), \nu_{\mathbb{Q}^d}(w_1), \dots, \nu_{\mathbb{Q}^d}(w_n)\},$$

where $\nu_{\mathbb{Q}^d} :\subseteq (\Sigma^d)^* \to \mathbb{Q}^d$ is the standard binary notation of rational numbers defined by

$$\nu_{\mathbb{Q}^d}(\langle w^1, w^2, \dots, w^d \rangle)$$
$$:= \left(\sum_{i=0}^{|w^1|-1} w_i^1 \cdot 2^{-(i+1)}, \sum_{i=0}^{|w^2|-1} w_i^2 \cdot 2^{-(i+1)}, \dots, \sum_{i=0}^{|w^d|-1} w_i^d \cdot 2^{-(i+1)} \right)$$

and "[", "]", and "||" are special symbols used to separate two finite sequences. For a finite set of finite sequences $\{w_0, \dots, w_m\}$, for convenience we introduce the mapping ι which translates the set into a finite sequence defined by $\iota(w_0, \dots, w_m) := [w_0 \| \dots \| w_m]$. Note that $\nu_{\mathbb{Q}^d}(\langle w^1, \dots, w^d \rangle) = (\min \rho(w^1), \dots, \min \rho(w^d))$ for our representation ρ introduced in equation (2.2). The standard representation of the topological space $(\mathcal{K}^*, d_{\mathrm{H}})$, given by Brattka and Weihrauch (1999, Definition 4.8), is defined in the following manner.

Definition 2.39: Standard representation of figures
Define the representation $\kappa_{\mathrm{H}} :\subseteq \Sigma^\omega \to \mathcal{K}^*$ of figures by $\kappa_{\mathrm{H}}(p) = K$ if $p = w_0 \| w_1 \| w_2 \| \dots$,

$$d_{\mathrm{H}}(K, \nu_Q(w_i)) < 2^{-i}$$

for each $i \in \mathbb{N}$, and $\lim_{i\to\infty} \nu_Q(w_i) = K$.

Note that the topology on $\mathcal{K}^*$ induced by the Hausdorff metric d_{H} coincides with the *Vietoris topology* on $\mathcal{K}^*$ (Beer, 1993). This representation κ_{H} is known to be an *admissible representation* of the space $(\mathcal{K}^*, d_{\mathrm{H}})$, which is the key concept in TTE (Schröder, 2002a; Weihrauch, 2000), and is also known as the Σ_1^0-admissible representation proposed by de Brecht and Yamamoto (2009).

Vietoris topology

admissible representation

2.6.2 Computability and Learnability of Figures

First, we show computability of figures in $\kappa(\mathcal{H})$.

> **Theorem 2.40**
> *For every figure $K \in \kappa(\mathcal{H})$, K is κ_{H}-computable.*

Proof. It is enough to prove that there exists a computable function f such that $\kappa(H) = \kappa_{\mathrm{H}}(f(H))$ for all $H \in \mathcal{H}$. Fix a hypothesis $H \in \mathcal{H}$ such that $\kappa(H) = K$. For all $k \in \mathbb{N}$ and for H_k defined by

$$H_k := \left\{ w \in (\Sigma^d)^* \;\middle|\; w \sqsubseteq v \text{ with } v \in H^m \text{ for some } m, \text{ and } |w| = k \right\},$$

we can easily check that

$$d_{\mathrm{H}}(K, \nu_Q(\iota(H_k))) < \mathrm{diam}(k) = \sqrt{d} \cdot 2^{-k}.$$

Moreover, for each k, $\sqrt{d} \cdot 2^{-g(k)} < 2^{-k}$, where

$$g(k) = \lceil k + \log_2 \sqrt{d} \rceil.$$

Therefore there exists a computable function f which translates H into a representation of K given as follows: $f(H) = p$ such that $p = w_0 \| w_1 \| \dots$ such that $\iota(H_{g(k)}) = w_k$ for all $k \in \mathbb{N}$. □

Thus a hypothesis H can be viewed as a "code" of a Type-2 machine that produces a κ_{H}-representation of the figure $\kappa(H)$.

Both informants and texts are also representations (in the sense of TTE) of compact sets. Define the mapping η_{INF} by $\eta_{\mathrm{INF}}(\sigma) := K$ for every $K \in \mathcal{K}^*$ and informant σ of K, and the mapping η_{TXT} by $\eta_{\mathrm{TXT}}(\sigma) := K$ for every $K \in \mathcal{K}^*$ and text σ of K. Trivially $\eta_{\mathrm{INF}} < \eta_{\mathrm{TXT}}$ holds, that is, some Type-2 machine can translate η_{INF} to η_{TXT}, but no machine can translate η_{TXT} to η_{INF}. Moreover, we have the following hierarchy of representations.

> **Lemma 2.41**
> $\eta_{\mathrm{INF}} < \kappa_{\mathrm{H}}$*,* $\eta_{\mathrm{TXT}} \not\leq \kappa_{\mathrm{H}}$*, and* $\kappa_{\mathrm{H}} \not\leq \eta_{\mathrm{TXT}}$*.*

Proof. First we prove $\eta_{\mathrm{INF}} \leq \kappa_{\mathrm{H}}$, that is, there is some computable function f such that $\eta_{\mathrm{INF}}(\sigma) = \kappa_{\mathrm{H}}(f(\sigma))$. Fix a figure K and its informant $\sigma \in \mathrm{dom}(\eta_{\mathrm{INF}})$. For all $k \in \mathbb{N}$, we have

$$d_{\mathrm{H}}(K, \mathrm{Pos}_k(K)) \leq \mathrm{diam}(k) = \sqrt{d} \cdot 2^{-k}$$

and $\mathrm{Pos}_k(K)$ can be obtained from σ. Moreover, for each k, $\sqrt{d} \cdot 2^{-g(k)} < 2^{-k}$, where

$$g(k) = \lceil k + \log_2 \sqrt{d} \rceil.$$

Therefore there exists a computable function f that translates σ into a representation of K as follows: $f(\sigma) = p$, where $p = w_0 \| w_1 \| \dots$ such that $w_k = \iota(\mathrm{Pos}_{g(k)}(K))$ for all $k \in \mathbb{N}$.

Second, we prove $\eta_{\mathrm{TXT}} \not\leq \kappa_{\mathrm{H}}$. Assume that the opposite, $\eta_{\mathrm{TXT}} \leq \kappa_{\mathrm{H}}$ holds. Then there exists a computable function f such that $\eta_{\mathrm{TXT}}(\sigma) = \kappa_{\mathrm{H}}(f(\sigma))$ for every figure

$K \in \mathcal{K}^*$. Fix a figure K and its text $\sigma \in \mathrm{dom}(\eta_{\mathrm{TXT}})$. This means that for any small $\varepsilon \in \mathbb{R}$, f can find a prefix $\sigma[m]$ such that $d_{\mathrm{H}}(K, I) \leq \varepsilon$, where $I = \{\rho(w) \mid (w, 1) \in \mathrm{range}(\sigma[m])\}$. However, since σ contains only positive examples, no computable function can find such m. It follows that $\eta_{\mathrm{TXT}} \not\leq \kappa_{\mathrm{H}}$.

Third, we prove $\kappa_{\mathrm{H}} \not\leq \eta_{\mathrm{INF}}$ and $\kappa_{\mathrm{H}} \not\leq \eta_{\mathrm{TXT}}$. There is a figure K such that $K \cap \rho(w) = \{x\}$ for some $w \in \Sigma^*$, *i.e.*, K and $\rho(w)$ intersect in only one point x. Such a w must be in σ as a positive example, that is, $w \in \mathrm{Pos}(K)$. However, a representation of K can be constructed without w. There exists an infinite sequence $p \in \kappa_{\mathrm{H}}$ with $p = w_0 \| w_1 \| \ldots$ such that $x \notin \nu_Q(w_k)$ for all $k \in \mathbb{N}$. Therefore there is no computable function that outputs an example $(w, 1)$ from p, meaning that $\kappa_{\mathrm{H}} \not\leq \eta_{\mathrm{INF}}$ and $\kappa_{\mathrm{H}} \not\leq \eta_{\mathrm{TXT}}$. □

Here we interpret *learning* of figures as *computation* based on TTE. If we see the output of a learner, *i.e.*, an infinite sequence of hypotheses, as an infinite sequence encoding a figure, the learner can be viewed as a translator of codes of figures. Naturally, we can assume that the hypothesis space $\mathcal{H}$ is a discrete topological space, that is, every hypothesis $H \in \mathcal{H}$ is isolated and is an open set itself. Define the mapping $\lim_{\mathcal{H}} : \mathcal{H}^{\omega} \to \mathcal{H}$, where $\mathcal{H}^{\omega}$ is the set of infinite sequences of hypotheses in $\mathcal{H}$, by $\lim_{\mathcal{H}}(\tau) := H$ if τ is an infinite sequence of hypotheses that converges to H, *i.e.*, there exists $n \in \mathbb{N}$ such that $\tau(i) = \tau(n)$ for all $i \geq n$. This coincides with the *naïve Cauchy representation* given by Weihrauch (2000) and Σ^0_2-*admissible representation* of hypotheses introduced by de Brecht and Yamamoto (2009). For any set $\mathcal{F} \subseteq \mathcal{K}^*$, let $\mathcal{F}_{\mathrm{D}}$ denote the space $\mathcal{F}$ equipped with the discrete topology, that is, every subset of $\mathcal{F}$ is open, and the mapping $\mathrm{id}_{\mathcal{F}} : \mathcal{F} \to \mathcal{F}_{\mathrm{D}}$ be the identity on $\mathcal{F}$. The computability of this identity is not trivial, since the topology of $\mathcal{F}_{\mathrm{D}}$ is finer than that of $\mathcal{F}$. Intuitively, this means that $\mathcal{F}_{\mathrm{D}}$ has more informative than $\mathcal{F}$. We can interpret learnability of $\mathcal{F}$ as computability of the identity $\mathrm{id}_{\mathcal{F}}$. The results in the following are summarized in Figure 2.5.

naïve Cauchy representation

Σ^0_2-admissible representation

Theorem 2.42
A set $\mathcal{F} \subseteq \mathcal{K}^*$ is **FigEx-Inf**-*learnable (resp.* **FigEx-Txt**-*learnable) if and only if the identity* $\mathrm{id}_{\mathcal{F}}$ *is $(\eta_{\mathrm{INF}}, \kappa \circ \lim_{\mathcal{H}})$-computable (resp. $(\eta_{\mathrm{TXT}}, \kappa \circ \lim_{\mathcal{H}})$-computable).*

Proof. We only prove the case of **FigEx-Inf**-learning, since we can prove the case of **FigEx-Txt**-learning in exactly the same way.

The "only if" part: There is a learner $\mathbf{M}$ that **FigEx-Inf**-learns $\mathcal{F}$, hence for all $K \in \mathcal{F}$ and all $\sigma \in \mathrm{dom}(\eta_{\mathrm{INF}})$, $\mathbf{M}_\sigma$ converges to a hypothesis $H \in \mathcal{H}$ such that $\kappa(H) = K$. Thus

$$\mathrm{id}_{\mathcal{F}} \circ \eta_{\mathrm{INF}}(\sigma) = \kappa \circ \lim_{\mathcal{H}}(\mathbf{M}_\sigma), \tag{2.11}$$

and this means that $\mathrm{id}_{\mathcal{F}}$ is $(\eta_{\mathrm{INF}}, \kappa \circ \lim_{\mathcal{H}})$-computable.

The "if" part: For some $\mathbf{M}$, the above equation (2.11) holds for all $\sigma \in \mathrm{dom}(\eta_{\mathrm{INF}})$. This means that $\mathbf{M}$ is a learner that **FigEx-Inf**-learns $\mathcal{F}$. □

Here we consider two more learning criteria, **FigFin-Inf**-learning and **FigFin-Txt**-learning, where the learner generates only one correct hypothesis and halts. This learning corresponds to *finite learning*, or *one shot learning*, introduced by Gold (1967) and Trakhtenbrot and Barzdin (1970) and it is a special case of learning with a bound of *mind change complexity*, the number of changes of hypothesis, introduced by Freivalds and Smith (1993) and used to measure the complexity of learning classes (Jain *et al.*, 1999b). We obtain the following theorem.

finite learning

one shot learning

$$\begin{tikzcd}
\textsc{Inf}(\mathcal{F}) \arrow[r, "\mathbf{M}"] \arrow[d, "\eta_{\mathrm{INF}}"'] & \mathcal{H}^{\omega} \arrow[d, "\kappa \circ \lim_{\mathcal{H}}"] \\
\mathcal{F} \arrow[r, "\mathrm{id}_{\mathcal{F}}"'] & \mathcal{F}_{\mathrm{D}}
\end{tikzcd}
\qquad
\begin{tikzcd}
\textsc{Inf}(\mathcal{K}^*) \arrow[r, "\mathbf{M}"] \arrow[d, "\eta_{\mathrm{INF}}"'] & \mathcal{H}^{\omega} \arrow[d, "\gamma \equiv \kappa_{\mathrm{H}}"] \\
\mathcal{K}^* \arrow[r, "\mathrm{id}"'] & \mathcal{K}^*
\end{tikzcd}$$

Figure 2.5 | The commutative diagram representing **FigEx-Inf**-learning of $\mathcal{F}$ (left), and **FigEfEx-Inf**-learning of $\mathcal{F}$ (both left and right). In this diagram, $\textsc{Inf}(\mathcal{F})$ denotes the set of informants of $K \in \mathcal{F}$.

Theorem 2.43
A set $\mathcal{F} \subseteq \mathcal{K}^*$ is **FigFin-Inf**-*learnable (resp.* **FigFin-Txt**-*learnable) if and only if the identity* $\mathrm{id}_{\mathcal{F}}$ *is $(\eta_{\mathrm{INF}}, \kappa)$-computable (resp. $(\eta_{\mathrm{TXT}}, \kappa)$-computable).*

Proof. We only prove the case of **FigFin-Inf**-learning, since we can prove the case of **FigFin-Txt**-learning in exactly the same way.

The "only if" part: There is a learner **M** that **FigFin-Inf**-learns $\mathcal{F}$, hence for all $K \in \mathcal{F}$ and all $\sigma \in \mathrm{dom}(\eta_{\mathrm{INF}})$ of K, we can assume that $\mathbf{M}_\sigma = H$ such that $\kappa(H) = K$. Thus we have

$$\mathrm{id}_{\mathcal{F}} \circ \eta_{\mathrm{INF}}(\sigma) = \kappa(\mathbf{M}_\sigma). \tag{2.12}$$

This means that $\mathrm{id}_{\mathcal{F}}$ is $(\eta_{\mathrm{INF}}, \kappa)$-computable.

The "if" part: For some **M**, the above equation (2.12) holds for all $\sigma \in \mathrm{dom}(\eta_{\mathrm{INF}})$. This means that **M** is a learner that **FigFin-Inf**-learns $\mathcal{F}$. □

Finally, we show a connection between effective learning of figures and the computability of figures. Since **FigEfEx-Txt** $= \emptyset$ (Theorem 2.31), we only treat learning from informants. We define the representation $\gamma :\subseteq \mathcal{H}^{\omega} \to \mathcal{K}^*$ by $\gamma(p) :=$ K if $p = H_0, H_1, \ldots$ such that $H_i \in \mathcal{H}$ and $d_{\mathrm{H}}(K, \kappa(H_i)) \leq 2^{-i}$ for all $i \in \mathbb{N}$.

Lemma 2.44
$\gamma \equiv \kappa_{\mathrm{H}}$.

Proof. First we prove $\gamma \leq \kappa_{\mathrm{H}}$. For the function $g : \mathbb{N} \to \mathbb{R}$ such that

$$g(i) = \lceil i + \log_2 \sqrt{d}\, \rceil,$$

we have $\mathrm{diam}(g(i)) = \sqrt{d} \cdot 2^{-g(i)} \leq 2^{-i}$ for all $i \in \mathbb{N}$. Thus there exists a computable function f such that, for all $p \in \mathrm{dom}(\gamma)$, $f(p)$ is a representation of κ_{H} since, for an infinite sequence of hypotheses $p = H_0, H_1, \ldots$, all f has to do is to generate an infinite sequence $q = w_0 \| w_1 \| w_2 \| \ldots$ such that $w_i = \iota(H_{g(i)}^{g(i)})$ for all $i \in \mathbb{N}$, which results in

$$d_{\mathrm{H}}(K, \nu_Q(w_i)) \leq \mathrm{diam}(g(i)) = \sqrt{d} \cdot 2^{-g(i)} \leq 2^{-i}$$

for all $i \in \mathbb{N}$.

Next, we prove $\kappa_H \leq \gamma$. Fix $q \in \text{dom}(\kappa_H)$ with $q = w_0 \| w_1 \| \dots$. For each $i \in \mathbb{N}$, let $w_i = \iota(w_{i,0}, w_{i,1}, \dots, w_{i,n})$. Then the set $\{w_{i,0}, \dots, w_{i,n}\}$, which we denote H_i, becomes a hypothesis. From the definition of κ_H,

$$d_H(K, \kappa(H_i)) \leq 2^{-i}$$

holds for all $i \in \mathbb{N}$. This means that, for the sequence $p = w_0, w_1, \dots, \gamma(p) = K$. We therefore have $\gamma \equiv \kappa_H$. □

By using this lemma, we interpret effective learning of figures as the computability of two identities (Figure 2.5).

Theorem 2.45
A set $\mathcal{F} \subseteq \mathcal{K}^*$ is **FigEfEx-Inf**-*learnable if and only if there exists a computable function f such that f is a $(\eta_{\text{INF}}, \kappa \circ \lim_{\mathcal{H}})$-realization of the identity* $\text{id}_{\mathcal{F}}$*, and f is also a $(\eta_{\text{INF}}, \gamma)$-realization of the identity* $\text{id} : \mathcal{K}^* \to \mathcal{K}^*$.

Proof. We prove the latter half of the theorem, since the former part can be proved exactly as for Theorem 2.42.

The "only if" part: We assume that a learner **M** **FigEfEx-Inf**-learns $\mathcal{F}$. For all $K \in \mathcal{K}^*$ and all $\sigma \in \text{dom}(\eta_{\text{INF}})$,

$$\text{id} \circ \eta_{\text{INF}}(\sigma) = \gamma(\mathbf{M}_\sigma)$$

holds since the identity id is $(\eta_{\text{INF}}, \gamma)$-computable.

The "if" part: For some **M**, $\text{id} \circ \eta_{\text{INF}}(\sigma) = \gamma(\mathbf{M}_\sigma)$ for all $\sigma \in \text{dom}(\eta_{\text{INF}})$. It follows that **M** is a learner that **FigEfEx-Inf**-learns $\mathcal{F}$. □

Thus in **FigEfEx-Inf**- and **FigEfEx-Txt**-learning of a set of figures $\mathcal{F}$, a learner **M** outputs a hypothesis P with $\kappa(P) = K$ in finite time if $K \in \mathcal{F}$, and **M** outputs the "standard" representation of K if $K \in \mathcal{K}^* \setminus \mathcal{F}$ since we prove that $\gamma \equiv \kappa_H$ in Lemma 2.44. Informally, this means that there is not too much loss of information of figures even if they are not explanatory learnable.

2.7 Summary

We have proposed the learning of figures using self-similar sets based on the Gold-style learning model towards a new theoretical framework of binary classification focusing on computability, and demonstrated a learnability hierarchy under various learning criteria (Figure 2.3). The key to the computable approach is amalgamation of discretization of data and the learning process. We showed a novel mathematical connection between fractal geometry and the Gold-style learning model by measuring the lower bound of the size of training data with the Hausdorff dimension and the VC dimension. Furthermore, we analyzed our learning model using TTE (Type-2 Theory of Effectivity) and presented several mathematical connections between computability and learnability.

Although we theoretically showed that fractals (self-similar sets) can be effectively used as a representation system for learning of figures, there may be more efficient systems in terms of time and space complexities. To examine learning under such systems, our framework can be used as a basis.

In these days, most methods in machine learning are based on a statistical approach (Bishop, 2007). The reason is that many data in the real world are in analog (real-valued) form, and the statistical approach can treat such analog data directly in theory. However, all learning methods are performed on computers. This means that all machine learning algorithms actually treat discretized digital data and, now, most research pays no attention to the gap between analog and digital data. In this chapter we have proposed a novel and completely computable learning method for analog data, and have analyzed the method precisely. This work will be a theoretical foundation for computable learning from analog data, such as classification, regression, and clustering.

Part II

From Theory to Practice

“the whole Universe is made up of these two classes together”
— George Boole, *An Investigation of the Laws of Thought*

3

CODING DIVERGENCE

coding divergence

The aim of this chapter is to give a practical learning method for continuous data in a computational manner started from theoretical analysis. We integrate two fundamental processes: *discretization* of continuous data and learning of a model that explains them. In particular, we treat the problem of measuring the difference between two sets of real-valued data, since such a problem is basic for a lot of tasks in machine learning, *e.g.*, classification and clustering. We propose a novel measure, called *coding divergence*, to measure similarity between such sets, and construct a classifier using it. The key concept of the divergence is the simple procedure *separation*: The coding divergence measures the complexity of separating a positive dataset from a negative dataset, which corresponds to the height of a decision tree separating them, intuitively.

Motivation

negative control
positive control

In experimental science, a lot of experiments are designed including a *negative control* and a *positive control*. Positive controls confirm that the procedure is competent for observing the effect, and negative controls confirm that the procedure is not observing an unrelated effect. Testing the effect of a treatment group is the objective of such an experiment. A typical situation is to test the effect of a new drug, where the result of applying the new drug (treatment group) is compared against placebo (negative control).

statistical hypothesis test

The standard method for the above task is a *statistical hypothesis test*, which has developed from the works by Neyman and Pearson (1928, 1933) and Fisher (1925, 1956). However, it is well known that there exist many fundamental problems in statistical hypothesis testing such as non-verifiable assumptions for populations and arbitrary p values (Johnson, 1999) when we apply such methods to actual datasets. As a result, even in the top journals such as *Nature*, *Science*, and *Cell*, we can easily find inappropriate usage of statistical hypothesis testing[1]. Thus an alternative method to analyze experimental results is required.

All the scientists have to do is to judge to which controls a treatment group belongs and, obviously, this typical task in experimental science can be viewed as

[1] *Nature* says "please help us" to deal with various statistical methods appropriately (`http://www.nature.com/nature/authors/gta/`).

classification in machine learning, that is, a pair of negative and positive controls is given as a training dataset, and a classifier labels a treatment group (corresponds to a test set) “negative” or “positive” using the training set. Note that labels of all elements in the test set are same. Therefore, comparing the similarity between the negative control and the test set to the one between the positive control and the test set is a direct way, and the coding divergence, which will be introduced in this chapter, can achieve such task.

classification

Note that lots of existing classification methods based on statistics could not work well, since most of them have been proposed for *large* datasets, while typical datasets are *small* in experimental science such as physiology. Moreover, most classification methods considered in the machine learning community usually classify each test datum, whereas we have to classify a set of test data in controlled experiments, and the coding divergence is designed for the task.

Contributions

The *coding divergence*, proposed in this chapter, uses no statistics and no probabilistic models, and purely depends on the topological structure (in the sense of mathematics) of the *Cantor space* Σ^ω, which is known as the standard topological space of the set of infinite sequences (Weihrauch, 2000). Thus, for example, it has no assumptions for the probability distributions of datasets. This property enables us to develop a robust machine learning algorithm for various datasets.

We identify each real-valued datum with a real vector in the d-dimensional Euclidean space $\mathbb{R}^d$, and realize discretization of a real-valued datum through an topological mapping from $\mathbb{R}^d$ into the Cantor space Σ^ω. A topology uses *open sets* to axiomatize the notion of approximation, and it enables us to treat each discretized datum as a base element of an open set of the Cantor space.

open set

The simplest model, which is consistent with the given two sets, is learned in the Cantor space with assuming that one set is a set of positive examples and the other set is a set of negative examples, *i.e.*, the learned model explains all positive examples and does not explain any negative examples. This model corresponds to the *minimum open set* in the Cantor space Σ^ω. The coding divergence is obtained from the length of the code that encodes the learned model in the Cantor space.

minimum open set

Figure 3.1 shows two examples of computing the *binary-coding divergence* (the coding divergence with the binary encoding) where each datum is a vector in $\mathbb{R}^2$. The unit cube $\mathcal{I} = [0,1] \times [0,1]$ is partitioned recursively, and each partitioned cube is encoded by the binary encoding (see Section 3.2.2 and Figure 3.3). In the left panel (Figure 3.1a), to separate one set from the other set, we need only one partition in each dimension, and need $2/10$ symbols par datum. In contrast in the right panel (Figure 3.1b), three recursive partitions are needed in each dimension, and we need $26/10$ symbols par datum. The binary-coding divergence is determined directly from these numbers of symbols, that is, $2/10+2/10 = 0.4$ in Figure 3.1a and $26/10 + 26/10 = 5.2$ in Figure 3.1b.

binary-coding divergence

This chapter is organized as follows: we give mathematical background in Section 3.2. Section 3.3 is the main part in this chapter, where we introduce the coding divergence and a classifier using the divergence. We analyze performance of the classifier experimentally in Section 3.4. Section 3.5 summarizes this chapter.

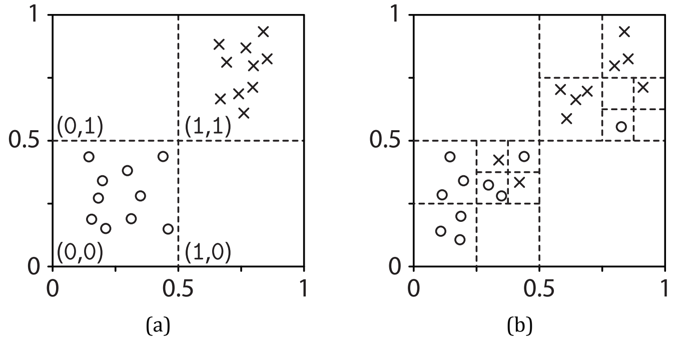


Figure 3.1 | Two examples of computing the binary-coding divergence. Each ○ and × denotes a real-valued datum, and each cube is encoded by the binary encoding (same as the base-2 embedding in Definition 3.3). In the left panel (a), we need a pair of codes (0,0) (encoding [0, 1/2] × [0, 1/2]) to separate ○ from ×, and (1,1) (encoding [1/2, 1] × [1/2, 1]) to separate × from ○, hence we can separate them with 2/10 symbols par datum, and the binary-coding divergence is $2/10 + 2/10 = 0.4$ (For each pair of codes, we ignore symbols "(", ",", and ")", and actually tupled into one code. See the equation (2.4)). In contrast in the right panel (b), we need codes (00,00), (00,01), (010,010), (011,011), and (110,100) to separate ○ from ×, and (10,10), (11,11), (111,101), (010,011), and (011,010) to separate × from ○. Thus they are separated by $(4 + 4 + 6 + 6 + 6)/10$ symbols par datum, and the divergence is $26/10 + 26/10 = 5.2$.

3.1 Related Work

Liu *et al.* (2008) used the similar idea of our divergence to anomaly detection. They constructed decision trees by partitioning intervals randomly and recursively, and used the height of them to measure the complexity of separating each datum. Compared to the method, our contribution is as follows: We treat the way of partition as the process of discretization and formulate it as a mapping (encoding) of $\mathbb{R}^d$ into the Cantor space Σ^ω, and show that the height of a decision tree corresponds to the size of an open set in the topological space Σ^ω. This gives the theoretical justification for the process "partition". Furthermore, this is the first time to perform classification using such idea.

Discretization is also used to construct decision trees from continuous data. For example, C4.5 (Quinlan, 1996) is one of famous algorithms to develop decision trees together with discretization. Our approach is different from them since we realize discretization process by methods of encoding real numbers, give theoretical support by modern mathematical theories in Computable Analysis, and integrate the process into computational learning.

Kernel methods including Support Vector Machines (SVMs) are known to be one of the most famous and powerful classification methods (Bishop, 2007), where any form of data, for example sequences, images, and graphs, are mapped into a high dimensional feature space, which is the d-dimensional Euclidean space $\mathbb{R}^d$, or more generally the infinite-dimensional Hilbert space, to measure the similarity between each pair of data. In contrast, our strategy is inverted: Every real-valued datum in $\mathbb{R}^d$ is mapped into the Cantor space Σ^ω. The "discrete" space Σ^ω might seem to be strange as a feature space, but is natural for machine learning from real-valued data with discretization.

Some studies use the Kolmogorov complexity for machine learning such as clustering (Cilibrasi and Vitányi, 2005; Li *et al.*, 2003) by measuring the similarity between a pair of data. Compared to the methods, our approach has mainly two advantages as follows: First, the coding divergence is computable (Section 3.3.3) whereas the Kolmogorov complexity is not, hence they use actual compression algorithms such as `gzip` to obtain an approximate value of the Kolmogorov complexity, which result in a gap between theory and practice; second, our approach is more general than theirs, because we can treat any continuous objects through encoding of them, however they cannot since they do not consider encoding process of objects.

3.2 Mathematical Background

We introduce our mathematical background using the framework of Type-2 Theory of Effectivity (TTE) studied in the area of computable analysis (Weihrauch, 2000). Remember that $\uparrow w$ denotes the set $\{ p \in \Sigma^\omega \mid w \sqsubseteq p \}$ and, in addition, we denote by $\uparrow W$ the set $\{ p \in \Sigma^\omega \mid w \sqsubseteq p$ for some $w \in W \}$ for a set of finite sequences $W \subseteq \Sigma^*$. The *size* of W is defined by $|W| := \sum_{w \in W} |w|$. For example, if $W = \{11, 0, 100\}$, then $|W| = 2 + 1 + 3 = 6$.

3.2.1 The Cantor Space

First, we introduce the *Cantor topology* into the set of infinite sequences Σ^ω, which is known as the standard topology on it (Weihrauch, 2000).

Cantor topology

Definition 3.1: The Cantor topology
Define $\tau_{\Sigma^\omega} := \{ \uparrow W \mid W \subseteq \Sigma^* \}$. We say that τ_{Σ^ω} is the *Cantor topology* over Σ, and the topological space $(\Sigma^\omega, \tau_{\Sigma^\omega})$ is the *Cantor space*.

Cantor space

We abbreviate $(\Sigma^\omega, \tau_{\Sigma^\omega})$ as Σ^ω if τ_{Σ^ω} is understood from the context. Then, the set $\{ \uparrow w \mid w \in \Sigma^* \}$ becomes a *base* of the topology τ_{Σ^ω}.

Example 3.2
Let an alphabet $\Sigma = \{0, 1\}$. For example, the set $\uparrow 00 = \{ p \in \Sigma^\omega \mid 00 \sqsubseteq p \}$ is a base element of the Cantor space Σ^ω, and $\uparrow 10$ is also a base element. Of course, both sets $\uparrow 00$ and $\uparrow 10$ are open sets in the space Σ^ω, *i.e.*, $\uparrow 00 \in \tau_{\Sigma^\omega}$ and $\uparrow 10 \in \tau_{\Sigma^\omega}$. Assume $W = \{00, 10\}$. Then the set $\uparrow W = \{ p \in \Sigma^\omega \mid 00 \sqsubseteq p$ or $10 \sqsubseteq p \}$ is an open set. Note that $\uparrow W = 00\Sigma^\omega \cup 10\Sigma^\omega$. □

The Cantor space Σ^ω can be visualized by a tree, where each infinite sequence $p \in \Sigma^\omega$ corresponds to an infinite descending path from the root (Figure 3.2). Then a base element $\uparrow w$ is a full subtree and an open set is a union of full subtrees, and the length $|w|$ corresponds to the depth of the root of the subtree.

3.2.2 Embedding the Euclidean Space into the Cantor Space

Let us approach to the d-dimensional Euclidean space $\mathbb{R}^d$ through the Cantor space Σ^ω using topological mappings between $\mathbb{R}^d$ and Σ^ω.

We say that a surjective function $\rho :\subseteq \Sigma^\omega \to \mathbb{R}^d$ is a *representation* of $\mathbb{R}^d$, and an injective function $\varphi :\subseteq \mathbb{R}^d \to \Sigma^\omega$ is an *embedding* of $\mathbb{R}^d$, meaning that repre-

representation

embedding

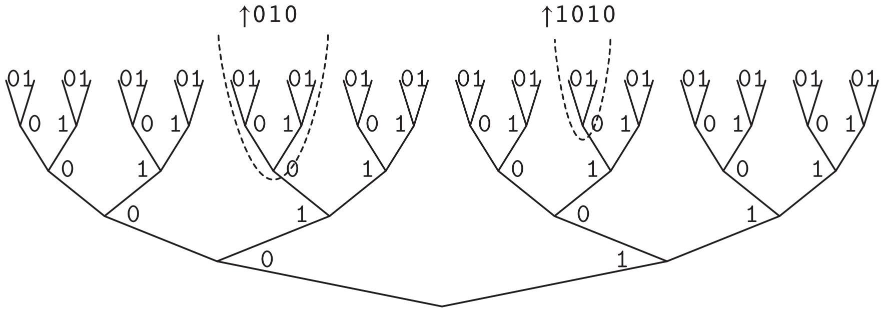


Figure 3.2 | Tree representation of the Cantor space over Σ= {0, 1}. Subtrees ↑010 and ↑1010 are base elements, and the set ↑010 ∪ ↑1010 is open.

sentations and embeddings are dual concepts. In the previous chapter, the binary representation was used as the canonical representation of real numbers. In the field of computable analysis, computability of $\mathbb{R}^d$ or other continuous objects (*i.e.*, uncountable sets) are treated through representations with Σ^ω (Schröder, 2002a; Weihrauch, 2000), and here we apply this theoretical framework to machine learning. For simplicity, we turn an embedding φ into a bijective function by replacing the codomain Σ^ω by its actual image $\varphi(\mathbb{R}^d)$, and assume that a representation $\rho = \varphi^{-1}$.

When we identify an element in $\mathbb{R}^d$ with an object with analog quantity, an embedding φ can be viewed as a mathematical realization of an encoding method or an actual A/D converter, where a continuous signal (analog quantity) is converted to a sequence of bits (digital quantity). The true value x of the analog quantity is encoded as an infinite sequence $\varphi(x)$ by the embedding φ, and any prefix w of $\varphi(x)$ is a discretized digital value. The prefix w tells us x is in the interval $\varphi^{-1}(\uparrow w)$, and the width of the interval

$$\left|\,\varphi^{-1}(\uparrow w)\,\right| = \sup\left\{\, d(x,y) \;\middle|\; x, y \in \varphi^{-1}(\uparrow w)\,\right\}$$

(d is the Euclidean metric) corresponds to an error of w. This modeling reflects the fundamental property of observation together with discretization, that is, every (discretized) real-valued datum must have some error intrinsically (Baird, 1994).

We now define the standard embedding, the base-β embedding.

base-β embedding

Definition 3.3: Base-β embedding

Assume $d = 1$. The *base-β embedding* of the unit interval $\mathcal{I} = [0,1]$ is a mapping $\varphi_\beta : \mathcal{I} \to \Sigma^\omega$ that maps x to an infinite sequence p composed as follows: For all $i \in \mathbb{N}$, $x = 0$ implies $p_i = 0$, and $x \neq 0$ implies

$$p_i = \begin{cases} 0 & \text{if } z < x \leq z + \beta^{-(i+1)}, \\ 1 & \text{if } z + \beta^{-(i+1)} < x \leq z + \beta^{-(i+1)+1}, \\ \dots & \\ \beta - 1 & \text{if } z + \beta^{-(i+1)+(\beta-2)} < x \leq z + \beta^{-(i+1)+(\beta-1)}, \end{cases}$$

where

$$z = \sum_{j=0}^{i-1} p_j \beta^{-(j+1)}$$

if $i \neq 0$, and $z = 0$ otherwise.

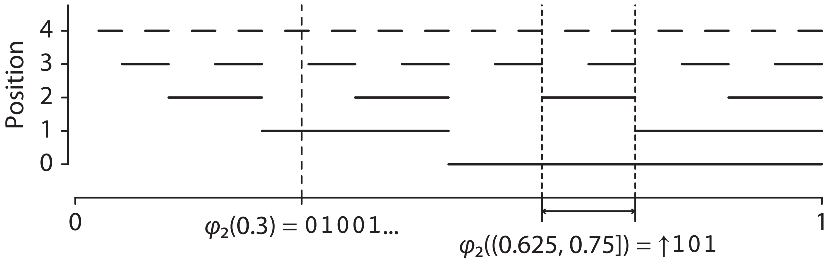

Position
0
1
2
3
4
0
1
$\varphi_2(0.3) = 01001...$
$\varphi_2((0.625, 0.75]) = \uparrow 101$

Figure 3.3 | The (one-dimensional) binary embedding φ_2. The position i is 1 if it is on the line, and 0 otherwise.

Especially, we call the base-2 embedding the *binary embedding*. Figure 3.3 denotes the binary embedding φ_2, where each horizontal line means that the corresponding position is 1 on the line and 0 otherwise. For example, let $p = \varphi_2(0.3)$. Then $p_0 = 0$ since the position 0 is not on the line, and $p_1 = 1$ since the position 1 is on the line, and so on.

binary embedding

By using the tupling function (see Equation (2.4)), we define the *d-dimensional base-β embedding* φ_β^d from $\mathcal{I} \subset \mathbb{R}^d$ to Σ^ω by

d-dimensional base-β embedding

$$\varphi_\beta^d(x_1, x_2, \dots, x_d) \coloneqq \langle \varphi_\beta(x_1), \varphi_\beta(x_2), \dots, \varphi_\beta(x_d) \rangle. \tag{3.1}$$

Example 3.4
For the binary embedding φ_2, we have $\varphi_2(0.3) = 010\dots$ and $\varphi_2(0.5) = 011\dots$. Thus $\varphi_2^2(0.3, 0.5) = \langle 010\dots, 011\dots \rangle = 001101\dots$. □

We abbreviate d of φ_β^d if it is understood from the context.

3.3 Coding Divergence

We give a novel measure of the difference between sets in the d-dimensional Euclidean space $\mathbb{R}^d$, called the *coding divergence*[1], with assuming that an element in $\mathbb{R}^d$ corresponds to a numerical (real-valued) data point. This divergence is the main contribution in this chapter. We design a classifier using it in Subsection 3.3.2, and construct a learner that learns the divergence in Subsection 3.3.3.

coding divergence

3.3.1 Definition and Properties

Let X and Y be a pair of nonempty finite sets in the unit interval $\mathcal{I} \subset \mathbb{R}^d$. We encode them by an embedding φ from $\mathcal{I}$ to the Cantor space Σ^ω.

We set a *model* of given data as an *open set* in the Cantor space, since this property "open" is desirable for machine learning as follows: For a set P, if P is open, then there exists a set of finite sequences W such that $\uparrow W = P$ (remember that $W\Sigma^\omega = \{ p \in \Sigma^\omega \mid w \sqsubseteq p \text{ for some } w \in W \}$). This means that P is *finitely observable* (Smyth, 1992), that is, for any infinite sequence p, we can decide whether or not $p \in P$ by observing just some prefix of p. Thus from sets of infinite sequences $\varphi(X)$ and $\varphi(Y)$, we can obtain an open set as a model of them in *finite time*

model

open set

We say that a set of infinite sequences $P \subseteq \Sigma^\omega$ is *consistent* with an ordered pair $(\varphi(X), \varphi(Y))$ if $P \supseteq \varphi(X)$ and $P \cap \varphi(Y) = \emptyset$, hence if we see $\varphi(X)$ and $\varphi(Y)$ as

consistent

[1] The usage of the word "divergence" follows the original definition of the Kullback-Leibler divergence (Kullback and Leibler, 1951).

positive and negative examples, respectively, then the set P "explains" all elements in $\varphi(X)$ and does not explain any elements in $\varphi(Y)$.

Definition 3.5: Coding divergence
Given an embedding $\varphi : \mathcal{J} \to \Sigma^{\omega}$. For a pair of nonempty finite sets $X, Y \subset \mathcal{J}$, define the *coding divergence with respect to φ* by

coding divergence

$$C_{\varphi}(X, Y) := \begin{cases} \infty & \text{if } X \cap Y \neq \emptyset, \\ D_{\varphi}(X; Y) + D_{\varphi}(Y; X) & \text{otherwise,} \end{cases}$$

where D_{φ} is the *directed coding divergence with respect to φ* defined by

directed coding divergence

$$D_{\varphi}(X; Y) := \frac{1}{\#X} \min \{\, |O| \mid O \text{ is open, and consistent with } (\varphi(X), \varphi(Y)) \,\}.$$

The standard embedding for the coding divergence is the base-β embedding φ_{β} and, especially, the coding divergence with respect to φ_{β} is written by $C_{\beta}(X, Y)$. If $\beta = 2$, we call $C_2(X, Y)$ the *binary-coding divergence*.

binary-coding divergence

Intuitively, the procedure to obtain the directed coding divergence is as follows: We encode given sets X, Y into the Cantor space Σ^{ω} by an embedding φ, find the simplest model (minimum open set) O that is consistent with an ordered pair of sets of infinite sequences $(\varphi(X), \varphi(Y))$, and measure the size $|O|$. Thus the directed coding divergence $D_{\varphi}(X; Y)$ can be viewed as the result of *consistent learning* (cf. Subsection 2.3.2) from positive examples $\varphi(X)$ and negative examples $\varphi(Y)$ in the computational learning theory context (Jain *et al.*, 1999b).

consistent learning

Example 3.6
Suppose that $X = \{\sqrt{0.2}, 0.8\}$ and $Y = \{\pi / 10\}$. Then in the binary embedding φ_2 (Definition 3.3), these sets are encoded into infinite sequences as follows: $\varphi_2(X) = \{011\ldots, 111\ldots\}$ and $\varphi_2(Y) = \{010\ldots\}$. Thus for an open set $\uparrow V$ with $V = \{011, 1\}$, $\uparrow V$ is minimum and consistent with $(\varphi_2(X), \varphi_2(Y))$. Therefore we have $D_2(X; Y) = (3 + 1)/2 = 2$. Similarly, $\uparrow W$ with $W = \{010\}$ is minimum and consistent with $(\varphi_2(Y), \varphi_2(X))$, hence $D_2(Y; X) = 3/1 = 3$. Consequently, $C_2(X, Y) = D_2(X; Y) + D_2(Y; X) = 5$. □

It is trivial that the coding divergence is not a metric since $C_{\varphi}(X, Y) \neq 0$ for all nonempty finite sets $X, Y \subset \mathcal{J}$.

Lemma 3.7
The coding divergence satisfies the following conditions:

1. $C_{\varphi}(X, Y) > 0$.
2. $C_{\varphi}(X, Y) = C_{\varphi}(Y, X)$.

Furthermore, it does not satisfy the triangle inequality.

Proof. The conditions $C_{\varphi}(X, Y) > 0$ and $C_{\varphi}(X, Y) = C_{\varphi}(Y, X)$ are proved directly from the definition. We can easily find an example, where the triangle inequality does not hold. For example, let $X = \{0.1\}$, $Y = \{0.8\}$, and $Z = \{0.2\}$, and assume that we use the binary embedding. We have $C_2(X, Y) = 2$, $C_2(Y, Z) = 2$, and $C_2(Z, X) = 6$, thus $C_2(X, Y) + C_2(Y, Z) < C_2(Z, X)$. □

3.3.2 Classification Using Coding Divergence

We construct a *lazy learner* that classifies a given dataset using the coding divergence. It classifies a test set by measuring its similarity (coding divergence) to a training dataset. It is quite simple and has no parameters, hence we can apply it to any type of real-valued datasets without any consideration. We will show its robustness in Section 3.4.

lazy learner

Assume that there are two classes A and B, and a pair (X, Y) is given as a training dataset (X and Y are nonempty finite sets in $\mathbb{R}^d$), where X (resp. Y) belongs to A (resp. B). Moreover, suppose that we have a dataset $Z \subset \mathbb{R}^d$ in which every datum is unlabeled, and we just know that all labels of elements in Z are same.

The classifier performs as follows: First, it computes $C_\varphi(X, Z)$ and $C_\varphi(Y, Z)$, and next, judges

$$Z \text{ belongs to the class} \begin{cases} A & \text{if } C_\varphi(X, Z) > C_\varphi(Y, Z), \\ B & \text{otherwise.} \end{cases}$$

Generally, the computability of $C_\varphi(X, Z)$ and $C_\varphi(Y, Z)$ is not guaranteed. However, if we use the base-β embedding φ_β such as the binary embedding (Definition 3.3), then it is effectively computable. We give the learner ψ that learns the coding divergence with respect to φ_β in the next subsection.

3.3.3 Learning of Coding Divergence

Here we integrate discretization of encoded real-valued data (infinite sequences) and learning of the coding divergence with respect to φ_β, and construct a learner that learns the divergence $C_\beta(X, Y)$ from $\varphi_\beta(X)$ and $\varphi_\beta(Y)$.

Procedure 3.1 shows the learner ψ that learns the coding divergence $C_\beta(X, Y)$ from a pair of given sets of encoded infinite sequences $\varphi_\beta(X)$ and $\varphi_\beta(Y)$. It continues to discretize each sequence, obtains longer and longer prefixes, and generates approximate values of the $C_\beta(X, Y)$. For inputs $\varphi_\beta(X)$ and $\varphi_\beta(Y)$, the output of ψ (finite or infinite sequence) is denoted by $\psi(\varphi_\beta(X), \varphi_\beta(Y))$, hence for example $\psi(\varphi_\beta(X), \varphi_\beta(Y))(0)$ is the first output, $\psi(\varphi_\beta(X), \varphi_\beta(Y))(1)$ is the second output, and so on.

To learn the coding divergence, a learner has to judge whether or not an open set O is consistent with given sets $(\varphi_\beta(X), \varphi_\beta(Y))$, and we show that it is decidable in finite time.

> **Lemma 3.8**
> *For every open set O and every $P, Q \subseteq \Sigma^\omega$, it is decidable that whether or not O is consistent with (P, Q) in finite time.*

Proof. Let $O = W\Sigma^\omega$ and $k = \max_{w \in W} |w|$, and define $P[k] := \{\, p[k] \mid p \in P \,\}$ and $Q[k] := \{\, q[k] \mid q \in Q \,\}$. We show that we can check the consistency only using W, $P[k]$, and $Q[k]$ (thus it is decidable). The condition $O \supseteq P$ holds if and only if for all $x \in P[k]$, there exists $w \in W$ such that $w \sqsubseteq x$. Moreover, $O \cap Q = \emptyset$ if and only if $y \not\sqsubseteq w$ and $w \not\sqsubseteq y$ for all $w \in O$ and $y \in Q[k]$. □

If $X \cap Y = \emptyset$, then ψ halts in finite time, and the last output is exactly the same as $C_\beta(X, Y)$. Otherwise if $X \cap Y \neq \emptyset$, then ψ continues to output approximate values

Procedure 3.1: Learner ψ that learns $C_\beta(X, Y)$

```
Input: Pair $(\varphi_\beta(X), \varphi_\beta(Y))$    (both $X$ and $Y$ are nonempty finite sets in $\mathcal{J}$)
Output: Finite or infinite sequence converging to $C_\beta(X, Y)$

function MAIN($\varphi_\beta(X), \varphi_\beta(Y)$)
    LEARNING($\varphi_\beta(X), \varphi_\beta(Y), \emptyset, \emptyset, \#X, \#Y, 0$)

function LEARNING($P, Q, H_1, H_2, m, n, k$)
    $V \leftarrow$ DISCRETIZE($P, k$)
    $W \leftarrow$ DISCRETIZE($Q, k$)
    $H_1 \leftarrow H_1 \cup \{v \in V \mid v \notin W\}$
    $H_2 \leftarrow H_2 \cup \{w \in W \mid w \notin V\}$
    $P \leftarrow \{p \in P \mid p \notin H_1\Sigma^\omega\}$
    $Q \leftarrow \{q \in Q \mid q \notin H_2\Sigma^\omega\}$
    output $m^{-1} \sum_{v \in H_1} |v| + n^{-1} \sum_{w \in H_2} |w|$
    if $P = \emptyset$ and $Q = \emptyset$ then halt
    else return LEARNING($P, Q, H_1, H_2, m, n, k + 1$)

function DISCRETIZE($P, k$)
    return $\{p[n] \mid p \in P\}$, where $n = (k + 1)d - 1$
```

of the $C_\beta(X, Y) = \infty$ forever. We can easily prove that for all $i, j \in \mathbb{N}$ with $i < j$,

$$\psi(\varphi_\beta(X), \varphi_\beta(Y))(i) \le \psi(\varphi_\beta(X), \varphi_\beta(Y))(j), \text{ and}$$
$$\lim_{i \to \infty} \psi(\varphi_\beta(X), \varphi_\beta(Y))(i) = \infty.$$

effective
Type-2 machine

This means that we can obtain more and more accurate values of the coding divergence, even though we cannot obtain the exact value in finite time. This property corresponds to the *effective* computability realized by the *Type-2 machine*, where while a computer reads more and more precise information (longer and longer prefixes) of the input, it produces more and more accurate approximations of the result. This machine is a natural extension of the usual Turing machine, and used to introduce computability of continuous objects (Weihrauch, 2000). Furthermore, this property is an effective version of the *consistency* in statistical context.

Example 3.9
Let us consider the case in Figure 3.1, and assume that X and Y consist of all ○ and ×, respectively. Let $P = \varphi_2(X)$ and $Q = \varphi_2(Y)$. Every pair of codes is tupled into one code by the tupling function. In Figure 3.1a, DISCRETIZE$(P, 0)$ returns $\{00\}$ (corresponds to $(0, 0)$) and DISCRETIZE$(Q, 0)$ returns $\{11\}$ (corresponds to $(1, 1)$). Thus ψ outputs $2/10 + 2/10 = 0.4$ and halts. In Figure 3.1b, both DISCRETIZE$(P, 0)$ and DISCRETIZE$(Q, 0)$ return $\{00, 11\}$, and DISCRETIZE$(P, 1)$ returns $\{0000, 0001, 0011, 1110\}$, and DISCRETIZE$(Q, 1)$ returns $\{1100, 1111, 0011, 1110\}$. Thus $H_1 = \{0000, 0001\}$ and $H_2 = \{1100, 1111\}$. Furthermore, for the rest of data, DISCRETIZE performs similarly, and finite sequences 001100, 001111, 111000 and 111011, 001110, 001101 are added to H_1 and H_2, respectively. Thus ψ outputs $26/10 + 26/10 = 5.2$ and halts. □

3.4 Experiments

We evaluate the performance of the classifier given in Subsection 3.3.2 by experiments to analyze the coding divergence empirically. We compared accuracy of classification performed by the classifier obtained from sensitivity and specificity to those of other classification methods.

Theoretically, our classifier can be applied to actual analog data such as (raw) real-valued data and continuous signals. However, it is difficult to collect such type of datasets and apply our classifier to them directly. Thereby in the following we use just discretized real-valued data stored in databases.

3.4.1 Methods

First, we construct the learning algorithm **M** for empirical experiments that computes an approximate value of the coding divergence with respect to the base-β embedding and always halts in finite time (Algorithm 1). The algorithm **M** does not receive infinite sequences, but receives just finite sequences.

Define for each $i \in \mathbb{N}$,

$$\varphi_\beta(X)[i] := \left\{ w \;\middle|\; |w| = i \text{ and } p \in \uparrow w \text{ for some } p \in \varphi_\beta(X) \right\}.$$

The algorithm **M** is slightly different from the learner ψ, since it receives only finite sequences $\varphi_\beta(X)[i]$ and $\varphi_\beta(Y)[j]$. We can get the exact coding divergence (*i.e.*, $\mathbf{M}(\varphi_\beta(X)[i], \varphi_\beta(Y)[j]) = C_\beta(X, Y)$) in the usual situation where $\varphi_\beta(X)[i] \cap \varphi_\beta(Y)[j] = \emptyset$ holds. Moreover, the output of **M** (denoted by $\mathbf{M}(\varphi_\beta(X)[i], \varphi_\beta(Y)[j])$) has the *monotonicity* with respect to i and j: For all pairs of i, j and i', j' with $i \le i'$ and $j \le j'$, we have monotonicity

$$\begin{aligned} &\left| C_\beta(X, Y) - \mathbf{M}(\varphi_\beta(X)[i], \varphi_\beta(Y)[j]) \right| \\ \ge &\left| C_\beta(X, Y) - \mathbf{M}(\varphi_\beta(X)[i'], \varphi_\beta(Y)[j']) \right|, \end{aligned}$$

and

$$\lim_{i,j \to \infty} \left| C_\beta(X, Y) - \mathbf{M}(\varphi_\beta(X)[i], \varphi_\beta(Y)[j]) \right| = 0.$$

Thus, intuitively, if we obtain more and more accurate data (longer and longer sequences), then approximation of the coding divergence becomes better and better, meaning that **M** is an *effective* algorithm.

The computational complexity of **M** is $O(mnd)$, where m and n are the cardinality of X and Y, respectively, since in each level k, updating V and W (lines 7 and 8 in Algorithm 1) takes $O(mnd)$.

To treat data that are not in the unit interval $\mathcal{I}$, we use min-max normalization that maps a value x to x', where

$$x' = \frac{x - \min\{X \cup Y \cup Z\}}{\max\{X \cup Y \cup Z\} - \min\{X \cup Y \cup Z\}}.$$

The classifier with the above algorithm **M** was implemented in R version 2.12.1 (R Development Core Team, 2011). We tested the performance of it by evaluating accuracy, the standard error measure of classification (Han and Kamber, 2006).

```
Algorithm 3.2: Learning algorithm M that learns $C_\beta(X, Y)$

Input: Pair $(\varphi_\beta(X)[i], \varphi_\beta(Y)[j])$   $(X, Y \subset \mathcal{I}$ and $i, j \in \mathbb{N})$
Output: Approximate value of $C_\beta(X, Y)$

function MAIN($V, W$)
    $(D_1, D_2) \leftarrow$ LEARNING($V, W, 0, 0, 0, \min\{i, j\}$)
    return $D_1 \,/\, \#V + D_2 \,/\, \#W$

function LEARNING($V, W, D_1, D_2, k, k_{\mathrm{max}}$)
    $V_{\mathrm{dis}} \leftarrow$ DISCRETIZE($V, k$)
    $W_{\mathrm{dis}} \leftarrow$ DISCRETIZE($W, k$)
    $V_{\mathrm{sep}} \leftarrow \{v \in V_{\mathrm{dis}} \mid v \notin W_{\mathrm{dis}}\}$
    $W_{\mathrm{sep}} \leftarrow \{w \in W_{\mathrm{dis}} \mid w \notin V_{\mathrm{dis}}\}$
    $D_1 \leftarrow D_1 + \sum_{v \in V_{\mathrm{sep}}} |v|$
    $D_2 \leftarrow D_2 + \sum_{w \in W_{\mathrm{sep}}} |w|$
    $V \leftarrow \{v \in V \mid v \notin V_{\mathrm{sep}}\Sigma^\omega\}$
    $W \leftarrow \{w \in W \mid w \notin W_{\mathrm{sep}}\Sigma^\omega\}$
    if $V = \emptyset$ and $W = \emptyset$ then return $(D_1, D_2)$
    else if $k = k_{\mathrm{max}}$ then return $(D_1 + n\#V, D_2 + n\#W)$
    else return LEARNING($P, Q, D_1, D_2, k + 1, k_{\mathrm{max}}$)

function DISCRETIZE($V, k$)   // $V \subset \Sigma^*$
    return $\{v[n] \mid v \in V\}$   $(n = (k + 1)d - 1)$
```

Ten datasets were collected from UCI Machine Learning Repository (Frank and Asuncion, 2010): *abalon*, *transfusion*, *sonar*, *glass*, *segmentation*, *ionosphere*, *madelon*, *magic*, *waveform*, and *yeast*. Every datum in each dataset is real-valued type and belongs to one of two classes (if there are more than two classes, we picked up just two classes). We used the size of each dataset 10 and 30, since one of applications is controlled experiments in life science, where such small sizes are typical.

We repeated the following procedure 10,000 times:

1. Sample d attributes (d is 1, 2, or 3),
2. collect n data for a training set X and for a test set T_+ from one class by independent random sampling without replacement, and Y and T_- from the other class, where $n = 10$ or 30,
3. Using X and Y, classify test sets T_+ and T_- by our method and other classification methods.

The binary-coding divergence was used throughout all experiments.

Let t_{pos} be the number of true positives, that is, the number of the case in which T_+ is classified correctly, and t_{neg} be the number of true negatives. We calculated the accuracy by $(t_{\mathrm{pos}} + t_{\mathrm{neg}})/20000$, since the sensitivity is $t_{\mathrm{pos}}/10000$ and the specificity is $t_{\mathrm{neg}}/10000$. For reference, we used SVM with the RBF kernel, SVM with the polynomial kernel, and k-nearest neighbor classifiers ($k = 1$ and 5). We used the function `ksvm` in the "kernlab" package for SVM (Karatzoglou *et al.*, 2004), and the function `knn` in the "class" package for the k-nearest neighbor classifiers, which have been implemented in R. Note that these methods classify each element in a test set, thereby we classified T_+ (or T_-) to the class in which greater number of elements in T_+ (or T_-) are classified.

3.4.2 Results and Discussions

The experimental result of accuracy is shown in Figure 3.4. Let us compare the result using the binary-coding divergence to those of other methods. In most cases, the accuracy of our classifier shows the highest value, and only in some cases other methods are more accurate than our method (*e.g.*, low dimensional datasets in *yeast*). This result shows the robustness of our classifier for various datasets. The simple process "separation by encoding" of our theoretical background might cause this robustness.

Moreover, results of other methods highly depend on kinds of datasets. For example, 1- and 5-nearest neighbor methods are better than SVMs in *ionosphere*, but are worse in *abalon*. This means that these methods require adjustment of parameters depending on datasets. However, our method has no parameters and does not need any adjustment. Thus we can apply our method effectively without special background knowledge about datasets.

3.5 Summary

In this chapter, we have proposed a novel measure of the difference between sets, called the coding divergence, and integrated discretization of analog data and learning of models that explains given data through computational learning of it, where the extracted model corresponds to the minimum open set in the Cantor space Σ^ω. The key idea is *separation* of intervals, which corresponds to realization of an encoding process of analog data by a mathematical embedding of the Euclidean space into the Cantor space. Furthermore, we have constructed a classifier, the lazy learner using the coding divergence, and shown the robust performance of it by empirical experiments.

Our mathematical framework is general and has possibility to develop further in the field of machine learning and data mining, since any type of datasets can be handled through an appropriate embedding from such objects to the Cantor space. This approach is new and, intuitively, opposite to the one using kernel functions.

For any other measures or metrics used in learning, the property of *invariance* is usually required to guarantee the soundness of them. Since the difference between two coding divergences is used for learning, providing the invariance of the difference with respect to geometric transformation, such as parallel translation, rotation, and other affine transformations, is an important future work.

Furthermore, since our proposed measure is quite simple, it can be applied to various tasks in machine learning and data mining. For example, we can measure the difference between clusters by the coding divergence. This means that we can perform hierarchical clustering using the coding divergence directly. Thus applying our measure to such tasks is a future work.

Another future work is an application for other actual datasets. For example, image matching is an important topic in computer vision, where each image is assumed to be a set of real vectors. One of well-known distances is the Hausdorff metric used in the previous chapter, and some metrics based on the Hausdorff

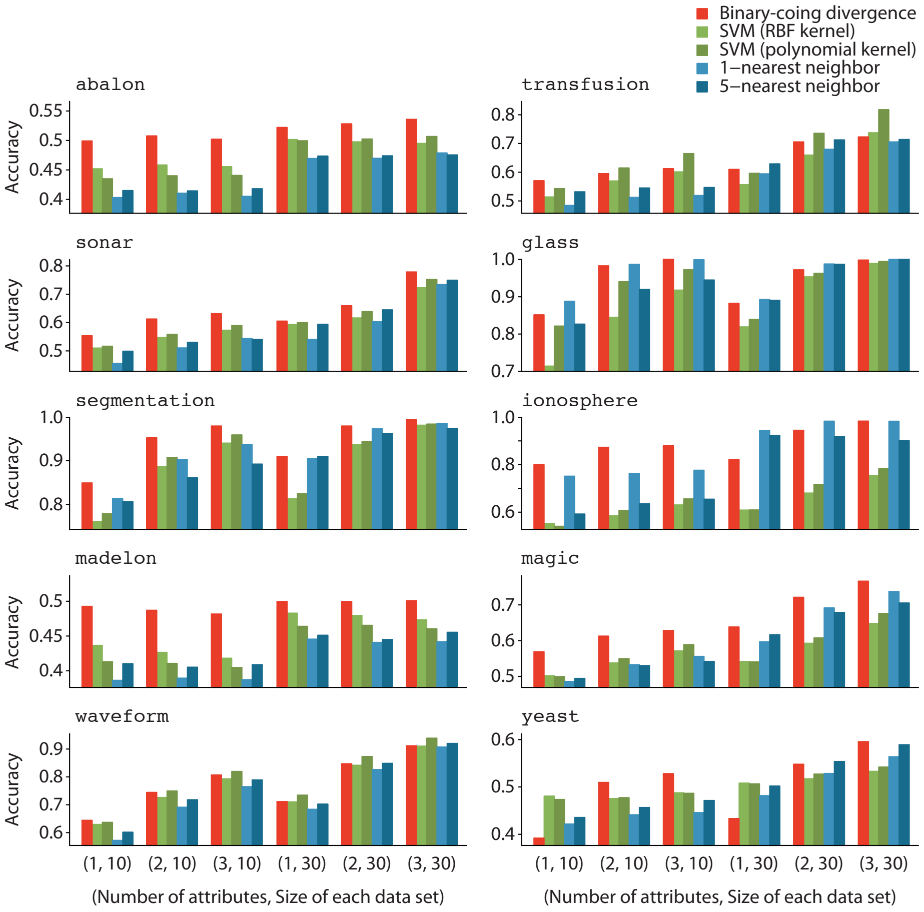


Figure 3.4 | Experimental results of accuracy using real-valued datasets collected from UCI repository. We applied our classifier (using the binary-coding divergence) and other four classification methods: SVM with the RBF kernel, SVM with the polynomial kernel, k-nearest-neighbor classifiers ($k = 1, 5$), to ten real-valued datasets: *abalon*, *transfusion*, *sonar*, *glass*, *segmentation*, *ionosphere*, *madelon*, *magic*, *waveform*, and *yeast*. We examined six cases: the number of used attributes are 1, 2, or 3, and the size of each sampled training/test datasets are 10 or 30.

metric have been proposed (Huttenlocher *et al.*, 1993; Zhao *et al.*, 2005). Trivially our method can be applied to such topics. In preliminary experiments, we have checked that the accuracy of classification using the coding divergence is better than that using the Hausdorff metric. Thus our method might be a better classifier than that with the Hausdorff metric.

3.6 Outlook: Data Stream Classification on Trees

data stream

In this section, we show our preliminary results as outlook of the coding divergence: classification of *data streams*. Machine learning from data streams (Babcock *et al.*, 2002), which rapidly grow without limit, is one of challenging problems and, to date, many techniques have been proposed (Aggarwal *et al.*, 2004; Aggarwal and Yu, 2010; Gaber *et al.*, 2005; Gama and Kosina, 2011). For instance, Domingos and Hulten (2000) proposed VFDT for data stream classification, which learns a decision tree whose size is kept small by using Hoeffding bounds. However, since the amount of data increasingly grows like the data deluge[2], a more scalable and effective algorithm is required.

trie

We construct an algorithm, called CODE (Classifier with the binary encODing on trEes), for classification of numerical data streams. It incrementally constructs a *trie*-like tree structure from received labeled data by discretizing them based on the *binary encoding scheme*. The coding divergence is used to measure the similarity between stored labeled data on a tree and an unlabeled datum. The succinct tree structure leads fast computation of the coding divergence, which can be applied to massive data streams.

3.6.1 CODE approach

We revise the coding divergence to evaluate the similarity between a set of stored labeled data points (training data) X and an unlabeled data point (test datum) x.

> **Definition 3.10: Similarity based on the binary encoding**
> Given a dataset X and a datum x, define
>
> $$s(x; X) := \frac{1}{\#X} \min \left\{ \omega_X(\uparrow W) \;\middle|\; \uparrow W \supseteq \varphi_2(X) \text{ and } \uparrow W \cap \{\varphi_2(x)\} = \emptyset \right\},$$
>
> where
>
> $$\omega_X(\uparrow w) := |w| \cdot \#(\uparrow w \cap \varphi_2(X)) \quad \text{and}$$
> $$\omega_X(\uparrow W) := \sum_{w \in W} \omega_X(\uparrow w).$$

Note that if we define $\omega_X(\uparrow w) := |w|$, $s(x; X)$ is exactly the same as the directed coding divergence $D(X; \{x\})$, thus the similarity $s(x; X)$ can be viewed as *weighted* coding divergence between x and X.

Next, we show that the similarity is effectively calculated using a trie-like data structure, which naturally extends the domain of data from $\mathcal{I}^d$ to $\mathbb{R}^d$. Procedure 3.3

[2] This phrase is from the special issue of *Science* (`http://www.sciencemag.org/site/special/data/`).

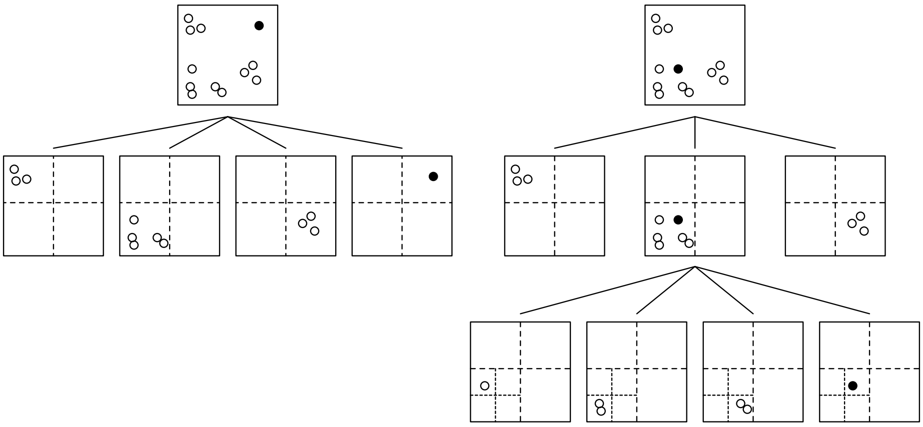


Figure 3.5 | Examples of calculating similarities. White and black points are labeled and unlabeled data, respectively. In the left-hand side, the similarity between labeled and unlabeled data is $((3 + 5 + 3) \cdot 2)/11 = 2$. In the right-hand side, the similarity is $((3 + 3) \cdot 2 + (1 + 2 + 2) \cdot 4)/11 = 32/11 = 2.91$.

constructs a tree $\mathcal{T}$ from labeled data, which corresponds to training phase for making a model, and computes the similarity between x and $\mathcal{T}$ if the input is unlabeled, which corresponds to test phase using the model. In any tree $\mathcal{T}$, we assume that every node v has three properties: $\mathrm{key}(v)$, $\mathrm{org}(v)$, and $\mathrm{num}(v)$, where $\mathrm{key}(v)$ is the key of v, $\mathrm{org}(v)$ is a value of an original datum pointed from v, and $\mathrm{num}(v)$ is the number of leaf nodes of v. Then, for any unlabeled x, the similarity between x and $\mathcal{T}$ computed by Procedure 3.3 is exactly the same as the similarity $s(x; X)$.

The binary encoding process of x is performed by the simple arithmetic operation $\lfloor x \cdot 2^k \rfloor$ at each discretization level k, and this operation is applicable to any real number in $\mathbb{R}$. We thus can apply our algorithm to any dataset, where each datum is a point in the d-dimensional Euclidean space $\mathbb{R}^d$. Figure 3.5 illustrates two examples of computing the similarity on trees.

Procedure 3.3 incrementally updates a tree $\mathcal{T}$ and its time complexity is $O(N + D)$, where $N = \max_{v \in \mathcal{T}} \#\{v' \in \mathcal{T} \mid v' \text{ is a child of } v\}$ and D is the maximum depth of $\mathcal{T}$. The number N grows exponentially when the dimension d increases. To escape from this "curse of dimensionality", we introduce the additional input parameter a and we do not add a node if $\sum_{j=1}^{d} |c^j - c'^j| \le a$ holds for some v's child v' such that $\mathrm{key}(v') = (c'^1, \ldots, c'^d)$ at the line 4 in Procedure 3.3.

Here we present the entire CODE algorithm in Procedure 3.4, which achieves multi-class classification from numerical data streams using the similarity defined above. Let $\mathcal{L} = \{l_1, l_2, \ldots, l_m\}$ be the domain of class labels. CODE makes m trees $\mathcal{T}_{l_1}, \mathcal{T}_{l_2}, \ldots, \mathcal{T}_{l_m}$ for respective labels using Procedure 3.3 from labeled data and, if it receives an unlabeled datum, it computes the similarity for each tree and classifies the datum to the class whose similarity is maximum.

3.6.2 Experiments

We experimentally evaluate CODE to determine its scalability and effectiveness.

Methods

We used Mac OS X version 10.6.5 with two 2.26-GHz Quad-Core Intel Xeon CPUs and 12 GB of memory. CODE was implemented in C and compiled with gcc 4.2.1, which can be used in R, version 2.12.2 (R Development Core Team, 2011).

Procedure 3.3: Construction of tree and calculation of the similarity

```
Input: Data point $x$ and tree $\mathcal{T}$
Output: Updated tree $\mathcal{T}$ or the similarity between $x$ and $\mathcal{T}$
function TREE($x, \mathcal{T}$)
$k \leftarrow 1, v \leftarrow$ the root of $\mathcal{T}$
repeat
    $(c^1, \dots, c^d) \leftarrow (\lfloor x^1 \cdot 2^k \rfloor, \dots, \lfloor x^d \cdot 2^k \rfloor)$
    if a node $v$ has a child $v'$ with $\mathrm{key}(v') = (c^1, \dots, c^d)$ then
        if $x$ is unlabeled then
            $s \leftarrow s + \sum\{\mathrm{num}(u) \cdot k \cdot d \mid u$ is a brother of $v\}$
        end if
        if a node $v'$ is a leaf then
            $y \leftarrow \mathrm{org}(v')$
            add a node $v''$ as a child of $v'$,
              where $\mathrm{key}(v'') = (\lfloor y^1 \cdot 2^k \rfloor, \dots, \lfloor y^d \cdot 2^k \rfloor)$ and $\mathrm{org}(v'') = y$
        end if
        $v \leftarrow v'$
    else
        if $x$ is labeled then
            add a $v$'s child $v'$, where $\mathrm{key}(v') = (c^1, \dots, c^d)$ and $\mathrm{org}(v') = x$
            return $\mathcal{T}$
        else if $x$ is unlabeled then
            return $(s + \mathrm{num}(v) \cdot k \cdot d)$ / the number of leaf nodes of $\mathcal{T}$
        end if
    end if
    $k \leftarrow k + 1$
until forever
```

To evaluate scalability with respect to the data size, synthetic data were generated randomly using the R `clusterGeneration` package (Qiu and Joe, 2006), where the parameter *sepVal* was 0.35, $d = 5$, and the number of data points $n = 1,000,000$. The number of classes was 5. We used $990,000$ data as labeled data to evaluate scalability in constructing trees and the remaining $10,000$ data as unlabeled test data to check accuracy of classification. Moreover, to check effectivity in classification, five real datasets were collected from the UCI machine learning repository (Frank and Asuncion, 2010): *iris*, *waveform*, *vertebral*, *wine*, and *yeast*, and we obtained accuracy by ten-fold cross validation.

We adopted two tree-based methods as control: one is the tree method implemented in the R `tree` package as a typical tree-based batch classification algorithm, and the other is VFDT (Domingos and Hulten, 2000) which is a representative fast tree-based classification algorithm for data streams. We used VFDT implemented as a module of the VFML toolkit (Hulten and Domingos, 2003). Notice that, since the tree method in R is a batch algorithm, we first stored all data and applied the method to them, whereas CODE and VFDT incrementally make (train) trees whenever labeled data arrive.

Procedure 3.4: CODE procedure

```
Input: Data stream $x_1, x_2, x_3, \ldots$
Output: Sequence of labels for each unlabeled datum
repeat
    read $x_i$
    if $x_i$ is labeled then
        $\mathcal{T}_l \leftarrow \textsc{Tree}(x_i, \mathcal{T}_l)$, where $x_i$'s label is $l$
    else if $x_i$ is unlabeled then
        output $\mathrm{argmax}_{l \in \mathcal{L}} \textsc{Tree}(x_i, \mathcal{T}_l)$
    end if
    $i \leftarrow i + 1$
until forever
```

Results and Discussion

Figure 3.6 shows results; running time and accuracy of classification, for synthetic data. Since the tree construction algorithm of CODE is simple, it is faster than the tree algorithm in R and VFDT, especially for large datasets. Moreover, accuracy of CODE is higher than VFDT when the data size is small. These results indicate that CODE is a scalable high-throughput algorithm and can be applied for classification of massive data stream.

In real UCI datasets, CODE is competitive compared to the tree method and VFDT (Figure 3.7) and shows the best performance in two datasets (*waveform* and *vertebral*). Thus CODE should work efficiently and effectively for various size of datasets including small and large datasets.

To summarize, CODE has been shown to be competitive compared to other algorithms in terms of scalability and effectiveness. We will continue to investigate the presented approach. To treat more complex data streams, such as a stream containing concept drift, is one of future works.

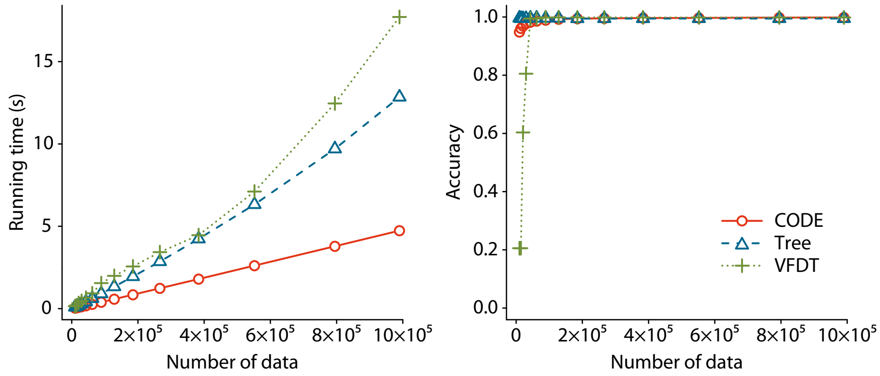


Figure 3.6 | Experimental results for synthetic data.

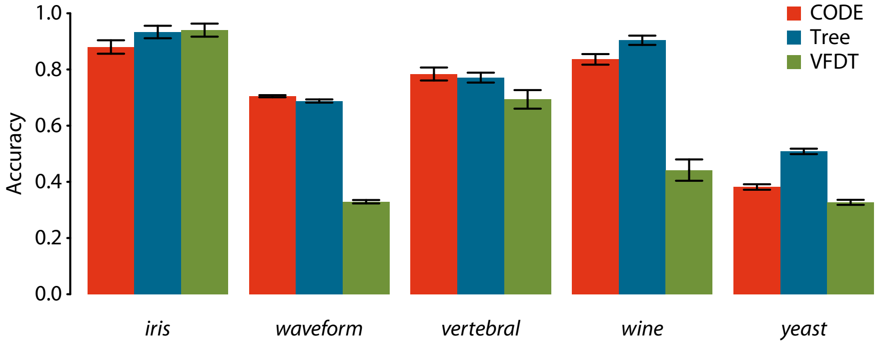


Figure 3.7 | Experimental results for real data. Data show mean ± s.e.m.

4

MINIMUM CODE LENGTH AND GRAY CODE FOR CLUSTERING

CLUSTERING is a fundamental task in data analysis, and many clustering algorithms have been developed in the fields of machine learning and knowledge discovery (Berkhin, 2006; Halkidi *et al.*, 2001; Jain *et al.*, 1999a). Several clustering algorithms have been recently proposed that focus on the *compression* of data.

compression

minimum description length (MDL)

Kontkanen *et al.* (2005) proposed the *minimum description length* (MDL) approach to clustering by taking advantage of an information theoretic framework. However, data encoding has to be optimized to find the best clusters; that is, all encoding schemes are considered within the clustering process under the MDL criterion. As a result, their approach takes quadratic time with respect to the data set size and can only be handled in practice using a stochastic algorithm (Kontkanen and Myllymäki, 2008). Cilibrasi and Vitányi (2005) proposed a clustering algorithm based on the *Kolmogorov complexity*. Since their method measures the distance between two data points on the basis of compression of finite sequences (*i.e.*, discrete variables), it is difficult to apply it to multivariate data of continuous variables. Moreover, although there are other approaches (Keogh *et al.*, 2007; Li *et al.*, 2001) that focus on compression of data, they perform simple agglomerative hierarchical clustering, so it takes quadratic time with respect to the data set size. These approaches are therefore not suitable for clustering massive data sets.

Kolmogorov complexity

minimum code length (MCL)

Here we propose a new measure, called the *minimum code length* (MCL), to score the quality of a given clustering result under a *fixed* encoding scheme. This use of fixed encoding enables the performance of fast (*i.e.*, linear complexity with respect to the data set size) and exact clustering since we do not need to optimize data encoding. We present a clustering algorithm, called COOL (COding-Oriented cLustering), that always finds the best clusters; *i.e.*, the globally optimal clusters which minimizes MCL, and requires only the lower bounds for the number and size of the clusters. The discretization of continuous variables with the fixed encoding scheme coincides with the clustering process itself — a hierarchy of clusters is introduced automatically by increasing the accuracy of discretization.

embedding

Mathematically, an encoding, or *embedding*, is a mapping from real numbers to infinite sequences over some alphabet (Tsuiki, 2002), and discretization is realized

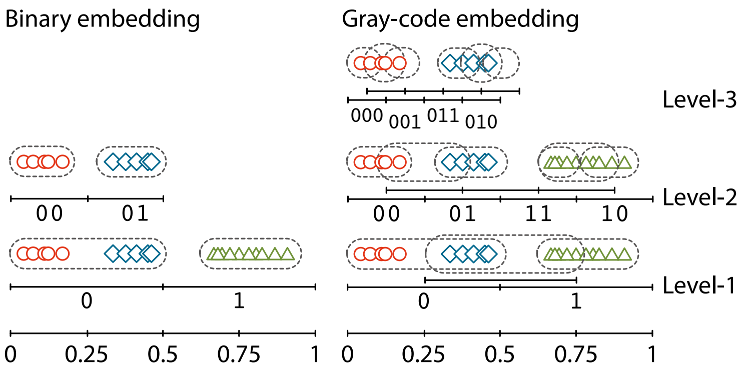


Figure 4.1 | Examples of computing MCL with binary (left) or Gray code (right) embedding. These one-dimensional data sets are in [0,1] and partitioned into three clusters, ○, ◊, and Δ. Level means the length of each prefix. With binary embedding, cluster Δ is separated at level 1, and ○ and ◊ are separated at level 2. They are encoded by 1, 00, and 01, respectively, so the MCL is 1 + 2 + 2 = 5. With Gray code embedding, the intervals overlap, and adjacent clusters are merged at each level. As a result, Δ is separated at level 2, and ○ and ◊ are separated at level 3. Their representatives are {11,10}, {000,001}, and {011,010}, respectively, so the MCL is 4 + 6 + 6 = 16.

by truncation of infinite sequences. For example, in the binary embedding φ_{B}, every real number in $[0, 1]$ is translated into an infinite sequence composed of $\mathtt{0}$ and $\mathtt{1}$; *e.g.*, $\varphi_{\mathrm{B}}(0) = \mathtt{000}\ldots$, $\varphi_{\mathrm{B}}(0.2) = \mathtt{001}\ldots$, and $\varphi_{\mathrm{B}}(0.4) = \mathtt{011}\ldots$, where the first bit is $\mathtt{0}$ if the value is in the interval $[0, 0.5]$ and $\mathtt{1}$ if in $(0.5, 1]$. If these sequences are truncated at the first bit, all of them become $\mathtt{0}$, and hence they are considered as in the same cluster since they cannot be distinguished. If they are then truncated at the second bit, both 0 and 0.2 become $\mathtt{00}$, and 0.4 becomes $\mathtt{01}$. Thus, two clusters are generated: $C_1 = \{0, 0.2\}$ and $C_2 = \{0.4\}$. This means that representatives of C_1 and C_2 are $\mathtt{00}$ and $\mathtt{01}$, respectively. Finally, if they are truncated at the third bit, 0 and 0.2 are separated. The hierarchy is therefore constructed as $\{\{0, 0.2, 0.4\}\}$, $\{\{0, 0.2\}, \{0.4\}\}$, and $\{\{0\}, \{0.2\}, \{0.4\}\}$.

The complexity of making clusters can be measured by the length of the cluster representatives. In the above example, $2 + 2 = 4$ for clusters $\{0, 0.2\}$ and $\{0.4\}$ ($\mathtt{00}$ and $\mathtt{01}$), and $3 + 3 + 2 = 8$ for $\{0\}$, $\{0.2\}$, and $\{0.4\}$ ($\mathtt{000}$, $\mathtt{001}$, and $\mathtt{01}$). We call these values the MCL since we cannot distinguish a pair of data points from different clusters if their truncated codes are shorter than the MCL.

Since COOL does not optimize an embedding scheme within the clustering process, the clustering result strongly depends on the embedding used. This means that we have to carefully choose an appropriate embedding for effective clustering. In this chapter, we consider the *Gray code* as an embedding scheme for COOL — resulting in an algorithm we call G-COOL. Gray code was originally developed for binary encoding of natural numbers and has become especially important in conversion between analog and digital information (Knuth, 2005). From the geometrical point of view, Gray code embedding is the partitioning of each interval into *overlapping* smaller intervals. This enables clusters with arbitrary shapes to be found, which cannot be done with binary embedding. There is theoretical support for clustering by G-COOL as shown in Lemma 4.9 and Theorem 4.11. Figure 4.1 illustrates examples of computing the MCL with binary and Gray code embedding.

Gray code

The motivation for using Gray code scheme comes from Computable Analysis (Weihrauch, 2000), a well-established mathematical framework for addressing analytical and computational aspects of real numbers through representation of real numbers as infinite sequences. Computability for real numbers depends on the embedding method used, and computation makes sense only if the method meets a key mathematical property: "admissibility" (see the book (Weihrauch, 2000) for its mathematical definition and properties). It is thus natural that the clustering results depends on the embedding method. Gray code has been shown to be admissible (Tsuiki, 2002) whereas binary embedding is not, and this property is a key for embedding that can detect arbitrarily shaped clusters.

This chapter is organized as follows: Section 4.1 introduces the MCL, and Section 4.2 gives a formal definition of clustering based on the MCL and explains the integration of COOL with the computation of the MCL. In Section 4.3, we introduce Gray code embedding and analyze G-COOL theoretically. Section 4.4 describes the experiments, presents the results, and discusses them. Section 4.5 summarized the key points with reviewing related work.

Related Work

Many types of *shape-based clustering*, or *spatial clustering*, methods have been proposed for finding arbitrarily shaped clusters, including partitional algorithms (Chaoji *et al.*, 2009, 2011), the mass-based clustering algorithm (Ting and Wells, 2010), density-based clustering algorithms (*e.g.*, DBSCAN (Ester *et al.*, 1996) and DENCLUE (Hinneburg and Keim, 1998)), agglomerative hierarchical clustering algorithms (*e.g.*, CURE (Guha *et al.*, 1998), CHAMELEON (Karypis *et al.*, 1999)), and grid-based algorithms (*e.g.*, STING (Wang *et al.*, 1997) and Wave Cluster (Sheikholeslami *et al.*, 1998)). However, most of them are not practical. Their clustering results are sensitive to the input parameters, which have to be tuned manually, so they work well only if all parameters are tuned appropriately by the user. As a result, these methods are not well suited for users who are not specialized in machine learning. Furthermore, most of them are not scalable: their time complexity is quadratic or cubic with respect to data size. Compared to these methods, G-COOL is robust to the input parameters and always finds the globally optimal clusters under the MCL criterion. Moreover, G-COOL is usually faster than most of these methods since the time complexity is linear with respect to data size.

Many cluster validity methods have been proposed for quantitative evaluation of clustering results (Handl *et al.*, 2005). These measures are usually divided into two categories: internal (*e.g.*, connectivity and Silhouette width) and external (*e.g.*, F-measure and Rand index). The internal measures are intrinsic to actual clustering while the external measures need information that may not be available in an actual situation. Our proposed measure, MCL, can be categorized as an internal measure. Its effectiveness has been shown experimentally (see Section 4.4).

Notation

In the following, $\mathbb{R}^d$ denotes the d-dimensional Euclidean space. A *data point* x is a vector in $\mathbb{R}^d$, and a *dataset* X is a finite set of data points. For a pair of sets X and Y, $X \setminus Y$ means the relative complement of Y in X.

Clustering is the partition of a dataset X into K subsets $C_1, \dots, C_K$, called *clusters*, where $C_i \neq \emptyset$, $C_i \cap C_j = \emptyset$ with $i \neq j$, and $\bigcup_{i \in \{1,\dots,K\}} C_i = X$. Here we say that a set $\mathcal{C} = \{C_1, \dots, C_K\}$ holding the above properties is a *partition* of X and denote the set of all possible partitions by $\mathcal{C}(X)$; *i.e.*, $\mathcal{C}(X) = \{\mathcal{C} \mid \mathcal{C} \text{ is a partition of } X\}$. For a cluster C, $\#C$ denotes the number of data points in C.

The set of finite and infinite sequences over an alphabet Σ is denoted by Σ^* and Σ^ω, respectively. The length $|w|$ of a finite or an infinite sequence w is the number of positions for symbols other than $\perp$ (the undefinedness character) in w. For example, if $w = 11\perp100\perp\perp\perp\ldots$, $|w| = 5$. For a set of sequences W, the size of W is defined by $|W| \coloneqq \sum_{w\in W} |w|$.

An *embedding* of $\mathbb{R}^d$ is an injection φ from $\mathbb{R}^d$ to Σ^ω (cf. Subsection 3.2.2). For a pair of infinite sequences p, q, we write $p \leq q$ if $p(i) = q(i)$ for all i with $p(i) \neq \perp$, where $p(i)$ denotes the ith position (including 0) of p. This means that q is more specific than p since $\perp$ denotes undefinedness. Moreover, if $w\perp^\omega \leq p$ for $w \in \Sigma^*$, we write $w \sqsubset p$ (w is a prefix of p). Remember that $\uparrow w = \{p \in \text{range}(\varphi) \mid w \sqsubset p\}$ for $w \in \Sigma^*$ and $\uparrow W = \{p \in \text{range}(\varphi) \mid w \sqsubset p \text{ for some } w \in W\}$ for $W \subseteq \Sigma^*$.

4.1 Minimum Code Length

The minimum code length, or MCL, is used to measure partitions under a fixed embedding φ. We define, for $p \in \text{range}(\varphi)$ and $P \subset \text{range}(\varphi)$,

$$\Phi(p\,|\,P) \coloneqq \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} p \in \uparrow w, \text{ and } P \cap \uparrow v = \emptyset \\ \text{for all } v \text{ with } |v| = |w| \text{ and } p \in \uparrow v \end{array} \right\}.$$

Every element in $\Phi(p \mid P)$ is a prefix of p that discriminates p from P. Trivially, $\Phi(p\,|\,P) = \emptyset$ if $p \in P$.

The MCL is introduced here in accordance with the above preparations.

Definition 4.1: MCL
Given an embedding φ, for a partition $\mathcal{C} = \{C_1, \ldots, C_K\}$ of a dataset X, we define

$$\text{MCL}(\mathcal{C}) \coloneqq \sum_{i\in\{1,\ldots,K\}} L_i(\mathcal{C}),$$

where

$$L_i(\mathcal{C}) \coloneqq \min \left\{ |W| \;\middle|\; \begin{array}{l} \varphi(C_i) \subseteq \uparrow W \text{ and} \\ W \subseteq \displaystyle\bigcup_{x\in C_i} \Phi\left(\varphi(x) \mid \varphi(X \setminus C_i)\right) \end{array} \right\}.$$

Intuitively, this gives the code length of the *maximally compressed representatives* of a given partition through discretization using fixed embedding φ since the following property holds: For a partition $\mathcal{C}$ of X, if we discretize each data point $x \in X$ into a finite sequence $c(x)$ with φ (*i.e.*, $c(x) \sqsubset \varphi(x)$) such that $\left|\bigcup_{x\in X} c(x)\right| < \text{MCL}(\mathcal{C})$, then there must exist a pair of data points $x, y \in X$ satisfying $\uparrow c(x) \cap \uparrow c(y) \neq \emptyset$ and $x \in C_i, y \in C_j$ with $i \neq j$. Therefore, we cannot discriminate x from y and thus cannot find the partition $\mathcal{C}$ from compressed codes $c(X)$.

Example 4.2
Suppose we use binary embedding φ_B. Assume a one-dimensional dataset $X = \{0.1, 0.2, 0.8, 0.9\}$ and partitions $\mathcal{C}_1 = \{\{0.1, 0.2\}, \{0, 8, 0.9\}\}$ and $\mathcal{C}_2 = \{\{0.1\},$

$\{0.2, 0.8\}, \{0.9\}\}$. Then, $\mathrm{MCL}(\mathcal{C}_1) = L_1(\mathcal{C}_1) + L_2(\mathcal{C}_1) = 1 + 1 = 2$ since $\varphi_{\mathrm{B}}([0, 0.5]) = {\uparrow}0$ and $\varphi_{\mathrm{B}}((0.5, 1]) = {\uparrow}1$, and $\mathrm{MCL}(\mathcal{C}_2) = L_1(\mathcal{C}_2) + L_2(\mathcal{C}_2) + L_3(\mathcal{C}_2) = 3 + (3 + 3) + 3 = 12$ because we have $\varphi_{\mathrm{B}}([0, 0.125]) = {\uparrow}000$, $\varphi_{\mathrm{B}}((0.125, 0.25]) = {\uparrow}001$, $\varphi_{\mathrm{B}}((0.75, 0.875]) = {\uparrow}110$, and $\varphi_{\mathrm{B}}((0.875, 1]) = {\uparrow}111$. Note that $\varphi_{\mathrm{B}}([0, 0.25]) = {\uparrow}00$, hence 0.1 and 0.2 cannot be discriminated using code 00, and that $\varphi_{\mathrm{B}}((0.75, 1]) = {\uparrow}11$, hence 0.8 and 0.9 cannot be discriminated using 11. □

The MCL is calculated for $O(nd)$ time complexity by using a radix sort, where n is the size of X (*i.e.*, $n = \#X$), and d is the dimension of X. This is why if the discretized dataset $\{p(0)p(1) \dots p(k-1) \mid p \in \varphi(X)\}$ at level k is sorted in advance, each data point simply needs to be compared with the subsequent one for each dimension, and the MCL is obtained by checking from $k = 1, 2, 3, \dots$.

4.2 Minimizing MCL and Clustering

We formulate clustering using the MCL as a criterion and describe clustering algorithm COOL, which finds the globally optimal partition that *minimizes* the MCL.

4.2.1 Problem Formulation

The clustering problem with the MCL is defined as follows.

Definition 4.3: Clustering under the MCL criterion
Clustering of a dataset X under the MCL criterion means finding the globally optimal partition that minimizes the MCL with more than K clusters; that is, finding $\mathcal{C}_{\mathrm{op}}$ such that

$$\mathcal{C}_{\mathrm{op}} \in \operatorname*{argmin}_{\mathcal{C} \in \mathfrak{C}(X)_{\geq K}} \mathrm{MCL}(\mathcal{C}),$$

where $\mathfrak{C}(X)_{\geq K} = \{\mathcal{C} \in \mathfrak{C}(X) \mid \#\mathcal{C} \geq K\}$.

In this framework, we assume that a lower bound on the number of clusters K is given to avoid overgeneralization since, if we search for the optimal partition $\mathcal{C}_{\mathrm{op}}$ in $\mathfrak{C}(X)$ (*i.e.*, all possible partitions) instead of $\mathfrak{C}(X)_{\geq K}$, we always have the nonsense result $\mathcal{C}_{\mathrm{op}} = \{X\}$.

4.2.2 COOL Algorithm

Our COOL algorithm efficiently solves the optimization problem (Definition 4.3) by integrating the computation of MCL within the clustering step. By contrast, naïve approach that would compare the MCLs of all possible partitions would result in an algorithm with exponential time complexity. The pseudo-code of COOL is shown in Algorithm 4.1.

COOL is a *level-wise* clustering algorithm that finds the optimal partition $\mathcal{C}_{\mathrm{op}}$ by enumerating level-k partitions ($k = 1, 2, 3, \dots$).

level-k partition

Definition 4.4: Level-k partition
For a dataset X and an embedding φ, the *level-k partition* $\mathcal{C}^k$ is defined as follows: Every pair of data points $x, y \in X$ are contained in the same cluster if and only if $v = w$ for some $v \sqsubset \varphi(x)$ and $w \sqsubset \varphi(y)$ with $|v| = |w| = k$.

Algorithm 4.1: COOL algorithm

```
Input: Dataset $X$, lower bound on the cluster size $K$, and noise parameter $N$
Output: Optimal partition $\mathcal{C}_{\mathrm{op}}$ and noise data
Function Cool($X, K, N$)
  Find partitions $\mathcal{C}^1_{\geq N}, \dots, \mathcal{C}^m_{\geq N}$ such that $\#\mathcal{C}^{m-1}_{\geq N} < K \leq \#\mathcal{C}^m_{\geq N}$
  $(\mathcal{C}_{\mathrm{op}}, \mathrm{MCL}) \leftarrow$ FindClusters($X, K, \{\mathcal{C}^1_{\geq N}, \dots, \mathcal{C}^m_{\geq N}\}$)
  return $(\mathcal{C}_{\mathrm{op}}, X \setminus \bigcup \mathcal{C}_{\mathrm{op}})$
Function FindClusters($X, K, \{\mathcal{C}^l, \dots, \mathcal{C}^m\}$)
  if $K = 1$ then
    return $(\mathcal{C}^l, |W|)$, where $\varphi(\bigcup \mathcal{C}^l) \subseteq \uparrow W$ and $|w| = l$ for all $w \in W$
  end if
  Find $k$ such that $\#\mathcal{C}^{k-1} < K \leq \#\mathcal{C}^k$
  $\mathcal{C}_{\mathrm{op}} \leftarrow \mathcal{C}^k$
  $\mathrm{MCL} \leftarrow \mathrm{MCL}(\mathcal{C}^k)$
  for each $C$ in $\mathcal{C}^l \cup \dots \cup \mathcal{C}^k$
    $L \leftarrow \min\{\, |W| \mid \varphi(C) \subseteq \uparrow W$ and $|w| = j$ for all $w \in W \,\}$
    $(\mathcal{C}, L') \leftarrow$ FindClusters($X \setminus C, K - 1, \{\mathcal{C}^j, \dots, \mathcal{C}^k\}$)
    if $L + L' < \mathrm{MCL}$ then
      $\mathcal{C}_{\mathrm{op}} \leftarrow C \cup \mathcal{C}$
      $\mathrm{MCL} \leftarrow L + L'$
    end if
  end for
  return $(\mathcal{C}_{\mathrm{op}}, \mathrm{MCL})$
```

This means that if $x, y \in X$ are in the same cluster, there exists a *chain* of data points $z_1, z_2, \dots, z_m$ ($m \geq 2$) such that, for all $i \in \{1, 2, \dots, m-1\}$, $z_1 = x$, $z_m = y$, and $w_i = w_{i+1}$ for some $w_i \sqsubset \varphi(z_i)$ and $w_{i+1} \sqsubset \varphi(z_{i+1})$ with $|w_i| = |w_{i+1}| = k$. Obviously, the level-k partition is determined uniquely. The time complexity of finding the level-k partition is $O(nd)$, where n and d are the size and dimension of the dataset, respectively, since, if the discretized dataset $\{p_0 p_1 \dots p_{k-1} \mid p \in \varphi(X)\}$ at level k is initially sorted using a radix sort, clusters are constructed by comparing each data point to the next data point for each dimension.

The most important feature of the level-k partition is that the optimal partition $\mathcal{C}_{\mathrm{op}}$ in Definition 4.3 is obtained by searching for only clusters contained in the level-k partition.

Lemma 4.5
For every cluster $C \in \mathcal{C}_{\mathrm{op}}$, C is contained in some level-k partition, that is, $C \in \mathcal{C}^k$ for some $k \in \mathbb{N}$.

Proof. Let $\mathcal{C}$ be a partition such that, for every $C \in \mathcal{C}$, $C \in \mathcal{C}^k$ for some k, and a pair of clusters $C, C' \in \mathcal{C}$ is fixed. Then, from the definition of the level-k partition, the following condition holds: For all pairs of clusters D, D' such that $D \cup D' = C \cup C'$ and $D \cap D' = \emptyset$, we have $\mathrm{MCL}(\mathcal{C}) \leq \mathrm{MCL}(\mathcal{C}')$, where $\mathcal{C}' = (\mathcal{C} \setminus \{C, C'\}) \cup \{D, D'\}$. Therefore, for the optimal partition $\mathcal{C}_{\mathrm{op}}$, $C \in \mathcal{C}^k$ with $k \in \mathbb{N}$ for all $C \in \mathcal{C}_{\mathrm{op}}$. □

The level-k partition has a *hierarchical* structure: For each cluster $C \in \mathcal{C}^k$, there must exist a set of clusters $\mathcal{D} \subseteq \mathcal{C}^{k+1}$ such that $\bigcup \mathcal{D} = C$. Thus, COOL works through divisive hierarchical clustering. The MCL of the level-k partition used in line 6 of the function FindClusters in Algorithm 4.1 can thus be easily calculated: Let $\mathcal{C}^k$ be a set of clusters $\{C_1, \dots, C_K\}$. For each C_i and for the minimum level l such that $C_i \in \mathcal{C}^l$,

$$L_i(\mathcal{C}^k) = \min\{|W| \mid \varphi(C_i) \subseteq {\uparrow}W \text{ and } |w| = l \text{ for all } w \in W\}$$

holds. This means that we can obtain the MCL of the level-k partition by checking only sequences with length l.

Next we show that COOL can solve the optimization problem in Definition 4.3.

> **Proposition 4.6**
> *The* COOL *algorithm* (*Algorithm 4.1*) *always outputs the globally optimal partition* $\mathcal{C}_{\mathrm{op}}$.

Proof. Let $\#\mathcal{C}_{\mathrm{op}} = K$. Then there must exist $k \in \mathbb{N}$ such that $K \leq \#\mathcal{C}^k$ and $\#\mathcal{C}^{k'} < K$ for all $k' < k$ since the number of clusters in the level-k partition $\#\mathcal{C}^k$ increases monotonically with respect to increase of k. Fix a cluster $C \in \mathcal{C}_{\mathrm{op}}$, and let $\mathcal{C}'_{\mathrm{op}}$ be the optimal partition for the dataset $X \setminus C$. Then we can easily check that $\mathcal{C}'_{\mathrm{op}} \cup \{C\}$ coincides with $\mathcal{C}_{\mathrm{op}}$. Moreover, from Lemma 4.5 and the definition of the level-k partition, for all $C \in \mathcal{C}_{\mathrm{op}}$, $C \in \mathcal{C}^l$ for some $l \in \{1, \dots, k\}$. Thus, COOL finds the optimal partition $\mathcal{C}_{\mathrm{op}}$ by recursive computing in Algorithm 4.1 (lines 4 - 7) with fixing each cluster in $\mathcal{C}^1 \cup \dots \cup \mathcal{C}^k$. □

COOL can find the globally optimal partition $\mathcal{C}_{\mathrm{op}}$ efficiently, and its time complexity is $O(nd)$ and $O(nd + K!)$ in the best and worst cases, respectively, since finding m partitions in the first line of the function Cool takes $O(nd)$, and the function FindClusters takes $O(K!)$ in the worst case. Usually, $K \ll n$ holds, so complexity becomes $O(nd)$.

noise parameter

Furthermore, noise is directly removed by COOL using a lower bound on the size of each cluster N, which we call the *noise parameter*. For a partition $\mathcal{C}$, we denote the set $\{C \in \mathcal{C} \mid \#C \geq N\}$ by $\mathcal{C}_{\geq N}$. For example, let a dataset $X = \{0.1, 0.4, 0.5, 0.6, 0.9\}$ and $\mathcal{C} = \{\{0.1\}, \{0.4, 0.5, 0.6\}, \{0.9\}\}$. Then, $\mathcal{C}_{\geq 2} = \{\{0.4, 0.5, 0.6\}\}$, and two data points, 0.1 and 0.9, are detected as noise.

4.3 G-COOL: COOL with Gray Code

We use Gray code embedding for COOL, and show its powerful clustering ability by theoretical analysis. We call COOL with Gray code embedding G-COOL.

4.3.1 Gray Code Embedding

Gray code embedding is illustrated in Figure 4.2. Its rich mathematical properties are described elsewhere (Tsuiki, 2002). Gray code was originally simply binary encoding of natural numbers, as mentioned in introduction. For example, natural numbers $1, 2, \dots, 8$ are represented in Gray code as 000, 001, 011, 010, 110, 111, 101, 100, whereas, in binary code, they are represented as 000, 001, 010, 011, 100,

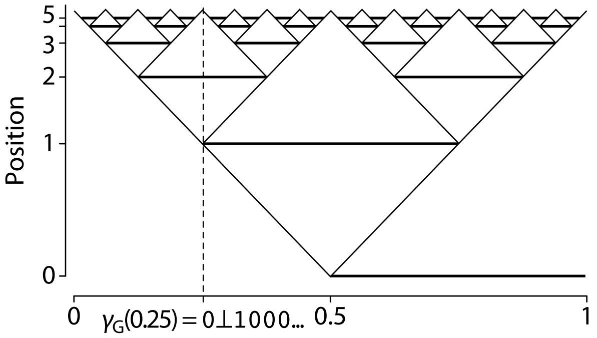


Figure 4.2 | Gray code embedding φ_G. Position i is 1 if it is on the line, $\bot$ if on the end point, and 0 otherwise. Diagonal lines are auxiliary lines. For example, if $p = \varphi_G(0.25)$, $p = 0\bot1000\ldots$ because position 0 is not on the line, 1 is on the end point, 2 is on the line, and every $i \geq 3$ is not on the line.

101, 110, 111. The importance of Gray code is that only one bit differs between one code and its successor, that is, the Hamming distance between them is always one. Here $\mathcal{I}$ denotes the unit interval $[0,1] \times \ldots \times [0,1] \subset \mathbb{R}^d$, and $\Sigma^\omega_{\bot,d}$ denotes the set of infinite sequences for which, in each sequence, at most d positions are $\bot$. For example, if $\Sigma = \{0,1\}$ and $d = 2$, then $0\bot100\ldots \in \Sigma^\omega_{\bot,d}$, $\bot\bot110\ldots \in \Sigma^\omega_{\bot,d}$, and $0\bot1\bot\bot0\ldots \notin \Sigma^\omega_{\bot,d}$. In the following, we consider only real vectors in $\mathcal{I}$.

Definition 4.7: Gray code embedding

Gray code embedding

(*One-dimensional*) *Gray code embedding* is an injective function, $\varphi_G : \mathcal{I} \to \Sigma^\omega_{\bot,d}$ $(d = 1)$, that maps $x \in \mathcal{I}$ to an infinite sequence $p_0p_1p_2\ldots$: For each i, $p_i := 1$ if

$$2^{-i}m - 2^{-(i+1)} < x < 2^{-i}m + 2^{-(i+1)}$$

holds for an odd number m, $p_i := 0$ if the same holds for an even number m, and $p_i := \bot$ if $x = 2^{-i}m - 2^{-(i+1)}$ for some integer m.

Moreover, by using the *tupling function* (see Equation (2.4)), we can define *d-dimensional Gray code embedding* $\varphi^d_G : \mathcal{I} \to \Sigma^\omega_{\bot,d}$ as

$$\varphi^d_G(x_1,\ldots,x_d) := \langle \varphi_G(x_1),\ldots,\varphi_G(x_d)\rangle.$$

We abbreviate d of φ^d_G if it is understood from the context.

Example 4.8

For one-dimensional data points $x = 0.2$, $y = 0.5$, and $z = 0.7$, we have $\varphi_G(x) = 0010\ldots$, $\varphi_G(y) = \bot100\ldots$, and $\varphi_G(z) = 1110\ldots$ with the Gray code embedding, while $\varphi_B(x) = 0001\ldots$, $\varphi_B(y) = 0111\ldots$, and $\varphi_B(z) = 1011\ldots$ with the binary embedding. For a two-dimensional data point (x,y), we have $\varphi_G(x,y) = 0\bot011000\ldots$, and for a three-dimensional data point (x,y,z), $\varphi_G(x,y,z) = 0\bot1$ $011101000\ldots$ with Gray code embedding. □

4.3.2 Theoretical Analysis of G-COOL

Here we show that G-COOL achieves internal cohesion and external isolation without any distance calculation or data distribution. In the following, we measure the distance between $x, y \in \mathbb{R}^d$ by the L_∞ metric, where the distance is defined by

$$d_\infty(x,y) := \max_{i\in\{1,\ldots,d\}} |x_i - y_i|.$$

id	Value	Level 1		Level 2	
		Binary	Gray	Binary	Gray
a	0.14	0	0	00	00
b	0.48	0	0, ⊥1	01	01, ⊥10
c	0.51	1	1, ⊥1	10	11, ⊥10
d	0.73	1	1, ⊥1	10	11, 1⊥1
e	0.77	1	1	11	10, 1⊥1

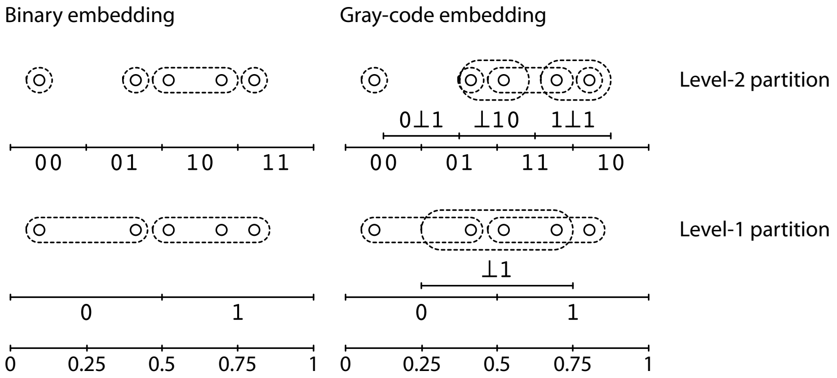


Figure 4.3 | Examples of level-1 and 2 partitions with binary and Gray code embedding.

Lemma 4.9
For the level-k partition $\mathcal{C}^k$ of a dataset X with Gray code embedding φ_{G}, two data points $x, y \in X$ are in the same cluster if $d_\infty(x, y) < 2^{-(k+1)}$ and are not in the same cluster only if $d_\infty(x, y) \geq 2^{-(k+1)}$.

Proof. From the definition of Gray code embedding, if $d_\infty(x, y) < 2^{-(k+1)}$ for $x, y \in X$, there must exist a finite sequence w with $|w| = k$ such that $w \sqsubset \varphi_{\mathrm{G}}(x)$ and $w \sqsubset \varphi_{\mathrm{G}}(y)$. Thus, x and y are in the same cluster in the level-k partition $\mathcal{C}^k$. This means that, if x and y are in the different clusters in $\mathcal{C}^k$, $d_\infty(x, y) \geq 2^{-(k+1)}$. □

Informally, the *redundancy* of Gray code embedding enables the powerful property described in the above lemma, that is, for an infinite sequence $p = \varphi_{\mathrm{G}}(x)$, there may be two prefixes, $v_1 \sqsubset p$ and $v_2 \sqsubset p$ with $|v| = |w|$.

Example 4.10
Let us consider the situation illustrated in Figure 4.3, where we have five data points: $x_{\mathrm{a}} = 0.14$, $x_{\mathrm{b}} = 0.48$, $x_{\mathrm{c}} = 0.51$, $x_{\mathrm{d}} = 0.73$, and $x_{\mathrm{e}} = 0.77$. In binary embedding, the unit interval $\mathcal{I} = [0, 1]$ is divided into two intervals $[0, 0.5]$ and $[0.5, 1]$ at level-1, while it is divided into three intervals $[0, 0.5]$, $[0.25, 0.75]$, and $[0.5, 1]$ in Gray code embedding. Thus, there are three overlapping clusters $\{x_{\mathrm{a}}, x_{\mathrm{b}}\}$ (encoded as 0), $\{x_{\mathrm{b}}, x_{\mathrm{c}}, x_{\mathrm{d}}\}$ (encoded as ⊥1), and $\{x_{\mathrm{c}}, x_{\mathrm{d}}, x_{\mathrm{e}}\}$ (encoded as 1). Actually, there is only one cluster $\{x_{\mathrm{a}}, x_{\mathrm{b}}, x_{\mathrm{c}}, x_{\mathrm{d}}, x_{\mathrm{e}}\}$ since they are merged. At level-2, we have four clusters with binary embedding although some data points such as x_{b} and x_{c} are close. On the other hand, we have two *natural* clusters $\{x_{\mathrm{a}}\}$ and $\{x_{\mathrm{b}}, x_{\mathrm{c}}, x_{\mathrm{d}}, x_{\mathrm{e}}\}$ with Gray code embedding. □

Intuitively, this lemma theoretically supports the claim that G-COOL finds *natural* clusters. For a data point $x \in X$, we say that a data point $y \in X$ is the *nearest*

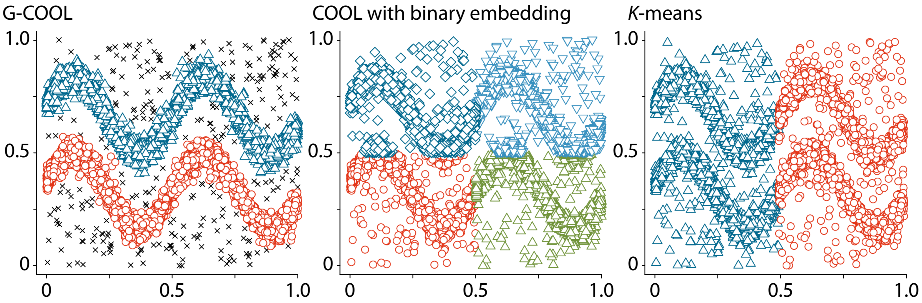


Figure 4.4 | Clustering results for G-COOL and COOL with binary embedding ($K=2, N=50$) and K-means ($K=2$). Dataset size is 10,500 (where 500 points are noise). G-COOL detects two natural clusters. The other two methods cannot find such clusters.

neighbor of x if $y \in \operatorname{argmin}_{x' \in X} d_\infty(x, x')$.

> **Theorem 4.11**
> *The optimal partition $\mathcal{C}_{\mathrm{op}}$ of a dataset X generated by* G-COOL *has the following property: For every data point $x \in C$ with $C \in \mathcal{C}_{\mathrm{op}}$ and $\#C \geq 2$, its nearest neighbor $y \in C$.*

Proof. From Lemma 4.5, every cluster $C \in \mathcal{C}_{\mathrm{op}}$ is contained in $\mathcal{C}^k$ for some k. Thus, from Lemma 4.9, any $x \in C$ with $C \in \mathcal{C}_{\mathrm{op}}$, $\#C = 1$ or its nearest neighbor $y \in C$. □

This property of Gray code (Lemma 4.9) enables clusters with the condition in Theorem 4.11 to be quickly found, whereas the naïve solution results in more than $O(n^2)$. Figure 4.4 illustrates the results of G-COOL for a two-dimensional dataset for which K-means could not find natural clusters. We can see that COOL with binary embedding also failed to find such clusters.

4.4 Experiments

We empirically evaluate the effectiveness of G-COOL and the proposed measure, MCL. We use low-dimensional synthetic and real datasets, which are common in spatial clustering setting.

4.4.1 Methods

Environment

G-COOL was implemented in R version 2.12.2 (R Development Core Team, 2011), and all experiments were performed in R. We used Mac OS X version 10.6.5 with two 2.26-GHz Quad-Core Intel Xeon CPUs and 12 GB of memory.

Datasets

The synthetic datasets were used to evaluate robustness against the number of clusters, the size of the datasets, and noise existence. They were randomly generated using the R `clusterGeneration` package (Qiu and Joe, 2006), and the parameters were set as follows: $sepVal = 0.34$, $numNonNoisy = 2$, $numNoisy = 1$, $numOutlier = 500$, and $rangeN = c(1000, 2000)$. We generated 20 datasets to obtain the mean and s.e.m. (standard error of the mean). The size of each cluster

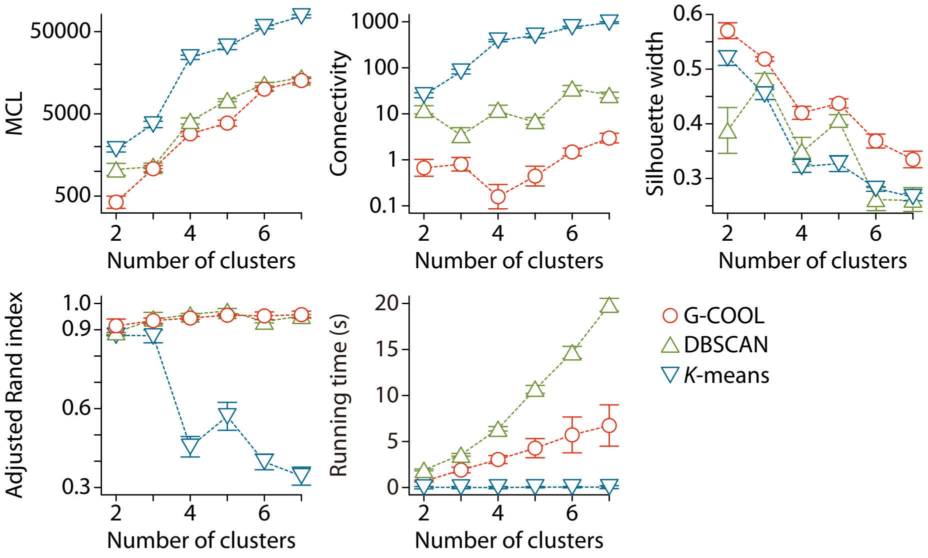


Figure 4.5 | Experimental results for synthetic datasets. MCL and connectivity should be minimized, and Silhouette width and adjusted Rand index should be maximized. Data show mean ± s.e.m.

was around 1,500, so the size of the datasets varied from ~3,000 to ~10,500. Each dataset was three-dimensional, where one dimension was composed of noise and about 500 data point were added to each dimension as noise.

Five real datasets were collected from the Earth-as-Art website[1], which contains geospatial satellite images (see Table 4.1 and Figure 4.7). Similar datasets were used in experiments with the state-of-the-art study (Chaoji *et al.*, 2011). Each image was pre-processed using ImageJ software (Rasband, 1997–2011); they were reduced to 200 × 200 pixels and translated into binary images.

With G-COOL, each dataset was translated using min-max normalization (Han and Kamber, 2006) so that the dataset was in the unit interval $\mathcal{I}$, where each value x of ith dimension X_i of a dataset X was mapped to $x' = (x - \min X_i)/(\max X_i - \min X_i)$ and the runtime for the translation was included in the G-COOL running time.

Control Methods

As control methods, we used K-means and DBSCAN because K-means is the standard clustering algorithm and DBSCAN is a typical method for finding arbitrarily shaped clusters, and their source codes are publicly available. DBSCAN was executed using the R `fpc` package. We tuned the parameters of DBSCAN to obtain the best results.

Evaluation

With the synthetic datasets, performance was evaluated using internal and external measures. As internal measures, we used the MCL (with Gray code), the connectivity (takes values in $[0, \infty]$, to be minimized) (Handl *et al.*, 2005), and the Silhouette width (takes values in $[-1, 1]$, to be maximized) (Rousseeuw, 1987). As an external measure, we used the adjusted Rand index (takes values in $[-1, 1]$, to

[1] `http://eros.usgs.gov/imagegallery/`

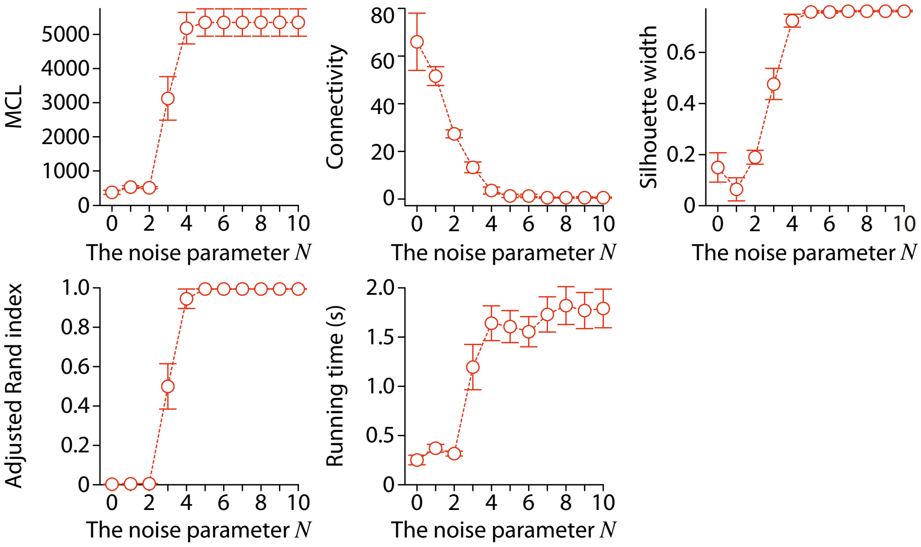


Figure 4.6 | Clustering speed and quality for G-COOL and synthetic datasets. MCL and connectivity should be minimized, and Silhouette width and adjusted Rand index should be maximized. G-COOL shows robust performance if the noise parameter N is large enough. Data show mean ± s.e.m.

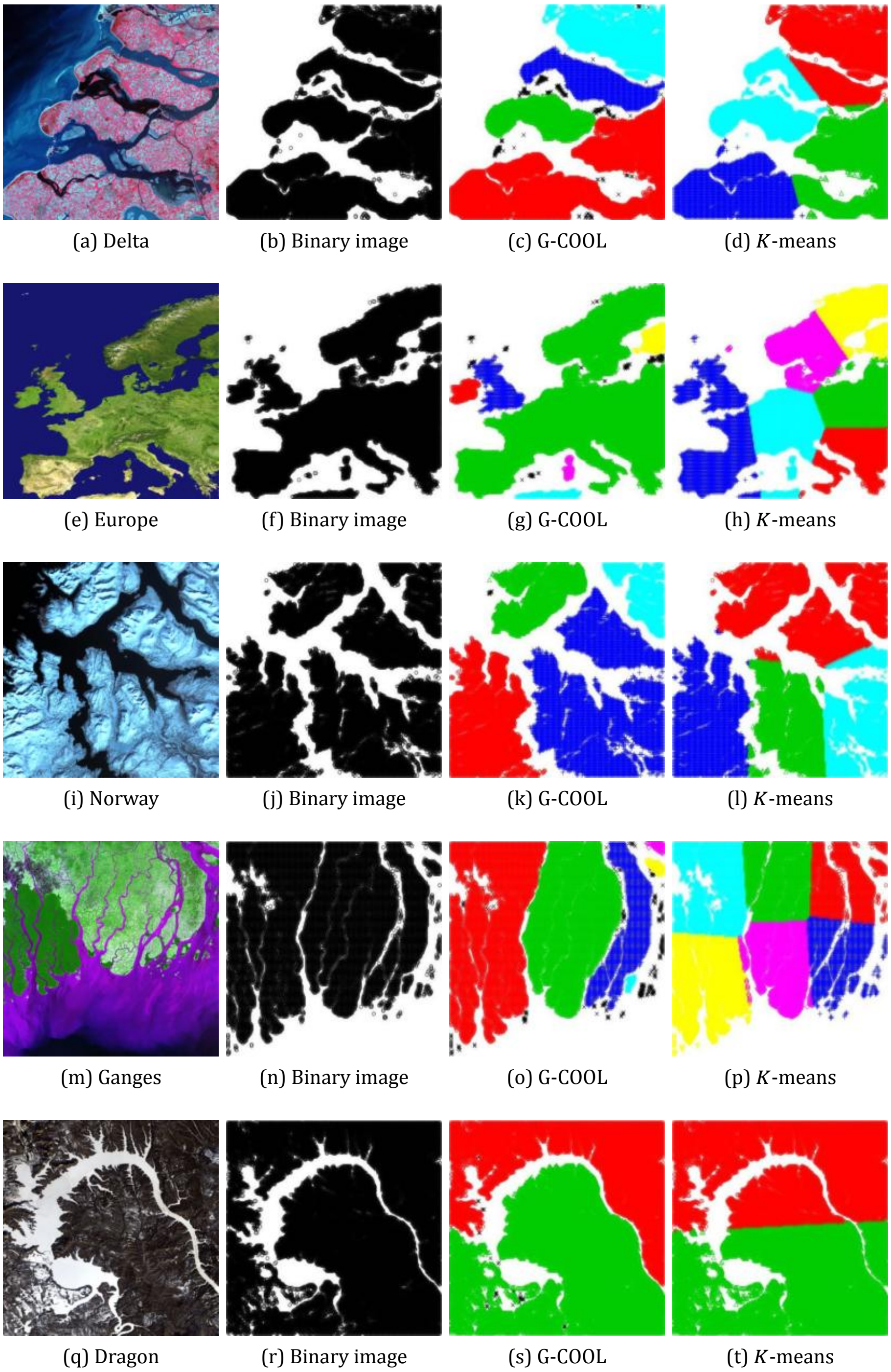
be maximized) (Hubert and Arabie, 1985), which takes into account the ground truth and is popular in the clustering literature. The measures were calculated using the R `clValid` (Brock *et al.*, 2008), `cluster` (Maechler *et al.*, 2005), and `clues` (Chang *et al.*, 2010) packages, respectively. For the real datasets, we used the MCL and simply show scatter plots of the results since we had no information on the ground truth.

4.4.2 Results and Discussion

The results obtained with the synthetic datasets (Figure 4.5) show that the quality of clusters obtained with G-COOL is significantly higher for three of the four quality measures (determined by paired t-test) and is competitive for the other one (adjusted Rand index). Moreover, it is faster than DBSCAN. These results show that the MCL works reasonably well as a measure of cluster quality compared to existing ones.

Note that we need to input only the lower bounds for the number and size of the clusters in G-COOL, whereas we have to tune the parameters carefully in DBSCAN and other shape-based (spatial) clustering algorithms. Therefore, G-COOL is more efficient and effective than existing clustering algorithms. Moreover, as shown in Figure 4.6, cluster quality is stable with respect to the noise parameter N (*i.e.*, lower bound on cluster size) even if the dataset contains noise, when N is large enough. If N is too small, then each noise is detected as a cluster. Thus, when clustering using G-COOL, all we have to do is set the parameter large enough, meaning that G-COOL is equally useful as K-means.

For all the real datasets, G-COOL finds natural clusters (N was set as 50 for all the datasets), as shown in Figure 4.7, whereas K-means results in inferior clustering quality (we did not perform DBSCAN since it takes too much time and needs manual tuning of the input parameters). Moreover, MCL of clustering results for G-COOL is much smaller than those for K-means (Table 4.1). These results show that G-COOL is robust and that it can find arbitrarily shaped clusters without careful tuning of the input parameters.



(a) Delta (b) Binary image (c) G-COOL (d) K-means

(e) Europe (f) Binary image (g) G-COOL (h) K-means

(i) Norway (j) Binary image (k) G-COOL (l) K-means

(m) Ganges (n) Binary image (o) G-COOL (p) K-means

(q) Dragon (r) Binary image (s) G-COOL (t) K-means

Figure 4.7 | Results for real datasets obtained from satellite images by G-COOL and *K*-means. G-COOL finds all natural clusters whereas *K*-means does not.

Name	n	K	Running time (s)		MCL	
			G-COOL	K-means	G-COOL	K-means
Delta	20748	4	1.158	0.012	4010	4922
Dragon	29826	2	0.595	0.026	3906	7166
Europe	17380	6	2.404	0.041	2320	12210
Norway	22771	5	0.746	0.026	1820	6114
Ganges	18019	6	0.595	0.026	2320	12526

Table 4.1 | Running time (in seconds) and MCL for real datasets. In table, n and K denote the number of data points and clusters, respectively. Clustering results are shown in Figure 4.7.

4.5 Summary

We have proposed an internal measure, the minimum code length (MCL), to evaluate the results of clustering and presented a clustering algorithm COOL that always finds the globally optimal partition; *i.e.*, clusters that have the minimum MCL. Intuitively, COOL produces the maximally compressed clusters using a fixed encoding scheme and does not take optimization of encoding into account. Moreover, Gray code is used for the encoding, resulting in an algorithm called G-COOL. Theoretically and empirically, G-COOL has been shown to be noise tolerant and to have the ability to find arbitrarily shaped clusters efficiently. The result is an effective solution to two essential problems, how to measure the goodness of clustering results and how to find good clusters.

G-COOL's results are robust to changes in the input parameters, and does not assume a data distribution and does not need a distance calculation. Thus, it can be effectively applied to other machine learning tasks, such as anomaly detection. Theoretical analysis of relationship between admissibility of encoding schemes in computing real numbers and the ability of clustering to detect arbitrarily shaped clusters is necessary future work.

5

CLUSTERING USING BINARY DISCRETIZATION

The K-means algorithm (MacQueen, 1967) is widely used due to its simplicity and efficiency while it is one of the earliest clustering algorithms. The main drawback is its limited clustering ability; that is, non-convex clusters cannot be found and noise is not filtered since the separation of data is based on a Voronoi diagram. To find clusters with arbitrary shapes, many *spatial clustering*, or *shape-based clustering*, algorithms have been proposed (Han *et al.*, 2001) (see Section 5.4 for related work). One of major applications of such algorithms is the identification of similar regions from geographical databases. However, most of them are not scalable. Their time complexity is quadratic or even cubic with respect to the number of data points. Moreover, the clustering results are sensitive to the input parameters, which have to be tuned manually, and they detect reasonable clusters only if all parameters are tuned appropriately.

spatial clustering

shape-based clustering

Ideally, a clustering algorithm should be

- *fast*, applicable to massive data sets with a complexity that scales linearly with its size;
- *robust* in the sense that its does not require too much parameter tuning in order to produce meaningful results;
- *adaptive*, be able to detect non-convex clusters.

The contribution of this chapter is to present a spatial clustering algorithm, called BOOL (Binary cOding Oriented cLustering), that meets all the above three requirements. The efficiency and effectiveness of our BOOL algorithm are shown experimentally (see Section 5.3). The key idea is to translate input data onto binary representations using *binary discretization* and use such binary codes to sort and merge data points for high-speed large scale clustering. A hierarchy of clusters is naturally produced by increasing the precision of the discretization. Each hierarchy level is constructed in only two steps: discretization of each numerical data point and agglomeration of adjacent smaller clusters.

binary discretization

The clustering procedure of BOOL is illustrated in Figure 5.1. We assume that every data point is in a two-dimensional space $[0,1] \times [0,1]$. First, data points

	Original data		Level 1		Level 2	
	1	2	1	2	1	2
x_1	0.15	0.32	0	0	0	1
x_2	0.66	0.71	1	1	2	2
x_3	0.72	0.86	1	1	2	3
x_4	0.48	0.89	0	1	1	3
x_5	0.74	0.48	1	0	2	1

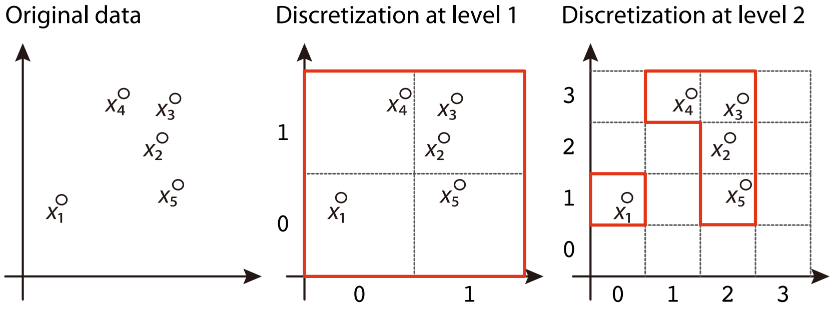


Figure 5.1 | Example of clustering using BOOL.

are discretized at level 1; *i.e.*, each value $x(i)$ ($i = 1$ or 2) is discretized to 0 if $0 \leq x(i) \leq 0.5$ and 1 if $0.5 < x(i) \leq 1$. For instance, two data points $(0.15, 0.32)$ and $(0.48, 0.89)$ are discretized to $(0, 0)$ and $(0, 1)$, respectively. Next, clusters using the following definition are constructed; two data points are considered to be in the same cluster if and only if the number of positions where two values differ is at most one and the difference in the values is less than l (the precise condition is given in Definition 5.2). Intuitively, if $l = 1$, the two values lie in the same square or the adjacent squares from the geometrical point of view. Thus, at this level, all data are in the same cluster for all $l \geq 1$. Next, all data points are discretized at level 2; each value $x(i)$ is discretized to 0 if $0 \leq x(i) \leq 0.25$, 1 if $0.25 < x(i) \leq 0.5$, 2 if $0.5 < x(i) \leq 0.75$, and 3 if $0.75 < x(i) \leq 1$. If $l = 1$, two clusters are found: a data point x_1 is in one cluster, and x_2, x_3, x_4, and x_5 are in the other cluster.

Due to the naïve approach of this clustering process, the time complexity is quadratic with respect to the number of data points (see Section 5.1) since each point has to be compared to all the other ones. However, if the database is sorted in advance, each data point simply needs to be compared with the subsequent one, so clustering can be performed with linear complexity (see Section 5.2).

This chapter is organized as follows: Section 5.1 introduces the naïve version of BOOL to clearly explain the clustering process and analyze the relationship between BOOL and DBSCAN (Ester *et al.*, 1996), and Section 5.2 discusses the speeding up by using sorting. The experimental results are presented and discussed in Section 5.3. Related works are reviewed in Section 5.4, and the key points are summarized in Section 5.5.

5.1 Clustering Strategy

In this section, we formulate the clustering problem and describe the naïve version of our BOOL algorithm to enable the reader to get a clear understanding of the

Algorithm 5.1: Naïve BOOL

```
Input: Database $\tau = (H, X)$, lower bound on number of clusters $K$,
       noise parameter $N$, and distance parameter $l$
Output: Partition $\mathcal{C}$
function NaïveBool($\tau, K, N$)
  $k \leftarrow 1$    // $k$ is level of discretization
  repeat
    $\mathcal{C} \leftarrow \{\, \{x\} \mid x \in \mathrm{set}(X) \,\}$
    for each object $x \in \mathrm{set}(X)$
      for each object $y \in \mathrm{set}(X)$
        if $d_0(\Delta^k(x), \Delta^k(y)) \leq 1$ and $d_\infty(\Delta^k(x), \Delta^k(y)) \leq l$ then
          delete clusters $C \ni x$ and $D \ni y$ from $\mathcal{C}$, and add $C \cup D$
        end if
      end for
    end for
    $\mathcal{C} \leftarrow \{\, C \in \mathcal{C} \mid \#C \geq N \,\}$    // noise filtering
    $k \leftarrow k + 1$
  until $\#\mathcal{C} \geq K$
  output $\mathcal{C}$
```

clustering process. Note that results obtained with the naïve version and with the speeded-up version are exactly the same — time complexity is the only difference between them. The pseudo-code of naïve BOOL is shown in Algorithm 5.1.

5.1.1 Formulation of Databases and Clustering

We treat the target data set as a *table*, or a *relation* (Date, 2003; Garcia-Molina *et al.*, 2008; Simovici and Djeraba, 2008); τ which is a pair (H, X) of a *header* H and a *body* X. A header H is a finite set of *attributes*, where each $h \in H$ is referred to as the *domain* of h, denoted by $\mathrm{Dom}(h)$. A body X is a sequence of *tuples*, or *data points*, $x_1, x_2, \dots, x_n$, where each tuple x_i is defined as a total function from H to $\mathrm{Dom}(H) = \{\mathrm{Dom}(h) \mid h \in H\}$ such that $x_i(h) \in \mathrm{Dom}(h)$ for all $h \in H$. We denote the number of tuples, the table size, n by $|\tau|$. When we treat the body X as a set in the set theoretic manner, we denote it by $\mathrm{set}(X)$, that is, $\mathrm{set}(X) = \{x_1, x_2, \dots, x_n\}$. This means that we do not take the order and multiplicity into account in $\mathrm{set}(X)$.

Throughout this chapter, the domain $\mathrm{Dom}(h)$ of each attribute $h \in H$ is considered to be the closed interval $[0, 1] \subset \mathbb{R}$. For our experiments using synthetic and real data sets (Section 5.3), we first perform pre-processing (min-max normalization) (Han and Kamber, 2006) so that each value for each attribute is in the domain $[0, 1]$. We assume that the header H is always the set of natural numbers $\{1, 2, \dots, d\}$, so each tuple corresponds to a real vector in the d-dimensional Euclidean space $\mathbb{R}^d$. We call a tuple an *object* and a relation a *database*.

Let J be a subset of the header H. For each object x, the *projection* of x on J, denoted by $x|_J$, is exactly the same as the restriction of x to J, which is the function from J to $\mathrm{Dom}(H)$ such that $x|_J(h) = x(h)$ for all $h \in J$. For a table $\tau = (H, X)$,

τ			
H	1	2	3
x_1	0.66	0.71	0.27
x_2	0.72	0.86	0.46
x_3	0.14	0.53	0.04

$\Delta^1(\tau)$			
H	1	2	3
$\Delta^1(x_1)$	1	1	0
$\Delta^1(x_2)$	1	1	0
$\Delta^1(x_3)$	0	1	0

$\Delta^2(\tau)$			
H	1	2	3
$\Delta^2(x_1)$	2	2	1
$\Delta^2(x_2)$	2	3	1
$\Delta^2(x_3)$	0	2	0

Table 5.1 | Database τ, and discretized databases $\Delta^1(\tau)$ and $\Delta^2(\tau)$.

the projection of τ is the database $\tau|_J = (J, X|_J)$, where $X|_J$ is defined by $X|_J := x_1|_J, x_2|_J, \dots, x_n|_J$.

Clustering is the partitioning of the set of objects set(X) of a database τ into K disjoint subsets C_1, C_2, ..., C_K, called *clusters*, such that $C_i \neq \emptyset$, $C_i \cap C_j = \emptyset$ with $i \neq j$, and $\bigcup_{i \in \{1,\dots,K\}} C_i = \text{set}(X)$. Here, a database is treated as a sequence of objects (tuples) to take ordering and multiplicity of objects into account just like the relational database theory, while a cluster is a finite set of objects in the set theoretic manner and ordering and multiplicity is not considered. We say that a set of clusters $\mathcal{C} = \{C_1, \dots, C_K\}$ with the above properties is a *partition* of set(X). The number of objects in C is denoted by $\#C$.

5.1.2 Naïve BOOL

BOOL performs divisive hierarchical clustering, so the number of clusters increases monotonically with the hierarchy level, and the union of some clusters becomes a cluster at the previous lower level. At each level, BOOL determines the number of clusters automatically. This ability is the major difference between BOOL and other divisive hierarchical methods.

First, we introduce discretization of objects in a database based on binary encoding scheme. Recall that the domain of each attribute is assumed to be in the interval $[0, 1]$.

Definition 5.1: Discretization
For an object x in a database τ, *discretization at level k* is an operator Δ^k for x, where each value $x(i)$ is mapped to a natural number m such that $x(i) = 0$ implies $m = 0$, and $x(i) \neq 0$ implies

discretization

$$m = \begin{cases} 0 & \text{if } 0 < x(i) \leq 2^{-k}, \\ 1 & \text{if } 2^{-k+1} < x(i) \leq 2^{-k+2}, \\ \dots & \\ 2^k - 1 & \text{if } 2^{-k+(k-1)} < x(i) \leq 1. \end{cases}$$

We use the same operator Δ^k for discretization of a database τ; *i.e.*, each object x of τ is discretized to $\Delta^k(x)$ in the database $\Delta^k(\tau) = (H, \Delta^k(X))$. Table 5.1 shows an example of discretization at levels 1 and 2. In the following, we fix the level of discretization as k and explain the construction of clusters at level k.

Next, we make clusters through discretization Δ^k by measuring the distances between objects using the L_0 and L_∞ metrics, where the distance between a pair of objects x and y is defined by

$$d_0(x, y) = \sum_{i=1}^{d} \delta(x(i), y(i)), \text{ where } \delta = \begin{cases} 0 & \text{if } x(i) = y(i), \\ 1 & \text{if } x(i) \neq y(i), \end{cases}$$

$$d_\infty(x, y) = \max_{i \in \{1,\dots,d\}} |x(i) - y(i)|$$

l-neighborhood at level k

in the L_0 and L_∞ metrics, respectively. Here, for an object x in a database $\tau = (H, X)$ and a natural number $l \in \mathbb{N}$, we define x's *l-neighborhood at level k* by

$$N_l^k(x) := \left\{ y \in \mathrm{set}(X) \;\middle|\; d_0(\Delta^k(x), \Delta^k(y)) \leq 1 \text{ and } d_\infty(\Delta^k(x), \Delta^k(y)) \leq l \right\}. \tag{5.1}$$

distance parameter

We call l the *distance parameter*. This condition means that the number of attributes for which two values differ is at most one and that the difference in the values is less than l (see Figure 5.2). Note that, if $l = 1$,

$$N_1^k(x) = \{y \in \mathrm{set}(X) \mid d_1(\Delta^k(x), \Delta^k(y)) \leq 1\}$$

holds, where d_1 is the Manhattan distance (L_1 metric).

directly reachable at level k
reachable at level k

Definition 5.2: Reachability
Given a distance parameter $l \in \mathbb{N}$ and a database $\tau = (H, X)$. An object $x \in \mathrm{set}(X)$ is *directly reachable at level k* from an object $y \in \mathrm{set}(X)$ if $x \in N_l^k(y)$. Moreover, x is *reachable at level k* from an object $z \in \mathrm{set}(X)$ if there exists a chain of objects $z_1, z_2, \dots, z_p$ ($p \geq 2$) in τ such that $z_1 = z$, $z_p = x$, and z_{i+1} is directly reachable at level k from z_i for every $i \in \{1, 2, \dots, p-1\}$.

Trivially, this notion of reachability is symmetric; that is, if an object x is reachable at level k from y, then y is reachable from x. Consequently, a partition of a database at discretization level k is naturally produced.

level-k partition

Definition 5.3: Level-k partition
A partition of a database $\tau = (H, X)$ is a *level-k partition*, denoted by $\mathcal{C}^k$, if it satisfies the following condition: For all pairs of objects $x, y \in \mathrm{set}(X)$, the pair is in the same cluster if and only if y is reachable at level k from x.

The level-k partition is determined uniquely. Intuitively, if we see each discretized object as a cluster, then adjacent clusters are agglomerated when $l = 1$. It takes $O(n^2 d)$ time to construct the level-k partition with the naïve version shown in Algorithm 5.1, where n is the number of objects and d is the number of attributes, since we have to calculate distances d_0 and d_∞ for all pairs of objects. In the next section, we discuss speeding up of this process by sorting the database.

Example 5.4
Consider the database τ given in Table 5.1. Fix $l = 1$. Then, level-1 partition $\mathcal{C}^1 = \{\{x_1, x_2, x_3\}\}$ since

$$d_0(\Delta^1(x_1), \Delta^1(x_2)) = 0 \text{ and } d_\infty(\Delta^1(x_1), \Delta^1(x_2)) = 0,$$
$$d_0(\Delta^1(x_1), \Delta^1(x_3)) = 1 \text{ and } d_\infty(\Delta^1(x_1), \Delta^1(x_3)) = 1,$$
$$d_0(\Delta^1(x_2), \Delta^1(x_3)) = 1 \text{ and } d_\infty(\Delta^1(x_2), \Delta^1(x_3)) = 1,$$

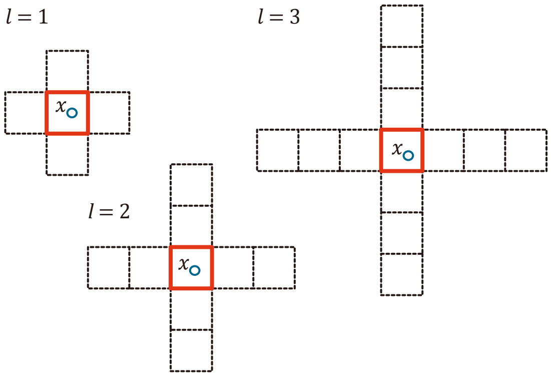


Figure 5.2 | Illustration of l-neighborhood. For each distance parameter l and an object x, if an object y is in a dotted square, $y \in N_l^k(x)$ defined in the equation (5.1).

hence $N_1^1(x_1) = \{x_2, x_3\}$, $N_1^1(x_2) = \{x_1, x_3\}$, and $N_1^1(x_3) = \{x_1, x_2\}$; x_2 and x_3 are reachable at level 1 from x_1. Level-2 partition $\mathcal{C}^2 = \{\{x_1, x_2\}, \{x_3\}\}$ since

$$d_0(\Delta^2(x_1), \Delta^2(x_2)) = 1 \text{ and } d_\infty(\Delta^2(x_1), \Delta^2(x_2)) = 1,$$
$$d_0(\Delta^2(x_1), \Delta^2(x_3)) = 2 \text{ and } d_\infty(\Delta^2(x_1), \Delta^2(x_3)) = 2,$$
$$d_0(\Delta^2(x_2), \Delta^2(x_3)) = 3 \text{ and } d_\infty(\Delta^2(x_2), \Delta^2(x_3)) = 2,$$

hence $N_1^2(x_1) = \{x_2\}$, $N_1^2(x_2) = \{x_1\}$, and $N_1^2(x_3) = \emptyset$; x_1 and x_2 are not reachable from x_3, and x_2 is reachable from x_1. □

Level-k partitions have a hierarchical structure, that is, for the level-k and $k+1$ partitions $\mathcal{C}^k$ and $\mathcal{C}^{k+1}$, the following condition holds: For every cluster $C \in \mathcal{C}^k$, there exists a set of clusters $\mathcal{D} \subseteq \mathcal{C}^{k+1}$ such that $\bigcup \mathcal{D} = C$. This is why, for two objects x and y, if y is directly reachable at level k from x, then y is directly reachable at level k' from x for all $k' \leq k$. BOOL uses this property; it increases discretization level k to make more than K clusters, where K is the input parameter denoting the lower bound on the number of clusters.

Furthermore, by using the lower bound on cluster size N, noise filtering can be done easily and directly.

> **Definition 5.5: Noise**
> Given a natural number $N \in \mathbb{N}$. Let $\mathcal{C}^k$ be the level-k partition of a database $\tau = (H, X)$. For each cluster $C \in \mathcal{C}^k$, every object in C is *noise* if $\#C < N$.

noise

We call N the *noise parameter*. For example, for the level-2 partition $\mathcal{C}^2$ in Example 5.4, the cluster $\{x_3\}$ is detected as noise if $N = 2$.

noise parameter

5.1.3 Relationship between BOOL and DBSCAN

The BOOL approach can be viewed as a mix of hierarchical clustering with density-based clustering. BOOL uses only L_0 and L_∞ metrics to define reachability and this restriction allows for sorting to be leveraged for clustering shown in the next section. Here we theoretically discuss the relationship between BOOL and DBSCAN (Ester *et al.*, 1996), which is a typical density-based clustering algorithm, with respect to the reachability.

First, we prepare the notion of density-reachability in DBSCAN referring the literature (Ester *et al.*, 1996). For an object x, the *Eps-neighborhood* of x is

Eps-neighborhood

$$N_{\mathrm{Eps}}(x) = \{y \in \mathrm{set}(X) \mid \mathfrak{d}(x, y) \leq \mathrm{Eps}\}, \tag{5.2}$$

directly density-reachable

where $\mathfrak{d}$ is an arbitrary distance function such as Euclidean distance. Then, an object x is *directly density-reachable* from y with respect to Eps and MinPts if

$$q \in N_{\mathrm{Eps}}(x) \text{ and } \#N_{\mathrm{Eps}(y)} \geq \mathrm{MinPts},$$

density-reachable

density-connected

and an object x is *density-reachable* from z with respect to Eps and MinPts if there is a chain of objects $z_1, z_2, \dots, z_p$ such that $z_1 = z$, $z_p = x$, and z_{i+1} is directly density-reachable from z_i. Here, in DBSCAN, each cluster is defined to be a set of density-connected points which is maximal with respect to density-reachability, where an object x is *density-connected* to an object y with respect to Eps and MinPts if there is an object o such that both x and y are density-reachable from o with respect to Eps and MinPts.

Compared to our approach, we can easily show the following relationship between BOOL and DBSCAN since, if MinPts $= 1$, the property of density-connectivity is exactly the same as density-reachability.

> **Theorem 5.6**
> *Given a database $\tau = (H, X)$. Assume that the distance parameter $l = 1$. The level-k partition $\mathcal{C}^k$ of τ is exactly the same as the output of DBSCAN for the discretized input database $\Delta^k(\tau)$ if the distance $\mathfrak{d}$ in the equation* (5.2) *is the Manhattan distance d_1 and parameters are set as* Eps $= l$ *and* MinPts $= 1$.

For $l > 1$, if we modify the definition of the Eps-neighborhood by

$$N_{\mathrm{Eps}}(x) = \{y \in \mathrm{set}(X) \mid d_0(x, y) \leq 1 \text{ and } d_\infty(x, y) \leq \mathrm{Eps}\},$$

we have the same result. In contrast, the definition of noise is different. Noise in DBSCAN is defined using MinPts as low density area, whereas BOOL does not use any density; no object is noise in BOOL if MinPts $= 1$.

The time complexity of DBSCAN is $O(n^2 d)$, thus it is difficult to apply it for massive data in data mining and knowledge discovery. It is known that the complexity can be reduced to $O(nd \log n)$ using k-d tree for data indexing, but practically DBSCAN is still computationally expensive (Han and Kamber, 2006). In the next section, we show that linear complexity with respect to the number of objects can be realized in BOOL by sorting binary representations of original data, which results in speedups when compared with DBSCAN and other algorithms.

Moreover, DBSCAN is quite sensitive to the parameter settings as described by Han and Kamber (2006). The hierarchical clustering of BOOL using the input parameter l of a *natural number* realizes robust clustering, and manual parameter setting in BOOL is quite easy.

5.2 Speeding Up of Clustering through Sorting

In this section, we discuss speeding up of the naïve version of BOOL by sorting the objects, resulting in linear time complexity. The pseudo-code of speeded-up BOOL is shown in Algorithm 5.2.

First, we introduce a sorting operator. We sort a given database τ by using values in the discretized database $\Delta^k(\tau)$.

Algorithm 5.2: Speeded-up BOOL

```
Input: Database $\tau = (H, X)$, lower bound on number of clusters $K$,
       noise parameter $N$, and distance parameter $l$
Output: Partition $\mathcal{C}$
function Bool($\tau, K, N$)
  $k \leftarrow 1$    // $k$ is discretization level
  repeat
    $\mathcal{C} \leftarrow$ MakeHierarchy($\tau, k, N$)
    $k \leftarrow k + 1$
  until $\#\mathcal{C} \geq K$
  output $\mathcal{C}$
function MakeHierarchy($\tau, k, N$)
  $\mathcal{C} \leftarrow \{\, \{x\} \mid x \in \mathrm{set}(X) \,\}$
  $X_\mathrm{D} \leftarrow \Delta^k(X)$    // discretize $X$ at level $k$
  $X \leftarrow S_{\sigma_1} \circ S_{\sigma_2} \circ \ldots \circ S_{\sigma_d}(X)$, where $\sigma_i = X_\mathrm{D}|_i$
  $h \leftarrow d$
  repeat
    $X \leftarrow S_\sigma(X)$ such that $\sigma = X_\mathrm{D}|_h$
    $\mathcal{C} \leftarrow$ Aggl($\tau, \mathcal{C}, h$)
    $h \leftarrow h - 1$
  until $h = 0$
  $\mathcal{C} \leftarrow \{C \in \mathcal{C} \mid \#C \geq N\}$
  output $\mathcal{C}$
function Aggl($\tau, \mathcal{C}, h$)
  for each object $x \in \mathrm{set}(X)$
    $y \leftarrow$ successive object of $x$
    if $y$ is directly reachable from $x$ (i.e., $y \in N_l(x)$) then
      delete clusters $C \ni x$ and $D \ni y$ from $\mathcal{C}$, and add $C \cup D$
    end if
  end for
  output $\mathcal{C}$
```

Definition 5.7: Sorting
Let $\tau = (H, X)$ be a database and σ be a sequence of natural numbers such that the size of σ is same as that of X. The expression $S_\sigma(\tau)$ is the database τ for which objects are sorted in the order indicated by σ, where ties keep the original order.

Obviously, this operation S can be realized by using a standard sorting algorithm such as quick sort. Here we only consider sorting of database τ by using a column in the discretized database $\Delta^k(\tau)$, so we can use *bucket sort* since the domain of each attribute of $\Delta^k(\tau)$ is from 0 to $2^k - 1$, and this size can reasonably fit in the bucket of the main memory for most real databases (k is usually less than 10 in practice). The operation of S should therefore take only $O(n)$ time, where n is the number of objects.

bucket sort

Table 5.2 | Database $\tau = (\{1,2,3\}, X)$ and sorted database $S_\sigma(\tau)$, where $\sigma = \Delta^2(X)|_2$.

τ				σ	$S_\sigma(\tau)$			
H	1	2	3		H	1	2	3
x_1	0.66	0.71	0.27	2	x_1	0.66	0.71	0.27
x_2	0.72	0.86	0.46	3	x_3	0.14	0.53	0.04
x_3	0.14	0.53	0.04	2	x_2	0.72	0.86	0.46

Example 5.8
Consider database τ given in Table 5.2 (same as in Table 5.1). and let σ be the sequence $\Delta^2(X)|_2$; *i.e.*, the column of the discretized database $\Delta^2(\tau) = (\{1,2,3\}, \Delta^2(X))$ for attribute 2. Then, $S_\sigma(\tau)$ becomes the database in Table 5.2. □

Next we describe the main theorem used for speeding up BOOL. It states clustering can be completed by comparing each object to the next object d times with sorting.

Theorem 5.9
For any database τ and any natural number k, the output of the function MakeHierarchy$(\tau, k, 0)$ *in Algorithm 5.2 is the level-k partition $\mathcal{C}^k$.*

Proof. Suppose that $\mathcal{C}$ is the output of the function MakeHierarchy. From the function Aggl, it is trivial that, for all pairs of objects x and y of τ such that they are in the same cluster in $\mathcal{C}$, the same holds in $\mathcal{C}^k$. Thus, it is enough to prove that, for all pairs of objects x and y, if y is directly reachable from x, they are in the same cluster in $\mathcal{C}$.

Fix two objects x and y such that y is directly reachable from x. We can assume that there is no object z such that $d_0(\Delta^k(z), \Delta^k(x)) = 0$ or $d_0(\Delta^k(z), \Delta^k(y)) = 0$ (*i.e.*, $z = x$ or $z = y$) without loss of generality. Then there must exist $h \in H = \{1, 2, \dots, d\}$ such that $|x(h) - y(h)| \leq l$ and $|x(i) - y(i)| = 0$ for all $i \in H$ with $i \neq h$. By sorting $S_\sigma(X)$ using $\sigma = \Delta^k(X)|_h$ (line 6 of the function), it is easy to see that y becomes the successive object of x since other attributes have already been sorted, so they are in the same cluster. Thus $\mathcal{C}$ is exactly the same as $\mathcal{C}^k$. □

We can easily check that the time complexity of the function MakeHierarchy of Algorithm 5.2 is $O(nd)$ in the best case and $O(nd^2)$ in the worst case, since checking the condition with the distances d_0 and d_∞ (line 3 of function Aggl) takes $O(1)$ time in the best case and $O(d)$ time in the worst case. Thus, the time complexity of BOOL is $O(ndk)$ or $O(nd^2k)$, where $\#\mathcal{C}^{k-1} < K$ and $\#\mathcal{C}^k \geq K$, and it is usually $O(nd)$ or $O(nd^2)$ since $k \ll n$ holds. Most real databases used in spatial clustering have only two or three attributes, so BOOL is fast although the complexity is quadratic with respect to the number of attributes in the worst case (actual running time is shown in next section).

Example 5.10
We explain the clustering process of the function MakeHierarchy in BOOL using an example shown in Figure 5.3. We consider the level-2 partition of a database τ (Figure 5.3, (a)) with the number of attributes $d = 2$, and assume that the distance parameter $l = 1$.

First, the database τ (Figure 5.3, (a)) is discretized to $\Delta^2(\tau)$ in line 2 in the function (Figure 5.3, (b)). Next, it is sorted two times by bucket sort for each attribute

(a) Original database τ

H	1	2
x_1	0.36	0.11
x_2	0.48	0.29
x_3	0.42	0.61
x_4	0.19	0.21
x_5	0.72	0.88
x_6	0.92	0.63
x_7	0.40	0.51
x_8	0.97	0.03
x_9	0.36	0.87
x_{10}	0.77	0.81

→

(b) Discretized database $\Delta^2(\tau)$

H	1	2
x_1	1	0
x_2	1	1
x_3	1	2
x_4	0	0
x_5	2	3
x_6	3	2
x_7	1	2
x_8	3	0
x_9	1	3
x_{10}	3	3

→

(c) Sort by attribute 2

H	1	2
x_1	1	0
x_4	0	0
x_8	3	0
x_2	1	1
x_3	1	2
x_6	3	2
x_7	1	2
x_5	2	3
x_9	1	3
x_{10}	3	3

→

(d) Sort by attribute 1

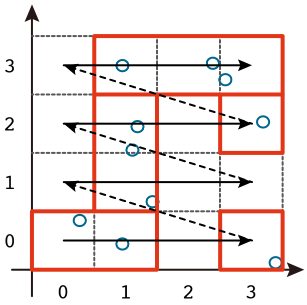
H	1	2
x_4	0	0
x_1	1	0
x_2	1	1
x_3	1	2
x_7	1	2
x_9	1	3
x_5	2	3
x_8	3	0
x_6	3	2
x_{10}	3	3

→

(e) Sort by attribute 2 and compare each object to the subsequent object

→

H	1	2	Class
x_4	0	0	A
x_1	1	0	A
x_8	3	0	B
x_2	1	1	C
x_3	1	2	D
x_7	1	2	D
x_6	3	2	E
x_9	1	3	F
x_5	2	3	F
x_{10}	3	3	F



→

(f) Sort by attribute 1 and compare each object to the subsequent object

→

H	1	2	Class
x_4	0	0	A
x_1	1	0	A
x_2	1	1	A
x_3	1	2	A
x_7	1	2	A
x_9	1	3	A
x_5	2	3	A
x_8	3	0	B
x_6	3	2	A
x_{10}	3	3	A

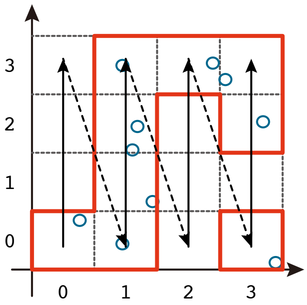


Figure 5.3 | Illustrative example of clustering process by speeded-up BOOL.

radix sort

in line 3 (Figure 5.3, (c) and (d)). This sorting is exactly the same as *radix sort* if we see each discretized object $\Delta^2(x) = m$ as the integer $10m(1) + m(2)$. Then, the discretized database is sorted again together with merging clusters in lines 5–9 (Figure 5.3, (e) and (f)). For each attribute h, BOOL sorts the database by bucket sort in advance, and compare each object to the subsequent object. If one object is directly reachable from the other object, BOOL categorizes these objects into the same cluster (*i.e.*, gives the same class label). By repeating this clustering process for each attribute, every pair of objects are in the same cluster if and only if one is reachable from the other. In this example, by sorting with attribute 2, six clusters A, B, C, D, E, and F are found (Figure 5.3, (e)) and, by sorting with attribute 1, five clusters A, C, D, E, and F are merged into one cluster (Figure 5.3, (f)). □

5.3 Experiments

We evaluate BOOL experimentally to determine its scalability and effectiveness for various types of databases including: synthetic databases with two attributes, real databases with three attributes generated from natural images, large real databases with two attributes generated from geospatial images, and real databases with various sizes of attributes from the UCI repository.

5.3.1 Methods

Environment

We used Mac OS X version 10.6.5 with two 2.26-GHz Quad-Core Intel Xeon CPUs and 12 GB of memory. BOOL was implemented in C and compiled with `gcc` 4.2.1. All experiments were performed in the R environment, version 2.12.2 (R Development Core Team, 2011).

Databases

Synthetic and real databases were used. To evaluate scalability with respect to the size of the databases and the number of clusters, synthetic databases were generated randomly using the R `clusterGeneration` package (Qiu and Joe, 2006), with parameter *sepVal* set to 0.34. Each database had two attributes ($d = 2$), and the number of objects (n) and clusters (K) varied from 100 to 1,000,000 and from 5 to 40, respectively. We used databases with $n = 10{,}000$ and $K = 5$ to evaluate the robustness of BOOL. Each database contained $n/100$ outliers (noise points). The databases were generated 20 times, and the mean and s.d. (standard deviation) were calculated. Four synthetic databases (DS1, DS2, DS3, and DS4) were taken from the CLUTO website[1]; they had been used as benchmarks in the evaluations of other spatial clustering algorithms such as CHAMELEON (Karypis *et al.*, 1999), SPARCL (Chaoji *et al.*, 2009), and ABACUS (Chaoji *et al.*, 2011).

The real databases consisted of four natural images, four large satellite images, and six databases from the UCI machine learning repository (Frank and Asuncion, 2010). The natural images (shown in Figure 5.7, (a), (d), (g), and (j)) were

[1] http://glaros.dtc.umn.edu/gkhome/views/cluto/

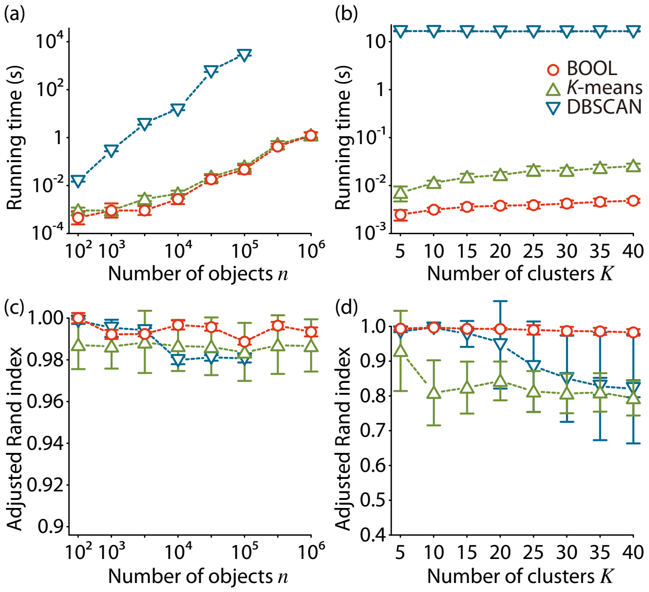


Figure 5.4 | Clustering speed and quality for randomly generated synthetic databases, where $K = 5$ for (a) and (c) and $n = 10{,}000$ for (b) and (d). Adjusted Rand index should be maximized. Data show mean ± s.d.

taken from the website of the Berkeley segmentation database and benchmark[2] (Martin *et al.*, 2001); they were the same ones used as benchmarks in a previous study by Chaoji *et al.* (2011). Each image was 481 × 321 in size, and 154,401 pixels in total, and the RGB (red-green-blue) values for each pixel were obtained by pre-precessing. Finally, each image was translated into a database with 3 attributes and 154,401 objects. Four another real databases were collected from the Earth-as-Art website[3], which contains geospatial satellite images. Similar databases were used in the literature (Chaoji *et al.*, 2011). Each image was originally composed of $7,296 \times 7,296$ pixels (Figures 5.8 (a), (e), (i), and (m)) and was translated into the binary image using ImageJ software (Rasband, 1997–2011) (Figures 5.8 (b), (f), (j), and (n)). The six UCI databases (ecoli, sonar, shuttle, wdbc, wine, and wine quality) were used to test the effectiveness of BOOL for databases with various attribute sizes.

Every database was translated using min-max normalization (Han and Kamber, 2006) in advance so that each value was in the unit interval [0, 1], The runtime for this translation was included in the BOOL running time.

Control Algorithms

We used K-means and DBSCAN as control algorithms. K-means is a commonly used and efficient clustering algorithm, and DBSCAN is a well-known and typical spatial clustering algorithm. DBSCAN was executed using the R `fpc` package. For databases DS1 - DS4 and the natural image databases, we also compared the running time of BOOL with those of state-of-the-art clustering algorithms ABACUS (Chaoji *et al.*, 2011) and SPARCL (Chaoji *et al.*, 2009). Their source codes are not publicly available, so we simply used the reported results (Chaoji *et al.*, 2011) to avoid implementation biases. The four synthetic databases and four natural im-

[2] `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/`
[3] `http://eros.usgs.gov/imagegallery/`

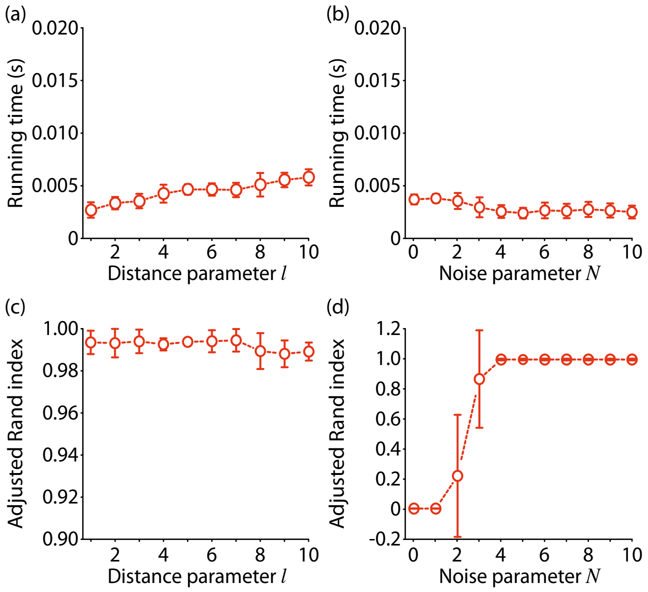


Figure 5.5 | Clustering speed and quality for various values of distance parameter l and noise parameter N. Adjusted Rand index should be maximized. Data show mean ± s.d.

ages are identical to those used in a previous evaluation (Chaoji *et al.*, 2011), so the comparison here is reasonable.

Evaluation

Clustering quality was evaluated using the adjusted Rand index (takes values in $[-1, 1]$, to be maximized) (Hubert and Arabie, 1985), which takes into account the ground truth and is popular in the clustering literature. For the synthetic databases DS1 - DS4, natural images, and satellite images, we simply show scatter and contour plots of the results since we had no information on the ground truth. Moreover, the number of clusters was not clear for the natural images, so we used the MakeHierarchy function in Algorithm 5.2 to obtain clusters at some hierarchy level. For K-means, we used the reported number of clusters (Chaoji *et al.*, 2011). All parameters for BOOL and DBSCAN were tuned to obtain the best results.

5.3.2 Results and Discussion

As discussed below, our proposed algorithm, BOOL, shows the best performance in terms of both clustering speed and quality. The results indicate that BOOL is efficient and effective for spatial clustering.

The clustering speed and quality results for randomly generated synthetic databases are plotted in Figure 5.4. For BOOL, the distance and noise parameters, l and N, were set to 1 and $n/100$, respectively. In most cases, BOOL shows the best performance. BOOL is faster than K-means and from two to six orders of magnitude faster than DBSCAN. Moreover, the quality of the obtained clusters is higher than those for K-means and DBSCAN in terms of the adjusted Rand index. The clustering speed and quality of BOOL for various values of N and l are plotted in Figure 5.5, where N was fixed to 100 for Figures 5.5, (a) and (c), and l to 1 for Figures 5.5, (b) and (d). Every database contains noise points, and BOOL treats each one as a cluster if N is small ($N = 0, 1, 2,$ and 3). This is why the adjusted Rand index is low for

Name	n	d	K	BL	KM	DB	AB	SP
DS1	8000	2	6	**0.004**	0.008	9.959	1.7	1.8
DS2	8000	2	6	**0.004**	0.008	10.041	1.3	1.5
DS3	10000	2	9	**0.010**	0.036	15.832	1.9	2.5
DS4	8000	2	8	**0.005**	0.018	9.947	1.7	1.8
Horse	154401	3	–	**0.253**	0.674	–	31.2	41.8
Mushroom	154401	3	–	**0.761**	1.449	–	29.3	–
Pyramid	154401	3	–	**0.187**	0.254	–	11.3	–
Road	154401	3	–	**0.180**	0.209	–	14.9	–
Delta	27543260	2	6	**34.398**	273.231	–	–	–
Europe	23071720	2	6	**27.607**	44.985	–	–	–
Norway	33879834	2	4	**23.866**	46.302	–	–	–
Ganges	24546151	2	8	**32.977**	58.273	–	–	–

Table 5.3 | Running time (in seconds) for synthetic databases, real databases from natural images, and real databases from geospatial satellite images with BOOL (BL), K-means (KM), DBSCAN (DB), ABACUS (AB), and SPARCL (SP). In table, n, d, and K denote number of objects, attributes, and clusters, respectively. Results for ABACUS and SPARCL are taken from the literature (Chaoji *et al.*, 2011). Clustering results are shown in Figures 5.6, 5.7, and 5.8.

small N. These results show that BOOL is robust with respect to N and l and, for lots of databases, all we have to do is set l as 1 and N large enough ($N \geq 4$).

The results for the synthetic databases DS1–DS4 are summarized in Table 5.3 and illustrated in Figure 5.6. For all databases, BOOL is the fastest and about three orders of magnitude faster than the two state-of-the-art clustering algorithms, ABACUS and SPARCL, when $(l, N) = (1, 100)$, $(1, 100)$, $(7, 100)$, and $(11, 40)$ for DS1, DS2, DS3, and DS4, respectively. BOOL finds all reasonable clusters in DS1 and DS2. It also effectively filters out the noise, as shown in Figure 5.6, whereas K-means does not. Note that ABACUS and SPARCL cannot detect noise effectively, as shown in previous studies (Chaoji *et al.*, 2009, 2011). BOOL does not find some clusters in DS3 and DS4, which is counterintuitive. The reason is that such clusters are connected by dense noise, so BOOL cannot divide them into smaller clusters. However, the results are still much better than those with K-means. These results indicate that BOOL is the fastest spatial clustering algorithm that can find arbitrarily shaped clusters for use in typical spatial clustering tasks.

For the real databases obtained from the natural images, BOOL again shows the best performance; *i.e.*, it is the fastest of the compared clustering algorithms (Table 5.3) for $(k, l, N) = (7, 1, 50)$, $(7, 15, 20)$, $(8, 5, 100)$, and $(10, 15, 400)$ for Horse, Mushroom, Pyramid, and Road, respectively (k is the hierarchy level). BOOL is faster than K-means, and about two orders of magnitude faster than ABACUS and SPARCL (we did not perform DBSCAN since it takes too much time and needs manual tuning of the input parameters). Moreover, the quality of the clustering is much better than that with K-means, as shown in Figure 5.7. These results show that BOOL is efficient and effective for spatial clustering for natural images. Furthermore, they show that BOOL works well for hierarchical clustering, meaning that we can input the hierarchy level directly rather than entering the lower bound on the number of clusters.

For all the real databases from geospatial satellite images, BOOL is faster than K-means and finds natural clusters (parameters l and N were set as 1 and 100000, respectively, for all the databases), as shown in Figures 5.8, (c), (g), (k), and (o), whereas K-means results in inferior clustering quality (Figures 5.8, (d), (h), (l), and (p)). These results show that BOOL has high scalability and that it can find arbitrarily shaped clusters without careful tuning of the input parameters.

Table 5.4 | Results for UCI databases with BOOL (BL), *K*-means (KM) and DBSCAN (DB). In table, *n*, *d*, and *K* denote the number of objects, attributes, and clusters, respectively. For all databases, BOOL showed the equal or better performance compared to *K*-means and DBSCAN.

Name	n	d	K	Running time			Adjusted Rand index		
				BL	KM	DB	BL	KM	DB
ecoli	336	7	8	**0.001**	0.002	0.111	**0.5745**	0.4399	0.1223
sonar	208	60	2	**0.004**	0.005	0.149	**0.0133**	0.0064	0.0006
shuttle	14500	9	7	**0.025**	0.065	–	**0.7651**	0.1432	–
wdbc	569	30	2	**0.004**	**0.004**	0.222	**0.6806**	0.4914	0.5530
wine	178	13	3	**0.002**	**0.002**	0.047	**0.4638**	0.3347	0.2971
wine q.	4898	11	7	**0.019**	0.026	7.601	**0.0151**	0.0099	0.0134

Finally, for the UCI databases, BOOL again shows the best performance compared to K-means and DBSCAN (Table 5.4), showing that BOOL works well not only for low-dimensional databases but also for high-dimensional databases. These results show that BOOL should work efficiently and effectively for various types of databases including small databases, large databases, and databases containing many clusters.

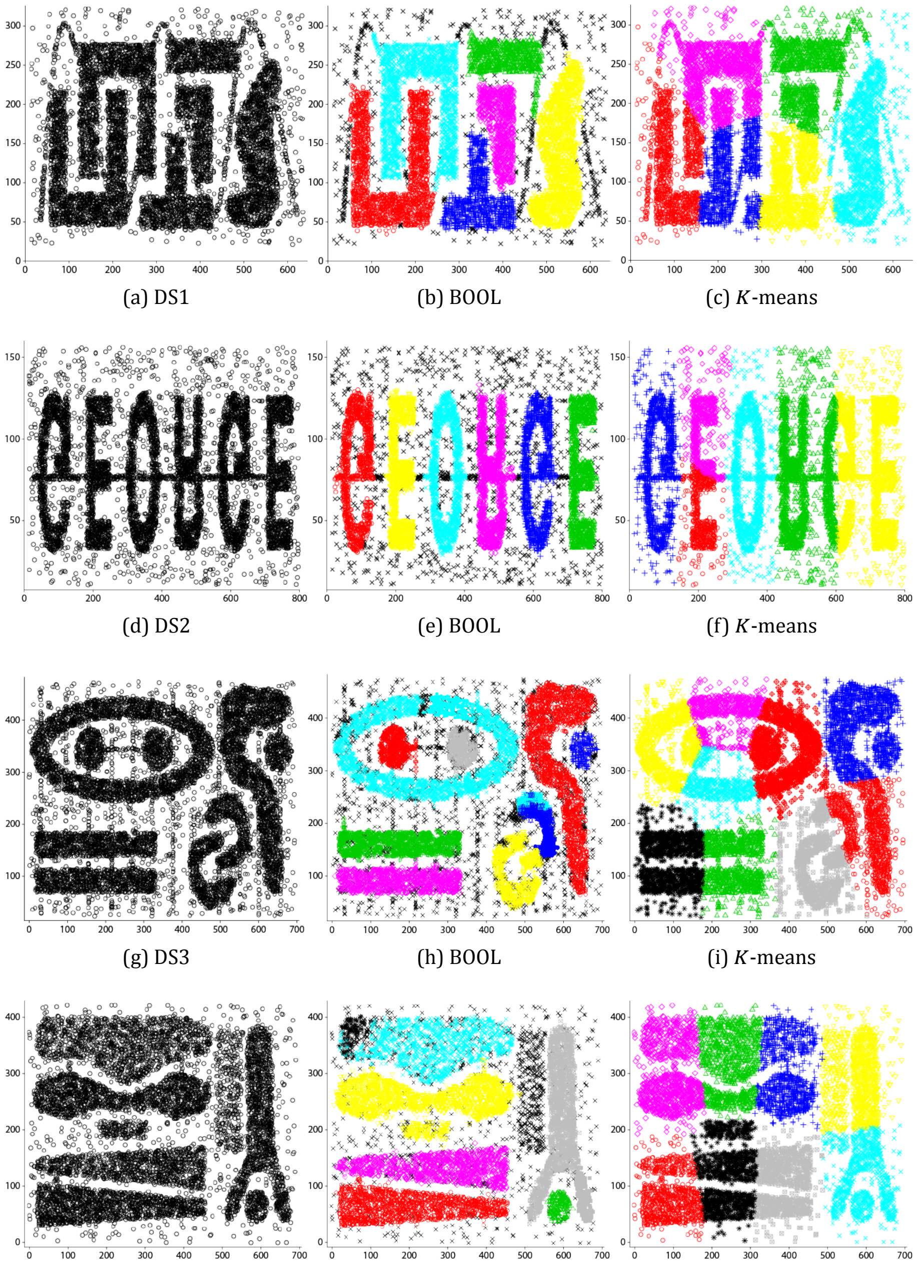
5.4 Related Work

discretization

From the point of view of *discretization*, several techniques (Fayyad and Irani, 1993; Liu *et al.*, 2002; Skubacz and Hollmén, 2000) have been proposed to treat multivariate data in the discrete manner. In particular, recently Lin *et al.* (2007, 2003) have been proposed SAX, symbolic representation for time series data, which is becoming a more and more popular discretization technique. They showed that the quality of clusters becomes better with SAX in both agglomerative hierarchical clustering and partitional clustering using K-means. However, since the above discretization methods including SAX are just data preprocessing, it is difficult to directly achieve efficient spatial clustering for finding arbitrarily shaped clusters. On the other hand, in our BOOL approach, the discretization process is integrated into the clustering process and clusters are efficiently detected with effective use of sorting (see Example 5.10 and Figure 5.3). Informally, the geometric information of "topology" (*i.e.*, metric) of the Euclidean space is appropriately *embedded* into the discrete space through the binary encoding, and this embedding enables effective and efficient spatial clustering with binary words of original data.

Many clustering algorithms have been proposed for arbitrary shape clustering (Berkhin, 2006; Halkidi *et al.*, 2001; Jain *et al.*, 1999a), and some notable clustering algorithms, SPARCL (Chaoji *et al.*, 2009) and ABACUS (Chaoji *et al.*, 2011), have recently been proposed for massive databases. Although we do not compare BOOL with these algorithms in exactly the same way, our experiments clearly shows that BOOL is the fastest among these state-of-the-art clustering algorithms and that it is robust to the input parameters. Other new approaches include a mass-based clustering MassTER algorithm (Ting and Wells, 2010). It also has linear time complexity with respect to the data size, but their results indicate that it is much slower than BOOL. For instance, MassTER takes 59 seconds for a database with $n = 70{,}000$ and $d = 3$.

In the following, we briefly review spatial (shape-based) clustering algorithms, including density-based clustering algorithms, agglomerative hierarchical clustering algorithms, and grid-based algorithms, and compare them to BOOL.



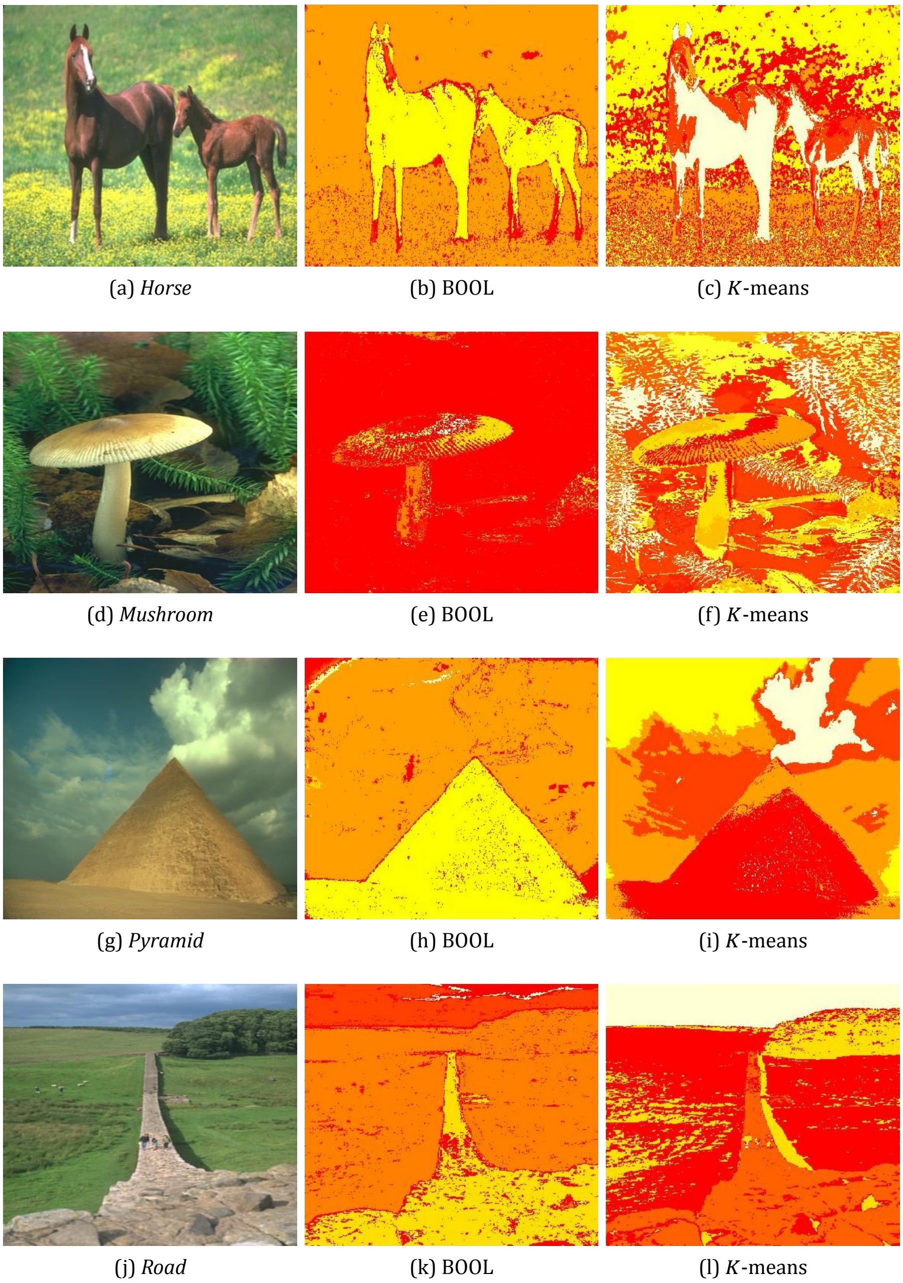
(a) DS1 (b) BOOL (c) *K*-means

(d) DS2 (e) BOOL (f) *K*-means

(g) DS3 (h) BOOL (i) *K*-means

(j) DS4 (k) BOOL (l) *K*-means

Figure 5.6 | Synthetic databases DS1 - DS4 and clustering results obtained using BOOL and *K*-means.



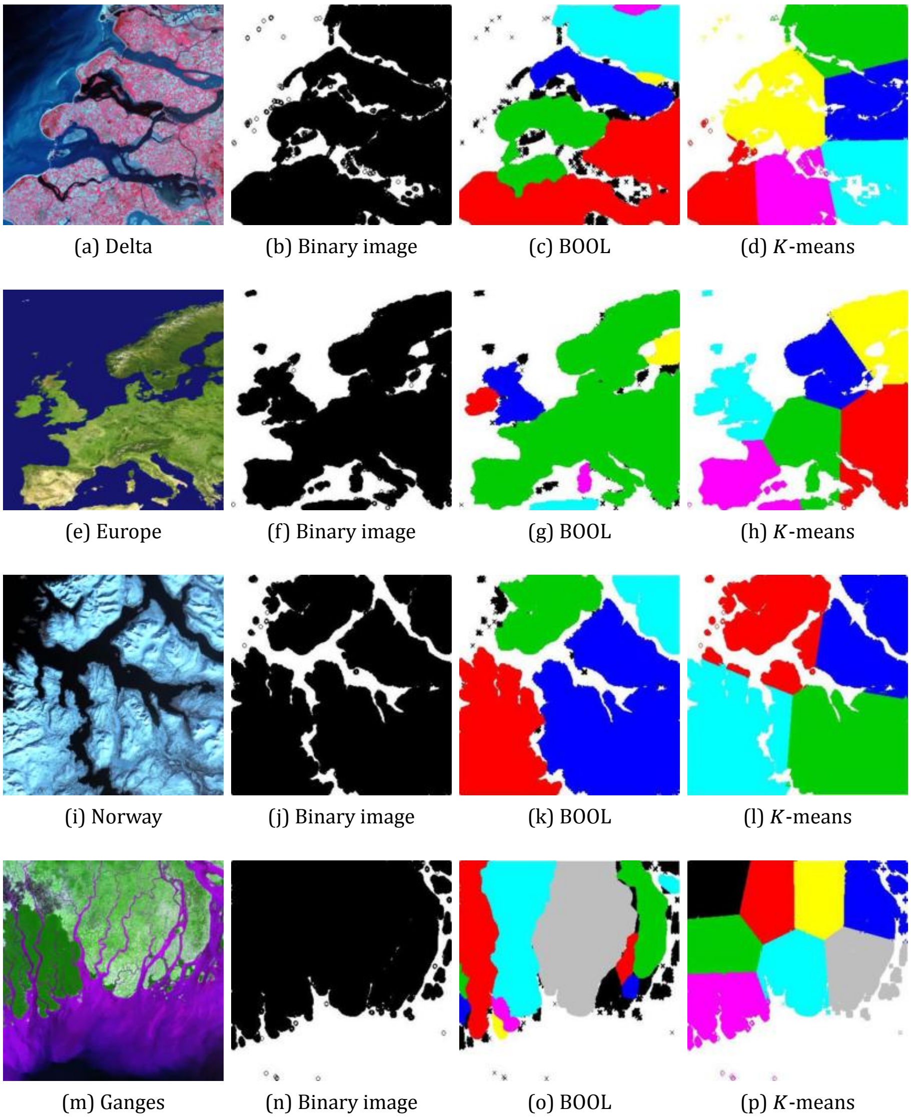
(a) *Horse* (b) BOOL (c) K-means

(d) *Mushroom* (e) BOOL (f) K-means

(g) *Pyramid* (h) BOOL (i) K-means

(j) *Road* (k) BOOL (l) K-means

Figure 5.7 | Four natural images and corresponding contour maps obtained using BOOL and K-means from preprocessed databases with three attributes (RGB). Pixels of the same color are in the same cluster.


(a) Delta (b) Binary image (c) BOOL (d) K-means

(e) Europe (f) Binary image (g) BOOL (h) K-means

(i) Norway (j) Binary image (k) BOOL (l) K-means

(m) Ganges (n) Binary image (o) BOOL (p) K-means

Figure 5.8 | Results for geospatial satellite images obtained using BOOL and K-means. BOOL finds most of natural clusters whereas K-means does not.

Density-based clustering, such as that used in DBSCAN (Ester *et al.*, 1996) and DENCLUE (Hinneburg and Keim, 1998), is one of the most commonly used approaches to detect arbitrarily shaped clusters. However, it is quite sensitive to the parameter settings, as described by (Han and Kamber, 2006) and it is computationally expensive. Despite BOOL and DBSCAN have close relationship as shown in Section 5.1.3, our experiments show that BOOL is robust and much faster than DBSCAN (see Section 5.3).

Guha *et al.* (1998) introduced the hierarchical agglomerative clustering algorithm CURE for finding arbitrarily shaped clusters. The key to CURE is data partitioning to obtain fine cluster representatives. This approach is similar to ours since we obtain binary words as representatives by discretization and agglomerate clusters by using the obtained representatives. However, the time complexity of CURE is large and quadratic with respect to data size, so CURE needs data sampling when it is applied to massive data sets. In contrast, the time complexity of BOOL is linear, and it is much faster than CURE. Moreover, it can perform exact clustering on massive data sets. Another hierarchical approach, CHAMELEON (Karypis *et al.*, 1999), achieves spatial clustering by using a graph partitioning algorithm. However, several parameters must be tuned for effective clustering.

Unlike CURE, which is based on data partitioning, grid-based algorithms such as BANG (Schikuta and Erhart, 1997), STING (Wang *et al.*, 1997), and WaveCluster (Sheikholeslami *et al.*, 1998) focus on space partitioning. They summarize data information into a grid structure and achieve clustering by using this structure. For example, STING assembles summary statistics into a tree of grid-cells. This idea of data summarization using space partitioning is the same as in BOOL. However, compared to other such algorithms, BOOL is very simple — it simply discretizes each data point by using binary encoding, without using statistics. This simplicity enables it to achieve fast and robust clustering by effectively using data sorting. In contrast, BOOL shares the major issue about *high-dimensional data* with the above space partitioning algorithms. High-dimensional data are usually too sparse to clustering since the number of partitioned regions exponentially increases with respect to the number of dimensions. In the extreme case in BOOL, every object is not reachable from any other objects even at discretization level 1, hence BOOL does not work well for such high-dimensional data. Thus combining other techniques such as dimension reduction methods is needed.

5.5 Summary

We have developed a spatial clustering algorithm called BOOL that is the fastest such algorithm with respect to finding arbitrarily shaped clusters. It is robust, highly scalable, and noise tolerant, and especially effective for low-dimensional data such as geographical images. The key to performance is discretization based on the binary encoding and sorting of data points. BOOL can be viewed as a restricted version of DBSCAN, which is a typical density-based clustering algorithm, but our hierarchical approach with discretization realizes not only highly speeding up but also robust clustering which can be easily handled by the user.

BOOL is simple, hence it can be extended to other data mining tasks such as anomaly detection, semi-supervised learning, and structured data clustering. Furthermore, BOOL can be used for preliminary exploration, especially for various tasks in computer vision and pattern recognition from massive images.

Part III

With Formal Concept Analysis

“je n’y emploie point d’autres caractères que 0 et 1,
et puis allant à deux, je recommence”

— Gottfried Wilhelm Leibniz, *Explication De L’arithmétique Binaire*

6

SEMI-SUPERVISED CLASSIFICATION AND RANKING

mixed-type data

NUMEROUS MIXED-TYPE DATA including both discrete (binary or nominal) and continuous (real-valued) variables are collected by researchers in various research domains from biology to economics. However, despite recent rapid development of many data analysis techniques in the fields of machine learning, data mining, and knowledge discovery, only few algorithms such as the decision tree-based classifier (Murthy, 1998) can directly handle such mixed-type data. In particular, to the best of our knowledge, no learning algorithm treats mixed-type data in a *semi-supervised* manner based on discrete approaches.

semi-supervised learning

Semi-supervised learning is a special form of classification (Zhu and Goldberg, 2009; Chapelle *et al.*, 2006); a learning algorithm uses both labeled and unlabeled data to learn classification rules. In real tasks, it is often difficult to obtain enough labeled data since the task of labeling has a high cost in terms of time and money, whereas lots of unlabeled data can be collected easily. The goal of semi-supervised learning is to construct a better classifier using such large amount of unlabeled data together with labeled data in short supply.

Formal Concept Analysis (FCA)

closed set lattice

To effectively use unlabeled mixed-type data for learning (training), we in this chapter propose a novel semi-supervised learning algorithm, called SELF (SEmi-supervised Learning via Formal Concept Analysis), which can directly treat mixed-type data. SELF adopts a popular semi-supervised learning strategy, called cluster-and-label (Dara *et al.*, 2002; Demiriz *et al.*, 1999), where a clustering algorithm is first applied, followed by labeling each cluster using labeled data. One of the remarkable features of SELF is that it performs the clustering process using Formal Concept Analysis (FCA) (Davey and Priestley, 2002; Ganter and Wille, 1998), which is a mathematical theory for data analysis and knowledge representation introduced by Wille (1982). Recently, Pasquier *et al.* (1999) proposed to use closed patterns (itemsets) obtained by FCA as condensed "lossless" representations of patterns. This new approach has been the subject of further research and extensions (Beslon *et al.*, 2010; Plantevit *et al.*, 2010; Saquer and Jitender, 2010; Zaki, 2000). In SELF, the labeling process is performed on a *closed set lattice*, which is the result of FCA. Informally, this structure describes the maximally general classification rules that explain the training data, thus preventing overfitting. Moreover, each classification rule can be *weighted* using the closed set lattice by counting the number of clusters classified by the rule, resulting in the *preference* of class labels as a partial order of them for each unlabeled datum. Furthermore, FCA and closed set lattices enable us to naturally treat incomplete data including missing values.

preference

To summarize, this chapter provides a contribution to both the fields of semi-supervised learning and FCA:

1. *To semi-supervised learning*: we present a novel approach based on an algebraic framework without assuming any data distribution.
2. *To FCA*: we study a novel application, semi-supervised learning, using FCA and closed set lattices.

The behavior of SELF is outlined as a flowchart in Figure 6.1, and this chapter is organized along it after discussing about related work in Section 6.1. The data preprocessing phase to construct a context from a given dataset to apply FCA is explained in Subsection 6.2.1. Missing values are handled in this phase. The learning phase is described in Subsections 6.2.2 and 6.2.3; Subsection 6.2.2 shows data clustering and making closed set lattices by FCA and Subsection 6.2.3 explains the training algorithm of SELF to learn classification rules. Classification by learned rules is considered in Subsection 6.2.4. Section 6.3 gives empirical evaluation of SELF and, finally, the key points and future work are summarized in Section 6.4.

6.1 Related Work

Many studies used FCA for machine learning (Kuznetsov, 2004), such as classification (Ganter and Kuznetsov, 2000, 2003), clustering (Zhang *et al.*, 2008), frequent pattern and association rule mining (Jaschke *et al.*, 2006; Pasquier *et al.*, 1999; Valtchev *et al.*, 2004), and bioinformatics (Blinova *et al.*, 2003; Kaytoue *et al.*, 2011b; Kuznetsov and Samokhin, 2005). Ganter and Kuznetsov (2000) investigated the problem of binary classification of real-valued data and proposed algorithms based on the *JSM-method* that produces hypotheses (classifiers) using positive and negative examples. Their idea of using the lattice structure derived by FCA for classification is similar to our approach, but the way of treating continuous variables is different. Their method discretizes continuous variables by inequations, called *conceptual scaling* (Ganter and Wille, 1998), that are given *a priori*, while SELF automatically discretizes them along with the learning process and no background knowledge and assumption about data are needed.

JSM-method

conceptual scaling

On the other hand, in machine learning context, decision tree-based algorithms such as C4.5 (Quinlan, 1993, 1996) can directly treat mixed-type data by discretizing continuous variables, and there are several discretization techniques (Fayyad and Irani, 1993; Liu *et al.*, 2002; Skubacz and Hollmén, 2000) to treat continuous variables in a discrete manner. Our approach is different from them since we integrate discretization process into learning process and avoid overfitting using closed set lattices effectively. SELF uses cluster-and-label, or called label propagation, which is a popular approach in semi-supervised learning as mentioned in the beginning of this chapter (Cheng *et al.*, 2009; Dara *et al.*, 2002; Demiriz *et al.*, 1999; Hwang and Kuang, 2010; Wang and Zhang, 2006). First SELF makes clusters without label information by FCA, followed by giving preferences of class labels for each cluster. However, to date, most of such approaches are designed for only continuous variables and, to the best of our knowledge, no semi-supervised learning

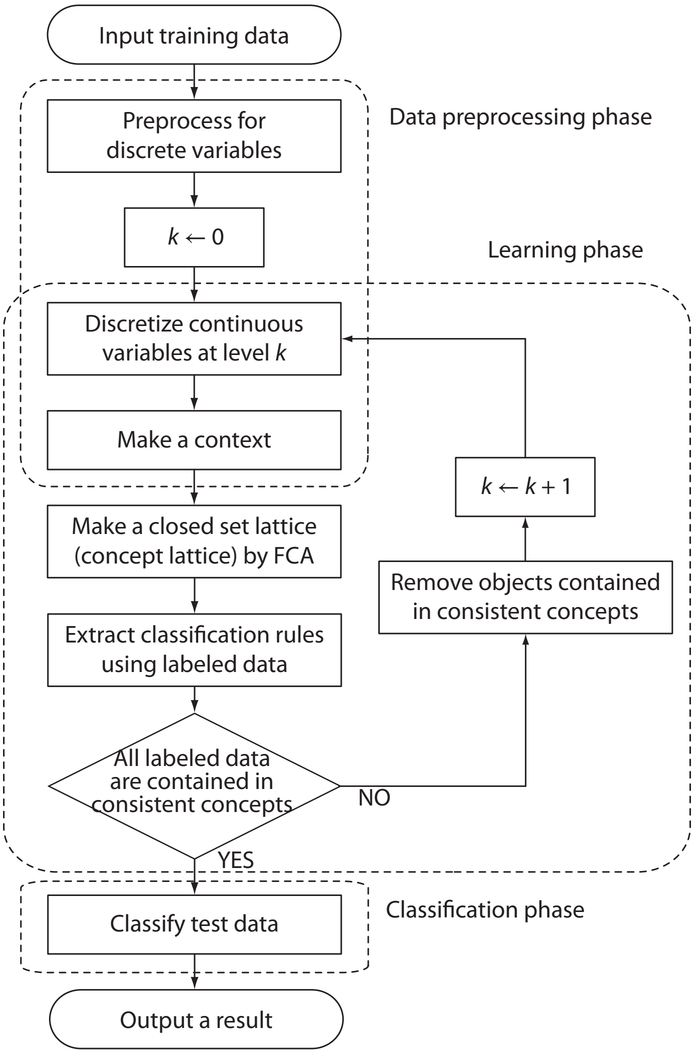

Input training data
Data preprocessing phase
Preprocess for discrete variables
$k \leftarrow 0$
Learning phase
Discretize continuous variables at level k
Make a context
$k \leftarrow k+1$
Make a closed set lattice (concept lattice) by FCA
Remove objects contained in consistent concepts
Extract classification rules using labeled data
All labeled data are contained in consistent concepts
NO
YES
Classify test data
Classification phase
Output a result

Figure 6.1 | A flowchart of the proposed SELF algorithm. It learns classification rules from training data and applies them to classify test data. Here we say that a concept is *l*-consistent if all labels contained in the concept are same.

algorithm based on cluster-and-label can treat mixed-type data including discrete variables appropriately. Since SELF uses FCA for clustering, it needs no distance calculation and no data distribution, which is one of features of SELF.

There exists only one study by Kok and Domingos (2009) which is related to the idea of putting original data on lattices. They proposed a learning algorithm via hypergraph lifting, which constructs clusters by hypergraphs and learns on them. Their idea is thus similar to ours since we also "lift" raw data to the space of a closed set lattice via FCA. However, it is difficult to treat continuous variables in their approach, thereby our approach can be more useful for machine learning and knowledge discovery from mixed-type data.

SELF achieves not only semi-supervised learning but also label ranking using the preference for each class label. Recently, the concept of preference has attracted more and more attention in artificial intelligence including machine learning and knowledge discovery, resulting in formalization of the research topic of "preference learning" (Fürnkranz and Hüllermeier, 2010). In particular, label ranking (Cheng *et al.*, 2010; Hülermeier *et al.*, 2008; Vembu and Gärtner, 2010) has been treated in preference learning as an extension of traditional supervised classification, where the objective is to obtain a ranker which gives a (partial) order of labels for each datum. SELF is the first algorithm that treats label ranking of mixed-type data by weighting each classification rule through closed set lattices.

6.2 The SELF Algorithm

We present the SELF algorithm in this section, which is the main part of this chapter. The behavior of SELF is illustrated in Figure 6.1; first it performs data preprocessing to make a context from a given dataset, second it constructs concept lattices by FCA, and third it learns the preference for each class label. Notations about databases are same as those in Chapter 5 (see Subsection 5.1.1).

6.2.1 Data Preprocessing

The aim of data preprocessing is to construct a (formal) context, a binary matrix specifying a set of objects and their attributes, to apply FCA to training data. Since we are interested in mixed-type data, we consider two types of variables, *discrete* and *continuous*. Formally, if a feature $h \in H$ is discrete, $\mathrm{Dom}(h) = S \cup \{\bot\}$ for some countable set S. For instance, $S = \{\mathbf{T}, \mathbf{F}\}$ if the feature h is binary and S is a (finite) set of symbols if j is nominal (categorical). If h is continuous, $\mathrm{Dom}(h) = \mathbb{R} \cup \{\bot\}$, where $\mathbb{R}$ is the set of real numbers.

> **Definition 6.1: Context**
> In FCA, we call a triplet (G, M, I) *context*. Here G and M are sets and $I \subseteq G \times M$ is a binary relation between G and M. The elements in G are called *objects*, and those in M are called *attributes*.

context

For a given table $\tau = (H, X)$, we always identify the set of objects G with $\mathrm{set}(X) = \{x_1, x_2, \dots, x_n\}$.

In the data preprocessing, for each feature $h \in H$ of a table τ, we independently construct a context (G, M_h, I_h) and combine them into a context (G, M, I). For this process, we always *qualify* attributes to be disjoint by denoting each element m of the attribute M_h by $h.m$ following the notations used in the database systems literature (Garcia-Molina *et al.*, 2008).

qualify

First, we focus on preprocessing for discrete variables. Since a context is also a discrete representation of a dataset, this process is directly achieved as follows: For each feature h, the set of attributes

$$M_h = \{\, h.m \mid m \in \mathrm{Dom}(h) \setminus \{\bot\} \,\}$$

and, for each value $x_i(h)$,

$$(x_i, h.m) \in I_h \text{ if and only if } x_i(h) = m.$$

In this way, discrete values are translated into a context and missing values are naturally treated. Algorithm 6.1 performs this translation.

Algorithm 6.1: Data preprocessing for discrete variables

```
Input: Table $\tau = (H, X)$ whose variables are discrete
Output: Context $(G, M_{\mathrm{D}}, I_{\mathrm{D}})$

function ContextD($\tau$)
    $G \leftarrow \mathrm{set}(X)$
    for each $h \in H$
        $M_h \leftarrow \{\, h.m \mid m \in \mathrm{Dom}(h) \setminus \{\bot\} \,\}$
        $I_h \leftarrow \{\, (x, h.x(h)) \mid x \in G$ and $x(h) \neq \bot \,\}$
    end for
    combine all $(G, M_h, I_h)$ with $h \in H$ into $(G, M_{\mathrm{D}}, I_{\mathrm{D}})$
    return $(G, M_{\mathrm{D}}, I_{\mathrm{D}})$
```

Example 6.2
Given a table $\tau = (H, X)$ with $H = \{1, 2, 3\}$ and $X = x_1, x_2$ such that

$$(x_1(1), x_1(2), x_1(3)) = (\mathbf{T}, \bot, \mathrm{C}),$$
$$(x_2(1), x_2(2), x_2(3)) = (\mathbf{F}, \mathbf{F}, \bot).$$

This table can be represented in the following manner.

H		1	2	3
X	x_1	**T**	$\bot$	C
	x_2	**F**	**F**	$\bot$

The domains are given as $\mathrm{Dom}(1) = \mathrm{Dom}(2) = \{\mathbf{T}, \mathbf{F}\}$ and $\mathrm{Dom}(3) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$. Here we have

$$\begin{aligned} G &= \{x_1, x_2\}, \\ (M_1, I_1) &= (\{1.\mathbf{T}, 1.\mathbf{F}\}, \{(1, 1.\mathbf{T}), (x_2, 1.\mathbf{F})\}), \\ (M_2, I_2) &= (\{2.\mathbf{T}, 2.\mathbf{F}\}, \{(x_2, 2.\mathbf{F})\}), \\ (M_3, I_3) &= (\{3.\mathrm{A}, 3.\mathrm{B}, 3.\mathrm{C}\}, \{(x_1, 3.\mathrm{C})\}). \end{aligned}$$

Thus we have the context (G, M, I) such that

$$\begin{aligned} M &= M_1 \cup M_2 \cup M_3 = \{1.\mathbf{T}, 1.\mathbf{F}, 2.\mathbf{T}, 2.\mathbf{F}, 3.\mathrm{A}, 3.\mathrm{B}, 3.\mathrm{C}\}, \\ I &= I_1 \cup I_2 \cup I_3 = \{(x_1, 1.\mathbf{T}), (x_1, 3.\mathrm{C}), (x_2, 1.\mathbf{F}), (x_2, 2.\mathbf{F})\}. \end{aligned}$$

It is visualized as a cross-table as follows:

	1.**T**	1.**F**	2.**T**	2.**F**	3.A	3.B	3.C
x_1	×						×
x_2		×		×			

□

Second, we make a context from continuous variables using discretization. This process is embedded in the learning process (see Figure 6.1) and discretizing resolution increases along with the process. The degree of resolution is denoted by a

Algorithm 6.2: Data preprocessing for continuous variables

```
Input: Table $\tau = (H, X)$ whose variables are continuous,
       discretization level $k$
Output: Context $(G, M_{\mathrm{C}}, I_{\mathrm{C}})$

function ContextC($\tau, k$)
  $G \leftarrow \mathrm{set}(X)$
  for each $h \in H$
    $M_h \leftarrow \{ h.1, h.2, \dots, h.2^k \}$
    Normalize values in the feature $h$ by min-max normalization
    $I_h \leftarrow \emptyset$
    for each $x \in G$
      if $x(h) = 0$ then $I_h \leftarrow I_h \cup \{(x, 1)\}$
      else if $x(h) \neq 0$ and $x(h) \neq \bot$ then
        $I_h \leftarrow I_h \cup \{(x, h.a)\}$, where $(a-1)/2^k < x(h) \leq a/2^k$
      end if
    end for
  end for
  combine all $(G, M_h, I_h)$ with $h \in H$ into $(G, M_{\mathrm{C}}, I_{\mathrm{C}})$
  return $(G, M_{\mathrm{C}}, I_{\mathrm{C}})$
```

discretization level

natural number k, called *discretization level* and, in the following, we explain how to discretize continuous variables at fixed level k. First we use min-max normalization (Han and Kamber, 2006) so that every datum is in the closed interval $[0, 1]$. For every feature h, each value $x(h)$ is mapped to a value $y(h)$ such that

$$y(h) = \frac{x(h) - \min_{x \in \mathrm{set}(X)} x(h)}{\max_{x \in \mathrm{set}(X)} x(h) - \min_{x \in \mathrm{set}(X)} x(h)}.$$

Next, we discretize values in $[0, 1]$ and make a context using the binary encoding of real numbers, following the approach we have used in the previous chapters. At discretization level k, M_h for a feature $h \in H$ is always the set $\{h.1, h.2, \dots, h.2^k\}$. For each value $x_i(h)$, if $x_i(h) = 0$, then $(x_i, h.1) \in I_h$. Otherwise if $x_i(h) \neq 0$, then $(x_i, h.a) \in I_h$ if and only if

$$\frac{a-1}{2^k} < x_i(h) \leq \frac{a}{2^k}.$$

If $x_i(h) = \bot$, then $(x_i, m) \notin I_h$ for all $m \in M_h$. This means that if we encode the value $x_i(h)$ as an infinite sequence $p = p_0 p_1 p_2 \dots$, a context at level k is decided by the first k bits $p_0 p_1 \dots p_{k-1}$. Each value is converted to exactly one relation of a context if it is not missing. Algorithm 2 shows the above process for making a context from continuous variables.

Example 6.3
Given a table $\tau = (H, X)$ with $H = \{1, 2, 3, 4\}$ and $X = x_1, x_2$ such that

$$(x_1(1), x_1(2), x_1(3)) = (\mathbf{T}, \mathrm{C}, 0.35, 0.78),$$
$$(x_2(1), x_2(2), x_2(3)) = (\bot, \bot, 0.813, \bot).$$

It can be represented as follows:

H		1	2	3	4
X	x_1	**T**	C	0.35	0.78
	x_2	$\perp$	$\perp$	0.813	$\perp$

where the first and second features are discrete with $\text{Dom}(1) = \{\mathbf{T}, \mathbf{F}\}$ and $\text{Dom}(2) = \{\text{A}, \text{B}, \text{C}\}$, and the third and forth are continuous. Assume that discretization level $k = 1$. We have

$$\begin{aligned} G &= \{x_1, x_2\}, \\ (M_1, I_1) &= (\{1.\mathbf{T}, 1.\mathbf{F}\}, \{(x_1, 1.\mathbf{T})\}), \\ (M_2, I_2) &= (\{2.\text{A}, 2.\text{B}, 2.\text{C}\}, \{(x_1, 2.\text{C})\}), \\ (M_3, I_3) &= (\{3.1, 3.2\}, \{(x_1, 3.1), (x_2, 3.2)\}), \\ (M_4, I_4) &= (\{4.1, 4.2\}, \{(x_1, 4.2)\}). \end{aligned}$$

Thus we have the context (G, M, I) such that $M = M_1 \cup M_2 \cup M_3 \cup M_4$ and $I = I_1 \cup I_2 \cup I_3 \cup I_4$, which is visualized as a cross-table as follows:

	1.**T**	1.**F**	2.A	2.B	2.C	3.1	3.2	4.1	4.2
x_1	×				×	×			×
x_2							×		

□

6.2.2 Clustering and Making Lattices by FCA

From a context obtained by the data preprocessing, we generate closed sets as clusters of data points and construct closed set lattices by FCA. First we summarize FCA (see literatures (Davey and Priestley, 2002; Ganter and Wille, 1998) for detail). We always assume that a given table τ is converted into a context (G, M, I) by Algorithms 6.1 and 6.2.

Definition 6.4: Concept
For subsets $A \subseteq G$ and $B \subseteq M$, we define

$$\begin{aligned} A' &:= \{m \in M \mid (g, m) \in I \text{ for all } g \in A\}, \\ B' &:= \{g \in G \mid (g, m) \in I \text{ for all } m \in B\}. \end{aligned}$$

concept, extent, intent

Then, a pair (A, B) with $A \subseteq G$ and $B \subseteq M$ is called a *concept* of a context (G, M, I) if $A' = B$ and $A = B'$. The set A is called an *extent* and B an *intent*.

Galois connection

closed

Each operator $'$ is a *Galois connection* between the power set lattices on G and M, respectively, hence the mapping $''$ becomes a closure operator on the context (G, M, I). This means that, for each concept (A, B), A and B are (algebraic) *closed* sets. Note that a subset $A \subseteq G$ (resp. $B \subseteq M$) is the extent (resp. intent) of some concept if and only if $A'' = A$ (resp. $B'' = B$). Thus a set of objects $A \subseteq G$ forms a cluster if and only if $A'' = A$. Each object usually belongs to more than one cluster, hence this method is not “crisp” clustering.

concept lattice

The set of concepts over (G, M, I) is written by $\mathfrak{B}(G, M, I)$ and called the *concept lattice*. If we focus on either one of the set of objects or attributes, this lattice is called the *closed set lattice*. In particular, in the context of frequent pattern mining, a set of attributes corresponds to an itemset and the lattice is called the closed itemset lattice. For a pair of concepts $(A_1, B_1) \in \mathfrak{B}(G, M, I)$ and $(A_2, B_2) \in \mathfrak{B}(G, M, I)$, we write $(A_1, B_1) \leq (A_2, B_2)$ if $A_1 \subseteq A_2$. Then $(A_1, B_1) \leq (A_2, B_2)$ holds if and only if $A_1 \subseteq A_2$ (and if and only if $B_1 \supseteq B_2$). This relation $\leq$ becomes an order on $\mathfrak{B}(G, M, I)$ in the mathematical sense and $\langle \mathfrak{B}(G, M, I), \leq \rangle$ becomes a complete lattice.

closed set lattice

Let $\mathcal{C} \subseteq \mathfrak{B}(G, M, I)$. A concept $(A, B) \in \mathcal{C}$ is a *maximal element* of $\mathcal{C}$ if $(A, B) \leq (X, Y)$ and $(X, Y) \in \mathcal{C}$ imply $(A, B) = (X, Y)$ for all $(X, Y) \in \mathcal{C}$. We write the set of maximal elements of $\mathcal{C}$ by $\mathrm{Max}\mathcal{C}$.

Many algorithms are available for constructing closed set lattices, or concept lattices, and the algorithm proposed by Makino and Uno (2004) is known to be one of the fastest algorithms. Their algorithm enumerates all maximal bipartite cliques in a bipartite graph with $O(\Delta^3)$ delay, where Δ is the maximum degree of the given bipartite graph, that is,

$$\Delta = \max\left\{ \#J \;\middle|\; J \subseteq I, \text{ where } \begin{array}{l} g = h \text{ for all } (g, m), (h, l) \in J, \text{ or} \\ m = l \text{ for all } (g, m), (h, l) \in J \end{array} \right\}$$

($\#J$ is the number of elements in J) in the FCA context. Since we can easily check that each context and concept exactly coincide with a bipartite graph and a maximal bipartite clique, respectively (Figure 6.2), we can use their algorithm directly. For empirical experiments, we use the program LCM by Uno *et al.* (2005) provided by the authors to enumerate all concepts and construct the closed set lattice.

Example 6.5
Given the following context:

	1	2	3	4	5
x_1	×	×		×	
x_2		×		×	×
x_3			×		
x_4		×			×

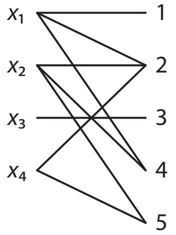


Figure 6.2 | The bipartite graph corresponding to the context in Example 6.5.

There exist eight concepts in total;

$$(\emptyset, \{1, 2, 3, 4, 5\}), (\{x_1\}, \{1, 2, 4\}), (\{x_2\}, \{2, 4, 5\}), (\{x_3\}, \{3\}),$$
$$(\{x_1, x_2\}, \{2, 4\}), (\{x_2, x_4\}, \{2, 5\}), (\{x_1, x_2, x_4\}, \{2\}), \text{ and } (\{x_1, x_2, x_3, x_4\}, \emptyset),$$

and $\Delta = 3$. We show the closed set lattice in Figure 6.3. Let

$$\mathcal{C} = \left\{ \begin{array}{l} (\emptyset, \{1, 2, 3, 4, 5\}), (\{x_1\}, \{1, 2, 4\}), (\{x_2\}, \{2, 4, 5\}), \\ (\{x_1, x_2\}, \{2, 4\}), (\{x_2, x_4\}, \{2, 5\})\} \end{array} \right\}.$$

Then $\max \mathcal{C} = \{(\{x_1\}, \{1, 2, 4\}), (\{x_1, x_2\}, \{2, 4\}), (\{x_2, x_4\}, \{2, 5\})\}$. □

6.2.3 Learning Classification Rules

Here we present the main learning algorithm of SELF in Algorithm 3, which obtains a set of classification rules from a table τ for training. In this chapter, a *classification rule* is a pair of a set of attributes and a label since, intuitively, every unlabeled

classification rule

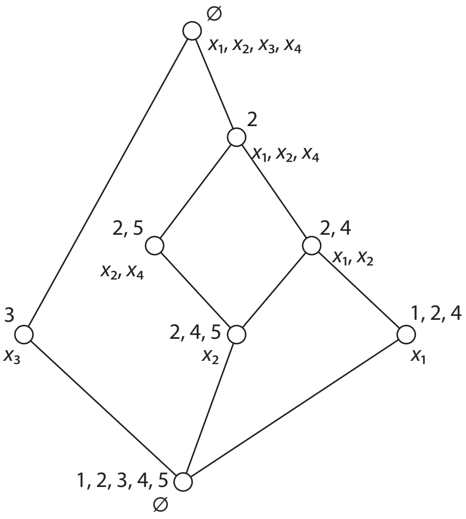


Figure 6.3 | The closed set lattice (concept lattice) constructed from the context given in Example 6.5. In this diagram, each dot denotes a concept, which are treated as a cluster in SELF.

tuple (datum) is classified to the associated label if it has the same attributes. SELF generates a set of classification rules at each discretization level. We give the precise algorithm of classification in the next subsection.

label

We introduce some notations. For each object $g \in G$, we denote a *label*, an identifier of a class, of g by $\Lambda(g)$, and if g is unlabeled; *i.e.*, the label information is missing, we write $\Lambda(g) = \bot$. Moreover, we define

$$\Gamma(G) := \{\, g \in G \mid \Lambda(g) \neq \bot \,\}$$

hence objects in $\Gamma(G)$ are labeled and those in $G \setminus \Gamma(G)$ are unlabeled.

l-consistent

Definition 6.6: *l*-consistency
For a concept $(A, B) \in \mathfrak{B}(G, M, I)$, we say that it is *l-consistent* if $\Gamma(A) \neq \emptyset$ and $\Lambda(g) = \Lambda(h)$ for all $g, h \in \Gamma(A)$.

Note that a concept with $\Gamma(A) = \emptyset$ (all labels are missing) is not l-consistent.

First SELF performs data preprocessing and makes the context (G, M, I) from a given table at each discretization level k using the algorithms given in Section 6.2.1. Second it constructs the concept lattice $\mathfrak{B}(G, M, I)$ using both labeled and unlabeled tuples and finds l-consistent concepts using labeled tuples (objects). Third it outputs the sets of classification rules such that

$$\begin{aligned} \mathcal{R}_k &= \{\, (B, \lambda) \mid (A, B) \in \mathrm{Max}\mathcal{C}_k \text{ and } \lambda = \Lambda(g) \text{ with } g \in \Gamma(A) \,\}, \quad \text{where} \\ \mathcal{C}_k &= \{\, (A, B) \in \mathfrak{B}(G, M, I) \mid (A, B) \text{ is } l\text{-consistent} \,\} \end{aligned}$$

at discretization level k. The lattice enables us to avoid overfitting since, informally, attributes of maximal concepts correspond to the most general classification rules. If some objects that are not contained in l-consistent concepts remains, it refines discretization; *i.e.*, increases discretization level, and repeats the above procedure for the remaining objects.

Moreover, SELF *weights* each classification rule. For a classification rule $R = (B, \lambda)$, the weight $\omega(R)$ is defined as follows:

$$\omega(R) := \#\{\, (C, D) \in \mathfrak{B}(G, M, I) \mid D \supseteq B \,\}.$$

Algorithm 6.3: SELF algorithm

```
Input: Table $\tau = (H, X)$
Output: Classification rules $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_K$

function MAIN($\tau$)
  Divide $\tau$ vertically into two tables $\tau_\mathrm{D}$ and $\tau_\mathrm{C}$, where $\tau_\mathrm{D}$ contains all
    discrete variables in $\tau$ and $\tau_\mathrm{C}$ contains all continuous variables in $\tau$
  $(G, M_\mathrm{D}, I_\mathrm{D}) \leftarrow$ CONTEXTD($\tau_\mathrm{D}$)
    // make a context from discrete variables of $\tau$
  $k \leftarrow 1$    // $k$ is discretization level
  LEARNING($\tau_\mathrm{C}, G, M_\mathrm{D}, I_\mathrm{D}, k$)    // use this function recursively

function LEARNING($\tau_\mathrm{C}, G, M_\mathrm{D}, I_\mathrm{D}, k$)
  $(G, M_\mathrm{C}, I_\mathrm{C}) \leftarrow$ CONTEXTC($\tau_\mathrm{C}, k$)
    // make a context from continuous variables of $\tau$ at level $k$
  make $(G, M, I)$ from $(G, M_\mathrm{D}, I_\mathrm{D})$ and $(G, M_\mathrm{C}, I_\mathrm{C})$
  construct the concept lattice $\mathfrak{B}(G, M, I)$ from $(G, M, I)$
  $\mathcal{C} \leftarrow \{\, (A, B) \in \mathfrak{B}(G, M, I) \mid (A, B)$ is $l$-consistent $\}$
  $\mathcal{R}_k \leftarrow \{\, (B, \Lambda(g)) \mid (A, B) \in \mathrm{Max}\mathcal{C}$ and $g \in \Gamma(A) \,\}$
  output $\mathcal{R}_k$
  $G \leftarrow G \setminus \{\, g \mid g \in A$ for some $(A, B) \in \mathcal{C} \,\}$
  remove corresponding attributes and relations from $M_\mathrm{D}$ and $I_\mathrm{D}$
  remove corresponding tuples from $\tau_\mathrm{C}$
  if $\Gamma(G) = \emptyset$ then halt
  else LEARNING($\tau_\mathrm{C}, G, M_\mathrm{D}, I_\mathrm{D}, k + 1$)
  end if
```

Intuitively, the weight of a rule R means its importance since it is the number of clusters classified by the rule. Using the weight of rules, label ranking is achieved (see the next subsection).

Example 6.7
Given a dataset $\tau = (H, X)$ and its labels as follows:

H		1	2	3	Label
X	x_1	**T**	C	0.28	1
	x_2	**F**	A	0.54	1
	x_3	**T**	B	$\bot$	$\bot$
	x_4	**F**	A	0.79	2
	x_5	**T**	C	0.81	$\bot$

where $\mathrm{Dom}(1) = \{\mathbf{T}, \mathbf{F}\} \cup \{\bot\}$, $\mathrm{Dom}(2) = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\} \cup \{\bot\}$, and $\mathrm{Dom}(3) = \mathbb{R} \cup \{\bot\}$. At discretization level 1, we have the following context:

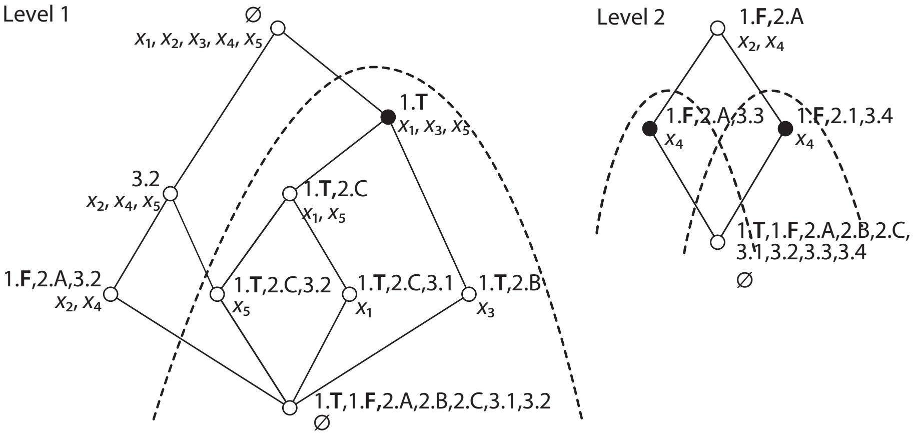


Figure 6.4 | The closed set lattices (concept lattices) at discretization levels 1 and 2 constructed during the learning phase in Example 6.7. In these diagrams, each black dot denotes the maximal *l*-consistent concept in the set of concepts covered by the dotted line.

	1.**T**	1.**F**	2.A	2.B	2.C	3.1	3.2
x_1	×				×	×	
x_2		×	×				×
x_3	×			×			
x_4		×	×				×
x_5	×				×		×

We show the closed set lattice in the left-hand side in Figure 6.4. By SELF, we obtain $\mathcal{R}_1 = \{(\{1.\mathbf{T}\}, 1)\}$ since the concept $(\{x_1, x_3, x_5\}, \{1.\mathbf{T}\})$ is the maximal l-consistent concept, and there is no l-consistent concept that contains x_2 or x_4. This classification rule means “For a tuple x, if $x(1) = \mathbf{T}$, then x is classified to the class 1”. The weight is calculated as $\omega(\{1.\mathbf{T}\}, 1) = 6$. SELF removes objects x_1, x_3, and x_5 contained in the l-consistent concepts and proceeds to the next level. At discretization level 2, we have the following context:

	1.**T**	1.**F**	2.A	2.B	2.C	3.1	3.2	3.3	3.4
x_2		×	×					×	
x_4		×	×						×

The right-hand side in Figure 6.4 shows the closed set lattice of the above context, and we obtain $\mathcal{R}_2 = \{(\{1.\mathbf{F}, 2.\mathrm{A}, 3.3\}, 1), (\{1.\mathbf{F}, 2.\mathrm{A}, 3.4\}, 2)\}$. For instance, the first rule means “For a tuple x, if $x(1) = \mathbf{F}$, $x(2) = \mathrm{A}$, and $0.5 < x(3) \leq 0.75$, its class label is 1”. The weight are 2 for both rules. □

We show that SELF always stops in finite time if there are no conflicting objects. Namely, for a table $\tau = (H, X)$, if there is no pair $x, y \in \mathrm{set}(X)$ such that $\Lambda(x) \neq \Lambda(y)$ and $x(h) = y(h)$ for all $h \in H$, Algorithm 3 stops in finite time. This statement is proved in the following way: if discretization level k is large enough, we have the concept lattice $\mathfrak{B}(G, M, I)$, where for every object $x \in G$, there exists a concept (A, B) such that $A = \{x\}$ since there is no pair $x, y \in G$ satisfying $x(h) = y(h)$ for all $h \in H$. Thus each object x with $\Lambda(x) \neq \bot$ must be contained in some l-consistent concept, and the algorithm stops. Note that the algorithm works even if $\Gamma(G) = G$; *i.e.*, all objects have labels, hence it also can be viewed as a supervised classification method.

Algorithm 6.4: Classification

```
Input: Classification rules $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_K$, table $v = (H, y)$, and
        the set of labels $\mathcal{L}$
Output: Preference of each label

function CLASSIFY($\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_K, v$)
  Divide $v$ vertically into two tables $v_\mathrm{D}$ and $v_\mathrm{C}$, where $v_\mathrm{D}$ contains all
    discrete variables in $v$ and $v_\mathrm{C}$ contains all continuous variables in $v$
  $(G, M_\mathrm{D}, I_\mathrm{D}) \leftarrow$ CONTEXTD($v_\mathrm{D}$)
    // make a context from discrete values of $v$
  for each $\lambda \in \mathcal{L}$
    lPref($\lambda$) $\leftarrow 0$
    for each $k \in \{1, 2, \dots, K\}$
      $(G, M_\mathrm{C}, I_\mathrm{C}) \leftarrow$ CONTEXTC($v_\mathrm{C}, k$)
      // make a context from continuous values of $v$ at level $k$
      make a context $(G, M, I)$ from $(G, M_\mathrm{D}, I_\mathrm{D})$ and $(G, M_\mathrm{C}, I_\mathrm{C})$
      lPref($\lambda$) $\leftarrow$ lPref($\lambda$) + $\sum_{R \in Q} \omega(R)$, where
        $Q = \{ (B, \lambda) \in \mathcal{R}_k \mid (y, b) \in I$ for all $b \in B \}$
    end for
    output lPref($\lambda$)
  end for
```

The time complexity of learning by SELF is $O(nd) + O(\Delta^3 N)$ such that

$$N = \max_{k \in \{1,2,\dots,K\}} \#\mathfrak{B}(G_k, M_k, I_k),$$

where (G_k, M_k, I_k) is the context at discretization level k and K is the level where SELF stops since data preprocessing takes $O(nd)$, making a concept lattice takes less than $O(\Delta^3 N)$, and obtaining classification rules takes less than $O(N)$.

6.2.4 Classification

Now we have sets of classification rules $\mathcal{R}_1$, $\mathcal{R}_2$, …, $\mathcal{R}_K$ for each discretization level from training mixed-type data including labeled and unlabeled data using Algorithms 6.1, 6.2, and 6.3. In this section, we show how to classify a new unlabeled datum using the rules. We assume that such a new datum is given as a table $v = (H, y)$, where the body l-consists of only one tuple y.

Algorithm 6.4 performs classification using the obtained rules $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_K$. The algorithm is levelwise; *i.e.*, at each level k, it makes a context (G, M, I) from the table $v = (H, y)$ and apply the set of rules $\mathcal{R}_k$ to it. Let $\mathcal{L}$ be the domain of class labels. SELF checks all rules in $\mathcal{R}_k$ at each discretization level k and, for each label $\lambda \in \mathcal{L}$, it outputs the preference of the label λ.

l-preference

Definition 6.8: *l*-preference
For a label $\lambda \in \mathcal{L}$, its *l-preference* is defined as

$$\mathrm{lPref}(\lambda) \coloneqq \sum_{k=1}^{K} \sum_{R \in Q} \omega(R),$$

where

$$Q = \{\, (B, \lambda) \in \mathcal{R}_k \mid (y, b) \in I \text{ for all } b \in B \,\}$$

by summing up weights of rules.

Note that the set G is always a singleton $\{y\}$ in the classification phase. The result means that if $\mathrm{lPref}(\lambda) > \mathrm{lPref}(\lambda')$ for labels λ and λ', λ is preferable than λ', and vice versa, and if $\mathrm{lPref}(\lambda) = \mathrm{lPref}(\lambda)$, the preference of λ and λ' are same, resulting in the *partial order* over the set of labels $\mathcal{L}$. Thus the task of label ranking is performed by the preference lPref. Moreover, if we pick up the label

$$\lambda \in \mathrm{argmax}_{\lambda \in \mathcal{L}} \mathrm{lPref}(\lambda), \tag{6.1}$$

multiclass classification is also achieved directly.

Example 6.9
Let us consider the case discussed in Example 6.7. A tuple y such that

$$(y(1), y(2), y(3)) = (\mathbf{T}, \mathrm{B}, 0.45)$$

satisfies only the rule $(\{1.\mathbf{T}\}, 1) \in \mathcal{R}_1$. Thus we have $\mathrm{lPref}(1) = 6$ and $\mathrm{lPref}(2) = 0$ for labels 1 and 2, respectively. A tuple z with

$$(z(1), z(2), z(3)) = (\mathbf{F}, \mathrm{A}, 0.64)$$

satisfies only the rule $(\{1.\mathbf{F}, 2.\mathrm{A}, 3.3\})$, hence $\mathrm{lPref}(1) = 0$ and $\mathrm{lPref}(2) = 2$. □

6.3 Experiments

Here we empirically evaluate SELF. Our experiments consist of two parts: one is about multiclass classification, and the other is about label ranking.

6.3.1 Methods

Environment

SELF was implemented in R version 2.12.1 (R Development Core Team, 2011) and all experiments were performed in the R environment. For enumeration of all concepts and construction of a closed set lattice from a context, we used LCM[1] distributed by Uno *et al.* (2005), which was implemented in C.

[1] http://research.nii.ac.jp/~uno/codes.htm

Name	# Data	# Classes	# Features	
			Discrete	Continuous
abalone	4177	28	1	7
allbp	2800	3	2	3
anneal	798	5	28	10
arrhythmia	452	13	5	5
australian	690	2	7	4
crx	690	2	9	6
echocardiogram	131	2	1	7
heart	270	2	7	6
hepatitis	155	2	13	6
horse colic	368	2	8	2

Table 6.1 | Statistics for datasets collected from UCI repository used for experiments.

Datasets

We collected ten mixed-type datasets from UCI repository (Frank and Asuncion, 2010): *abalone*, *allbp*, *anneal*, *arrhythmia*, *australian*, *crx*, *echocardiogram*, *heart*, *hepatitis*, and *horse colic*. Their basic statistics are summarized in Table 6.1. Datasets *allbp*, *anneal*, *arrhythmia*, *australian*, *crx*, *echocardiogram*, *hepatitis*, and *horse colic* included missing values, which were directly treated in SELF. In other learning algorithms, we ignored all tuples which have missing values since they cannot treat such datasets appropriately.

In label ranking, we used four datasets: *abalone*, *allbp*, *anneal*, and *arrhythmia*, which have more than three classes. The other datasets had only two classes and could not be used for label ranking evaluation.

Control Algorithms

In multiclass classification, three learning algorithms were adopted: the decision tree-based classifier implemented in R supplied in the `tree` package (Ripley, 1996), SVM with the RBF kernel ($C = 5$ and $\gamma = 0.05$) in the `kernlab` package (Karatzoglou *et al.*, 2004), and the k nearest neighbor algorithm ($k = 1$ and 5) in the `class` package. Notice that only the decision tree-based algorithm can treat mixed-type data directly, which is one of typical such learning algorithms. All discrete values were treated as continuous in SVM and kNN.

Evaluation

In classification, for each dataset, the following procedure was repeated 20 times and the mean and s.e.m. (standard error of the mean) of accuracy was obtained: 1) the number of labeled data or features was fixed, where the range was from 10 to 100 and 2 to 10, respectively, 2) labeled training data were sampled randomly, 3) labels of the remaining data were predicted by respective learning algorithms, and 4) the accuracy was obtained. The equation (6.1) was used to determine the most preferable label for each unlabeled datum. If there exists more than two such labels, we chose the smallest one.

We adopted two criteria: *correctness* and *completeness*, which was used in the literature (Cheng *et al.*, 2010), to evaluate partial orders of labels in label ranking. Correctness coincides with the *gamma rank correlation* (Goodman and Kruskal,

correctness
completeness

1979), which is the normalized difference between the number of correctly ranked pairs and that of incorrectly ranked pairs. Let $\mathcal{L}$ be the set of class labels and we denote by $\prec_*$ the ground truth of the partial order over the set of labels $\mathcal{L}$. Assume that $\prec$ is a predicated partial order. Here we define

$$C := \#\{\,(\lambda, \lambda') \in \mathcal{L} \times \mathcal{L} \mid \lambda \prec \lambda' \text{ and } \lambda \prec_* \lambda'\,\},$$
$$D := \#\{\,(\lambda, \lambda') \in \mathcal{L} \times \mathcal{L} \mid \lambda \prec \lambda' \text{ and } \lambda' \prec_* \lambda\,\}.$$

Then, the correctness is defined by

$$\mathrm{CR}(\prec, \prec_*) := \frac{C - D}{C + D}.$$

Trivially, the correctness takes a value in $[-1, 1]$, and $\mathrm{CR}(\prec, \prec_*) = 1$ if $\prec = \prec_*$ and $\mathrm{CR}(\prec, \prec_*) = -1$ if $\prec$ is the inversion of$\prec_*$. Thus the correctness should be maximized. Moreover, to evaluate the degree of completeness of a predicted partial order, we use the completeness defined as follows:

$$\mathrm{CP}(\prec) := \frac{C + D}{\#\{(\lambda, \lambda') \in \mathcal{L} \times \mathcal{L} \mid \lambda \prec_* \lambda' \text{ or } \lambda' \prec_* \lambda\}}.$$

The completeness takes a value in $[0, 1]$ and should be maximized.

6.3.2 Results

Multiclass Classification

We evaluated SELF in multiclass classification. Specifically, we examined SELF's behavior with respect to the number of labeled data and the number of features; the number of labeled data was varied from 10 to 100, and the number of features from 2 to 10. When we fixed the number of labeled data, we used all features for *abalone*, *anneal*, *australian*, *crx*, *echocardiogram*, *heart*, and *hepatitis*, and only used features 1, 2, 3, 18, 20 in *allbp*, 1, 2, …, 6, 22, 22, …, 25 in *arrhythmia*, and 1, 2, 4, 5, … 11 in *horse colic*, since we could not finish experiments in reasonable time for such dense datasets. The above features seem to be representative for each dataset. Otherwise if we fixed the number of features, we examined two cases in which the number of labeled data for training were 10 or 100. Such small amount of labeled data is typical in evaluation in semi-supervised learning; for example, the numbers 10 and 100 were adopted in benchmarks in the literature[1] (Chapelle *et al.*, 2006, §21).

To analyze effectivity of unlabeled data in the semi-supervised manner, we trained SELF in two ways; one is using both labeled and the remaining all unlabeled data for training, and the other is using only labeled data for training without any unlabeled data. In the following, we denote "SELF" in the former case and "SELF (w/o)" in the latter case. All experiments were carried out in the *transductive setting* (Vapnik and Sterin, 1977), that is, test data coincide with the unlabeled training data. This setting is common in empirical evaluation of semi-supervised learning methods (Chapelle *et al.*, 2006, §21).

[1] This content is available at `http://olivier.chapelle.cc/ssl-book/benchmarks.pdf`.

For control, three learning algorithms were adopted: the decision tree-based classifier, SVM with the RBF kernel, and the k nearest neighbor algorithm ($k = 1$ and 5). All the above algorithms are typical for supervised learning and hence did not use unlabeled data in training.

Figure 6.5 and Figures 6.6, 6.7 show the accuracy with respect to changes in the number of labeled data and the number of features, respectively. In every case, the accuracy of SELF was much better than that of SELF (w/o), and the accuracy was getting better according as the number of labeled data increases. Moreover, SELF's performance is getting better with increase in the number of features. SELF therefore can effectively use unlabeled data and features for learning.

In comparison with the tree algorithm which can treat mixed-type data directly, SELF showed better performance in all datasets in Figure 6.5. Moreover, compared to other learning algorithms of SVM and kNN, SELF also achieved the best performance in *abalone*, *anneal*, and *horse colic*. When the number of labeled data is small (about 10 – 40), SELF outperformed other learning algorithms in all datasets except *allbp*, as shown in Figures 6.5 and 6.6.

Label Ranking

We examined effectivity of SELF for label ranking. In consideration of the lack of benchmark data for label ranking, we adopted the following procedure for label ranking: we trained SELF using all labeled data on the respective dataset and obtained the ranking for each datum, and used them as the ground truth. Cheng *et al.* (2010); Hülermeier *et al.* (2008) who studied label ranking used the naïve Bayes classifier to make the ground truth of rankings from datasets. However, the mathematical theory is totally different from those of SELF, hence their approach is not appropriate to our case.

Figure 6.8 shows the results of label ranking by SELF with varying the number of labeled data, and Figures 6.9 and 6.10 show those with respect to the number of features, where the number of labeled data is 10 for Figure 6.9 and 100 for Figure 6.10. The correctness of SELF is better than SELF (w/o) in *abalone*, and is similar between them in the other datasets for all conditions. In contrast, the completeness of SELF is much higher than that of SELF (w/o) in most cases. The main reason might be that lots of data are not classified to any class in SELF (w/o).

6.3.3 Discussion

Our experiments about classification (Figures 6.5, 6.6, 6.7) show that SELF has competitive performance compared to other machine learning algorithms, where unlabeled data can be used effectively in training. This result means that data clustering using the closed set lattices works well for semi-supervised learning of mixed-type data. Moreover, SELF can explicitly produce classification rules like the decision tree-based algorithm, hence SELF's results can be easily interpreted. Furthermore, in label ranking (Figures 6.8 – 6.10), SELF outperformed SELF (w/o) in most cases in terms of completeness, and the performance got higher with increase of the number of labeled data. Our results therefore show that unlabeled data are also effectively used in SELF in the task of label ranking.

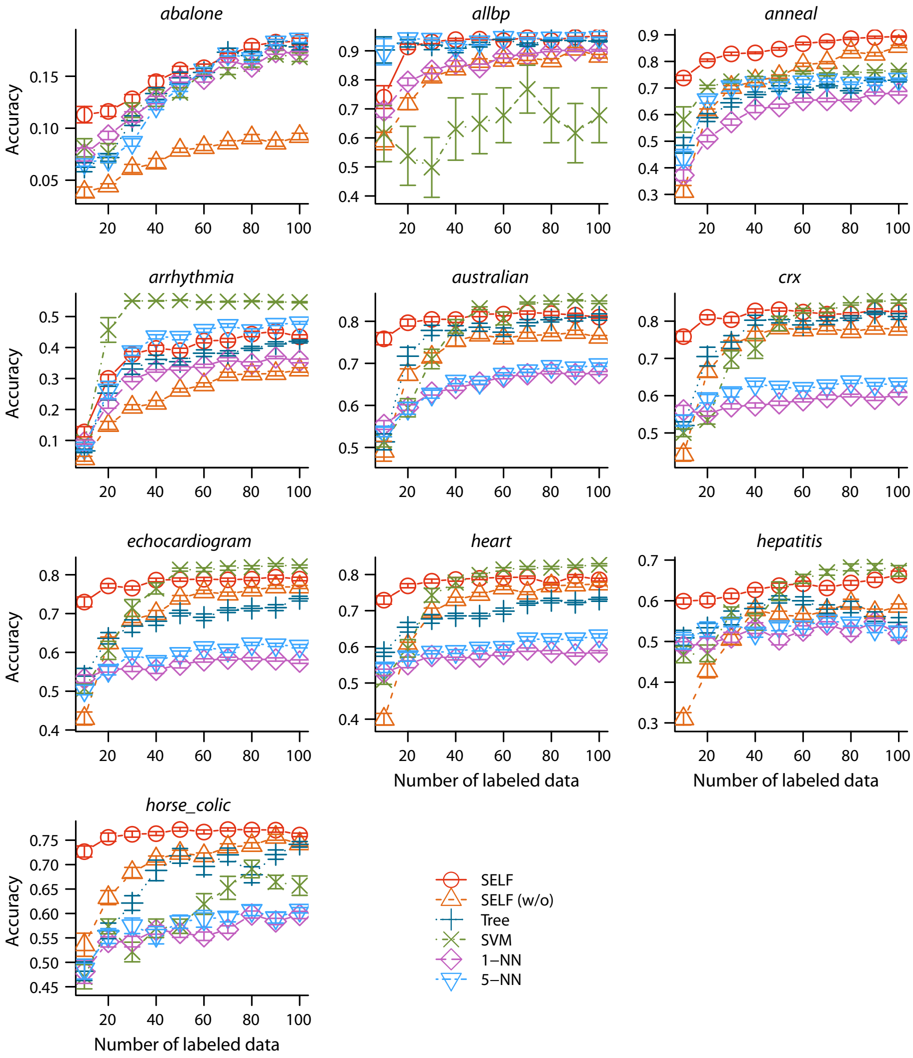


Figure 6.5 | Experimental results of accuracy for ten mixed-type datasets from UCI repository with varying the number of labeled data. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), and compared them to the decision tree-based classifier (Tree), SVM with the RBF kernel (SVM), and the *k*-nearest neighbor algorithm (1-NN, 5-NN). Data show mean ± s.e.m.

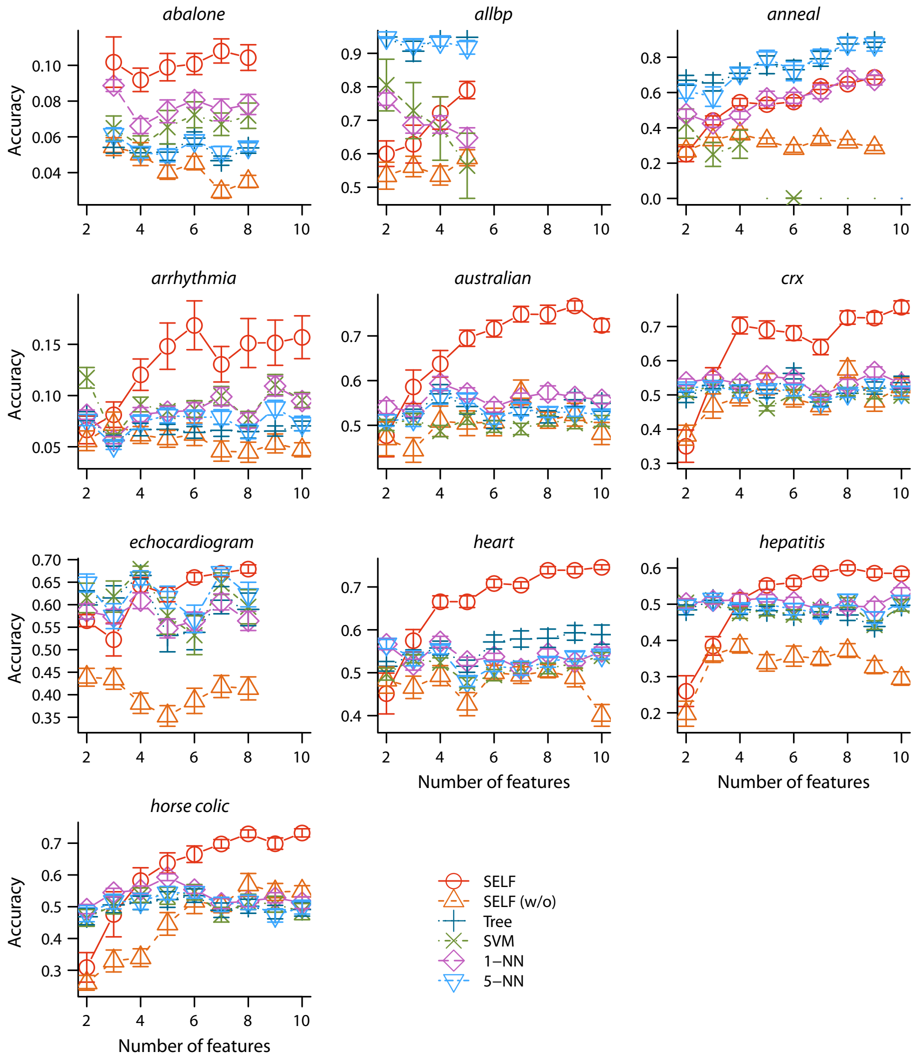


Figure 6.6 | Experimental results of accuracy for ten mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 10 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), and compared them to the decision tree-based classifier (Tree), SVM with the RBF kernel (SVM), and the *k*-nearest neighbor algorithm (1-NN, 5-NN). Data show mean ± s.e.m.

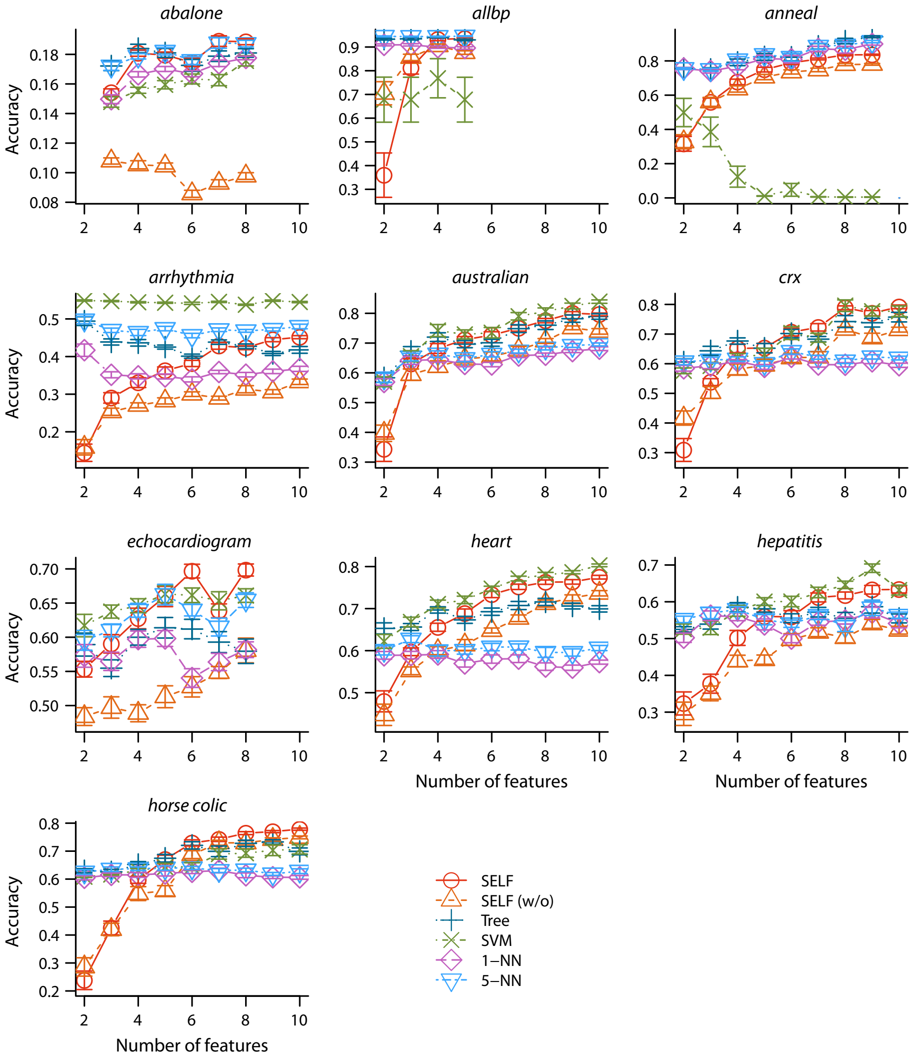


Figure 6.7 | Experimental results of accuracy for ten mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 100 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), and compared them to the decision tree-based classifier (Tree), SVM with the RBF kernel (SVM), and the *k*-nearest neighbor algorithm (1-NN, 5-NN). Data show mean ± s.e.m.

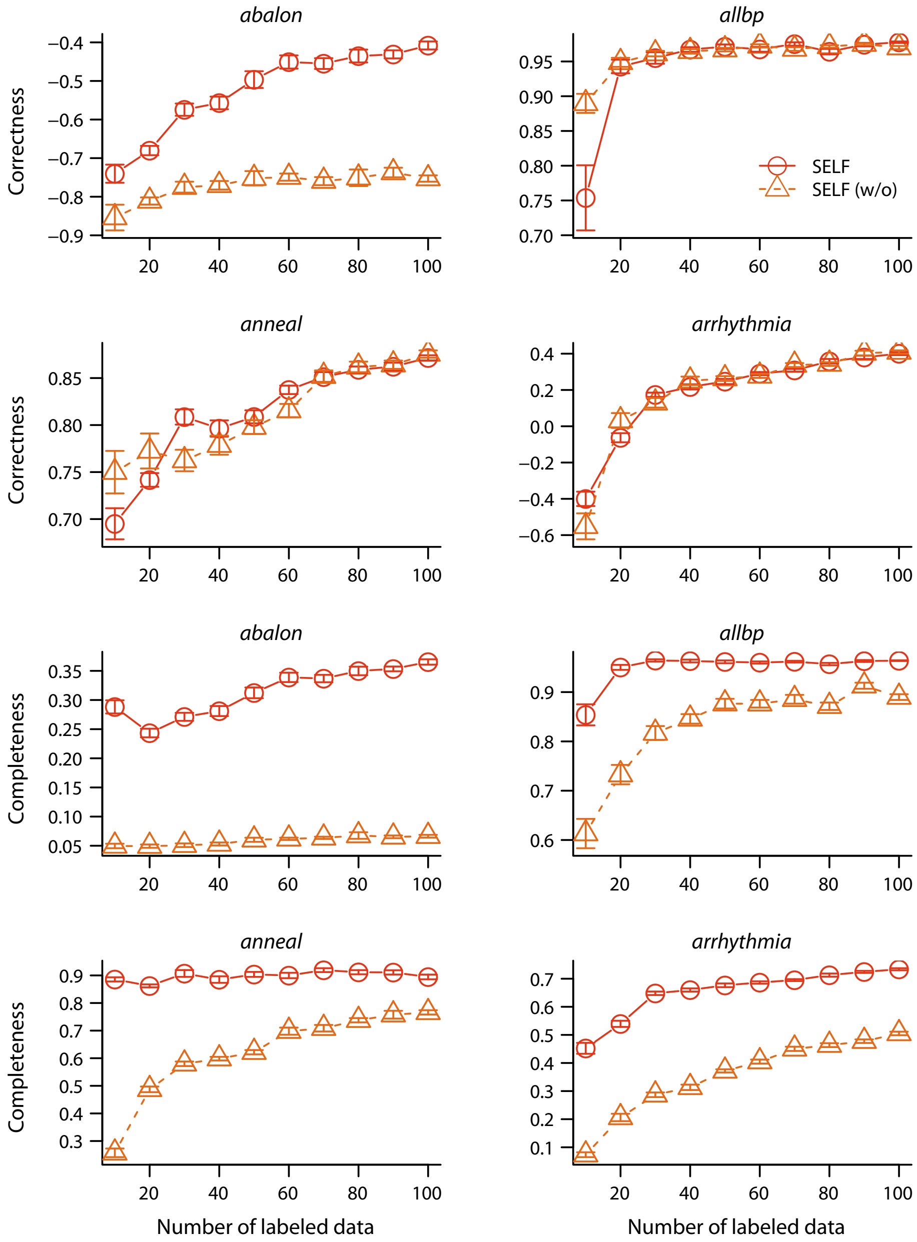


Figure 6.8 | Experimental results of correctness and completeness (should be maximized) for four mixed-type datasets from UCI repository with varying the number of labeled data. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), Data show mean ± s.e.m.

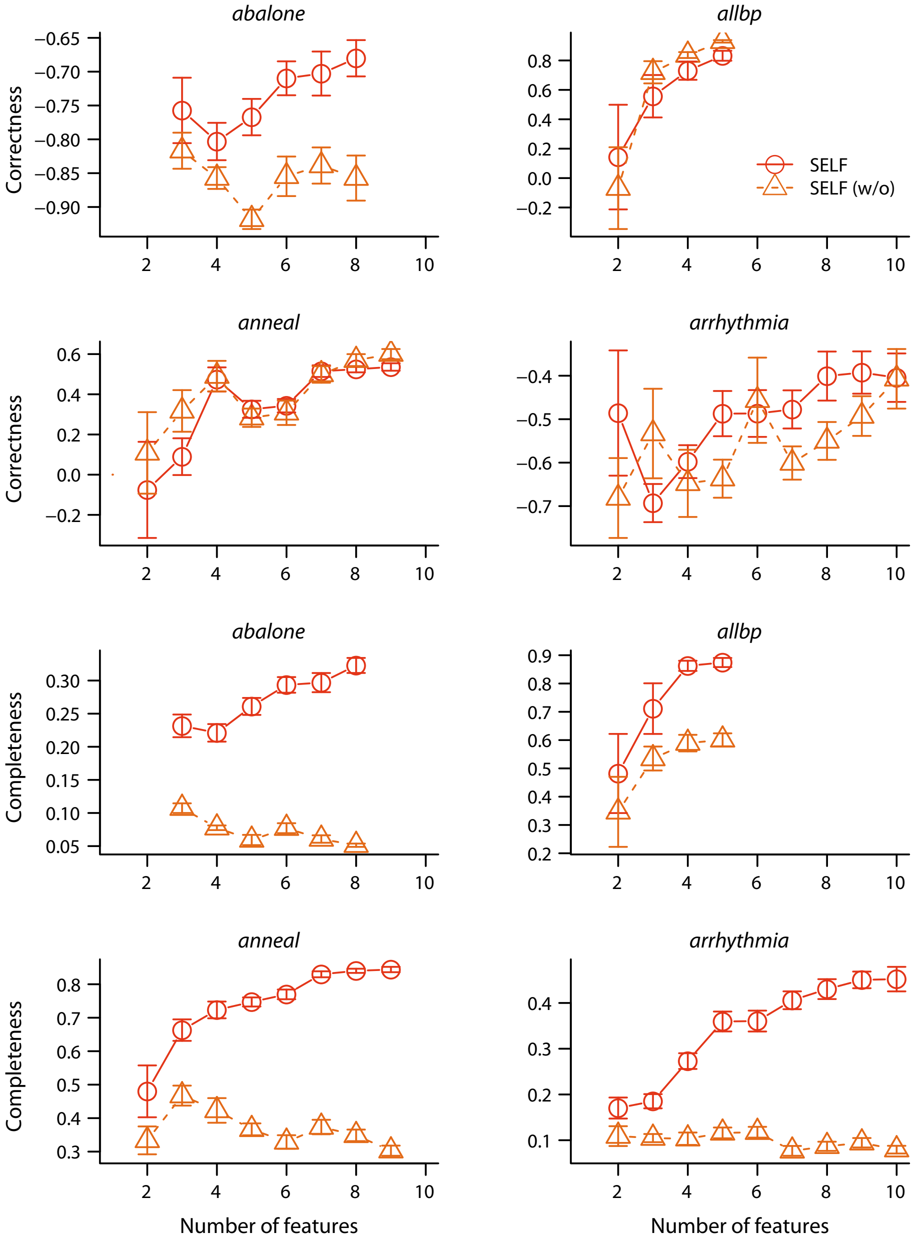


Figure 6.9 | Experimental results of correctness and completeness (should be maximized) for mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 10 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), Data show mean ± s.e.m.

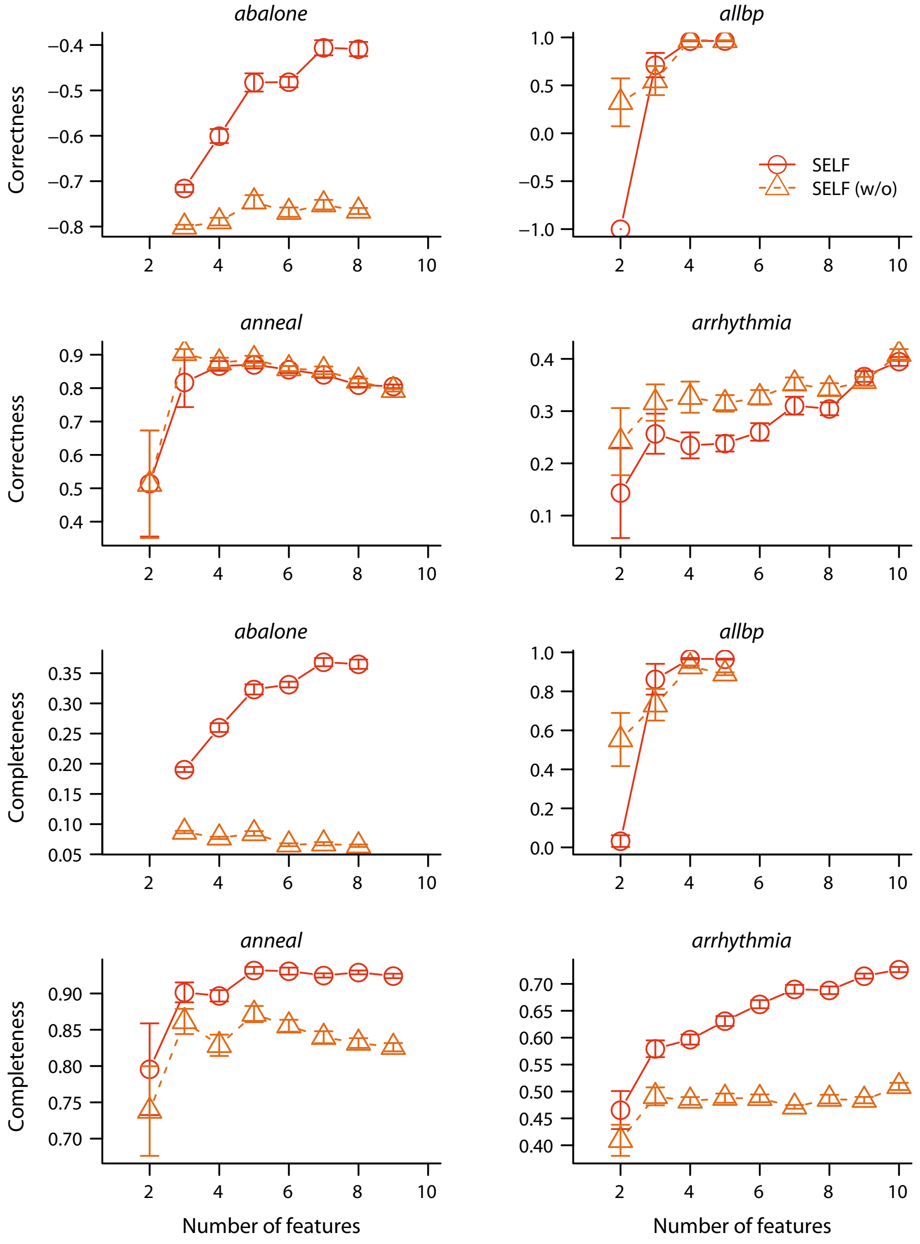


Figure 6.10 | Experimental results of correctness and completeness (should be maximized) for mixed-type datasets from UCI repository with varying the number of features. The number of labeled data was fixed at 100 in each experiment. We performed SELF using both labeled and unlabeled data (SELF) and using only labeled data (SELF (w/o)), Data show mean ± s.e.m.

6.4 Summary

We have proposed a novel semi-supervised learning algorithm, called SELF, for mixed-type data including both discrete and continuous variables, and experimentally showed its competitive performance. The key strategy is data clustering with closed set lattices using FCA, and the present study shows the effectivity of the lattices in semi-supervised learning. To our best knowledge, this approach is the first direct semi-supervised method for mixed-type data, and also the first one to exploit closed set lattices in semi-supervised learning. Moreover, we can directly treat missing values on SELF, meaning that SELF can be used for various practical datasets. To date, many semi-supervised learning methods use data distribution and probabilities, whereas SELF uses only the algebraic structure of data without any background knowledge. Our results with lattice-based data analysis provide new insight to machine learning and knowledge discovery.

There are two future works; one is analysis of SELF from the FCA point of view. Refinement of discretization of continuous variables must have some connection with *reduction* of a context (Ganter and Wille, 1998) since if we extend a context by refining real-valued variables, the original attributes are removed by reduction. Moreover, ambiguity of data, such as intervals including truth values, might be treated using such techniques in FCA. Thereby analysis of mathematical connection between them is a future work. The other is theoretical analysis in the computational learning theory context. de Brecht and Yamamoto (2010) have proposed *Alexandrov concept space* for learning from positive data. Our proposed method might be an instance of the study, since the concept lattice is similar to the Alexandrov space. Thus theoretical analysis of our framework is also a future work.

7

LIGAND FINDING BY MULTI-LABEL CLASSIFICATION

Receptors are protein molecules located at the surface of cells, which receive chemical signals from outside of the cells. Since receptors have crucial roles for signal processing in organisms, to date, enormous studies have been devoted to investigate their biochemical functions. The key approach in an experiment is to use receptor specificity with respect to a *ligand*, which triggers a cellular response by binding to a receptor, for controlling the receptor actions (Figure 7.1). However, finding new convenient ligands is difficult; choosing ligand candidates relies on expert knowledge of biologists and conducting experiments to test whether or not candidates work *in vivo* or *in vitro* costs high in terms of time and money. Thus an *in silico* approach is required for helping biologists.

receptor

ligand

In this chapter, we adopt a machine learning, or knowledge discovery and data mining, approach to find candidates of ligands. Specifically, we formulate the problem of ligand finding as *multi-label classification* recently discussed in the field of *preference learning* (Fürnkranz and Hüllermeier, 2010), where each training datum used for learning is associated with not a single class label but a set of possible labels. Here, for each ligand, receptors to which it binds correspond to class labels of the ligand, and our goal is to predict labels (*i.e.*, receptors) of ligands from databases of receptors and ligands. A ligand can often bind to more than two receptors; this is why our problem is not traditional single-label but multi-label classification. Moreover, we try to predict labels in a *semi-supervised* manner (Chapelle *et al.*, 2006; Zhu and Goldberg, 2009). Semi-supervised learning is a special form of classification, where a learning algorithm uses both labeled and unlabeled data in training. Commonly, only few labeled data are assumed to be available since the labeling task costs high in a real situation. Semi-supervised learning therefore fits to our goal since, in our problem, only few ligands for each receptor have been discovered yet lots of ligands for other receptors are available.

multi-label classification

preference learning

semi-supervised learning

Formally, the problem of semi-supervised multi-label classification is stated as follows: Given a sequence $X = x_1, x_2, \dots, x_n$, where each x_i is a *tuple*, or *feature vector*, and a domain of labels $\mathcal{L}$. Each tuple x_i is associated with a set of labels $L_i \subseteq \mathcal{L}$. Since we consider semi-supervised learning, $L_i = \emptyset$ is allowed. The goal is, for any tuple (test datum) y, to predict the *preference* of labels with respect to

tuple

feature vector

preference

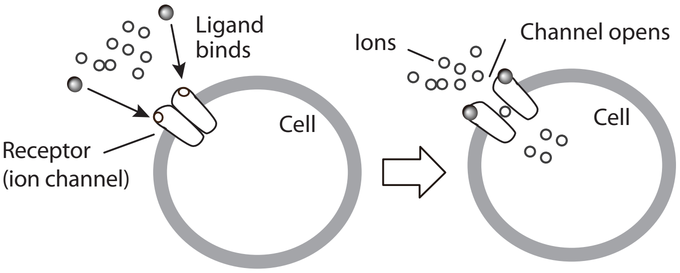


Figure 7.1 | A ligand-gated ion channel, which is a typical receptor.

y, and we can decide whether or not y is associated with a label λ for each $\lambda \in \mathcal{L}$.

Information about receptors and ligands is donated to various databases, such as KEGG[1], and in this chapter we use the IUPHAR database (Sharman *et al.*, 2011)[2]. In the database, every ligand is characterized by seven features as follows: hydrogen bond acceptors, hydrogen bond donors, rotatable bonds, topological polar surface area, molecular weight, XLogP, and number of Lipinski's rules broken. We abbreviate them in this chapter HBA, HBD, RB, TPS, MW, XLogP, and NLR, respectively. Here, TPS, MW, and XLogP take *continuous* (real-valued) values while the others, HBA, HBD, RB, and NLR, take *discrete* values. Thus to design an effective classifier for the IUPHAR database, we have to appropriately treat *mixed-type data* including both discrete and continuous variables.

mixed-type data

Recently, semi-supervised learning is one of active research fields in machine learning and knowledge discovery, and now various semi-supervised learning algorithms have already been developed (Chapelle *et al.*, 2006; Zhu and Goldberg, 2009). However, most of them are designed for real-valued variables (not discrete variables) and cannot be applied to mixed-type data directly and, moreover, they do not treat multi-label classification considered in this chapter. Therefore, in this chapter, we construct a new learning algorithm, called LIFT (L̲igand FI̲nding via F̲ormal ConcepT̲ Analysis), which is designed for semi-supervised multi-label classification of mixed-type data and hence it solves the ligand finding problem from the IUPHAR database. The basic strategy of LIFT is similar to the learning method SELF presented in the previous chapter, which directly handles mixed-type data in the semi-supervised manner. Since SELF cannot treat multi-label classification, we redesign the essential algorithm of SELF to fit to multi-label classification.

LIFT uses "label propagation", or cluster-and-label, which is a typical approach in semi-supervised learning (Dara *et al.*, 2002; Demiriz *et al.*, 1999). This means that it first makes clusters without label information, followed by giving preferences of class labels for each cluster. In LIFT, the clustering process is achieved by *Formal Concept Analysis* (FCA) (Davey and Priestley, 2002; Ganter and Wille, 1998), which is a mathematical data analysis technique originally proposed by Wille (1982). One of successful applications of FCA in data mining is for frequent pattern and association rule mining proposed by Pasquier *et al.* (1999), where closed patterns (itemsets) obtained by FCA is used as condensed "lossless" representations of original patterns. Using FCA, informally, we can introduce a lattice structure, called a *concept lattice* or a *closed set lattice*, which is a partially ordered set of data clusters with respect to subset inclusion, into original data. Many studies used FCA for machine learning and knowledge discovery, such as classification

Formal Concept Analysis (FCA)

concept lattice
closed set lattice

[1] `http://www.genome.jp/kegg/`
[2] `http://www.iuphar-db.org/index.jsp`

(Ganter and Kuznetsov, 2003), clustering (Zhang *et al.*, 2008), and bioinformatics (Blinova *et al.*, 2003; Kaytoue *et al.*, 2011b; Kuznetsov and Samokhin, 2005), but ligand finding presented in this chapter is a novel application of FCA.

To date, no study treats machine learning for ligand finding in the (multi-class) classification point of view. Recently, to the best of our knowledge, there exists only one related study by Ballester and Mitchell (2010), which investigated a machine learning approach to predict the *affinity* of ligands, the strength of docking. Another approach was performed by King *et al.* (1996) for modeling structure-activity relationships (SAR), which can be applied to ligand finding. However, their goal is to understand the chemical model by describing relations using inductive logic programming (ILP), thus their approach is different from ours. On the other hand, most *in silico* studies about receptors and ligands tried to construct a predictive model using domain-specific knowledge, such as the potential energy of a complex, the two-dimensional co-ordinates, and the free energy of binding (Huang *et al.*, 2006; Moitessier *et al.*, 2008), and lots of scoring methods were proposed; *e.g.*, AMBER (Cornell *et al.*, 1995), AutoDock (Huey *et al.*, 2007), and DrugScore (Gohlke *et al.*, 2000). However, to use such a method, some domain-specific background knowledge is required and results depend on them. In contrast, our approach relies on only databases, thereby the user does not need any background knowledge and can easily use and understand results.

affinity

This chapter is organized as follows: Section 7.1 presents the LIFT algorithm. Section 7.2 gives experimental results with methodologies and discussion. Finally we summarize our results and discuss our future works in Section 7.3.

7.1 The LIFT Algorithm

We present the algorithm, LIFT (LIgand Finding via Formal ConcepT Analysis), in this section. LIFT uses exactly the same algorithms for lattice construction from a given database. Namely, Algorithms 6.1 and 6.2 are used for data preprocessing, and FCA is used for clustering (see Section 6.2.1 and 6.2.2 for detail). In the following, we write the concept lattice (the set of concepts) $\mathfrak{B}(G, M, I)$ generated from a given table τ at discretization level k by $\mathfrak{B}^k(\tau)$.

Example 7.1
Given a table $\tau = (H, X)$, where $H = \{\text{HBD}, \text{TPS}, \text{MW}\}$ and $X = x_1, x_2, x_3$ such that

$$\begin{aligned}
(x_1(\text{HBD}), x_1(\text{TPS}), x_1(\text{MW})) &= (0, 0.61, 0.98),\\
(x_2(\text{HBD}), x_2(\text{TPS}), x_2(\text{MW})) &= (0, 0.44, 0.74),\\
(x_3(\text{HBD}), x_3(\text{TPS}), x_3(\text{MW})) &= (1, 0.72, 0.34),
\end{aligned}$$

which is visualized as follows:

H		HBD	TPS	MW
	x_1	0	0.61	0.98
X	x_2	0	0.44	0.74
	x_3	1	0.72	0.34

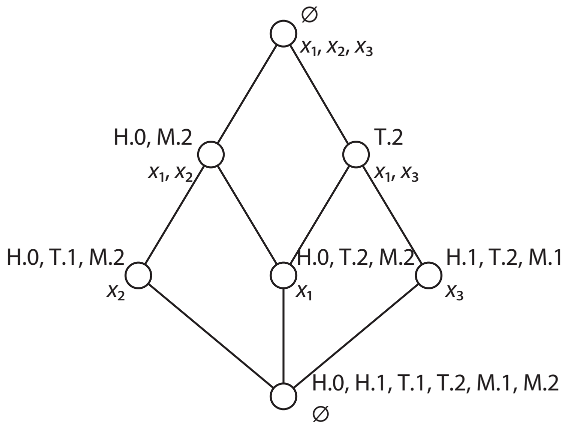


Figure 7.2 | The concept lattice constructed from the context in Example 7.1. Feature names HBD, TPS, and MW are abbreviated as H, T, and M, respectively.

Let discretization level $k = 1$. For each feature, we have the context as follows:

$$\begin{aligned}
M_{\text{HBD}} &= \{\text{HBD.0}, \text{HBD.1}\},\\
I_{\text{HBD}} &= \{(x_1, \text{HBD.0}), (x_2, \text{HBD.0}), (x_3, \text{HBD.1})\},\\
M_{\text{TPS}} &= \{\text{TPS.1}, \text{TPS.2}\},\\
I_{\text{TPS}} &= \{(x_1, \text{TPS.2}), (x_2, \text{TPS.1}), (x_3, \text{TPS.2})\},\\
M_{\text{MW}} &= \{\text{MW.1}, \text{MW.2}\},\\
I_{\text{MW}} &= \{(x_1, \text{MW.2}), (x_2, \text{MW.2}), (x_3, \text{MW.1})\}.
\end{aligned}$$

Thus we have the context (G, M, I) such that

$$\begin{aligned}
G &= \{x_1, x_2, x_3\},\\
M &= M_{\text{HBD}} \cup M_{\text{TPS}} \cup M_{\text{MW}},\\
I &= I_{\text{HBD}} \cup I_{\text{TPS}} \cup I_{\text{MW}},
\end{aligned}$$

which is visualized as a cross-table in the following.

	HBD.0	HBD.1	TPS.1	TPS.2	MW.1	MW.2
x_1	×			×		×
x_2	×		×			×
x_3		×		×	×	

The set $\mathfrak{B}^k(\tau)$ consists of seven concepts in total:

$$\begin{aligned}
&(\emptyset, \{\text{HBD.0}, \text{HBD.1}, \text{TPS.1}, \text{TPS.2}, \text{MW.1}, \text{MW.2}\}),\\
&(\{x_1\}, \{\text{HBD.0}, \text{TPS.2}, \text{MW.2}\}), (\{x_2\}, \{\text{HBD.0}, \text{TPS.1}, \text{MW.2}\}),\\
&(\{x_3\}, \{\text{HBD.1}, \text{TPS.2}, \text{MW.1}\}), (\{x_1, x_2\}, \{\text{HBD.0}, \text{MW.2}\}),\\
&(\{x_1, x_3\}, \{\text{TPS.2}\}), and(\{x_1, x_2, x_3\}, \emptyset).
\end{aligned}$$

We show the corresponding concept lattice in Figure 7.2. □

Example 7.2
Let us consider a table $\tau = (H, X)$ such that $H =$ {TPS, MW, XLogP} and $X =$

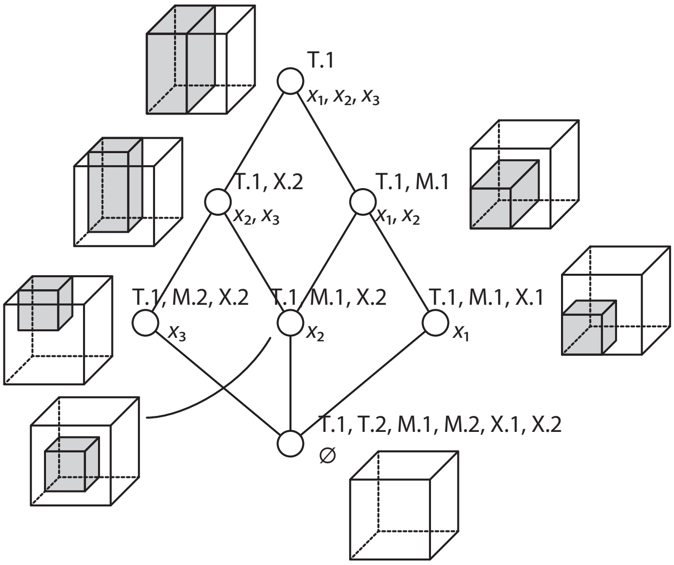


Figure 7.3 | The concept lattice from the context in Example 7.2 with its geometric interpretation.

x_1, x_2, x_3 such that

$$\begin{aligned}(x_1(\text{TPS}), x_1(\text{MW}), x_1(\text{XLogP})) &= (0.23, 0.12, 0.18),\\ (x_2(\text{TPS}), x_2(\text{MW}), x_2(\text{XLogP})) &= (0.35, 0.03, 0.74),\\ (x_3(\text{TPS}), x_3(\text{MW}), x_3(\text{XLogP})) &= (0.41, 0.79, 0.91),\end{aligned}$$

which is represented as follows:

H		TPS	MW	XLogP
X	x_1	0.23	0.12	0.18
	x_2	0.35	0.03	0.74
	x_3	0.41	0.79	0.91

The concept lattice (Figure 7.3) constructed from the context:

	T.1	T.2	M.1	M.2	X.1	X.2
x_1	×		×		×	
x_2	×		×			×
x_3	×			×		×

corresponds to a hierarchy of cubes in three-dimensional Euclidean space. □

7.1.1 Multi-label Classification and Ranking

Here we discuss classification on concept lattices using label information. Our strategy is to design *preference*, a kind of *weight*, for each label of a given test datum (unlabeled tuple) y based on concepts produced by FCA, and achieve multi-label classification based on the preference. Moreover, we show that label ranking can be performed using the preference.

preference

First LIFT translates a tuple y of a table $\upsilon = (H, y)$ into a context with just one object using Algorithms 6.1 and 6.2; *i.e.*, G is the singleton $\{y\}$. We always assume that the header H is exactly the same as that of a table $\tau = (H, X)$ of training data.

classification rule

m-consistency

The key idea is, for each concept $(A, B) \in \mathfrak{B}^k(\tau)$ obtained from a table τ of training data, to treat the set of attributes B as a *classification rule*. For an unlabeled tuple y, we check whether or not the object y has the all attributes of the concept (A, B), since this condition means that the object y has the same properties of the objects A, meaning that y is classified to the same class of objects in A. We call this property *m-consistency* which is formally defined as follows:

m-consistent

Definition 7.3
For a context $(\{y\}, M, I)$ and a concept (A, B), the object y is m-*consistent* with (A, B) if both conditions

$$B \subseteq \{\, m \in M \mid (y, m) \in I \,\} \text{ and } B \neq \emptyset$$

hold.

The notion of m-consistency has the monotonicity with respect to the order $\leq$ on a concept lattice. If an object y is m-consistent with a concept (A, B), it is m-consistent with any concept (C, D) such that $(A, B) \leq (C, D)$, and if an object y is not m-consistent with a concept (A, B), it is not m-consistent with any concept (C, D) such that $(C, D) \leq (A, B)$. Thus, for the set of concepts $\mathfrak{B}^k(\tau)$, if we define

$$\mathcal{C}(y) := \left\{ (A, B) \in \mathfrak{B}^k(\tau) \;\middle|\; y \text{ is } m\text{-consistent with } (A, B) \right\},$$

there always exist finite concepts $(A_1, B_1), (A_2, B_2), \dots, (A_l, B_l)$ such that

$$\bigcup_{i \in \{1, 2, \dots, l\}} \uparrow\!(A_i, B_i) = \mathcal{C}(y),$$

where

$$\uparrow\!(A, B) = \left\{ (C, D) \in \mathfrak{B}^k(\tau) \;\middle|\; (A, B) \leq (C, D) \right\}.$$

Here we give the formal definition of the m-preference of a label. We denote the set of labels associated with a tuple (object) x by $\Lambda(x)$. Notice that $\Lambda(x)$ was a single label in the previous chapter since we considered single-class classification, but here it is a set of labels to take multi-class into account. Thereby, for a set of tuples (objects) A, $\Lambda(A)$ denotes the set $\bigcup_{x \in A} \Lambda(x)$. LIFT allows unlabeled data for training, hence $\Lambda(x)$ could be empty, meaning that the object x is unlabeled. This is why LIFT is a semi-supervised learning algorithm.

m-preference at discretization level *k*

Definition 7.4: *m*-preference at discretization level *k*
Given tables $\tau = (H, X)$ and $\upsilon = (H, y)$ with $|\upsilon| = 1$. For each discretization level k and each label $\lambda \in \mathcal{L}$, we define the *m-preference of λ at discretization level k* with respect to the tuple y by

$$\mathrm{mPref}_y^k(\lambda \,|\, \tau) := \sum \left\{ \#\Lambda(A)^{-1} \;\middle|\; \begin{array}{l} y \text{ is } m\text{-consistent with } (A, B) \in \mathfrak{B}^k(\tau) \\ \text{such that } \lambda \in \Lambda(A) \end{array} \right\}$$

where $\#\Lambda(A)$ denotes the number of elements in $\Lambda(A)$, and we assume $\#\Lambda(A)^{-1} = 0$ if $\#\Lambda(A) = 0$ for simplicity.

We do not take the size $\#A$ of the extent A into account, since the distribution of training data with respect to labels is often biased, especially in biological data.

Example 7.5
Let $\tau = (H, X)$ be a table in Example 7.2 (see Figure 7.3) and tables $\upsilon = (H, y)$ and $\sigma = (H, z)$ be

$$(y(\text{TPS}), y(\text{MW}), y(\text{XLogP})) = (0.12, 0.41, 0.31),$$
$$(z(\text{TPS}), z(\text{MW}), z(\text{XLogP})) = (0.31, 0.22, 0.89).$$

Assume that $\Lambda(x_1) = \{\text{A}\}$, $\Lambda(x_2) = \{\text{B}\}$, and $\Lambda(x_3) = \{\text{C}\}$ in X. Binary relations I_υ and I_σ at discretization level 1 for objects y and z are

$$I_\upsilon = \{(y, \text{TPS.1}), (y, \text{MW.1}), (y, \text{XLogP.1})\},$$
$$I_\sigma = \{(z, \text{TPS.1}), (z, \text{MW.1}), (z, \text{XLogP.2})\}.$$

The object y is m-consistent with three concepts $(\{x_1, x_2, x_3\}, \{\text{TPS.1}\})$, $(\{x_1, x_2\}, \{\text{TPS.1, MW.1}\})$, and $(\{x_1\}, \{\text{TPS.1, MW.1, XLogP.1}\})$, and z is m-consistent with four concepts $(\{x_1, x_2, x_3\}, \{\text{TPS.1}\})$, $(\{x_1, x_2\}, \{\text{TPS.1, MW.1}\})$, $(\{x_2, x_3\}, \{\text{TPS.1, XLogP.2}\})$, and $(\{x_2\}, \{\text{TPS.1, MW.1, XLogP.2}\})$. Thus we have the m-preference

$$\text{mPref}^1_y(\text{A} \mid \tau) = \frac{1}{3} + \frac{1}{2} + 1 = 1.83,\ \text{mPref}^1_y(\text{B} \mid \tau) = \frac{1}{3} + \frac{1}{2} = 0.83,$$
$$\text{mPref}^1_y(\text{C} \mid \tau) = \frac{1}{3} = 0.33,$$
$$\text{mPref}^1_z(\text{A} \mid \tau) = \frac{1}{3} + \frac{1}{2} = 0.83,\ \text{mPref}^1_z(\text{B} \mid \tau) = \frac{1}{3} + \frac{1}{2} + \frac{1}{2} + 1 = 2.33,$$
$$\text{mPref}^1_z(\text{C} \mid \tau) = \frac{1}{3} + \frac{1}{2} = 0.83.$$

These results of m-preferences reflect the similarity between data, since y and z are most similar to the first and second tuples of X, respectively. □

It is easy to achieve multi-class classification from the m-preference at some fixed discretization level. However, this m-preference would not be enough to exploit information from obtained data. We show a simple representative case in the following, which shows the *anti-monotonicity* of the notion of m-consistency with respect to discretization level.

anti-monotonicity

Example 7.6
Let $\tau = (H, X)$ be a table such that $H = \{\text{HBD, TPS}\}$ and $X = x_1, x_2$, where

$$(x_1(\text{HBD}), x_1(\text{TPS})) = (0, 0.56),\ \Lambda(x_1) = \{\text{A}\}$$
$$(x_2(\text{HBD}), x_2(\text{TPS})) = (0, 0.91),\ \Lambda(x_2) = \{\text{B}\}$$

and $\upsilon = (H, y)$ be a table such that

$$(y(\text{HBD}), y(\text{TPS})) = (0, 0.11),$$

which is shown as follows:

H		HBD	TPS	Label
X	x_1	0	0.56	A
	x_2	0	0.91	B
y		0	0.11	

At discretization level 1, we have the context

	H.0	T.1	T.2
x_1	×		×
x_2	×		×
y	×	×	

and there are two concepts

$$(\{x_1, x_2\}, \{\text{HBD.0}, \text{TPS.2}\}),\ (\emptyset, \{\text{HBD.0}, \text{TSP.1}, \text{TPS.2}\}).$$

The object y is not m-consistent with any concept, hence

$$\text{mPref}^1_y(\text{A} \mid \tau) = 0,\ \text{mPref}^1_y(\text{B} \mid \tau) = 0.$$

However, at discretization level 2 with the following context,

	H.0	T.1	T.2	T.3	T.4
x_1	×			×	
x_2	×				×
y	×	×			

there are four concepts

$$(\{x_1, x_2\}, \{\text{HBD.0}\}),\ (\{x_1\}, \{\text{HBD.0}, \text{TPS.3}\}),$$
$$(\{x_2\}, \{\text{HBD.0}, \text{TPS.4}\}),\ (\emptyset, \{\text{HBD.0}, \text{TSP.1}, \text{TPS.2}\}),$$

and y is m-consistent with the concept $(\{x_1, x_2\}, \{\text{HBD.0}\})$, thus

$$\text{mPref}^2_y(\text{A} \mid \tau) = 0.5,\ \text{mPref}^2_y(\text{B} \mid \tau) = 0.5.$$

Therefore y can be classified to both classes A and B. □

Ideally, every discretization levels should be taken into account to obtain the m-preference of labels. One of straightforward ways is to obtain the m-preference of a label by summing up m-preferences for each discretization level. However, if we define the m-preference by

$$\text{mPref}_y(\lambda \mid \tau) := \sum_{k \geq 1} \text{mPref}^k_y(\lambda \mid \tau),$$

this m-preference goes to infinity in many cases. We therefore introduce the maximum level $k_{\max}$ of discretization as a parameter.

m-preference

Definition 7.7: *m*-preference
Given tables τ and v, where $|v| = 1$, and a natural number $k_{\max}$. For each label $\lambda \in \mathcal{L}$, we define the *m-preference* of λ by

$$\text{mPref}_y(\lambda \mid \tau) := \sum_{k=1}^{k_{\max}} \text{mPref}^k_y(\lambda \mid \tau)$$

for a tuple y.

Algorithm 7.1: LIFT algorithm

```
Input: Tables $\tau = (H, X)$ and $\upsilon = (H, y)$, and maximum level $k_{\max}$
Output: $m$-preference mPref$_y$ for each label $\lambda \in \mathcal{L}$

function LIFT($\tau, \upsilon, k_{\max}$)
  $k \leftarrow 1$    // $k$ is discretization level
  for each label $\lambda \in \mathcal{L}$
      mPref$_y(\lambda \mid \tau) \leftarrow 0$    // initialization
  end for
  return LEARNING($\tau, \upsilon, k, k_{\max}$)

function LEARNING($\tau, \upsilon, k, k_{\max}$)
  make a concept lattice $\mathfrak{B}^k(\tau)$ from $\tau$ using Algorithms 6.1 and 6.2
  for each label $\lambda \in \mathcal{L}$
      compute the $m$-preference mPref$^k_y(\lambda \mid X)$ at discretization level $k$
      mPref$_y(\lambda \mid X) \leftarrow$ mPref$_y(\lambda \mid X)$ + mPref$^k_y(\lambda \mid X)$
  end for
  if $k = k_{\max}$ then
      return (mPref$_y(\lambda \mid \tau))_{\lambda \in \mathcal{L}}$
  else
      return LEARNING($\tau, \upsilon, k + 1, k_{\max}$)
  end if
```

We abbreviate "$\mid \tau$" of the expression mPref$_y(\lambda \mid \tau)$ if it is understood from context. We give the LIFT algorithm in Algorithm 7.1, which calculates the m-preference for each label.

Example 7.8

Let us consider a table $\tau = (H, X)$ with $H = \{$HBD, RB, TPS, MW$\}$, where labels are associated with each tuple, and a table $\upsilon = (H, y)$ with an unlabeled tuple y, as shown in the following.

H		HBD	TPS	MW	Labels	
	x_1	0	0.98	0.88	A	
X	x_2	1	0.41	0.48	B	C
	x_3	2	0.12	0.71	A	C
y		0	0.77	0.79		

Assume that $k_{\max} = 2$. At discretization level 1 with the context

	H.0	H.1	H.2	T.1	T.2	M.1	M.2
x_1	×				×		×
x_2		×		×		×	
x_3			×	×			×
y	×				×		×

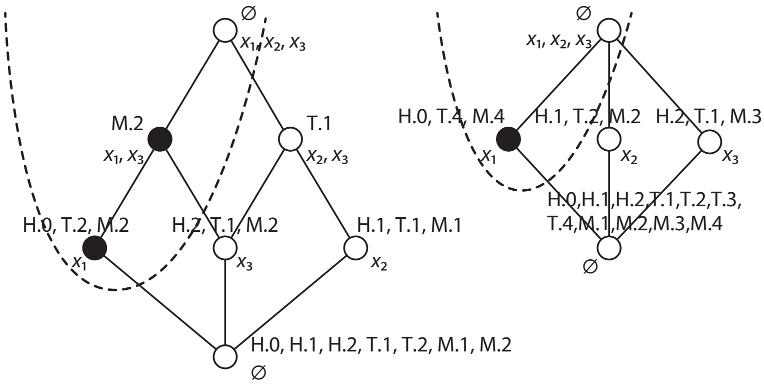


Figure 7.4 | Concept lattices constructed from contexts $\mathfrak{B}^1(\tau)$ (left) and $\mathfrak{B}^2(\tau)$ (right) in Example 7.8. The tuple *y* is *m*-consistent with concepts denoted by black dots.

we have

$$\mathrm{mPref}^1_y(\mathrm{A}) = 1.5,\ \mathrm{mPref}^1_y(\mathrm{B}) = 0,\ \text{and}\ \mathrm{mPref}^1_y(\mathrm{C}) = 0.5,$$

since y is m-consistent with two concepts

$$(A_1, B_1) = (\{x_1, x_3\}, \{\mathrm{MW.2}\})\ \text{and}$$
$$(A_2, B_2) = (\{x_1\}, \{\mathrm{HBD.0}, \mathrm{TPS.2}, \mathrm{MW.2}\}),$$

where $\Lambda(A_1) = \{\mathrm{A}, \mathrm{C}\}$ and $\Lambda(A_2) = \{\mathrm{A}\}$ (see Figure 7.4). Remember that we always ignore the concept whose attribute is empty. At discretization level 2 with the context

	H.0	H.1	H.2	T.1	T.2	T.3	T.4	M.1	M.2	M.3	M.4
x_1	×						×				×
x_2		×			×				×		
x_3			×	×						×	
y	×						×				×

we have

$$\mathrm{mPref}^2_y(\mathrm{A}) = 1, \mathrm{mPref}^2_y(B) = 0,\ \text{and}\ \mathrm{mPref}^2_y(\mathrm{C}) = 0,$$

since y is m-consistent with only one concept $(\{x_1\}, \{\mathrm{HBD.0}, \mathrm{TPS.4}, \mathrm{MW.4}\})$. Finally we have

$$\mathrm{mPref}_y(\mathrm{A}) = 2.5, \mathrm{mPref}_y(\mathrm{B}) = 0,\ \text{and}\ \mathrm{mPref}_y(\mathrm{C}) = 0.5$$

for each label. □

From the m-preference obtained by LIFT, multi-label classification can be performed, that is, an unlabeled tuple y is associated with a set of labels $L \subseteq \mathcal{L}$ such that $L = \{\lambda \in \mathcal{L} \mid \mathrm{mPref}_y(\lambda) \neq 0\}$. Furthermore, a partial order $\preceq$ of labels can be derived from m-preferences, where $\lambda_i \preceq \lambda_j$ (λ_j is preferable than λ_i) if $\mathrm{mPref}_y(\lambda_i) \leq \mathrm{mPref}_y(\lambda_j)$. Thus we can also achieve the label ranking problem using the m-preference.

The time complexity of LIFT is $O(nd) + O(\Delta^3 N)$, which is same as that of SELF.

Example 7.9
For training and test data given in Example 7.8, labels A and C are associated with y since both $\mathrm{mPref}_y(\mathrm{A})$ and $\mathrm{mPref}_y(\mathrm{C})$ are larger then 0. Moreover, we have the order $\mathrm{B} \le \mathrm{C} \le \mathrm{A}$ of label ranking for the tuple y. □

7.2 Experiments

We evaluate the LIFT algorithm using real data of ligands and receptors compared to SVM and the decision tree-based algorithm. We also experimentally measure the effectiveness of unlabeled data for training in semi-supervised learning by LIFT.

7.2.1 Methods

Environment

LIFT was implemented in R and all experiments were performed in R version 2.12.2 (R Development Core Team, 2011). LIFT uses LCM distributed by Uno *et al.* (2005) to construct a concept lattice $\mathfrak{B}^k(\tau)$, which was implemented in C. In all experiments, we used Mac OS X version 10.6.5 with two 2.26-GHz Quad-Core Intel Xeon CPUs and 12 GB of memory.

Databases

We collected the entire 1,782 ligand data in the IUPHAR database (Sharman *et al.*, 2011)[3]. In the database, each ligand is characterized by seven features: HBA, HBD, RB, TPS, MW, XLogP, and NLR as described in introduction of this chapter. Receptors, which corresponds to class labels, are classified into families, such as 5-Hydroxytryptamine receptors and Acetylcholine receptors, hence we picked up the eleven largest families from the database and used respective families as datasets for each training. Statistics of receptor families is shown in Table 7.1. In semi-supervised learning of LIFT, entire ligands except the focusing receptor family were used as unlabeled training data.

Learning Algorithms

To measure the effectiveness of unlabeled ligand data, we used LIFT in two cases: only labeled data were used in training in the first case (denoted by LIFT (w/o) in Figure 7.5), and all ligands except test data were used as unlabeled training data in the second case. The maximum level $k_{\max}$ was set at 5 throughout all experiments. As a control method for evaluation of LIFT, we adopted SVM with the RBF kernel and the decision tree-based method implemented in R (Ripley, 1996), since the tree method is a typical learning algorithm which can be applied to mixed-type data containing both discrete and continuous variables. We used the function `ksvm` in the kernlab package for SVM (Karatzoglou *et al.*, 2004), where all discrete values are treated as continuous. Note that these control methods are typical supervised learning methods and cannot use unlabeled data in the learning phase. Moreover,

[3] `http://www.iuphar-db.org/index.jsp`

Family name	# Ligands	# Receptors
5-Hydroxytryptamine receptors	286	53
Acetylcholine receptors	100	68
Adenosine receptors	162	40
Adrenoceptors	111	35
Dopamine receptors	69	40
Histamine receptors	120	37
Neuropeptide Y receptors	76	34
Metabotropic glutamate receptors	73	9
Transient receptor potential channels	78	58
Voltage-gated potassium channels	61	71
Ionotropic glutamate receptors	81	14

Table 7.1 | Families of receptors. The number of ligands and receptors correspond to the data size and the number of class labels, respectively.

since they are algorithms designed for single-label classification, we just used the first label for each training datum.

Evaluation

Let $v = (H, Y)$ be a test table with $Y = y_1, y_2, \dots, y_n$ and the domain of labels $\mathcal{L}$ be $\{\lambda_1, \lambda_2, \dots, \lambda_l\}$. Assume that we have the m-preference $\{\mathrm{mPref}_y(\lambda_i \,|\, \tau) \mid 1 \le i \le l\}$ for each label $\lambda_i \in \mathcal{L}$ by LIFT, where

$$\mathrm{mPref}_y(\lambda_{p_1} \,|\, \tau) \ge \mathrm{mPref}_y(\lambda_{p_2} \,|\, \tau) \ge \mathrm{mPref}_y(\lambda_{p_3} \,|\, \tau) \ge \dots \ge \mathrm{mPref}_y(\lambda_{p_l} \,|\, \tau).$$

for each tuple $y \in \mathrm{set}(Y)$. In LIFT, we define the accuracy $\mathrm{acc}(v)$ by

$$\mathrm{acc}(v) := \frac{\sum_{i=1}^{n} \#\left\{ \lambda_j \in \Lambda(y_i) \,\middle|\, j \in \{p_1, p_2, \dots, p_{\#\Lambda(y_i)}\} \right\}}{\sum_{i=1}^{n} \#\Lambda(y_i)},$$

which takes values in $[0, 1]$ and to be maximized. This means that when y is associated with q labels, we check whether or not each label is in top-q labels determined by the m-preference. Notice that we do not take labels $\lambda_{p_{\#\Lambda(y)+1}}, \dots, \lambda_{p_l}$ into account to obtain the accuracy since the database has only positive information and $\lambda \notin \Lambda(y)$ *does not* means that the ligand y does not bind to the receptor λ.

For the decision-tree based method and SVM, the accuracy is obtained by

$$\mathrm{acc}(v) := \frac{\#\{y_i \mid 1 \le i \le n,\ f(y_i) \in \Lambda(y_i)\}}{n},$$

where $f(y_i)$ is the output for the tuple y_i by respective learning algorithms.

Mean and s.e.m. (standard error of the mean) of accuracy was obtained for each dataset (receptor family) by 10-fold crossvalidation.

7.2.2 Results and Discussion

Results are shown in Figure 7.5. These results clearly show that LIFT is more effective than the typical classification algorithms of SVM and the tree algorithm for ligand finding. Accuracy obtained by LIFT is significantly higher than that by SVM and the tree algorithm in eight cases out of eleven cases (checked by paired t-test,

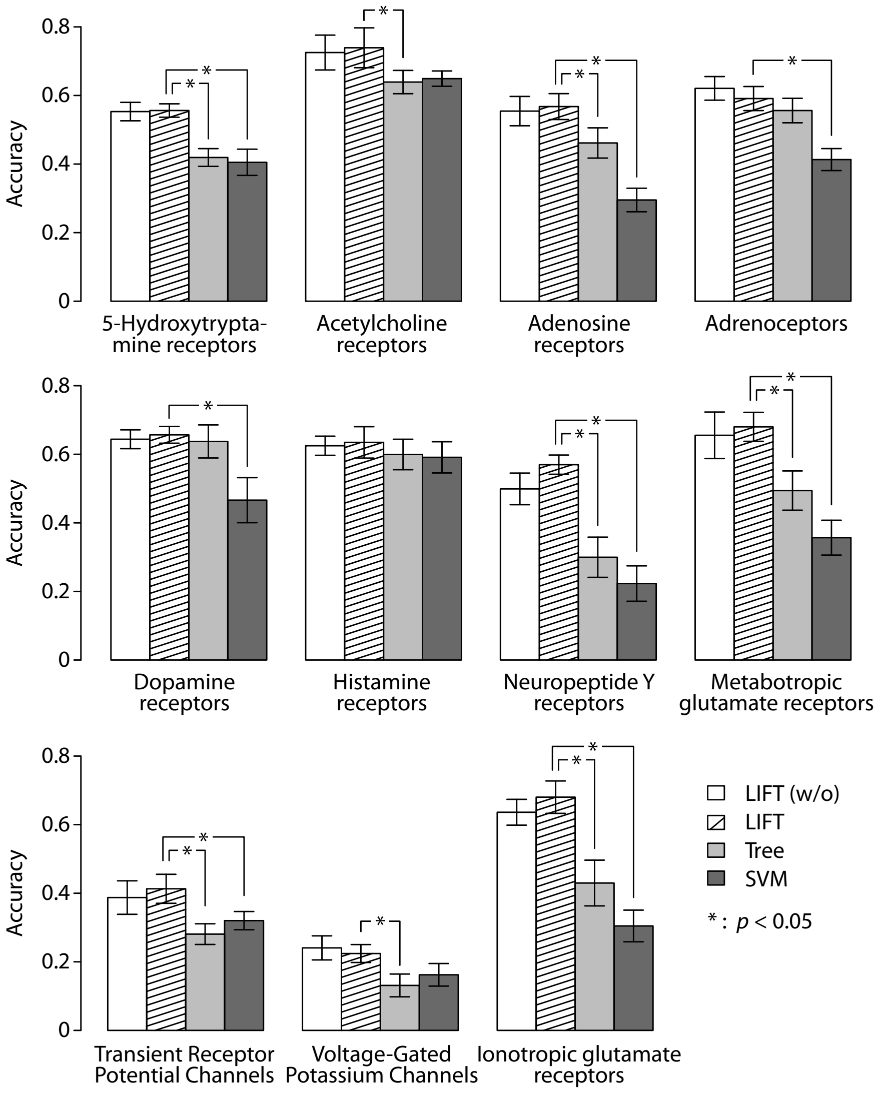


Figure 7.5 | Accuracy for each receptor family obtained by LIFT without unlabeled training data (LIFT (w/o)), LIFT, the tree algorithm, and SVM with the RBF kernel. Data show mean ± s.e.m.

$\alpha = 0.05$). One of reasons of the difference might be that LIFT can treat multi-labels effectively whereas SVM and the tree algorithm cannot. Moreover, SVM treated discrete values as continuous, presumably resulting in lower accuracy.

Since each family has many classes from 9 to 71, accuracy of LIFT, which is more than 50 % in most cases, is high enough. Furthermore, unlabeled training data can be used effectively in LIFT in the semi-supervised manner. Our results therefore indicate that LIFT should be valuable for finding new ligands and contribute to biology and biochemistry.

By using LIFT, we can find new ligand candidates from any training data, hence LIFT can be used as a tool for actual biological experiments to narrow down new ligand candidates. Checking such candidates obtained by LIFT in biological experiments is a future work.

7.3 Summary

In this chapter, we have proposed the semi-supervised learning algorithm, called LIFT, for ligand finding from databases. LIFT performs preference learning, that is, multi-label classification and ranking, in the semi-supervised manner. First, every dataset is translated into a (formal) context, followed by clustering of it by FCA by putting on a concept lattice, where each continuous (real-valued) value is discretized based on the binary encoding scheme. Then, on the lattice, the preferences (formally, m-preferences) of class labels for unlabeled test data are obtained by taking labels of training data into account.

Since LIFT is a flexible learning algorithm, it can be applied to any databases in various domains. Thus considering contributions to other domains is one of the future works. Another future work is to treat incremental databases in LIFT, because lots of databases are frequently updated whereas LIFT cannot directly treat such incremental databases. LIFT can display weighted classification rules, which are easily-interpreted, thus analysis of learned rules from biological point of view is also a future work. Furthermore, using biological background knowledge such as the structure of a receptor for learning is an interesting future work.

8

CONCLUSION

THIS THESIS DEVELOPED several approaches to computational learning. Namely, Part I analyzed theories of computational learning, Part II developed machine learning algorithms using discretization, and Part III presented preference learning algorithms using Formal Concept Analysis.

In Chapter 2, we presented learning of *figures*, nonempty compact sets in Euclidean space, based on the *Gold-style learning model* for a computable foundation for binary classification of multivariate data. Encoding real vectors with no numerical error requires infinite sequences, resulting in the gap between each real vector and its discretized representation used for the actual machine learning process. Our motivation was to bridge the gap to overcome poor computational analysis in binary classification as well as in other machine learning tasks such as regression and clustering. In the chapter, we amalgamated two processes; discretization and binary classification. Each learning target, the set of real vectors classified as positive, is treated as a figure. A learner receives discretized vectors as input data and outputs a sequence of discrete representations of the target figure in the form of *self-similar sets*, known as *fractals*. The generalization error of each output is measured by the *Hausdorff metric*. Using this learning framework, we revealed the hierarchy of learnabilities under various learning criteria in the track of traditional analysis of learnabilities in the Gold-style learning model. Moreover, we showed a mathematical connection between machine learning and fractal geometry by measuring the complexity of learning using the *Hausdorff dimension* and the *VC dimension*. Furthermore, we analyzed computability aspects of learning of figures using the framework of Type-2 Theory of Effectivity (TTE).

In Chapter 3, we proposed a novel measure of the difference between two sets of data, called the *coding divergence*, and unify two processes discretization and learning computationally. Discretization of continuous data was realized by a topological mapping (in the sense of mathematics) from the d-dimensional Euclidean space $\mathbb{R}^d$ into the Cantor space Σ^ω, and the simplest model was learned in the Cantor space, which corresponds to the minimum *open set* separating the given two sets of data. Furthermore, we constructed a classifier using the divergence, and experimentally illustrated robust performance of it.

We proposed in Chapter 4 approaches to exploit compression algorithms for clustering numerical data. Our first contribution was to design a measure, called the *Minimum Code Length* (MCL), that can score the quality of a given clustering result under the light of a *fixed* encoding scheme. Our second contribution was to propose a general strategy to translate any encoding method into a cluster algorithm, called COOL (COding-Oriented cLustering). COOL has a low computational cost since it scales linearly with the data set size. The clustering results of COOL were also shown to minimize MCL. To illustrate further this approach, we considered the *Gray Code* as the encoding scheme to present G-COOL. G-COOL can find clusters of arbitrary shapes and remove noise. Moreover, it is robust to change in the input parameters; it requires only two lower bounds for the number of clusters and the size of each cluster, whereas most algorithms for finding arbitrarily shaped clusters work well only if all parameters are tuned appropriately. G-COOL was theoretically shown to achieve internal cohesion and external isolation and was experimentally shown to work well for both synthetic and real data sets.

In Chapter 5, we presented a fast spatial clustering algorithm for multivariate data, called BOOL (Binary cOding Oriented cLustering), which can detect arbitrarily shaped clusters and is noise tolerant. BOOL handles data using a two-step procedure: data points are first discretized and represented as binary words; clusters are then iteratively constructed by agglomerating smaller clusters using this representation. This latter step is carried out with linear complexity with respect to the number of data points by sorting such binary representations, which results in dramatic speedups when compared with other clustering techniques. Experiments showed that BOOL is faster than *K*-means, and about two to three orders of magnitude faster than two state-of-the-art algorithms that can detect non-convex clusters of arbitrary shapes. We also showed that results of BOOL are robust to changes in parameters, whereas most algorithms for arbitrarily shaped clusters are known to be overly sensitive to such changes. The key to the robustness of BOOL is the hierarchical structure of clusters introduced automatically by increasing the precision of the discretization.

We proposed in Chapter 6 a new approach for semi-supervised learning using *closed set lattices*, which have been recently used for frequent pattern mining within the framework of the data analysis technique of *Formal Concept Analysis* (FCA). We presented a learning algorithm, called SELF (SEmi-supervised Learning via FCA), which performs as a multiclass classifier and a label ranker for *mixed-type data* containing both discrete and continuous variables, whereas only few learning algorithms such as the decision tree-based classifier can directly handle mixed-type data. From both labeled and unlabeled data, SELF constructs a closed set lattice, which is a partially ordered set of data clusters with respect to subset inclusion, via FCA together with discretizing continuous variables, followed by learning classification rules through finding *maximal* clusters on the lattice. Moreover, it can *weight* each classification rule using the lattice, which gives a partial order of preference over class labels. We experimentally illustrated the competitive performance of SELF in classification and ranking compared to other learning algorithms using UCI datasets.

To date, enormous studies have been devoted to investigate biochemical functions of *receptors*, which have crucial roles for signal processing in organisms. *Ligands* are key tools in experiments since receptor specificity with respect to them enables us to control activity of receptors. However, finding ligands is difficult; choosing ligand candidates relies on expert knowledge of biologists and conducting test experiments *in vivo* or *in vitro* has a high cost. In Chapter 7, we investigated the ligand finding problem with a machine learning approach by formalizing the problem as *multi-label classification* mainly discussed in the area of *preference learning*. We developed an algorithm, called LIFT (Ligand FInding via Formal ConcepT Analysis), for multi-label classification, which can treat ligand data in databases in a *semi-supervised* manner. The key to LIFT is to achieve clustering by putting an original dataset on *lattices* using the data analysis technique of *Formal Concept Analysis* (FCA), followed by obtaining the preference for each label using the lattice structure. Experiments using real data of ligands and receptors in the IUPHAR database showed that LIFT effectively solves the task compared to other machine learning algorithms.

To conclude, we believe that our studies on computational learning will become one of key approaches to machine learning for next generation. Discrete structure manipulation is now becoming more and more important to effectively and efficiently treat *big data* and *structured data* (Minato, 2011). This thesis is a bridge between data analysis in the real world, mainly for multivariate data, and discrete and computational manipulations of such data on computers.

APPENDIX A

MATHEMATICAL BACKGROUND

Mathematical background of this thesis is summarized. Namely, sets, functions, topology, and metric space. See literatures (Dieudonné, 1960; Hopcroft and Ullman, 1979) for complete explanation about the basic concepts.

A.1 Sets and Functions

Let A and B be sets. If all elements in A are contained in B, we write $A \subseteq B$. If both $A \subseteq B$ and $A \neq B$ hold, we write $A \subset B$. The notation $A \setminus B$ means the relative complement of B in A. The cardinality of a set A is denoted by $\#A$, and the power set of A is denoted by 2^A. The Cartesian product of two sets A and B, denoted by $A \times B$, is the set of ordered pairs such that

$$A \times B = \{\, (a,b) \bigm| a \in A \text{ and } b \in B \,\}.$$

correspondence
source
target
graph
image

A *correspondence* from a set A to a set B is a triplet $f = (A, B, G(f))$, where $G(f) \subseteq A \times B$. The set A is called the *source*, B the *target*, and $G(f)$ the *graph* of f.

For a subset $X \subseteq A$, the *image* of X under f is defined by

$$f(X) \coloneqq \{\, b \in B \bigm| (a,b) \in G(f) \text{ for some } a \in X \,\},$$

inverse

and the *inverse* of f is defined by

$$f^{-1} \coloneqq \Big(B, A, G(f^{-1}) \Big), \text{ where } G(f^{-1}) \coloneqq \{\, (b,a) \bigm| (a,b) \in G(f) \,\}.$$

domain
range

The *domain* and the *range* of f are written by $\mathrm{dom}(f)$ and $\mathrm{range}(f)$, respectively, which are defined as

$$\begin{aligned} \mathrm{dom}(f) &\coloneqq f^{-1}(B), \\ \mathrm{range}(f) &\coloneqq f(A). \end{aligned}$$

partial function

A correspondence $f = (A, B, G(f))$ is written by $f :\subseteq A \rightrightarrows B$ in general. In particular, for a correspondence $f :\subseteq A \rightrightarrows B$, if an image $f(\{a\})$ has only one elements for every $a \in \mathrm{dom}(f)$, f is called a *partial function* from A to B, written by $f :\subseteq A \to B$. Moreover, if $\mathrm{dom}(f) = A$ holds for a partial function $f :\subseteq A \to B$, f is called a *total function*, written by $f : A \to B$. If f is a total or partial function, f is called a *function* or a *mapping*.

total function

function, mapping

For a partial function $f :\subseteq A \to B$, we define

$$f(a) := \begin{cases} f(\{a\}) & \text{if } a \in \mathrm{dom}(f), \\ \mathrm{div} & \text{if } a \notin \mathrm{dom}(f). \end{cases}$$

A.2 Topology and Metric Space

Let $\mathbb{N}$ be the set of natural numbers including 0, $\mathbb{Q}$ the set of rational numbers, and $\mathbb{R}$ the set of real numbers. The set $\mathbb{N}^+$ (resp. $\mathbb{R}^+$) is the set of positive natural (resp. real) numbers. The d product of $\mathbb{R}$, $\mathbb{R} \times \mathbb{R} \times \ldots \times \mathbb{R}$, is denoted by $\mathbb{R}^d$. Note that $\mathbb{R}^1$ is exactly the same as $\mathbb{R}$. A point x in the space $\mathbb{R}^d$ is denoted by

$$(x^1, x^2, \ldots, x^d) \quad \text{or} \quad \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^d \end{bmatrix}.$$

For any d pairs of real numbers p^i and q^i with $p^i < q^i$ $(i = 1, \ldots, d)$, the set of elements $(x^1, x^2, \ldots, x^d)$ of $\mathbb{R}^d$ such that $p^i < x^i < q^i$ for all $i \in \{1, 2, \ldots, d\}$ is called an *open interval*, written by

open interval

$$(p^1, q^1) \times (p^2, q^2) \times \ldots \times (p^d, q^d);$$

the set of elements $(x^1, x^2, \ldots, x^d)$ of $\mathbb{R}^d$ such that $p^i \le x^i \le q^i$ for all $i \in \{1, 2, \ldots, d\}$ is called a *closed interval*, written by

closed interval

$$[p^1, q^1] \times [p^2, q^2] \times \ldots \times [p^d, q^d].$$

Let X be a set. A *metric* on X is a function

metric

$$\mathfrak{d} : X \times X \to \mathbb{R}$$

such that, for all $x, y, z \in X$,

$$\begin{array}{ll} \mathfrak{d}(x, y) \ge 0 & \text{(non-negativity)}, \\ \mathfrak{d}(x, y) = 0 \text{ if and only if } x = y & \text{(identity)}, \\ \mathfrak{d}(x, y) = \mathfrak{d}(y, x) & \text{(symmetry)}, \\ \mathfrak{d}(x, z) \le \mathfrak{d}(x, y) + \mathfrak{d}(y, z) & \text{(triangle inequality)}. \end{array}$$

A pair $(X, \mathfrak{d})$ denotes a *metric space* X together with $\mathfrak{d}$, that is, for all pairs of elements $x, y \in X$, the distance between them is defined by the metric $\mathfrak{d}$. One of the most popular metric spaces is the *Euclidean space*, where $X = \mathbb{R}^d$ and the metric $\mathfrak{d}$ is the *Euclidean metric* d_{E} such that

metric space

Euclidean space

Euclidean metric

$$d_{\mathrm{E}}(x, y) := \sqrt{\sum_{i=1}^{d} (x^i - y^i)^2}$$

Minkowski metric
L_p metric

for all $x, y \in \mathbb{R}^d$ with $x = (x^1, \dots, x^d)$ and $y = (y^1, \dots, y^d)$. Generally, the *Minkowski metric*, or *L_p metric*, is defined as

$$d_p(x, y) := \left(\sum_{i=1}^{d} \left| x_i - y_i \right|^p \right)^{1/p} .$$

Manhattan metric
Chebyshev metric
Maximum metric

If $p = 1$, this metric coincides with the *Manhattan metric*, and if $p = 2$, it with the Euclidean metric. We also use the case $p = \infty$ (*e.g.*, Subsection 4.3.2), called the *Chebyshev metric* of *Maximum metric*, where

$$d_\infty(x, y) := \max_{i \in \{1,2,\dots,d\}} \left| x_i - y_i \right| .$$

ball

In the following, we consider the d-dimensional Euclidean space $\mathbb{R}^d$ equipped with the Euclidean metric d_{E}. For a point $a \in \mathbb{R}^d$ and $\varepsilon \in \mathbb{R}^+$, the *ball* $B(a, \varepsilon)$ of radius ε centered at a is defined by

$$B(a, \varepsilon) := \left\{ x \in \mathbb{R}^d \;\middle|\; d_{\mathrm{E}}(a, x) < \varepsilon \right\}.$$

inner point, neighborhood
open kernel
outer point
exterior
boundary point
boundary

For a subset $S \subseteq \mathbb{R}^d$ and its element $a \in S$, if there exists ε such that $B(a, \varepsilon) \subseteq S$, a is called an *inner point* of S, and S is called a *neighborhood* of a. The set of inner points of S is said as the *open kernel* of S, denoted by S°. An inner point of $\mathbb{R}^d \setminus S$ is an *outer point* of S. The set of outer points of S is called the *exterior* of S, denoted by S^{e}. Moreover, a point in S which is neither an inner point nor an outer point is said as a *boundary point*. The set of boundary points $\mathbb{R}^d \setminus (S^\circ \cup S^{\mathrm{e}})$ is the *boundary* of S, denoted by S^{b}.

closure

For a subset $S \subseteq \mathbb{R}^d$, the set $S^\circ \cup S^{\mathrm{b}}$ is called the *closure* of S, denoted by $\overline{S}$. In general,

$$S^\circ \subseteq S \subseteq \overline{S}$$

holds. In particular, if

$$S = S^\circ,$$

open set

we call S an *open set*, and if

$$S = \overline{S},$$

closed set

we call it a *closed set*.

Let S be a subset of $\mathbb{R}^d$. If for every arbitrary collection $\{O_i\}_{\iota \in I}$ of open sets with $I \subseteq \mathbb{N}$ such that

$$S \subseteq \bigcup_{i \in I} O_i,$$

there exists a finite subset $\{i_1, i_2, \dots, i_m\}$ of I such that

$$S \subseteq O_{i_1} \cup O_{i_2} \cup \dots \cup O_{i_m},$$

then S is called a *compact set*. The set of compact sets of $\mathbb{R}^d$ is denoted by $\mathcal{K}$, and the set of nonempty compact sets of $\mathbb{R}^d$ is denoted by $\mathcal{K}^*$.

compact set

For a subset $S \subseteq \mathbb{R}^d$, if $S \subseteq B(a, \varepsilon)$ for some ball $B(a, \varepsilon)$, S is said as *bounded*. In the Euclidean space $\mathbb{R}^d$, we can apply the well-known *Heine–Borel theorem*: For a subset $S \subseteq \mathbb{R}^d$, S is compact if and only if S is closed and bounded.

bounded

Heine–Borel theorem

Let us consider a sequence $(x_i)_{i\in\mathbb{N}}$ over X in a metric space $(X, \mathfrak{d})$. If for arbitrary $\varepsilon \in \mathbb{R}^+$ there exists a natural number n_0 such that

$$\mathfrak{d}(x_m, x_n) < \varepsilon$$

for all m, n with $m > n_0$ and $n > n_0$, $(x_i)_{i\in\mathbb{N}}$ is called a *Cauchy sequence*. If every Cauchy sequence has a limit, the metric space $(X, \mathfrak{d})$ is said as *complete*.

Cauchy sequence

complete

In a metric space $(X, \mathfrak{d})$, the *diameter* of a nonempty subset S is defined by

diameter

$$|S| \coloneqq \sup \{\, \mathfrak{d}(x, y) \mid x, y \in S \,\}$$

for all $x, y \in S$. A set S is *countable* if there is a bijection from $\mathbb{N}$ to S. A set $\mathcal{U}$ is a *cover* of $S \subseteq S$ if $\mathcal{U}$ is countable and

countable

cover

$$S \subseteq \bigcup_{U \in \mathcal{U}} U,$$

and $\mathcal{U}$ is a δ-*cover* of X if $\mathcal{U}$ is a cover of X and $|U| \le \delta$ for all $U \in \mathcal{U}$.

δ-cover

Symbols

Sets and Topological Spaces

$\mathbb{N}$	The set of natural numbers including 0
$\mathbb{N}^+$	The set of positive natural numbers; *i.e.*, $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$
$\mathbb{Q}$	The set of rational numbers
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^+$	The set of positive real numbers
d	The number of dimensions ($d \in \mathbb{N}^+$)
$\mathbb{R}^d$	d-dimensional Euclidean space
$\mathcal{K}^*$	The set of figures (nonempty compact subsets of $\mathbb{R}^d$)
(a, b)	The open interval $\{ x \in \mathbb{R} \mid a < x < b \}$
$[a, b]$	The closed interval $\{ x \in \mathbb{R} \mid a \leq x \leq b \}$
$\mathcal{I}^d$	The unit interval $[0, 1] \times \ldots \times [0, 1]$
K, L	Figures (nonempty compact sets)
$\mathcal{F}$	Set of Figures
$\#X$	The number of elements in set X
d_{E}	The Euclidean metric, cf. Section 2.2
d_{H}	The Hausdorff metric, cf. Section 2.2

Fractal Geometry and Dimensions

CT	Contraction for real numbers, cf. Section 2.2
C	Finite set of contractions, cf. Section 2.2
$\mathbf{CT}$	Contraction for figures, cf. Section 2.2
$\mathfrak{H}$	The Hausdorff measure, cf. Subsection 2.5.1
$\dim_{\mathrm{H}}$	The Hausdorff dimension, cf. Subsection 2.5.1
$\dim_{\mathrm{B}}$	The box-counting dimension, cf. Subsection 2.5.1
$\dim_{\mathrm{S}}$	The similarity dimension, cf. Subsection 2.5.1
$\dim_{\mathrm{VC}}$	The VC dimension, cf. Subsection 2.5.1

Strings

Σ	Alphabet
Σ^d	The set of finite sequences whose length are d; *i.e.*, $\Sigma^d = \{a_1 a_2 \ldots a_d \mid a_i \in \Sigma\}$
Σ^*	The set of finite sequences
Σ^+	The set of finite sequences without the empty string λ

Σ^ω	The set of infinite sequences (Cantor space)
λ	The empty string
u, v, w	Finite sequences
$w \sqsubseteq p$	w is a prefix of p ($w \sqsubset p$ is $w \sqsubseteq p$ and $w \neq p$)
$\uparrow w$	The set $\{p \in \Sigma^\omega \mid w \sqsubset p\}$
$\uparrow W$	The set $\{p \in \Sigma^\omega \mid w \sqsubseteq p$ for some $w \in W\}$, cf. Section 3.2
$\langle \cdot \rangle$	The tupling function; *i.e.*, $\langle p^1, p^2, \ldots, p^d \rangle \coloneqq p_0^1 p_0^2 \ldots p_0^d p_1^1 p_1^2 \ldots p_1^d p_2^1 p_2^2 \ldots p_2^d \ldots$, cf. Section 2.2
$\|w\|$	The length of w If $w = \langle w^1, \ldots, w^d \rangle \in (\Sigma^d)^*$, $\|w\| = \|w^1\| = \cdots = \|w^d\|$
$\|W\|$	The size of W defined by $\sum_{w \in W} \|w\|$, cf. Section 3.2
$\mathrm{diam}(k)$	The diameter of the set $\rho(w)$ with $\|w\| = k$; *i.e.*, $\mathrm{diam}(k) = \sqrt{d} \cdot 2^{-k}$, cf. Section 2.2
p, q	Infinite sequences
V, W	Set of finite or infinite sequences

Computable Analysis

ρ	Binary representation, cf. Section 2.2
ξ, ζ	Representation; *i.e.*, a mapping from finite or infinite sequences to some objects, cf. Subsection 2.6.1
$\xi \leq \zeta$	ξ is reducible to ζ, cf. Subsection 2.6.1
$\xi \equiv \zeta$	ξ is equivalent to ζ, cf. Subsection 2.6.1
$\nu_{\mathbb{Q}^d}$	Representation for rational numbers, cf. Subsection 2.6.1
ν_{Q}	Representation for finite sets of rational numbers, cf. Subsection 2.6.1
φ	Embedding
β	Base, cf. Definition 3.3
φ_β	Base-β embedding, cf. Definition 3.3
φ_{G}	Gray code embedding, cf. Definition 4.7
τ_{Σ^ω}	Cantor topology, cf. Definition 3.1

Learning Theory

$\mathcal{H}$	The hypothesis space (the set of finite sets of finite sequences), cf. Section 2.2
$\mathcal{H}_N$	The set $\{\, H \in \mathcal{H} \mid \#H \in N \,\}$, cf. SubSection 2.3.1
H	Hypothesis, cf. Section 2.2
h	Classifier of hypothesis H, cf. Section 2.2
κ	The mapping from hypotheses to figures, cf. Equation (2.7)
$\mathbf{M}$	Learner, cf. Section 2.2
σ	Presentation (informant or text), cf. Section 2.2
$\mathrm{Pos}(K)$	The set of finite sequences of positive examples of K; *i.e.*, $\{w \mid \rho(w) \cap K \neq \emptyset\}$, cf. Section 2.2
$\mathrm{Pos}_k(K)$	The set $\{w \in \mathrm{Pos}(K) \mid \|w\| = k\}$, cf. Subsection 2.3.1
$\mathrm{Neg}(K)$	The set of finite sequences of negative examples of K; *i.e.*, $\{w \mid \rho(w) \cap K = \emptyset\}$, cf. Section 2.2

Databases

$\tau = (H, X)$	Table, pair of header H and body X
$\mathrm{set}(X)$	The set of tuples of body X
h	Feature (element in H)
x, y	Tuple
$x(h)$	Value of x for attribute $h \in H$
$x\|_J$	Projection of x on $J \subseteq H$
$\|\tau\|$	Number of tuples
$\perp$	Missing value
n	Number of data (objects)
d	Number of features
$\mathrm{Dom}(h)$	Domain of the feature h

Clustering

C, D	Cluster (set of objects)
$\mathcal{C}, \mathcal{D}$	Partition (set of clusters)
K	Number of clusters
k	Discretization level
N	Noise parameter
l	Distance parameter

Classification and Ranking

λ	Label
$\mathcal{L}$	The domain of labels, cf. Subsection 6.2.4
$\Lambda(x)$	Single Label (Chapter 6) or set of labels (Chapter 7) of object (tuple) x, cf. Subsection 6.2.3 and Subsection 7.1.1
$\Gamma(G)$	Set of labeled objects in G, cf. Subsection 6.2.3
R	Classification rule (pair of set of attributes and label), cf. Subsection 6.2.3
$\omega(R)$	Weight of classification rule R, cf. Subsection 6.2.3
$\mathrm{lPref}(\lambda)$	l-preference of label λ, cf. Definition 6.8
$\prec_*$	True partial order, cf. Subsection 6.3.1
$\prec$	Predicted partial order, cf. Subsection 6.3.1
$\mathrm{CR}(\prec, \prec_*)$	Correctness of $\prec$, cf. Subsection 6.3.1
$\mathrm{CP}(\prec)$	Completeness of $\prec$, cf. Subsection 6.3.1
$\mathrm{mPref}_y^k(\lambda \mid \tau)$	m-preference of label λ at level k for tuple y with respect to τ, cf. Subsection 7.1.1
$\mathrm{mPref}_y(\lambda \mid \tau)$	m-preference of label λ for tuple y with respect to τ (abbreviated as $\psi_y(\lambda)$ if τ is understood from context), cf. Subsection 7.1.1

Formal Concept Analysis

G	The set of objects, cf. Subsection 6.2.1
M	The set of attributes, cf. Subsection 6.2.1

I	Binary relation between G and M, cf. Subsection 6.2.1
(G, M, I)	Context, cf. Subsection 6.2.1
g	Object, identified with tuple, cf. Subsection 6.2.1
m	Attribute, cf. Subsection 6.2.1
$h.m$	Qualified attribute generated from feature h, cf. Subsection 6.2.1
$''$	Closure operator, cf. Subsection 6.2.2
$\mathfrak{B}(G, M, I)$	Concept lattice, cf. Subsection 6.2.2
$\mathfrak{B}^k(\tau)$	Concept lattice generated from table τ at discretization level k, cf. Section 7.1
$k_{\max}$	Maximum discretization level, cf. Definition 7.7

Bibliography

C. C. Aggarwal and P. S. Yu. On classification of high-cardinality data streams. In *Proceedings of 2010 SIAM International Conference on Data Mining*, pages 802–813, 2010.

C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On demand classification of data streams. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 503–508, 2004.

D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

D. Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.

K. Apsītis, S. Arikawa, R. Freivalds, E. Hirowatari, and C. H. Smith. On the inductive inference of recursive real-valued functions. *Theoretical Computer Science*, 219 (1–2):3–12, 1999.

B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2002.

D. C. Baird. *Experimentation: An Introduction to Measurement Theory and Experiment Design*. Benjamin Cummings, 3 edition, 1994.

P. J. Ballester and J. B. O. Mitchell. A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking. *Bioinformatics*, 26(9):1169–1175, 2010.

M. F. Barnsley. *Fractals Everywhere*. Morgan Kaufmann, 2 edition, 1993.

Y. M. Barzdin. Inductive inference of automata, languages and programs (in Russian). In *Proceedings of the International Congress of Mathematicians*, volume 2, pages 455–460, 1974.

E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural computation*, 1(1):151–160, 1989.

G. A. Beer. *Topologies on Closed and Closed Convex Sets*, volume 268 of *Mathematics and Its Applications*. Kluwer Academic Publishers, 1993.

S. Ben-David and E. Dichterman. Learning with restricted focus of attention. *Journal of Computer and System Sciences*, 56(3):277–298, 1998.

P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

G. Beslon, D. P. Parsons, J. M. Peña, C. Rigotti, and Y. Sanchez-Dehesa. From digital genetics to knowledge discovery: Perspectives in genetic network understanding. *Intelligent Data Analysis*, 14(2):173–191, 2010.

C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.

V. G. Blinova, D. A. Dobrynin, V. K. Finn, S. O. Kuznetsov, and E. S. Pankratova. Toxicology analysis by means of the JSM-method. *Bioinformatics*, 19(10): 1201–1207, 2003.

L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, 1975.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

V. Brattka and G. Presser. Computability on subsets of metric spaces. *Theoretical Computer Science*, 305(1-3):43–76, 2003.

V. Brattka and K. Weihrauch. Computability on subsets of Euclidean space I: Closed and compact subsets. *Theoretical Computer Science*, 219(1-2):65–93, 1999.

G. Brock, V. Pihur, S. Datta, and S. Datta. clValid: An R package for cluster validation. *Journal of Statistical Software*, 25(4):1–22, 2008.

J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of International Congress on Logic, Methodology and Philosophy of Science*, pages 1–12, 1960.

F. Chang, W. Qiu, R. H. Zamar, R. Lazarus, and X. Wang. clues: An R package for nonparametric clustering based on local shrinking. *Journal of Statistical Software*, 33(4):1–16, 2010.

V. Chaoji, M. A. Hasan, S. Salem, and M. J. Zaki. SPARCL: An effective and efficient algorithm for mining arbitrary shape-based clusters. *Knowledge and Information Systems*, 21(2):201–229, 2009.

V. Chaoji, G. Li, H. Yildirim, and M. J. Zaki. ABACUS: Mining arbitrary shaped clusters from large datasets based on backbone identification. In *Proceedings of 2011 SIAM International Conference on Data Mining*, pages 295–306, 2011.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, 2006. URL `http://www.kyb.tuebingen.mpg.de/ssl-book`.

H. Cheng, Z. Liu, and J. Yang. Sparsity induced similarity measure for label propagation. In *12th IEEE International Conference on Computer Vision*, pages 317–324, 2009.

W. Cheng, M. Rademaker, B. De Baets, and E. Hüllermeier. Predicting partial orders: Ranking with abstention. In J. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6321 of *Lecture Notes in Computer Science*, pages 215–230. Springer, 2010.

R. Cilibrasi and P. M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *Journal of the American Chemical Society*, 117(19):5179–5197, 1995.

R. Dara, S. C. Kremer, and D. A. Stacey. Clustering unlabeled data with SOMs improves classification of labeled real-world data. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 3, pages 2237–2242, 2002.

C. J. Date. *An Introduction to Database Systems*. Addison Wesley, 8 edition, 2003.

B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 2 edition, 2002.

M. de Brecht. *Topological and Algebraic Aspects of Algorithmic Learning Theory*. PhD thesis, Graduate School of Informatics, Kyoto University, 2010.

M. de Brecht and A. Yamamoto. Σ^0_α-admissible representations. In *Proceedings of the 6th International Conference on Computability and Complexity in Analysis*, 2009.

M. de Brecht and A. Yamamoto. Topological properties of concept spaces (full version). *Information and Computation*, 208:327–340, 2010.

C. De La Higuera and J.-C. Janodet. Inference of ω-languages from prefixes. In N. Abe, R. Khardon, and T. Zeugmann, editors, *Algorithmic Learning Theory*, volume 2225 of *Lecture Notes in Computer Science*, pages 364–377. Springer, 2001.

S. E. Decatur and R. Gennaro. On learning from noisy and incomplete examples. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 353–360, 1995.

A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. In *Proceedings of Artificial Neural Networks in Engineering*, pages 809–814, 1999.

J. Dieudonné. *Foundations of Modern Analysis*. Academic Press, 1960.

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82 (3):247–261, 1989.

T. Elomaa and J. Rousu. Necessary and sufficient pre-processing in numerical range discretization. *Knowledge and Information Systems*, 5(2):162–182, 2003.

M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, 2003.

U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1029, 1993.

H. Federer. *Geometric Measure Theory*. Springer, 1996.

R. Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, 1925.

R. Fisher. *Statistical Methods and Scientific Inference*. Oliver and Boyd, 1956.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

R. Freivalds and C. H. Smith. On the role of procrastination in machine learning. *Information and Computation*, 107(2):237–271, 1993.

N. Friedman, M. Goldszmidt, and T. J. Lee. Bayesian network classification with continuous attributes: Getting the best of both discretization and parametric fitting. In *Proceedings of the 15th International Conference on Machine Learning*, pages 179–187, 1998.

J. Fürnkranz and E. Hüllermeier, editors. *Preference learning*. Springer, 2010.

M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: A review. *SIGMOD Record*, 34:18–26, 2005.

J. Gama and P. Kosina. Learning decision rules from data streams. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1255–1260, 2011.

J. Gama and C. Pinto. Discretization from data streams: applications to histograms and data mining. In *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, pages 23–27, 2006.

B. Ganter and S. Kuznetsov. Formalizing hypotheses with concepts. In B. Ganter and G. W. Mineau, editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues*, volume 1867 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2000.

B. Ganter and S. Kuznetsov. Hypotheses and version spaces. In A. de Moor, W. Lex, and B. Ganter, editors, *Conceptual Structures for Knowledge Creation and Communication*, volume 2746 of *Lecture Notes in Computer Science*, pages 83–95. Springer, 2003.

B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1998.

H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems: The complete book*. Prentice Hall Press, 2008.

H. Gohlke, M. Hendlich, and G. Klebe. Knowledge-based scoring function to predict protein-ligand interactions1. *Journal of molecular biology*, 295(2):337–356, 2000.

E. M. Gold. Limiting recursion. *The Journal of Symbolic Logic*, 30(1):28–48, 1965.

E. M. Gold. Language identification in the limit. *Information and Control*, 10(5): 447–474, 1967.

S. A. Goldman, S. S. Kwek, and S. D. Scott. Learning from examples with unspecified attribute values. *Information and Computation*, 180(2):82–100, 2003.

L. Goodman and W. Kruskal. *Measures of Association for Cross Classifications*. Springer, 1979.

S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. *Information Systems*, 26(1):35–58, 1998.

M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2):107–145, 2001.

J. Han and M. Kamber. *Data Mining*. Morgan Kaufmann, 2 edition, 2006.

J. Han, M. Kamber, and A. K. H. Tung. Spatial clustering methods in data mining: A survey. In H. J. Miller and J. Han, editors, *Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS*. Taylor and Francis, 2001.

J. Handl, J. Knowles, and D. B. Kell. Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201, 2005.

A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 58–65, 1998.

E. Hirowatari and S. Arikawa. Inferability of recursive real-valued functions. In M. Li and A. Maruoka, editors, *Algorithmic Learning Theory*, volume 1316 of *Lecture Notes in Computer Science*, pages 18–31. Springer, 1997.

E. Hirowatari and S. Arikawa. A comparison of identification criteria for inductive inference of recursive real-valued functions. *Theoretical Computer Science*, 268 (2):351–366, 2001.

E. Hirowatari, K. Hirata, T. Miyahara, and S. Arikawa. Criteria for inductive inference with mind changes and anomalies of recursive real-valued functions. *IEICE Transactions on Information and Systems*, 86(2):219–227, 2003.

E. Hirowatari, K. Hirata, T. Miyahara, and S. Arikawa. Refutability and reliability for inductive inference of recursive real-valued functions. *IPSJ Digital Courier*, 1:141–152, 2005.

E. Hirowatari, K. Hirata, and T. Miyahara. Prediction of recursive real-valued functions from finite examples. In T. Washio, A. Sakurai, K. Nakajima, H. Takeda, S. Tojo, and M. Yokoo, editors, *New Frontiers in Artificial Intelligence*, volume 4012 of *Lecture Notes in Computer Science*, pages 224–234. Springer, 2006.

J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley Publishing Company, 1979.

N. Huang, C. Kalyanaraman, K. Bernacki, and M. P. Jacobson. Molecular mechanics methods for predicting protein–ligand binding. *Physical Chemistry Chemical Physics*, 8(44):5166–5177, 2006.

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, 1985.

R. Huey, G. M. Morris, A. J. Olson, and D. S. Goodsell. A semiempirical free energy force field with charge-based desolvation. *Journal of computational chemistry*, 28(6):1145–1152, 2007.

E. Hülermeier, J. Fünkranz, W. Cheng, and K. Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16–17):1897–1916, 2008.

G. Hulten and P. Domingos. VFML – a toolkit for mining high-speed time-changing data streams. 2003. URL `http://www.cs.washington.edu/dm/vfml/`.

D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, pages 850–863, 1993.

T. Hwang and R. Kuang. A heterogeneous label propagation algorithm for disease gene discovery. In *SIAM International Conference on Data Mining*, pages 583–594, 2010.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999a.

S. Jain. Hypothesis spaces for learning. *Information and Computation*, 209(3): 513–527, 2011.

S. Jain, D. Osherson, J. S. Royer, and A. Sharma. *Systems That Learn*. The MIT Press, 2 edition, 1999b.

S. Jain, E. Kinber, R. Wiehagen, and T. Zeugmann. Learning recursive functions refutably. In N. Abe, R. Khardon, and T. Zeugmann, editors, *Algorithmic Learning Theory*, volume 2225 of *Lecture Notes in Computer Science*, pages 283–298, 2001.

S. Jain, Q. Luo, P. Semukhin, and F. Stephan. Uncountable automatic classes and learning. *Theoretical Computer Science*, 412(19):1805–1820, 2011.

K. P. Jantke. Monotonic and non-monotonic inductive inference. *New Generation Computing*, 8(4):349–360, 1991.

R. Jaschke, A. Hotho, C. Schmitz, B. Ganter, and G. Stumme. TRIAS–An algorithm for mining iceberg tri-lattices. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 907–911, 2006.

D. H. Johnson. The insignificance of statistical significance testing. *The journal of wildlife management*, 63(3):763–772, 1999.

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab–an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

G. Karypis, H. Eui-Hong, and V. Kumar. CHAMELEON: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

M. Kaytoue, S. O. Kuznetsov, and A. Napoli. Revisiting numerical pattern mining with formal concept analysis. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1342–1347, 2011a.

M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining gene expression data with pattern structures in formal concept analysis. *Information Sciences*, 181:1989–2001, 2011b.

M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

A. S. Kechris. *Classical Descriptive Set Theory*. Springer, 1995.

E. Keogh, S. Lonardi, C. Ratanamahatana, L. Wei, S.-H. Lee, and J. Handley. Compression-based data mining of sequential data. *Data Mining and Knowledge Discovery*, 14:99–129, 2007.

R. Khardon and D. Roth. Learning to reason with a restricted view. *Machine Learning*, 35(2):95–116, 1999.

E. Kinber. Monotonicity versus efficiency for learning languages from texts. In *Algorithmic Learning Theory*, volume 872 of *Lecture Notes in Computer Science*, pages 395–406. Springer, 1994.

R. D. King, S. H. Muggleton, A. Srinivasan, and M. J. E. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93(1):438–442, 1996.

D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations*. Addison-Wesley Professional, 2005.

S. Kobayashi. Approximate identification, finite elasticity and lattice structure of hypothesis space. Technical Report CSIM 96-04, Department of Computer Science and Information Mathematics, The University of Electro-Communications, 1996.

S. Kok and P. Domingos. Learning Markov logic network structure via hypergraph lifting. In *Proceedings of the 26th International Conference on Machine Learning*, pages 505–512, 2009.

P. Kontkanen and P. Myllymäki. An empirical comparison of NML clustering algorithms. In *Proceedings of Information Theory and Statistical Learning*, pages 125–131, 2008.

P. Kontkanen, P. Myllymäki, T. Silander, and H. Tirri. A bayesian approach to discretization. In *Proceedings of the European Symposium on Intelligent Techniques*, pages 265–268, 1997.

P. Kontkanen, P. Myllymäki, W. Buntine, J. Rissanen, and H. Tirri. An MDL framework for data clustering. In P. Grünwald, I. J. Myung, and M. Pitt, editors, *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2005.

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

S. O. Kuznetsov. Machine learning and formal concept analysis. In P. Eklund, editor, *Concept Lattices*, volume 2961 of *Lecture Notes in Computer Science*, pages 287–312. Springer, 2004.

S. O. Kuznetsov and M. V. Samokhin. Learning closed sets of labeled graphs for chemical applications. In S. Kramer and B. Pfahringer, editors, *Inductive Logic Programming*, volume 3625 of *Lecture Notes in Computer Science*, pages 190–208. Springer, 2005.

S. Lange and T. Zeugmann. Monotonic versus non-monotonic language learning. In *Nonmonotonic and Inductive Logic*, volume 659 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 1993.

S. Lange and T. Zeugmann. Characterization of language learning front informant under various monotonicity constraints. *Journal of Experimental & Theoretical Artificial Intelligence*, 6(1):73–94, 1994.

S. Lange, T. Zeugmann, and S. Zilles. Learning indexed families of recursive languages from positive data: A survey. *Theoretical Computer Science*, 397(1–3): 194–232, 2008.

M. Li, J. H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154, 2001.

M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi. The similarity metric. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 863–872, 2003.

J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 1–11, 2003.

J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data mining and knowledge discovery*, 15(2):107–144, 2007.

F. T. Liu, K. M. Ting, and Z. H. Zhou. Isolation forest. In *Proceedings of 8th IEEE International Conference on Data Mining*, pages 413–422, 2008.

H. Liu, F. Hussain, C. L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423, 2002.

P. M. Long and L. Tan. PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. *Machine Learning*, 30(1):7–21, 1998.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.

M. Maechler, P. Rousseeuw, A. Struyf, and M. Hubert. Cluster analysis basics and extensions, 2005.

K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Algorithm Theory – SWAT 2004*, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2004.

B. B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman, 1982.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of 8th International Conference on Computer Vision*, volume 2, pages 416–423, 2001.

W. Merkle and F. Stephan. Refuting learning revisited. *Theoretical Computer Science*, 298(1):145–177, 2003.

L. Michael. Partial observability and learnability. *Artificial Intelligence*, 174(11): 639–669, 2010.

L. Michael. Missing information impediments to learnability. In *24th Annual Conference on Learning Theory*, pages 1–2, 2011.

S. Minato. Overview of erato minato project: The art of discrete structure manipulation between science and engineering. *New Generation Computing*, 29(2): 223–228, 2011.

E. Minicozzi. Some natural properties of strong-identification in inductive inference. *Theoretical Computer Science*, 2(3):345–360, 1976.

N. Moitessier, P. Englebienne, D. Lee, J. Lawandi, and C. R. Corbeil. Towards the development of universal, fast and highly accurate docking/scoring methods: a long way to go. *British journal of pharmacology*, 153(S1):S7–S26, 2008.

T. Motoki, T. Shinohara, and K. Wright. The correct definition of finite elasticity: Corrigendum to identification of unions. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, page 375, 1991.

Y. Mukouchi and S. Arikawa. Towards a mathematical theory of machine discovery from facts. *Theoretical Computer Science*, 137(1):53–84, 1995.

Y. Mukouchi and M. Sato. Refutable language learning with a neighbor system. *Theoretical Computer Science*, 298(1):89–110, 2003.

N. Müller. The iRRAM: Exact arithmetic in C++. In J. Blanck, V. Brattka, and P. Hertling, editors, *Computability and Complexity in Analysis*, volume 2064 of *Lecture Notes in Computer Science*, pages 222–252. Springer, 2001.

S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

J. Neyman and E. S. Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference: Part I. *Biometrika*, 20(1):175–240, 1928.

J. Neyman and E. S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A*, 231:289–337, 1933.

T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26:1955–1988, 2005.

S. Oishi. Why research on numerical computation with result verification? (in Japanese). *Fundamentals Review*, 2(2):9–19, 2008.

N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

D. Perrin and J.-É. Pin. *Infinite words*. Elsevier, 2004.

M. Plantevit, A. Laurent, D. Laurent, M. Teisseire, and Y. W. Choong. Mining multidimensional and multilevel sequential patterns. *ACM Transactions on Knowledge Discovery from Data*, 4(1):4–37, 2010.

W. Qiu and H. Joe. Generation of random clusters with specified degree of separation. *Journal of Classification*, 23:315–334, 2006.

J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2011. URL `http://www.R-project.org`.

W. S. Rasband. *ImageJ*. U. S. National Institutes of Health, Bethesda, Maryland, USA, 1997–2011. URL `http://imagej.nih.gov/ij/`.

B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

A. Sakurai. Inductive inference of formal languages from positive data enumerated primitive-recursively. In *Algorithmic Learning Theory*, pages 73–83. JSAI, 1991.

J. Saquer and S. Jitender. Using closed itemsets for discovering representative association rules. In Z. Ras and S. Ohsuga, editors, *Foundations of Intelligent Systems*, volume 1932 of *Lecture Notes in Computer Science*, pages 495–504. Springer, 2010.

E. Schikuta and M. Erhart. The BANG-clustering system: Grid-based data analysis. In *Proceedings of Advances in Intelligent Data Analysis Reasoning about Data*, volume 1280 of *Lecture Notes in Computer Science*, pages 513–524. Springer, 1997.

M. Schröder. Extended admissibility. *Theoretical Computer Science*, 284(2): 519–538, 2002a.

M. Schröder. *Admissible representations for continuous computations*. PhD thesis, dem Fachbereich Informatik, der FernUniversit¨at – Gesamthochschule in Hagen, 2002b.

E. Y. Shapiro. Inductive inference of theories from facts. Technical report, Department of Computer Science, Yale University, 1981.

E. Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1983.

J. L. Sharman, C. P. Mpamhanga, M. Spedding, P. Germain, B. Staels, C. Dacquet, V. Laudet, A. J. Harmar, and NC-IUPHAR. IUPHAR-DB: New receptors and tools for easy searching and visualization of pharmacological data. *Nucleic Acids Research*, 39:D534–D538, 2011. Database Issue.

G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 428–439, 1998.

D. A. Simovici and C. Djeraba. *Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics*. Springer, 2008.

M. Skubacz and J. Hollmén. Quantization of continuous input variables for binary classification. In *Intelligent Data Engineering and Automated Learning — IDEAL 2000. Data Mining, Financial Engineering, and Intelligent Agents*, volume 1983 of *Lecture Notes in Computer Science*, pages 42–47. Springer, 2000.

M. B. Smyth. Topology. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 641–761. Oxford University Press, 1992.

N. R. Tavana and K. Weihrauch. Turing machines on represented sets, a model of computation for analysis. *Logical Methods in Computer Science*, 7(2):1–21, 2011.

K. M. Ting and J. R. Wells. Multi-dimensional mass estimation and mass-based clustering. In *Proceedings of 10th IEEE International Conference on Data Mining*, pages 511–520, 2010.

B. Trakhtenbrot and Y. M. Barzdin. Konetschnyje awtomaty (powedenie i sintez), 1970. English Translation: Finite automata-behavior and synthesis, Fundamental Studies in Computer Science 1, 1975.

H. Tsuiki. Real number computation through Gray code embedding. *Theoretical Computer Science*, 284(2):467–485, 2002.

A. M. Turing. On computable numbers, with the application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 1(42):230–265, 1937.

T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 77–86, 2005.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

P. Valtchev, R. Missaoui, and R. Godin. Formal concept analysis for knowledge discovery and data mining: The new challenges. In P. Eklund, editor, *Concept Lattices*, volume 2961 of *Lecture Notes in Computer Science*, pages 352–371. Springer, 2004.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2): 264–280, 1971.

V. Vapnik and A. Sterin. On structural risk minimization or overall risk in a problem of pattern recognition. *Automation and Remote Control*, 10(3):1495–1503, 1977.

S. Vembu and T. Gärtner. Label ranking algorithms: A survey. In *Preference Learning*, pages 45–64. Springer, 2010.

F. Wang and C. Zhang. Label propagation through linear neighborhoods. In *Proceedings of the 23rd international conference on Machine learning*, pages 985–992, 2006.

W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 186–195, 1997.

K. Weihrauch. *Computable Analysis: An Introduction*. Springer, 2000.

K. Weihrauch. The computable multi-functions on multi-represented sets are closed under programming. *Journal of Universal Computer Science*, 14(6): 801–844, 2008.

K. Weihrauch and T. Grubba. Elementary computable topology. *Journal of Universal Computer Science*, 15(6):1381–1422, 2009.

R. Wiehagen. A thesis in inductive inference. In J. Dix, K. P. Jantke, and P. H. Schmitt, editors, *Nonmonotonic and Inductive Logic*, volume 543 of *Lecture Notes in Computer Science*, pages 184–207. Springer, 1991.

R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In *Ordered Sets*, pages 445–470. D. Reidel Publishing Company, 1982. This article is included in Formal Concept Analysis, LNCS 5548, 314–339, Springer (2009).

K. Wright. Identification of unions of languages drawn from an identifiable class. In *Proceedings of the 2nd Annual Workshop on Computational Learning Theory*, pages 328–333, 1989.

M. J. Zaki. Generating non-redundant association rules. In *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 34–43, 2000.

T. Zeugmann and S. Zilles. Learning recursive functions: A survey. *Theoretical Computer Science*, 397(1-3):4–56, 2008.

T. Zeugmann, S. Lange, and S. Kapur. Characterizations of monotonic and dual monotonic language learning. *Information and Computation*, 120(2):155–173, 1995.

Y. Zhang, B. Feng, and Y. Xue. A new search results clustering algorithm based on formal concept analysis. In *Proceedings of 5th International Conference on Fuzzy Systems and Knowledge Discovery*, pages 356–360. IEEE, 2008.

C. Zhao, W. Shi, and Y. Deng. A new Hausdorff distance for image matching. *Pattern Recognition Letters*, 26(5):581–586, 2005.

X. Zhu and A. B. Goldberg. *Introduction to semi-supervised learning*. Morgan and Claypool Publishers, 2009.

Publications by the Author

Books

[B1] 小林茂夫, 杉山麿人.: 生命科学研究に成功するための統計法ノート, 講談社, 2009.

Journal Papers

[J1] Sugiyama, M., Yamamoto, A.: Semi-Supervised Learning on Closed Set Lattices, *Intelligent Data Analysis*, IOS Press, 17(3), 2013, to appear.

[J2] Sugiyama, M., Imajo, K., Otaki, K., Yamamoto, A.: Semi-Supervised Ligand Finding Using Formal Concept Analysis, *IPSJ Transactions on Mathematical Modeling and Its Applications* (*TOM*), The Information Processing Society of Japan (IPSJ), accepted.

Peer-reviewed Conference Proceedings

[P1] Sugiyama, M., Hirowatari, E., Tsuiki, H., Yamamoto, A.: Learning Figures with the Hausdorff Metric by Self-similar Sets (extended abstract), In *Proceedings of 6th Workshop on Learning with Logics and Logics for Learning* (*LLLL2009*), pp. 27–34, Kyoto, Japan, Jul. 6–7, 2009.

[P2] Sugiyama, M., Hirowatari, E., Tsuiki, H., Yamamoto, A.: Learning Figures with the Hausdorff Metric by Fractals, Hutter, M. and Stephan, F. and Vovk, V. and Zeugmann, T. (eds.), *Algorithmic Learning Theory*, *LNCS* 6331, pp. 315–329, Springer, 2010 (*Proceedings of 21th International Conference on Algorithmic Learning Theory* (*ALT 2010*), Canberra, Australia, Oct. 6–8, 2010)

[P3] Sugiyama, M., Yamamoto, A.: The Coding Divergence for Measuring the Complexity of Separating Two Sets, In *Proceedings of 2nd Asian Conference on Machine Learning* (*ACML2010*), *JMLR Workshop and Conference Proceedings*, vol. 13, pp. 127–143, Tokyo, Japan, Nov. 8–10, 2010

[P4] Sugiyama, M., Yamamoto, A.: Fast Clustering Based on the Gray-Code (extended abstract), In *Proceedings of 7th Workshop on Learning with Logics and Logics for Learning* (*LLLL2011*), p. 42, Osaka, Japan, Mar. 29–30, 2011

[P5] Sugiyama, M., Yamamoto, A.: Semi-Supervised Learning for Mixed-Type Data via Formal Concept Analysis, Andrews, S., Polovina, S., Hill, R., Akhgar, B. (eds.), *Conceptual Structures for Discovering Knowledge*, *LNCS* 6828, pp. 284–297, Springer, 2011 (*Proceedings of the 19th International Conference on Conceptual Structures* (*ICCS2011*), Derby, UK, Jul. 25–29, 2011)

[P6] Sugiyama, M., Yamamoto, A.: The Minimum Code Length for Clustering Using the Gray Code, Gunopulos, D. and Hofmann, T. and Malerba, D. and Vazirgiannis, M. (eds.), *Machine Learning and Knowledge Discovery in Databases*, *LNCS* 6913, pp. 365–380, Springer, 2011 (*Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (*ECML PKDD 2011*), Athens, Greece, Sep. 5–9, 2011)

[P7] Sugiyama, M., Yoshioka, T., Yamamoto, A.: High-throughput Data Stream Classification on Trees, In *Proceedings of Second Workshop on Algorithms for Large-Scale Information Processing in Knowledge Discovery* (*ALSIP 2011*), Kagawa, Japan, Dec. 1–2, 2011

[P8] Sugiyama, M., Yamamoto, A.: A Fast and Flexible Clustering Algorithm Using Binary Discretization, In *Proceedings of the 2011 IEEE International Conference on Data Mining* (*ICDM 2011*), pp. 1212–1217, Vancouver, Canada, Dec. 11–14, 2011

Conference Proceedings

[C1] Sugiyama, M., Imajo, K., Otaki, K., Yamamoto, A.: Discovering Ligands for TRP Ion Channels Using Formal Concept Analysis, In *Proceedings of the 21st International Conference on Inductive Logic Programming* (*ILP 2011*), Windsor Great Park, UK, Jul.31–Aug.3, 2011.

Index