Rule Extraction from Support Vector Machines: A Geometric Approach

Lu Ren

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTING

CITY UNIVERSITY LONDON

Feb 2008

Acknowledgement

I am deeply indebted to my supervisor, Dr. Artur d'Avila Garcez, for his guidance, insight and encouragement throughout the course of my doctoral program and for his careful reading of and constructive criticisms and suggestions on drafts of this thesis and other works.

I owe thanks to many office-mates and friends for their help, discussions and friendship.

I would like to thank the numerous anonymous referees who have reviewed parts of this work and whose valuable comments have contributed to the clarification of many of the ideas presented in this thesis. I would also like to thank my friends for their helpful comments on the draft of the thesis.

Finally, I sincerely thank my husband Dr. Hai Wang, my parents Dahong Ren and Minying Li who were always with me no matter how dubious my decisions were. They always give me warm encouragement and love in my years of study.

Abstract

Despite the success of connectionist systems in prediction and classification problems, critics argue that the lack of symbol processing and explanation capability makes them less competitive than symbolic systems.

Rule extraction from neural networks makes the interpretation of the behaviour of connectionist networks possible by relating sub-symbolic and symbolic processing. However, most rule extraction methods focus only on specific neural network architectures and present limited generalization performance.

Support Vector Machine is an unsupervised learning method that has been recently applied successfully in many areas, and offers excellent generalization ability in comparison with other neural network, statistical, or symbolic machine learning models.

In this thesis, an algorithm called *Geometric and Oracle-Based Support Vector Machines Rule Extraction* (GOSE) has been proposed to overcome the limitations of other rule-extraction methods by extracting comprehensible models from Support Vector Machines (SVM). This algorithm views the extraction as a geometric task. Given a trained SVM network, GOSE queries the synthetic instances and draws conjunction rules by approximating the optimization problem. The extracted rule set also represents the approximation of the SVM classification boundary. Unlike previous works in SVM rule-extraction, GOSE is broadly applicable to different networks and problems because it need not rely on training examples and network architectures. Theoretical proof guarantees that GOSE is capable of approximating the behavior of SVM networks.

Empirical experiments are conducted on different SVM networks from binary classification networks to multi-class networks in various classification domains. The result of experiments demonstrates that GOSE can extract comprehensible rules with high levels of accuracy and fidelity for their corresponding networks. GOSE also exhibits superior consistency. After analyzing and applying several optimizing measures, the complexity of GOSE was improved. In brief, GOSE provides a novel way to explain how an SVM network functions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years, non-symbolic artificial intelligence (AI) techniques have been successfully utilized in many areas and applications. Although these techniques are able to achieve high accuracy, tolerate noise and deal with various types of data such as images, audio and speech, developers prefer not to make use of those empirical AI techniques, because there is insufficient explanation for why and how forecasting results are obtained from AI networks. For example, when neural networks are used for stock predictions, they only provide mechanisms for forecasting, and the knowledge inside the networks, which appears in the form of units and weights, is incomprehensible to users. Therefore, users must blindly trust the predicted results and are unable to validate the prediction results.

Hence, the integration of symbolic and connectionist aims to provide an explanation facility for this problem in non-symbolic AI [55]. Rule extraction offers a solution. The purpose of rule extraction is defined as [52]:

'Given a trained neural network and the data on which it was trained, produce a

description of the network's hypothesis that is comprehensible yet closely approximates the network's predictive behavior.'

Rule extraction has attracted widespread attention because it enhances the utility of non-symbolic AI approaches. [66] presents the following desiderata of rule extraction:

1. **'user explanation' capability:** to use an explicit structure to interpret the internal behavior of a non-symbolic network so that users are able to understand the problem at hand.

2. **transparency of Artificial Neural Network (ANN) structure:** to provide a transparent ANN structure for users to verify whether or not the system requirements have been followed.

3. **generalization improvement:** to provide a mechanism to reduce the generalized error. Users are able to use the extracted rules to identify the circumstances that are not covered and adjust the data set accordingly.

4. **enhancement of knowledge acquisition:** to assist in the improvement of knowledge-based expert systems. The rules gained from the ANN may be added to the existing knowledge base.

Development of Rule Extraction

In past years, many research has been done on rule extraction. Some of notible work are described as follows. Gallant [35] and Saito et al. [43] are pioneers in the development of rule extraction. [35] presents an approach to building up a medical diagnostic expert system from a multi-layer network by using a parallel distributed processing model. [43] presents a technique to extract Boolean rules from individual neurons in both hidden layers and output layers of the ANN.

The *KT* algorithm, an important technique developed by Fu [53], maps the outputs of both hidden units and output units into Boolean rules.

One of the earliest work on rule extraction to treat the network as a black box is called *Validity Interval Analysis (VIA)* [10]. *Validity intervals (VI)* are defined in order to set range restrictions for the activation patterns in the networks. *VIA* then iteratively filters those activation values inconsistent with the weights and biases of the network [19].

Towell and Shavlik [91] present techniques that search the links entering a unit, which ensures the combination of the link weights exceed the bias. The *SUBSET* algorithm tries to find subsets of incoming weights that exceed the bias on a unit. Moreover, another algorithm, *MofN*, which is inherent to the *SUBSET* algorithm, addresses the knowledge inside the network in the form of *" if (M of the following N antecedents are true) then ...."*.

In [92], it treats rule extraction as a learning task is proposed. The method uses training examples and queries to extract the conjunctive and *MofN* rules that could describe the trained neural networks.

Garcez et al. [2] propose another novel approach. It defines a partial ordering on the input vectors of a trained neural network and extracts the pruning and simplification rules. This extraction algorithm is able to be applied on both regular networks and non-regular networks.

While, Setiono [71] [72] focuses on employing the pruned network to extract rules. The network is pruned by removing the redundant connections, and the rules are then generated from the hidden units with a small number of activation values.

Except the traditional trained ANN, many algorithms are presented to extract rules from recurrent networks [102] [63] [17], genetic algorithms (GAs) [83] and

Self-Organizing Map (SOM) [56].

Recently, Support Vector Machine (SVM) started to be considered for rule extraction. The algorithm proposed by Barakat and Diederich [57] first generates patterns for prediction, and then trains those patterns by using decision trees, whose outputs are a set of rules. [59] introduces another approach for rule-extraction from SVMs. It offers a procedure that uses support vectors from SVMs and prototypes, gained from clustering, to define geometric ellipsoids in the input space. The ellipsoids are then mapped to 'if-then' rules.

Current Challenges in Rule Extraction

Although, there exists a substantial amount of work on rule-extraction in a wide range of problem domains.The current development of rule extraction techniques still faces certain challenges [12]:

1. Although successful solutions are found in specific situations, the enhancement of the generality of rule extraction techniques is still under development.
2. To date, an important issue is that the rule extraction techniques are often designed for specific ANN architectures.
3. In most given situations, high accuracy, fidelity and comprehensibility are unable to be achieved at the same time.

1.2 Our approach

In this thesis, we tackle those issues in the above section by proposing a new anytime rule extraction algorithm called Geometric and Oracle-Based Support Vector

Machines Rules Extraction (GOSE). GOSE extracts the rules by approximating the area covered by a class, which is obtained from the trained SVM network. The algorithm contains several steps: querying, clustering, searching, extracting and post-processing.

GOSE extracts the hypercube rules by using the classification boundary and synthetic training instances without considering the inner structure and the support vectors. All we assume is that an SVM is given which we can query and find the classification it gives for input vectors $\mathbf{x}_i$. This step makes GOSE work in a general manner. After querying, clustering is applied to group those $\mathbf{x}_i$ with the same y_i into a set of clusters. Clustering procedure aims to reduce the complexity of GOSE. Then, by means of a binary search algorithm, we look for the points $\mathbf{P}$ that lie on the SVM classification boundaries. It is an preliminary for the rule extraction. Subsequently, an initial optimal rule set can be extracted for the points in $\mathbf{P}$ and synthetic training instance set by solving an optimization problem whereby we attempt to find the largest consistent hypercubes in the input space. Finally, several post-processing measures are applied to this initial rule set in order to derive (a relatively small number of) generalized rules.

GOSE employs SVM networks as the training networks because of their excellent generalization capability and classification accuracy. It treats the SVM as an oracle (black-box), uses synthetic data for querying and rule extraction, thus making fewer assumptions about the training process and the SVM training data. Hence, the algorithm does not depend on the availability of specific training sets for rule extraction. Meanwhile, the knowledge in SVM networks is captured from the information encoded in the geometry of the SVM classification boundary, so that GOSE is not restricted to any specific SVM classifier such as the linear classifier considered in [22]. We examined rule accuracy, fidelity and comprehensibility in three benchmark applications: the Monk's problems, the Iris flower dataset and

the breast cancer-wisconsin dataset. The results indicate that the high correctness of our approach through a high fidelity.

1.3 Thesis Statement

The content of this thesis demonstrates that the combination of connectionist and symbolic techniques can benefit from each other in a novel way.

Here, a hypothesis is put forward: *'The approach proposed in this thesis research is able to provide generalized rules with high accuracy, fidelity, comprehensibility and consistency. At the same time, it is applicable to a wide range of learning models.'* The following chapters will give details of the approach.

1.4 Thesis Contribution

There are three main contributions of this thesis:

1. It presents a novel algorithm, GOSE, to extract rules from SVMs. GOSE views the rule extraction problem as a geometric task without considering the structures of learning models and important support vectors. Hence, it is widely appropriate to a broad class of hyperplane learning models. In addition, the synthetic data generator proposed in GOSE makes the approach independent of any specific training examples and applicable to a variety of problem domains.

2. It implements an extensive evaluation of the rule extraction method. The empirical and analytical evaluation shows a clear synergy between SVMs and symbolic rules.

3. It provides theoretical proofs on quasi-soundness and quasi-completeness for the rule extraction method, which guarantees that each extracted rule approximates the classification of SVM and that a set of rules obtained from GOSE is able to approximate the behavior of the SVM network.

1.5 Thesis Structure

The structure of the thesis is organized as follows:

- Chapter 2 provides some background knowledge. Section 2.1 describes the mathematical models used in this thesis. Section 2.3 provides a review of the SVM network and correlated common SVM algorithms. The rest three sections give a brief introduction to decision boundaries, clustering and the Monte-Carlo methodology.

- Chapter 3 provides a literature review on rule extraction. The rule-extraction techniques are categorized by the translucency criterion as decompositional, pedagogical, compositional and eclectic approaches. The last section in this chapter presents the existing works on SVM rule extraction.

- Chapter 4 details the proposed algorithm, GOSE, which aims to extract interval rules from hyperplane-based classifiers. Unlike previous SVM rule-extraction approaches, it draws rules without dependence on the support vectors and training examples. This novel perspective scales GOSE to diverse network architectures and problem domains.

- Chapter 5 makes an empirical evaluation of GOSE. It is applied to five classification data sets with different attribute types, training set sizes and SVM

networks. Furthermore, Section 5.6 provides a comparison between the results of the experiments with those of other major methods.

- Chapter 6 presents the analytical evaluation of GOSE on scalability and generality. The scalability is discussed at each step of GOSE in terms of complexity. This chapter then presents the generality of GOSE. The final section of this chapter provides a theoretical verification of GOSE.

- Finally, Chapter 7 summarizes the contributions and the limitations of GOSE and puts forward corresponding future work.

Chapter 2

Background

In this chapter, we summarize SVMs and other concepts used in this thesis.

2.1 Some Mathematical Preliminaries

Variance

Variance is a statistic measure that calculates the deviation of the data in a group. The formula is

$$\sigma^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n-1} \tag{2.1}$$

where $\overline{X}$ is the mean of the data in the group.

Probability

Probability is how likely an event A is to occur. The formula is

$$P(A) = \frac{The\ number\ of\ ways\ an\ event\ A\ can\ occur}{The\ total\ number\ of\ possible\ outcomes} \tag{2.2}$$

Joint Probability

Joint Probability is the probability when both events A and B occur together. The formula is written as $P(A \bigcap B)$.

Conditional Probability

Conditional Probability is the probability of the occurrence of an event A, given the occurrence of the other event B. The formula is

$$P(A|B) = \frac{P(A \bigcap B)}{P(B)} \tag{2.3}$$

Hypothesis Testing

Hypothesis Testing is the statistical assessment of the probability that a hypothesis is true. It contains several preparations.

1. The null hypothesis H_0 and the alternative hypothesis H_a must be stated
2. An analysis plan must be formulated to assess the null hypothesis.
3. A set of sample data must be collected and analyzed with respect to the analysis plan.
4. The results must be evaluated to determine whether H_0 or H_a is true.

Central Limit Theorem

The *Central Limit Theorem* [93] consists of three statements as follows.

1. As sample size increases, the sampling distribution of sample means approaches a normal distribution.

2. The mean of the sampling distribution of sample means equals the mean of the population from which the samples are drawn.

3. The variance of the sampling distribution of sample means is equal to the variance of the population from which the samples are drawn divided by the size of the samples.

Confidence Level

Confidence Level for a population parameter, such as *mean*, *variance*, is an interval with an associated probability p. p is generated from a random sample of an underlying population.

Mean Value Theorem

Given a section of a smooth curve, there should be a point on that section whose the derivative of the curve is equal to the "average" derivative of the section.

2.2 Decision Boundary

A decision boundary is '*a line determined by the weight and the bias vectors, for which the output is zero*' [15]. In [42], it gives a more generalized definition. Here, we extends and consolidates it as follows.

Definition 2.2.1. *Decision Boundary.*

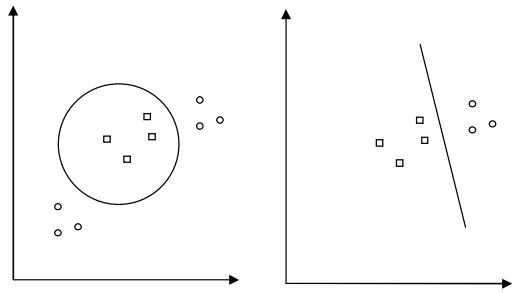


Figure 2.1: Boundary of two dichotomies

Let $\mathbf{Z}$ *be a set of points in a m-dimensional space. Suppose that there are* l *classes which are called as* $\gamma_1,...,\gamma_l$*. A decision boundary between each pair of classes is then represented as*

$$\{\mathbf{z}|h(\mathbf{z}) = \theta\}$$

where $\mathbf{z} \in \mathbf{Z}$ *is a set of points on the boundary and* θ *is the value of one decision boundary* $h(\mathbf{z})$*. Then, for the points on the two sides of the decision boundary, the following should be true:*

$$h(\mathbf{z}') > \theta \quad , \quad \mathbf{z}' \in \gamma_i$$
$$h(\mathbf{z}'') < \theta \quad , \quad \mathbf{z}'' \in \gamma_j$$

which means

$$(h(\mathbf{z}') - \theta)(h(\mathbf{z}'') - \theta) < 0$$

Note that, according to different dimensions, a decision boundary can be a point, line, hyper plane, curved surface or curved hyper-surface (see Figure 2.1).

2.3 Support Vector Machine

Support Vector Machine (SVM) is a method that determines the maximum-margin hyperplane. In geometry, a maximum-margin hyperplane is a hyperplane which separates two clouds of points, and the distances between the hyperplane and each cloud are maximum and equal. SVM algorithms are a set of related supervised learning methods in machine learning.

SVM has been proven as an efficient technique for classification and regression. The SVM algorithm is a nonlinear generalization of the Generalized Portrait Algorithm developed by Vapnik in the sixties [87]. It is a type of learning machines based on the statistical learning theory. It implements the *Structural Risk Minimization Inductive Principle* to obtain a good generalization from limited-size data sets as details below [85].

2.3.1 Theory of SVM

A learning machine takes a set of input-output and produces a model. However, some complex models drawn from this process might over-fit. Therefore, SVM uses *Structural Risk Minimization* to solve this problem.

Suppose a set of examples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ which are generated from an unknown distribution $P(\mathbf{x}, y)$. The number of the examples is n, and the dimension of the input space is m.

$$\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\} \qquad \mathbf{x}_i \in \mathbb{R}^m,\ y_i \in \{-1, 1\},\ 1 \leq i \leq n.$$

Given a set of decision functions,

$$\{f(\mathbf{x}, \alpha)\} \qquad \mathbf{x} \in \mathbb{R}^m, \alpha \in \Lambda$$

where Λ is a set of adjustable parameters and α is used to label the function $f(\mathbf{x}, \alpha)$. For a given input $\mathbf{x}$ and a parameter α^*, if the output is always the same, then $f(\mathbf{x}, \alpha^*)$ is deterministic. Hence, the function $f(\mathbf{x}, \alpha)$ generated upon the particular parameter α^* is called as 'trained machine' [13].

The aim of a learning machine is to find out a function $f(\mathbf{x}, \alpha)$ to correctly classify the unseen examples with the smallest *expected risk*, as defined below.

Definition 2.3.1. *Expected Risk (ER).*

$$ER(\alpha) = \int |f(\mathbf{x}, \alpha) - y| P(\mathbf{x}, y) d\mathbf{x} dy$$

In the above equation, $|f(\mathbf{x}, \alpha) - y|$ is the loss function. Therefore, the *expected risk* is used to evaluate the goodness of fit of a function $f(\mathbf{x}, \alpha)$ on predicting the correct label y for every point $\mathbf{x}$ drawn from $P(\mathbf{x}, y)$.

As a result of unknown $P(\mathbf{x}, y)$, the real value of *expected risk* cannot be computed. Hence, the *empirical risk* is put forward to approximate the *expected risk* via the samples of $P(\mathbf{x}, y)$.

Definition 2.3.2. *Empirical Risk (EM)*

$$EM(\alpha) = \frac{1}{n} \sum_{i=1}^{n} |f(\mathbf{x}_i, \alpha) - y_i|$$

The law of large numbers proves that the *empirical risk* will be close to the *expected risk*. The *empirical risk minimization* principle shows that if EM converges to ER, then the minimum of EM will converge to that of ER.

Definition 2.3.3. *Expected Risk Minimization (R)*

$$R(\alpha) = \arg\min(ER(\alpha))$$

Definition 2.3.4. *Empirical Risk Minimization (ERM)*

$$ERM(\alpha) = \arg\min(EM(\alpha))$$

Vapnik and Chervoneniks [86] [89] guarantee that the *empirical risk minimization* principle does hold if *uniform convergence* [96] in probability takes place and *empirical risk minimization* method is *strictly consistent* [88].

In other word, the *empirical risk* is the training error which is drawn from the examples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, while the *expected risk* is the testing error which is drawn from the testing points $\{\mathbf{x}'_1, ..., \mathbf{x}'_n\}$. The testing points are given to predict $\{y'_1, ..., y'_n\}$. The convergence between the training error and the testing error represents the generalization ability of the *empirical risk minimization* method.

Hence, in the context of the *empirical risk minimization* method, Vapnik [89] provides a definition of *VC dimension* which is able to indicate the bounds on the testing error.

Definition 2.3.5. *The VC dimension.*

Suppose that there are n examples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ in a two-class classification problem. Let h be the largest number of points which are correctly separated by functions $\{f(\mathbf{x}, \alpha)\}$. The VC dimension represents the maximum number of examples that can be shattered, which is denoted as h.

Suppose that there is a function $f(\mathbf{x}, \alpha)$, and a set of points $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$. $f(\mathbf{x}, \alpha)$ can shatter the points if and only if there exists an α^* such that f makes no error when this set of points are evaluated.

From the definition, it can be seen that *shattering* plays an important role in VC theory.

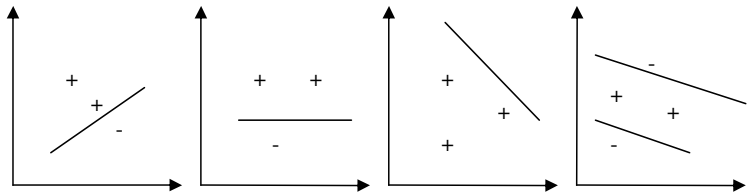


Figure 2.2: The example of VC dimension

Example. Given a function $f = w\mathbf{x} + b$ which represents a straight line, and the points shown in Figure 2.2 are inputs. From the figure, there is a line to separate three non-collinear points. However, it cannot ensure to shatter four points (for example, the last graph in Figure 2.2). Hence, the number of maximum points shattered by f in this example is 3. Therefore, the VC dimension h is 3 as well.

Hence, based on VC dimension, Vapnik [89] deduces that under probability $1 - \eta$:

$$R(\alpha) \leq ERM(\alpha) + \sqrt{\frac{h(\ln \frac{2n}{h} + 1) - \ln \frac{\eta}{4}}{n}} \tag{2.4}$$

where h is the VC dimension, $\alpha \in \Lambda$ and n is the number of examples.

Equation 2.4 presents that if $ERM(\alpha)$ decreases, the $R(\alpha)$ also decreases. It is especially efficient when the example size is large so that $R(\alpha)$ is mostly determined by $ERM(\alpha)$. However, Equation 2.4 does not ensure a small *expected risk* if both n/h and *empirical risk* are small. In 1980s, Vapnik proposed a technique named *Structural Risk Minimization* [86] to choose an appropriate VC dimension.

Definition 2.3.6. *Structural Risk Minimization (SRM).*

It is a method to minimize the expected risk over both empirical risk and complexity penalty.

$$Expected\ Risk \leq Empirical\ Risk + Complexity\ Penalty$$

The procedure is outlined below:

Suppose the set of functions $\{f(\mathbf{x}, \alpha_k)\}$ *can be divided into a hierarchy of nested subsets in the order of increasing complexity such as polynomials of increasing degree. A structure* S_k *refers to the set of possible decision boundaries obtained from* $f(\mathbf{x}, \alpha_k)$*.* $S_1, S_2, ..., S_k$ *have the following properties:*

1. $S_1 \subset S_2 \subset ...S_k$.

2. $S^* = \bigcup_k S_k$, S^* *is the aggregation of all the functions.*

3. *Each element of the structure* S_k *has an independent VC dimension* h_k*. With increasing of* k, h_k *is also increasing.*

$$h_1 \leq h_2 \leq .... \leq h_k$$

The complexity penalty ϵ *is defined in [88] as:*

$$\epsilon = \sqrt{\frac{h(\ln \frac{2n}{h} + 1) - \ln \frac{\eta}{4}}{n}}$$

Hence the fourth property of SRM is

4. *Each element of the structure* S_k *has an independent VC confidence* τ_k*, where* $\tau_k = \sqrt{\frac{h(\ln \frac{2n}{h}+1)-\ln \frac{\eta}{4}}{n}}$*. With increasing* k, τ_k *is increasing as well.*

$$\tau_1 \leq \tau_2 \leq .... \leq \tau_k$$

The definition above corresponds to Figure 2.3. It can be seen that the *empirical risk* decreases with the increase of h. While the complexity penalty increases with the increase of h. Therefore, the optimal model is able to be found at the point where the sum of *empirical risk* and VC confidence are the minimum.

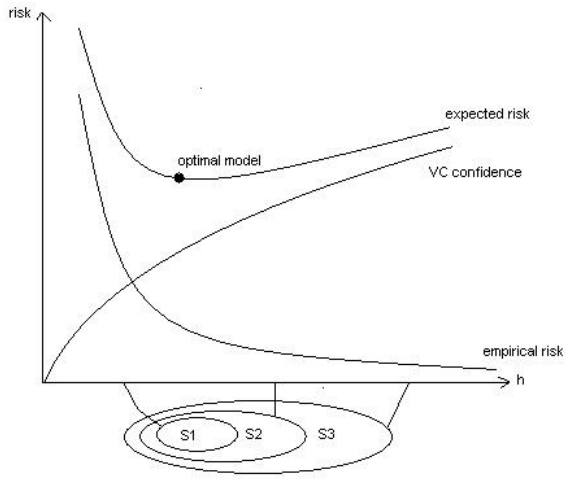


Figure 2.3: The bound of risk

SRM is an inductive principle to select the proper model from a finite data set. A model from an inductive learning process is often accompanied by the consequent problem of over-fitting[1]. SRM provides a trade-off between the model's complexity and the quality of fitting the training data which is called *empirical risk*.

SVM uses SRM for a better generalization performance. The next section will introduce the detail of SVM.

2.3.2 Support Vector Classification

In this section, we describe the Support Vector Machine (SVM) approach. We start from the simplest case: the linear separable problem in which the training data is linearly separable. Then the linear nonseparable problem is introduced. This problem deals with the training data which are unable to be cleanly separated by the linear hyperplane. It allows a few data to fall on the wrong side of the separating hyperplane. Finally, the general case nonlinear SVM on nonseparable

[1]A model is too complex to fit the noise.

data is analyzed.

Linear separable problem

Suppose that there is a set of training examples:

$$\{(\mathbf{x}_i, y_i)\}, \qquad y_i \in \{+1, -1\}, \mathbf{x}_i \in \mathbb{R}^m, i = 1, ..., n.$$

where n is the number of training examples, m is the dimension of the inputs and y_i is either $+1$ or -1.

Equation 2.5 is the function of the hyperplane separating the positive examples from the negative examples.

$$w \cdot \mathbf{x} + b = 0 \tag{2.5}$$

where w is a weight vector and b is a bias. The goal of a SVM network in a classification problem is to find an optimal hyperplane, to which the distance from the closest points in each class is maximum. This distance is called ρ. Suppose that all the training examples satisfy the following constraints:

$$\begin{aligned} w \cdot \mathbf{x}_i + b &\geq +1, \qquad \textbf{for} \quad y_i = +1 \\ w \cdot \mathbf{x}_i + b &\leq -1, \qquad \textbf{for} \quad y_i = -1 \end{aligned} \tag{2.6}$$

If there are two points $\mathbf{x}_1$ and $\mathbf{x}_2$ (see Figure 2.4) lying on the hyperplanes $h_1 : w \cdot \mathbf{x}_i + b \geq +1$ and $h_2 : w \cdot \mathbf{x}_i + b \leq -1$, then the perpendicular distances from the origin to these two points are $|1 - b| / w^T w$ and $|-1 - b| / w^T w$. As the perpendicular distance from the hyperplane $w \cdot \mathbf{x}_i + b = 0$ to the origin is $|b| / w^T w$, the ρ is simply $2/w^T w$. Note that in the separable case, no training point falls between h_1 and h_2.

In order to maximize the margin of ρ, it is equivalent to minimize $w^T w$. Therefore,

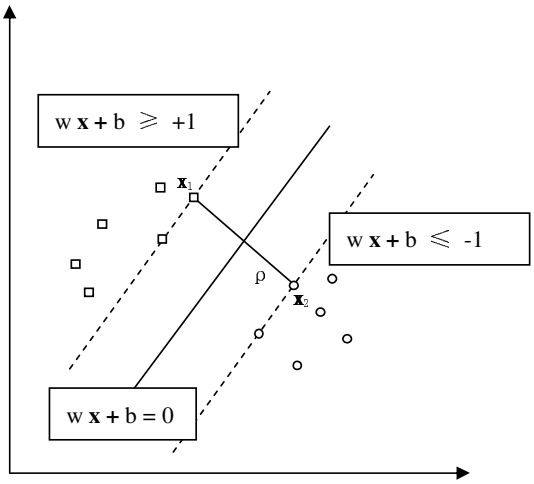


Figure 2.4: Two-class linear separable problem

the optimization problem changes to:

$$\begin{aligned} Minimize \quad & \frac{1}{2}w^T w \\ s.t. \quad & y_i(w \cdot \mathbf{x}_i + b) \geq 1 \end{aligned} \tag{2.7}$$

Lagrangian J has been used to solve this optimal problem which is:

$$J = \frac{1}{2}w^T w - \sum_{i=1}^{n} \alpha_i (y_i(w^T \mathbf{x}_i + b) - 1) \tag{2.8}$$

where the Lagrange multipliers $\alpha_i \geq 0$.

From Equation 2.8, with respect to the primal variables w and b, the Lagrange J can be minimized. The following two optimal conditions by partial derivative are obtained:

$$\frac{\partial J}{\partial w} = 0 \tag{2.9}$$

$$\frac{\partial J}{\partial b} = 0 \tag{2.10}$$

which lead to

$$\begin{aligned} w &= \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\ \sum_{i=1}^{n} \alpha_i y_i &= 0 \end{aligned} \tag{2.11}$$

Equation 2.8 is the primal problem of Lagrangian J. By replacing Equation 2.11 into Equation 2.8, it changes to a so-called dual (quadratic) problem which is usually solvable in practice. If the primal problem has an optimal solution, the dual problem also has an optimal solution [27]. The dual problem becomes:

$$\begin{aligned} Maximize \qquad & Q = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \\ s.t. \qquad & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{2.12}$$

With respect to α_i, Q will be maximized. Hence, the optimal conditions would be obtained under partial derivative.

$$\frac{\partial Q}{\partial \alpha_i} = 0 \tag{2.13}$$

The reason for transforming the primal problem to dual problem is that there are many inequality constraints in the primal problem. Comparing Equation 2.12 with Equation 2.8 and 2.7, fewer variables are found in the quadratic function $Q(\alpha_i, \mathbf{x})$ than $J(w, b, \alpha_i, \mathbf{x})$, so the constraints in the dual form are greatly simplified.

Furthermore, Vapnik [89] shows that the optimal solutions such as w, b should follow *Karush-Kuhn-Tucker conditions*.

Definition 2.3.7. *Karush-Kuhn-Tucker Theorem (KKT). If there exists a solution to minimize (maximize) an objective function $f(x)$ under some inequality constraints $\{g_k(x) \leq 0\}_{k=0}^{m}$, the following several conditions should hold:*

1. $\nabla f(x) - \sum_{k=1}^{m} \lambda_k \nabla g_k(x) = 0, \quad for \ k = 0, 1, ..., m$

2. $\{g_k(x) \leq 0\}_{k=0}^{m}$

3. $\lambda_k \geq 0, \quad for \ k = 0, 1, ..., m$

4. $\sum_{k=1}^{m} \lambda_k g_k(x) = 0, \quad for \ k = 0, 1, ..., m$

Hence, for the primal problem in Equation 2.8, the KKT conditions are

$$\begin{aligned} &\frac{\partial J}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \\ &\frac{\partial J}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \\ &y_i(\mathbf{x}_i \cdot w + b) - 1 \geq 0 \\ &\alpha_i(y_i(w \cdot \mathbf{x}_i + b) - 1) = 0 \\ &\alpha_i \geq 0 \end{aligned} \tag{2.14}$$

In next section, the discussion will focus on the linear nonseparable problem. In most real-life cases, it is not possible to separate the training data without encountering classification errors. Hence, for the linear nonseparable problem, SVM aims to find an optimal linear hyperplane by minimizing the probability of classification errors.

Linear nonseparable problem

Suppose a set of linear nonseparable training examples are given. Hence, there should be some examples in this training set violating the constraints of Equation 2.7. To deal with this case, slack variables $\{\xi_i\}_{i=1}^{n}$ [89] have been introduced, which take the violations of the constraints in Equation 2.7 into account. Figure 2.5

depicts a linear nonseparable case, where points $\mathbf{x}_3$ and $\mathbf{x}_4$ are two examples misclassified by the hyperplane with nonzero values ξ_3 and ξ_4.

Therefore, Boser et al. [7] transform the optimization problem as follows:

$$\begin{aligned} Minimize \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(w \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{2.15}$$

where C is a user-defined parameter.

The Lagrangian J is defined as the usual manner:

$$J = \frac{1}{2}w^T w + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i[1 - \xi_i - y_i(\mathbf{x}_i^T w + b)] - \sum_{i=1}^{n}\mu_i\xi_i$$

where μ_i are the Lagrange multipliers used to enforce the nonnegativity of the slack variables ξ_i.

KKT conditions in Equation 2.16 are verified to satisfy this optimization problem.

$$\begin{aligned} & \frac{\partial J}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i = 0 \\ & \frac{\partial J}{\partial b} = \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & \frac{\partial J}{\partial \xi_i} = C - \mu_i - \alpha_i = 0 \\ & y_i(\mathbf{x}_i \cdot w + b) - 1 + \xi_i \geq 0 \\ & \alpha_i[y_i(w \cdot \mathbf{x}_i + b) - 1 + \xi_i] = 0 \\ & \mu_i\xi_i = 0 \\ & \alpha_i \geq 0, \mu_i \geq 0, \xi_i = 0 \end{aligned} \tag{2.16}$$

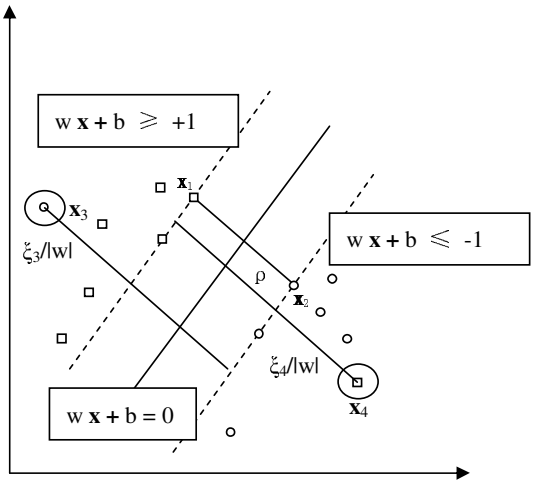


Figure 2.5: Two-class linear nonseparable problem

The following is the dual form of this problem.

$$\begin{aligned} Maximize \qquad & Q = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \\ s.t. \qquad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{aligned} \tag{2.17}$$

The optimal solution for w is

$$w = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{2.18}$$

The solution of non-separable problem is analogous to the separable problem. The only difference is that the α_i have an upper bound C. Furthermore, the introduction of those slack variables ensures that SVM can control the hyperplane violation and achieve the largest ρ simultaneously.

Nonlinear problem

In real world, many examples are not linearly separable in the input space. Figure 2.6(a) indicates an example of the nonlinear problem where the line used to classify the points which belong to two classes is not straight.

The strength of SVM is that it can also separate the nonlinear data by mapping them into a high dimensional feature space H. SVM is still looking for the linear hyperplane but under the feature space, rather than the input space (see Figure 2.6(b)). The mapping function is

$$\Phi : \mathbb{R}^d \rightarrow H \tag{2.19}$$

Let $\{\Phi_j(\mathbf{x})\}_{j=1}^{d}$ be a set of transformations on $\mathbf{x}$ from the input space to the feature space, where d denotes the dimension of the feature space. Hence, the hyperplane in the feature space could be defined as:

$$\sum_{j=1}^{d} w_j \Phi_j(\mathbf{x}) + b = 0$$

The above equation can be simplified to:

$$\sum_{j=0}^{d} w_j \Phi_j(\mathbf{x}) = 0$$

where $\Phi_0(\mathbf{x}) = 1$ so that $w_0 = b$.

Suppose a vector $\Phi(\mathbf{x})$ is defined as:

$$\Phi(\mathbf{x}) = [\Phi_0(\mathbf{x}), \Phi_1(\mathbf{x}), ..., \Phi_d(\mathbf{x})]^T$$

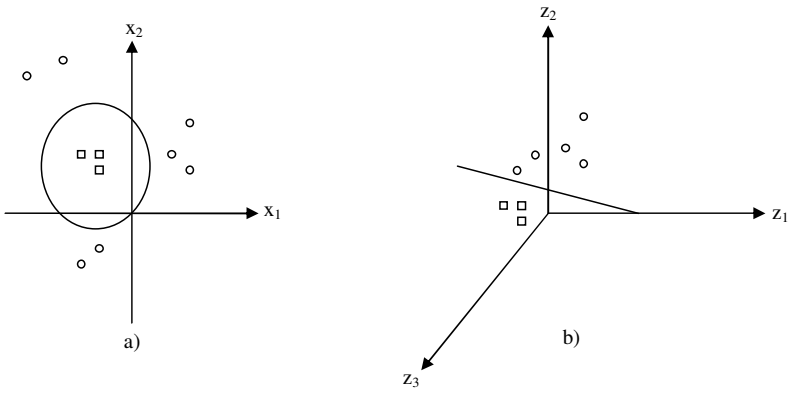


Figure 2.6: Mapping from original to feature space

This leads to:

$$w^T \Phi(\mathbf{x}) = 0 \tag{2.20}$$

By adapting Equation 2.11 for a feature space, the following may be obtained:

$$w = \sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i) \tag{2.21}$$

where $\Phi(\mathbf{x}_i)$ denotes the feature vector of the i^{th} example in the input space, and there are n examples in the input space.

Therefore, Equation 2.20 can be transformed to:

$$\sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i) \Phi(\mathbf{x}) = 0 \tag{2.22}$$

Figure 2.6 illustrates an example that a linear separable hyperplane can be found, after the data are mapped into high dimensions (i.e. in the figure, it is transformed from 2-D into 3-D). Although the linear optimal hyperplane can be constructed in high dimensional space, how to treat this complicated feature space is a NP-hard problem. The dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$ in Equation 2.22 induces the concept of

Kernel [89] so that one does not need to consider the feature space.

Definition 2.3.8. *Kernel (K).*

$$K(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) = \sum_{j=1}^{d} \Phi_j(\mathbf{x}_i)\Phi_j(\mathbf{x}) \tag{2.23}$$

From the equation above, it is interesting to see that the kernel provides a kind of mechanism to construct a linear hyperplane without considering what the feature space is. Hence, by substituting Equation 2.23 into Equation 2.22, the optimal hyperplane then becomes

$$\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) = 0 \tag{2.24}$$

Example. Given $\mathbf{x} = [x_1, x_2]$, $\mathbf{x}_i = \mathbf{z} = [z_1, z_2]$, $\Phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]$ and Kernel function $K(\mathbf{x}_i, \mathbf{x}) = K(\mathbf{z}, \mathbf{x}) = (\mathbf{z} \cdot \mathbf{x})^2 = z_1^2x_1^2 + z_2^2x_2^2 + 2z_1z_2x_1x_2$ then,

$$\begin{aligned} \Phi(\mathbf{z}) \cdot \Phi(\mathbf{x}) &= (z_1^2, z_2^2, \sqrt{2}z_1z_2) \cdot (x_1^2, x_2^2, \sqrt{2}x_1x_2) \\ &= z_1^2x_1^2 + z_2^2x_2^2 + 2z_1z_2x_1x_2 \\ &= (\mathbf{z} \cdot \mathbf{x})^2 = K(\mathbf{z}, \mathbf{x}) \end{aligned}$$

Hereby, the dual form of a nonlinear SVM could be stated as follows [32]:

> *Given the training sample* $\{(\mathbf{x}_i, y_i)_{i=1}^n\}$*, find the Lagrange multipliers* $\{\alpha_i\}_{i=1}^n$ *that maximize the objective function*
>
> $$Q = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$
>
> *subject to the constraints:*
>
> 1) $\sum_{i=1}^{n} \alpha_i y_i = 0$
>
> 2) $0 \leq \alpha_i \leq C$

A kernel $K(\mathbf{x}_i, \mathbf{x})$ which associates with a feature space should satisfy *Mercer theorem* [89]:

$$K(\mathbf{x}_i, \mathbf{x}) = \int \int K(\mathbf{x}_i, \mathbf{x})\Phi(\mathbf{x}_i)\Phi(\mathbf{x})d\mathbf{x}_i d\mathbf{x} \geq 0 \tag{2.25}$$

The commonly used kernels are listed in Table 2.1:

linear	$K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x} \cdot \mathbf{x}_i$
Polynomial	$K(\mathbf{x}, \mathbf{x}_i) = (1 + \mathbf{x}^T \cdot \mathbf{x}_i)^d$
Gaussian	$K(\mathbf{x}, \mathbf{x}_i) = \exp(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2})$
Sigmoid	$K(\mathbf{x}, \mathbf{x}_i) = \tanh(k(\mathbf{x}, \mathbf{x}_i) + \upsilon)$

Table 2.1: Examples of Kernels

XOR problem

XOR problem [78] is to find the hyperplane which can separate the examples in Table 2.2. The kernel is shown as follows.

$$K(\mathbf{x}_i, \mathbf{x}) = (1 + \mathbf{x}^T\mathbf{x}_i)^2 \tag{2.26}$$

Input x	Output y
$(-1, -1)$	-1
$(-1, +1)$	$+1$
$(+1, -1)$	$+1$
$(+1, +1)$	-1

Table 2.2: XOR Problem

We find that

$$K = \begin{pmatrix} (k_{1j}) \\ (k_{2j}) \\ (k_{3j}) \\ (k_{4j}) \end{pmatrix}_{j=1}^{4} \tag{2.27}$$

Let $i = 1$, $x_1 = (-1, -1)$. Then

$$\begin{aligned}(K_{1i}) &= (K(\mathbf{x}_1, \mathbf{x}_1), K(\mathbf{x}_1, \mathbf{x}_2), K(\mathbf{x}_1, \mathbf{x}_3), K(\mathbf{x}_1, \mathbf{x}_4)) \\ &= ((1+2)^2, (1+0)^2, (1+0)^2, (1+0)^2).\end{aligned}$$

Finally,

$$K = \begin{pmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{pmatrix} \tag{2.28}$$

Then, the dual form is obtained as follows.

$$Q = \alpha_1+\alpha_2+\alpha_3+\alpha_4-\frac{1}{2}(9\alpha_1^2-2\alpha_1\alpha_2-2\alpha_1\alpha_3+2\alpha_1\alpha_4+9\alpha_2^2+2\alpha_2\alpha_3-2\alpha_2\alpha_4+9\alpha_3^2-2\alpha_3\alpha_4+9\alpha_4^2) \tag{2.29}$$

According to Equation 2.13, the optimal functions with respect to the Lagrange multipliers via partial derivative are therefore:

$$\begin{aligned}\frac{\partial Q}{\partial \alpha_1} &= 1 - (+9\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4) = 0 \\ \frac{\partial Q}{\partial \alpha_2} &= 1 - (-\alpha_1 + 9\alpha_2 + \alpha_3 - \alpha_4) = 0 \\ \frac{\partial Q}{\partial \alpha_3} &= 1 - (-\alpha_1 + \alpha_2 + 9\alpha_3 - \alpha_4) = 0 \\ \frac{\partial Q}{\partial \alpha_4} &= 1 - (+\alpha_1 - \alpha_2 - \alpha_3 + 9\alpha_4) = 0\end{aligned}$$

Hence, by solving above equations, the optimal value for each α_i equals $\frac{1}{8}$. By replacing the value of α_i in Equation 2.29, Q equals $\frac{1}{4}$.

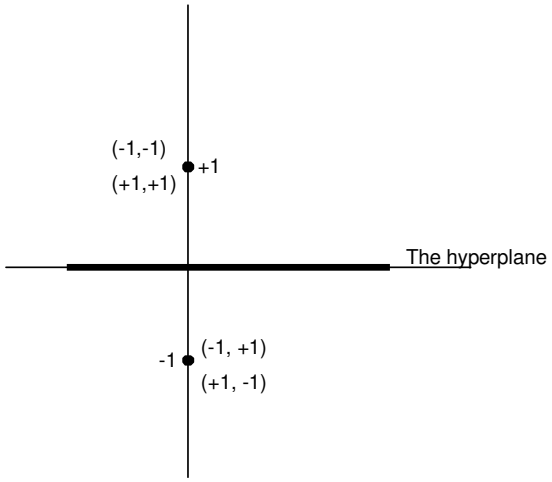


Figure 2.7: The hyperplane of XOR problem in feature space.

Correspondingly, by extending Equation 2.24 in terms of this example, the following formula can be obtained:

$$\begin{aligned}
\sum_{i=1}^{4} \alpha_i y_i K(1+\mathbf{x}^T\mathbf{x}_i)^2 &= \alpha_1 y_1 K(1+\mathbf{x}^T\mathbf{x}_1)^2 + \alpha_2 y_2 K(1+\mathbf{x}^T\mathbf{x}_2)^2 + \alpha_3 y_3 K(1+\mathbf{x}^T\mathbf{x}_3)^2 \\
&+ \alpha_4 y_4 K(1+\mathbf{x}^T\mathbf{x}_4)^2 \\
&= \frac{1}{8}[(-1)(1-x_1-x_2)^2 + (+1)(1-x_1+x_2)^2 + (+1)(1+x_1-x_2)^2 \\
&+ (-1)(1+x_1+x_2)^2] \\
&= -8x_1x_2 \\
&= 0
\end{aligned}$$

where $\mathbf{x}$ (denoted as $[x_1, x_2]$) is a two-dimensional vector in XOR problem.

Therefore, the optimal hyperplane becomes (see Figure 2.7)

$$-x_1x_2 = 0$$

From above function, it is clear that the examples $\mathbf{x}_1$ to $\mathbf{x}_4$ tell us more about this problem than the classification network, the calculations of Q and the kernel given

in this problem.

For this typical XOR problem, at first, only the inputs, the outputs and the corresponding kernel are known. However, it is still difficult to understand why the inputs provided in Table 2.2 could lead to such outputs. Because of the optimal hyperplane function, it implicitly explains the relationship between the inputs and the outputs, which is much more useful for our understanding. The circumstance in XOR problem appears a common difficulty for many cases. Thus, having a rule extraction which is to find a type of the symbolic expressions to articulate the relationship between the inputs and the outputs of learning models is highly desirable.

2.3.3 Support Vector Machine Algorithms

There are various SVM algorithms. In this section, three of the most common used algorithms are described.

Sequential Minimal Optimization (SMO)

SMO is a fast way to train SVMs. The large quadratic programming (QP) optimization problem in SVM (Equation 2.17) is broken into a set of small QP problems. Let the QP problem used to train an SVM be

$$\begin{aligned} Maximize \qquad & Q = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ s.t. \qquad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{aligned} \tag{2.30}$$

Unlike other methods, SMO has three components: a small QP problem for solving

two multipliers α_1 and α_2, a heuristic to select which multipliers to optimize and a method to compute b. For solving two Lagrange multipliers, SMO starts computing the constraints on α_1 and α_2, and then works out the constrained maximum. Depending on the inequality and equality constraints in Equation 2.30, the relation between α_1 and α_2 is shown in Figure 2.8.

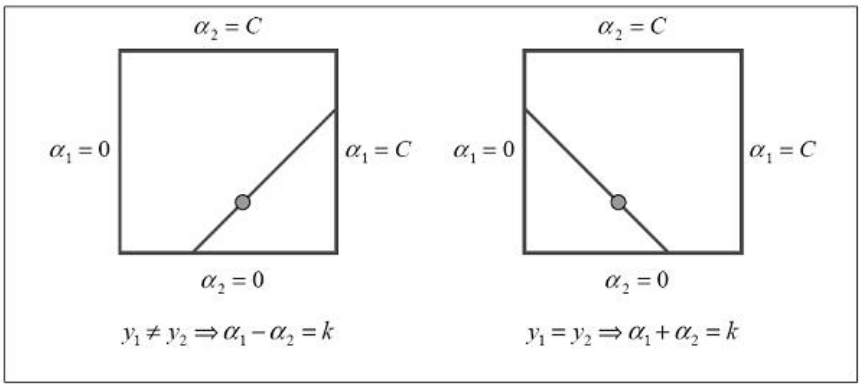


Figure 2.8: Relations of two Lagrange multipliers α_1 and α_2 [40]

The inequality constraint $0 \leq \alpha_i \leq C$ makes α_1 and α_2 lie in a box while the equality constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$,$n = 2$, causes the multipliers to lie on a diagonal line in the box. Therefore, the small QP problem is to find an optimal solution on this diagonal line. The algorithm first computes the value of α_2. As shown by Figure 2.8,

if $y_1 \neq y_2$, the lower bound L and the upper bound H of α_2 are:

$$L = max(0, \alpha_2 - \alpha_1), \quad H = min(C, C + \alpha_2 - \alpha_1) \tag{2.31}$$

and if $y_1 = y_2$, the lower bound L and the upper bound H of α_2 are:

$$L = max(0, \alpha_2 + \alpha_1 - C), \quad H = min(C, \alpha_2 - \alpha_1) \tag{2.32}$$

Thus, by replacing α_1 with α_2, the small objective function could change to a form with only one multiplier α_2. While the second derivative of the objective function in Equation 2.30 can be:

$$\eta = \frac{d^2Q}{d\alpha_2^2} = 2K(\mathbf{x}_1, \mathbf{x}_2) - K(\mathbf{x}_1, \mathbf{x}_1) - K(\mathbf{x}_2, \mathbf{x}_2)$$

Normally, there is a maximum value for α_2 along the diagonal line. Hence, the second derivative η should be less than zero according to the derivative calculus. The optimal value α_2^* then becomes

$$\alpha_2^* = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \tag{2.33}$$

where $E_i = u_i - y_i$, E_i is the error on the i^{th} training example, u_i is the output of SVM and y_i is the target output. In order to meet the inequality constraint, the final optimal value of α_2 is:

$$\alpha_2^{**} = \left\{ \begin{array}{lll} H & if & \alpha_2^* > H \\ \alpha_2^* & if & L \leq \alpha_2^* \leq H \\ L & if & \alpha_2^* < L \end{array} \right\} \tag{2.34}$$

Next, α_1 is computed:

$$\alpha_1^* = \alpha_1 + y_1 y_2(\alpha_2 - \alpha_2^{**}) \tag{2.35}$$

SMO uses heuristics to select two multipliers. It repeatedly loops the training examples to find those falling to agree with the KKT. These violating examples are chosen for optimization. The iteration stops when all the examples satisfy the KKT. Due to these characteristics, SMO can perform in a linear complexity. Research shows that SMO is 1.7 times faster than a standard projected conjugate

gradient chunking algorithm on MNIST database[2] [40].

Directed Acyclic Graphs Support Vector Machines (DAGs-SVM)

DAGs-SVM is used to combine many binary-class classifiers into a multi-class classifier. DAGs-SVM constructs $\frac{k(k-1)}{2}$ classifiers for a k-class classification problem, one classifier for a pair of classes. From [39], it can be found that DAGs-SVM is based on Rooted Binary Directed Acyclic Graph (Rooted Binary DAG). Given a sequence of data $\mathbf{X}$ and a set of binary functions $F = \{f : \mathbf{X} \to \{0, 1\}\}$, the class decision on k classes can be implemented by using a Rooted Binary DAG with k leaves. The algorithm puts labels on each leave. Inside Rooted Binary DAG, there are $\frac{k(k-1)}{2}$ internal nodes labelled with an element of F. The whole graph has one root node at the top, two nodes in the second layer and k leaves at the final layer. Figure 4.3 shows a typical example of DAG.

Suppose an input $\mathbf{x} \in \mathbf{X}$ starts at the root node. Next, it goes to the left edge if $f(\mathbf{x}) = 0$; or the right edge, if $f(\mathbf{x}) = 1$. Then the value of f at the next node is evaluated till reaching the final leaf node in Figure 2.9. At the leaf node, the class that $\mathbf{x}$ belongs to can be decided. [39] indicates that DAGs-SVM algorithm is superior to other multi-class SVM algorithms in complexity.

Least Squares Support Vector Machines

Least Square Support Vector Machine (LSSVM) is a least square version for SVM classifiers. It solves a set of linear equations, instead of quadratic programming that classical SVM uses.

[2]The MNIST database of Handwritten upper-case letters is a subset derived from NIST Special Database 19 Handprinted Forms and Characters Database.

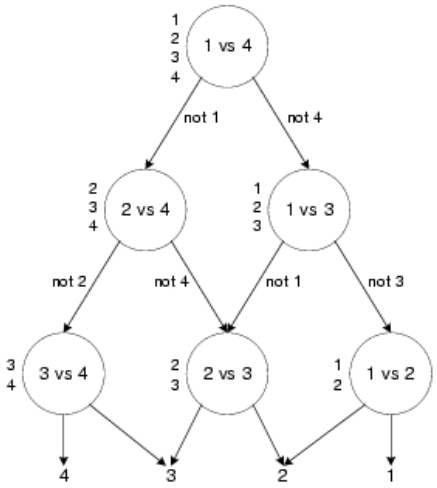


Figure 2.9: Using Rooted Binary DAG to decide the best class within four classes [39].

LSSVM formulates SVM classification problem as:

$$\begin{aligned} Mininimize \quad & \frac{1}{2}w^T w + C\frac{1}{2}\sum_{i=1}^{n}\xi_i^2 \\ s.t. \quad & y_i(w^T\Phi(\mathbf{x}_i) + b) = 1 - \xi_i, i = 1, ..., n. \end{aligned} \tag{2.36}$$

The Lagrangian J is identified as:

$$J = \frac{1}{2}w^T w + C\frac{1}{2}\sum_{i=1}^{n}\xi_i^2 - \sum_{i=1}^{n}\alpha_i y_i[w^T\Phi(\mathbf{x}_i + b)] - 1 + \xi_i \tag{2.37}$$

where α_i are Lagrange multipliers.

The conditions for optimality then become

$$\begin{aligned}
&\frac{\partial J}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i) \\
&\frac{\partial J}{\partial b} = 0 \rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \\
&\frac{\partial J}{\partial \xi_i} = 0 \rightarrow \alpha_i = C\xi_i, i = 1, ..., n \\
&\frac{\partial J}{\partial \alpha_i} = 0 \rightarrow y_i[w^T \Phi(\mathbf{x}_i) + b] - 1 + \xi_i = 0, i = 1, ..., n
\end{aligned} \tag{2.38}$$

Equation 2.38 can be written as the solution to the following set of linear equations [45]

$$\left[\begin{array}{ccc|c} I & 0 & 0 & -Z^T \\ 0 & 0 & 0 & -Y^T \\ 0 & 0 & C^I & -I \\ \hline Z & Y & I & 0 \end{array}\right] \left[\begin{array}{c} w \\ b \\ \xi \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ \hline \overrightarrow{1} \end{array}\right]$$

where $Z = [\Phi(\mathbf{x}_1^T)y_1; ...; \Phi(\mathbf{x}_n^T)y_n]$, I is an identity vector, $Y = [y_1; ...; y_n]$, $\vec{1} = [1; ...; 1]$, $\xi = [\xi_1; ...; \xi_n]$ and $\alpha = [\alpha_1; ...; \alpha_n]$. The solution is also given as

$$\left[\begin{array}{c|c} 0 & -Y^T \\ \hline Y & ZZ^T + C^{-1}I \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \vec{1} \end{array}\right]$$

Mercer's condition can also be applied to ZZ^T where

$$\begin{aligned}
ZZ^T &= y_i y_j \Phi(\mathbf{x}_i^T)\Phi(\mathbf{x}_j) \\
&= y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), 1 \leq i, j \leq n
\end{aligned}$$

Hence the solution for the quadratic programming is replaced by the linear set of equations.

2.4 Clustering

Clustering is to put the data with similarities into groups [62]. It is used in this thesis as a prerequisite to automatically find the decision boundary. There are various references/algorithms regarding clustering in data mining, statistics, and machine learning. Generally, the algorithms are classified into two main categories: hierarchical methods and partitioning methods. Hierarchical clustering algorithms can be further subdivided into two types: agglomerative and divisive. The agglomerative clustering starts with a single point and recursively merges into appropriate clusters, while the divisive clustering iteratively splits the data group into clusters. SLINK [74], COBWEB [24], and Chameleon [23] are several basic algorithms for hierarchical clustering which were put forward between 1970s and 1990s. The other major clustering type, partitioning methods can also be further categorized into probabilistic clustering, K-Medoids methods, and density-based algorithms [61] [36] [28].

The next section gives an introduction on hierarchical clustering which is employed in our approach.

2.4.1 Hierarchical Clustering

Hierarchical clustering has agglomerative methods and divisive methods, where agglomerative techniques are more commonly used.

An agglomerative hierarchical clustering first considers each data as a cluster. It is an iterative procedure. At each particular stage, the agglomerative method joins two closest clusters together. There are different ways to compute the distance

between two clusters. These methods are called linkage functions.

$$\begin{aligned} Single\ linkage : Dist(C_1, C_2) &= min\{dist(p, q)\} \\ Complete\ linkage : Dist(C_1, C_2) &= max\{dist(p, q)\} \\ Average\ linkage : Dist(C_1, C_2) &= \frac{1}{\#(C_1) \cdot \#(C_2)} \sum\sum (dist(p, q)) \end{aligned}$$

where $p \in cluster\ C_1$, $q \in cluster\ C_2$, and $\#(C_1)$, $\#(C_2)$ is the number of data in each cluster.

In the single linkage function, the distances between each pair (p, q) are computed, and the minimum value of (p, q) refers to the distance between C_1 and C_2. Analogously, for the complete linkage, the distance between clusters C_1 and C_2 is given by the maximum distance between p and q. While, for the average linkage, the average distances between all pairs (p, q) stands for the distance between two clusters.

The following are the most commonly used distance functions $dist(p, q)$.

$$\begin{aligned} Manhatton\ distance :\ dist(p, q) &= |p - q| \\ Euclidean\ distance :\ dist(p, q) &= \sqrt{(p - q)^2} \end{aligned}$$

The agglomerative hierarchical clustering can be viewed as a hierarchical cluster tree (see Figure 2.10).

Example. Suppose that a set of inputs $\mathbf{X} = \{[1, 2]; [1, 3]; [4, 3]; [3, 1]\}$ is given, and a single linkage with an Euclidean distance function is chosen. Initially, each input is considered as a cluster. Table 2.3 shows the distance between each pair of inputs. Obviously, **X1** and **X2** are two clusters with the closest distance. After merging **X1** and **X2**, a set of new clusters, $C_1 = \{\mathbf{X1}, \mathbf{X2}\}$ and $C_2 = \{\mathbf{X3}\}$ and $C_3 = \{\mathbf{X4}\}$, is constructed. Then the clustering algorithm continues to compute the distances between these clusters. It is an iteration process until the distance

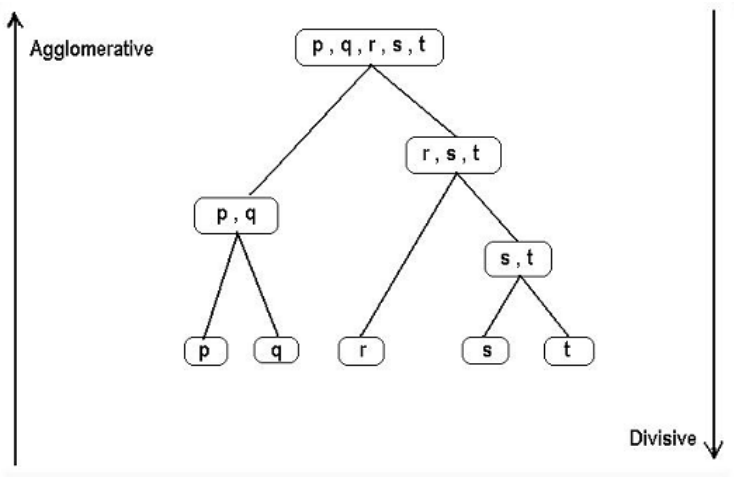


Figure 2.10: Hierarchical Clustering

between each pair of clusters is larger than a predetermined threshold.

Dist	X1	X2	X3	X4
X1	0	1	3.16	2.23
X2	1	0	3	2.83
X3	3.16	3	0	2.24
X4	2.23	2.83	2.24	0

Table 2.3: Distances between pairs of the inputs

2.5 Monte Carlo and Quasi Monte Carlo Method

2.5.1 Monte Carlo Method

Monte Carlo method used in GOSE is to approximate a nonlinear integral equation which is a constraint of a minimum optimization problem. It is a widely used class of computational algorithms. The basic process involves using the independent random numbers to perform the simulation and provide approximate solutions to a variety of mathematical problems.

The most common application of the Monte Carlo method is Monte Carlo integration. The following section presents one of the fundamental algorithms.

2.5.2 Crude Monte Carlo Method

Given a m-dimensional definite integral G in the form of,

$$G = \int_{[a,b]} g(\mathbf{x}) d\mathbf{x}, \qquad [a,b] = [a_1,b_1] \times [a_2,b_2] \times ... \times [a_s,b_s]$$

where $\mathbf{x} = [x_1, ..., x_s]$ and $a_i \leq x_i \leq b_i$, $1 \leq i \leq s$.

Suppose the Monte Carlo algorithm select N samples (denoted by $\mathbf{x}_1, ..., \mathbf{x}_N$) uniformly from the integration region. Then the estimate of the integral becomes

$$\frac{\prod_{i=1}^{s}(b_i - a_i)}{N} \sum_{i=1}^{N} g(\mathbf{x}_i) \approx G \tag{2.39}$$

Let

$$\mu = \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{x}_i)$$

Then the variance of those N samples can be estimated by using

$$\mathbf{s}^2 = \frac{1}{N-1} \sum_{i=1}^{N} (g(\mathbf{x}_i) - \mu)^2$$

According to *Central Limit Theorem*, the variance of the estimate of G is thus

$$\sigma^2 = \frac{\mathbf{s}^2}{N} \tag{2.40}$$

Equation 2.40 demonstrates that the variance σ decreases with the increase of N. For large N, it ensures that μ of a set of independent samples converges towards the integral.

N	μ	$\mathbf{s}$	σ
10^2	1.7163	0.4760	0.0476
10^3	1.7196	0.4864	0.0154
10^4	1.7224	0.4953	0.0050
10^5	1.7194	0.4933	0.0016
10^6	1.7186	0.4918	0.00049
10^7	1.7182	0.4919	0.00015

Table 2.4: The mean, variance and estimated variance on $e^{\mathbf{x}}$ under different number of N

Example. Given a function $g(\mathbf{x}) = e^{\mathbf{x}}$, where $0 \leq \mathbf{x} \leq 1$ and the dimension of $\mathbf{x}$ is one, the integral of $g(\mathbf{x})$ then equals $\int_0^1 e^{\mathbf{x}} = 1.7183$.

To calculate the integral, we also can generate N samples in $[0, 1)$, and approximate $\int_0^1 g(\mathbf{x})$ with $\frac{\sum_{i=1}^{N} g(\mathbf{x})}{N}$. Table 2.4 shows the approximate values of the integral of $g(\mathbf{x})$ for different N.

The estimate variance of crude Monte Carlo is important because it tells that σ decreases with the increase of the sample size rather than the dimensionality of samples.

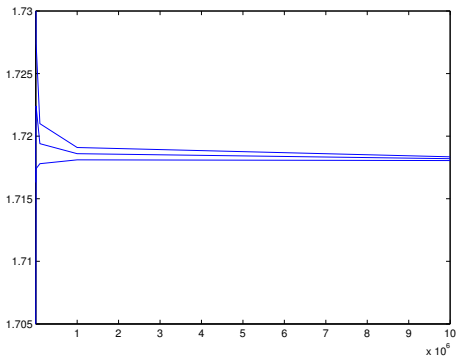


Figure 2.11: The mean of different N samples converges to the integral.

2.5.3 Quasi-Monte Carlo

Quasi-Monte Carlo is a method for the computation of an integral based on low-discrepancy sequences [30]. It approximates the integral in a similar way to Monte

Carlo which is

$$\int_{[a,b]} g(\mathbf{x}) \approx \frac{\prod_{i=1}^{s}(b_i - a_i)}{N} \sum_{i=1}^{N} g(\mathbf{x}_i)$$

where $\mathbf{x}_1, ..., \mathbf{x}_N$ is a low-discrepancy sequence.

The discrepancy of a low-discrepancy sequence $\mathbf{x}_1, ..., \mathbf{x}_N$ can be defined as follows.

Let Q be a cube of I^s, I is a unit cube which equals $[0, 1)$, and $m(Q)$ be the volume of Q. The discrepancy is

$$D_N = \sup_{Q \in I^s} \left| \frac{\sharp Q}{N} - m(Q) \right| \tag{2.41}$$

where $\sharp Q$ is the number of points in Q.

The discrepancy of the sequence is then bounded by a constant $\frac{(\log N)^s}{N}$.

One of the most important error bounds for quasi-Monte Carlo is the approximation error. The *Koksma-Hlawka inequality* [30] shows that the approximation error is bounded by two things: the function $g(\mathbf{x})$ and the discrepancy D_N.

Let $g(\mathbf{x})$ have a bounded variation $V_g(I)$ on I^s. Then for any $\mathbf{x}_1, ..., \mathbf{x}_N \in I^s$.

$$\left| \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{x}_i) - \int_0^1 g(\mathbf{x}) d\mathbf{x} \right| \leq V_g(I) D_N \tag{2.42}$$

where the bounded variation $V_g(I)$ refers to a real-valued functions whose total variation is bounded. The total variation is defined to be:

$$V_g(I) = \sup \left\{ \sum_{i=1}^{n} |g(\mathbf{x}_{i+1}) - g(\mathbf{x}_i)| \middle| P = \{\mathbf{x}_0, ..., \mathbf{x}_n\} \textit{ is a partition of } I \right\}$$

Consider two examples as follows:

Example 1. If $g(\mathbf{x}) = \mathbf{x}$ which is monotonically increasing, then for any partition $P = \{\mathbf{x}_0, ..., \mathbf{x}_n\}$ on I

$$\sum_{i=1}^{n} |g(\mathbf{x}_i) - g(\mathbf{x}_{i-1})| = \sum_{i=1}^{n} [g(\mathbf{x}_i) - g(\mathbf{x}_{i-1})] = g(1) - g(0) = 1$$

Hence, $g(\mathbf{x})$ has bounded variation and $V_g(I) = 1$.

Example 2. If $g(\mathbf{x}) = 1 - \mathbf{x}^3$ which is continuous on I and differentiable on $[0, 1]$ with $\sup_{0 \leq \mathbf{x} \leq 1} |g'(\mathbf{x})| \leq M = 1$, then, by the Mean Value Theorem

$$\sum_{i=1}^{n} |g(\mathbf{x}_i) - g(\mathbf{x}_{i-1})| = \sum_{i=1}^{n} |g'(\mathbf{x})[\mathbf{x}_i - \mathbf{x}_{i-1}]| \leq \sum_{i=1}^{n} M[\mathbf{x}_i - \mathbf{x}_{i-1}] = M(1 - 0) = 1$$

Thus, $g(\mathbf{x})$ has bounded variation and $V_g(I) = 1$.

From Equation 2.42, two factors affect the value of the approximation error. One is the roughness of the integrand $g(\mathbf{x})$ (also known as the variant of $g(\mathbf{x})$). The other is the sample quality which is the similarity between the distribution of samples $F_{\{\mathbf{x}_i\}}$ and the common distribution of $g(\mathbf{x})$ called as F. The closer between $F_{\{\mathbf{x}_i\}}$ and F, the smaller the error is.

D_N is bounded by a constant $\frac{(\log N)^s}{N}$ (see Equation 2.41). Hence, quasi-Monte Carlo demonstrates a N^{-1} convergence on the estimated error. Comparing with the Monte-Carlo method which has $N^{-\frac{1}{2}}$ convergence (see Equation 2.40), quasi-Monte Carlo is able to *converge fast* on the size of error.

The low-discrepancy sequence involved in quasi-Monte Carlo integration is a deterministic sequence which provides improvements in the integration. Several such sequences have been proposed such as Halton sequences [25] and Niederreiter's (t, m, s)-nets and (t, s) sequences [30].

Niederreiter's Sequence

Niederreiter defines the sequences in a regular distribution behavior called (t, m, s) nets and (t, m) sequences [30].

A (t, m, s)-*net* is a sequence of points from I^s which have a certain equidistribution property. It means that all subintervals of I^s contain a certain number of points of the sequence. Those subintervals are called as *elementary intervals*.

An *elementary interval* is a subinterval of I^s

$$E = \prod_{i=1}^{s} [t_i b^{-k_i}, (t_i + 1)b^{-k_i})$$

where base b is a positive integer, k_i are nonnegative integers, and t_i are integers with $0 \leq t_i < b^{k_i}$.

Suppose t and m are nonnegative integers with $t < m$. For a finite sequence $\mathbf{x}_1$,...,$\mathbf{x}_n \in I^s$ where $n = b^m$, if every elementary interval in base b with volume b^{t-m} has b^t points of the sequence, then it is known as a (t, m, s)-net.

Let the equidistribution property extend to infinite sequences. If for all $m > t$ and $k \geq 0$, the finite sequence consisting of the $\mathbf{x}_n$ with $kb^m \leq n \leq (k + 1)b^m$ is a (t, m, s)-*net* in base b. $t \geq 0$, an infinite sequence $\mathbf{x}_1, \mathbf{x}_2, ... \in I^s$ is regarded as a (t, s)-*sequence* in base b.

Chapter 3

Literature Review

With the development of connectionist expert systems, how to explain the inference process in the system has become an important research area. At the end of 1980s, Gallant began to present a routine for extracting propositional rules from a simple network [35].

Rule extraction extracts symbolic rules from the ANNs or other knowledge expert systems to explain the knowledge embeded in the networks. The integration of symbolic and connectionist enhances users' understanding on the problem at hand. For example, given two classes in Figure 3.1, the users cannot tell why the data are broken into these two kinds of groups. The symbolic rules in Figure 3.2 illustrate the regularities in the groups.

To date, most rule extraction algorithms have been directed towards presenting the outputs as a set of rules using propositional logics [80]. Many research efforts have been concentrated on extracting knowledge from neural networks using fuzzy logics [47]. Some algorithms also have been employed to extract automata from recurrent neural networks [95]. Recently generating regression rules [73] has been studied as well.

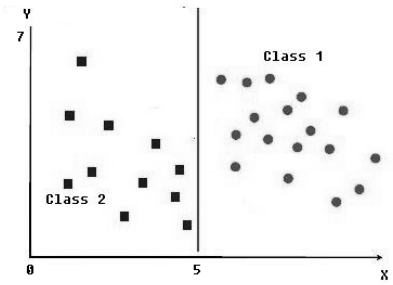


Figure 3.1: Two classes classification

$$X \in [0,5] \wedge Y \in [0,7] \rightarrow class2$$
$$X \in [5,\infty] \wedge Y \in [0,7] \rightarrow class1$$

Figure 3.2: Rules extracted from data groups

In 1998, Tickle et al. [12] categorized rule extraction techniques with five general notions:

1. **Rule format** refers to knowledge representation forms such as Boolean, fuzzy, logic rule (i.e. $\neg$ a $\wedge$ b $\rightarrow$ c) etc

2. **Rule quality** indicates rule fidelity (i.e. how many unseen examples are classified different with rules and learning models), accuracy (i.e. how many unseen examples are correctly predicted) and consistency (i.e. the same classification results of unseen examples are generated under different training sessions) and comprehensibility (i.e. the size of the resulting rule set)

3. **Translucency** reveals the associations between extracted rules and architectures of neural networks. It contains four categories: *'decompositional'*, *'pedagogical'*, *'eclectic'* and *'compositional'*.

4. **Algorithmic complexity** addresses the efficiency of the algorithm.

5. **Portability** measures how well the approaches can be applied to other neural

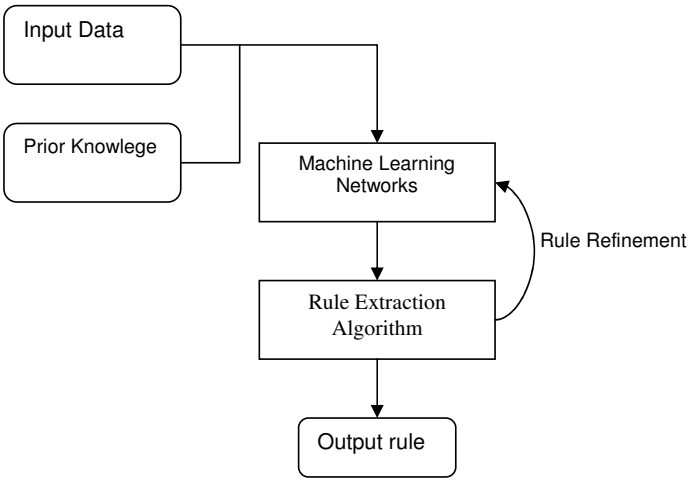


Figure 3.3: Rule extraction system

network architectures.

With the emergence of different techniques in the area of rule extraction, the epitome of the system to extract rules is as follows (see Figure 3.3). Data together with domain knowledge are input into machine learning networks such as trained artificial neural networks, recurrent neural networks etc. Then the results of classification are used by various rule extraction algorithms. The outcomes are the symbolic rules needed for decision making. The rule refinement process further improves the rule quality such as correctness, fidelity etc.

Since rule extraction integrates the connectionist and symbolic approaches, users of rule extraction systems could benefit from explanation capability and knowledge acquisition.

Andrews et al. [66] identifie three types of techniques for extracting rules from neural networks at the level of granularity, i.e. depending on whether the neural networks play an implicit role or not in a rule extraction algorithm. For instance, in some rule extraction techniques, only the inputs and outputs of a neural network are considered regardless of the inner structure. However, the current rule

extraction are not strictly divided into these three types. With the development of the techniques, i.e. measures for rule extraction; insertion and refinement [14], [12] adds another category called *'compositional'*. The following gives the descriptions on these four categories of rule extraction techniques.

- *'Decompositional'* extraction focuses on the extraction at the level of each unit of a neural network. Rules are extracted from the neurons in both hidden and output layers. This approach extracts rules based on the architecture of the neural network as a whole [41].

- *'Compositional'* extraction extracts rules by analyzing the ensembles of neurons in a network rather than one single neuron. It takes into account the rules which represent the transitions from old states of units to the new states.

- *'Pedagogical'* extraction treats the neural network as a black box. It extracts rules directly from the input and output values. Such techniques are used in hybrid approaches with a symbolic learning algorithm. The main purpose of this sort of approach is to find out the relations between the inputs and outputs and to specify the functions of the neural networks as rules.

- *'Eclectic'* extraction combines the elements of *'decompositional'* and *'pedagogical'* extractions. It utilizes the knowledge in neural network architectures to complement a symbolic training algorithm.

Most rule extraction approaches [53] [91] [12] deal with multi-layer perceptrons - the most common network type. Extraction from some other architectures like SOM [8] [4] [56] and recurrent networks [102] [63] exist as well. Recently, rule

extraction research has begun to focus on SVM [57] [59] [22] since its learning algorithm has certain advantages like better generalization ability, insensitivity to the training examples etc.

In the next few sections, we will illustrate the different rule extraction approaches in details. The approaches are categorized into four groups: *'decompositional'*, *'compositional'*, *'pedagogical'* and *'eclectic'*.

3.1 Decompositional Approach

The earliest works in this area [43] [46] extract boolean rules from each neuron in both hidden and output layers of the ANN.

Towell and Shavlik [91] apply searching of the weights entering a unit which ensures the combination of the input values exceeds the bias. If the criterion is satisfied, rules are extracted from each non-input unit in a trained ANN. The algorithm called *SUBSET* extracts individual rules (e.g. '**If** a **then** $b = no$ **else** $b = yes$'). It extracts rules at the levels of both hidden and output units. The rules drawn from those single units are aggregated into conjunction rules.

The *MofN* algorithm [91], which is based on *SUBSET*, represents the knowledge inside the network in the form of ' **If** *(M of the following N antecedents are true)* **then** *yes*'. $MofN$ assumes that the groups of antecedents could form equivalent classes in which each antecedent should have an equivalent importance. Hence, it first groups the links of units into equivalent classes using a standard clustering method, and eliminates those groups that are unable to influence the calculation of the consequent. After eliminating those unimportant groups, the *MofN* algorithm makes the rest of groups stay intact and retrains the networks by using back-propagation in order to obtain the optimized bias on each unit. Finally, rules are

extracted by directly translating the bias and incoming weights of each unit. In [91], Towell and Shavlik comment that it performs more efficiently than *SUBSET*.

The *KT* algorithm developed by Fu [53] [49] maps the results of both hidden and output layers into boolean rules. The rule has a form like '**if** $a > 2$ **then** *true*'.

The above algorithms have the similar basic motif which searches each (hidden and output) unit and treats the outputs as a step function or a propositional rule.

Example 1. Given an ANN network (see Figure 3.4) and an activation function

$$f(x_j) = sign(\sum_{i=1}^{N} w_{ij} - \theta_j)$$

where w_{ij} is the weight of the link from unit i to unit j (e.g. the weight is 2 from unit B to unit A), θ_j is the bias of unit j, and N is the number of links to unit j.

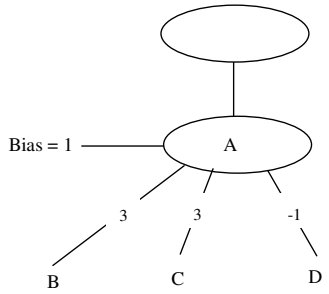


Figure 3.4: A unit of ANN

Considering Figure 3.4 and the activation function above, the rules can be found in different forms such as boolean rules or $MofN$ rules:

RuleNet / The Connectionist Scientist Game in [16] employs a technique similar to scientific induction. The explicit hypotheses about a domain are formed in order to extract symbolic condition-action rules. Those hypotheses are tested and refined iteratively in the connectionist network by injecting the rules back into

if B then A
if C then A
if B, D then A
if B, C then A
if C, D then A
if B, C, D then A

OR

if 1 of {B, C} then A
if 2 of {B, C, D} then A

Figure 3.5: Rules extracted from a unit in Figure 3.4

the network. The training process continues until the final rules reveal adequate domain characteristics. However this approach limits the ANN architecture to a specific problem domain and thereby lacks generality [91] [79].

RULEX developed by Andrew and Geva [67] [68] automatically extracts symbolic rules from the local responsive units (LRU) of a Constrained Error Backpropagation (CEBP) MLP. Each LRU is formed by each dimension of the input which is called ridge. The output of each LRU is the sum of the activations of the ridges. Therefore, propositional if-then rules are extracted within the threshold of LRU outputs.

COMBO, proposed by Krishnan [70], suggests to order weights to limit the search space of the neural network. *COMBO* builds a *combination tree* by sorting the incoming weights of a particular node in descending scalar. The combination tree is then systematically transformed to the propositional rules as follows:

IF

$$\sum w_p + bias\ \ on\ \ neuron > threshold\ \ on\ \ neuron$$

THEN

$$concept\ \ corresponding\ \ to\ \ the\ \ neuron\ \ is\ \ true$$

where w_p are the weights at the node of the combination tree.

A similar idea appears in [21] which also relies on the ordering of weights. By contrast, [21] extracts *MofN* rules from the minimal weight vectors which are found with respect to the partial order defined on weight vectors.

Saito and Nakano [44] put forward a new approach, *RF5*, for drawing scientific laws in the form of:

$$y_t = c_0 + \sum_{i=1}^{h} c_i x_{i1}^{w_{i1}} ... x_{il}^{w_{il}}$$

where y_t is the consequence of the law, the parameters c_i and w_{ij} are unknown real numbers and h is an unknown integer. The hidden layer architecture of ANN is composed of 'product unit', and the training algorithm is based on a quasi-newton method called BPQ. After training, the best law candidates are selected in terms of a *minimum description length* criterion.

The works of Setiono [71] [72] introduce the concept of pruned network. The network is first pruned and only distinct connections are left. Then the rules are generated from the hidden units with a small number of activation values. If the connections to the hidden units are not sufficiently small, then the hidden units are split and a new hidden layer is inserted.

Works related to the pruned network for rule extraction were developed by Ishikawa in 1990s. Two methods [54] [55] are used to find out the patterns with large contribution to the networks. The basic idea of [54] is to eliminate the unnecessary connections by a *Structural Learning with Forgetting* (SLF) [34]. The criterion function of SLF is,

$$J_f = J + \varepsilon' \sum_{i,j} |w_{ij}| = \sum_k (o_k - t_k)^2 + \varepsilon' |w_{ij}|$$

where J is the quadratic criterion in back-propagation learning, J_f is a total cri-

terion, w_{ij} is the weight from unit j to unit i, o_k is the output of unit k, t_k is the target value, the second term of the formula is a penalty criterion, and ε' is its relative weight. The hidden unit is deleted if it has small contribution to J.

The approach in [55] is based on SLF [54]. The criterion function changes to:

$$M = J + \lambda E_w$$

where E_w is known as a regularizer and is a function of connection weights, and λ is a regularization parameter. The dominant rules are created with a large λ value while the detail rules are those with a small λ value.

3.2 Compositional Approach

Since early 1990s, [102] [63] [17] have borrowed the knowledge of symbolic representation for use in recurrent networks. In recurrent networks, each unit in a network may send outputs to other units and receive inputs from the same unit.

Tickle et al. [12] introduce the fourth intermediate category - '*compositional*', which is to accommodate for *Recurrent Neural Networks* (RNNs). Most RNN rule extraction approaches are based on ensembles of neurons rather than individual units.

The basic algorithm of RNN rule extraction consists of four steps [26]:

1. continuous state space quantization,
2. state generation,
3. rule construction,
4. rule pruning,

Some works [102] [63] [17] focus on searching the state space, and use clustering to quantize the state. The result of the extraction algorithm is to extract deterministic finite-state automata from a recurrent network.

In 1992, Watrous and Kuhn [76] proposed a RNN rule extraction technique by using interval clustering and sampling. It is one of the influential works to use sampling to split individual state unit activations into intervals [26]. The intervals obtained from quantization are utilized to extract minimal and deterministic finite-state automata rules.

A RNN rule extraction algorithm based on *Self-Organizing-Map* (SOM) is developed in [64]. A Kohonen SOM with the star topology of neurons is employed to quantize the state space into distinct regions representing corresponding states of a *Deterministic Mealy Machine* [26] being learned.

In [65], Tino et. al investigate the knowledge induction process associated with training recurrent neural networks (RNNs) on single long chaotic symbolic sequences. It replaces SOM with dynamic cell structures (DCS) for quantization. A stochastic *Mealy Finite State Automata* is then extracted.

3.3 Pedagogical Approach

The earliest works on 'pedagogical' rule extraction was introduced by Saito and Nakano [43]. The ANN is treated as a transparent architecture and the rules about a medical diagnostic problem are extracted only from the input and output values.

Thrun [10] describes an approach to extract rules by using *Validity Interval Analysis* (VIA). *Validity Interval* (VI) is defined to constrain the activation patterns in the networks. The VI of a unit represents the maximum range of its activation values.

The initial VIs usually are arbitrary. Then VIA iteratively refines the VIs by throwing out those active values which are inconsistent with the weights and biases of the network in both forward and backward phases.

Given a simple ANN in Figure 3.6, the activation function is

$$net_j = \sum_{i=1}^{N} w_{ij} * x_i - \theta_j$$

$$x_j = \sigma_j(net_j), \text{ where } \sigma(net_j) = \frac{1}{1+e^{-net_j}}$$

where w_{ij} is the weight of the link between unit i and j, θ_j is the bias of unit j, and N is the number of the links to unit j. After obtaining an initial set of VIs, there are two types of results. One is the intervals covering all the activation patterns; the other is a contradiction (i.e. an empty interval which means no activation patterns consistent with the initial intervals). VIA algorithm excludes the second type of results and interprets the first type of results into rules (see Figure 3.7).

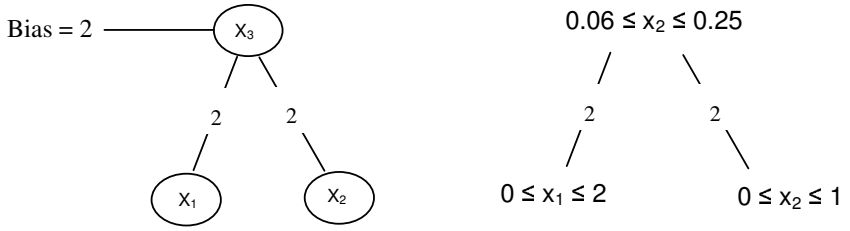


Figure 3.6: Simple example of VIA algorithm in forward phase

if $0 \leq x_1 \leq 2$ and $0 \leq x_2 \leq 1$ then $0.06 \leq x_3 \leq 0.25$

Figure 3.7: Rules extracted from a unit(Fig 3.6)

In Figure 3.6, suppose that VIs $[0, 2]$ and $[0, 1]$ are initially set to unit x_1 and unit x_2. The minimum net_3 of unit x_3 then equals $w_{13} * x_1^{min} + w_{23} * x_2^{min} = 2*0+2*0 = 0$

while the maximum net_3 of unit x_3 equals $w_{13} * x_1^{max} + w_{23} * x_2^{max} = 2 * 2 + 2 * 1 = 6$. Applying the values of net_3 into the activation function, the range of x_3 is $[0.06, 0.25]$. Since the output is consistent with the initial intervals, the intervals are mapped to an interval rules (see Figure 3.7).

Craven and Shavlik [50] describe the *'Rule-extraction-as-learning'* approach which is an important work in the field of pedagogical rule extraction. The algorithm reads the training examples one by one, classifies them and generates rules from the example if the existing rule set does not cover it. The new rule r is then generalized by dropping one of the rule conditions. If all the examples covered by r are in the same class c, then r is accepted. The process stops until one of the following two 'stopping criteria' are met.

1. The extracted rule set is sufficiently accurate.

2. No new rule is created after a certain number of iterations.

RULENEG developed by Pop et al. [18] is to extract conjunctive rules, where every symbolic rule can be stated as a disjunction of conjunctions. After initializing a new rule r for a certain training example $\mathbf{a}$, if it is not covered by the existing rules, *RULENEG* negates every attribute value of $\mathbf{a}$ to construct a new pattern $\mathbf{a}'$. If the class of $\mathbf{a}'$ is not equivalent to that of $\mathbf{a}$, the attribute of $\mathbf{a}$ and its value are then added to r. The *RULENEG* algorithm is a simplification of the algorithm developed by Craven and Shavlik [50].

The *BRAINNE* system [79] focuses on the closeness between inputs and outputs. It first changes the architecture of a network from M inputs and N outputs to $M+N$ inputs and N outputs so that the network structure is retrained. The second step is to compare the weights for the links between each of the initial M inputs and the corresponding hidden units with the weights from each of the additional N

inputs and related hidden units. Thus, the smallest difference shows the greatest contribution of the original inputs to the outputs.

The Interval Analysis (IA) algorithm [69] is a method to extract propositional rules in the pedagogical category. It introduces the concept of *Hypercube*.

Definition 3.3.1. *Hypercube. Consider an ANN with m inputs* $x_1, x_2, ..., x_m$*, and each input* x_i*,* $(i = 1, ..., m)$ *can take a continuous range* $[a, b]$*. The hypercube is the conjunction of these ranges.*

The IA method includes the following steps. Given a *class* A and a minimum hypercube of size Q,

1. Start with a certain hypercube in the input space which belongs to A, grow the input space until the outputs begin to include another class.
2. Repeat 1 till all the potential hypercubes are found. Those hypercube should be subject to the constraint that the hypercube cannot include any input space that belongs to the other class rather than the target class A.
3. Start from each face of the previously found hypercube, enlarge and move the hypercubes to cover the remaining input space for class A as much as possible without overlapping with other hypercubes.
4. Check if the size of the new hypercube is smaller than Q.
5. Repeat 3 and 4 for every hypercube in the previously found hypercube set.
6. Search the input space to guarantee none of noncontiguous area for class A has been missed.

The *TREPAN* algorithm [50] extracts rules in the form of decision tree. The fundamental part of *TREPAN* is that it introduces the notion of *Oracle*, which

determines the decision tree by using queries. The expansion of trees is based on the best-first method, which means the nodes having the greatest effects on the fidelity of the extracted tree have the first priority to expand. Each node of the tree represents a binary or $MofN$ rule.

REFNE designed by Jiang and Chen [101] utilizes the trained neural network ensembles to generate instances and then applies to the rule creation process defined by Fu [53] to extract symbolic rules from those instances. Note that a neural network ensemble is a model in which a collection of neural networks is used to solve a specific problem. It is believed to significantly improve the generalization ability of a neural network based system [48].

Recurrent neural networks (RNNs) are composed of inputs, outputs and the state space. Most RNN rule extractions fall in the group of *'compositional'*. [5] [6] are the only works focusing on the inputs/outputs of a trained recurrent network to extract deterministic finite-state automata (DFAs) instead of looking inside the internal state. The methods increase the fidelity of DFAs in the cases of large sets of strings.

The Self-Organizing Map (SOM) is frequently applied to symbolic interpretation because it is very effective for hierarchical data. Siponen et al. [56] use some significance measures [56] such as frequency to interpret the information in the clusters generated by SOM learning.

Another SOM rule extraction technique [8] classifies data with generalized relevance learning vector quantization (GRLVQ), one type of SOM clustering algorithms. After training, GRLVQ employs a decision tree approach to generate rules. Each node N of the tree has C^N children. An interior node N is labelled by an indexed I^N and real values. Each leaf L of the tree is labelled with a class number.

In Section 3.5, a deep description for those rule extraction methods with SVM

networks will be given. They are pedagogical so they are introduced here.

To date, few algorithms on rule extraction from SVM networks have been proposed [59] [57] [22]. Nũñez and Angulo [59] introduce an approach for extracting rules from SVMs. It indicates a procedure that uses support vectors from SVM and prototypes gained from K-Means clustering to define geometric ellipsoids in the input space. The ellipsoids are then mapped to IF-THEN rules. However, this approach does not perform well for a large number of patterns [57].

The method for SVM rule extraction developed by Barakat and Diederich [57] refers to two data sets. The first data set is the original one for SVM learning, while the second one is generated with the same attributes but modified values. The basic view of the second data set is to generate the generalized patterns for prediction. The resulting patterns are then trained by a decision tree system and the corresponding rule sets are drawn.

Another rule discovery algorithm based on SVM is developed by Fung [22]. The algorithm formulates a linear SVM hyperplane classifier and approximates the classifier with a set of rules. However, this work is only tested on training data and linear classifiers.

3.4 Eclectic Approach

Eclectic approach combines the characteristics of decompositional and pedagogical approaches. Ultsch [4], for instance, uses neural nets with SOM to extract the regularities out of the training examples. Then a rule generator called *sig**, which considers the significance of the range of an attribute value, transforms them into PROLOG rules.

The *DEDEC* [11] is another vital technique. Finding minimal information distinguishing a given pattern from others is the major advantage of this algorithm. Another key characteristic of *DEDEC* is to rank input cases in the order of importance. The importance of the input cases is assessed by the contribution of input units to the ANN outputs. It is achieved by using the scales of weight vectors for each input units. Then, rules are extracted from those typical patterns.

The method for symbolic rule extraction developed by Garcez et al. [2] is based on the idea of 'regularity', an ordering on the set of input vectors. The network is decomposed into smaller subnetworks called *Basic Neural Structure* (BNS), and regularities can be found. As a result, a number of simple rules can be derived from BNS. The final rule set is the result of combining those rules and simplifying them.

Genetic algorithm (GA) has been used to optimize the structure of neural networks and rule quality. [83] uses GA to find a good neural network topology. The neural network topology is then passed to a rule extraction algorithm for rule extraction. Next, the quality of the extracted rules is evaluated under the criteria of accuracy and comprehensibility. After that, the evaluation results are fed back to GA as fitness values. Finally, the extracted rules are those from the best topology of last generation.

3.5 SVM-based Rule Extraction Approach

Since the early 1990s, various algorithms to extract rules from trained neural networks have been proposed, notably [10, 2, 69, 44, 53, 18]. Some of these search for rules by decomposing the networks and extracting rules for each unit, and some extract rules directly from the input-output values of the networks, thus treat-

ing them as black-boxes. Recently, SVMs have been taken into account for rule extraction because of their excellent generalization capability.

The SVM+Prototype method [59] is one of the vital approaches in the area of SVM rule extraction. It uses support vectors and prototypes to draw the region covered by a rule. There are two types of target rules in this method:

- equation rules: IF $a \cdot x_1^2 + b \cdot x_2^2 + c \cdot x_1 \cdot x_2 + f \leq g$ THEN class,
  where $a, b, c, f, g \in \mathbb{R}$
- interval rules: IF $x_1 \in [a, b]$ and $x_2 \in [c, d]$ THEN class,
  where $a, b, c, d \in \mathbb{R}$

The prototype in [59] is defined as:

Definition 3.5.1. *Prototype. Suppose that the examples are clustered into some groups C_1, C_2,...C_n, where n is the number of clusters. The prototype is the center of a cluster C_i, $1 \leq i \leq n$.*

For example, a group with three points $\{[1, 2], [6, 5], [5, 6]\}$. The prototype is $[\frac{1+6+5}{3}, \frac{2+5+6}{3}]$ which is $[4, 4.33]$.

Geometrically, the graph of an equation rule is an ellipsoid while the graph of an interval rule is a hyper-rectangle. [59] chooses a prototype point and the farthest support vector to this prototype to construct a axis of this ellipsoid. Next, it determines vertices to build the rest of the axes. The vertices may be the following points: a) the support vector itself; b) a point which is the farthest to the prototype; c) a point derived from a support vector. The difference between an equation rule and an interval rule is that the axes of the interval rule are parallel. Figure 3.8 shows these two types of rules in geometry.

The procedure of the rule extraction is described as follows [59]. Firstly, the algorithm begins with a single prototype and creates a general ellipsoid/hyper-rectangle. Then, it applies a partition test to examine if the region of this ellipsoid / hyper-rectangle belongs to the same class. And the ellipsoid/hyper-rectangle is divided. If the result of the test is negative, a new rule is then generated. The procedure may stop when all partition tests are negative.

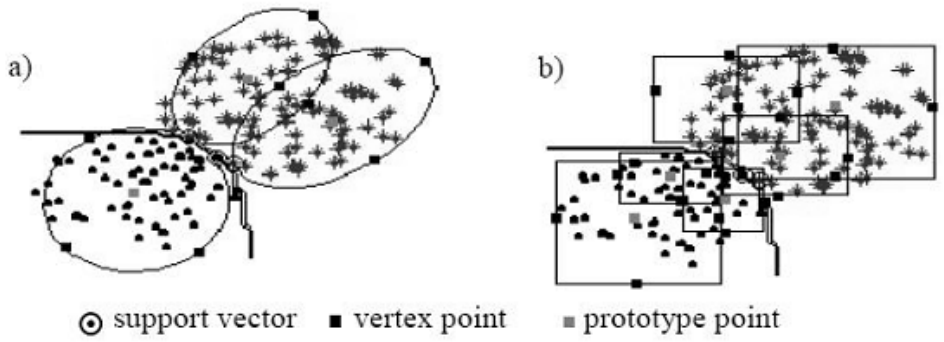


Figure 3.8: a) Equation-rule b) Interval rule [59]

However, a large number of patterns and overlaps within different attributes decreases the comprehensibility of the extracted rules and increases the algorithm complexity.

Barakat et al. [57] propose an approach to extract rules from SVMs from three data sets:

A) A modified 'Pima Indian Diabetic' data set used for SVM training. Examples are randomly selected from this data set and noisy data are removed. Hence, a SVM model is obtained after training.

B) A data set with the same attributes as the first one but different values used for generalization testing. The SVM model is employed to predict the class for each record in the second data set.

C) A data set used to merge A and B. This data set is trained by decision tree or classification tree algorithms to extract symbolic interpretations.

This approach combines the strength of generalization of SVM and explanation of tree learning algorithm. Nevertheless, in the context of the criteria for rule extraction [12], the limitations of computational complexity make this approach inappropriate.

In [22], it describes an algorithm to approximate the linear SVM classifier. The linear classifier is defined in Equation 2.7. The antecedents of the rule is in the form of:

$$\wedge_{i=1}^{m} l_i \leq x_i \leq u_i$$

where i indicates an element of the input $\mathbf{x}$, m is the dimension of $\mathbf{x}$. l_i and u_i are arbitrary real values.

As each rule covers a corresponding positive/negative region for the classifier, Fung [22] defines:

$$A_- = \{x | w \cdot x < \gamma, l_i \leq x_i \leq u_i, 1 < i < n\}$$
$$A_+ = \{x | w \cdot x > \gamma, l_i \leq x_i \leq u_i, 1 < i < n\}$$

where γ is an arbitrary real value. A_- and A_+ represent the regions belonging to negative and positive classes. Figure 3.9 shows the areas A_- and A_+ that rules cover are rectangles. The more rectangles, the closer we get to the linear classifier.

The approach in Fung [22] establishes a diagonal matrix T and a vector $b = \{u_i, if\ w_i < 0; l_i, if\ w_i > 0\}$ to convert original linear rule extraction problem

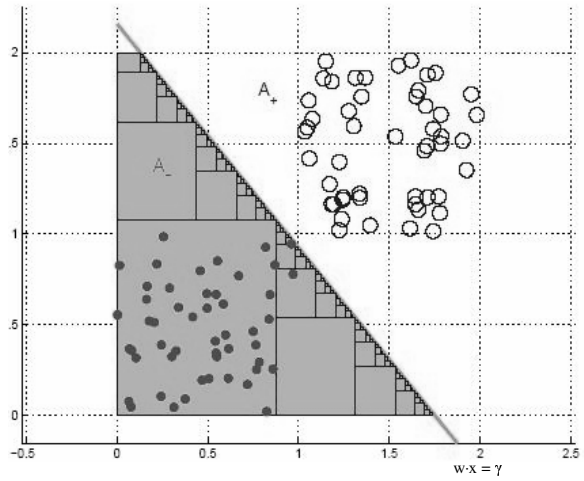


Figure 3.9: The regions of A_- and A_+ approximate the area in which the points are classified by a linear classifier [22]

into the following form in terms of linear transformations.

$$T_{ii} = \frac{sign(w_i)}{u_i - l_i}, \quad i \in \{1, ..., n\} \tag{3.1a}$$

$$w_i > 0 \Rightarrow 0 \leq y_i = \frac{T_{ii}}{u_i - l_i} = \frac{x_i - l_i}{u_i - l_i} \leq 1, w_i < 0 \Rightarrow 0 \leq y_i = \frac{T_{ii}}{u_i - l_i} = \frac{u_i - x_i}{u_i - l_i} \leq 1 \tag{3.1b}$$

$$\Downarrow$$

$$x = T^{-1}y + b \tag{3.1c}$$

$$\Downarrow$$

$$w \cdot x = \gamma \Rightarrow w \cdot T^{-1}y = \gamma + w \cdot b \tag{3.1d}$$

$$\Downarrow$$

$$(\frac{w \cdot T^{-1}}{\gamma - w \cdot b})y = 1 \tag{3.1e}$$

Then, the optimal rules are extracted which cover as much as possible region. [22] uses *log* function because it is strictly increasing. The optimization problem is as follows:

$$\begin{aligned} \underset{(\mathbf{x}\in\mathbb{R}^s)}{Max} \quad & log(\prod_{i=1}^{s} x_i) \\ s.t. \quad & \sum_{i=1}^{n} w_i x_i = \gamma, \quad 0 \leq x_i \leq 1 \end{aligned} \tag{3.2}$$

In the context of Karush-Kuhn-Tucker theorem and Langarian, the solution turns to:

$$\tilde{x_i} = \frac{1}{\lambda w_i} = \frac{\gamma}{n w_i} \Rightarrow x_i^* = \begin{cases} \frac{1}{\lambda^* w_i}, & if \quad \tilde{x_i} \leq 1, i \in \{1, ...n\} \\ 1 & otherwise \end{cases} \tag{3.3a}$$

and,

$$\lambda^* = \frac{n_I}{\gamma - \sum_{i\in A} w_i} \tag{3.3b}$$

where $A = \{i/\tilde{x_i} > 1\}$ and $n_I = n - |A|$.

The third step is to compute $\mathbf{x}$ by using Equation 3.1. Finally a new rule is generated upon $\mathbf{x}$.

However, there are some limitations of this method such as portability, it can only be applied to linear classifiers, and generalization can only be examined by training examples.

In 2005, Barakat and Diederich put forward another approach [58] to extract rules from the support vectors provided by SVM networks. This eclectic approach involves three stages:

1. The learning stage is to train an SVM and get an SVM model.

2. The rule generation stage includes three phases: a) select the unlabeled sup-

port vectors. b) ask the queries from the trained SVM network and construct a synthetic data set. c) train the synthetic data set on a machine learning technique such as decision tree to obtain the symbolic rules.

3. The evaluation stage is to assess the fidelity and accuracy of the rules.

This work demonstrates that using information, such as support vectors, which provide vital information on the separating hyperplane, can draw the rules with high accuracy and fidelity.

Fu [99] describes a new SVM rule extraction *RulExSVM*. The extracted rules are hyper-rectangular and in IF-THEN form. Each rule has a support vector and is generated directly from SVM and the decision function. It contains three parts. The initial part is for generating rules. Figure 3.10 illustrates how to obtain the initial rules from SVM and decision boundary. Suppose black points refer to the support vectors of class A_1 and white points are the support vectors of class A_2. Along each axis, a line can be extended from a support vector. If there are cross points between the line and the decision boundary, then according to the values of the support vector and its corresponding cross points, a rule can be generated by determining the upper and lower limit of the hyper-rectangular. Afterwards, it eliminates those redundant rules to produce a more concise rule set.

However, there are also some outliers falling into the regions of the initial rule, such as, the point I in Figure 3.10. The purpose of the second part, tuning phase, is to remove this kind of outliers. The details are as follows.

1. Search all examples of class A_2 falling into the rule region of class A_1. Let K be the set of these outliers.

2. Calculate the distances from a $k \in K$ to boundaries along each dimension.

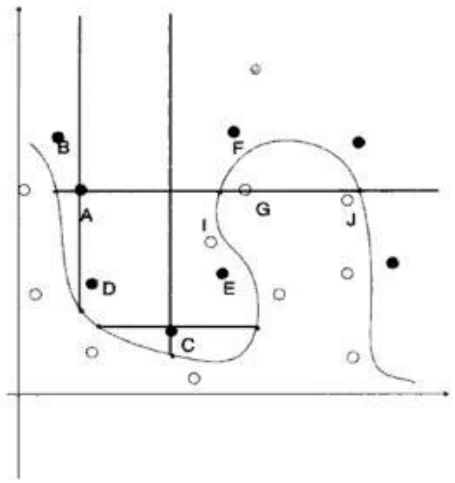


Figure 3.10: A two-dimension example to get the cross points for the initial phase of *RulExSVM* [58]

3. Remove k by shrinking the rule along the dimension which contributes most to the volume of the hyper-rectangle.

RulExSVM presents a novel SVM rule extraction algorithm by using the information of support vectors and classification boundaries. However, according to the algorithm in [99], this may lead to low rule fidelity because the tuning phase may not detect and remove outliers completely if only based on training examples.

Additionally, there is some research concentrating on creating fuzzy rules from SVMs rather than rectangular rules.

Jame et.al [100] investigate the connection between fuzzy classifiers and SVM. It proves that a fuzzy classifier can be a translation invariant kernel[1] under some assumptions. After evaluating the kernel function, the IF-part of a fuzzy rule is given by the support vectors. Then a fuzzy rule-based classification system called the positive definite fuzzy classifier is built based on the training examples and a support vector machine. The experiment results show good generalization.

[1]A kernel $K(\vec{x}, \vec{z})$ is translation invariant if $K(\vec{x}, \vec{z}) = K(\vec{x} - \vec{z})$ [100]

Chaves et al. [20] propose another fuzzy rule extraction from SVM. It pre-defines n overlapping fuzzy sets $\{C_{ij}\}$ for each dimension. Note that $\{C_{ij}\}$ is the j^{th} fuzzy set for the i^{th} dimension. Then, the algorithm projects the support vectors on each dimension and evaluates the membership degree[2] of each support vector projection. Subsequently, each support vector projection is assigned to a C_{ij} with the maximum membership degree. The last step is to generate IF-THEN fuzzy rules. Given a support vector $p = [p_1, ...p_s]$ for class A_p and fuzzy sets $\{C_{im^i}\}$ with the highest membership degree for p on each dimension, $1 \leq i \leq s$ and $m^i \in 1, ...n$, the rule for p will be

$$\text{IF } x_1 \text{ is } C_{1s^1} \text{ ,..., } x_m \text{ is } C_{ms^m} \text{ THEN } \mathbf{x} \text{ is for class } A_p$$

[2]Suppose $\mu_A(\mathbf{x})$ is the membership function for any $\mathbf{x}$ in a fuzzy set $\bar{\mathbf{X}}$. If $a \in \bar{\mathbf{X}}$, then $\mu_A(a)$ is membership degree for a.

Chapter 4

The GOSE Algorithm

As discussed in the previous chapter, most rule extraction algorithms suffer from a lack of generality, a balance between correctness and fidelity, or both. In this chapter, we present a novel rule extraction algorithm called *Geometric and Oracle-Based Support Vector Machines Rule Extraction* (GOSE), which is designed to alleviate these limitations.

As a pedagogical algorithm, GOSE utilizes the points on the SVM classification boundary and synthetic training instances to construct a set of optimized hypercube rules. The area covered by those rules is maximized and approximates the *area of interest*. The definitions of the *hypercube rule* and the *area of interest* are given as follows.

Definition 4.0.2. *Hypercube Rule: It is said that an m-dimensional hypercube H characterizes a rule if every point in the scope of H falls into the same class classified by the SVM network. More precisely:*

$$H = \left\{ \mathbf{x}_i = [x_{i1}, ..., x_{im}] \;\middle|\; [l_1, ..., l_m] \leq [x_{i1}, ..., x_{im}] \leq [u_1, ..., u_m] \rightarrow A_k, i \geq 1 \right\},$$

where $[l_1, ..., l_m]$ *and* $[u_1, ..., u_m]$ *are the upper and lower bounds on* H*.* A_k *indicates a class label, and* m *is the dimension of the input space.*

Note that the form of a hypercube rule can also be represented in a conjunctive form:

$$\bigwedge l_j \leq x_{ij} \leq u_j \rightarrow A_k \quad 1 \leq j \leq m$$

Definition 4.0.3. *Area of Interest. This is the whole region covered by a class* A_k *in the input space.*

$$I(A_k) = \left\{ \mathbf{x}_i \;\middle|\; \textit{the class of } \mathbf{x}_i \; = A_k,\; l_j \leq x_{ij} \leq u_j,\; 1 \leq j \leq m, i \geq 1 \right\},$$

where $\mathbf{x}_i$ *is an* m*-dimensional input vector.* u_j *and* l_j *are the upper and lower bounds of* x_{ij}*.* x_{ij} *refers to the* j^{th} *dimension of* $\mathbf{x}_i$*. There is no vector* $\mathbf{x}_i \in I(A_k)$ *such that the classification of* $\mathbf{x}_i$ *equals the other class rather than* A_k*.*

The aim of GOSE is to use the classification boundary and synthetic training instances to extract the hypercube rules without considering the inner structure and the support vectors of the SVM network. The role of the network is merely to answer queries from the inputs [50] so that the network is regarded as an *oracle*. The advantage of an *oracle* is that it is independent of the training examples and training architectures. GOSE is therefore capable of running on all kinds of SVM networks. It is even believed to be adequate for many other hyperplane-based learning models. Although some SVM rule extraction algorithms also consider the SVM network as a '**black box**', they have to use the same set of training examples to train the SVM network and to extract rules from the network. At the same time, for these algorithms, the support vectors of the trained SVM model also need to be applied to their rule extraction processes. In contrast, GOSE provides a much more general foundation for implementing the rule extraction method with fewer

assumptions on the architecture and training process than the existing methods.

4.1 Overview of the Approach

GOSE is a novel approach for rule extraction, as it treats the problem of extracting symbolic rules from an SVM as an approximation task so that the extracted rules simulate the behavior of the SVM network.

Figure 4.1 presents an overview of the GOSE algorithm. Given a set of examples from a problem domain, GOSE uses these examples to generate a sequence of new inputs $\mathbf{x}_i$ ($i \geq 1$) based on a density estimator $M(\mathbf{x})$. Each $\mathbf{x}_i$ is then input into the trained SVM to determine its class label y_i and construct an instance $(\mathbf{x}_i, y_i)$. After the selection procedure, a set of synthetic training instances $\mathbf{S} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq n\}$ is obtained. We call this procedure *querying* (see Figure 4.1). After *querying*, a *clustering* process is imposed on those inputs $\mathbf{x}_i$ with the same y_i, in order to group them into a set of clusters. GOSE then searches for a set of points $\mathbf{P} = \{\mathbf{p}_k : k \geq 1\}$ that lie on the SVM classification boundary by using a binary search algorithm. Afterwards, starting from each point in $\mathbf{P}$ and $\mathbf{S}$, GOSE extracts rules by solving optimization problems, and an initial rule set $\mathbf{R}$ is acquired. The purpose of this optimization problem is to extract rules that maximize the area and the number of training instances $(\mathbf{x}_i, y_i)$ covered. After $\mathbf{R}$ is obtained, several *post-processing* measures, such as rule extending, rule pruning, non-overlapping rule construction and rule selection, are applied to $\mathbf{R}$ for rule simplification. Therefore, a relatively small number of generalized hypercube rules are derived. These *post-processing* techniques not only promote the comprehensibility of $\mathbf{R}$ but also offer a high level of control of the estimation errors of the rules. In what follows, we will explain each of the above steps in detail. Note that this thesis focuses on the classification problems.

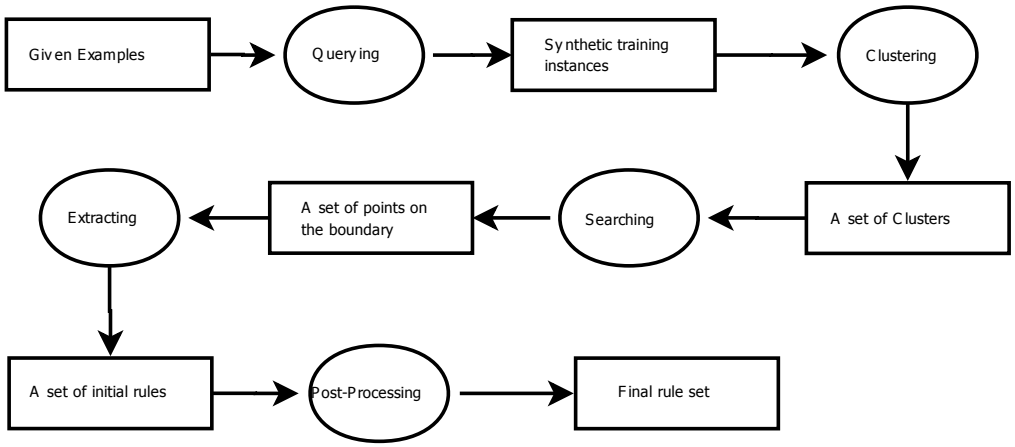


Figure 4.1: An overview of GOSE

Although GOSE uses geometry to extract rules that may have some similarities to other SVM rule extraction algorithms, it is substantially novel in many ways:

1. GOSE can extract rules from synthetic training instances in which the inputs are produced from a specific data generator.

2. GOSE treats the SVM network as an *oracle* which makes it independent of the original training examples, support vectors and the inner structure of the network.

3. GOSE is capable of being applied to any SVM network, either linear or nonlinear.

4. GOSE adopts a series of post-processing measures to improve the quality of the rules.

The key steps of GOSE are described in detail in the rest of this thesis.

4.2 Querying

The *querying* step in GOSE involves generating synthetic training instances. The generality of GOSE is derived from the utilization of those instances for rule extraction.

4.2.1 Synthetic Instances Generation

Most SVM rule extraction algorithms employ given training examples to generate rules and mimic the behavior of the networks. A major limitation of conventional extraction algorithms is that those training examples might not contain enough information about the networks. In contrast, GOSE is not limited to use the training examples given in a problem domain; it also employs synthetic training instances for rule extraction. Therefore, GOSE could create extracted rules in a much more general manner. GOSE initially produces a large amount of instances in the input space. It then selects those instances with high frequencies for rule extraction.

We use and extend the querying method in [50]. Suppose n synthetic instances are needed to extract rules from the SVM network. Figure 4.2 presents the process of *querying*.

As can be seen in Figure 4.2, the function DRAWCLASS is called to determine how many classes are involved in a classification problem. The details of DRAWCLASS are shown in Figure 4.3. Initially, a large number of random data $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_h\}$[1], which are uniformly distributed in the input space (L and U being the upper and lower bounds of the input space), are produced from a data generator $g(\mathbf{x})$. These

[1] h is an arbitrary large integer.

QUERYING

Input: the lower and upper bounds of the input space L and U, a constraint $c1$, a set of training examples $(\mathbf{X}, Y) = \{(\mathbf{X}_1, Y_1), ..., (\mathbf{X}_N, Y_N)\}$ from the input space, the size of synthetic instances n, a density estimator $M(\mathbf{x})$, and a random data generator $g(\mathbf{x})$
Output: a set of synthetic instance $\mathbf{S} = \{(\mathbf{x}_i, y_i),\ 1 \leq i \leq n\}$
(1) **Initialize** the number of iteration T and $\mathbf{S}$
(2) $A = DRAWCLASS(L, U, g(\mathbf{x}))$
(3) $\mathbf{x} := \{\mathbf{x}_i : 1 \leq i < T\} = DRAWINPUTS(c1, L, U, M(\mathbf{x}), \mathbf{X}, g(\mathbf{x}), T)$
(4) **for each** $i \leq T$
(5) class label $y_i = QUERYSVM(\mathbf{x}_i)$
(6) $\mathbf{S} := \mathbf{S} \bigcup (\mathbf{x}_i, y_i)$
(7) $\mathbf{S} = SELECTINSTANCE(\mathbf{S}, A, n)$

Figure 4.2: The QUERYING function which is composed of four functions: DRAWCLASS, DRAWINPUTS, QUERYSVM and SELECTINSTANCE

data $\mathbf{Z}$ are then input into the SVM network to acquire class labels for them. For large h, we believe that the data in $\mathbf{Z}$ are able to spread throughout the input space. Therefore, a set of classes ($A = \{A_k | 1 \leq k \leq CN, CN$ is the total number of classes $\}$) can be obtained after filtering the duplicates.

DRAWCLASS

Input: the lower and upper bounds of the input space L and U, and a random data generator $g(\mathbf{x})$
Output: the class set A
(1) **Initialize** A
(2) **Generate** a set of uniformly distributed random data $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_h\}$ from $g(\mathbf{x})$, which are located between L and U, $h \geq 1$
(3) **for each** $i < h$
(4) class label $y_i = QUERYSVM(\mathbf{x}_i)$
(5) **If** $y_i \notin A$
(6) $A := A \bigcup y_i$

Figure 4.3: The DRAWCLASS function: use a sequence of random data $\mathbf{z}_i, 1 \leq i \leq h$ in the input space as inputs to query SVM network and obtain the class labels for the problem domain.

The second part of the *querying* step is the DRAWINPUTS procedure (Figure 4.4). This is a key step for GOSE. In order to classify the inputs that are not in the

training set, GOSE should initially create them. Usually, these inputs are generated subject to some constraints, such as the requirement that the inputs must have a relatively high probability of appearing in the problem domain (symbolized by a user-defined parameter $c1$) and the condition that the value of each attribute be limited in the input space of the problem domain.

DRAWINPUTS

Input: a constraint $c1$, the lower and upper bounds of the input space L and U, a random data generator $g(\mathbf{x})$, a density estimator $M(\mathbf{x})$ and the number of iteration times T
Output: the input set $\mathbf{x} = \{\mathbf{x}_i : i \geq 1\}$
(1) **Initialize** the input set $\mathbf{x}$
(2) **Generate** T random data $\{\mathbf{x}_i : 1 \leq i \leq T\}$ based on $g(\mathbf{x})$
(3) **for each** iteration $i < T$ **do**
(4) **If** $L \leq \mathbf{x}_i \leq U$ **then**
(5) **Calculate** $M(\mathbf{x}_i)$
(6) **if** $M(\mathbf{x}_i) > c1$ **then**
(7) $\mathbf{x} := \mathbf{x} \bigcup \mathbf{x}_i$

Figure 4.4: The DRAWINPUTS function: call a random data generator $g(\mathbf{x})$ to create uniformly distributed data; use a density estimator $M(\mathbf{x})$ to reserve those data whose probabilities are larger than an arbitrary number $c1$. Meanwhile, the outputs should lie between L and U

.

Figure 4.4 shows the DRAWINPUTS routine. The function takes a density estimator $M(\mathbf{x})$, a data generator $g(\mathbf{x})$, the number of iterations T, a constraint $c1$ and the boundaries of the input space L and U as inputs. This is a T time-iteration procedure, undertaken after T random input vectors $\{\mathbf{x}_i, 1 \leq i \leq T\}$ are generated using $g(\mathbf{x})$. For each $\mathbf{x}_i$, the function checks if it satisfies the constraint $L \leq \mathbf{x}_i \leq U$. Only those in the input space are retained. Then DRAWINPUTS function then calculates the probability distribution of $\mathbf{x}_i$ based on the density estimator $M(\mathbf{x})$. If the value of $M(\mathbf{x}_i)$ is larger than the constraint $c1$, then this $\mathbf{x}_i$ is kept. Next, the algorithm sorts all the $\{\mathbf{x}_i : i \geq 1\}$ that satisfy the above two conditions in a descending order of the values of $M(\mathbf{x}_i)$. The reason for this is that GOSE uses

only n instances. The sorting ensures that the synthetic training instances for rule extraction are those that have a higher likelihood of taking place.

As shown in Figure 4.4, six parameters, T, $c1$, L, U, $g(\mathbf{x})$ and $M(\mathbf{x})$, are needed to be set. T is a large integer so that sufficient inputs can be created from the iteration. $c1 \in [0, 1)$ is used as a constraint to filter the inputs that have a low probability distribution. L and U comprise information on the problem domain given in advance.

For $g(\mathbf{x})$, although the generator of a low-discrepancy sequence shows a highly correlated manner[2] for the data being generated, the *quasi-random* property indicates clearly that the low-discrepancy sequence is not random. Hence, in our approach, we employ the uniform random data generator $g(\mathbf{x})$ to produce a large amount of inputs.

$M(\mathbf{x})$ is a vital parameter for the DRAWINPUTS function. As described by Craven [50], one method is to take into account the actual distribution[3] in the problem domain. Based on this approach, GOSE might not able to pick up the data from a uniform distribution over the whole input space, where it may find rules that simulate the behavior of the SVM network at a high fidelity level. However, it can focus on extracting rules that have an especially high level of fidelity in the part of the input space where instances are most likely to be found, as presented in Figure 4.4.

One way to find a distribution model is to construct a model of the underlying distribution of obtainable data and use this model to draw new synthetic data. Unlike TREPEN [92], which uses the *kernel density estimates of individual features* for con-

[2]The n^{th} data "knows" where the $(n-1)^{th}$ data is in the input space, which reduces the possibility that n data are gathered together.

[3]In our experiments, as the actual distribution of the problem domain is unknown, we will assume the distribution of $\mathbf{X} = \{\mathbf{X}_1, ..., \mathbf{X}_N\}$ as the actual distribution, where $\mathbf{X}$ is the input set of the training examples.

tinuous attributes and *empirical distributions* for discrete attributes, GOSE uses a *multivariate kernel density estimate* both for discrete and for continuous features. Those discrete values are represented by integers, which are considered a special type of continuous value in GOSE. Simultaneously, the *multivariate kernel density estimate* allows for *relations within attributes*. The definition of the *multivariate kernel density estimate* is as follows:

$$M(\mathbf{x}_i) = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{\prod_{j=1}^{m} h_j}[\frac{1}{(\sqrt{2\pi})^m}e^{-\frac{1}{2}(\|\frac{\mathbf{x}_i-\mathbf{x}_k}{h}\|)}] \tag{4.1}$$

where $\{\mathbf{X}_k, 1 \leq k \leq N\}$ indicates a set of the given training examples, N is the number of those examples used in the estimate and h is the bandwidth of the Gaussian kernel, which is denoted by a vector $[h_1, h_2, ...h_m]$.

The optimal bandwidth can be approximated using the method presented in [97]:

$$h_j = \{\frac{4}{(m+2)N}\}^{\frac{1}{m+4}}\sigma_j \tag{4.2}$$

for $j = 1, 2, ..., m$, where σ_j is the standard deviation of the j^{th} attribute and can be replaced by its training example estimator in practical implementation.

After obtaining the synthetic inputs $\{\mathbf{x}_i, i \geq 1\}$, GOSE feeds them into a trained SVM network to obtain the classification $\{y_i, i \geq 1\}$ for these inputs. This step is illustrated in the QUERYSVM procedure (see Figure 4.7). The synthetic instances $\{(\mathbf{x}_i, y_i), i \geq 1\}$ are then constructed. The trained SVM network here is treated as an *oracle* that answers the queries from $\mathbf{x}_i$.

The fourth step is SELECTINSTANCE. GOSE defines a factor n so that it is much more flexible in its ability to choose differing sizes of instances. In order to select the instances with much higher $M(\mathbf{x})$, those $\mathbf{x}_i$ $(i \geq 1)$ with the same class label

are sorted first. Suppose that there are CN classes involved in the classification problem and that the number of $\mathbf{x}_i$ for each class is $n(G)$. In this case, we choose the first n/CN instances, or the whole group of instances if $n(G) < n/CN$, to build up the synthetic training instance set S.

SELECTINSTANCE

Input: a set of instances $\{(\mathbf{x}_i, y_i), i \geq 1\}$, a parameter n indicating the number of instances used in rule extraction, and a class set A obtained from $DRAWCLASS$
Output: the instance set S
(1) **Initialize** S, $n' = \frac{n}{CN}$ where CN is the number of class in A
(2) **For** each class A_k in A, $1 \leq k \leq CN$
(3) **Find** G = the group of instances $\{(\mathbf{x}_i, y_i), where\ y_i = A_k\}$
(4) **Sort** G by the values of $M(\mathbf{x})$ in descending order
(5) $S := S \bigcup min(G(1 : n'), n(G))$, where $n(G)$ is the number instances in G.

Figure 4.5: The SELECTINSTANCE function: select the first $\frac{n}{CN}$ instances in G which have the same class label or the entire group of instances G if the size of G is no larger than $\frac{n}{CN}$ and build up an synthetic instance set S so as to apply for rule extraction

Example 4.1. Suppose that there are five training examples $\{(\mathbf{X}_k, Y_k), 1 \leq k \leq 5\}$, as shown in Figure 4.6(a). The SVM network has been trained based on these five training examples, which were given before GOSE is applied. Let $A = \{-1, +1\}$, $m = 2$, $L = [1, 1]$, $U = [10, 10]$ and $c1 = 0.01$. We aim to draw four instances so that $n = 4$. By applying Equation 4.2, h equals $[0.2714, 0.1129]$. The above example clearly illustrates the QUERYING procedure.

Firstly, GOSE initializes $T = 10$. As can be seen from Figure 4.6, 10 synthetic data $\mathbf{x}_i$, $1 \leq i \leq 10$ are made. After this, GOSE verifies whether each $\mathbf{x}_i$ is located between L and U. If not, this $\mathbf{x}_i$ is removed (such as $\mathbf{x}_9$ in Figure 4.6(b)). GOSE subsequently computes $M(\mathbf{x}_i)$ for the rest of $\mathbf{x}_i$. The results are also given in the table (shown in Figure 4.6(b)). If the value of $M(\mathbf{x}_i)$ is larger than $c1$, GOSE will retain this $\mathbf{x}_i$. In this example, data $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_6$, $\mathbf{x}_7$ and $\mathbf{x}_8$ are kept, in order to query the SVM. After calling the QUERYSVM function, the class labels are

determined. Then, for each class, SELECTINSTANCE sorts $(\mathbf{x}_i, y_i)$ in accordance with the $M(\mathbf{x}_i)$ values (see Figure 4.6(c)). Since n has laid the number of instances used in rule extraction, in this example, SELECTINSTANCE must select $\frac{n}{2} = 2$ for each class (Figure 4.6(d)). As a result, the instances in Figure 4.6(d) are obtained for rule extraction.

	X_1	X_2	Y
$\mathbf{X}_1$	1	2	$+1$
$\mathbf{X}_2$	3	4	$+1$
$\mathbf{X}_3$	2	6	-1
$\mathbf{X}_4$	3	5	-1
$\mathbf{X}_5$	10	4	-1

(a)

	x_1	x_2	$M(\mathbf{x}_i)$
$\mathbf{x}_1$	3	4	0.0227
$\mathbf{x}_2$	4	6	0.0166
$\mathbf{x}_3$	8	5	0.0098
$\mathbf{x}_4$	3	7	0.0092
$\mathbf{x}_5$	9	2	0.0024
$\mathbf{x}_6$	2	6	0.0201
$\mathbf{x}_7$	1	4	0.0184
$\mathbf{x}_8$	1	2	0.0131
$\mathbf{x}_9$	0	3	
$\mathbf{x}_{10}$	5	8	0.0014

(b)

	x_1	x_2	$M(\mathbf{x}_i)$	class
$\mathbf{x}_1$	3	4	0.0227	$+1$
$\mathbf{x}_2$	4	6	0.0166	$+1$
$\mathbf{x}_6$	2	6	0.0201	-1
$\mathbf{x}_7$	1	4	0.0184	-1
$\mathbf{x}_8$	1	2	0.0131	-1

(c)

	x_1	x_2	$M(\mathbf{x}_i)$	class
$\mathbf{x}_1$	3	4	0.0227	$+1$
$\mathbf{x}_2$	4	6	0.0166	$+1$
$\mathbf{x}_6$	2	6	0.0201	-1
$\mathbf{x}_7$	1	4	0.0184	-1

(d)

Figure 4.6: **QUERYING Example**. Table (a) is the training examples $\mathbf{X}_i$ to build up the multivariate estimator. Table (b) shows 10 synthetic data $\{\mathbf{x}_i, 1 \leq i \leq 10\}$ and their outcomes of $M(\mathbf{x}_i)$ if $\mathbf{x}_i$ locates in the input space. Table (c) shows that 5 data are kept for QUERYSVM procedure, and the synthetic instances are then generated. Table (d) shows the final result by applying SELECTINSTANCE.

4.2.2 SVM Networks

The SVM network used in GOSE is considered a *black box*. All we need to know are key input-output patterns, rather than the inner structure. This complies with

Thrun's desideratum for a general rule extraction method of *no training requirement*. Our rule extraction method does not rely on any special training procedure, and neither does it make any assumption about the network's structure. It can be applied to any SVM classifiers, (such as Sequential Minimal Optimization (SMO) [40] and DAGs-SVM [39] (see Chapter 2)), regardless of the algorithms used to construct the classifier.

The SVM networks used here carry out classification tasks, but GOSE is not limited to classification. The QUERYSVM function is denoted as follows.

QUERYSVM

Input: an input $\mathbf{x}$ and a SVM network $f(\mathbf{x})$
Output: the class label y, the distance d from $\mathbf{x}$ to the separating hyperplane
(1) $[y, d] = f(\mathbf{x})$

Figure 4.7: The QUERYSVM function: $f(\mathbf{x})$ denotes the function embedded in a trained SVM.

4.3 Clustering

Since there must be a decision boundary between individual classes, the points lying on the decision boundary could be found between pairs of instances that have different class labels. However, for a large number of training instances, if we search each pair of the instances for different classes, this may lead to high complexity and low efficiency. Hence, the step of clustering has been used to create a balance between complexity and prediction accuracy.

In GOSE, we adopt hierarchical clustering (see Chapter 2) on the synthetic training instances S. This starts by considering each individual point as a cluster and merges the clusters by measuring the distance between two clusters with the same class

labels. Since the mergence of the clusters is relevant only to those training instances with the same class, just the inputs $\mathbf{x}_i$ are involved in the distance calculation. Our approach uses one of the following linkage functions:

1. *Single linkage* uses the smallest distance between data $\mathbf{x}_i^r$ and $\mathbf{x}_j^s$ in the two clusters r and s. If the sizes of r and s are n_r and n_s, then $d(r,s) = min(dist(\mathbf{x}_i^r, \mathbf{x}_j^s)), i \in (1,...,n_r), j \in (1,...,n_s)$;

2. *Complete linkage* uses the largest distance between data $\mathbf{x}_i^r$ and $\mathbf{x}_j^s$ in the two clusters r and s, such that $d(r,s) = max(dist(\mathbf{x}_i^r, \mathbf{x}_j^s))$.

In order to reduce the randomness of the number of clusters, a *stopping criterion* is defined for the clustering process. Given q clusters $\{r_h,\ h = 1,2,...,q\}$, the classes of the clusters are identical, and the number of data in each cluster r_h is n^{r_h}. It is obvious that the mean and variance of each cluster relate to the data $\mathbf{x}_i, 1 \leq i \leq n^{r_h}$ inside this cluster. Hence, the mean m^{r_h} of each cluster r_h is:

$$m^{r_h} = \frac{1}{n^{r_h}} \sum_{i=1}^{n^{r_h}} \mathbf{x}_i$$

and the variance s^{r_h} is

$$s^{r_h} = \frac{1}{n^r} \sum_{i=1}^{n^{r_h}} (\mathbf{x}_i - m^{r_h})^2$$

Hence, the intra-cluster deviation is defined as follows:

$$s^{intra} = \sqrt{\sum_{h=1}^{q} (s^{r_h} * p(r_h))} \tag{4.3}$$

where $p(r_h) = \frac{n^{r_h}}{\sum_{h=1}^{q} n^{r_h}}$.

And the inter-cluster mean and deviation are specified in Equation 4.4.

$$m^{inter} = \sum_{h=1}^{q}(m^{r_h} * p(r_h)) \tag{4.4a}$$

$$s^{inter} = \sqrt{\sum_{h=1}^{q}[(m^{r_h} - m^{inter})^2 * p(r_h)]} \tag{4.4b}$$

Definition 4.3.1. *The stopping criterion D is the ratio between* s^{intra} *and* s^{inter}.

If $\frac{s^{intra}}{s^{inter}} > \epsilon$, then GOSE will stop merging the data further. Note that ϵ is a user-defined parameter.

	$[0,2]$	$[5,8]$	$[7,6]$	$[7,4]$
$\mathbf{x}_1 = [0,2]$	0	7.81	8.06	7.28
$\mathbf{x}_2 = [5,8]$	7.81	0	2.82	4.47
$\mathbf{x}_3 = [7,6]$	8.06	2.82	0	2
$\mathbf{x}_4 = [7,4]$	7.28	4.47	2	0

Table 4.1: Distances between two instances

Example 4.2. Consider a set of instances $S = \{([0,2],+1), ([5,8],+1), ([7,6],+1), ([7,4],+1)\}$ and $\epsilon = 0.5$. Suppose that the *single linkage* function is applied to group these instances into several clusters and that the distance function is Euclidean. The procedure of clustering on S is as follows.

1. Step 1: from Table 4.1, it can be found that $\mathbf{x}_3 = [7,6]$ and $\mathbf{x}_4 = [7,4]$ have the smallest distance. If $\mathbf{x}_3$ and $\mathbf{x}_4$ are joined together, then there are three new clusters $C1 = \{([0,2],+1)\}$, $C2 = \{([5,8],+1)\}$ and $C3 = \{([7,6],+1), ([7,4],+1)\}$. The centroids of these clusters subsequently become $m^{C1} = [0,2]$, $m^{C2} = [5,8]$ and $m^{C3} = \frac{[7+7,6+4]}{2} = [7,5]$.

2. Step 2: the intra-cluster deviation s^{intra} is calculated.

$$s^{intra} = \sqrt{[0,0] * \frac{1}{4} + [0,0] * \frac{1}{4} + [0,2] * \frac{2}{4}} = [0,1]$$

3. Step 3: we compute the inter-cluster mean and deviation,

$$m^{inter} = [0,2] * \frac{1}{4} + [5,8] * \frac{1}{4} + [7,5] * \frac{2}{4} = [4.75, 5]$$

and

$$\begin{aligned} s^{inter} &= \sqrt{[0-4.75, 2-5]^2 * \frac{1}{4} + [5-4.75, 8-5]^2 * \frac{1}{4} + [7-4.75, 5-5]^2 * \frac{2}{4}} \\ &= [2.8614, 2.1213] \end{aligned}$$

4. Step 4: since $\frac{s^{intra}}{s^{inter}} = 0.17 < \epsilon$, we can carry out the clustering on $([7,6], +1)$ and $([7,4], +1)$

5. Step 5: for the new clusters $C1 = \{([7,6], +1), ([7,4], +1)\}$, $C2 = \{([5,8], +1)\}$ and $C3 = \{([0,2], +1)\}$, $C1$ and $C2$ are found to have the smallest distance. If the clusters $C1$ and $C2$ are merged, then there will be two clusters: $C1 = \{([7,6], +1), ([7,4], +1), ([5,8], +1)\}$ and $C2 = \{([0,2], +1)\}$. The centroids of these clusters then become $m^{C1} = [6.33, 6]$, $m^{C2} = [0, 2]$, while the intra-deviation $s^{intra} = [1.6330, 2.8284]$, the inter-mean $m^{inter} = [4.7475, 5.0000]$ and inter-deviation $s^{inter} = [2.7410, 1.7321]$. Therefore, the ratio $\frac{s^{intra}}{s^{inter}} = 0.89 > \epsilon$ and GOSE stops the mergence on $C1$ and $C2$.

Figure 4.8 shows the clustering process. The process of clustering aims to provide an appropriate number of clusters for the next step, SEARCHING, so that the points found on the classification boundary are not redundant, but correct rules are still able to be extracted.

CLUSTERING

Input: a set of synthetic instances $\{(\mathbf{x}_i, y_i) : where\ y_i = A_p,\ 1 \leq i \leq n\}$,
a stopping criterion ϵ and the linkage function $d(r, s)$ where r and s denote two clusters.
Output: a set of clusters $C = \{c_j, 1 \leq j \leq q\}$,
(1) **Consider** each $\mathbf{x}_i$ as a cluster c_i, then $q = n$.
(2) **Compute** $d(c_k, c_h), 1 \leq k, h \leq q$ for each pair of clusters
(3) **Merge** the clusters into a new set of clusters C_{new}
(4) **Calculate** s^{intra} and s^{inter}
(5) **If** $\frac{s^{intra}}{s^{inter}} > \epsilon$ **then**
(6) **Stop** clustering
(7) $C = C_{new}$ and $q =$ the number of clusters in C_{new} goto (2)

Figure 4.8: Pseudo Code for the procedure of clustering

4.4 Searching

The searching step searches for and locates the points on the decision boundaries. Consider clusters $P_1, ..., P_a$, which fall into class A_1, and clusters $N_1, ..., N_b$, which fall into class A_2. In the context of the theory of SVM, we believe that there is a hyperplane lying between two different classes. GOSE uses Zhang and Liu's measure [42] to automatically look for the points on an SVM's decision boundary[4].

The idea is to consider all pairs (p, n) s.t. $p \in P_j (1 \leq j \leq a)$ and $n \in N_k (1 \leq k \leq b)$. For each p, a corresponding point n, whose distance to p is minimum, can be found in N_k. Similarly, for each n, the corresponding point p, whose distance to n is minimum, can be found in P_j as well. Suppose that a trained SVM model is given. For each pair of points (p, n), there are two outputs for both p and n respectively after calling the QUERYSVM function. The outputs are the class labels for p and n and the distance from p or n to the hyperplane. Let d_1 represent the distance from p to the hyperplane and d_2 represent the same for n. In order to find the point lying on the hyperplane, a binary search procedure (see Figure4.9) is performed

[4]Notice that for simplicity we have been considering two classes, but that our extraction algorithm is applicable to any number of classes.

on (p, n). In other words, if $|d_1 - d_2| > \varepsilon$, the mid-point q between p and n is chosen. The SVM network classifies q and computes the distance between q and the hyperplane. If the class of q equals that of p, then p is replaced by q; otherwise, n is replaced by q. The process carries on until $|d_1 - d_2| < \varepsilon$ is achieved, where ε denotes an arbitrary small number.

The procedure of searching is described in Figures 4.9 and 4.10. Figure 4.10 shows how SEARCHING works on the multi-classification problem. For each pair of points, BINARYSEARCH is called to locate the points on the separating boundary.

BINARYSEARCH

Input: a pair of points (p, n) where p and n belong to different classes, an arbitrary small number ε.
Output: a point q lying on the hyperplane
(1) **Initialize** $a_1 = p$, $a_2 = n$
(2) $[A_1, d_1] = QUERYSVM(a_1)$ and $[A_2, d_2] = QUERYSVM(a_2)$
(3) **While** $|d_1 - d_2| > \varepsilon$
(4) **Let** $q = \frac{a_1 + a_2}{2}$
(5) $[y, d_3] = QUERYSVM(q)$
(6) **If** $y == A_1$ **then** $a_1 = q$ and $d_1 = d_3$
(7) **else** $a_2 = q$ and $d_2 = d_3$
(8) **Return** q

Figure 4.9: Pseudo Code for the procedure of binary search

Example 4.3. Given a set of clusters $C1 = \{([1, 2], +1), ([1, 3], +1)\}$, $C2 = \{([2, 1], -1), ([3, 1], -1)\}$, let a trained SVM model be $f(\mathbf{x}) = x_1 - x_2 = 0$, where $\mathbf{x} = [x_1, x_2]$ and $\varepsilon = 0.01$. Consider the point $[2, 1]$ in $C2$; by using the Euclidean distance function, the nearest point to the point $[2, 1]$ is $[1, 2]$ in $C1$. After calling BINARYSEARCH, the point $q = [1.5, 1.5]$ lying on the hyperplane is obtained.

Figure 4.11 provides an example that shows the results after the *clustering* and *searching* steps. For presentation purposes, the figure only shows partial results. If no clustering is applied on the points in Figure 4.11, the searching has to be carried out 84 times. Conversely, only 28 searches are processed for the scenario in

SEARCHING

Input: a set of classes A, a set of clusters $C = \{C_{hj}, 1 \leq h \leq n^j, 1 \leq j \leq CN\}$, where CN denotes the number of class involved, n^j indicates the number of cluster for the j^{th} class and $C_{hj} = \{\mathbf{x}_i, 1 \leq i \leq n\}$, and an arbitrary small number ε.
Output: a set of points q lying on the hyperplane

```
(1)  For each A_j in A
(2)      For each A_k in A where k ≠ j
(3)          Let C^1 = {C_hj, 1 ≤ h ≤ n^j} and C^2 = {C_hk, 1 ≤ h ≤ n^k}
(4)          For each C_hj in C^1
(5)              Let C_1 = C_hj
(6)              For each x_i in C_1, q_i = x_i
(7)                  For each C_hk in C^2
(8)                      Let C_2 = C_hk
(9)                      Find the nearest point x'_i in C_2, n_i = x'_i
(10)                     Construct a pair (p_i, n_i)
(11)                     q = BINARYSEARCH((p_i, n_i), ε)
(12)                     q := q ∪ q'
```

Figure 4.10: Pseudo Code for the procedure of searching

the figure. That is to say, *clustering* could decrease the complexity of *searching*.

4.5 Extracting

A rule can be intuitively interpreted as a hypercube in the geometrical m-dimensional input space, and a set of constraints have been applied for each such hypercube.

The main idea of our rule extraction approach is to find a set of optimal rules that 1) covers the maximum area of the *area of interest* and 2) covers the largest cardinality of synthetic instances at the same time.

Suppose that there is a set of points X lying on the SVM decision boundary, where X is the result of *searching* (see Section 4.4), and a set of synthetic training instances S generated from *querying* for classes $A = \{A_p, 1 \leq p \leq CN\}$, and the SVM function $f(\mathbf{x})$.

To realize the first goal of the rule extraction algorithm, we try to solve the following

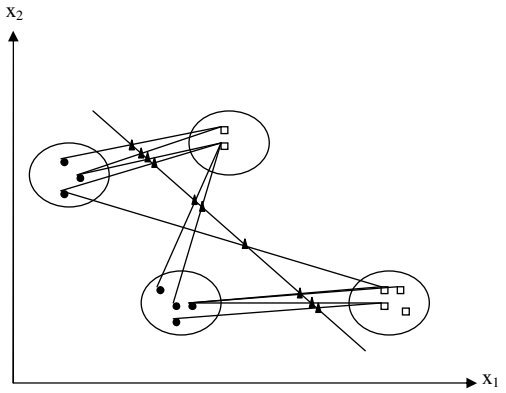


Figure 4.11: Partial results from clustering and searching.

optimization problem:

$$maximize \qquad \prod_{i=1}^{m}(x_i - \mathbf{x}_{0_i}) \tag{4.5a}$$

$$subject\ to \qquad l \leq \mathbf{x} \leq u \tag{4.5b}$$

$$\int_l^u |f(\mathbf{x}) - A_p|\, d\mathbf{x} = 0 \tag{4.5c}$$

where $\mathbf{x}_{0_i}$ denotes the i^{th} element of the vector $\mathbf{x}_0 \in X$ ($\mathbf{x}_0$ indicates a starting point of this optimization problem), x_i is the i^{th} element of $\mathbf{x}$, and l and u are the m-dimensional vectors providing lower and upper bounds to this optimization problem.

The objective function (Equation 4.5a) aims to maximize the volume of the hypercube that a rule covers, and it has two constraints. One is a bound constraint to limit the optimal $\mathbf{x}^*$ in a given area, while the other is a nonlinear constraint that is used to exclude the points that have different class labels.

The values of l and u in Equation 4.5b can be calculated based on the lower and upper bounds of the input space. For example (see Figure 4.12(a)), suppose the scope of the input space is $[L_1, L_2] \leq \mathbf{x} \leq [U_1, U_2]$, and $\mathbf{x}_0 = [\mathbf{x}_{0_1}, \mathbf{x}_{0_2}]$ is a point

lying on the SVM boundary. Note that when we change $\triangle \mathbf{x}_{0_1}$ on $\mathbf{x}_{0_1}$ or $-\triangle \mathbf{x}_{0_2}$ on $\mathbf{x}_{0_2}$ ($\triangle \mathbf{x}_{0_1}$, $\triangle \mathbf{x}_{0_2} \geq 0$) the SVM classification on $\mathbf{x}_0$ is class A_p. Hence, it is reasonable to assume that an optimal point can be found and that a rule for class A_p in a rectangle between points $\mathbf{x}_0$ and $[U_1, L_2]$ can be constructed. Here, $[U_1, L_2]$ is defined as an *orientation* for $\mathbf{x}_0$. l and u are then narrowed down to $l = [\mathbf{x}_{0_1}, L_2]$ and $u = [U_1, \mathbf{x}_{0_2}]$.

If more than one *orientation* is found for class A_p, then the principal orientations have to be selected. Let $\mathbf{v} = [v_1, v_2]$ stand for an orientation for $\mathbf{x}_0$. In the above example, $\mathbf{v} = [U_1, L_2]$. Selecting the orientations for $\mathbf{x}_0$ involves deciding how to compute the significance of each orientation. Equation 4.6 shows conditional probability estimation that GOSE uses to determine the significance of each orientation. The estimation represents the probability distribution of $\mathbf{x}$ lying in the area between $\mathbf{v}$ and $\mathbf{x}_0$, given class A_p. In Equation 4.6, $P(\min(\mathbf{v}, \mathbf{x}_0) \leq \mathbf{x} \leq \max(\mathbf{v}, \mathbf{x}_0) \bigcap class = A_p)$ indicates the probability of $\mathbf{x}$ falling into the area between $\mathbf{v}$ as well as belonging to class A_p, and $P(class = A_p)$ is the possibility that the classification of $\mathbf{x}$ equals A_p. Assume that the distribution of synthetic training examples is similar to that of the problem domain. Hence, the probabilities $P(\min(\mathbf{v}, \mathbf{x}_0) \leq \mathbf{x} \leq \max(\mathbf{v}, \mathbf{x}_0) \bigcap class = A_p)$ and $P(class = A_p)$ could be worked out from the synthetic data set. The value of $P(\min(\mathbf{v}, \mathbf{x}_0) \leq \mathbf{x} \leq \max(\mathbf{v}, \mathbf{x}_0) | class = A_p)$ is then calculated by dividing $P(\min(\mathbf{v}, \mathbf{x}_0) \leq \mathbf{x} \leq \max(\mathbf{v}, \mathbf{x}_0) \bigcap class = A_p)$ by $P(class = A_p)$. The end result is that those orientations that have the maximum probabilities are selected as the principal orientations on $\mathbf{x}$.

$$P(\min(\mathbf{v}, \mathbf{x}_0) \leq \mathbf{x} \leq \max(\mathbf{v}, \mathbf{x}_0) | class = A_p) = \frac{P(\min(\mathbf{v}, \mathbf{x}_0) \leq \mathbf{x} \leq \max(\mathbf{v}, \mathbf{x}_0) \bigcap class = A_p)}{P(class = A_p)} \tag{4.6}$$

As presented in Equation 4.5c, the nonlinear constraint is a multi-dimension inte-

gral on a linear/nonlinear function. GOSE uses a quasi-Monte Carlo method [94] to approximate the integration because it is a superior method with many advantages such as improved convergence and more uniformity. Therefore, the hypercube H is considered to be composed of the points that are uniformly distributed as:

$$\frac{1}{n}\sum_{i=1}^{n}|f(\mathbf{a}_i)-A_p| \approx \int_l^u |f(\mathbf{x})-A_p|\,d\mathbf{x}$$

where $\mathbf{a}_i$ is a low-discrepancy sequence inside the hypercube $[l,u]$, where $1 \leq i \leq n$, and n here means the number of points selected for approximation in the H. The estimation error then becomes,

$$\epsilon = \left| \int_l^u |f(\mathbf{x})-A_p|\,dx - \frac{1}{n}\sum_{i=1}^{n}|f(\mathbf{a}_i)-A_p| \right|$$

From the above, it can be shown that the larger n is, the closer the approximation approaches the integral. It is clear that the complexity[5] increases with the rise of n. Therefore, in order to strike a balance between error estimation, fidelity, accuracy prediction and complexity, a proper n has to be chosen. In the cross-validation experiments, we found $n = 1000$ as a suitable number for our benchmark datasets.

With this, the standard *pattern search algorithm* is applied to obtain a solution $\mathbf{x}^*$ to the optimization problem. Charles and Dennis analyze the generalization of the pattern search by evaluating the objective function [1]. After obtaining the optimal point $\mathbf{x}^*$, together with the starting point $\mathbf{x}_0$, the antecedents of a rule can be constructed by picking the minimum and maximum values of $\mathbf{x}^*$ and $\mathbf{x}_0$, as shown in Figure 4.13. Figure 4.12(a) gives an example of a hypercube rule with the starting point $\mathbf{x}_0$.

Finally, to find the set of rules covering all the synthetic training instances, Equa-

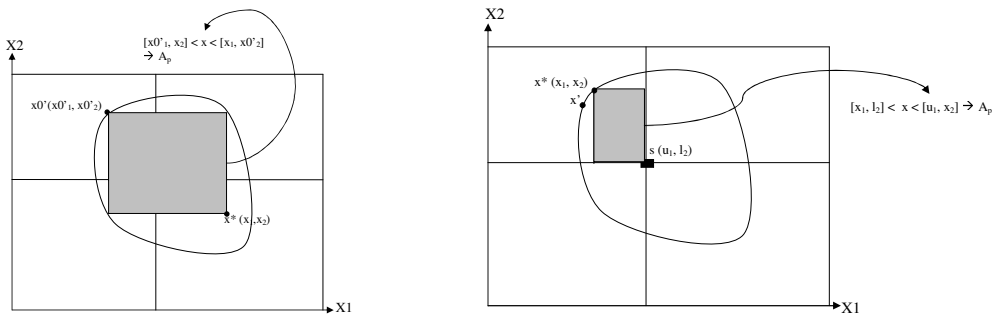
[5]The complexity issue is discussed in Chapter 6.

tion 4.5 is used again, and in it, $\mathbf{x}_0$ is replaced with $s \in S$ (see Figure 4.12(b)). The process is the same, which ensures the extracted rules cover most of the synthetic training instances as well as the maximum area of the *area of interest*.

Unlike the existing work of Fung [22], GOSE can be applied to nonlinear SVM kernels. This is because the *extracting* step of GOSE is based on the points lying on the classification boundary without consideration of the inner structure of the SVM network, and the points on the boundary are obtained through the BINARYSEARCH function without dependence on any element relevant to the SVM network. Therefore, GOSE can be applied regardless of the kernel of the SVM network.

Figure 4.13 summarizes the *extracting* algorithm and its associated rule generation algorithm, as discussed above.

The rule set obtained from *extracting* may contain overlapping rules, for which a set of post-processing measures in the next section are employed to solve this problem.



(a) (b)

Figure 4.12: *left:* extracting the rule from the starting point $[\mathbf{x}_{0_1}, \mathbf{x}_{0_2}]$ which is obtained from *Searching*; *right:* extracting the rule to cover the synthetic data and approximate the area that A_p covers, the starting point is $[s_1, s_2]$.

Extracting

Input: A set of points X on the SVM boundary obtained from the *searching* step; a set of training data S obtained from *querying* step and the class label A_p.
Output: A set of rules $R = \{r\}$,
where $r = \bigwedge l_i \leq x_i \leq u_i \rightarrow A_p$, $1 \leq i \leq m$

(1) **for each** $t \in X$
(2) **Construct** the lower and upper bound l and u by finding the orientations of $\mathbf{x}$
(3) **Apply** pattern search algorithm [1]with $\mathbf{x}_0 = t$ to obtain $\mathbf{x}^*$
(4) **Call** rule generation algorithm with parameters $\mathbf{x}^*$ and $\mathbf{x}_0$ to construct a rule r and make $R := \cup r$
(5) **for each** $s \in S$ and its corresponding $t \in X$
(6) **Construct** the lower and upper bound l and u by finding the orientations of s
(7) **Apply** pattern search algorithm [1] with $\mathbf{x}_0 = s$ to obtain $\mathbf{x}^{*'}$.
(8) **Call** rule generation algorithm with parameters $\mathbf{x}^{*'}$ and s to construct a rule r' and make $R := R \cup r'$

Rule Generation Algorithm

Input: m-dimensional points $\mathbf{x}^*$ and $\mathbf{x}_0$
Output: a rule r

(1) **Let** lower bound $l = [min(\mathbf{x}_1^*, \mathbf{x}_{0_1}), ..., min(\mathbf{x}_m^*, \mathbf{x}_{0_m})]$
(2) **Let** upper bound $u = [max(\mathbf{x}_1^*, \mathbf{x}_{0_1}), ..., max(\mathbf{x}_m^*, \mathbf{x}_{0_m})]$
(3) **Generate** $r = \bigwedge l_i \leq x_i \leq u_i \rightarrow A_p$, $1 \leq i \leq m$

Figure 4.13: Rule extraction algorithm

4.6 Post-Processing

After the initial rules are extracted through the *extracting* algorithm, GOSE employs a set of post-processing measures to obtain the final rule set. The purposes of these post-processing measures are to detect generalized rules, to prune rules with high error estimation and to construct non-overlapping rules with high coverage rate.

The notions of *non-overlapping* and *coverage rate* are defined as follows.

Definition 4.6.1. *Non-overlapping Rule: Given two rules,* $r_1 = \bigwedge a_i \leq x_i \leq b_i \rightarrow A_p$ *and* $r_2 = \bigwedge c_i \leq x_i \leq d_i \rightarrow A_p$, r_1 *and* r_2 *are said to be non-overlapping iff* $b_i \leq c_i$ *or* $a_i \geq d_i$, *for any* i, $1 \leq i \leq m$.

Definition 4.6.2. *Coverage rate is the rate between the number of testing data that are predicted correctly by a rule and the entire testing data.*

In the following sections, we will explain each post-processing step in detail.

4.6.1 Rule Extending

Given that the input space of a problem domain is from $[L_1, ..., L_m]$ to $[U_1, ..., U_m]$ and that of a rule is r which is $[l_1, ..., l_m] \leq \mathbf{x} \leq [u_1, ...u_m] \rightarrow A_p$, the rule-extending step attempts to extend r into a larger scope. At the same time, the new rule r' still satisfies the constraint that the area covered by r' belongs to the same class. To exhaustively find all the potential rules in an extended scope, a topology is used to achieve this.

Let the original value of the antecedents of r be 0 and the new value of a rule be 1. For example, if the 1^{st} dimension of the antecedents is extended to L_1, then r

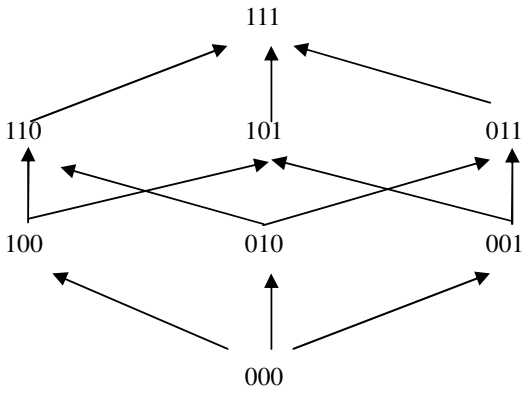


Figure 4.14: Ordering on extending to the edge of the problem domain.

becomes $l' = [L_1, l_2, ..., l_m] \leq \mathbf{x} \leq u' = [u_1, ..., u_m] \rightarrow A_p$. Hence, the new value $[L_1, u_1]$ on dimension one is regarded as 1. The definition of topology is defined as follows.

Definition 4.6.3. *Topology: An arrangement in which each element means that the value of every dimension is mapped to* 0 *or* 1 *according to the above regulations.*

Figure 4.14 shows an example of a topology where the dimension of the input space is 3.

We assume that the rule r initially constructs a rule set R. The function has $m-1$ iterations, and for each iteration, r_j in R is picked, and every dimension of r_j is extended, where $1 \leq j \leq N$ and N is the number of rules in R. At the first iteration, there is only one rule in R that is $r_j = r$, $j = 1$. Subsequently, each dimension i is extended to L_i, and the new value $[L_i, u_i]$ is verified if it satisfies the constraint $\frac{1}{n}\sum_{i=1}^{n}(|f(\mathbf{x}_i) - A_p|) = 0$. Next, for the same dimension, r is then extended to U_i, and a similar verification is performed on this new value. If there is any extension on the value of the i^{th} dimension of r, the new rule r' is kept for the next iteration. After going through each dimension of r_j, all the new rules, r', are put together for a new rule set R.

Finally, if the value of the i^{th} dimension equals the scale of the input space, which is believed to be applicable throughout the total range of the i^{th} dimension, GOSE then filters this dimension from the antecedents of the rule.

The complexity becomes exponential if the algorithm goes through every element in the topology. It increases with the rise of the dimensionality of the input space and even grows to be intractable in the worst case. Hence, in practice, an optimizing measure known as the *cracking of topology* has been adopted.

Firstly, the definition of a *clash* of the topology is given. A *clash* is an occurrence when the new region of a rule consists of the points for another class. The rule can be represented as an element in the topology.

When a *clash* is identified on a certain element, the rule-extending process would not continue on with the remaining elements that have connections with the element that has the *clash*. This is called *cracking of topology*.

Example 4.4. Consider a three-dimensional problem. The antecedents of an initial rule are interpreted as 000. It is then easy to make a structure in the order of Figure 4.14. Given such an ordering, some conclusions can be drawn. If an element in Figure 4.14 deviates from $\frac{1}{n}\sum_{i=1}^{n}|f(\mathbf{a}_i) - A_p| = 0$, then a clash would be detected, which indicates that no other element along the ordering of this element would satisfy $\frac{1}{n}\sum_{i=1}^{n}|f(\mathbf{a}_i) - A_p| = 0$ (Figure 4.15).

For instance, let a rule be $r = \{[1, 2, 2] \leq \mathbf{x} \leq [3, 4, 2]\} \rightarrow A_p$, and the input space of the problem domain be $[0, 10]$. Hence, the rule can be mapped to the topology in Figure 4.14. Initially, the lower bound of the first dimension of r is expanded to 0, and the estimated value $\frac{1}{n}\sum_{i=1}^{n}(|f(\mathbf{x}_i) - A_p|)$ of the new rule $r' = \{[0, 2, 2] \leq \mathbf{x} \leq [3, 4, 2]\}$ is examined if it equals 0. The related element in Figure 4.14 is 100. If the region covered by r' is not for one class, the new value cannot be kept. Then the 1^{st} dimension of r is extended to 10 and the

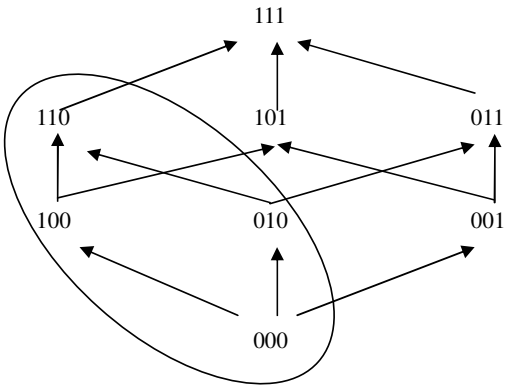


Figure 4.15: An example of the *cracking of topology*: If 001 is not a satisfiable element, then for 011, 101 and 111, the new regions of the related rules take the same sth dimension into account as 001. Hence, 011, 101 and 111 are not satisfiable elements either. GOSE will not continue to process these elements. To some extent, the rule-extending carries on only for the relevant part in the ellipse.

estimated value $\frac{1}{n}\sum_{i=1}^{n}(|f(\mathbf{x}) - A_p|)$ is checked again. If it still does not equal 0, the element 100 is considered as a cracking of topology, and the connected elements such as 110, 101 and 111 will not be processed. Finally, if one of the rules becomes $r = \{[1,0,2] \leq \mathbf{x} \leq [3,10,10]\} \rightarrow A_p$, then it can be simplified into $r' = \{1 \leq x_1 \leq 3 \bigwedge 2 \leq x_3 \leq 10\} \rightarrow A_p$, where r' is equivalent to r with a better comprehensibility.

4.6.2 Rule Pruning

The *rule pruning* stage aims to prune those rules that have a relatively large estimated error. GOSE uses a *t-test* to analyze the null hypothesis that the mean of the estimated value and the expected value of the integral of a rule r are equal, that is the mean of the estimated value equals 0.

As GOSE uses quasi-Monte Carlo method to approximate the integration function in Equation 4.5c, there is a potential error between the approximation value and the integral. Most existing studies use the Koksma-Hlawka inequality [3] to state

RULE EXTENDING

Input: $r = \{[l_1, ..l_m] \leq \mathbf{x} \leq [u_1, ..u_m]\} \rightarrow A_p$, and the scope of the input space $[L, U]$
Output: a set of new rules r'

(1) **Initialize** $clashElement = [\ ]$, $oldIndex = [\ ]$, $newIndex = zeros(m)$ and $tempR = r$
(2) **For each** $i = 1 : (m-1)$
(3) $r' = tempR$
(4) $tempR = [\ ]$
(5) $oldIndex = newIndex$
(6) **For each** r_j in r'
(7) h equals to the last location of 1 in $oldIndex(j)$
(8) **For each** $k = h + 1 \leq m$
(9) $tempIndex = [oldIndex(j)_{1...(k-1)}, 1, oldIndex(j)_{k...m}]$
(10) **If** one of elements in $clashElement$ called $cE = tempIndex$
(11) go to (8)
(12) **Extend** r_{jk} to L_k
(13) **Check** if satisfies $\int_{l'}^{u'} (|f(\mathbf{x}) - A_p|)d\mathbf{x} = 0$ by using approximation
(14) **If** satisfied,
construct a new rule r'_j with the new value on dimension k
(15) **Extend** r_{jk} to U_k
(16) **Check** if satisfies $\int_{l'}^{u'} (|f(\mathbf{x}) - A_p|)d\mathbf{x} = 0$ by using approximation
(17) **If** satisfied,
construct a new rule r'_j with the new value on dimension k
(18) **If** not satisfied for both L_k and U_k
(19) $clashElement := clashElement \bigcup oldIndex(j)$
(20) **else** $tempR := tempR \bigcup r'_j$
(21) $oldIndex(j)_k = 1$
(22) $newIndex := newIndex \bigcup oldIndex(j)$
(23) **If** $tempR$ is empty
(24) go to (26)
(25) **else** $r' = tempR$
(26) **Return** r'

Figure 4.16: RULE EXTENDING function: $oldIndex$ and $newIndex$ hold the mapping values of each rule in the topology.

the limit of the integration error.

$$\left| \frac{1}{n} \sum_{i=1}^{n} |f(\mathbf{x}_i) - A_p| - \int_L^U |f(\mathbf{x}) - A_p| \right| \leq V(f) D_N^*$$

However, it is usually difficult to calculate the total variation $V(f)$, which makes it problematic to estimate error through Koksma-Hlawka inequality.

Morohosi and Fushimi [29] introduce a statistical method for quasi-Monte Carlo error estimation. The rule pruning of GOSE is based on their method. The general scheme of the method is as follows.

Suppose a rule r with an area ranging from l to u, M data sets $\{\mathbf{x}_i^{(j)}\}_{i=1}^n$, where $j = 1, ..., M$, $l \leq \mathbf{x}_i^{(j)} \leq u$ and $\{\mathbf{x}_i^{(j)}\}_{i=1}^n$ is a set of pseudorandom data. For each data set, the value of Equation 4.7 is computed.

$$S^{(j)} = \frac{1}{n} \sum_{i=1}^{n} |f(\mathbf{x}_i) - A_P| \, , j = 1, ..., M \tag{4.7}$$

The estimate of the mean $\hat{I}$ is calculated by

$$\hat{I} = \frac{1}{M} \sum_{j=1}^{M} S^{(j)} \tag{4.8}$$

so that the error of the integral is estimated using the variance of the evaluated values.

$$\hat{\sigma^2} = \frac{1}{M(M-1)} \sum_{j=1}^{M} (S^{(j)} - \hat{I})^2 \tag{4.9}$$

Hence, the *t-test* turns out to be:

$$t = \frac{\hat{I}}{\frac{\hat{\sigma}}{\sqrt{M}}} \tag{4.10}$$

GOSE sets a significance level to specify how close the approximation value is to the expected value 0. If t is larger than the standard value at the significance level, the rule is rejected. Otherwise it is accepted.

Those rules rejecting the null hypothesis are removed. Therefore, GOSE's pruning is able to ensure that GOSE approximates the behavior of the SVM network.

RULE PRUNING

Input: a set of rules $R = \{r_i, 1 \leq i \leq n_R\}$ where n_R is the number of rules in R, and the standard value at significance level $t1$
Output: a set of pruning rule R'
(1) **Initialize** R'
(2) **For each** r_i in R
(3) **Compute** *t-test* value of r_i
(4) **If** $t < t1$ **then** $R' := R' \bigcup r_i$

Figure 4.17: RULE PRUNING function

4.6.3 Non-overlapping Rule Construction

As mentioned in the *extracting* section, there could exist overlapping rules. To remove the intersections between rules and improve the comprehensibility of rules, the characteristics of non-overlapping rules is identified, that is at least one dimension of each of two rules do not intersect with each other. For example, let r_1 be $[a_1, .., a_m] \leq \mathbf{x} \leq [b_1, ..., b_m] \rightarrow A_p$ and r_2 be $[c_1, ...c_m] \leq \mathbf{x} \leq [d_1, .., d_m] \rightarrow A_p$. If $a_i \leq c_i \leq b_i \leq d_i$, $1 \leq i \leq m$, then the overlap of r_1 and r_2 is $\{[c_1, .., c_m] \leq \mathbf{x} \leq [b_1, ..b_m]\}$. Suppose r_2 does not change and r_1 has to be divided. For each dimension i, a non-overlapping rule can be constructed in three steps:

Part 1. Keep the original value $a_j \leq x_j \leq b_j$ of r_1 for those dimensions $j < i$.

Part 2. Use the non-overlapping value $a_i \leq x_i \leq c_i$ for the dimension $j = i$.

Part 3. Use the overlapping values $c_j \leq x_j \leq b_j$ instead of the original values of r_1 for those dimensions $j > i$.

As a result, the non-overlapping rule is the concatenation of these three parts.

Example 4.5. Given two rules $r_1 = \{[1, 4, 3] \leq \mathbf{x} \leq [4, 7, 6]\} \rightarrow A_p$ and $r_2 = \{[2, 5, 4] \leq \mathbf{x} \leq [5, 9, 8]\} \rightarrow A_p$, the intersection part of these rules is $[2, 5, 4] \leq \mathbf{x} \leq [5, 7, 6]$. If $i = 2$ and r_2 remains, then r_1 should be split into three parts:

part 1. $1 \leq x_1 \leq 4$;
part 2. $4 \leq x_2 \leq 5$;
part 3. $4 \leq x_3 \leq 6$.

Then the non-overlapping rule is $[1, 4, 4] \leq \mathbf{x} \leq [4, 5, 6] \rightarrow A_p$.

4.6.4 Rule Selection

The last step of post-processing is *rule selection*. This discards those rules with zero coverage rate.

The aim of this step is to extract those rules with extensive information. In GOSE, this means that the rules predicting no data in our experiments are removed. Note that the selection does not change the predictive behavior of GOSE, it simply deletes extraneous rules (see Figure 4.19).

NON-OVERLAP

Input: a set of rules R **Output:** a non-overlap rule set RS
(1) **Initialize** $RS = [\,]$
(2) **for each** r in R
(3) **If** $RS = [\,]$ **then** $RS := RS \bigcup r$
(4) **else** for each r' in RS
(5) $r =$ Call SPLIT (r, r')
(6) $R := R \bigcup r$

SPLIT

Input: two rules $r_1 = \{[a_1, ...a_m] \leq x \leq [b_1, ..b_m]\}$ and $r_2 = \{[c_1, ..c_m] \leq x \leq [d_1, ..d_m]\}$
Output: a set of non-overlap rules R
(1) $[L, U] =$ INTERSECTION (r_1, r_2)
(2) **If** L and U are empty then return
(3) **For each** $1 \leq i \leq m$
(4) **If** $i = 1$
(5) $p1$ is empty
(6) **else**
(7) $p1 = \{[a_1, ...a_{(i-1)}] \leq [b_1, ..b_{(i-1)}]\}$
(8) **If** $a_i < L_i$ then
(9) $p2 = \{a_i \leq x_i \leq L_i\}$
(10) **else if** $b_i > U_i$
(11) $p2 = \{U_i \leq x_i \leq b_i\}$
(12) **If** $i = m$
(13) $p3$ is empty
(14) **else**
(15) $p3 = \{[L_{(i+1)}, ..L_m] \leq x_{(i+1)} \leq [U_{(i+1)}, ..U_m]\}$
(16) $r' = p1 \bigwedge p2 \bigwedge p3$
(17) $R := R \bigcup r'$

INTERSECTION

Input: two rules $r_1 = \{[a_1, ...a_m] \leq x \leq [b_1, ..b_m]\}$ and $r_2 = \{[c_1, ..c_m] \leq x \leq [d_1, ..d_m]\}$
Output: overlapping part of two rules $L \leq \mathbf{x} \leq U$
(1) **Initialize** L and U
(2) **Organize** a_i, b_i, c_i and d_i in ascending order
(3) **If** $c_i < b_i$ or $a_i < d_i$
(4) $L_i = c_i$ or $L_i = a_i$
(5) $U_i = b_i$ or $U_i = d_i$

Figure 4.18: NON-OVERLAP function

RULE SELECTION

Input: a set of rules $R = \{r_i, 1 \leq i \leq n_R\}$ where n_R is the number of rules in R
Output: a final set of rule R'
(1) **Initialize** R'
(2) **For each** r_i in R
(3) **Compute** the coverage rate cv of r_i
(4) **If** $cv > 0$ **then** $R' := R' \bigcup r_i$

Figure 4.19: RULE SELECTION function

4.6.5 Discussion

As described in the above sections, the four post-processing measures are implemented in the order of *rule extending*, *rule pruning*, *non-overlapping rule construction* and *rule selection*. Different orders could lead to different results. For example, if *non-overlapping rule construction* is carried out before *rule extending*, then new overlapping would be generated after *rule extending*. Moreover, the *rule pruning* stage reduces the workload of non-overlap rule construction. *Rule selection* has to be run after *non-overlapping rule construction*, because this step leads to the final rule set. Furthermore, it can also be run one more time before *non-overlapping rule construction* so that the workload of *non-overlapping rule construction* is reduced by filtering those rules with zero coverage. The running of *rule extending* should be ahead of *rule pruning* because *rule extending* could enlarge the area that a rule covers and might bring about large estimated errors.

4.7 Chapter Summary

This chapter has presented the GOSE algorithm which extracts conjunctive rules from SVMs. GOSE is novel in that, unlike previous SVM approaches, it treats the SVMs as *oracles*. The primary advantages of this approach are

1. It is general in its applicability.

2. It is blind to the structure of the network.

3. It is scalable to large problem domains.

Since few real-world problems are able to exhaustively enumerate all of the data, the training set obtained from most real-world problem domains is usually small. GOSE can generate synthetic training instances from a density estimator, which allows GOSE to be employed effectively in a very general manner.

A second advantage is that it uses the SVM networks as *'black boxes'*, which only answer the queries for the inputs. Therefore, this makes the approach independent of the structure of the networks. In Chapter 5, different SVM algorithms are applied to GOSE, showing that good results can still be achieved.

Another key aspect of GOSE is that it extracts rules by approximating the SVM classification. As illustrated in [50], it is not possible to extract accurate rules with perfect fidelity. GOSE therefore tries to extract conjunction rules that simulate the behavior of SVMs with small errors. It mines optimal rules, attempting to reduce the approximation error and maximize the fidelity.

Chapter 5

Empirical Evaluation of GOSE

This chapter provides an empirical evaluation of GSVMORC. The experiments presented here show that GOSE can solve a variety of classification tasks. The experimental studies presented in this chapter aim to demonstrate the influence of the parameters of GOSE and investigate the quality of acquired rule sets. We conduct two types of experiments.

Experiment Type I: to evaluate GOSE with various sizes of synthetic training instances under the same number of clusters.

Experiment Type II: to evaluate GOSE with various numbers of clusters under the same size of synthetic training instances.

Six criteria are used to evaluate the performance of GOSE, namely: accuracy, fidelity, consistency, comprehensibility, scalability and generality. Section 5.1 briefly explains these evaluation criteria.

5.1 Evaluation Criteria

GOSE is evaluated from the following six aspects.

1. Accuracy: GOSE should have an accurate prediction on unseen examples.
2. Fidelity: GOSE should produce symbolic representations that can precisely simulate the behavior of the learning systems.
3. Consistency: GOSE should produce the same classification on unseen examples under various training sessions.
4. Comprehensibility: GOSE should generate symbolic representations that are easily understandable to users.
5. Scalability: GOSE should be scalable to deal with large hypothesis spaces.
6. Generality: GOSE should require no specific training data or network architecture.

The evaluation presented in this chapter mainly concerns the first four aspects, which comprise rule equality. Each of the rule quality measures is described in detail in Section 5.2.

5.2 Rule Quality

The quality of the extracted rules is regarded as the most important element for rule extraction algorithms [38]. Accuracy, fidelity, consistency and comprehensibility are the individual aspects of rule quality.

Given a set of unseen examples $\{(\mathbf{x}_1, y_1), ...(\mathbf{x}_N, y_N)\}$, y_i^{BB} and y_i^{WB} indicate the predictions made by SVM networks (Black Box) and extracted rules (White Box) [38] where $1 \leq i \leq N$. The following equations then denote whether y_i^{WB} equals its original label y_i or its corresponding SVM label y_i^{BB}.

$$n^{ow} = \begin{cases} 0, & if \quad y_i \neq y_i^{WB}; \\ 1, & if \quad y_i = y_i^{WB}. \end{cases} \tag{5.1}$$

$$n^{bw} = \begin{cases} 0, & if \quad y_i^{BB} \neq y_i^{WB}; \\ 1, & if \quad y_i^{BB} = y_i^{WB}. \end{cases}$$

Hence, for classification problems, the *accuracy* of rule extraction is expressed as the percentage of unseen examples that are correctly classified. *Fidelity* is usually defined as the percentage of unseen examples whose y_i^{BB} and y_i^{WB} are the same.

$$accuracy = \frac{\sum_{i=1}^{N} n^{ow}}{N} \tag{5.2a}$$

$$fidelity = \frac{\sum_{i=1}^{N} n^{bw}}{N} \tag{5.2b}$$

Comprehensibility is another important factor to determine the rule quality. The specific measures that we use to assess the comprehensibility of GOSE are

(i) the number of rules to predict unseen examples and

(ii) the average number of antecedents for a single rule.

For a given symbolic interpretation, fewer and simpler rules are more comprehensible than complex rules.

Another desirable quality of the GOSE algorithm is its consistency. There are several definitions of this concept. In [84], it is defined as consistent if similar rules can be produced on the same data set under differing training sessions, while [66] considers an algorithm to be consistent if two extracted rule sets produce the same classification of unseen examples.

The two definitions above concentrate either on the similarity of predictions of the extracted rules or on the rules themselves. As the main motivation of rule extraction is to obtain a comprehensible interpretation with high accuracy and fidelity, this aspect of rule quality is often overlooked. We believe that this is mainly owing to a lack of a straightforward definition of similarity. We therefore now extend the method in [38] to make the definition of *consistency* oversee both aspects of similarity.

Given two rule sets A and B, we deem that a measure of consistency between two rule sets denoted by σ has the following properties:

- σ consists of two components: 1) the similarity between the predictions of A and B; 2) the similarity between A and B themselves. $\sigma(A, B)$ adds these two components together.

$$\sigma(A, B) = \frac{1}{2}(\sigma^{pred}(A, B) + \sigma^{rule}(A, B))$$

- If A is consistent with B, then B should also be consistent with A.
- If two rule sets A and B are exactly the same, the predictions of A and B are also the same. Hence, σ becomes maximum, i.e.

$$\sigma(A, A) = 1.$$

- If two rule sets provide completely different classifications for each unseen example, in that case, σ should be minimal

$$\sigma(A, \overline{A}) = 0.$$

$\sigma^{pred}(A, B)$ equals the rate between the number of examples correctly predicted by both A and B and the number of examples correctly predicted by either A or B.

$$\sigma^{pred}(A, B) = \frac{N_{A \cap B}}{N_{A \cup B}} \tag{5.3}$$

Next, $\sigma^{rule}(A, B)$ is calculated by using the algorithm proposed in [38]. At first, for each rule r in A or B, the number of examples that are classified by r is summed up and denoted as N_r. If a rule $r \in A$ or B covers n_r examples, a rule r' in another rule set can be found to predict most of these n_r examples that have the same class as r. The number of the examples classified by r' is then denoted as N_r^{other}. So,

$$\sigma^{rule}(A, B) = \frac{\sum_{r \in A} N_{r'}^{other} + \sum_{r \in B} N_{r'}^{other}}{\sum_{r \in A} N_r + \sum_{r \in B} N_r} \tag{5.4}$$

As a result,

$$\sigma(A, B) = \frac{1}{2}\left(\frac{N_{A \cap B}}{N_{A \cup B}} + \frac{\sum_{r \in A} N_{r'}^{other} + \sum_{r \in B} N_{r'}^{other}}{\sum_{r \in A} N_r + \sum_{r \in B} N_r}\right). \tag{5.5}$$

Example 5.1. Table 5.1 is an example used to clarify the meaning of consistency. In Table 5.1, the number of examples correctly classified by both A and B is nine while ten examples are correctly predicted by one of these two rule sets. $\sigma^{pred}(A, B)$ thus equals 0.9. The second step is to calculate the similarity between the rule sets themselves. The following gives the details of the calculation on $\sigma^{rule}(A, B)$. For instance, in Table 5.1, the rule $A_2 \in A$ predicts five examples correctly, so that

Observation	Prediction (A)	Rule (A)	Prediction (B)	Rule (B)	Real Value
1	Yes	A1	Yes	B1	Yes
2	No	A3	Yes	B3	No
3	Yes	A1	Yes	B1	Yes
4	Yes	A2	Yes	B1	Yes
5	Yes	A2	Yes	B1	Yes
6	Yes	A2	Yes	B2	Yes
7	Yes	A2	Yes	B1	Yes
8	No	A3	No	B3	No
9	No	A3	No	B3	No
10	Yes	A2	Yes	B2	Yes

Table 5.1: An example of consistency

$N_{A1} = 2 \Rightarrow N_{A1}^{other} = 2$	$N_{B1} = 5 \Rightarrow N_{B1}^{other} = 2$
$N_{A2} = 5 \Rightarrow N_{A2}^{other} = 3$	$N_{B2} = 2 \Rightarrow N_{B2}^{other} = 2$
$N_{A3} = 3 \Rightarrow N_{A3}^{other} = 2$	$N_{B3} = 2 \Rightarrow N_{B3}^{other} = 2$

Table 5.2: Similarity between A and B

$N_{A2} = 5$. For each rule in B that predicts those five examples with the same class as A_2, the rule B_1 finds the highest number. Hence, $N_{A2}^{other} = 3$. The result of comparing A and B is shown in Table 5.2.

Then

$$\sigma^{rule}(A, B) = \frac{7 + 6}{10 + 9} = 0.68$$

Finally,

$$\sigma(A, B) = \frac{1}{2}(0.9 + 0.68) = 0.79 = 79\%$$

It is not easy to give an intuitive meaning to this number. However, to some extent, it illustrates the degree of similarity between A and B. We believe that the high consistency leads to a better overall performance of the rule extraction algorithms.

5.3 Monk's Problems

This section evaluates GOSE in relation to the Monk's problems [77]. These problems are used as benchmarks for performance comparison between a variety of symbolic and non-symbolic learning techniques [81] such as ID5R [37] and the AQ [90] program. These algorithms provide a baseline for evaluating the quality of the extracted rules from GOSE.

All of the three Monk's problems have seven attributes, which include an Id feature for each instance. The other attributes are categorical, labelled as $a1, a2, a3, a4, a5, a6$. All instances in the Monk's problems are divided into two classes: $class1 = 0$ and $class2 = 1$.

Monk-1 trains an SVM network with 124 examples selected from 432 instances, Monk-2 trains a network with 169 examples selected from 432 instances and Monk-3 trains a network with 122 examples out of 432 instances. Note that 5% class noise is added to the training set of Monk-3.

5.3.1 Algorithm

For Monk-1 and Monk-3, the SVM is trained using Sequential Minimal Optimization (SMO) algorithm [40]. The kernel of the SVM is a radial basis function with differing length scales for each input. The kernel is given as follows.

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\sum_{i=1}^{m}((x_i - z_i)^2 * \tau_i)}{s}) \tag{5.6}$$

where τ is an m-dimension vector holding the length scales for the inputs. The length scale for each dimension used for the SVM network is chosen as $(9, 7, 3, 0, 5, 2)$ and (9, 9, 3, 0, 5, 2) for Monk-1 and Monk-3 respectively. This is because that

the SVM network is capable of obtaining the highest prediction accuracy at those values.

For Monk-2, a multi-class one-against-one algorithm with a Gaussian kernel is applied. The one-against-one algorithm is also known as pairwise coupling [82]. Equation 5.7 is chosen as the kernel for SVM training.

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}) \tag{5.7}$$

where the bandwidth σ is set to 3.3. This is because a high forecasting accuracy is achieved by the SVM network at this value.

In experiments, synthetic instance sets have firstly to be generated for all Monk's problems. The synthetic instances are those whose outcomes of the density estimator $M(\mathbf{x})$ (see Chapter 4) are larger than a threshold $c1$. $c1$ is one of the parameters of the experiments. For a high-dimension input space, the likelihood of data appearing in a specific part of the space is small. Hence, $c1$ is set at a relative small value, i.e. 0.000001. Subsequently, after the *querying* step, GOSE picks N instances $\{(\mathbf{x}_i, y_i), 1 \leq i \leq N\}$ with the highest probability. N is a user-defined parameter. This allows GOSE to optionally adjust the size of the training set for rule extraction. Here, for Experiment Type I, N is set as 0, 20, 40, 80 and 100 for Monk-1 and Monk-3, and $N = 0, 50, 100, 150, 200$ for Monk-2.

Another parameter of GOSE is the stopping criterion D for *clustering*, which is employed to decide the number of clusters. In Experiment Type I, in order to assess the rule quality under the fixed number of clusters, the instances for each class are deemed to fall within one cluster. Hence, the stopping criterion is not used in this part. For Experiment Type II, diverse values are set for D to partition the instances into a certain number of clusters. For all Monk's data sets, if D is equal to the relevant values in Table 5.4, then this enables the synthetic instance

set to be divided into 1, 2, 3, 4 and 5 clusters correspondingly.

Other parameters which need to be considered are the significance level of the *t-test* in *rule pruning*, the number of the random data used in *querying* to choose the training instances for the rule extraction algorithm and the points used for quasi-Monte Carlo integration. The significance level of the *t-test* is set to 0.05, which is a standard value used in statistics. For all the domains in this chapter, the value of the first parameter is set to 100,000. We consider $100,000$ to be a number from which enough synthetic training instances can be generated for rule extraction.

The third parameter is set to 1000, We choose 1000 as the value of the third parameter because the lower bound of the convergence of quasi-Monte Carlo integration for 1000 is $O(1/N) = O(0.0001)$, which is an acceptable rate. Although, when the dimensionality increases, the upper bound of the convergence (maximum error) $O(\ln((\ln N)^s/N))$ becomes worsens, the experiments show that it is more common to obtain results closer to the best rate of convergence than to the theoretical worst case. In the experiments, $N = 1000$ makes a good balance between complexity and rule quality.

At the post-processing stage, the rule selection is run twice: first before the non-overlapping rule construction and then after it. It is believed that the first rule selection process could greatly decrease the complexity of the non-overlapping rule construction and that the second time could result in a comprehensible rule set.

5.3.2 Results

Figures 5.1, 5.2 and 5.3 show the results of accuracy on Monk's problems. Each of these values represents the average values of a three-fold cross-validation run. As the training and test sets for Monk's problems already exist, the cross-validation

run for Monk's problems is defined as follows:

A. Train the SVM network with the original training set.

B. Generate three synthetic training sets that have the same size and different instances.

C. Extract rules by using these three synthetic training sets and average the results.

Experiment Type I

It can be seen that, with the increase of N, the accuracy of GOSE also increases and converges to the accuracy of SVM. Figure 5.1 shows that, when N reaches 100, GOSE achieves 100% accuracy for Monk-1. Conversely for Monk-2, GOSE attains 84.8% for a 200-size training set compared with the 85.7% accuracy classified by the SVM network (see Figure 5.2). It is shown in Figure 5.3 that GOSE achieves an average 95% correctness for a 100-size training set of Monk-3, while the SVM network attains around 94% accuracy.

Table 5.3 shows the fidelity of Monk's problems to their respective SVM networks. The results of the fidelity are also the average of the cross-validation runs, similar to accuracy. From the table, it can be seen that the fidelity of Monk-1 is 100%, while Monk-2 and Monk-3 obtain 99% and 98.5% fidelity.

The results of accuracy and fidelity demonstrate that GOSE could predict unseen examples with a high correctness as well as present a good approximation to the behavior of SVMs.

The results in Table 5.3 show that GOSE has a high consistency on all Monk's problems when GOSE uses the same synthetic training set to extract symbolic

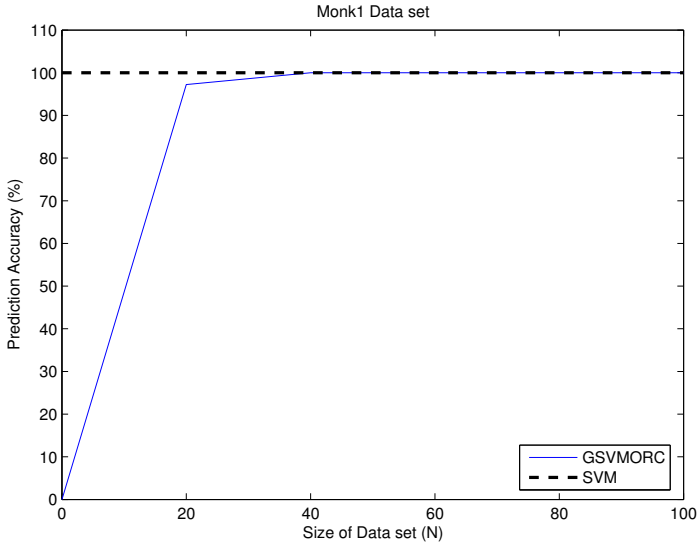


Figure 5.1: The accuracy of Monk-1 under different data size comparing with that of SVM

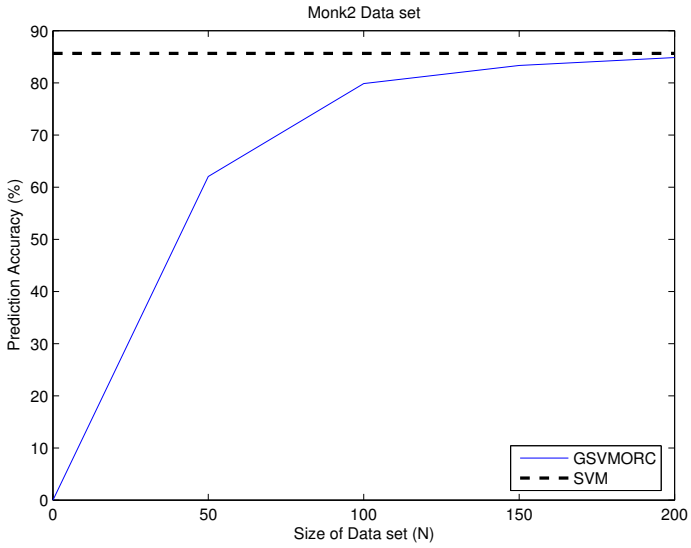


Figure 5.2: The accuracy of Monk-2 under different data size comparing with that of SVM

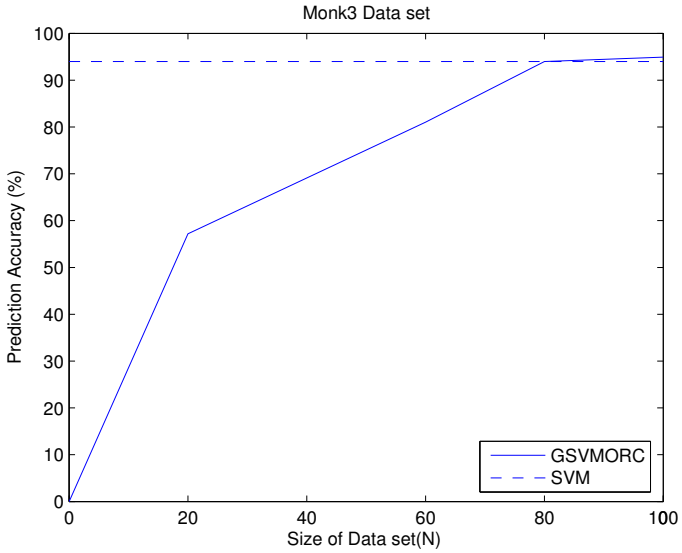


Figure 5.3: The accuracy of Monk-3 under different data size comparing with that of SVM

rules several times. However, if GOSE uses different synthetic training sets to classify the same unseen examples, the average similarities between two extracted rule sets are 100% for Monk-1 and Monk-3, and 88.1% for Monk-2. These results indicate that the high accuracy and fidelity of GOSE are not random events but happen in a consistent way.

The following are some of the extracted rules for Monk's problems.

For Monk-1, GOSE obtained four rules for all classes. On average, each rule has 2.37 conditions. This is from the 100 synthetic training instances, which cover 100% of the test cases.

$$a_1 = 1 \bigwedge a_2 = 1 \rightarrow 1$$

$$a_5 = 1 \rightarrow 1$$

The rules above have a high coverage, as the rule $a_1 = 1 \bigwedge a_2 = 1 \rightarrow 1$ covers 96

	Fidelity	Consistency (same synthetic train set)	Consistency (different synthetic train set)
Monk-1	100	100	100
Monk-2	99.12	100	88.1
Monk-3	98.5	100	100
Iris	100	100	90
Breast Cancer	100	100	84

Table 5.3: Test-set fidelity(%) and consistency for Monk's, Iris and Breast Cancer problems

out of 308 instances in the test set. The complete rule set is given in Appendix A.

For Monk-2, around 38 rules are extracted, with around 4.1 conditions per rule for class 1. For class 0, 24 rules with 5.8 conditions per rule are extracted. The following are examples of the rules. The complete rule set is also shown in Appendix A.

$$a_1 = 1 \bigwedge a_2 = 1 \bigwedge a_6 = 1 \rightarrow 1$$

$$a_1 = 1 \bigwedge a_3 = 1 \bigwedge a_4 = 1 \rightarrow 1$$

For Monk-3, 11 rules are extracted, with around 3.4 conditions per rule for class 1, while 6 rules with 2.7 conditions per rule are extracted for class 0. The following are examples of the rules. Again, the complete rule set is shown in Appendix A.

$$a_2 = 1 \bigwedge a_5 = [1, 2] \rightarrow 1$$

$$a_5 = 4 \rightarrow 0$$

Experiment Type II

The value of D rises with the merging of the clusters. When the number of clusters decreases, the intra-cluster variance of each cluster increases, since the new in-

stances included in the cluster have relatively farther distances than those already inside the cluster. Table 5.4 shows this fact.

data set	size	class	1 *cl*	2 *cl*	3 *cl*	4 *cl*	5 *cl*
Monk-1	20	0	Inf	3.6	3.5	3.4	3.0
Monk-1	20	1	Inf	0.54	0.37	0.24	0.17
Monk-2	50	0	Inf	1.69	1.58	1.41	1.32
Monk-2	50	1	Inf	1.77	1.62	1.43	1.12
Monk-3	20	0	Inf	0.73	0.73	0.37	0.25
Monk-3	20	1	Inf	1.5	0.43	0.18	0.13
Iris	30	1	Inf	0.79	0.64	0.35	0.19
Iris	30	2	Inf	2.53	1.66	1.05	0.81
Iris	30	3	Inf	2.47	0.72	0.5	0.42
Breast Cancer	30	1	Inf	97.55	86.35	77.50	70.42
Breast Cancer	30	-1	Inf	121.18	106.63	97.97	90.84

Table 5.4: The values of D for different number of clusters, where *cl* is the abbreviation of cluster

We believe that changing D, that is changing the number of clusters, will affect the accuracy of classification. Figures 5.4, 5.5 and 5.6 show that the prediction accuracy increases with decreasing D.

In addition, more than 90% of all the extracted rules after *rule extending* satisfy the hypothesis in the *rule pruning* phase, i.e. the approximation value is close to the expected value of the integral. This fact therefore indirectly demonstrates that the estimated error of quasi-Monte Carlo integration in the experiments conducted in this thesis is closer to the best convergence rate rather than to the theoretical worst case.

5.4 IRIS problem

This section investigates the use of GOSE to describe an SVM network on the Iris plant problem [77]. Iris is one of the best-known data sets in the classification literature.

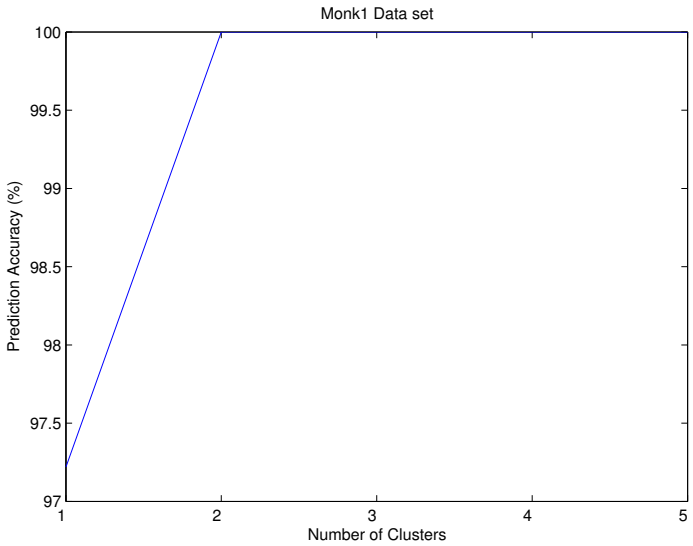


Figure 5.4: The prediction accuracy of Monk-1 by changing the stopping criterion D so that the number of cluster increases from 1 to 5. The $N = 20$ for this figure.

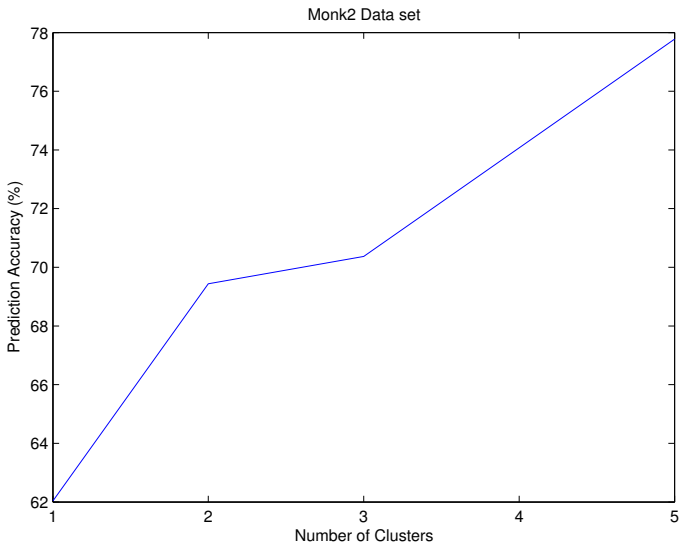


Figure 5.5: The prediction Accuracy of Monk-2 by changing the stopping criterion D so that the number of cluster increases from 1 to 5. The $N = 50$ for this figure.

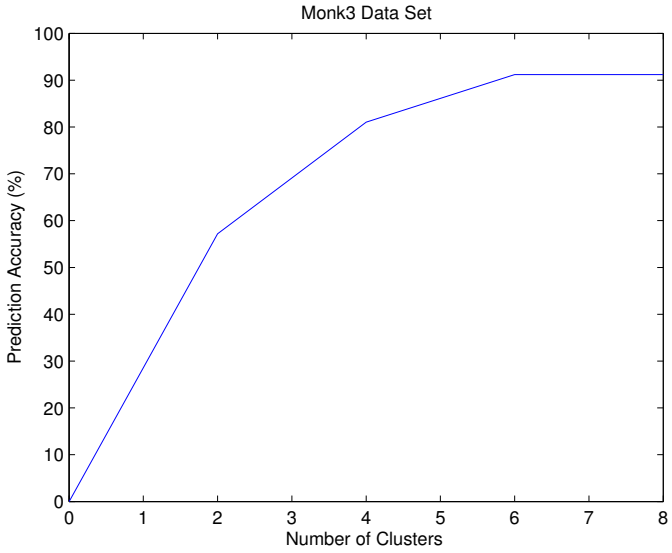


Figure 5.6: The prediction Accuracy of Monk-3 by changing the stopping criterion D so that the number of cluster increases from 1 to 5. The $N = 20$ for this figure.

	Lower Bound	Upper Bound
sepal length	4.3	7.9
sepal width	2	4.4
petal length	1	6.9
petal width	0.1	2.5

Table 5.5: The input space scope of Iris problem domain

The data for Iris consists of four continuous attributes: *sepal length, sepal width, petal length and petal width.* There are three plant classes: *Iris Setosa, Iris Versicolour and Iris Virginica.*

There are 150 instances in the data set. The input space of Iris is summarized in Table 5.5.

5.4.1 Algorithm

The trained network in Iris experiments uses a multi-class one-against-all algorithms with Gaussian kernels, called DAGSVM (Directed Acyclic Graph SVM) [39].

It constructs three classifiers, one for each pair of classes. The prediction results are obtained along with the DAGSVM method which is explained in Chapter 2.

Experiments on the Iris problem are conducted in a similar way to those on Monk's problems. They study the rule quality, such as accuracy and fidelity, on various sizes of the data sets and on the number of clusters.

A number of the synthetic instances are drawn at the constraint $c1$ within the input space of the Iris problem domain. Here, $c1$ is also set as 0.00001 which is the same as that in Monk's problems. GOSE then chooses N data from these synthetic instances, for rule extraction. N is set to 0, 30, 100, 200 and 300 in Experiment Type I for Iris.

Table 5.4 summarizes the intra-cluster errors for various numbers of clusters. For Experiment Type I, to test the accuracy and fidelity on the same number of clusters, the number of clusters D is fixed to 1, while in Experiment Type II, it is set to the corresponding values in Table 5.4.

All other parameters and processes are the same as in the Monk's problems.

5.4.2 Results

It is shown in Figure 5.7 that the prediction accuracy of the Iris problem reaches 89.33% when the accuracy of SVM is 91.33%. This value represents the average from a five-fold cross-validation run. The following outlines the process of the cross-validation run for the Iris problem domain.

a. Divide the data set into five parts, and select four of them as the training set and the rest as the testing set.

b. Train the SVM network with the training set.

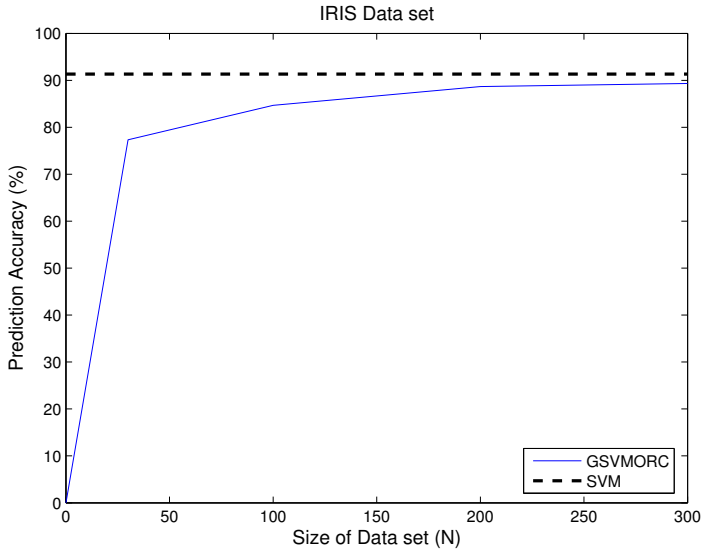


Figure 5.7: The accuracy under different size of data set

c. Generate a synthetic data set that has N instances, to extract rules.

d. Extract the rules using GOSE.

e. Predict the test set.

f. Loop five times and average the results.

Experiment Type I

It can be seen from Figure 5.7 that the prediction accuracy increases with increasing N. When N equals 30, the accuracy is only 77.33%. However, 84.67% is achieved when N equals 100. It finally reaches 89.33% at $N = 300$, which is a value near to the accuracy of SVM.

Table 5.3 shows the fidelity of the GOSE on the Iris problem. The result of fidelity is also obtained from the average of the cross-validation run reaching 100% in our experiments.

It can also be seen in Table 5.3 that GOSE has a high consistency on the Iris problem. For different sessions on the same test set, 100% consistency is gained on the same synthetic data set, and 85% consistency is gained on different synthetic data sets with the same unseen examples. The results of accuracy, fidelity and consistency demonstrate that GOSE has a good performance on the IRIS problem.

The following is an example of the extracted rules. GOSE obtains on average ten rules for each class, with four conditions per rule.

$$\textit{sepal length}= [4.3, 6.6] \bigwedge \textit{sepal width}= [2.0, 4.0] \bigwedge \textit{petal length}= [2.7, 5.0] \bigwedge \textit{petal width}= [0.4, 1.7] \rightarrow \textit{Iris Versicolour}$$

The rule above correctly predicts 45 out of 150 instances in the data set. Overall, 93% training and test examples in the Iris data set are predicted correctly. The details of the extracted rules are presented in Appendix A. From the above example, it can be seen that, although post-processing is applied for rule simplification, in some cases, the antecedents of each rule are not reduced significantly, owing to the irregularity of the nonlinear classification boundary.

Experiment Type II

Table 5.4 shows that the intra-cluster error decreases when the number of clusters increases. Figure 5.8 shows that GOSE is able to predict much more instances when the number of clusters increases. For example, GOSE classifies only 56% instances correctly when the cluster number is one. But it predicts 84% instances correctly when the number of clusters goes to six. It is interesting to note that the value of 84% is close to the accuracy of 89.33% when the train set contains 300 instances and has one cluster for each class.

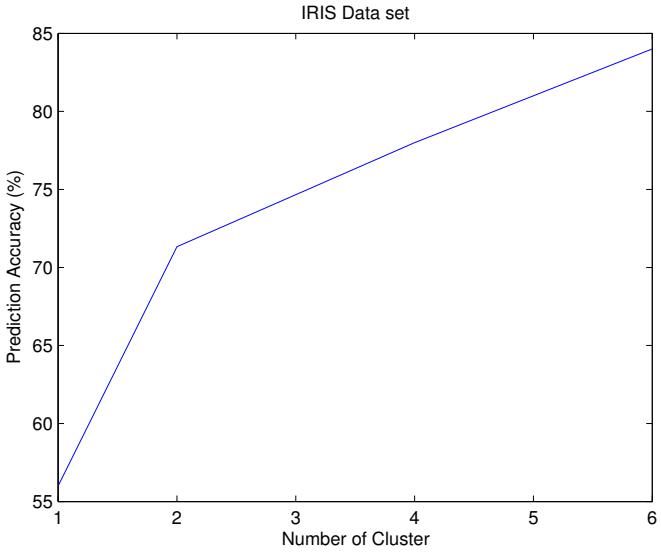


Figure 5.8: The association between the cluster number and prediction accuracy

5.5 BREAST CANCER problem

The Breast Cancer problem, which diagnoses the presence of breast cancer disease in patients, is another important benchmark in the classification domain. The data set contains 699 instances that have nine discrete attributes and a class attribute. The discrete attributes are: *Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli and Mitoses.* The class attribute has two values: 1 indicates benign and -1 malignant. The input space of the problem domain ranges from 1 to 10 for each attribute. There are 16 instances with missing attribute values.

5.5.1 Algorithm

We train the SVM using the SMO algorithm, which is similar to the one used for Monk-1 and Monk-3. The kernel of the SVM is a radial basis function with a

common length scale for each input. The common length scale is set to 4 in this experiment in order to achieve the highest prediction result.

For the Breast Cancer problem, experiments are divided into Experiment Type I and Type II to assess the quality of rules extracted from GOSE. These two types of tests are both conducted using a five-fold cross-validation methodology. The process is analogous to that for the Iris domain.

We select $N = 0, 50, 100, 150, 200$ synthetic instances for Experiment Type I, while the number of clusters is set to one so that D is fixed. On the other hand, in Experiment II, N is fixed to 30, while D is set to the relevant values in Table 5.4.

The other parameters and processes are the same as the ones used in the evaluation for Monk's problems.

5.5.2 Results

In Figure 5.9, it shows that the prediction accuracy obtained for the Breast Cancer problem is 90.14% compared with the 94.42% SVM accuracy when $N = 200$. This is the average of the five-fold cross-validation run. The process of the cross-validation run is the same as that in the Iris problem domain.

Experiment Type I

It is clear that the prediction accuracy increases from $N = 50$ to $N = 200$ (see Figure 5.9). When N equals 50, the accuracy is only around 60%. Although the rate of the increase reduces, it still causes the accuracy result to finally reach 90.14% at $N = 200$.

Table 5.3 also shows the fidelity of GOSE on the Breast Cancer problem. These

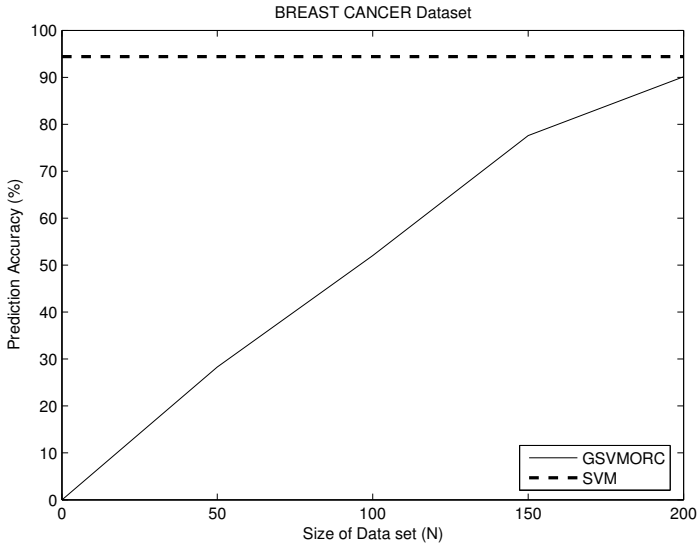


Figure 5.9: The accuracy under different size of data set

are the results from the average of the cross-validation run. The fidelity is 100% in our experiments.

Consistency is another measure with which to evaluate the rule quality. A high consistency is shown, also in Table 5.3. For various sessions on the same synthetic data set, 100% consistency is obtained. At the same time, GOSE achieves a consistency of 84% on a variety of synthetic data sets.

The following is an example of the extracted rules. GOSE obtained on average 26 rules for class 1 and 81 rules for class -1, with an average 7.2 conditions per rule. The final rule set classifies 90.14% of the test cases and 93% of the whole data set correctly.

$$a_3 = [4, 9] \bigwedge a_5 = [3, 9] \bigwedge a_6 = [10, 10] \bigwedge a_7 = [5, 9] \rightarrow -1$$

The details of the rules are presented in Appendix A. In the experiments, we also found that, if the distribution of the problem domain is diverse, then the number of

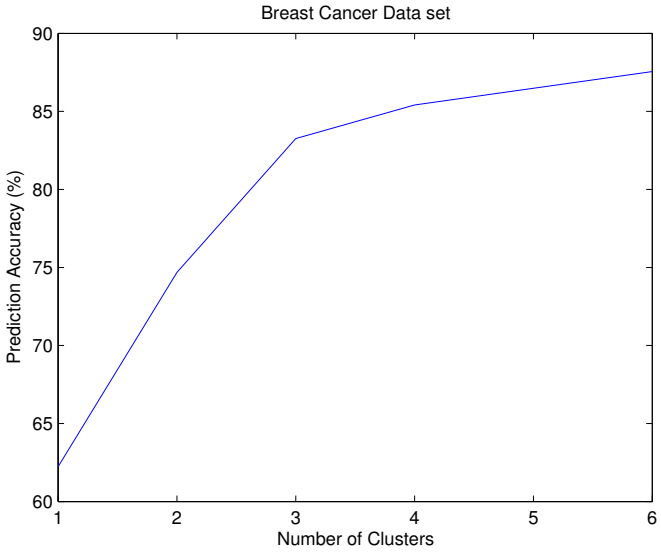


Figure 5.10: The accuracy under different size of data set

rules increases. In order to improve the comprehensibility of GOSE, future research is needed in this area.

Experiment Type II

It also can been seen in Table 5.4 that the intra-cluster error decreases when the number of clusters increases. Meanwhile, many more instances are predicted if more clusters are constructed. For instance, an accuracy of only 62.23% accuracy is gained when each class has one cluster, but 87.55% of instances are predicted when the number of clusters goes to six.

5.6 Comparison

We have shown how rules can be extracted from SVMs without needing to make many assumptions about the architecture, initial knowledge and training data set.

We have also demonstrated that GOSE is able to approximate and simulate the behavior of SVM networks correctly.

The accuracy and the fidelity of our algorithm are better than those obtained by the SVM rule extraction approach proposed in [59], which is an important work on rule extraction from SVMs. GOSE obtains 100% accuracy for Monk-1, while the SVM+prototype [31] predicts only 59.49% of instances correctly in the test set. Compared with the 63.19% test-set performance by the rule base and the 82.2% SVM classification rate, GOSE achieves 84.8% accuracy on the test set while the classification of SVM is 85.7%. GOSE also obtains 100% fidelity, but the SVM+prototype has just 92.59% and 75.95% agreement with SVM networks in the respective data sets. (Note that the performance measures for the SVM+prototype and other techniques originate from published papers and not our own experiments [31].)

In the Iris problem, the SVM+prototype [59] reports an accuracy rate of 71% for interval rules and a fidelity rate of 97.33% compared with the 96% accuracy of RBF SVM networks [60]. Our algorithm achieves a maximum fidelity rate (100%) with a far higher accuracy (89.33%), while SVM accuracy is 91.33%.

In the Breast Cancer problem, the *ExtractRules-PCM* approach of [22] achieves an average accuracy of 98% compared with an SVM accuracy of 95%. In contrast, our extraction algorithm shows high agreement between the rules and the SVM. A good fidelity indicates that the rule extraction method mimics the behavior of SVM networks, and a better understanding of the learning process is therefore obtainable.

In the Breast Cancer problem, the eclectic approach of [58] achieved 82% accuracy compared with an SVM accuracy of 95%. Our approach performs better than that approach. It achieves 89% accuracy which is more closer to the SVM accuracy

(94%).

In RuleExSVM [98], another vital algorithm for SVM rule extraction, rules are extracted based on the SVM classification boundary and support vectors. RuleExSVM has high rule accuracy and fidelity in the Iris and Breast Cancer problems. For example, for the Iris problem, RuleExSVM achieves 98% relative to the 97.5% of the SVM classification results, and in the Breast Cancer domain, 97.8% accuracy is obtained. The fidelity levels of these two domains range from 99.18% to 99.27%. However, RuleExSVM constructs the rules largely depending chiefly on the training samples and support vectors. It is difficult to apply to networks other than SVM networks. On the other hand, GOSE has a high level of generality in a wide array of networks.

Finally, on the issue of comprehensibility, *NeuroRule* [71], an approach to pruning neural networks and using decompositional extraction, produces five rules with 4.2 conditions per rule in the Breast Cancer domain compared to 107 rules with 7.2 conditions per rule in our approach. However, *NeuroRule* relies on special training procedures that facilitate the extraction of the rules. GOSE, on the other hand, is architecture-independent and has no special training requirements. It offers the highest fidelity rate and an interesting convergence property, as illustrated in the figures above.

5.7 Discussion

Like most rule extraction approaches, GOSE produces rules in the form of interval rules. However, there are a couple of advantages to using interval rules as symbolic interpretation. First, the interval rule is easily understood by users. Given a rule: $a_5 = 1 \rightarrow class = 1$, it is straightforward to determine the ranges of the

antecedent of the rule, which help users to quickly classify unseen examples. From this example, it can also be seen that a_5 is a critical element in the classification. A second advantage is that the geometric representation of the interval rule. This helps GOSE to approximate the behavior of the SVM.

The experiments in this chapter illustrate that GOSE is capable of obtaining rules which simultaneously have a high fidelity and accuracy, which is desirable because it avoids the scenario known as *fidelity-accuracy dilemma*[1] [103]. For example, the research by Zhou [103] shows that rule extraction algorithms can generalize better than their trained networks. One issue that these algorithms encounter is that, if they try to improve the fidelity of the extracted rules, the accuracy might become moderate. In contrast, GOSE realizes two goals: to generate an accurate learning system and to replicates the behavior of the trained SVM networks simultaneously.

Another interesting aspect of GOSE observed in the experiments is that, when the number of clusters increases or the data-set size increases, the gap between the prediction accuracy of GOSE and that of SVM networks becomes smaller. This is because GOSE extracts rules from the points lying on the boundary. The greater the number of clusters or instances used for rule extraction, the greater the number of points that can be found on the boundary. Hence, more dissimilar rules could be learned to cover many more parts of the *area of interest* (described in Chapter 4) and unseen examples. It is no coincidence, therefore, that all the graphs shown in this chapter are monotonically increasing.

This chapter has presented a series of experiments that empirically evaluated GOSE in various classification learning tasks. The aim of these experiments was to assess GOSE through prediction accuracy, fidelity, comprehensibility and consistency. We go over these criteria in turn.

[1]Obtaining the high fidelity and high accuracy at the same time is unreachable.

a) **Accuracy:** In the three classification domains considered in this chapter, the rules extracted by GOSE have a high accuracy level approaching that of SVM networks. In order to assess the comprehensibility of GOSE, we compare the results from the Monk's, Iris and Breast Cancer problems with those of other algorithms in these three domains. Overall, the rules extracted from SVM networks by GOSE are of comparable correctness to the rules learned from other SVM rule extraction methods, and in some cases, the GOSE rules are much accurate.

b) **Fidelity:** In the three classification domains considered in this chapter, all the fidelity levels of the extracted rules from GOSE are high, and some of them even reach 100%. As demonstrated in Section 5.6, GOSE obtains a better fidelity than some other SVM rule extraction algorithms.

c) **Comprehensibility:** GOSE extracts rules in the form of intervals. These types of rules express the knowledge in the SVM networks with simple syntactic complexity. In some domains, when the nonlinear kernel becomes complicated, the number of the premises in each rule is not appreciably small. This somewhat disappointing result points to several directions for future research, outlined in Chapter 7.

d) **Consistency:** This chapter evaluates consistency, which is an important criterion that may be overlooked. In [84], it states: '*It would be very hard to give any significance to a specific rule set if the extracted rules vary significantly between runs.*' When applying the same synthetic data set to extract rules in different runs, GOSE achieves 100% consistency. It can even obtain more than 80% similarity within the rule sets generated by different synthetic data sets. The results show that GOSE is able to produce consistent rule sets.

e) **Generality:** The generality of GOSE is evaluated by applying it in various

classification domains that involve both discrete and real valued features. The networks used vary in their kernel and training methods. As mentioned above, GOSE carefully selects query instances to extract symbolic rules, as opposed to specific training samples.

Chapter 6

Analytical Evaluation of GOSE

[51] and [66] argue that rule extraction algorithms should be evaluated for the criteria of accuracy, fidelity, comprehensibility, consistency, scalability and generality. The experiments presented in the previous chapter evaluated the accuracy, fidelity, consistency and comprehensibility of the extracted rules. In addition, as GOSE has been applied to various learned models and problem domains, the experiments indirectly demonstrate the generality of GOSE. In this chapter, a much detailed discussion of the scalability and generality of GOSE is provided. Section 6.1 addresses the issue of scalability by analyzing the computational complexity of GOSE. The second section assesses the generality of GOSE. A proof of quasi-soundness/quasi-completeness is presented in the final section.

6.1 Computational Complexity of GOSE

This section discusses the computational complexity of GOSE. Assume that there are n training examples whose dimension is m. The complexity of GOSE is then analyzed step by step. Recall that *querying* (Section 4.2) randomly selects a set

of synthetic instances derived from a common density estimator; *clustering* (Section 4.3) separates those synthetic instances into several groups; *searching* (Section 4.4) looks for the points lying on the SVM classification boundary; *extracting* (Section 4.5) constructs propositional interval rules by solving an optimization problem and *post-processing* (Section 4.6) extends, prunes and selects comprehensible rules.

6.1.1 Computational Complexity of Querying

The *querying* routine includes DRAWINPUTS, DRAWCLASS, QUERYSVM and SELECTINSTANCE (see Chapter 4). DRAWINPUTS involves generating T random instances, calculating their probability distributions from a certain density estimator and returning those instances whose values are larger than a threshold. The complexity of the DRAWINPUTS routine is dominated by the cost of calculating the values of $M(\mathbf{x})$ for T instances. Suppose that there are n training examples used for building up a density estimator. The complexity of this task is then $O(nT)$. The DRAWCLASS routine includes feeding T' uniformly distributed inputs into the SVM network to obtain the class labels. Assume that the time to classify an input is short. The complexity of DRAWCLASS is then $O(T')$. Suppose that T'' instances have passed through the constraints in DRAWINPUTS for QUERYSVM. Although the time for querying one input is considered to be short, to process the classification of T' instances $O(T'')$ is still needed. Finally, GOSE needs to select N instances from the results of QUERYSVM randomly. Obviously, the complexity of this task is $O(N)$. Therefore, the whole complexity of *querying* is $O(nT) + O(T') + O(T'') + O(N)$.

6.1.2 Computational Complexity of Clustering

Clustering involves calculating the distance of each pair of instances within two different clusters and merging the clusters until the stopping criterion is satisfied. Hence, the complexity of *clustering* is dominated by the loops of distance calculation and cluster merging. The maximum cost of *clustering* is then $O(\frac{(1+N)N}{2})$, assuming it takes a short time to compute the distance between two instances.

6.1.3 Computational Complexity of Searching

As presented in Section 4.4, *searching* aims to look for the points lying on the SVM boundary. Clearly, the complexity of *searching* is dominated by the complexity of the pair construction and the search for the points on the classification boundary (see Section 4.4). Suppose that there are n_1 clusters for class A_1 and n_2 clusters for class A_2 and m^j and m^h represent the number of instances in each cluster for class A_1 and A_2 respectively, where $1 \leq j \leq n_1$, $1 \leq h \leq n_2$. There are then $\sum_{j=1}^{n_1} n_2 m^j + \sum_{h=1}^{n_2} n_1 m^h$ pairs in total, in which two instances belong to differing classes. Let v_1 denote $\sum_{j=1}^{n_1} m^j$ and v_2 be $\sum_{h=1}^{n_2} m^h$. Hence, $\sum_{j=1}^{n_1} n_2 m^j + \sum_{h=1}^{n_2} n_1 m^h$ is equal to $n_2 v_1 + n_1 v_2$. The cost of pair construction is $O(2v_1 v_2)$ because, for any individual instance in a class, it has to go through all the instances in another class to find the proper instances in the closest proximity. Assume that one iteration for the binary search calculation takes a short time. Since each pair obtained from the pair construction employs a binary search algorithm and operates on it up to d times, the complexity of the binary search on two classes is then equivalent to $O(dn_2 v_1 + dn_1 v_2)$.

Now we extend *searching* to multi-class classification. Consider that there are M classes. For each class, $(M-1)$ timing pair constructions and binary searches are

required. Hence, the cost of pair construction is $O(2M(M-1)v_1v_2)$ and that of binary search is $O(M(M-1)(n_2v_1+n_1v_2))$. The complexity of *searching* is then $O(M(M-1)(n_2v_1+n_1v_2))+O(2M(M-1)v_1v_2)$.

6.1.4 Computational Complexity of Extracting

The main cost of *extracting* is in solving the optimization problem, and the discussion here therefore focuses on the complexity of the optimization problem set out in Section 4.5.

Since GOSE employs *pattern search* [1] to solve the optimization problem, the process must carry out a number of iterations, which are relevant to the dimensionality of the instance space. Let t_1 stand for the time for approximating the nonlinear constraint of the optimization problem. From the routine of *extracting*, it can be seen that, for each point on the boundary, the optimization function has to be run up to b times, while those synthetic training examples call the function only once. Therefore, the complexity of *extracting* is $O(t_1(tb+N))$, where t is the number of points on the classification boundary and N is the number of training instances.

6.1.5 Computational Complexity of Post-Processing

Post-processing has four steps: rule extending, rule pruning, non-overlapping rule construction and rule selection.

Let the number of initial rules from the *extracting* phase be t_1. Suppose the dimension of the input space is m and the time used for the integral approximation is t_2. From the discussion in Section 4.6.1, it can be seen that the complexity of *rule extending* relates to the number of elements that GOSE goes through in the topology. The following formula is used to calculate the number of elements at

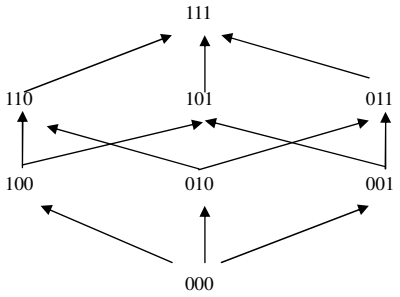


Figure 6.1: An example of the calculation of the number of elements if node 001 cannot satisfy the constraint that $\frac{1}{n}\sum_{i=1}^{n}(|f(\mathbf{x}_i) - A_p|) = 0$. Level 1: $n^1 = 2$, Level 2: $n^2 = 3 - 2 + 0 = 1$ and Level 3: $n^3 = 0$. The final number of elements is 3, greatly reduced from 8 for the worst case.

each tier of the topology (denoted by n^j, $1 \leq j \leq m$). The topology is the basis on the optimizing measure - the *cracking of topology* (shown in Section 4.6.1). Note that the topology has a bottom-to-top structure.

$$C_m^j - C_{m-j+1}^1 d_{j-1} + \sum_{i=1}^{d_{j-1}}(i-1) - d_j$$

Hence, the number of elements involved in *rule extending* equals $\sum_{j=1}^{m} n^j$, and the complexity of this step is $O(t_1 t_2 \sum_{j=1}^{m} n^j)$. Figure 6.1 provides an example showing that GOSE normally experiences fewer elements than the worst case and that the performance of GOSE can improve accordingly.

Consider that t_3 rules are obtained after *rule extending*. Since *rule pruning* has to do t_4 approximations, the cost of *rule pruning* then becomes $O(t_3 t_4 t_2)$. Given that t_5 rules are left after *rule pruning*, the cost of *non-overlapping rule construction* is $O(t_5 m)$. Note that t_5 indicates the number of rules for non-overlapping rules construction, and m is the dimension of the input space. Suppose that there are t_6 rules left after *non-overlap rule construction*. Then $O(t_6)$ is the time spent looking for the rules with a coverage rate larger than zero.

6.2 Generality of GOSE

Much of the rule extraction research described in Chapter 3 suffers from the lack of generality. Some are based on specific training architectures, while some have restrictions on training methods and data. In contrast, GOSE can be applied to a variety of classifiers.

GOSE assumes that (i) a trained model and (ii) the scope of the problem domain are given. The density estimator of the problem domain can either be given from a priori knowledge or built up from the training examples. The latter approach has been adopted in this thesis. In comparison to most SVM rule extraction methods, these assumptions are very weak. The only interface between GOSE and the trained model is that at which the instances are queried and the classification outcomes procured. Therefore, GOSE has no specific requirements on the training architecture and can even be applied to other non-SVM models.

For the experiments presented in Chapter 5, GOSE extracts rules for part of the instance space. GOSE is capable of drawing a complete rule set for the entire instance space by selecting the instances uniformly distributed in the instance space rather than choosing the instances that have the same distribution as the training examples.

6.3 Quasi-Soundness and Quasi-Completeness

In this section, the quasi-soundness and quasi-completeness of GOSE are theoretically proven.

Theorem 1. *Each rule $R : r \rightarrow A_p$ extracted by GOSE approximates the classification obtained by SVM. Note that r refers to the area associated to class A_p.*

Proof. The proof structure is similar to that given by Garcez et al. in 2001 [2].

First, we have to show that a rule R extracted either at the extracting stage or at the post-processing stage can be obtained by querying the SVM. This can be proven by contradiction.

Consider a set of m-dimensional input vectors and a SVM $f(\mathbf{x})$. If the extracted rule R is not obtainable by querying the network, then there must exist a point $\mathbf{x}_i$ in r such that the class output of $f(\mathbf{x}_i)$ is not equivalent to A_p. By the definition of the rule, all the points inside the area r covered by R should refer to the same class A_p. If a point $\mathbf{x}_i$ exists that belongs to the other class, this contradicts to the definition of the rule. Therefore, R must be obtainable by querying the network.

Subsequently, in order to guarantee all the points in r belonging to A_p, the constraint $\int_L^U |f(\mathbf{x}) - A_p| = 0$ must be satisfied, where L and U are the lower and upper bounds of the expected range of R.

At the implementation level, the quasi-Monte Carlo $\frac{1}{n}\sum_{i=1}^{n} |f(\mathbf{x}_i) - A_p| = 0$ is used to approximate the above integral process and obtain the rule R. There should then be a potential approximation error $E = \left| \int_{L'}^{U'} |f(\mathbf{x}) - A_p| - \int_L^U |f(\mathbf{x}) - A_p| \right|$, where L' and U' are the actual lower and upper bounds of R.

Although Equation 2.42 provides a measure for the error estimation, obtaining the variation on $|f(\mathbf{x}) - A_p|$ is difficult. The *rule pruning* step utilizes a statistical method to check if the extracted rule has a small error E. In our implementation, the significant level at the *rule pruning* step is set to 95%. Only the rules whose *t-test* outcomes satisfy the standard value at the significant level are kept, which means the estimation of the integral of r is close to the expected value of 0. This ensures that the approximation error is $E = \left| \int_{L'}^{U'} |f(\mathbf{x}) - A_p| - \int_L^U |f(\mathbf{x}) - A_p| \right| \leq \varepsilon$. Therefore, it can be concluded that the SVM classification on most points is the same as that of R with a rather small difference, and R is said to approximate the

classification of the SVM.

□

Theorem 2. *With an increasing number of rules, the rule set approximates the behavior of the SVM. Let S denote the area covered by the non-overlapping rule set $R = \{r_i \rightarrow A_p, i \geq 1\}$ and V represent the area of interest $I(A_p)$. When the number of rules increases, S approximates V, that is $\frac{|V-S|}{V} \leq \epsilon$, where ϵ is an arbitrary small number. Note that r_i refers to an area in class A_p.*

Proof. Give an input domain $X \subseteq \Re^m$, a set of classes $Y = \{A_p \mid 1 \leq p \leq CN\}$, where CN is the number of classes, and a classifier function $f : X \rightarrow Y$.

Firstly, we need to show that there is an upper bound on *the area of interest*. The definition of *the area of interest* (see Definition 4.0.3) clarifies that $I(A_p)$ has an upper bound equal to $\prod_{i=1}^{m}(u_i - l_i)$, where u_i and l_i are the upper and lower bounds for *the area of interest*.

Next, we need to show that any part of the *area of interest* can be approximated by a set of rules extracted by GOSE (lemma 3), and the more rules we have, the larger the area covered by the rules (lemma 4).

Lemma 3. *Consider V' as any part of the area of interest. V' should then be approximated by a set of rules extracted by GOSE whose area equals S_t. This can be represented as $\frac{|V'-S_t|}{V'} \leq \epsilon$.*

Proof. This can be proven by contradiction. Suppose that the intersection between V' and S_t is V_t.

Assume that V' cannot be approximated by S_t extracted by GOSE; then the area in V' that is not covered by the rule set is large. It also means that the difference between V' and V_t is large.

Since the difference between V' and V_t is also an area, a set of uniformly distributed synthetic instances can be generated inside it, and GOSE is able to extract rules based on these instances (see Chapter 4). Hence, the size of S_t increases, and the difference between V' and V_t decreases. This process can be continued until $|V' - V_t| \leq \varepsilon'$ so that $\frac{|V'-V_t|}{V'} \leq \varepsilon$, where ε and ε' are arbitrary small numbers. Then,

$$\frac{|V' - S_t|}{V'} = \frac{|V' - V_t - (S_t - V_t)|}{V'} \leq \frac{|V' - V_t| + |S_t - V_t|}{V'}$$

By deduction, it can be worked out that $\frac{|S_t-V_t|}{V'} \leq \varepsilon$.

1. Suppose that the area covered by a rule is S_1 and that the intersection area between S_1 and V is V_1. The difference between S_1 and V_1 refers to the part in S_1 that is classified as the other classes by an SVM. With respect to Theorem 1, an extracted rule from GOSE is known to approximate the classification obtained by an SVM. Hence, if the points belonging to the other classes in S_1 exist, they occupy only a very small part of S_1 so that the deviation at this small part cannot influence the approximation value of S_1, which is $\frac{1}{n}\sum_{i=1}^{n} |f(\mathbf{x}) - A_p|$. Therefore, by comparing this with V_1, it can be concluded that $\frac{|S_1-V_1|}{V_1} \leq \varepsilon$, where ε is a arbitrary small number.

2. Let us assume that $\frac{|S_t-V_t|}{V_t} \leq \varepsilon$, where t is an arbitrary integer.

   Then, for $S_{t+1} = S_t + S_1$ and $V_{t+1} = V_t + V_1$,

$$\frac{|S_{t+1} - V_{t+1}|}{V_{t+1}} = \frac{|S_t + S_1 - V_t - V_1|}{V_{t+1}} \leq \frac{|S_t - V_t|}{V_{t+1}} + \frac{|S_1 - V_1|}{V_{t+1}} = \frac{\varepsilon(V_1 + V_t)}{V_{t+1}} = \varepsilon$$

Therefore,

$$\frac{|V' - S_t|}{V'} = \frac{|V' - V_t - (S_t - V_t)|}{V'} \leq \frac{|V' - V_t| + |S_t - V_t|}{V'} \leq 2 * \varepsilon = \epsilon$$

where $\epsilon = 2 * \varepsilon$ is an arbitrary small number.

From the above, it can be demonstrated that V' can be approximated by a set of rules extracted by GOSE. □

Lemma 4. *The area S_{t+1} covered by $t+1$ rules is larger than the area S_t covered by t rules (t is an integer).*

Proof. As the rule is defined to be non-overlapping (Section 4.6), this means that there is no intersection between the rules. The volume covered by $t+1$ rules must then be larger than that covered by t rules.

In other words, if $S_{t+1} \leq S_t$, then there must exist at least two rules that overlap. This contradicts the definition of non-overlapping. Therefore, it can be concluded that $S_{t+1} > S_t$. □

Lemma 3 demonstrates that any part of the *area of interest* can be approximated by a set of rules. Lemma 4 shows that the greater the number of rules extracted, the larger the non-overlapping area covered by the rules. Therefore, when the number of rules increases, the area covered by the rules can finally approximate *the area of interest*. Furthermore, the difference between the area covered by the rules (denoted by S) and the *area of interest* (denoted by V) satisfies $\frac{|V-S|}{V} \leq \epsilon$, where ϵ is an arbitrary small number.

Hence, by increasing the number of rules, the rule set extracted by GOSE can approximate the behavior of SVM networks. □

6.4 Chapter Summary

This chapter has discussed the scalability and generality of GOSE and proven the quasi-soundness and quasi-completeness of the algorithm. The first section has analyzed the computational complexity of GOSE for each phase.

Section 6.2 discussed the generality of GOSE. GOSE is applicable not only to SVM models but also to a wide array of other learned models that do not have any special training regime.

Section 6.3 proved the quasi-soundness and quasi-completeness of GOSE. It demonstrated that any individual rule extracted by GOSE is obtainable by the SVM network and that the rule set extracted by GOSE is able to approximate the behavior of SVM as the number of rules increases.

Chapter 7

Conclusions

Support Vector Machines are a type of popular unsupervised learning method based on statistical learning theory. They have been applied to many classification and regression problems and have demonstrated good predictive performance. They also have good generalization in a wide variety of practical problem domains through implementation of the structural risk-minimization principle. However, SVMs (like other neural networks) lack the ability to explain results in a comprehensible form. Rule extraction, which combines the non-symbolic and symbolic paradigms, makes the interpretation of the behavior of connectionist networks possible. This thesis has described the development of a general rule extraction method for SVM networks, called GOSE.

In this concluding chapter, the contributions and limitations of GOSE are discussed. Several future work directions are proposed to resolve the limitations.

7.1 Contribution

This thesis is a contribution to describe Support Vector Machines as a set of understandable rules. The contributions are discussed below in detail.

1. The GOSE algorithm

One significant contribution of this thesis is the novel approach of GOSE to extract rules from SVMs. The approach views the rule extraction problem as a pure geometric task without considering the structure of the learning models and important support vectors. Compared with many existing SVM rule extraction algorithms that require specific training methods or employ support vectors, a major advantage of GOSE is that it is widely appropriate to a broad class of hyperplane learning models. One reason for this is that querying is the only interaction between GOSE and a given model. Another significant reason is that it is free to use synthetic training examples for extracting rules, which further improves the generalization of GOSE. Specifically, owing to the fact that GOSE extracts rules by using synthetic examples rather than existing training examples, this makes the approach independent from the specific training data and applicable in a variety of problem domains.

2. Extensive evaluation of a rule extraction method

In Chapter 5, GOSE was applied to various SVMs in three benchmark problem domains. These experiments are designed to evaluate GOSE on five criteria: accuracy, fidelity, consistency, comprehensibility and generality.

GOSE achieves a good balance between the accuracy and the fidelity of the rules compared with other SVM extraction work. In all of the domains, GOSE offers a high level of accuracy and fidelity to the relevant SVMs.

The experiments evaluate the consistency of extracted rules by measuring the similarities between rule sets and between individual rules. Consistency might be overlooked for the sake of testing a rule extraction method. However, the correctness of the algorithm is demonstrated if a constant performance is obtained.

The experiments also demonstrate the generality of GOSE by using different SVM algorithms in a variety of problem domains such as medical tasks, which have various network architectures, attribute types and training-set sizes.

The scalability of GOSE is measured by analyzing the time complexity of each step in the algorithm. Although some steps might have high complexity in the worst case scenario, it would be interesting to see that corresponding complexity is reduced to an acceptable range when the optimizing measure is used such as cracking of topology. Chapter 4 has described this measure.

3. Proof of rule extraction method on quasi-soundness and quasi-completeness

In Chapter 6the quasi-soundness and quasi-completeness of GOSE were proven. This means that each rule extracted by GOSE approximates the classification of SVMs and that the rule set is also able to approximate the behavior of SVMs. The *t-test* in rule selection ensures that the approximation of each individual rule is close to the expected value of the integral $\int_L^U |f(\mathbf{x}) - A_p|$ so that the rule is obtainable from the SVM. Since the final rule set is composed of non-overlapping rules, it guarantees that the area covered by the rule set increases as the number of rules increases. Therefore, the rule set is able to approximate the behavior of SVMs by increasing of the number of the rules.

In summary, as demonstrated in this thesis, there is a clear synergy between SVMs and symbolic rules. The research on the combination of those the two paradigms can lead to great benefits in both areas and to a better understanding of machine learning in general.

7.2 Limitations and Future Work

Despite the many contributions of GOSE, the performance of GOSE can be further improved by overcoming the following limitations.

i) The rule set can be further simplified compared with other SVM rule extraction methods.

ii) The complexity is higher than some SVM extraction algorithms.

iii) The algorithm can be further developed for regression problem domains.

The following future studies are designed to resolve these problems.

Simplification of Rule

The rule extraction work in this thesis focuses on the interval rules. Although the experiments presented in Chapter 5 show that GOSE can obtain a set of comprehensible rules, the comprehensibility of the rule set can still be improved compared with other rule extraction methods. For example, *Neurorule* [33] generates a rule set with an average of 4.2 antecedents per rule for a breast cancer data set, while GOSE has around 6 antecedents for each rule (see Chapter 5). For a rule extraction method, it is desirable to extract a compact rule set. One way to do this would be to change the form of the rules to the *MofN* rule or the regression rule to reduce the number and the size of rules. For instance, if the rules for Monk-2 in Appendix A are altered to *MofN* rules, this would largely decrease the number of rules. A specific new *MofN* rule deduced from the rules in Appendix A.0.1 might be *'3 of* 6 *inputs equal* 1 *then class* $= 0$*'*. Relating this to the feature of an interval rule, which includes numeric lower and upper bounds, the transformation from an interval rule to the $MofN$ format usually happens at a data set with discrete or integer values.

Reduction of Complexity

The general approach underlying GOSE is to draw rules from the concept of geometry. As it has been argued, this approach can be applied to a wide range of problems. However, owing to the complicated nature of nonlinear SVM boundaries and the increase of the dimensionality, the running time of GOSE might also increase greatly. From the analytical evaluation presented in Chapter 6, it can be seen that the complexity of GOSE is dominated by *extracting* and *post-processing*.

There are several potential methods that GOSE could use to reduce the complexity.

- *Reinforcement learning* [75] is a method that can acquire information to guide the GOSE inductive process. For example, the reward function of the *n-armed bandit problem* [75] or the action preference in *pursuit methods* [75] might suggest the preferential attributes for generalization so that further reductions can be achieved in the searching efforts undertaken on the topology made by GOSE in rule extending.

- The *domain knowledge* [9], which refers to the information, could generate *representative* synthetic training examples so that GOSE might extract fewer initial rules before *post-processing*, which may reduce the running time in the *post-processing* process.

- *Parallel computing* is used to implement the *extracting* and the *rule-extending* stage in parallel. The computational complexity of *extracting* is associated with N training examples and t points on the boundary (see Section 6.1.4). The increase/decrease of N and t can lead to the increase/decrease of the complexity. Therefore, one solution for reducing the time complexity in *extracting* relates to parallel computing. For example, given 30 instances, half of them belong to one class, which means one cluster for each class. After *searching*,

30 points are found on the boundary. Assume that a seconds are spent on solving every optimization problem. Regarding each point, an average of b times optimization procedures are called. If 30 threads run simultaneously, the consumed time is ab seconds as opposed to $30ab$ without parallel computing. The *rule-extending* algorithm could also be implemented on multiple rules at the same time. Hence, the complexity of the *rule-extending* algorithm would decrease further.

Application to Regression Problems

Since GOSE is based on the hyperplane of learning models, it could also be applied to the regression problems for which curves can be drawn by learning models. By combining with the relevant form of the regression rule, GOSE might be capable of finding the regression regularities within the features of the inputs. The format of a regression rule can be

$$y_i = h(x_i, \beta_0) + \varepsilon_i$$

where $h : \mathbb{R}^m \times \mathbb{R}^k \to \mathbb{R}$, β_0 denotes a set of k unknown parameters and m is the dimension of inputs.

For example, after obtaining the points lying on the boundary, a nonlinear regression rule $h(\mathbf{x})$ can be obtained by using least-square approximation. Then, a rule might be extracted, such as '*if* $\mathbf{x}$ *in Region 1, then* $y = h(\mathbf{x})$'.

Bibliography

[1] Charles. A and Dennis. J. E. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[2] Garcez. A. A, Broda. K, and Gabbay. D. Symbolic knowledge extraction from trained neural networks: a sound approach. *Artificial Intelligence*, 125(1–2):155–207, January 2001.

[3] Karaivanova. A, Dimov. I, and Ivanovska. S. A quasi-monte carlo method for integration with improved convergence. In *LSSC '01: Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers*, pages 158–165, London, UK, 2001. Springer-Verlag.

[4] Ultsch. A. Knowledge extraction from self-organizing neural networks. *Information and Classification*, pages 301–306, 1993.

[5] Vahed. A and Omlin. C. W. Rule extraction from recurrent neural networks using a symbolic machine learning algorithm. In *Proceedings 6th International Conference on Neural Information Processing*, volume 2, pages 712–717, 1999.

[6] Vahed. A and Omlin. C. W. A machine learning method for extracting symbolic knowledge from recurrent neural networks. *Neural Computation*, 16(1):59–71, 2004.

[7] Boser. B, Guyon. I, and Vapnik. V. A training algorithm for optimal margin classifiers. In *Computational Learning Theory*, pages 144–152, 1992.

[8] Hammer. B, Rechtien. A, Strickert. M, and Villmann. T. Rule extraction from self-organizing networks. In *International Conference on Artificial Neural Networks*, pages 370–375, 2002.

[9] Hjorland. B and Albrechtsen. H. Toward a new horizon in information science: domain analysis. *Journal of the American Society for Information Science*, 46(6):400–425, 1995.

[10] Thrun. S. B. Extracting provably correct rules from artificial neural networks. Technical report, Institut für Informatik III, Universität Bonn, 1993.

[11] Tickle. A. B, Oriowsk. M, and Diederich. J. Dedec: decision detection by rule extraction from neural networks. Technical report, Neurocomputing Res. Centre, Queensland University, Technol., Brisbane, Qld., September 1994.

[12] Tickle. A. B, Andres. R, Golea. M, and Diederich. J. The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6):1057–1068, 1998.

[13] Burges. C. J. C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[14] Giles. L. C and Omlin. W. C. Extraction, insertion, and refinement of symbolic rules in dynamically driven recurrent networks. *Connection Science*, 5:307–328, 1993. Special Issue on Architectures for Integrating Symbolic and Neural Processes.

[15] Lee. C and Landgrebe. A. D. Feature extraction based on decision boundaries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):388–400, 1993.

[16] McMillan. C, Mozer. M. C, and Smolensky. P. The connectionist scientist game: rule extraction and refinement in a neural network. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 424–430, 1991.

[17] Omlin. W. C and Giles. L. C. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

[18] Pop. E, Hayward. R, and Diederich. J. Ruleneg: Extracting rules from a trained ann by stepwise negation. Technical report, University of Queensland, 1994.

[19] Rumelhart. D. E, Hinton. G. E, and Williams. R. J. Learning internal representations by error propagation. pages 318–362, 1986.

[20] Chaves. Ad. C. F, Vellasco. M. M. B. R, and Tanscheit. R. Fuzzy rule extraction from support vector machines. In *Proceedings of the 5th International Conference on Hybrid Intelligent Systems*, pages 335–340, 2005.

[21] Maire. F. A partial order for the m-of-n rule extraction algorithm. *IEEE Transaction on Neural Networks*, 8(6):1542–1544, 1997.

[22] Fung. G, Sandilya. S, and Rao. B. R. Rule extraction from linear support vector machines. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 32–40, Chicago, Illinois, USA, 2005.

[23] Karypis. G, Han. E-H, and Kumar. V. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[24] Fisher. D. H. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[25] Halton. J. H. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerical Mathematics*, 2:84–90, 1960.

[26] Jacobsson. H. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263, 2005.

[27] Karloff. H. *Linear programming*. MA: Birkhäuser, 1991.

[28] Kriegel. P. H and Pfeifle. M. Density-based clustering of uncertain data. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 672–677, New York, NY, USA, 2005. ACM Press.

[29] Morohosi. H and Fushimi. M. A practical approach to the error estimation of quasi-monte carlo integration. In Niederreiter. H and Spanier. J, editors, *Monte Carol and Quasi-Monte Carlo Methods*, pages 377–390, Berlin, 1998. Springer.

[30] Niederreiter. H. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial Mathematics, Philadelphia, 1992.

[31] Núñez. H, Cecilio Angulo, and Andreu Català. Hybrid architecture based on support vector machines. In *Proceedings of International Work Conference on Artificial Neural Networks*, pages 646–653, 2003.

[32] Simon. H. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[33] Lu. H-J, Setiono. R, and Liu. H. Neurorule: A connectionist approach to data mining. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 478–489. Morgan Kaufmann, 1995.

[34] Cloete. I and Zurada. J. M. *Knowledge-based neurocomputing*. The MIT Press, Cambridge, MA, USA, 2000.

[35] Gallant. S. I. Connectionist expert systems. *Communications of the ACM*, (2):152–169, 1988.

[36] Bacher. J. A probabilistic clustering model for variables of mixed type. *Quality and Quantity*, 34(13):223–235, August 2000.

[37] Mahoney. J. J and Mooney. R. J. Initializing id5r with a domain theory: Some negative results. Technical Report AI91-154, University of Texas at Austin, 1991.

[38] Neumann. J. *Classification and evaluation of algorithms for rule extraction from artificial neural networks.* PhD thesis, University of Edingburgh, 1998.

[39] Platt. J, Cristianini. N, and Shawe-Taylor. J. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems*, volume 12, pages 547–553, Cambridge, 2000. MIT Press.

[40] Platt. C. J. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, April 1998.

[41] Rabunal. R. J, Dorado. J, Pazos. A, Pereira. J, and Rivero. D. A new approach to the extraction of ann rules and to their generalization capacity through gp. *Neural Computation*, 16(7):1483–1523, 2004.

[42] Zhang. J and Liu. Y. Svm decision boundary based discriminative subspace induction. *Pattern Recognition*, 38(10):1746–1758, 2005.

[43] Saito. K and Nakano. R. Medical diagnostic expert system based on pdp model. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 255–262. IEEE Press, 1988.

[44] Saito. K and Nakano. R. Law discovery using neural networks. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1078–1083, 1997.

[45] Suykens. J. A. K and Vandewalle. J. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[46] Bochereau. L and Bourgine. P. Extraction of semantic features and logical rules from a multilayer neural network. In *International Joint Conference on Neural Networks*, volume 2, pages 579–582, Washington DC, 1990.

[47] Castro. J. L, Mantas. C.J, and Benitez. J.M. Interpretation of artificial neural networks by means of fuzzy rules. *IEEE Transactions on Neural Networks*, 13(1):101–116, 2002.

[48] Hansen. K. L and Salamon. P. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, 1990.

[49] Fu. L.M. Rule generation from neural networks. *IEEE Transactions on systems, Man, and Cybernetics*, 24(8):1114–1124, 1994.

[50] Craven. M. *Extracting comprehensible models from trained neural networks.* PhD thesis, University of Wisconsin, Madison, WI, 1996.

[51] Craven. M and Shavlik. J. Rule extraction: where do we go from here? *University of Wisconsin Machine Learning Research Group Working Paper 99-1*, 1999.

[52] Craven. W. M. *Extracting comprehensible models from trained neural networks.* PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, 1996.

[53] Fu. L. M. Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 590–595, Anaheim CA, 1991.

[54] Ishikawa. M. Neural networks approach to rule extraction. In *ANNES '95: Proceedings. Second New Zealand International Two-Stram Conference on Artificial Neural Networks and Expert Systems*, page 6. IEEE Computer Society, November 1995.

[55] Ishikawa. M. Rule extraction by successive regularization. *Neural Networks*, 13(10):1171–1183, 2000.

[56] Siponen. M, Vesanto. J, Simula. O, and Vasara. P. An approach to automated interpretation of som. In *Advances in Self-Organizing Maps*, pages 89–94. Springer, 2001.

[57] Barakat. N and Diederich. J. Learning-based rule-extraction from support vector machines. In *The 14th International Conference on Computer Theory and applications*, pages 247–252, 2004.

[58] Barakat. N and Diederich. J. Eclectic rule extraction from support vector machines. *International Journal of Computational Intelligence*, 2(1):59–62, 2005.

[59] N*ũñ*ez. N, Angulo. C, and Catal*à*. A. Rule extraction from support vector machines. In *Proceedings of European Symposium on Artificial Neural Networks Bruges*, pages 107–112, Belgium, 2003.

[60] Nú nez. H, Angulo. C, and Català. Rule based learning systems for support vector machines. *Neural Processing Letters*, 24(1):1–18, August 2006.

[61] Raymond. T. Ng and Han. J. W. Clarans: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[62] Berkhin. P. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[63] Frasconi. P, Gori. M, Maggini. M, and Soda. G. Representation of finite state automata in recurrent radial basis function networks. *Machine Learning*, 23(1):5–32, 1996.

[64] Tino. P and Sajda. J. Learning and extracting initial mealy automata with a modular neural network model. *Neural Computation*, 7(4):822–44, July 1995.

[65] Tino. P and Koteles. M. Extracting finite-state representations from recurrent neural networks trained on chaotic symbolic sequences. *IEEE Transaction on Neural Networks*, 10(2):284–302, 1999.

[66] Andrews. R, Diederich. J, and Tickle. A. B. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8:373–389, 1995.

[67] Andrews. R and Geva. S. Rule extraction from a constrained error back propagation mlp. In *Proceedings of the 5th Australian Conference on Neural Networks*, pages 9–12, 1994.

[68] Andrews. R and Geva. S. Inserting and extracting knowledge from constrained error back propagation networks. In *Proceedings of the 6th Australian Conference on Neural Networks*, pages 29–32, 1995.

[69] Filer. R, Sethi. I, and Austin. J. A comparison between two rule extraction methods for continuous input data. In *Proceedings of NIPS'97 Rule Extraction From Trained Artificial Neural Networks Workshop*, pages 38–45, 1996.

[70] Krishnan. R. A systematic method for decompositional rule extraction from neural networks. *Proceedings of NIPS'97 Rule Extraction from Trained Artificial Neural Networks Workshop*, pages 38–45, 1996.

[71] Setiono. R. Extracting rules from neural networks by pruning and hidden unit splitting. *Neural Computation*, 9(1):205–225, 1997.

[72] Setiono. R. A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997.

[73] Setiono. R, Leow. W. K, and Zurada. J. M. Extraction of rules from artificial neural networks for nonlinearregression. volume 13, pages 564–577, 2002.

[74] Sibson .R. Slink: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[75] Sutton. S. R and Barto. G. A. *Reinforcement learning*. MIT Press, Cambridge, MA, 1998.

[76] Watrous. L. R and Kuhn. M. G. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414, May 1992.

[77] UCI Machine Learning repository. `http://mlearn.ics.uci.edu/MLRepository.html`.

[78] Haykins. S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, second edition edition, 2001.

[79] Sestito. S and Dillon. T. *Automated knowledge acquisition*. Prentice Hall (Australia), 1994.

[80] Thrun. S. Extracting rules from artificial neural networks with distributed representations. *Advances in Neural Information Processing Systems*, 7:505–512, 1995.

[81] Thrun. S, Bala. J, Bloedorn. E, Bratko. I, Cestnik. B, Cheng. J, De Jong. K, Dzeroski. S, Fahlman. S. E, Fisher. D, Hamann. R, Kaufman. K, Keller. S, Kononenko. I, Kreuziger. J, Michalski. R. S, Mitchell. T, Pachowicz. P, Reich, Vafaie. H, Van de Welde. W, Wenzel. W, Wnek. J, and Zhang. J. The monkś problems - a performance comparison of different learning algorithms. Technical report, Carnegie Mellon University, 1991.

[82] Hastie. T and Tibshirani. R. Classification by pairwise coupling. In *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[83] Santos. R. T, Nievola. J. C, and Freitas. A. A. Extracting comprehensible rules from neural networks via genetic algorithms. In *Proceedings of 2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 130–139, San Antonio, TX, USA, May 2000. IEEE.

[84] Johansson. U, Konig. R, and Niklasson. L. Automatically balancing accuracy and comprehensibility in predictive modeling. In *Proceedings of the 8th International Conference on Information Fusion*, volume 2, page 7, 2005.

[85] Cherkassky. V and Mulier. F. *Learning from data: concepts, theory, and methods*. John Wiley & Sons Inc., 1998.

[86] Vapnik. V. *Estimation of Dependences Based on Empirical Data:: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag, Secaucus, NJ, USA, 1982.

[87] Vapnik. V and Lerner. A. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

[88] Vapnik. N. V. *The nature of statistical learning theory*. Springer, New York, NY, USA, 1995.

[89] Vapnik. N. V. *Statistical learning theory*. John Wiley and Sons, INC, 1998.

[90] Bala. J. W., Bloedorn. E., De Jong. K. A., Kaufman. K., Michalski. R. S., Pachowicz. P. W., Vafaie. H., Wnek. J., and J. Zhang. A brief review of aq learning programs and their application to the monks problems. Technical Report MLI92-9, George Mason University, Fairfax, VA, 1992.

[91] Craven. M. W and Shavlik. W. J. The extraction of refined rules from knowledge based neural networks. *Machine Learning*, 131(1):71–101, 1993.

[92] Craven. M. W and Shavlik. J. W. Using sampling and queries to extract rules from trained neural networks. In *International Conference on Machine Learning*, pages 37–45, 1994.

[93] Feller. W. *An Introduction to Probability Theory and Its applications*, volume 2. John Wiley, New York, 3 edition, 1971.

[94] Morokoff. J. W and Caflisch. R. E. Quasi-Monte Carlo integration. *J. Comp. Phys.*, 122:218–230, 1995.

[95] Omlin. C. W, Giles. C. L, and Miller. C. B. Heuristics for the extraction of rules from discrete time recurrent neural networks. *Proceedings of the International Joint Conference on Neural Networks*, 1(7-11):33–38, 1992.

[96] Rudin. W. *Principles of Mathematical Analysis, 3rd edition.* ., McGraw-Hill, 1976.

[97] Scott. D. W. *Multivariate Density Estimation: Theory, Practice, and Visualization.* John Willey & Sons, New York, 1992.

[98] Fu. X-J, Ong. C-J, Keerthi. S, Gih G-H, and Goh L-P. Extracting the knowledge embedded in support vector machines. volume 1, pages 291–296, 2004.

[99] Fu. X-J and Wang. L-P. *Rule extraction from support vector machines.* Springer, 2005.

[100] Chen. Y-X and Wang. Jame. Z. Support vector learning for fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, 11:716–728, 2003.

[101] Jiang. Z and Chen. Y. Extracting symbolic rules from trained neural network ensembles. *AI Communications*, 16(1):3–15, 2003.

[102] Zeng. Z, Goodman. M, and Smyth. P. Learning finite state machines with selfclustering recurrent networks. *Neural Computation*, 5(6):978–990, 1993.

[103] Zhou. Z. Rule extraction: using neural networks or for neural networks. *Journal of Computer Science and Technology*, 19:249–253, 2004.

Appendix A

Rule Sets

The following rule sets show the detailed rules extracted for Monk's problems, Iris plant problem and Breast Cancer problem. We consolidate the rules from each run of cross-validation and apply them on the whole data set. The rule sets is expressed in the form of

$$\bigwedge a_i = [a_i, b_i] \rightarrow A_p \ (coverage \ r)$$

where $1 \leq i \leq m$, m is the dimension of the inputs, coverage r indicates the number of the inputs correctly predicted by the rule.

A.0.1 Monk's Problem

Monk-1

There are 8 rules extracted by the GSVMORC, as given below:

1. $a_5 = 1 \rightarrow 1 \ (r = 72)$
2. $a_1 = 3 \bigwedge a_2 = 3 \rightarrow 1 \ (r = 48)$
3. $a_1 = 2 \bigwedge a_2 = 2 \rightarrow 1 \ (r = 48)$
4. $a_1 = 1 \bigwedge a_2 = 1 \rightarrow 1 \ (r = 48)$
5. $a_1 = 1 \bigwedge a_2 = [2, 3] \bigwedge a_5 = [2, 4] \rightarrow 0 \ (r = 196)$
6. $a_1 = 3 \bigwedge a_2 = [1, 2] \bigwedge a_5 = [2, 4] \rightarrow 0 \ (r = 72)$
7. $a_1 = [2, 3] \bigwedge a_2 = 1 \bigwedge a_5 = [2, 4] \rightarrow 0 \ (r = 36)$
8. $a_1 = [1, 2] \bigwedge a_2 = 3 \bigwedge a_5 = [2, 4] \rightarrow 0 \ (r = 36)$

Monk-2

There are 62 rules extracted by the GSVMORC.

1. $a_1 = [1,2] \bigwedge a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 24)$
2. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_4 = [2,3] \bigwedge a_5 = [2,3] \bigwedge a_6 = 1 \rightarrow 1$ $(r = 16)$
3. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_4 = [2,3] \bigwedge a_5 = 4 \bigwedge a_6 = 1 \rightarrow 1$ $(r = 8)$
4. $a_1 = 1 \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_4 = [2,3] \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 12)$
5. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_4 = 1 \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 12)$
6. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = 1 \bigwedge a_6 = 1 \rightarrow 1$ $(r = 8)$
7. $a_1 = 1 \bigwedge a_2 = 1 \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 6)$
8. $a_1 = 1 \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = 1 \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 6)$
9. $a_1 = 1 \bigwedge a_2 = 3 \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_6 = 1 \rightarrow 1$ $(r = 6)$
10. $a_1 = [2,3] \bigwedge a_2 = 1 \bigwedge a_3 = 2 \bigwedge a_4 = 2 \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 6)$
11. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_4 = 2 \bigwedge a_5 = 1 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 4)$
12. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = 1 \bigwedge a_5 = 4 \bigwedge a_6 = 1 \rightarrow 1$ $(r = 4)$
13. $a_1 = [1,2] \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = 1 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 4)$
14. $a_1 = [2,3] \bigwedge a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_4 = [1,2] \bigwedge a_5 = [2,3] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 4)$
15. $a_1 = [2,3] \bigwedge a_2 = 1 \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = 1 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 4)$
16. $a_1 = [2,3] \bigwedge a_2 = 1 \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = 2 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 4)$
17. $a_1 = 1 \bigwedge a_2 = 2 \bigwedge a_3 = 2 \bigwedge a_4 = 2 \bigwedge a_5 = [2,4] \rightarrow 1$ $(r = 3)$
18. $a_1 = 2 \bigwedge a_2 = 3 \bigwedge a_3 = 2 \bigwedge a_4 = 1 \bigwedge a_6 = 1 \rightarrow 1$ $(r = 2)$
19. $a_1 = [2,3] \bigwedge a_2 = 1 \bigwedge a_3 = 2 \bigwedge a_4 = [1,2] \bigwedge a_5 = 2 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 2)$
20. $a_1 = [2,3] \bigwedge a_2 = 2 \bigwedge a_3 = 2 \bigwedge a_4 = 1 \bigwedge a_5 = [3,4] \bigwedge a_6 = 1 \rightarrow 1$ $(r = 2)$
21. $a_1 = [2,3] \bigwedge a_2 = 3 \bigwedge a_3 = 1 \bigwedge a_4 = 3 \bigwedge a_5 = 1 \rightarrow 1$ $(r = 2)$
22. $a_1 = 1 \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = [3,4] \bigwedge a_6 = 1 \rightarrow 1$ $(r = 2)$
23. $a_1 = 1 \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = 3 \bigwedge a_5 = [2,4] \bigwedge a_6 = 1 \rightarrow 1$ $(r = 1)$
24. $a_1 = [1,2] \bigwedge a_2 = 3 \bigwedge a_3 = 2 \bigwedge a_5 = 1 \bigwedge a_6 = 2 \rightarrow 1$ $(r = 1)$
25. $a_1 = 2 \bigwedge a_2 = 2 \bigwedge a_3 = 1 \bigwedge a_4 = [2,3] \bigwedge a_5 = [1,2] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 1)$

26. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = [2,3] \bigwedge a_6 = [1,2] \rightarrow 0$ $(r = 217)$

27. $a_1 = 1 \bigwedge a_3 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 16)$

28. $a_1 = 1 \bigwedge a_4 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 16)$

29. $a_2 = 1 \bigwedge a_4 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 16)$

30. $a_3 = 1 \bigwedge a_4 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 13)$

31. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = [1,2] \bigwedge a_4 = 3 \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 12)$

32. $a_1 = [2,3] \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_5 = 1 \bigwedge a_6 = 2 \rightarrow 0$ $(r = 12)$

33. $a_1 = 1 \bigwedge a_3 = 1 \bigwedge a_4 = 1 \rightarrow 0$ $(r = 9)$

34. $a_3 = 1 \bigwedge a_4 = 1 \bigwedge a_5 = [1,3] \bigwedge a_6 = 1 \rightarrow 0$ $(r = 8)$

35. $a_1 = 1 \bigwedge a_2 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 7)$

36. $a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 6)$

37. $a_1 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = 3 \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 6)$

38. $a_1 = [1,2] \bigwedge a_2 = 1 \bigwedge a_4 = 3 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 4)$

39. $a_2 = [1,2] \bigwedge a_3 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 4)$

40. $a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = 3 \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 4)$

41. $a_2 = 3 \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = [3,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 4)$

42. $a_1 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = [2,3] \bigwedge a_5 = [3,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 4)$

43. $a_2 = [1,2] \bigwedge a_4 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 3)$

44. $a_1 = 3 \bigwedge a_4 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 3)$

45. $a_1 = 3 \bigwedge a_2 = 3 \bigwedge a_4 = [2,3] \bigwedge a_5 = [2,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 3)$

46. $a_1 = [2,3] \bigwedge a_2 = 3 \bigwedge a_4 = [2,3] \bigwedge a_5 = [3,4] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 2)$

47. $a_1 = 3 \bigwedge a_2 = 3 \bigwedge a_3 = 2 \bigwedge a_6 = 2 \rightarrow 0$ $(r = 2)$

48. $a_1 = [1,2] \bigwedge a_2 = 1 \bigwedge a_4 = 1 \bigwedge a_5 = 4 \bigwedge a_6 = [1,2] \rightarrow 0$ $(r = 2)$

49. $a_1 = [1,2] \bigwedge a_3 = 1 \bigwedge a_4 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 2)$

50. $a_2 = 1 \bigwedge a_4 = [1,2] \bigwedge a_5 = 4 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 2)$

51. $a_1 = 1 \bigwedge a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_5 = [1,2] \bigwedge a_6 = [1,2] \rightarrow 0$ $(r = 2)$

52. $a_1 = 1 \bigwedge a_2 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 2)$

53. $a_1 = 1 \bigwedge a_2 = [1,2] \bigwedge a_5 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 2)$
54. $a_1 = 1 \bigwedge a_2 = 2 \bigwedge a_3 = 1 \bigwedge a_5 = 1 \rightarrow 0$ $(r = 2)$
55. $a_1 = 1 \bigwedge a_3 = 1 \bigwedge a_4 = [1,2] \bigwedge a_5 = 1 \rightarrow 0$ $(r = 1)$
56. $a_1 = [1,2] \bigwedge a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_4 = 1 \bigwedge a_5 = [1,2] \rightarrow 0$ $(r = 1)$
57. $a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_4 = [1,2] \bigwedge a_5 = [1,2] \bigwedge a_6 = 1 \rightarrow 0$ $(r = 1)$
58. $a_2 = [1,2] \bigwedge a_3 = 1 \bigwedge a_4 = 1 \bigwedge a_6 = 1 \rightarrow 0$ $(r = 1)$
59. $a_1 = 2 \bigwedge a_3 = 1 \bigwedge a_4 = [1,2] \bigwedge a_5 = 1 \rightarrow 0$ $(r = 1)$
60. $a_1 = 3 \bigwedge a_2 = 2 \bigwedge a_3 = 2 \bigwedge a_4 = 2 \bigwedge a_5 = [1,3] \bigwedge a_6 = 2 \rightarrow 0$ $(r = 1)$
61. $a_1 = 3 \bigwedge a_2 = [2,3] \bigwedge a_3 = [1,2] \bigwedge a_4 = [2,3] \bigwedge a_5 = 4 \bigwedge a_6 = 2 \rightarrow 0$ $(r = 1)$
62. $a_1 = 3 \bigwedge a_2 = [2,3] \bigwedge a_3 = 2 \bigwedge a_4 = 3 \bigwedge a_6 = 2 \rightarrow 0$ $(r = 1)$

Monk-3

There are 17 rules extracted for Monk-3 problem.

1. $a_2 = 1 \bigwedge a_5 = [1,2] \rightarrow 1$ $(r = 72)$
2. $a_1 = 1 \bigwedge a_2 = 2 \bigwedge a_5 = [1,3] \rightarrow 1$ $(r = 54)$
3. $a_2 = [1,2] \bigwedge a_5 = [1,3] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 36)$
4. $a_1 = [2,3] \bigwedge a_2 = 1 \bigwedge a_5 = [1,3] \rightarrow 1$ $(r = 16)$
5. $a_1 = 3 \bigwedge a_2 = [1,2] \bigwedge a_5 = [1,3] \rightarrow 1$ $(r = 12)$
6. $a_2 = [1,2] \bigwedge a_5 = 1 \rightarrow 1$ $(r = 12)$
7. $a_2 = [1,2] \bigwedge a_3 = 2 \bigwedge a_5 = [1,3] \rightarrow 1$ $(r = 9)$
8. $a_1 = 3 \bigwedge a_3 = 1 \bigwedge a_5 = 3 \rightarrow 1$ $(r = 4)$
9. $a_1 = 3 \bigwedge a_2 = 3 \bigwedge a_3 = 2 \bigwedge a_5 = [3,4] \bigwedge a_6 = 2 \rightarrow 1$ $(r = 3)$
10. $a_1 = [1,2] \bigwedge a_2 = 3 \bigwedge a_3 = 1 \bigwedge a_4 = [1,2] \bigwedge a_5 = [3,4] \rightarrow 1$ $(r = 2)$
11. $a_1 = 2 \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_5 = [3,4] \bigwedge a_6 = 1 \rightarrow 1$ $(r = 1)$
12. $a_1 = 2 \bigwedge a_2 = [2,3] \bigwedge a_3 = 1 \bigwedge a_5 = [2,4] \bigwedge a_6 = 1 \rightarrow 0$ $(r = 129)$
13. $a_5 = 4 \rightarrow 0$ $(r = 102)$
14. $a_1 = [1,2] \bigwedge a_2 = 3 \rightarrow 0$ $(r = 59)$

15. $a_2 = 3 \bigwedge a_3 = 2 \rightarrow 0$ $(r = 12)$

16. $a_2 = 3 \bigwedge a_5 = [1, 2] \rightarrow 0$ $(r = 12)$

17. $a_1 = 1 \bigwedge a_2 = 1 \bigwedge a_3 = 1 \bigwedge a_5 = [3, 4] \rightarrow 0$ $(r = 1)$

A.0.2 Iris Plant Problem

There are 31 rules in total extracted by the GSVMORC. class 1 refers to Iris sentosa, 2 refers to Iris Versicolour and 3 refers to Iris Virginica. The rules are listed as follows:

1. $a_1 = [4.3, 6.6] \bigwedge a_2 = [2.0, 4.0] \bigwedge a_3 = [2.7, 5.0] \bigwedge a_4 = [0.4, 1.7] \rightarrow 2$ $(r = 45)$

2. $a_1 = [6.5, 7.9] \bigwedge a_2 = [2.4, 4.3] \bigwedge a_3 = [2.3, 4.7] \bigwedge a_4 = [0.7, 1.9] \rightarrow 2$ $(r = 3)$

3. $a_1 = [6.8, 7.9] \bigwedge a_2 = [2.0, 3.9] \bigwedge a_3 = [4.7, 5.1] \bigwedge a_4 = [1.4, 1.7] \rightarrow 2$ $(r = 2)$

4. $a_1 = [6.6, 6.8] \bigwedge a_2 = [2.8, 3.4] \bigwedge a_3 = [3.2, 5.0] \bigwedge a_4 = [1.6, 1.7] \rightarrow 2$ $(r = 1)$

5. $a_1 = [4.3, 5.9] \bigwedge a_2 = [3.1, 4.0] \bigwedge a_3 = [2.7, 5.4] \bigwedge a_4 = [1.3, 1.8] \rightarrow 2$ $(r = 1)$

6. $a_1 = [4.3, 5.7] \bigwedge a_2 = [3.0, 3.5] \bigwedge a_3 = [1.0, 1.5] \bigwedge a_4 = [0.1, 0.3] \rightarrow 1$ $(r = 16)$

7. $a_1 = [4.3, 5.9] \bigwedge a_2 = [2.8, 4.4] \bigwedge a_3 = [1.5, 1.8] \bigwedge a_4 = [0.1, 0.3] \rightarrow 1$ $(r = 16)$

8. $a_1 = [4.3, 7.2] \bigwedge a_2 = [3.5, 4.4] \bigwedge a_3 = [1.0, 1.5] \bigwedge a_4 = [0.1, 1.8] \rightarrow 1$ $(r = 7)$

9. $a_1 = [4.3, 5.9] \bigwedge a_2 = [2.8, 4.4] \bigwedge a_3 = [1.6, 1.8] \bigwedge a_4 = [0.3, 1.2] \rightarrow 1$ $(r = 4)$

10. $a_1 = [4.3, 4.5] \bigwedge a_2 = [2.1, 2.8] \bigwedge a_3 = [1.0, 1.5] \bigwedge a_4 = [0.1, 0.3] \rightarrow 1$ $(r = 1)$

11. $a_1 = [4.3, 5.9] \bigwedge a_2 = [2.8, 3.0] \bigwedge a_3 = [1.0, 1.5] \bigwedge a_4 = [0.1, 0.3] \rightarrow 1$ $(r = 1)$

12. $a_1 = [4.3, 5.0] \bigwedge a_2 = [3.2, 4.1] \bigwedge a_3 = [1.8, 2.1] \bigwedge a_4 = [0.1, 0.3] \rightarrow 1$ $(r = 1)$

13. $a_1 = [5.0, 6.8] \bigwedge a_2 = [3.5, 4.1] \bigwedge a_3 = [1.8, 2.1] \bigwedge a_4 = [0.3, 0.6] \rightarrow 1$ $(r = 1)$

14. $a_1 = [4.3, 5.6] \bigwedge a_2 = [2.9, 4.4] \bigwedge a_3 = [1.5, 1.6] \bigwedge a_4 = [0.3, 1.8] \rightarrow 1$ $(r = 1)$

15. $a_1 = [5.9, 6.9] \bigwedge a_2 = [2.4, 3.4] \bigwedge a_3 = [5.4, 5.6] \bigwedge a_4 = [1.8, 2.3] \rightarrow 3$ $(r = 8)$

16. $a_1 = [6.5, 7.1] \bigwedge a_2 = [1.8, 3.5] \bigwedge a_3 = [5.7, 6.1] \bigwedge a_4 = [1.1, 2.4] \rightarrow 3$ $(r = 6)$

17. $a_1 = [5.4, 6.7] \bigwedge a_2 = [2.6, 3.3] \bigwedge a_3 = [4.9, 5.7] \bigwedge a_4 = [2.3, 2.4] \rightarrow 3$ $(r = 4)$

18. $a_1 = [6.0, 6.2] \bigwedge a_2 = [2.6, 3.1] \bigwedge a_3 = [4.0, 5.1] \bigwedge a_4 = [1.8, 1.9] \rightarrow 3$ $(r = 3)$

19. $a_1 = [5.8, 7.5] \bigwedge a_2 = [2.4, 3.9] \bigwedge a_3 = [6.1, 6.9] \bigwedge a_4 = [1.1, 2.5] \rightarrow 3$ $(r = 3)$

20. $a_1 = [6.0, 6.5] \bigwedge a_2 = [2.8, 3.4] \bigwedge a_3 = [5.1, 5.4] \bigwedge a_4 = [1.9, 2.2] \rightarrow 3$ $(r = 2)$

21. $a_1 = [7.5, 7.9] \bigwedge a_2 = [3.7, 3.9] \bigwedge a_3 = [6.3, 6.8] \bigwedge a_4 = [1.7, 2.5] \rightarrow 3$ $(r = 2)$

22. $a_1 = [7.1, 7.2] \bigwedge a_2 = [2.0, 3.3] \bigwedge a_3 = [5.7, 6.1] \bigwedge a_4 = [1.1, 2.4] \rightarrow 3$ $(r = 2)$

23. $a_1 = [7.5, 7.8] \bigwedge a_2 = [2.7, 3.7] \bigwedge a_3 = [6.3, 6.8] \bigwedge a_4 = [1.1, 2.5] \rightarrow 3$ $(r = 2)$

24. $a_1 = [5.9, 6.4] \bigwedge a_2 = 3.4 \bigwedge a_3 = [4.8, 6.1] \bigwedge a_4 = [2.3, 2.5] \rightarrow 3$ $(r = 1)$

25. $a_1 = [6.2, 6.3] \bigwedge a_2 = [2.5, 3.2] \bigwedge a_3 = [4.8, 6.1] \bigwedge a_4 = [1.8, 1.9] \rightarrow 3$ $(r = 1)$

26. $a_1 = [6.2, 6.3] \bigwedge a_2 = [2.5, 3.2] \bigwedge a_3 = [4.9, 5.1] \bigwedge a_4 = [1.7, 1.8] \rightarrow 3$ $(r = 1)$

27. $a_1 = [6.7, 6.8] \bigwedge a_2 = [2.0, 3.4] \bigwedge a_3 = [5.5, 5.7] \bigwedge a_4 = [2.4, 2.5] \rightarrow 3$ $(r = 1)$

28. $a_1 = [6.1, 6.7] \bigwedge a_2 = [2.4, 3.0] \bigwedge a_3 = [5.1, 5.4] \bigwedge a_4 = [1.8, 1.9] \rightarrow 3$ $(r = 1)$

29. $a_1 = [5.4, 6.3] \bigwedge a_2 = [2.6, 3.3] \bigwedge a_3 = [5.7, 6.1] \bigwedge a_4 = [2.4, 2.5] \rightarrow 3$ $(r = 1)$

30. $a_1 = [7.5, 7.9] \bigwedge a_2 = [2.4, 3.9] \bigwedge a_3 = [6.8, 6.9] \bigwedge a_4 = [1.7, 2.5] \rightarrow 3$ $(r = 1)$

31. $a_1 = [6.0, 7.0] \bigwedge a_2 = [2.5, 3.5] \bigwedge a_3 = [5.6, 5.7] \bigwedge a_4 = [1.1, 2.3] \rightarrow 3$ $(r = 1)$

A.0.3 Breast Cancer Problem

There are 107 rules in total extracted by GSVMORC. The rules are listed as follows:

1. $a_1 = [1, 5] \bigwedge a_2 = [1, 5] \bigwedge a_3 = [1, 9] \bigwedge a_4 = [1, 2] \bigwedge a_5 = [2, 4] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [1, 8] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 330)$

2. $a_1 = [1, 4] \bigwedge a_2 = [1, 3] \bigwedge a_3 = 1 \bigwedge a_4 = [1, 9] \bigwedge a_5 = 1 \bigwedge a_7 = [1, 3] \bigwedge a_8 = [1, 3] \bigwedge a_9 = [1, 2] \rightarrow 1$ $(r = 33)$

3. $a_1 = [2, 6] \bigwedge a_2 = [1, 5] \bigwedge a_3 = [2, 8] \bigwedge a_4 = [1, 5] \bigwedge a_5 = [1, 9] \bigwedge a_6 = [1, 2] \bigwedge a_7 = [1, 6] \bigwedge a_8 = [1, 7] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 12)$

4. $a_1 = [5, 5] \bigwedge a_2 = [1, 5] \bigwedge a_3 = [1, 3] \bigwedge a_4 = [1, 7] \bigwedge a_5 = [1, 9] \bigwedge a_6 = [1, 2] \bigwedge a_7 = [1, 9] \bigwedge a_8 = [1, 6] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 11)$

5. $a_1 = [1, 3] \bigwedge a_2 = [1, 9] \bigwedge a_3 = [1, 3] \bigwedge a_4 = [1, 6] \bigwedge a_5 = [2, 3] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [1, 5] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 10)$

6. $a_1 = [2, 6] \bigwedge a_2 = [1, 3] \bigwedge a_3 = [1, 2] \bigwedge a_4 = [1, 8] \bigwedge a_5 = [1, 2] \bigwedge a_7 = [1, 1] \bigwedge a_8 = [1, 4] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 5)$

7. $a_1 = [2, 5] \bigwedge a_2 = [1, 3] \bigwedge a_3 = 1 \bigwedge a_4 = [1, 9] \bigwedge a_5 = 2 \bigwedge a_7 = [2, 3] \bigwedge a_8 = [1, 4] \bigwedge a_9 = [1, 3] \rightarrow 1$ $(r = 4)$ 8. $a_1 = [1, 4] \bigwedge a_2 = [1, 9] \bigwedge a_3 = [1, 7] \bigwedge a_4 = [1, 3] \bigwedge a_5 = [3, 10] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 3] \bigwedge a_8 = [1, 6] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 3)$

9. $a_1 = [1, 3] \bigwedge a_2 = [1, 9] \bigwedge a_3 = [1, 2] \bigwedge a_4 = [1, 5] \bigwedge a_5 = [2, 2] \bigwedge a_6 = [4, 6] \bigwedge a_7 = [3, 5] \bigwedge a_8 = [1, 5] \bigwedge a_9 = [1, 2] \rightarrow 1$ $(r = 2)$

10. $a_1 = [1,4] \bigwedge a_2 = [1,4] \bigwedge a_3 = 1 \bigwedge a_4 = [1,8] \bigwedge a_5 = 2 \bigwedge a_7 = [2,3] \bigwedge a_8 = [1,3] \bigwedge a_9 = [1,2] \rightarrow 1$ $(r = 2)$

11. $a_1 = [1,4] \bigwedge a_2 = [1,6] \bigwedge a_3 = [1,3] \bigwedge a_4 = [1,2] \bigwedge a_5 = [2,3] \bigwedge a_6 = [1,4] \bigwedge a_7 = [1,7] \bigwedge a_8 = [1,8] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 2)$

12. $a_1 = [1,7] \bigwedge a_2 = [1,7] \bigwedge a_3 = [1,5] \bigwedge a_4 = [3,6] \bigwedge a_5 = [4,8] \bigwedge a_6 = [2,6] \bigwedge a_7 = [1,4] \bigwedge a_9 = [1,1] \rightarrow 1$ $(r = 2)$

13. $a_1 = [1,3] \bigwedge a_2 = [1,9] \bigwedge a_3 = [1,2] \bigwedge a_4 = [1,3] \bigwedge a_5 = 2 \bigwedge a_6 = [1,4] \bigwedge a_7 = [3,5] \bigwedge a_8 = [1,5] \bigwedge a_9 = [1,2] \rightarrow 1$ $(r = 1)$

14. $a_1 = [1,3] \bigwedge a_2 = [1,9] \bigwedge a_3 = [2,3] \bigwedge a_4 = [1,6] \bigwedge a_5 = [1,3] \bigwedge a_6 = [1,5] \bigwedge a_7 = [2,4] \bigwedge a_8 = [1,5] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 1)$

16. $a_1 = [1,4] \bigwedge a_2 = [2,6] \bigwedge a_3 = [2,3] \bigwedge a_4 = [1,2] \bigwedge a_6 = [1,4] \bigwedge a_7 = [1,7] \bigwedge a_8 = [1,8] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 1)$

17. $a_1 = [2,4] \bigwedge a_2 = [2,4] \bigwedge a_3 = 2 \bigwedge a_5 = 2 \bigwedge a_6 = [4,9] \bigwedge a_7 = [2,2] \bigwedge a_8 = [1,4] \bigwedge a_9 = [2,2] \rightarrow 1$ $(r = 1)$

18. $a_1 = [3,3] \bigwedge a_2 = [3,9] \bigwedge a_3 = [2,3] \bigwedge a_4 = [2,4] \bigwedge a_5 = [2,6] \bigwedge a_6 = [1,9] \bigwedge a_7 = [1,4] \bigwedge a_8 = [2,4] \bigwedge a_9 = [2,3] \rightarrow 1$ $(r = 1)$

19. $a_1 = [4,4] \bigwedge a_2 = [4,5] \bigwedge a_3 = [2,8] \bigwedge a_4 = [2,4] \bigwedge a_5 = [6,8] \bigwedge a_6 = [3,5] \bigwedge a_7 = [1,8] \bigwedge a_8 = [2,5] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 1)$

20. $a_1 = [5,5] \bigwedge a_2 = [5,9] \bigwedge a_3 = [6,8] \bigwedge a_4 = [1,4] \bigwedge a_5 = [5,9] \bigwedge a_6 = [4,8] \bigwedge a_7 = [2,4] \bigwedge a_8 = 4 \bigwedge a_9 = [1,4] \rightarrow 1$ $(r = 1)$

21. $a_1 = [7,7] \bigwedge a_2 = [1,3] \bigwedge a_3 = [2,6] \bigwedge a_4 = [2,4] \bigwedge a_5 = [2,9] \bigwedge a_6 = [1,1] \bigwedge a_7 = [2,8] \bigwedge a_8 = [1,6] \bigwedge a_9 = 1 \rightarrow 1$ $(r = 1)$

22. $a_1 = [3,7] \bigwedge a_2 = [2,5] \bigwedge a_3 = [2,3] \bigwedge a_4 = [4,5] \bigwedge a_5 = [2,3] \bigwedge a_7 = [2,4] \bigwedge a_8 = [3,9] \bigwedge a_9 = [2,3] \rightarrow 1$ $(r = 1)$

23. $a_1 = [2,6] \bigwedge a_2 = [1,2] \bigwedge a_3 = [1,4] \bigwedge a_4 = [2,7] \bigwedge a_5 = [2,3] \bigwedge a_6 = [1,9] \bigwedge a_7 = [1,2] \bigwedge a_8 = [1,9] \bigwedge a_9 = [1,3] \rightarrow 1$ $(r = 1)$

24. $a_1 = [1,5] \bigwedge a_2 = [2,6] \bigwedge a_3 = [1,7] \bigwedge a_4 = [2,3] \bigwedge a_6 = [1,5] \bigwedge a_7 = [2,3] \bigwedge a_8 = [1,9] \bigwedge a_9 = [1,2] \rightarrow 1$ $(r = 1)$

25. $a_1 = [1,5] \bigwedge a_2 = [1,5] \bigwedge a_3 = [1,6] \bigwedge a_4 = [3,5] \bigwedge a_5 = [1,9] \bigwedge a_6 = [1,4] \bigwedge a_7 = [2,6] \bigwedge a_9 = [1,2] \rightarrow 1$ $(r = 1)$

26. $a_3 = [4,9] \bigwedge a_5 = [3,9] \bigwedge a_6 = [10,10] \bigwedge a_7 = [5,9] \bigwedge a_9 = [1,10] \rightarrow -1$ $(r = 41)$

27. $a_1 = [6,10] \bigwedge a_7 = [6,10] \bigwedge a_8 = [2,10] \bigwedge a_9 = [3,9] \rightarrow -1$ $(r = 25)$

28. $a_1 = [6,10] \bigwedge a_6 = [3,9] \bigwedge a_7 = [7,10] \bigwedge a_9 = [1,3] \rightarrow -1$ $(r = 19)$

29. $a_5 = [8, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [7, 10] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 7)$

30. $a_2 = [3, 9] \bigwedge a_3 = [3, 9] \bigwedge a_4 = [6, 10] \bigwedge a_6 = [10, 10] \bigwedge a_7 = [5, 9] \bigwedge a_9 = [1, 10] \rightarrow -1$ $(r = 5)$

31. $a_1 = [7, 10] \bigwedge a_2 = [1, 6] \bigwedge a_3 = [1, 5] \bigwedge a_4 = 1 \bigwedge a_6 = [7, 10] \bigwedge a_7 = [2, 9] \bigwedge a_8 = [1, 9] \bigwedge a_9 = [1, 9] \rightarrow -1$ $(r = 4)$

32. $a_1 = [8, 9] \bigwedge a_2 = [6, 10] \bigwedge a_4 = [1, 9] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 4)$

33. $a_1 = [9, 10] \bigwedge a_3 = [6, 10] \bigwedge a_6 = [8, 10] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [2, 7] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 4)$

34. $a_1 = [8, 10] \bigwedge a_4 = [1, 3] \bigwedge a_6 = [2, 6] \bigwedge a_7 = [3, 7] \bigwedge a_8 = [2, 6] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 4)$

35. $a_1 = [5, 9] \bigwedge a_5 = [1, 8] \bigwedge a_6 = [7, 10] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [3, 9] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 4)$

36. $a_1 = [6, 10] \bigwedge a_2 = [2, 4] \bigwedge a_3 = [3, 10] \bigwedge a_4 = [2, 7] \bigwedge a_5 = [1, 9] \bigwedge a_6 = [10, 10] \bigwedge a_7 = [4, 9] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 3)$

37. $a_1 = [8, 10] \bigwedge a_2 = [2, 7] \bigwedge a_3 = [4, 6] \bigwedge a_4 = [5, 9] \bigwedge a_6 = [8, 10] \bigwedge a_7 = [2, 5] \bigwedge a_8 = [2, 6] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 3)$

38. $a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [4, 5] \bigwedge a_8 = [8, 10] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 3)$

39. $a_1 = [1, 8] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 3)$

40. $a_1 = [9, 10] \bigwedge a_3 = [2, 10] \bigwedge a_4 = [1, 9] \bigwedge a_6 = [1, 3] \bigwedge a_7 = [7, 10] \bigwedge a_8 = [6, 10] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 3)$

41. $a_2 = [6, 10] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 7] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 3)$

42. $a_1 = [9, 10] \bigwedge a_3 = [2, 10] \bigwedge a_4 = [1, 9] \bigwedge a_6 = [1, 8] \bigwedge a_7 = [1, 6] \bigwedge a_8 = [6, 10] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 3)$

43. $a_1 = [1, 6] \bigwedge a_5 = [8, 10] \bigwedge a_6 = [7, 10] \bigwedge a_7 = [6, 10] \bigwedge a_8 = [2, 10] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 3)$

44. $a_3 = [3, 9] \bigwedge a_5 = [5, 9] \bigwedge a_6 = [10, 10] \bigwedge a_7 = [5, 9] \bigwedge a_9 = [1, 10] \rightarrow -1$ $(r = 2)$

45. $a_3 = [5, 9] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 6] \bigwedge a_8 = [7, 9] \bigwedge a_9 = 2 \rightarrow -1$ $(r = 2)$

46. $a_3 = [7, 10] \bigwedge a_4 = [6, 9] \bigwedge a_5 = [2, 5] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 2)$

47. $a_1 = [1, 8] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 8] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [9, 10] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 2)$

48. $a_1 = [5,9] \bigwedge a_3 = [7,10] \bigwedge a_5 = [1,8] \bigwedge a_6 = [9,10] \bigwedge a_7 = [7,9] \bigwedge a_8 = [9,10] \bigwedge a_9 = [1,3] \rightarrow -1$ $(r = 2)$

49. $a_5 = [8,10] \bigwedge a_6 = [9,10] \bigwedge a_7 = [7,10] \bigwedge a_8 = [9,10] \bigwedge a_9 = [9,10] \rightarrow -1$ $(r = 2)$

50. $a_1 = [1,6] \bigwedge a_2 = [8,10] \bigwedge a_3 = [6,9] \bigwedge a_4 = [4,9] \bigwedge a_6 = [3,7] \bigwedge a_7 = [8,10] \bigwedge a_8 = [1,7] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 2)$

51. $a_1 = [1,6] \bigwedge a_5 = [1,8] \bigwedge a_6 = [7,9] \bigwedge a_7 = [7,9] \bigwedge a_8 = [9,10] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 2)$

52. $a_1 = [8,10] \bigwedge a_3 = [2,9] \bigwedge a_4 = [2,4] \bigwedge a_5 = [2,9] \bigwedge a_6 = [6,8] \bigwedge a_7 = [3,5] \bigwedge a_8 = [3,6] \bigwedge a_9 = [3,9] \rightarrow -1$ $(r = 2)$

53. $a_1 = [7,10] \bigwedge a_2 = [2,10] \bigwedge a_3 = [1,8] \bigwedge a_4 = [6,10] \bigwedge a_6 = [9,10] \bigwedge a_7 = [5,6] \bigwedge a_8 = [2,6] \bigwedge a_9 = [3,9] \rightarrow -1$ $(r = 2)$

54. $a_1 = [7,10] \bigwedge a_2 = [2,10] \bigwedge a_5 = [1,8] \bigwedge a_6 = [9,10] \bigwedge a_7 = [9,10] \bigwedge a_8 = [3,9] \bigwedge a_9 = [1,3] \rightarrow -1$ $(r = 2)$

55. $a_1 = [5,9] \bigwedge a_2 = [4,8] \bigwedge a_3 = [2,7] \bigwedge a_4 = 1 \bigwedge a_5 = [2,9] \bigwedge a_7 = [3,9] \bigwedge a_8 = [3,10] \bigwedge a_9 = [3,9] \rightarrow -1$ $(r = 1)$

56. $a_1 = [6,9] \bigwedge a_5 = [2,4] \bigwedge a_6 = [5,9] \bigwedge a_7 = [4,9] \bigwedge a_8 = [10,10] \bigwedge a_9 = [1,9] \rightarrow -1$ $(r = 1)$

57. $a_1 = [8,9] \bigwedge a_2 = 2 \bigwedge a_4 = [1,3] \bigwedge a_6 = [1,2] \bigwedge a_7 = [3,6] \bigwedge a_8 = [2,6] \bigwedge a_9 = [3,9] \rightarrow -1$ $(r = 1)$

58. $a_1 = [8,9] \bigwedge a_2 = 6 \bigwedge a_3 = [6,10] \bigwedge a_4 = [3,5] \bigwedge a_5 = [2,5] \bigwedge a_6 = [9,10] \bigwedge a_7 = [3,5] \bigwedge a_8 = [3,5] \bigwedge a_9 = [2,3] \rightarrow -1$ $(r = 1)$

59. $a_1 = [8,10] \bigwedge a_2 = 6 \bigwedge a_3 = [2,6] \bigwedge a_4 = [3,9] \bigwedge a_5 = [5,9] \bigwedge a_6 = [6,10] \bigwedge a_7 = [3,5] \bigwedge a_8 = [1,2] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 1)$

60. $a_1 = [6,8] \bigwedge a_2 = [4,7] \bigwedge a_3 = [5,6] \bigwedge a_4 = [8,9] \bigwedge a_5 = [4,9] \bigwedge a_6 = [9,10] \bigwedge a_7 = [2,4] \bigwedge a_8 = [3,4] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 1)$

61. $a_1 = [5,9] \bigwedge a_2 = [1,8] \bigwedge a_3 = [6,7] \bigwedge a_4 = [4,9] \bigwedge a_5 = [3,4] \bigwedge a_6 = [1,3] \bigwedge a_7 = [8,9] \bigwedge a_8 = [9,10] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 1)$

62. $a_1 = [5,9] \bigwedge a_2 = [2,3] \bigwedge a_3 = [1,2] \bigwedge a_5 = [4,8] \bigwedge a_6 = [9,10] \bigwedge a_7 = [7,9] \bigwedge a_8 = [1,2] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 1)$

63. $a_1 = [4,5] \bigwedge a_2 = [2,6] \bigwedge a_3 = [3,7] \bigwedge a_4 = [2,6] \bigwedge a_5 = [2,4] \bigwedge a_6 = [8,9] \bigwedge a_7 = [6,7] \bigwedge a_8 = [3,6] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 1)$

64. $a_1 = [2,8] \bigwedge a_2 = [3,9] \bigwedge a_3 = [4,10] \bigwedge a_4 = [7,10] \bigwedge a_5 = [1,2] \bigwedge a_6 = [9,10] \bigwedge a_7 = [4,5] \bigwedge a_8 = [1,2] \bigwedge a_9 = [1,2] \rightarrow -1$ $(r = 1)$

65. $a_1 = [7, 10] \bigwedge a_2 = [2, 8] \bigwedge a_3 = [2, 6] \bigwedge a_5 = [1, 2] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

66. $a_1 = [2, 3] \bigwedge a_2 = [5, 9] \bigwedge a_3 = [1, 7] \bigwedge a_5 = [4, 8] \bigwedge a_6 = [7, 9] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [5, 6] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

67. $a_1 = [8, 9] \bigwedge a_2 = [2, 6] \bigwedge a_5 = [2, 3] \bigwedge a_6 = [2, 9] \bigwedge a_7 = [5, 6] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

68. $a_1 = [8, 9] \bigwedge a_2 = [6, 10] \bigwedge a_3 = [9, 10] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [5, 8] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [7, 8] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

69. $a_1 = [2, 9] \bigwedge a_2 = [2, 6] \bigwedge a_3 = [2, 7] \bigwedge a_4 = [4, 6] \bigwedge a_5 = [4, 9] \bigwedge a_6 = [6, 7] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [7, 9] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

70. $a_1 = [7, 10] \bigwedge a_2 = [2, 8] \bigwedge a_3 = [2, 6] \bigwedge a_5 = [1, 8] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [9, 10] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

71. $a_1 = [8, 9] \bigwedge a_2 = [6, 10] \bigwedge a_3 = [9, 10] \bigwedge a_4 = [1, 9] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [2, 8] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

72. $a_1 = [9, 10] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

73. $a_1 = [1, 9] \bigwedge a_2 = [6, 10] \bigwedge a_3 = [6, 10] \bigwedge a_4 = [9, 10] \bigwedge a_5 = [2, 4] \bigwedge a_6 = [8, 10] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [3, 6] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

74. $a_1 = [7, 10] \bigwedge a_2 = [9, 10] \bigwedge a_3 = [6, 10] \bigwedge a_5 = [3, 8] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [7, 9] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

75. $a_2 = [6, 9] \bigwedge a_3 = [3, 6] \bigwedge a_5 = [4, 9] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [3, 5] \bigwedge a_8 = [6, 7] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

76. $a_1 = [1, 6] \bigwedge a_2 = [8, 10] \bigwedge a_3 = [6, 8] \bigwedge a_4 = [6, 9] \bigwedge a_5 = [1, 8] \bigwedge a_6 = [7, 10] \bigwedge a_7 = [9, 10] \bigwedge a_8 = [2, 3] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

77. $a_2 = [6, 10] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [1, 2] \bigwedge a_7 = [7, 10] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

78. $a_1 = [1, 8] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 8] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [9, 10] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

79. $a_1 = [8, 10] \bigwedge a_3 = [2, 9] \bigwedge a_4 = [3, 4] \bigwedge a_5 = [2, 9] \bigwedge a_6 = [4, 6] \bigwedge a_7 = [3, 5] \bigwedge a_8 = [3, 6] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

80. $a_1 = [9, 10] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [7, 9] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

81. $a_1 = [1, 7] \bigwedge a_2 = [2, 6] \bigwedge a_3 = [5, 9] \bigwedge a_5 = [4, 8] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [9, 10] \bigwedge a_8 = [7, 9] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 1)$

82. $a_2 = [1, 9] \bigwedge a_3 = [6, 10] \bigwedge a_5 = [3, 9] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [5, 6] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

83. $a_1 = [8, 9] \bigwedge a_3 = [6, 9] \bigwedge a_4 = [1, 9] \bigwedge a_5 = [2, 3] \bigwedge a_6 = [2, 5] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

84. $a_1 = [8, 9] \bigwedge a_2 = [6, 10] \bigwedge a_3 = [1, 6] \bigwedge a_4 = [1, 9] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [2, 8] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

85. $a_1 = [5, 9] \bigwedge a_3 = [7, 10] \bigwedge a_4 = [1, 9] \bigwedge a_6 = [1, 2] \bigwedge a_7 = [5, 6] \bigwedge a_8 = [7, 10] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

86. $a_1 = [9, 10] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

87. $a_1 = [1, 8] \bigwedge a_2 = [6, 10] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

88. $a_1 = [8, 9] \bigwedge a_2 = [6, 10] \bigwedge a_4 = [1, 9] \bigwedge a_5 = [4, 10] \bigwedge a_6 = [1, 2] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

89. $a_1 = [8, 9] \bigwedge a_2 = [6, 10] \bigwedge a_6 = [5, 10] \bigwedge a_7 = [1, 4] \bigwedge a_8 = [8, 9] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

90. $a_1 = [5, 9] \bigwedge a_3 = [7, 10] \bigwedge a_6 = [5, 7] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [7, 10] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

91. $a_1 = [1, 5] \bigwedge a_3 = [7, 10] \bigwedge a_5 = [4, 8] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [6, 9] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

92. $a_2 = [6, 10] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 7] \bigwedge a_8 = [1, 2] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

93. $a_2 = [1, 6] \bigwedge a_3 = [4, 9] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [1, 2] \bigwedge a_7 = [1, 7] \bigwedge a_8 = [1, 7] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

94. $a_1 = [1, 5] \bigwedge a_3 = [7, 10] \bigwedge a_5 = [1, 8] \bigwedge a_6 = [7, 9] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [3, 6] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 1)$ 29. $a_2 = [8, 10] \bigwedge a_3 = [6, 9] \bigwedge a_6 = [8, 9] \bigwedge a_7 = [1, 6] \bigwedge a_8 = [6, 10] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

95. $a_1 = [5, 9] \bigwedge a_3 = [7, 10] \bigwedge a_4 = [1, 9] \bigwedge a_6 = [1, 3] \bigwedge a_7 = [7, 9] \bigwedge a_8 = [7, 10] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

96. $a_1 = [9, 10] \bigwedge a_3 = [6, 10] \bigwedge a_6 = [8, 10] \bigwedge a_7 = [1, 5] \bigwedge a_8 = [2, 6] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

97. $a_2 = [8, 10] \bigwedge a_3 = [6, 9] \bigwedge a_4 = [4, 9] \bigwedge a_6 = [1, 3] \bigwedge a_7 = [8, 10] \bigwedge a_8 = [1, 6] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 1)$

98. $a_1 = [1, 9] \bigwedge a_2 = [8, 10] \bigwedge a_3 = [6, 9] \bigwedge a_4 = [1, 9] \bigwedge a_6 = [2, 8] \bigwedge a_7 = [1, 6] \bigwedge a_8 = [9, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

99. $a_5 = [4, 10] \bigwedge a_6 = [9, 10] \bigwedge a_7 = [4, 5] \bigwedge a_8 = [7, 10] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

100. $a_3 = [7, 10] \bigwedge a_5 = [1, 8] \bigwedge a_6 = [7, 9] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [1, 3] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 1)$

101. $a_1 = [8, 10] \bigwedge a_4 = [1, 3] \bigwedge a_6 = [2, 6] \bigwedge a_7 = [3, 6] \bigwedge a_8 = [2, 6] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

102. $a_2 = [6, 10] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [1, 3] \bigwedge a_7 = [7, 10] \bigwedge a_9 = [1, 2] \rightarrow -1$ $(r = 1)$

103. $a_2 = [6, 10] \bigwedge a_4 = [9, 10] \bigwedge a_6 = [1, 5] \bigwedge a_7 = [1, 7] \bigwedge a_8 = [1, 6] \bigwedge a_9 = [2, 3] \rightarrow -1$ $(r = 1)$

104. $a_1 = [1, 6] \bigwedge a_5 = [8, 10] \bigwedge a_6 = [7, 9] \bigwedge a_7 = [7, 10] \bigwedge a_9 = [1, 3] \rightarrow -1$ $(r = 1)$

105. $a_1 = [6, 10] \bigwedge a_7 = [6, 7] \bigwedge a_8 = [2, 9] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$

106. $a_3 = [8, 9] \bigwedge a_4 = [6, 10] \bigwedge a_7 = [1, 6] \bigwedge a_8 = [2, 6] \bigwedge a_9 = [3, 9] \rightarrow -1$ $(r = 1)$

107. $a_1 = [6, 10] \bigwedge a_6 = [3, 9] \bigwedge a_7 = [7, 10] \bigwedge a_9 = [9, 10] \rightarrow -1$ $(r = 1)$