# เคอร์เนลฟังก์ชันสำหรับซัพพอร์ตเวกเตอร์แมชชีน

นางสาว ธนัสนี เพียรตระกูล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2551

# KERNEL FUNCTIONS FOR SUPPORT VECTOR MACHINES

Miss Tanasanee  Phienthrakul

A Dissertation Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic year 2008

|  |  |
|---|---|
| Thesis Title | KERNEL FUNCTIONS FOR SUPPORT VECTOR MACHINES |
| By | Miss Tanasanee Phienthrakul |
| Field of Study | Computer Engineering |
| Advisor | Professor Boonserm Kijsirikul, Ph.D. |

---

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

[illegible] .................................. Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr. Ing)

THESIS COMMITTEE

[illegible] .................................. Chairman
(Professor Prabhas Chongstitvatana, Ph.D.)

[illegible] .................................. Advisor
(Professor Boonserm Kijsirikul, Ph.D.)

[illegible] .................................. Examiner
(Assistant Professor Athasit Surarerks, Ph.D.)

[illegible] .................................. Examiner
(Assistant Professor Yachai Limpiyakorn, Ph.D.)

[illegible] .................................. External Examiner
(Associate Professor Nachol Chaiyaratana, Ph.D.)

ธนัสนี เพียรตระกูล : เคอร์เนลฟังก์ชันสำหรับซัพพอร์ตเวกเตอร์แมชชีน. (KERNEL FUNCTIONS FOR SUPPORT VECTOR MACHINES) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ศ.ดร. บุญเสริม กิจศิริกุล, 145 หน้า.

เคอร์เนลฟังก์ชันถูกนำมาใช้ในซัพพอร์ตเวกเตอร์แมชชีน เพื่อคำนวณค่าผลคูณภายในบนปริภูมิแต่งเติมที่มีมิติสูงขึ้น ประสิทธิภาพของการจำแนกประเภทหรือการประมาณค่านั้น จะขึ้นอยู่กับเคอร์เนลฟังก์ชันที่เลือกใช้ มีเคอร์เนลฟังก์ชันอยู่จำนวนหนึ่งที่นิยมใช้กันมาก เช่น เคอร์เนลเชิงเส้น เคอร์เนลพหุนาม และ เรเดียลเบสิสฟังก์ชัน อย่างไรก็ตาม เคอร์เนลฟังก์ชันที่รู้จักกันดีเหล่านี้ อาจไม่เหมาะสมพอสำหรับปัญหาที่มีความยุ่งยากซับซ้อน หรือปัญหาขนาดใหญ่ งานวิจัยนี้จึงได้เสนอที่จะปรับปรุงประสิทธิภาพของซัพพอร์ตเวกเตอร์แมชชีน โดยใช้การรวมเชิงเส้นแบบไม่เป็นลบของฟังก์ชันที่รู้จักกันดีเหล่านี้ เคอร์เนลฟังก์ชันที่ได้สามารถปรับให้เหมาะสมกับปัญหาที่กำลังสนใจ และให้ผลการจำแนก หรือการประมาณค่าที่ดีในปริภูมิแต่งเติม

จากนั้น กลยุทธ์เชิงวิวัฒน์ ได้ถูกนำมาใช้เพื่อปรับค่าพารามิเตอร์ของซัพพอร์ตเวกเตอร์แมชชีน และ เคอร์เนลฟังก์ชันที่นำเสนอ เพื่อหลีกเลี่ยงปัญหาการสอดคล้องกับข้อมูลสอนมากเกินไป จึงได้ทำการออกแบบฟังก์ชันวัตถุประสงค์ในกระบวนการเชิงวิวัฒน์อย่างระมัดระวัง ฟังก์ชันวัตถุประสงค์ที่นำมาใช้ทดสอบและเปรียบเทียบในงานวิจัยนี้ ได้แก่ ความผิดพลาดบนข้อมูลสอน, การประเมินแบบไขว้บนข้อมูลที่แบ่งย่อย, การประมาณขอบเขตของความผิดพลาดที่แท้จริง, และ การใช้คุณสมบัติความเสถียรของซัพพอร์ตเวกเตอร์แมชชีน วิธีการต่างๆที่นำเสนอถูกทดสอบบนชุดข้อมูลมาตรฐานจำนวนหนึ่ง และยังได้ทำการทดสอบบนปัญหาจริง ซึ่งได้แก่ การจำแนกประเภทตามความรู้สึก และ การรู้จำลายมือเขียน

ยิ่งไปกว่านั้น เคอร์เนลฟังก์ชันที่เกิดจากการรวมกันของหลายๆเคอร์เนลที่มีความยืดหยุ่นมากขึ้นได้ถูกแทนในรูปแบบของต้นไม้ ขั้นตอนวิธีสำหรับสร้างเคอร์เนลฟังก์ชันในรูปแบบต้นไม้นี้มีชื่อเรียกว่า จีพีอีเอส ซึ่งประยุกต์การโปรแกรมเชิงพันธุกรรม และ กลยุทธ์เชิงวิวัฒน์ เพื่อสร้างเคอร์เนลฟังก์ชันแบบผสม และ หาค่าพารามิเตอร์ของเคอร์เนลฟังก์ชันนี้ วิธีการนี้ถูกทดสอบ และ เปรียบเทียบกับตัวจำแนกซัพพอร์ตเวกเตอร์แมชชีนมาตรฐานที่ใช้เคอร์เนลฟังก์ชันพหุนาม และ เรเดียลเบสิสฟังก์ชัน ที่มีการกำหนดค่าพารามิเตอร์หลายๆแบบ

ภาควิชา...วิศวกรรมคอมพิวเตอร์.... ลายมือชื่อนิสิต......น.ส...ธนัสนี......เพียรตระกูล...............
สาขาวิชา..วิศวกรรมคอมพิวเตอร์.... ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก.........[illegible]...............
ปีการศึกษา....2551.........................

## # 4671843021: MAJOR COMPUTER ENGINEERING
KEYWORDS: SUPPORT VECTOR MACHINES / KERNEL METHODS / EVOLUTIONARY ALGORITHMS / RADIAL BASIS FUNCTION KERNEL / KERNEL TREE

TANASANEE PHIENTHRAKUL: KERNEL FUNCTION FOR SUPPORT VECTOR MACHINES. ADVISOR: PROF. BOONSERM KIJSIRIKUL, Ph.D., 145 pp.

Kernel functions are used in support vector machines (SVMs) to compute inner product in a higher dimensional feature space. The performance of classification or approximation depends on the chosen kernel function. There are some popular kernel functions such as linear, polynomial, and radial basis function (RBF) kernels. However, these common kernel functions may not be sufficient for the complex or large problems. This research proposes to improve the performance of SVM by using the non-negative linear combination of these common kernel functions. The obtained kernel functions are more flexible and allow better discrimination or approximation in the feature space.

Then, the evolutionary strategies (ESs) are used for adjusting the parameters of SVM and the proposed kernel functions. In order to avoid the overfitting problem, the objective function in the evolutionary process is carefully designed. Training error, subset cross-validation, the bound of generalization error, and the stability of SVM are considered to be objective functions, and their experimental results are compared in this research. The proposed methods are experimented on benchmark datasets and real world problems, i.e. sentiment classification and handwritten recognition.

Moreover, more flexible combined kernel functions are represented as trees. An algorithm for creating these tree kernel functions is presented, called GPES. This algorithm applies the genetic programming (GP) and the evolutionary strategy (ES) for evolving the hybrid kernel functions and their parameters. The experimental results are compared with a standard SVM classifier using the polynomial and RBF kernels with various parameter settings.

| | |
|---|---|
| Department: .....Computer Engineering....... | Student's Signature: ...Tanasanee Phienthrakul... |
| Field of Study: ..Computer Engineering....... | Advisor's Signature: ......[illegible]...... |
| Academic Year: ....2008........................ | |

# ACKNOWLEDGEMENTS

First and foremost, I express my deep sense of gratitude to my advisor, Professor Dr. Boonserm Kijsirikul, for his continuous guidance and excellent support throughout this research. Since he had accepted me to study in the Doctorial of Philosophy Program, he granted many good opportunities to my life. He provided the financial support for my study, and encourages me for all activities that are the good experiences. During my research period, he helped me to correct all of my publications. His motivation and suggestion have inspired me at times of difficulty. His timely support has helped me at various stages of this work.

I sincerely thank my committee Dr. Prabhas Chongstitvatana, Dr. Athasit Surarerks, Dr. Yachai Limpiyakorn, and Dr. Nachol Chaiyaratana for their valuable comments and encouragement throughout my study at Chulalongkorn University. Moreover, I appreciate my supervisors, Dr. Mikael Boden at School of Information Technology and Electrical Engineering, University of Queensland, Australia and Dr. Manabu Okumura at Precision and Intelligence Laboratory, Tokyo Institute of Technology, Japan, for their valuable suggestion and comments. Moreover, I also would like to thank Ananlada Chotimongkol for proofreading my paper.

I want to thank Thailand Research Fund (TRF), the Royal Golden Jubilee Ph.D. Program (RGJ), the Royal Thai Government (Mahidol University), the 90th Anniversary of Chulalongkorn University Fund (Ratchadaphiseksomphot Endowment Fund), and Department of Computer Engineering for their financial support during all my study and especially during the academic exchange.

I am grateful to many people for help, active and passive support during the experiments and writing of my thesis. I would like to thank the members of Scientific Parallel Computer Engineering (SPACE) Laboratory for their cluster computing and the best care during my experiments. Thanks to the members of Machine Intelligence and Knowledge Discovery (MIND) Laboratory and all friends at Department of Computer Engineering for their comments on my works and their pressure to make me struggle on my thesis.

Finally, I would like to send the special thanks to my dad. He is an important spirit of me. Although he cannot perceive in this acknowledgement and does not have any congratulations to me, I hope he will be pleased with my achievement. Thanks to the God Almighty for sending him to be my lovely dad.

# CONTENTS

Page

Page

# LIST OF TABLES

Page

# LIST OF FIGURES

Page

# CHAPTER I

# INTRODUCTION

Support vector machines (SVMs) are learning algorithms proposed by Vapnik [1, 2], based on the idea of empirical risk minimization principle. They have been widely used in many applications such as pattern recognitions, function approximations, and clustering problems. There are two main reasons why this approach should work. First, the automatic selection of the optimal classifier capacity tailored on the given task is performed by minimizing the generalization error. Secondly, there is the computational shortcut which yields the ability to deal with nonlinear problems. This shortcut is called kernel function.

There are many types of kernel functions and each kernel function is suitable for different tasks. The appropriate kernel function and the suitable parameters of SVM and its kernel function are the great problems in many research studies. Hence, the main target of this research is to improve the performance of existing kernel functions. Moreover, a method for determining the optimal parameters of SVM and its kernel function will be proposed. The motivation, objective, scope, contribution, and research methodology of this research are presented in this chapter.

## 1.1 Motivation

Support vector machines have shown great promise in supervised classification problems including pattern recognition, character recognition, text classification, bioinformatics, image processing and others [3]. Basically, SVM performs a linear separation in an augmented feature space by means of a pre-defined kernel function that satisfies Mercer's theorem [4, 5]. This kernel function maps the input vectors into a very high dimensional space, possibly of infinite dimension, where a linear separation is more probable [5]. Then, a linear separating hyperplane is created by maximizing the margin between two classes in this space. Therefore, the complexity of the separating hyperplane depends on the nature and the properties of the chosen kernel function [5].

There are many types of kernel functions such as linear kernel, polynomial kernel, sigmoid kernel, and radial basis function (RBF) kernel. Although, these kernel functions, especially, polynomial kernel and the RBF kernel, are the most successful kernel functions in many problems, they still have the restrictions in some complex problems. Moreover, each kernel function is suitable for some tasks, and it must be chosen for the tasks under consideration by hand or using prior knowledge [6]. While SVMs have emerged as a powerful classification method in several areas of machine learning and data mining, researchers have had to rely on expert domain knowledge when choosing the kernel function and the parameters [3].

Hence, this research tries to define new kernel functions for SVM on classification and regression problems. The new kernel functions should be flexible with any problems and correspond to the Mercer's theorem. A method for creating these new kernel functions is combining existing kernels. Here, we propose the non-negative linear combination of multiple sub-kernel functions. Each sub-kernel functions are weighted and combined. The weights and the parameters of sub-kernels are the adjustable parameters of this new kernel function. In order to obtain an SVM that has good classification accuracy, a large number of kernel parameters are needed to be evaluated.

In general, these parameters are usually determined by a grid search. The parameters are varied with a fixed step-size in a range of values, but this kind of search consumes a lot of time. Although there are many research studies that attempt to propose the algorithm for the parameter selection, this research proposes to use the evolutionary algorithms for choosing the parameters of SVMs and kernel functions. The evolutionary algorithms are random processes and they can find the solutions in the restricted time. Besides, the genetic programming, which is a kind of the evolutionary algorithms, may help to create new forms of kernel functions. We expect that they will give a better result when compared with the traditional kernel functions.

## 1.2 Objective

The main objective of this study is to analyze and design new kernel functions for support vector machines by combining the existing Mercer’s kernels to improve the performance of classification or approximation on benchmark datasets.

## 1.3 Scope

The scopes of this study are as follows.

1. The proposed kernel functions are the combination of Mercer’s kernels, i.e. polynomial and RBF kernels, by using addition or multiplication. Moreover, these new kernel functions must still correspond to the Mercer’s theorem.

2. The proposed kernel functions are designed and tested for support vector machines in classification tasks and regression tasks.

3. The parameters of SVM and the proposed kernel functions are selected by the evolutionary strategies.

4. Performances of the new kernel functions are compared to those of the traditional kernels, i.e. polynomial and RBF kernels.

5. The percentage error of classification is used for measuring the performance of the proposed methods on classification problems. In regression problems, symmetric mean absolute percentage error (SMAPE) is used for evaluating the proposed methods.

6. The proposed methods are evaluated by 5-folds cross-validation.

7. In order to verify the performance, the proposed methods will be trained and tested on 12 binary classification datasets and 4 regression datasets from the UCI Machine Learning Repository [7].

## 1.4 Contribution

1. This research provides new kernel functions for support vector machines. These new kernel functions are the combination of the traditional kernel functions such as linear kernel, polynomial kernel, and radial basis function kernel. Therefore, these kernel functions are more flexible than the traditional kernels. Besides, these kernel functions could lead to the improvement in the performance of classification or prediction.

2. The evolutionary strategy is applied for selecting the parameters of SVM and the proposed kernel functions. This evolutionary algorithm is improved to be suitable for this parameter selection problem. Moreover, we offer the genetic programming for finding the suitable combined kernel functions and their parameters.

## 1.5 Research Methodology

The procedures of this research can be divided into 5 main steps as literature survey, analysis and design of new kernel functions, testing the proposed kernel functions, validation of results, and conclusion for writing documents. These procedures can be exhibited in Figure 1-1.

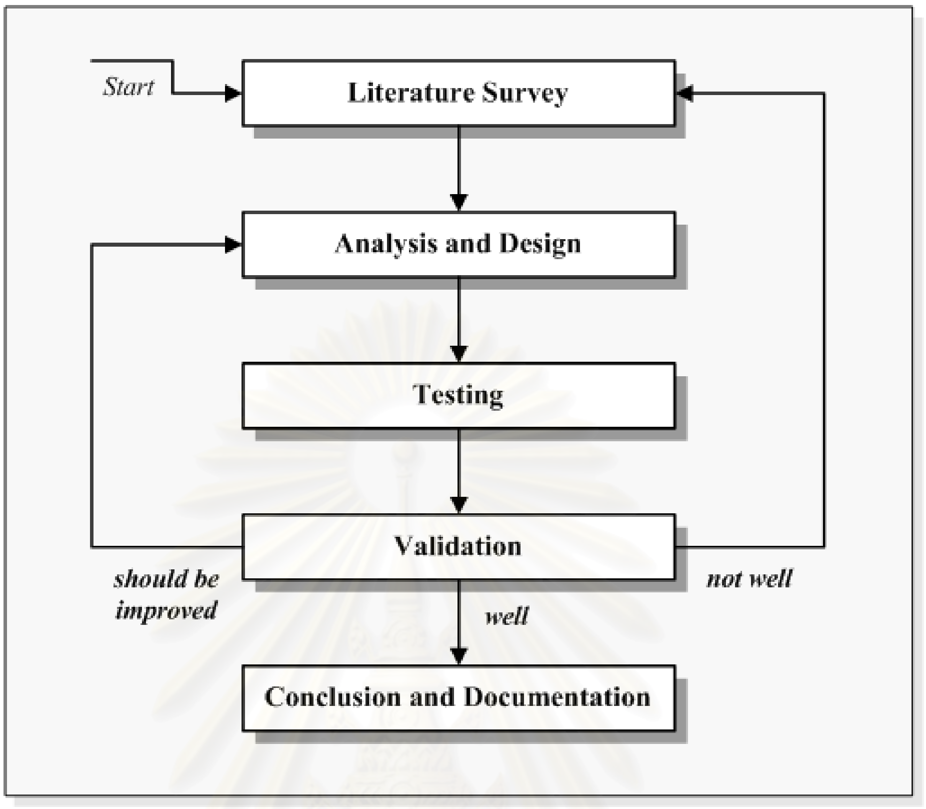


Figure 1-1: Procedures of Research

- **Literature Survey:** The principles of support vector machines are studied. Both support vector classification and support vector regression are reviewed. Application, advantage, and disadvantage of support vector machines are inspected. Then, kernel methods are surveyed. Research studies concerned with kernel functions and their benefits are considered. After that, the theories and applications of evolutionary algorithms (emphasis on evolutionary strategies and genetic programming) are explored. Furthermore, how to apply the evolutionary strategies to our problems are studied.

- **Analysis and Design:** The theories and related research studies that are studied from the previous step are used to analyze and design new kernel functions. The new kernel functions must correspond to the related theorem. The evolutionary algorithms are applied for adjusting the parameters of SVMs and their kernel functions.

- **Testing:** A new kernel function is used for support vector machines on both classification and regression problems. Some standard datasets are selected for testing. The results are compared with the traditional kernels.

**- Validation:** The results from experiments are validated. The information from the first step is used for analyzing results. If the results are not good enough, the proposed method will be improved or re-studying to propose a new method.

**- Conclusion and Documentation:** The proposed kernel and the experimental results are concluded and published in scientific papers. All studies and the recommendation are written in dissertation document.

## 1.6 Organization of Thesis

The remaining chapters of this thesis are organized as follows. The support vector machines will be described in Chapter 2. This algorithm will be considered on both classification problems and regression problems. The kernel methods and learning algorithm measurements are described in this chapter. Then, the evolutionary algorithms are expressed. The evolutionary strategies and the genetic programming are considered in this research. In addition, a number of related works are briefly reviewed.

In Chapter 3, the adaptive combined kernel functions are proposed for support vector machines. There are many ways to combine the Mercer’s kernels. However, this chapter proposes to use the non-negative linear combination of multiple Mercer’s kernels. The multi-scale RBF kernels, the multi-degree polynomial kernels, and the linear combination of both polynomial and RBF kernels are the examples of the combination that are presented in this research. Then, the evolutionary algorithms are applied for adjusting their parameters, and the objective function in evolutionary process is carefully designed.

The experimental setting and results are shown in Chapter 4. The proposed methods are compared to SVMs with the traditional kernels on both classification and regression problems. The performance of the proposed methods is discussed at the end of this chapter. After that, the proposed methods are applied to real world problems in Chapter 5. Sentiment classification and handwritten recognition are considered. Then, a choice to create new kernel functions will also be presented in Chapter 6. The evolutionary algorithms are applied to create these new forms of kernel functions and they are represented as trees. Finally, this research is concluded and some directions for the future work are presented in Chapter 7.

# CHAPTER II

# BACKGROUND AND LITERATURE REVIEW

In this chapter, the mathematical notation is defined. The support vector machines are reviewed on both classification and regression tasks. Then, kernel methods are described and some measurements of learning algorithms are explained. The evolutionary strategies and the genetic programming are illustrated in this chapter. Ultimately, a number of related works are briefly reviewed.

## 2.1 Notation

| | |
|---|---|
| $D$ | training set |
| $y \in Y$ | output and output space |
| $x \in X$ | input and input space |
| $F$ | feature space |
| $\Re$ | real numbers |
| $N$ | dimension of input space |
| $M$ | dimension of feature space |
| $m$ | training set size |
| $w$ | weight vector |
| $b$ | bias |
| $\langle x_i \cdot x_j \rangle$ | inner product of $x_i$ and $x_j$ |
| $d(\cdot)$ | distance function |
| $\lvert \cdot \rvert$ | absolute value |
| $\lVert \cdot \rVert_p$ | $p$-norm, default is 2-norm |
| $\rho$ | margin |
| $\alpha$ | Lagrange multiplier |
| $L$ | Lagrangian form |
| $\frac{\partial L}{\partial x}$ | partial derivative of $L$ with respect to $x$ |
| $f(\cdot)$, $g(\cdot)$ | real-valued function |
| $sign(x)$ | sign of $x$, it is +1 if $x \geq 0$ and it is -1 if $x < 0$ |
| $\xi$, $\xi^*$ | slack variable |

| | |
|---|---|
| $C$ | regularization parameter of SVM |
| $\Phi(\cdot)$ | mapping into feature space |
| $K(x_i, x_j)$ | kernel function $\langle \Phi(x_i) \cdot \Phi(x_j) \rangle$ |
| $n$ | number terms of sub-kernels in combined kernel function |
| $d$ | degree of polynomial kernel |
| $\gamma$ | width of radial basis function kernel |
| $\varepsilon$ | deviation of approximation |
| $L(\cdot)$ | loss function |
| $\ln$ | logarithm to base $e$ |
| $\log_a$ | logarithm to base $a$ |
| $h$ | VC-dimension |
| $H$ | hypothesis space |
| $S$ | set of instances |
| $\delta$ | confidence interval |
| $\hat{y}$ | predicted output |
| $R$ | generalization error |
| $R_{emp}$ | empirical error |
| $\kappa^2$ | bound of kernel function |
| $\mu$ | size of ES parent population |
| $\lambda$ | size of ES offspring population |
| $\vec{v}$ | vector of hyperparameter |
| $\vec{\sigma}$ | vector of standard deviation |
| $P(\cdot)$ | probability function |
| $fitness(\vec{v})$ | objective function of vector $\vec{v}$ or fitness function of $\vec{v}$ |
| $U(A)$ | uniform distribution within set $A$ |
| $N(\mu, \sigma)$ | normal distribution with mean $\mu$ and standard deviation $\sigma$ |

## 2.2 Support Vector Machines

A support vector machine (SVM) is a learning algorithm that can be divided into support vector classification (SVC) and support vector regression (SVR). SVC is a powerful method for separating the binary-class data in terms of a small subset, called support vectors, of the training examples, and SVR is an approximation method that estimates a real-valued function in terms of support vectors. In this part, both SVC and SVR are briefly reviewed.

### 2.2.1 Support Vector Classification

For simple pattern recognition tasks, SVM uses a linear separating hyperplane to create a classifier with the maximum margin [6, 8, 9]. Consider the problem of binary classification, where a training dataset is denoted as

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}, \tag{1}$$

where $x_i \in \Re^N$ is a sample data and $y_i \in \{-1, 1\}$ is its label [10]. A linear decision surface is defined by the following equation:

$$\langle w \cdot x \rangle + b = 0 . \tag{2}$$

Occasionally, there are multiple hyperplanes which can perform the separation. The goal of learning is to find $w \in \Re^N$ and the scalar $b$ such that the margin between positive and negative examples is maximized. An example of the decision surface and its margin are shown in Figure 2-1.

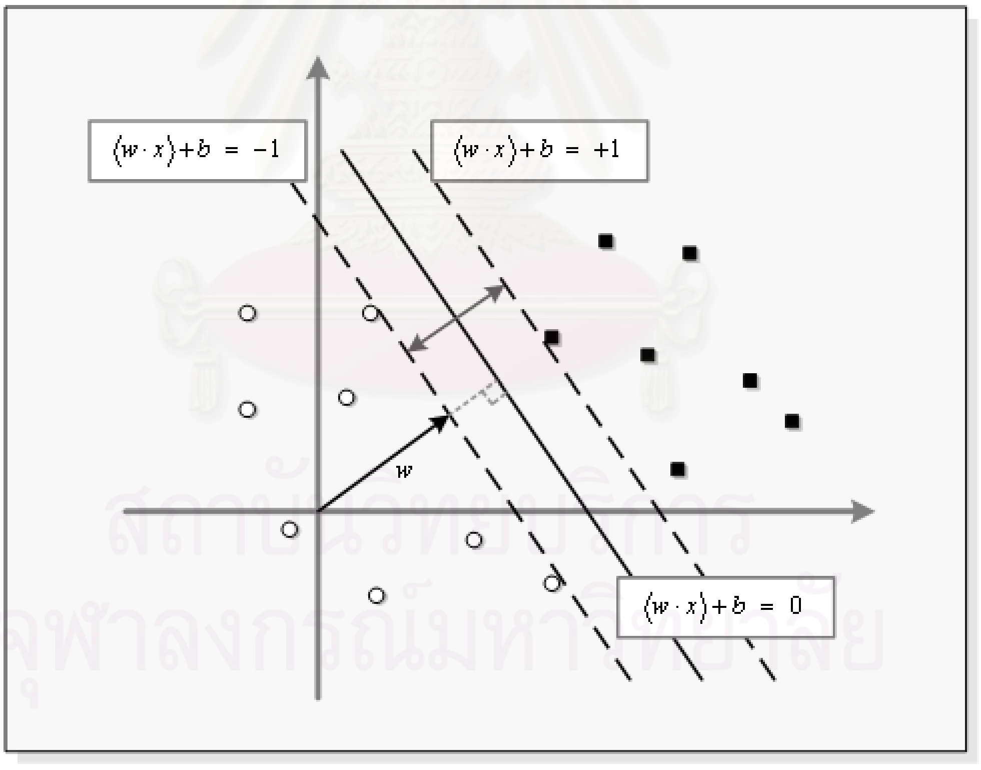

$\langle w \cdot x \rangle + b = -1$
$\langle w \cdot x \rangle + b = +1$
$w$
$\langle w \cdot x \rangle + b = 0$

Figure 2-1: An example of Decision Surface and Margin

The linear hyperplane can separate the data if and only if $\langle w \cdot x_i \rangle + b > 0$ if $y_i = +1$ and $\langle w \cdot x_i \rangle + b < 0$ if $y_i = -1$. It is appropriate to consider a canonical hyperplane, where the parameters $w$ and $b$ are constrained by $\min_i | w \cdot x_i + b | = 1$ [11]. A separating hyperplane in canonical form must satisfy the following constraints:

$$\begin{aligned} \langle w \cdot x_i \rangle + b &\geq +1 \text{ if } y_i = +1, \\ \langle w \cdot x_i \rangle + b &\leq -1 \text{ if } y_i = -1. \end{aligned} \tag{3}$$

These can be combined into one set of inequalities:

$$y_i \left( \langle w \cdot x_i \rangle + b \right) \geq 1, \ \forall i . \tag{4}$$

The distance $d(w,b;x_i)$ of a point $x_i$ from the hyperplane $(w,b)$ is

$$d(w,b;x_i) = \frac{| \langle w \cdot x_i \rangle + b |}{\| w \|}. \tag{5}$$

The optimal hyperplane is given by maximizing the margin $\rho$, subject to the constraints in inequality (4). The margin is given by:

$$\begin{aligned} \rho(w,b) &= \min_{x_i, y_i = -1} d(w,b;x_i) + \min_{x_i, y_i = +1} d(w,b;x_i) \\ &= \min_{x_i, y_i = -1} \frac{| \langle w \cdot x_i \rangle + b |}{\| w \|} + \min_{x_i, y_i = +1} \frac{| \langle w \cdot x_i \rangle + b |}{\| w \|} \\ &= \frac{1}{\| w \|} \left( \min_{x_i, y_i = -1} | \langle w \cdot x_i \rangle + b | + \min_{x_i, y_i = +1} | \langle w \cdot x_i \rangle + b | \right) \\ &= \frac{2}{\| w \|}. \end{aligned} \tag{6}$$

Hence, the hyperplane that optimally separates the data is the one that minimizes $\frac{1}{2} \| w \|^2$ subject to the constraints $y_i \left( \langle w \cdot x_i \rangle + b \right) \geq 1$, for $i = 1, \ldots, m$ [12].

This problem is a quadratic optimization problem [13]. Thus, the Lagrange multipliers $\alpha_i$ , for $i = 1, \ldots, m$ are introduced to form the Lagrangian. This gives Lagrangian:

$$L = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{m} \alpha_i \left( \left( y_i \left( \langle w \cdot x_i \rangle + b \right) \right) - 1 \right). \tag{7}$$

The Lagrangian has to be minimized with respect to $w$, $b$ and maximized with respect to $\alpha_i \geq 0$. The minimum with respect to $w$ and $b$ of the Lagrangian is given by:

$$\frac{\partial L}{\partial b} = 0 \quad \rightarrow \quad \sum_{i=1}^{m} \alpha_i \, y_i = 0 \tag{8}$$

$$\frac{\partial L}{\partial w} = 0 \quad \rightarrow \quad w = \sum_{i=1}^{m} \alpha_i \, y_i \, x_i \,. \tag{9}$$

The optimization problem becomes:

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i \cdot x_j \rangle \\ &\text{subject to} && \sum_{i=1}^{m} \alpha_i \, y_i = 0, \\ & && \alpha_i \geq 0, \; i = 1, \ldots, m. \end{aligned} \tag{10}$$

The decision function is then

$$f(x) = sign\left( \sum_{i=1}^{m} \alpha_i \, y_i \, \langle x_i \cdot x \rangle + b \right), \tag{11}$$

where $\alpha_i \geq 0$ is the coefficient associated with a vector $x_i$ and $b$ is an offset. A data example $x_i$ which corresponds to a non-zero $\alpha_i$ values are called ***support vector***. Figure 2-2 shows the support vectors and the optimal hyperplane.

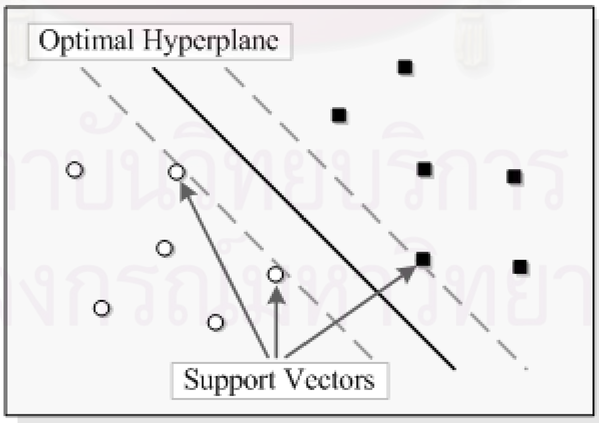


Figure 2-2: Support Vectors and Optimal Hyperplane

However, the quadratic programming solutions cannot be used in the case of overlapping because the constraints cannot be satisfied [6]. Figure 2-3 shows an example of unclassifiable data by a linear hyperplane. In such a situation, this algorithm must allow some data to be unclassified, or on the wrong side of a decision surface [6]. In practice, we allow a soft margin, and all data inside this margin are neglected.

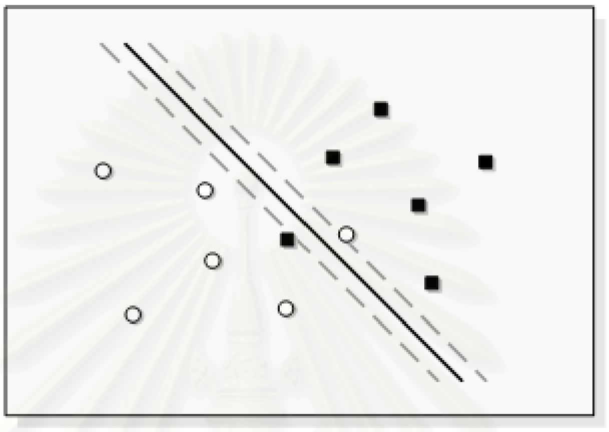


Figure 2-3: Unclassifying by a Linear Hyperplane

In soft margin SVM, the separating hyperplane can be achieved by

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\| w \|^2 + C\sum_{i=1}^{m} \xi_i \\ &\text{subject to} && y_i\left(\langle w \cdot x_i \rangle + b\right) \geq 1 - \xi_i\,, \\ & && \xi_i \geq 0 \text{ for all } i = 1,\ldots,m\,. \end{aligned} \tag{12}$$

The width of the soft margin can be controlled by a corresponding regularization parameter $C$ [6]. The constant $C > 0$ determines the trade-off between margin maximization and training error minimization [14].

In most cases, seeking a proper linear hyperplane in the original input space is not always possible. This problem can be solved by enabling these support vector machines to produce complex nonlinear boundaries in the original space. This is done by mapping the input space into a higher dimensional feature space through a mapping function. Then a linear separating is performed in the higher dimensional space [14]. Example of mapping from 2-dimensional input space into 3-dimensional feature space is shown in Figure 2-4.

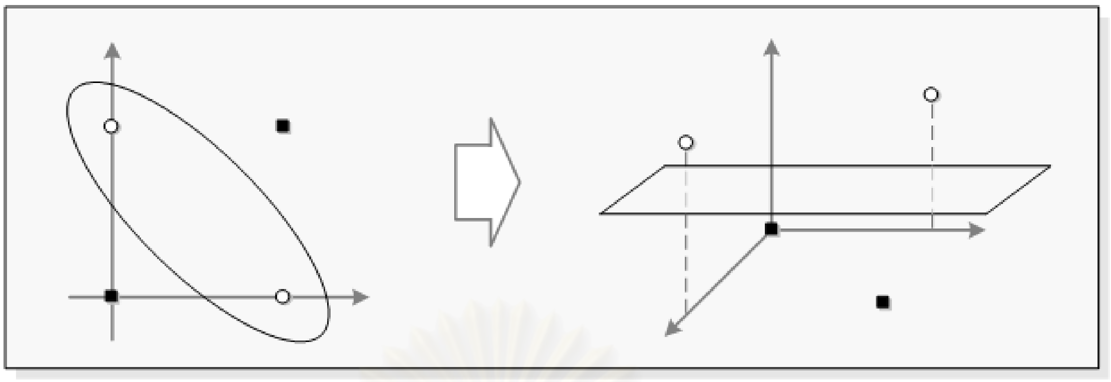


Figure 2-4: Mapping from 2-Dimensional Space into 3-Dimensional Space

This can be achieved by substituting $\Phi(x_i)$ into each training example $x_i$. However, a good property of SVM is that it is not necessary to know the explicit form of $\Phi$. Only the inner product in feature space, called ***kernel function*** $K(x_i, x_j) = \langle \Phi(x_i) \cdot \Phi(x_j) \rangle$, must be defined. This technique is called ***kernel trick***. The decision function becomes the following:

$$f(x) = sign\left( \sum_{i=1}^{m} \alpha_i \, y_i \, K(x_i, x) + b \right), \tag{13}$$

where $K(x_i, x)$ is the kernel function of vector $x_i$ and $x$, $\alpha_i \geq 0$ is a coefficient associated with a support vector $x_i$ and $b$ is an offset. A function which can be a kernel function must satisfy Mercer's theorem [10] that will be described in Section 2.3.

### 2.2.2 Support Vector Regression

In the regression problem, we want to predict a real-valued function. The idea of SVM can be applied to these problems. Suppose our training data are represented as

$$D = \{ (x_1, y_1), \ldots, (x_m, y_m) \} \subset X \times \Re, \tag{14}$$

where $X \subseteq \Re^N$ denotes the space of input patterns. In $\varepsilon$ – SV regression, our goal is to find a function $f(x)$ that has at most $\varepsilon$ deviation from the actually obtained target $y_i$ for all the training data, and at the same time is as flat as possible [15]. In other words, we do not care about errors as long as they are less than $\varepsilon$, but we do not accept any deviation larger than this [15]. The loss function that SVR chooses is $\varepsilon$-insensitive loss function [16], that is

$$L(y, f(x)) = \begin{cases} |y - f(x)| - \varepsilon & if |y - f(x)| > \varepsilon, \\ 0 & otherwise. \end{cases} \tag{15}$$

An example of a linear SVR is shown in Figure 2-5.

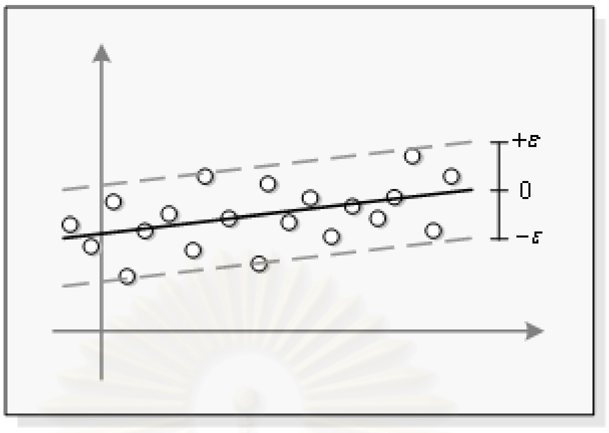


Figure 2-5: An Approximation with a Linear Regression

We begin by describing the case of linear approximation functions. SVR seeks to estimate a function $f$, taking in form:

$$f(x) = \langle w \cdot x \rangle + b, \tag{16}$$

where $w, x \in \Re^N$, $b \in \Re$. Flatness in this case means that one seeks a small $w$. One way to ensure this is to minimize the norm, i.e. $\|w\|^2 = \langle w \cdot w \rangle$.

However, in practice, there always exists various noise or imprecision for our training examples [16]. In those cases, we may want to allow for some errors. Soft margin loss function is adapted to SVR; one can introduce slack variables $\xi_i$, $\xi_i^*$ to cope with otherwise infeasible constraints of the optimization problem [17]. Hence we can write this problem as an optimization problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ &\text{subject to} && y_i - \langle w \cdot x_i \rangle - b \le \varepsilon + \xi_i \\ &&& \langle w \cdot x_i \rangle + b - y_i \le \varepsilon + \xi_i^* \\ &&& \xi_i,\ \xi_i^* \ge 0. \end{aligned} \tag{17}$$

The constant $C > 0$ determines the trade-off between the flatness of $f$ and the amount up to which deviations larger than $\varepsilon$ are tolerated. An example of soft margin linear SVR is shown in Figure 2-6.

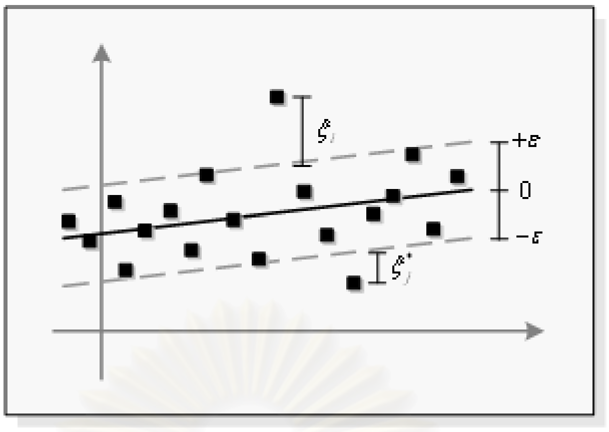


Figure 2-6: Soft Margin for a Linear Regression

For nonlinear problems, SVR maps input space $X$ into a high dimension, or infinite, feature space $F$, and then constructs a linear function with small weight under some constraints in feature space [16]. The feature space $F$ is obtained from mapping function $\Phi(x)$, and SVR uses the following to approximate the learning aim:

$$f(x) = \langle w \cdot \Phi(x) \rangle + b\,. \tag{18}$$

The weight $w \in F$ and bias $b \in \Re$ are gained by optimizing the following problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^{\,*}) \\ &\text{subject to} && y_i - \langle w \cdot \Phi(x_i) \rangle - b \;\le\; \varepsilon + \xi_i \\ &&& \langle w \cdot \Phi(x_i) \rangle + b - y_i \;\le\; \varepsilon + \xi_i^{\,*} \\ &&& \xi_i,\ \xi_i^{*} \;\ge\; 0. \end{aligned} \tag{19}$$

In most cases this optimization problem can be solved more easily in its dual formulation [15]. The dual problem of the quadratic programming in (19) is to

$$\begin{aligned} &\text{maximize} \quad -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right)\left\langle \Phi(x_i)\cdot\Phi(x_j)\right\rangle - \varepsilon\sum_{i=1}^{m}\left(\alpha_i + \alpha_i^*\right) + \sum_{i=1}^{m} y_i\left(\alpha_i - \alpha_i^*\right) \\ &\text{subject to} \quad \sum_{i=1}^{m}\left(\alpha_i - \alpha_i^*\right) \;=\; 0 \\ &\qquad 0 \;\le\; \alpha_i,\ \alpha_i^* \;\le\; C\,. \end{aligned} \tag{20}$$

An example of mapping from input space to feature space is shown in Figure 2-7.

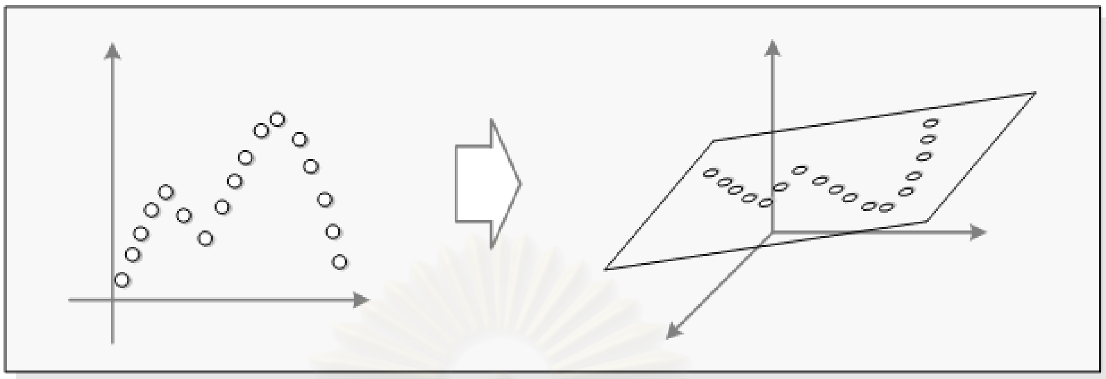


Figure 2-7: Mapping from Input Space into Feature Space

The final decision function is that

$$f(x)=\sum_{i=1}^{m}\left(\alpha_i-\alpha_i^*\right)\left\langle\Phi(x_i)\cdot\Phi(x)\right\rangle+b \tag{21}$$

and

$$w=\sum_{i=1}^{m}\left(\alpha_i-\alpha_i^*\right)\Phi(x_i). \tag{22}$$

According Karush-Kuhn-Tucker (KKT) conditions in [18] and [19], the bias $b$ can be obtained by solving the following four equations [16]:

$$\alpha_i\left(\varepsilon+\xi_i-y_i+\left\langle w\cdot\Phi(x_i)\right\rangle+b\right)=0 \tag{23}$$

$$\alpha_i^*\left(\varepsilon+\xi_i^*+y_i-\left\langle w\cdot\Phi(x_i)\right\rangle-b\right)=0 \tag{24}$$

$$\left(C-\alpha_i\right)\xi_i=0 \tag{25}$$

$$\left(C-\alpha_i^*\right)\xi_i^*=0. \tag{26}$$

The dimension of feature space is always high or even infinite, so it is impossible to compute the inner product directly in feature space [16]. Kernel function is a key step for SVR. Kernel function could make the computation in feature space easier in the original input space [16]. A kernel function is defined on the input space $X\times X$ that is

$$K(x_i,x_j)=\left\langle\Phi(x_i)\cdot\Phi(x_j)\right\rangle, \tag{27}$$

and (21) is changed to

$$f(x) = \sum_{i=1}^{m} \left(\alpha_i - \alpha_i^*\right) K\left(x_i, x\right) + b \,. \tag{28}$$

## 2.3 Kernel Methods

For nonlinear problems, both SVC and SVR utilize kernel techniques to produce complex nonlinear decision functions or nonlinear approximation functions inside the original space.  The relation between SVC, SVR, and kernel functions can be concluded as an image in Figure 2-8.

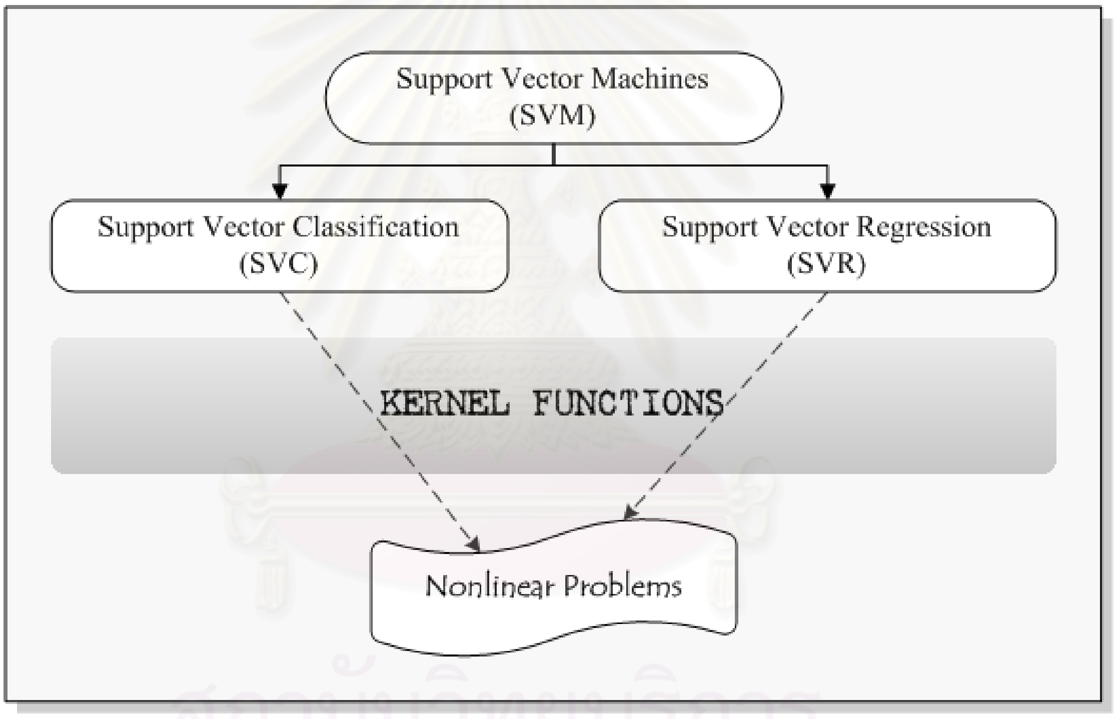


Figure 2-8: Relation between SVC, SVR, and Kernel Functions

The kernel functions map the input vectors in an input space $X \subseteq \Re^N$ into a higher dimensional feature space $F \subseteq \Re^M$, and finding the inner product in this feature space.  The concept of input space, feature space, and output space are shown in Figure 2-9.

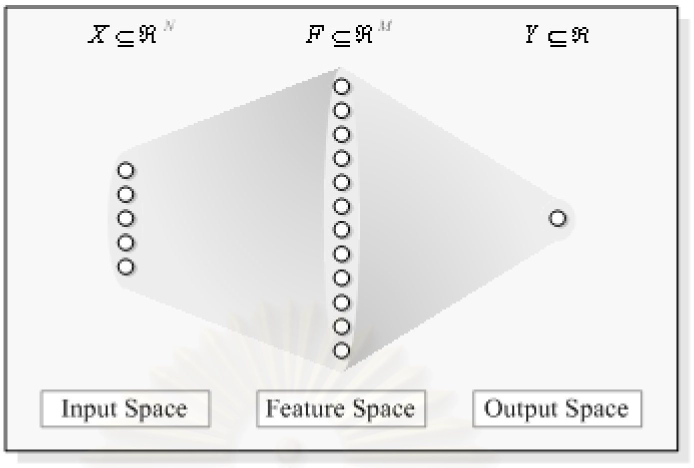


Figure 2-9: Input Space, Feature Space, and Output Space

For instance, the polynomial kernel

$$K(u,v) = \langle u \cdot v \rangle^d \tag{29}$$

can be shown to correspond to a mapping function $\Phi$ into the space spanned by all products of exactly $d$ dimensions of $\Re^N$ [4]. For $d = 2$ and $u, v \in \Re^2$, we have

$$\begin{aligned}\langle u \cdot v \rangle^2 &= (u_1 v_1 + u_2 v_2)^2 = u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 u_2 v_1 v_2 \\ &= \left\langle \left(u_1^2, u_2^2, \sqrt{2}\, u_1\, u_2\right) \cdot \left(v_1^2, v_2^2, \sqrt{2}\, v_1\, v_2\right)\right\rangle \\ &= \langle \Phi(u) \cdot \Phi(v) \rangle\end{aligned} \tag{30}$$

defining

$$\Phi: \ \Re^2 \ \rightarrow \ F = \Re^3, \tag{31}$$

$$(u_1, u_2) \mapsto \left(u_1^2, u_2^2, \sqrt{2}\, u_1\, u_2\right). \tag{32}$$

In SVM, it is not necessary to know the explicit form of mapping function $\Phi : X \rightarrow F$. Only the kernel function must be defined,

$$K(x_i, x_j) = \langle \Phi(x_i) \cdot \Phi(x_j) \rangle . \tag{33}$$

Each kernel corresponds to some feature space and because no explicit mapping to this feature space occurs, optimal linear separators or linear approximations can be found efficiently in the feature space with millions of dimensions [20]. Some of the kernel functions are shown in Table 2-1.

Table 2-1: Examples of Kernel Functions

| **Kernel Functions** | **Formula** |
|---|---|
| Linear | $K(x_i, x_j) = \langle x_i \cdot x_j \rangle$ |
| Polynomial | $K(x_i, x_j) = \left(1 + \langle x_i \cdot x_j \rangle\right)^d$ |
| Exponential Radial Basis Function | $K(x_i, x_j) = \exp\left(-\gamma \left\| x_i - x_j \right\|\right)$ |
| Gaussian Radial Basis Function (RBF) | $K(x_i, x_j) = \exp\left(-\gamma \left\| x_i - x_j \right\|^2\right)$ |
| Multi-Quadratic | $K(x_i, x_j) = -\sqrt{\left\| x_i - x_j \right\|^2 + c^2}$ |
| Sigmoid* | $K(x_i, x_j) = \tanh\left(\alpha \langle x_i \cdot x_j \rangle + \beta\right)$ |
| Thin Plate Spline [21] | $K(x_i, x_j) = \left\| x_i - x_j \right\|^2 \ln \left\| x_i - x_j \right\|$ |
| Moderate Decreasing [5] | $K(x_i, x_j) = k \left[ \exp\left( \dfrac{\gamma}{\left\| x_i - x_j \right\|^2 + \sigma^2} \right) - 1 \right]$ |

* Although the sigmoid function does not correspond to the Mercer's theorem, this function is used in many problems and sometimes it is called two-layer neural network kernel.

We notice that these kernel functions are either ***inner-product-based functions*** or ***distance-based functions***. Linear, polynomial, and sigmoid kernels are the examples of inner-product-based functions. Also, RBF, multi-quadratic, thin plate spline, and moderate decreasing kernels are the examples of distance-based functions. In general, the function which maps the input space into the augmented feature space is not explicitly known. However, the existence of such function is assured by Mercer's theorem [6].

**Theorem 2-1** (Mercer's theorem): Any symmetric function $K(x_i, x_j)$ in the input space can represent an inner product in feature space if

$$\iint K(x_i, x_j)\, g(x_i)\, g(x_j)\ dx_i\, dx_j \quad \geq \quad 0 \tag{34}$$

be valid for all $g \neq 0$ for which $\int g^2(u)\, du \ < \ \infty$ [4, 6].

Then the kernel function $K$ can be expanded in terms of $\Phi_i$

$$K(x_i, x_j) \;=\; \sum_{k=1}^{\infty} \lambda_k \, \Phi_k(x_i) \; \Phi_k(x_j) \tag{35}$$

with $\lambda_k \geq 0$ [6]. In this case, the mapping from input space to feature space is expressed as

$$\Phi : x \rightarrow \left( \sqrt{\lambda_1} \; \Phi_1(x) \; , \; \sqrt{\lambda_2} \; \Phi_2(x), \ldots \right) \tag{36}$$

such that $K$ can be the inner product

$$\Phi(x_i) \cdot \Phi(x_j) \;=\; \sum_{k=1}^{\infty} \lambda_k \, \Phi_k(x_i) \; \Phi_k(x_j) \;=\; K(x_i, x_j) \tag{37}$$

In addition, there are some operations on one or more kernels that always preserve the kernel property. These operations are illustrated in Proposition 2-1.

**Proposition 2-1** (Closure properties): Let $K_1$ and $K_2$ be kernel functions over $X \times X$, $X \subseteq R^N$, $a \in R^+$, $f(\cdot)$ a real-valued function on $X$, $\Phi : X \rightarrow R^M$ with $K_3$ a kernel function over $R^M \times R^M$, and $B$ a symmetric positive semi-definite $N \times N$ matrix. Then the following functions are kernel function [10]:

(i) $K(x_i, x_j) = K_1(x_i, x_j) + K_2(x_i, x_j)$,

(ii) $K(x, x_j) = aK_1(x, x_j)$,

(iii) $K(x_i, x_j) = K_1(x_i, x_j) K_2(x_i, x_j)$,

(iv) $K(x_i, x_j) = f(x_i) f(x_j)$,

(v) $K(x_i, x_j) = K_3(\Phi(x_i), \Phi(x_j))$,

(vi) $K(x_i, x_j) = x_i' B x_j$.

This proposition shows that kernel functions satisfy a number of closure properties [10]. More complicated kernel functions can be created from simple kernel functions by this proposition. Moreover, we can regard that a kernel function is a similarity measure between two data points [10]. Therefore, the kernel functions contain all of the information about the relative positions of the inputs in the feature space.

## 2.4 Measurements of Learning Algorithms

In order to evaluate the learning algorithms, we require some measurements that can compare the performance of several learning algorithms or judge the performance of an algorithm. There are many ways to evaluate our learning algorithms such as training accuracy or training error, VC-dimension, generalization performance, and cross-validation. Besides, for regression problems, there are many error functions that can be applied to the evaluation of the approximation algorithms such as percentage error (PE), mean square error (MSE), mean absolute percentage error (MAPE), etc. In this section, these measurements are briefly described.

### 2.4.1 Accuracy

The accuracy has been widely used as the main criterion for comparing the ability of a learning algorithm. This function measures the number of the correctly classified instances over the total number of instances. In binary classification, the accuracy is also used as a statistical measure of a learning algorithm, and it can be calculated by the following equation:

$$accuracy \;=\; 1-\frac{\sum_{i=1}^{m}\left|f(x_i)-y_i\right|}{2m}, \tag{38}$$

where $f(x_i)$ is a decision function of data $x_i \in \Re^N$ for $i = 1, 2, ..., m$, and $y_i \in \{-1, 1\}$ is the actual class of data $x_i$.

In the context of binary classification tasks, the terms true positive, true negative, false positive, and false negative are used to compare the given class (the class label assigned by a classifier) with the desired class (the actual class). Accuracy is closely related to sensitivity/specificity and precision/recall. This is illustrated by Figure 2-10.

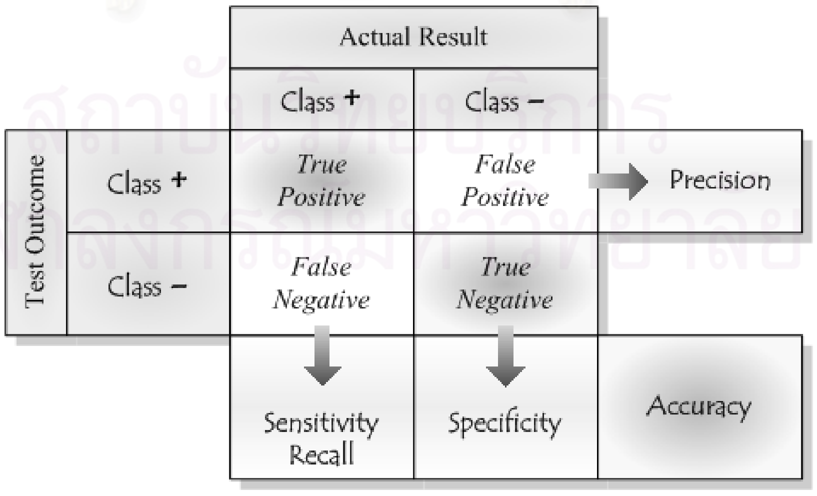


Figure 2-10: Sensitivity/Specificity and Precision/Recall

$$accuracy = \frac{true\ positive + true\ negative}{true\ positive + false\ positive + true\ negative + false\ negative} \tag{39}$$

**- *Precision/Recall***: The precision is the number of class members classified correctly over the total number of instances classified as class members; and the recall (or true positive rate) is the number of class members classified correctly over the total number of class members.

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{40}$$

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{41}$$

**- *Sensitivity/Specificity***: The sensitivity or the recall rate measures the proportion of actual positives which are correctly identified; and the specificity measures the proportion of negatives which are correctly identified.

$$sensitivity = \frac{true\ positive}{true\ positive + false\ negative} \tag{42}$$

$$specificity = \frac{true\ negative}{true\ negative + false\ positive} \tag{43}$$

### 2.4.2 VC-Dimension

The Vapnik-Chervonenkis dimension (VC-dimension: $h$) is a measure of the complexity of hypothesis space $H$. In many case, the VC-dimension can be used to state a tighter bounds on sample complexity. The VC-dimension measures the complexity of the hypothesis space $H$, not by the number of distinct hypotheses $|H|$, but instead by the number of distinct instances from input space $X$ that can be completely discriminated using $H$ [22]. Hence, the VC-dimension is a one-number summary of a learning machine's capacity [14]. The basic concept of the VC-dimension is presented for a two-class pattern recognition problem and then generalized for real approximation function [6].

First, we define the notion of shattering a set of instances. Consider some subset of instances $S \subseteq X$, each hypothesis from $H$ imposes some dichotomy on $S$; that is, each hypothesis partitions $S$ into the two subsets [22]. Given some instance set $S$, there are $2^{|S|}$ possible dichotomies, though $H$ may be unable to represent some of these [22]. We say that

$H$ shatters $S$ if every possible dichotomy of $S$ can be represented by some hypothesis from $H$ [22].

**Definition 2-1**: A set of instance $S$ is ***shattered*** by hypothesis space $H$ if and only if for every dichotomy of $S$ there exists some hypothesis in $H$ consistent with this dichotomy [22].

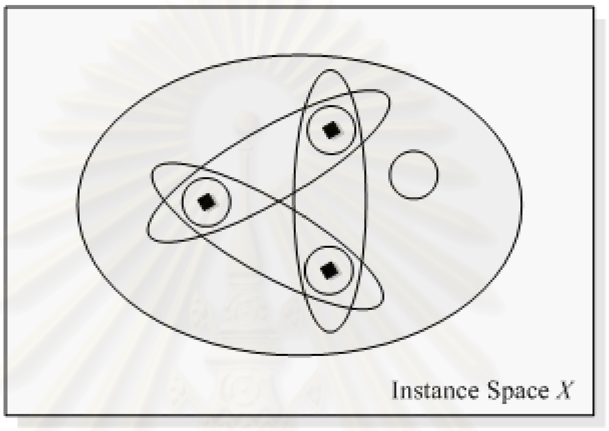


Figure 2-11: An Examples of Shattering

Figure 2-11 illustrates a set $S$ of three instances that is shattered by the hypothesis space. Moreover, each of the $2^3$ dichotomies of these three instances is covered by some hypothesis. If a set of instances is not shattered by a hypothesis space, then there must be some concept that can be defined over the instances but that cannot be represented by the hypothesis space [22].

The ability of $H$ to shatter a set of instances is a measure of its capacity to represent target concepts defined over these instances. In general, $H$ cannot shatter the instance space $X$ but it can shatter some large subset $S$ of $X$ [22]. Therefore, the larger subset of $X$ that can be shattered is reasonable to measure. The VC-dimension of $H$ is precisely this measure.

**Definition 2-2**: The ***VC-dimension*** of hypothesis space $H$ defined over instance space $X$ is the size of the largest finite subset of $X$ shattered by $H$. If arbitrarily large finite sets of $X$ can be shattered by $H$, then $VC(H) \equiv \infty$ [22].

Moreover, for any finite $H$, $VC(H) \leq \log_2|H|$. Note that if the VC-dimension is $h$, then there exists at least one set of $h$ instances in the input space that can be shattered [6]. This does not mean that every set of $h$ instances can be shattered by a given set of hypotheses.

For example, suppose the instance $X$ is the set of points in the two dimensional space, and $H$ is the set of all linear decision surfaces in this space. It is easy to see that any two distinct points in this two dimensional space can be shattered by $H$, because we can find four linear surfaces that include neither, either, or both points [6]. These are showed in Figure 2-12.

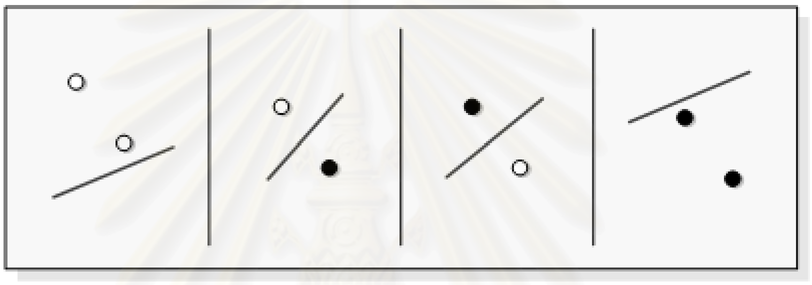


Figure 2-12: Shattering of Two Points

For three points, as long as the points are not co-linear, we can find $2^3 = 8$ linear surface that shatter them. All possible variations of three points shattered by a hypothesis are showed in Figure 2-13. Of course three co-linear points cannot be shattered [22], and they are showed in Figure 2-14. Moreover, there is no arrangement of four points in a two dimensional space which can be shattered by the linear surface that is shown in Figure 2-15. Therefore, the VC-dimension of $H$ is equal to 3.

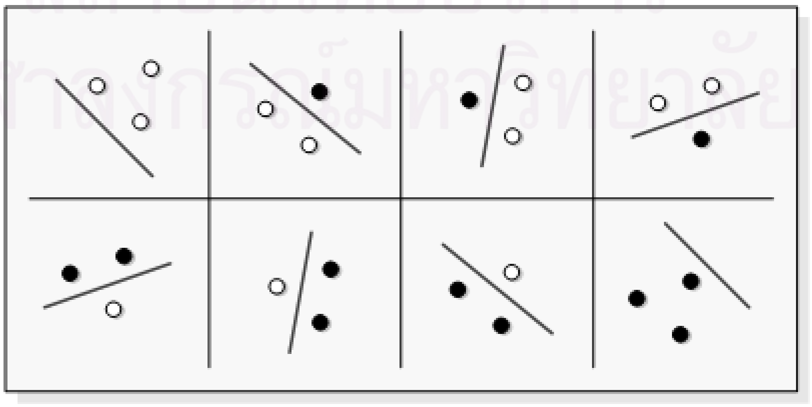


Figure 2-13: Shattering of Three Points

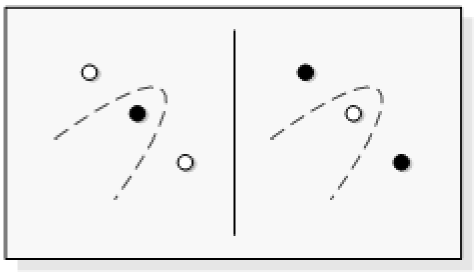


Figure 2-14: Cannot Shatter Three Co-Linear Points

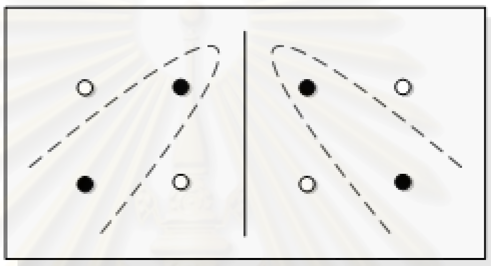


Figure 2-15: Cannot Shatter Four Points

More generally, in $N$ dimensional space, the VC-dimension of linear decision surfaces is $N+1$ [6, 22]. However, SVM can be shown to correspond to hyperplanes in feature spaces of possible infinite dimension [14]. The crucial point is that SVM corresponds to large margin hyperplanes. Once the margin enters, the capacity can be much smaller than the above general VC-dimension of hyperplanes [14]. For simplicity, we consider the case of hyperplanes containing the origin.

**Theorem 2-2**: Consider hyperplanes $\langle w \cdot x \rangle = 0$, where $w$ is normalized such that they are in canonical form with respect to a set of point $X^* = \{x_1, x_2, \ldots, x_m\}$; i.e.

$$\min_{i=1,2,\ldots,m} \left| \langle w \cdot x_i \rangle \right| = 1. \tag{44}$$

The set of decision functions $f(x) = sign(\langle w \cdot x \rangle)$ defined on $X^*$, and satisfying the constraint $\| w \| \le \Lambda$, has a VC-dimension satisfying

$$h \le R^2 \Lambda^2, \tag{45}$$

where $R$ is the radius of the smallest sphere centered at the origin and containing $X^*$ [14].

The proving process of this theorem is shown in [14]. This theorem states that we can control the VC-dimension by controlling the length of the weight vector $\|w\|$. There exists a similar result for the case where $R$ is the radius of the smallest sphere, not necessary centered at the origin, enclosed the data, and where it is allowed for the possibility that the hyperspheres have a nonzero offset $b$ [14]. In this case, we give a simple visualization in Figure 2-16, which shows it is plausible that enforcing a large margin amounts to reducing the VC-dimension [14].

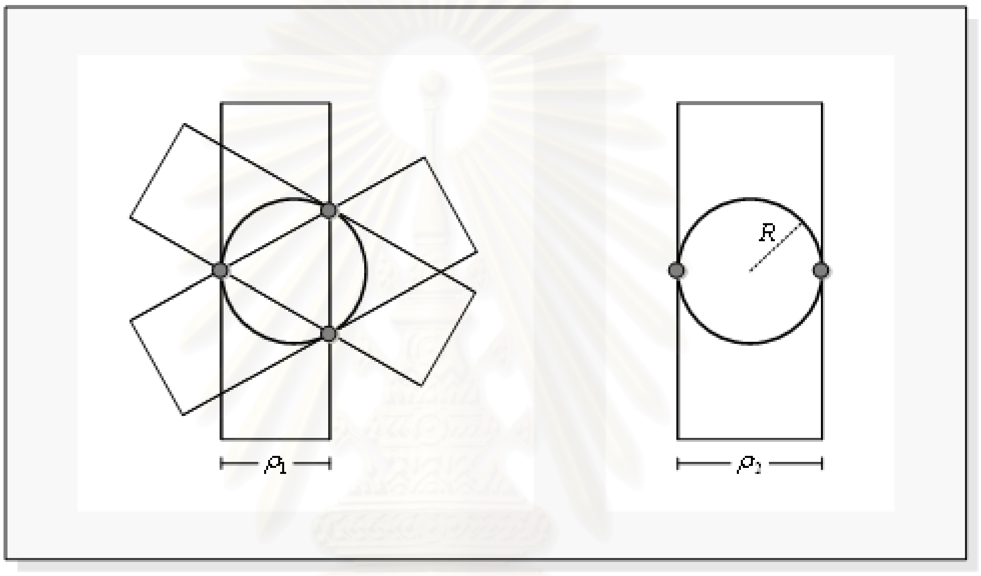


Figure 2-16: Visualization of Enforcing a Large Margin of Separation

Assume that the data points are contained in a ball of radius $R$. The hyperplanes with margin $\rho_1$ can be used to separate three points in all possible ways. However, the hyperplanes with the larger margin $\rho_2$ can be used to separate only two points, hence the VC-dimension in this case is two rather than three [14].

### 2.4.3 Generalization Performance

The result of running the machine learning algorithm can be expressed as a function $f(x)$ which takes a new example $x$ as input and than generates an output value. The precise form of the function $f(x)$ is determined during the training phase on the basis of the training data [23]. The ability to correctly classify new examples that differ from those used for training is known as ***generalization*** [23]. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so generalization is a central goal in pattern recognition [23].

Generalization analysis of a pattern classifier is concerned with determining the factor that affects the accuracy of the classifier [24]. One of the most popular assumptions originally proposed by Vapnik and Chervonenkis is to assume that the training and testing data are both generated according to the same probability distribution [24]. The bound on the generalization error is the probability of misclassifying a randomly chosen example, which holds with high probability over randomly chosen training sets [24]. This type of bound has something of the flavour of a statistical test, in that it allows one to infer that the error is small with the chosen significance level [24]. SVM applies the structural risk minimization principle, which controls both the empirical risk and a confidence interval at the same time [6]. A bound for large margin linear classifiers can be found on the VC-dimension of certain restrictions [24].

**Theorem 2-3**: Define the class $F$ of real-valued functions on the ball of radius $R$ in $\Re^N$ as

$$F = \{ x \mapsto \langle w \cdot x \rangle : \|w\| \le 1,\ \|x\| \le R \}. \tag{46}$$

There is a constant $c$ such that, for all probability distributions, with probability at least $1-\delta$ over $m$ independently generated examples $z$, if a classifier $h = sign(f) \in sign(F)$ has margin at least $\rho$ on all the examples in $z$, then the error of $h$ is no more than

$$\frac{c}{m}\left(\frac{R^2}{\rho^2}\log^2 m + \log(1/\delta)\right). \tag{47}$$

Furthermore, with probability at least $1-\delta$, every classifier $h \in sign(F)$ has error no more than

$$\frac{k}{m} + \sqrt{\frac{c}{m}\left(\frac{R^2}{\rho^2}\log^2 m + \log(1/\delta)\right)}, \tag{48}$$

where $k$ is the number of labeled examples in $z$ with margin less than $\rho$ [24].

Hence the generalization error of the SVM can be bounded even when the kernel determines an infinite dimensional feature space. The various quantities involved in the theorem statement can be calculated [24]. The performance of a machine learning algorithm is measured by plots of the generalization error values through the learning process and is called ***learning curves***. There are many methods for estimating the generalization performance. Various techniques aimed at resolving the trade-off between performance on training data and performance on unseen data. Besides, more classical statistical tool for resolving the trade-off between the performance on training data and the complexity of a model is the cross-validation technique.

### 2.4.4 Cross-Validation

The basic idea of the cross-validation is founded on the fact that good results on the training data do not ensure good generalization capability [6]. Generalization refers to the capacity of a learning model to give correct answer on unseen data. The most commonly used method for estimating generalization performance of a learning algorithm is to reserve a part of the data, called a ***test set*** or ***validation set***, which must not be used in any way during training. The test set must be a representative set of the cases that we want to generalize. After training, the learning models are run on the test set, and the error on the test set provides an unbiased estimate of the generalization error, assumed that the test set was chosen randomly.

The disadvantage of split-sample validation is that it reduces the amount of data available for both training and validation. Cross-validation is an improvement on split-sample validation that allows us to use all of the data for training. Moreover, we often are interested in comparing the performance of several learning algorithms or choosing the parameters of a learning algorithm. If our learning algorithms are trained on the full training set, this can lead to overfitting. Typically, the average of several smaller sets can yield a stronger regularization. Hence, K-folds cross-validation is considered.

This procedure first partitions the data into K disjoint subsets of equal size. Then, the learning algorithms are trained and validated K times (the folds), using each of K subsets in turn as the validation set, and using all remaining data as the training set [22]. Each sample is used exactly once in a validation set, and K-1 times in a training set. The K results from the folds then can be averaged (or combined) to produce a single estimation. The data partitioning of K-folds cross-validation is illustrated in Figure 2-17.

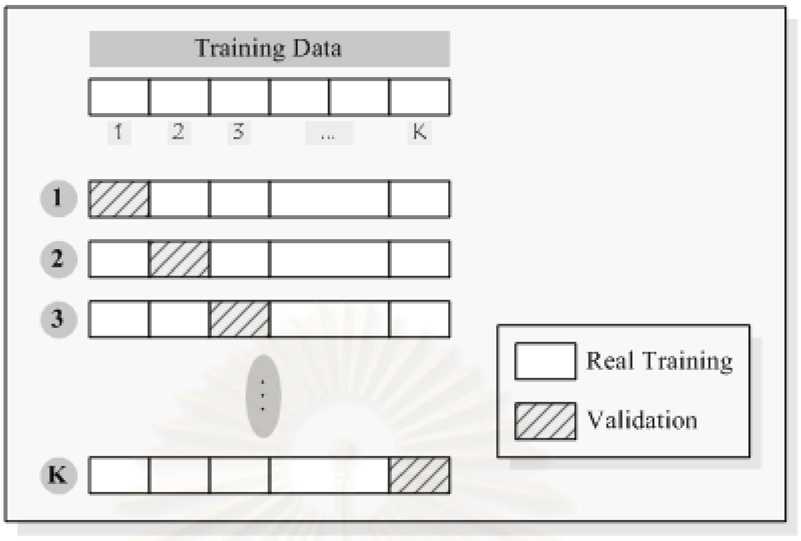


Figure 2-17: K-Folds Cross-Validation

For leave-one-out cross-validation (LOOCV), it uses a single sample as the validation data, and the remaining data as the training set. This is the same as K-folds cross-validation with K is equal to the number of samples in the original training data. Besides, there is the bootstrapping that is an improvement on cross-validation that often provides better estimation of generalization error at the cost of even more computing time.

The advantage of the cross-validation method over repeated subsets of data is that all samples are used for both training and validation, and each sample is used for validation exactly once. If we use cross-validation method to choose which of several learning models, the estimate of the generalization error of the model will be optimistic and we will obtain an unbiased estimate of the generalization error of the model. However, the disadvantage of cross-validation is that the learning algorithms must be re-trained many times.

### 2.4.5 Regression Evaluation

If the desired output consists of one or more continuous variables, then the task is called regression [23]. An example of regression problem is the curve fitting, which the decision stage consists of choosing a specific estimate $\hat{y} = f(x)$ for each input $x$ [23]. When estimating real-valued quantities, the difference values between the actual target values $y_i$ and the prediction values $\hat{y}_i$ are usually used for evaluating the performance of prediction. There are many evaluation functions for regression problems and some of them are illustrated in Table 2-2.

Table 2-2: Examples of Regression Evaluation Functions

| **Regression Evaluation Function** | **Formula** |
|---|---|
| Mean Error (ME) | $ME = \frac{\sum_{i-1}^{m}(y_i - \hat{y}_i)}{m}$ |
| Mean Absolute Error (MAE) or<br>Mean Absolute Deviation (MAD) | $MAE = \frac{\sum_{i-1}^{m}\lvert y_i - \hat{y}_i \rvert}{m}$ |
| Sum of Squared Error (SSE) | $SSE = \sum_{i-1}^{m}(y_i - \hat{y}_i)^2$ |
| Mean Squared Error (MSE) | $MSE = \frac{\sum_{i-1}^{m}(y_i - \hat{y}_i)^2}{m}$ |
| Root Mean Squared Error (RMSE) | $RMSE = \sqrt{\frac{\sum_{i-1}^{m}(y_i - \hat{y}_i)^2}{m}}$ |
| Standard Deviation of Error (SDE) | $SDE = \sqrt{\frac{\sum_{i-1}^{m}(y_i - \hat{y}_i)^2}{m-1}}$ |
| Mean Percentage Error (MPE) | $MPE = \frac{\sum_{i-1}^{m}\left(\frac{y_i - \hat{y}_i}{y_i}\right)\times 100}{m}$ |
| Mean Absolute Percentage Error (MAPE) | $MAPE = \frac{\sum_{i-1}^{m}\left\lvert\frac{y_i - \hat{y}_i}{y_i}\right\rvert\times 100}{m}$ |
| Symmetric Mean Absolute Percentage Error (SMAPE) | $SMAPE = \frac{\sum_{i-1}^{m}\left\lvert\frac{y_i - \hat{y}_i}{(y_i + \hat{y}_i)/2}\right\rvert\times 100}{m}$ |

Note: $m$ is the number of predicted values, $y_i$ and $\hat{y}_i$ for $i = 1,2,...,m$ are the actual target values and the predicted values, respectively.

In statistics, these evaluation functions of an estimator are used to quantify the amount by which an estimator differs from the true value of the quantity being estimated. Minimization of these functions is a key criterion in estimator selection. However, these functions may have the disadvantage of heavily weighting outliers. Especially, in the mean square error (MSE), the squaring of each term effectively weights large errors more heavily than small ones. This property has led researchers to use alternatives such as the mean absolute error (MAE) or the symmetric mean absolute percentage error (SMAPE).

The mean absolute error (MAE) is a common measure of forecast error in regression problems and time series analysis, where the terms mean absolute deviation (MAD) is sometimes used. The mean absolute percentage error (MAPE) is also measure of accuracy in a fitted time series value in statistics, specifically trending. It usually expresses accuracy as a percentage. MAPE is zero when having a perfect fit, but its upper level has no restriction. Moreover, MAPE may have a problem in calculation. The series with the lower numbers may have a very high MAPE, while the higher numbers of series may have a very low MAPE. In order to avoid this problem other measures, such as SMAPE, have been defined.

## 2.5 Evolutionary Algorithms

Evolutionary Algorithms (EA) have found a broad acceptance as robust optimization algorithms in the last ten years [25]. The idea of evolutionary algorithms uses some mechanisms inspired by biological evolution. In ecology, the evolution demonstrates optimized complex behavior at every level: the cell, the organ, the individual, and the population [26]. The biological methods can solve many types of problems, such as chaos, chance, temporality, and nonlinear interactivities [26]. These are also characteristics of problems that have been proved to be especially intractable to classic methods of optimization [26]. The evolutionary process can be applied to problems where heuristic solutions are not available or generally lead to unsatisfactory results [26].

The term “evolutionary computation” is used to describe the field of investigation that concerns all evolutionary algorithms. These evolutionary algorithms imitate nature and apply the ***genetic operators*** such as reproduction, selection, mutation, and recombination. These operations are applied to a population, or several sub-populations, of candidate solutions that are evaluated with respect to their ***fitness*** [27]. Thus, it is possible by an evolutionary loop to successively approximate the optimal state of the system to be investigated [27]. The main flowchart that describes every evolutionary algorithm applied to function optimization is depicted in Figure 2-18. The principle of ***variation*** and ***selection*** can be considered as the fundamental principle of the evolutionary algorithms [25]. These principles, combined with the change of the generation (reproduction), build up the fundamental components of the ***evolutionary loop*** [25].

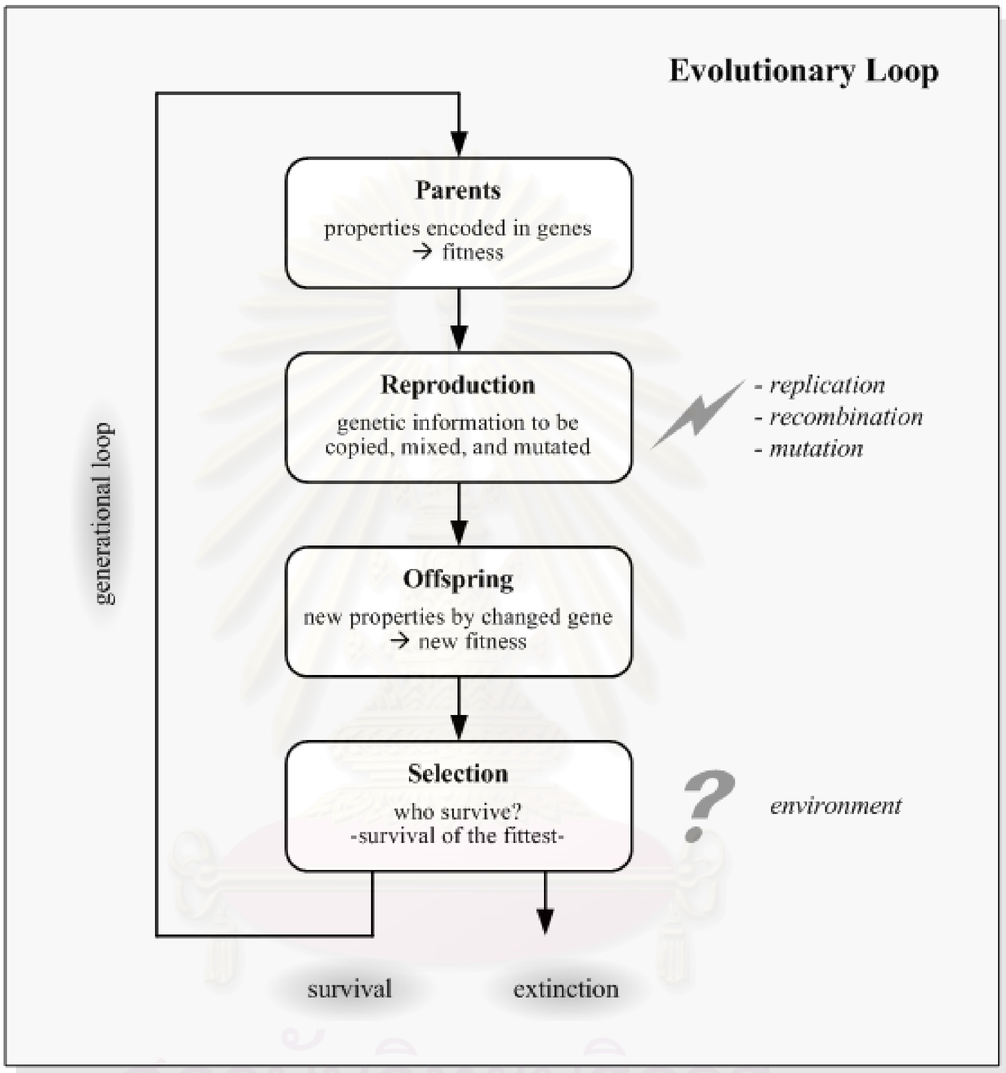


Figure 2-18: General Evolutionary Algorithm [25]

Some of practical advantages to using evolutionary algorithms are summarized [26].

- **Conceptual simplicity**: A primary advantage of evolutionary computation is that it is conceptually simple. The algorithm consists of initialization, which may be a purely random sampling of possible solutions, followed by iterative variation and selection within a number of generations. Thus the criterion needs not be specified with the precision that is required of some other methods [26].

**- Broad applicability**: Evolutionary algorithms can be applied to any problem that can be formulated as a function optimization task. It requires a data structure to represent solutions, a performance index to evaluate solutions, and variation operators to generate new solutions from old solutions. The human designers can choose a representation that follows their intuition. This flexibility allows for applying essentially the same procedure to discrete combinatorial problems, continuous-valued parameter optimization problems, mixed-integer problems, and so forth [26].

**- Outperformance over classic methods on real problems**: Real world function optimization problems often impose nonlinear constraints, require payoff functions that are not concerned with least squared error, involve non-stationary conditions, incorporate noise observations or random processing, or include other vagaries that do not conform well to the prerequisites of classic optimization techniques. Real world problems are often multi-model, and gradient-based methods rapidly converge to local optima or saddle points which may yield insufficient performance. In addition, in case of applying linear programming to problems with nonlinear constraints, this offers an almost certainly incorrect result because the assumptions required for the techniques are violated. In contrast, evolutionary computation can directly incorporate arbitrary constraints [26].

**- Potential to use knowledge and hybridize with other methods**: It is always reasonable to incorporate domain-specific knowledge into an algorithm when addressing particular real-world problems. Specialized algorithms can outperform unspecialized algorithms on a restrict domain of interest. Evolutionary algorithms offer a framework such that it is comparably easy to incorporate such knowledge. For example, specific variation operators may be useful when applied to particular representations. These can be directly applied as mutation or recombination operations. Knowledge can also be implemented into the performance index (or fitness function) [26].

Evolutionary algorithms can also be combined with more traditional optimization techniques. This may be as simple as the use of a conjugate-gradient minimization used for primary search with an evolutionary algorithm, or it may involve simultaneous application of algorithms. There may also be a benefit to seeding an initial population with solutions derived from other procedures. Further, evolutionary computation can be used to optimize the performance of neural networks, fuzzy systems, production systems, and other program structures [26].

**- Parallelism**: Evolution is a highly parallel process. As distributed processing computers become more readily available, there will be a corresponding increased potential for applying evolutionary algorithms to more complex problems. It is often the case that individual solutions can be evaluated independently of the evaluations assigned to competing solutions. The evaluation of each solution can be handled in parallel and only selection requires some serial processing [26].

**- Robustness to dynamic changes**: Traditional methods of optimization are not robust to dynamic changes in the environment and often require a complete restart in order to provide a solution. In contrast, evolutionary algorithms can be used to adapt solutions to changing circumstance. The available population of evolved solutions provides a basis for further improvement and in most cases it is not necessary to reinitialize the population at random [26].

**- Capability for self-optimization**: Most classic optimization techniques require appropriate settings of variables. This is true of evolutionary algorithms as well. However, there are many research studies of using the evolutionary process itself to optimize these parameters as part of the search for optimal solutions [26].

**- Ability to solve problems that have no known solutions**: An advantage of evolutionary algorithms comes from the ability to address problems for which there are no human experts. Although human expertise should be used when it is available, troubles with such expert systems are well known; the experts may not agree, may not be self-consistent, may not be qualified, or may simply be in error. However, most of applications in artificial intelligence require human expertise. They may be impressively applied to difficult problems requiring great computational speed, but they generally do not advance our understanding of intelligence. In contrast, evolutionary provides a method for solving the problem of how to solve problems. It is a recapitulation of the scientific method that can be used to learn fundamental aspects of any measurable environment [26].

The evolutionary algorithms consistently perform well approximating solutions to all types of problems because they do not make any assumption about the underlying fitness. They do not need gradient information and they can operate on each kind of parameter space, e.g. discrete, continuous, combinatorial, or even mixed variants [27]. They are successful in fields as diverse as engineering, industrial, management, art, biology, economics, marketing, operation research, social sciences, physics, politics, chemistry, and genetics. There are many evolutionary algorithm techniques that differ in the implementation details and the nature of the particular applied problem, such as genetic algorithm, genetic programming, evolutionary programming, and evolutionary strategy. In this research, evolutionary strategies and genetic programming are considered.

### 2.5.1 Evolutionary Strategies

The evolutionary strategy (ES) is one of the main branches of evolutionary algorithms, which was developed by Rechenberg and Schwefel [28-32] at the Technical University of Berlin and has been extensively studied in Europe. It was developed in order to conduct successive wing tunnel experiments for aerodynamic shape optimization, and it has been successfully used to solve various types of optimization problems. Moreover, it is significantly faster than traditional genetic algorithms [32, 33, 34]. The natural representation of ES is a fixed-length real-valued vector, which is manipulated primarily by mutation operators designed to perturb the real-valued parameters in useful ways [26]. However, ES practitioners have incorporated recombination operators into their systems. Usually, mutation and recombination have equal importance, as far as real-valued parameter optimization is considered, and they are applied to all individuals by default [25].

#### 2.5.1.1 Basic ES Algorithm

The basic ES algorithm, invented by Rechenberg, Schwefel, and Bienert in the mid-1960s, operates with population $Z$ of size $(\mu+\lambda)$ or $(\mu,\lambda)$ [25]. In these notations, $\mu$ stands for the number of parent individuals and $\lambda$ for the number of offspring. Consider an optimization problem for the fitness function $fitness(\mathbf{a})$ where $\mathbf{a}$ is an $N$-dimensional object parameter vector in the object parameter space $A$, $\mathbf{a} \in A$,

$$\mathbf{a} := (a_1, a_2, \ldots, a_N). \tag{49}$$

An individual consists of an object parameter vector set $\mathbf{a}$, the ***endogenous*** (i.e. evolvable) strategy parameter set $\mathbf{s}$, and its fitness value $fitness(\mathbf{a})$

$$v := (\mathbf{a}, \mathbf{s}, fitness(\mathbf{a})). \tag{50}$$

The endogenous strategy parameter set $\mathbf{s}$ serves for the self-adaptation of the ES algorithm. It does not take part in the calculation of the fitness of the individual; however, it is passed to the offspring depending on the fitness value of the individual [25].

A population consists of $\mu$ parents $v_i$, $i=1,\ldots,\mu$, and $\lambda$ descendants $\tilde{v}_j$, $j=1,\ldots,\lambda$. The parameters $\mu$ and $\lambda$ are ***exogenous*** strategy parameters, i.e. they are not changed by ES [25]. The populations of the parents and the descendants at time $t$ are symbolized as $Z_\mu^{(t)}$ and $\tilde{Z}_\lambda^{(t)}$, respectively.

$$Z_{\mu}^{(t)} = \left\{ v_i^{(t)} \right\} = \left( v_1^{(t)}, \ldots, v_{\mu}^{(t)} \right) \tag{51}$$

$$\tilde{Z}_{\mu}^{(t)} = \left\{ \tilde{v}_j^{(t)} \right\} = \left( \tilde{v}_1^{(t)}, \ldots, \tilde{v}_{\mu}^{(t)} \right) \tag{52}$$

Every point in the search space is an individual. ES uses a population of $\mu$ individuals to conduct the search for possibly better solutions. During each generation, $\lambda$ new individuals are produced by reproduction, recombination, and mutation. This means ES is simultaneously investigating several regions of the search space, which greatly decreases the amount of time required to locate good solutions [35].

There are several different versions of ES. The ($\mu+\lambda$)-ES and ($\mu,\lambda$)-ES are two of the more common versions. In the former, $\mu$ parents produce $\lambda$ offspring. The parents and the offspring compete equally for survival. In the latter, $\mu$ parents produce $\lambda > \mu$ offspring, but only the $\mu$ best offspring survive. Thus the lifespan of any solution is only a single generation [35]. The ($\mu+\lambda$)-ES and ($\mu,\lambda$)-ES can be expressed as the conceptual algorithm in Figure 2-19.

```
Begin
    $t := 0$;
    $initialize\left( Z_{\mu}^{(0)} := \left\{ \left( \mathbf{a}_i^{(0)}, \mathbf{s}_i^{(0)}, fitness\left( \mathbf{a}_i^{(0)} \right) \right) \right\} \right)$;
    Repeat
        For $j := 1$ To $\lambda$ Do Begin
            $\Im_j := reproduction\left( Z_{\mu}^{(0)} \right)$;
            $\mathbf{s}_j := s\_recombination\left( \Im_j \right)$;
            $\tilde{\mathbf{s}}_j := s\_mutation\left( \mathbf{s}_j \right)$;
            $\mathbf{a}_j := recombination\left( \Im_j \right)$;
            $\tilde{\mathbf{a}}_j := mutation\left( \mathbf{a}_j \right)$;
        End;
        $\tilde{Z}_{\lambda}^{(t)} := \left\{ \left( \tilde{\mathbf{a}}_j, \tilde{\mathbf{s}}_j, fitness\left( \tilde{\mathbf{a}}_j \right) \right) \right\}$;
        Case selection_type Of
            $(\mu+\lambda)$ : $Z_{\mu}^{(t+1)} := selection_{\mu}\left( Z_{\mu}^{(t)}, \tilde{Z}_{\lambda}^{(t)} \right)$;
            $(\mu,\lambda)$ : $Z_{\mu}^{(t+1)} := selection_{\mu}\left( \tilde{Z}_{\lambda}^{(t)} \right)$;
        End;
        $t := t+1$;
    Until stop_criterion;
End;
```

Figure 2-19: The ES Algorithm

Each generation (iteration) of the ES algorithm takes a population of individuals (potential solutions) and modifies the problem parameters to produce offspring (new solutions). The generation cycle takes place in the Repeat-Until-Loop (lines 4-18, Figure 2-19). Every descendant is produced step by step in lines 5-12. First, its parents are selected in line 6. If the self-adaptation facility is also implemented, the set of strategy parameters will be treated next [25]. The recombination and the mutation are applied to the set of object parameters in lines 9-10. Then, the new set of object parameters is evaluated according to its fitness. The selection (lines 13-16) follows the procreation of the offspring population. Lastly, the new parent population is built according to the type of the selection used [25]. Thereafter, the evolution starts anew, until the predefined stopping criterion is fulfilled [25]. As termination conditions, the standard stopping rules can be used:

- resource criteria:
    - maximum number of generations,
    - maximum cpu-time
- convergence criteria:
    - in the space of fitness values,
    - in the space of the object parameters,
    - in the space of strategy parameters [32].

#### 2.5.1.2 Fitness and Selection of ES

The first step in the ES algorithm is to initialize the population $Z$, whose the type of components $a_i$ of the object parameter vector $\mathbf{a}$, and the search space $A$ spanned by them, depend on the optimization problem. There are no restrictions to the applicability of the ES algorithm, i.e. all of the alternatives $a_i \in \Re$, or $a_i \in \mathrm{N}$, or $a_i \in \mathrm{B}$ are allowed; moreover, mixed variants of these are realizable, as well as more complex data structures [25]. The initial population of individuals is randomly generated but, ideally, should be uniformly distributed throughout the search space so that all regions may be explored.

The individuals are evaluated to determine their fitness. The goal of having a fitness evaluation is to give feedback to the learning algorithm regarding which individuals should have a higher probability of being allowed to multiply and reproduce and which individuals should have a higher probability of being removed from the population [36]. The fitness function is calculated on what we have earlier referred to as the training set [36]. The fitness function should be designed to give graded and continuous feedback about how well a program performs on the training set [36].

A fitness oriented selection operator is needed for each evolutionary algorithm in order to guide the search into promising regions of the object parameter space [32]. Thus, selection is the antagonist to the variation operators (also referred to genetic operators: mutation and recombination); it gives a direction [32]. Selection in ES is just like breeding; only those individuals with the promising properties, e.g., high fitness values (objective function values), get a chance of reproduction [32]. This selection technique is also called ***truncation*** or ***breeding*** selection. However, there are many choices for the selection techniques, and some of them are described.

- **Truncation Selection**: This selection method is widely used in ES algorithms where it is known as $(\mu,\lambda)$ selection [36]. A number $\mu$ of parents are allowed to breed $\lambda$ offspring, out of which the $\mu$ best are used as parents for the next generation [36]. The same method has been used for a long time in population genetics under the name truncation selection [36]. A variant of ES selection is $(\mu+\lambda)$ selection where, in addition to offspring, the parents participate in the selection process [36]. Plus selection guarantees the survival of the best individual found. Since it preserves the best individual such selection techniques are also called ***elitist*** [32]. Elitism is a sufficient condition for a selection operator should obey on order to prove the ES's global convergence property [32].

- **Fitness-Proportional Selection** [36]: Fitness-proportional selection is employed for generational selection and specifies probabilities for individuals to be given a change to pass offspring into the next generation. An individual $i$ is given a probability of

$$P(i) = \frac{fitness(i)}{\sum_i fitness(i)}. \tag{53}$$

Fitness-proportional selection has been the tool of choice for a long time in the genetic computation. It has been criticized for attaching differential probabilities to the absolute values of fitness.

- **Ranking Selection** [36]: Ranking selection is based on the fitness order, into which the individuals can be sorted. The selection probability is assigned to individuals as a function of their rank in the population. Mainly, linear and exponential ranking are used.

For linear ranking, the probability is a linear function of the rank:

$$P(i) = \frac{1}{N}\left(P^- + \left(P^+ - P^-\right)\left(\frac{i-1}{N-1}\right)\right), \tag{54}$$

where $P^-/N$ is the probability of the worst individual being selected, and $P^+/N$ is the probability of the best individual being selected, and $P^- + P^+ = 2$ should hold in order for the population size to stay constant.

For exponential ranking, the probability can be computed using a selection bias constant $C$:

$$P(i) = \left(\frac{C-1}{C^N - 1}\right)C^N - i \tag{55}$$

with $0 < C < 1$.

**- Tournament Selection** [36]: Tournament selection is not based on competition within the full generation but in a subset of the population. A number of individuals, called the tournament size, is selected randomly, and a selective competition takes place. Better individuals in the tournament are then allowed to replace the worse individuals. In the smallest possible tournament, two individuals compete. A better of the two is allowed to reproduce with mutation. The result of the reproduction is returned to the population, replacing the loser of the tournament.

The tournament size allows the user to adjust selection pressure. A small tournament size causes a low selection pressure, and a large tournament size causes high pressure. Tournament selection has recently a mainstream method for selection, mainly because it does not require a centralized fitness comparison between all individuals. This is not only accelerates evolution considerably, but also provides an easy way to parallelize the algorithm.

There are two main scenarios for generational selection [36]. The first scenario starts with a population of individuals with known fitness and performs a selection of individuals based on their fitness. These are then subjected to variation operations like crossover and mutation or passed on untouched via reproduction into the next generation. In this way, the pool of the following generation is filled with individuals. The next generation usually consists of the same number of individuals as the former one, and fitness computation follows in preparation for another round of selection and breeding. Figure 2-20(a) shows the procedure, known as ***mating selection*** [36].

Another scenario is different. It starts from a given population. A usually larger set of offspring is generated by randomly selecting parents. After fitness evaluation, this population is then reduced to the size of the original population. Thus, a smaller population can be used, as the selection is applied to the pool of offspring, possibly including the parents. Figure 2-20(b) outlines the procedure, also known as ***overproduction selection*** [36].

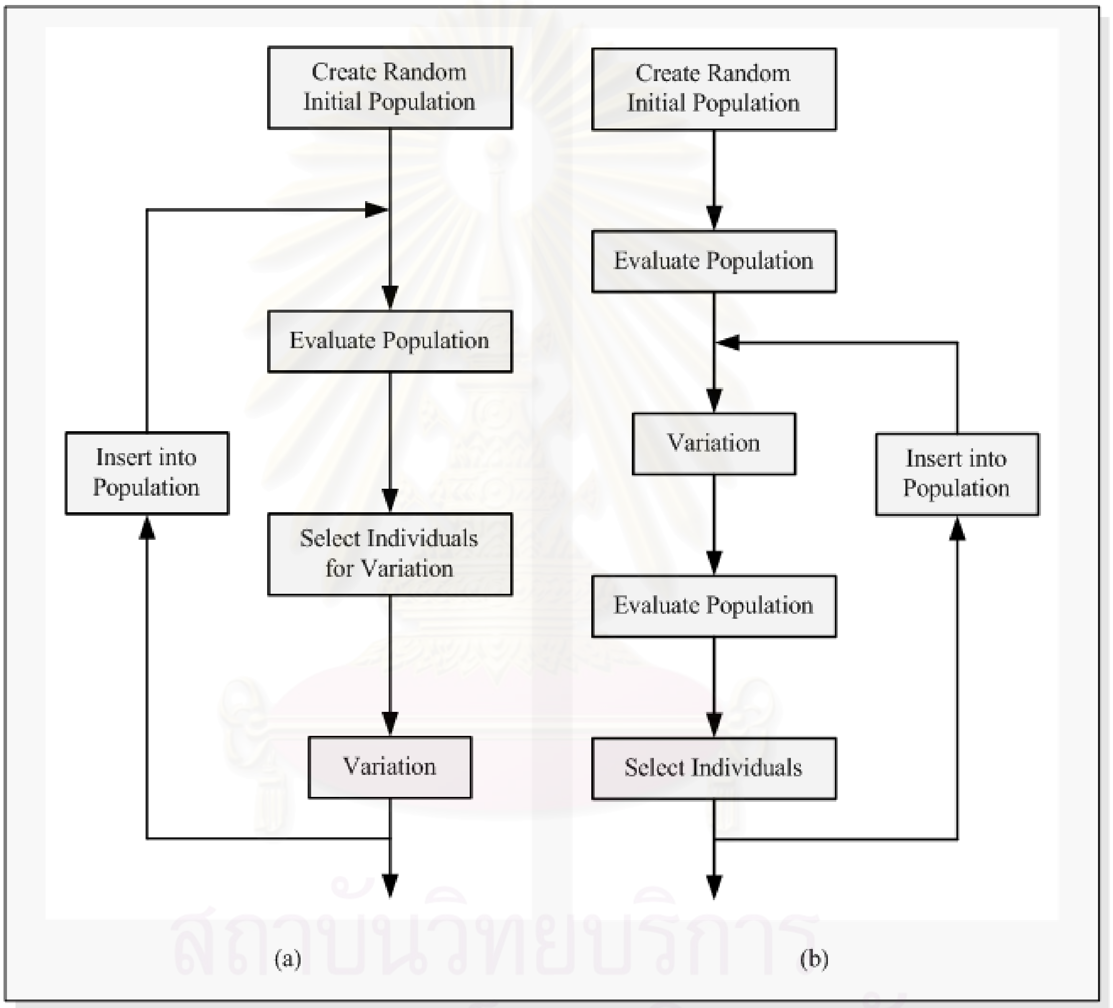


Figure 2-20: Two Selection Scenarios

**2.5.1.3 Genetic Operators of ES**

In order to create a new solution, ES uses two genetic operators, i.e. recombination and mutation. Some solutions are selected from the parent population by a selection method, and then a recombination function is applied. There are several recombining methods as following:

**- No Recombination**: In this method, the new solutions are randomly selected from $\mu$ solutions.

**- Global Intermediary Recombination**: The $\kappa$ solutions are randomly selected from $\mu$ solutions. A new solution $\vec{v}'$ is the average of $\kappa$ solutions.

$$\vec{v}' = \frac{1}{\kappa}\sum \vec{v}_i \tag{56}$$

**- Local Intermediary Recombination**: The $\kappa$ solutions are randomly selected from $\mu$ solutions. Each dimension of new solution ($b_i'$) is the average of two solutions from $\kappa$ selected solutions. Weights of two solutions may be equal or random.

$$\begin{aligned} b_i' &= u_i b_{k_1,i} + (1-u_i) b_{k_2,i} \\ u_i &= 1/2 \quad \text{or} \quad u_i \sim U([0,1]) \\ k_1, k_2 &\sim U(\{1,\ldots,\kappa\}) \end{aligned} \tag{57}$$

$b_{k,i}$ is the $i$ th dimension of solution $k \in \{1,\ldots,\kappa\}$. $U(x)$ is a random function that selects a value from $x$ by uniform random selection.

**- Discrete Recombination**: Randomly select $\kappa$ solutions from $\mu$ solutions. Each dimension of a new solution is randomly selected from $\kappa$ solutions.

$$\begin{aligned} b_i' &= b_{k_i,i} \\ k_i &\sim U(\{1,\ldots,\kappa\}) \end{aligned} \tag{58}$$

After recombination, the new solutions will be mutated. The mutation operator is usually a basic variation operator in ES [32]. The design of mutation operators is problem-dependent. While there is not an established design methodology, some rules have been proposed by analyzing successful ES implementations and by the theoretical considerations on reachability, unbiasedness, and scalability [32].

**- Reachability**: Given a parental state, the first requirement ensures that any other (finite) state can be reached within a finite number of mutation steps or generations [32]. This is also a necessary condition for proving global convergence.

**- Unbiasness**: The mutation is a variation operator. Thus, it should not use any fitness information but the search space information from the parental population [32]. Therefore, there is no preference of any of the selected individuals (parents) in ES, and the variation operators should not introduce any bias [32].

**- Scalability**: The scalability requirement states that the mutation strength or the average length of a mutation step should be tunable in order to adapt to the properties of the fitness landscape. The goal of adaptation is to ensure the evolvability of the ES system [32].

An example of the mutation operator is shown on real-valued search spaces. The standard deviation will be defined for this mutation. Considering $\Re^n$ search spaces and given the standard deviation $\sigma$ (mutation strength), the mutation of $\mathbf{a}$ is

$$mutation(\mathbf{a}) = \mathbf{a} + \mathbf{z}, \tag{59}$$

with

$$\mathbf{z} := \sigma(N_1(0,1), \ldots, N_n(0,1)), \tag{60}$$

where the $N_i(0,1)$ are independent random samples from the standard normal distribution.

The other variation of this mutation can be constructed by considering the standard deviation. This standard deviation can be divided into three cases:

**Case 1:** There is a standard deviation value that will be used for all dimensions of $\mathbf{a}$.

**Case 2:** There are $n$ standard deviation values. Each standard deviation will be used for each dimension of $\mathbf{a}$ (when $\mathbf{a}$ is the $n$-dimensional vector).

**Case 3:** There is a standard deviation vector; each element of this vector is a random number from a distribution.

Besides, in each generation, the standard deviation can be also mutated by another mutation operator.

### 2.5.2 Genetic Programming

Genetic Programming (GP) is a form of evolutionary computation in which the individuals in the evolving population are computer programs rather than bit strings [22]. Programs manipulated by GP are usually represented by a tree structure corresponding to the parse tree of the program. The leaves of tree represent input variables or numerical constants. Their values are passed to nodes, which perform some numerical or program operation before passing on the result further towards the root of the tree [17]. To apply genetic programming to a particular domain, the user must define the primitive functions and the terminals [22]. The genetic programming algorithm then uses an evolutionary search to explore the vast space of programs that can be described using these primitive functions and terminals [22].

**2.5.2.1 Initializing a GP Population** [36]

The first step in actually performing GP is to initialize the population, which means creating a variety of program structures for later evolution. One of the principle parameters of GP is the maximum size permitted for a program. This parameter is expressed as the maximum depth of a tree or the maximum total number of nodes in the tree. The initialization of a tree structure is fairly straightforward. Trees are built from basic units called ***function set*** and ***terminal set***.

The function set is composed of the statements, operators, and functions available to the GP system. The function set may be application-specific and be selected to fit the problem domain. The range of available functions is very broad; it may use any programming construction that is available in any programming language. Some examples follow:

- Boolean Functions: AND, OR, NOT, XOR
- Arithmetic Functions: PLUS, MINUS, MULTIPLY, DIVIDE
- Transcendental Functions: TRIGONOMETRIC, LOGARITHMIC
- Variable Assignment Functions
- Indexed Memory Functions
- Conditional Statements: IF-THEN-ELSE, SWITCH-CASE
- Control Transfer Statements: GOTO, CALL, JUMP
- Loop Statements: WHILE-DO, REPEAT-UNTIL, FOR-DO
- Subroutines

The terminal set is comprised of the inputs to the GP program, the constants supplied to the GP program, and the zero-argument functions with side-effects executed by the GP program. In fact, a terminal lies at the end of every branch in a tree structure. The functions and terminals used for GP should be powerful enough to be able to represent a solution to the problem.

In common, there are two different methods for initializing tree structures, which are called ***full*** and ***grow***. Grow produces trees of irregular shape because nodes are selected randomly from the function and the terminal sets throughout the entire tree, except the root node, which uses only the function set. Once a branch contains a terminal node, that branch has ended, even if the maximum depth has not been reached.

For example, we assume that the terminals and functions are

$$T = \{a,\ b,\ c,\ d,\ e\}, \tag{61}$$

$$F = \{+,\ -,\ \times,\ /,\ \%\}. \tag{62}$$

Figure 2-21 illustrates a tree that is initialized with grow method. This tree represents the function $((a-b)\ \%\ c)\times(d+e)$.

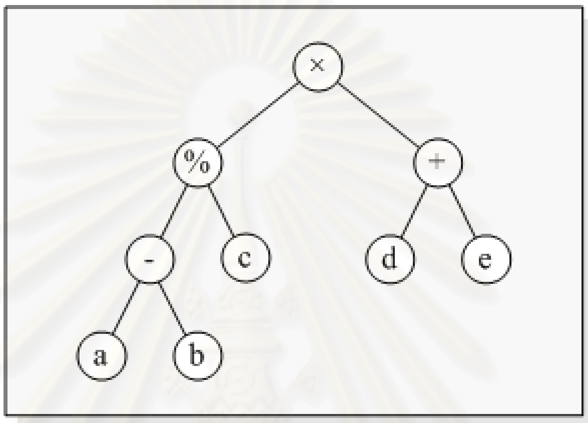


Figure 2-21: An Example of Tree - Initialized by the Grow Method

Instead of selecting nodes randomly from the function and the terminal sets, the full method chooses only functions until a node is at the maximum depth. Then, it chooses only terminals. The result is that every branch of the tree goes to the full maximum depth. If the number of nodes is used as a size measure, growth stops when the tree has reached the preset size parameter. The tree in Figure 2-22 has been initialized by the full method with a maximum depth of three. This tree represents the function $(a\%b)\times(c+d)$.

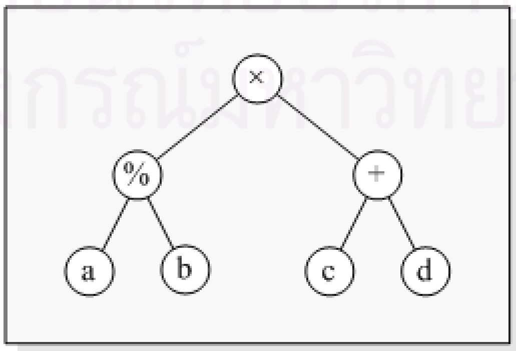


Figure 2-22: An Example of Tree – Initialized by the Full Method

**2.5.2.2 Genetic Operators of GP** [22, 36]

In evolutionary process, GP produces a new generation of individuals using crossover, mutation, and reproduction on each iteration. These three principal GP genetic operators are described.

- **Crossover**: The crossover operator combines the genetic material of two parents by swapping a part of one parent with a part of the other. Tree-based crossover is described graphically in Figure 2-23. The parents are shown in the upper half of the figure while the children are shown in the lower half.

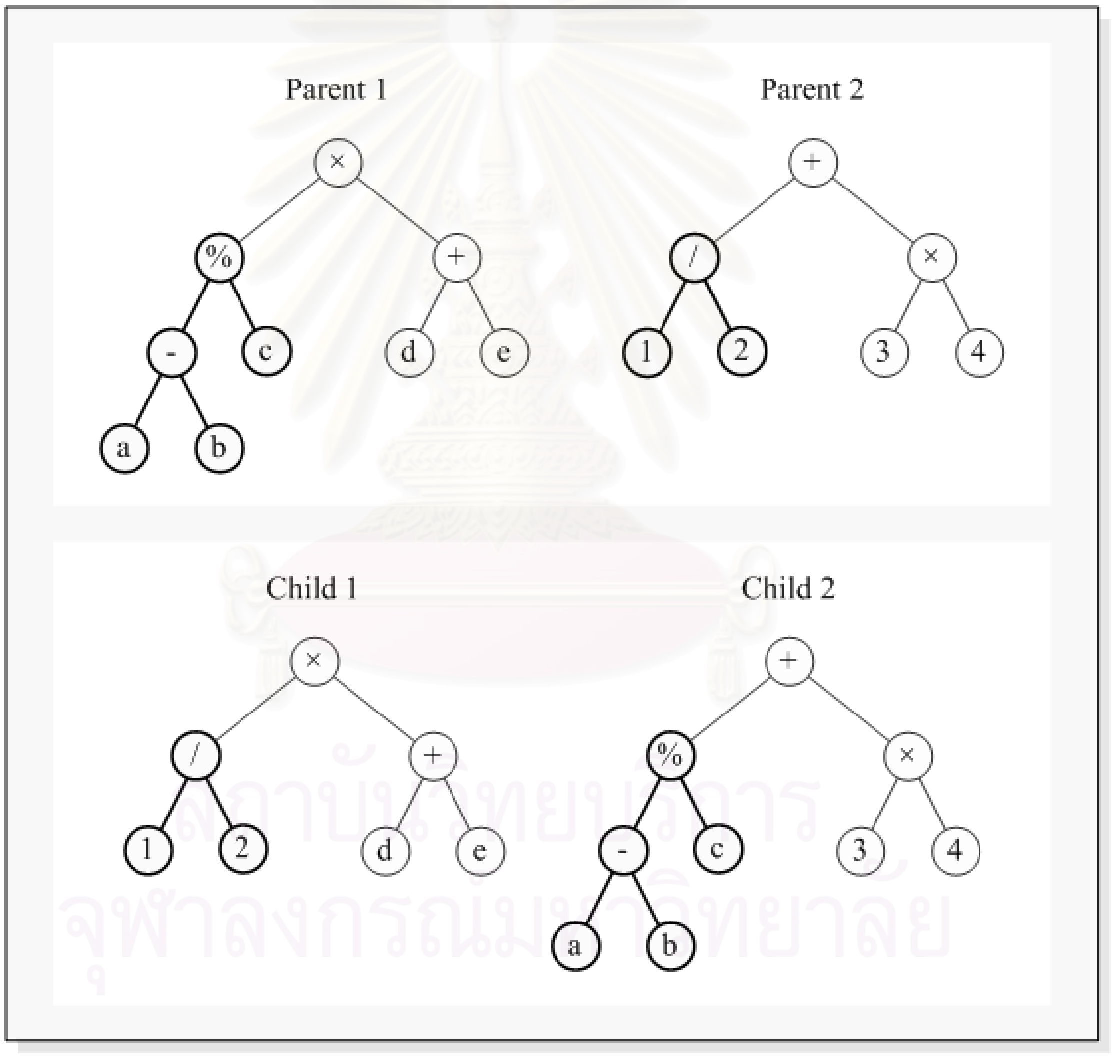


Figure 2-23: Crossover Operation on Two Parent Trees

The tree-based crossover in Figure 2-23 proceeds by the following steps:
-- Choose two individuals as parents, based on mating selection policy, e.g., fitness-proportional selection.
-- Select a random sub-tree in each parent. In Figure 2-23, the selected sub-trees are shown highlighted with darker lines.
-- Swap the selected sub-trees between two parents. The resulting individuals are the children. They are shown at the bottom of Figure 2-23.

- **Mutation**: Mutation operates on only one individual. Normally, after crossover has occurred, each child produced by the crossover undergoes mutation with a probability. The probability of mutation is a parameter of the GP run. When an individual has been selected for mutation, a point of the tree is randomly selected. Then, the existing sub-tree at that point is replaced with a new randomly generated sub-tree. The mutated individual is then placed back into the population.

- **Reproduction**: The reproduction operator is straightforward. An individual is selected. It is copied, and the copy is placed into the population. Hence, there are two versions of the same individual in the population.

#### 2.5.2.3 Fitness and Selection of GP [36]

GP must choose the members of the population for applying the genetic operators such as crossover, mutation, and reproduction. In making the choice, GP implements one of the most important parts of its model of the evolutionary learning, fitness-based selection. Fitness is the measure used by GP during simulated evolution of how well a program has learned to predict the outputs from the inputs. Fitness functions are very problem-specific.

One simple fitness function is to calculate the sum of the absolute value of the differences between actual output of the program and the output given by the training set. A common alternative fitness function is to calculate the sum of the squared differences between the actual output and the prediction, called the squared error. There are also other methods for calculating fitness. In co-evolution methods for fitness evaluation, individuals compete against each other without an explicit fitness value. In game-playing application, the winner in a game may be given a higher probability of reproduction than the loser. Moreover, in some cases, two different populations may be evolved simultaneously with conflicting goals.

After the quality of an individual has been determined by applying a fitness function, we have to decide whether to apply genetic operators to that individual and whether to keep it in the population or allow it to be replaced. This task is called selection and assigned to a special operator, the selection operator. There are various different selection operators, and a decision about the method of selection is one of the most important decisions to be made in a GP run. Selection is responsible for the speed of evolution and influence to the success of an evolutionary algorithm. Some details of selections have been described in ES. Those methods of selection can be applied for GP in the same way.

#### **2.5.2.4 Basic GP Algorithm** [36]

The preliminary steps for making a GP run are defining the terminal set, the function set, and the fitness function. Then, the parameters are defined such as population size, maximum individual size, crossover probability, mutation probability, selection method, and termination criterion, e.g., the maximum number of generations. For the basic GP run, it uses a generational evolutionary algorithm. Each generation is represented by a complete population of individuals. An entire new generation is created from the old generation in one cycle. The new generation replaces the old generation and the cycle continues. The execution cycle of the GP algorithm includes the following algorithms.

```
1   Begin
2       $t := 0$;
3       $initialize\ P(t)$;
4       $evaluate\ P(t)$;
5       Repeat
6           $t := t+1$;
7           $select\ P(t)\ from\ P(t-1)$;
8           $recombine\ P(t)$;
9           $evaluate\ P(t)$;
10      Until stop_criterion;
11  End;
```

Figure 2-24: The GP Algorithm

The algorithm in Figure 2-24 shows the structure of a basic GP algorithm. $P(t)$ denotes the population at generation $t$. The population is recombined through crossover and mutation process. The evolutionary strategies and the genetic programming will be applied to this research in order to optimize the parameters of kernel functions or search the optimal kernel functions.

## 2.6 Related Works

There are many research studies related to support vector machines, kernel methods, parameter selection, and evolutionary algorithms. However, this research focuses on the combined kernel functions and the parameter adjustment. Thus, the related works will be divided into two topics, i.e. construction of kernel functions and parameter selection.

### 2.6.1 Construction of Kernel Functions

Many research studies introduce the kernel methods and support vector machines. *An Introduction to Kernel-Based Learning Algorithm* [9] is an example of research about kernel methods. This paper was proposed by Müller, Mika, Rätsch, Tsuda, and Schölkopf. They introduced support vector machines, kernel Fisher discriminant analysis, and kernel principle component analysis. They gave background about Vapnik-Chervonenkis theory and kernel feature spaces. They illustrated the usefulness of kernel algorithms by discussing applications such as optical character recognition (OCR) and DNA analysis.

In order to construct a new kernel function, many research studies have been proposed to improve the kernel methods with the mathematical techniques. Zhang, Zhou, and Jiao have proposed *Support Vector Machines Based on Scaling Kernels* [37]. In this research study, scaling kernels were presented. These kernels are a multi-dimensional scaling function with translation vectors. SVMs based on scaling kernels can approximate any objective function in some space. After that, they proposed *Wavelet Support Vector Machine* [38]. The wavelet kernel was also a kind of multi-dimensional wavelet functions that can approximate arbitrary nonlinear functions. The existence of wavelet kernels was proved by the result of theoretic analysis. Computer simulations showed the feasibility and validity of wavelet support vector machines in regression and pattern recognition.

Fuzzy logic is another approach that was applied to kernel methods in several manners. *Fuzzy Kernel Perceptron* [39] that was proposed by Jiun-Hung Chen and Chu-Song Chen is a research in the area of fuzzy logic. This research incorporated the fuzzy perceptron (FP) and the Mercer's kernels. FP was adopted to find a linear separating hyperplane using the fuzzy theory so that vectors of high uncertainty have less influence on the training results. In this paper, FP was extended to become the fuzzy kernel perceptron (FKP) with the help of Mercer's kernels. The experiments compared FKP with the kernel perceptron, FP, and SVM.

Moreover, in *Fuzzy Support Vector Machines* [40], a fuzzy membership was applied to each input vector and the SVMs were reformulated. The different input vectors made different contributions to the learning of decision surface. After that, this work was extended by Zonghai Sun and Youxian Sun in *Fuzzy Support Vector Machine for Regression Estimation* [41]. They provided the fuzzy SVM for regression problems to construct the multi-layer SVM. In the first layer, a fuzzy membership was applied to each data point and SVMs were reformulated. The second layer was the generalize SVM which made the kernel function may not satisfy the Mercer's condition. Besides, in [42], the fuzzy logic was applied to the weight vector and the bias terms of SVM, and the desired outputs in training samples were also fuzzy numbers.

In research of Ayat et al. [5], a new SVM kernel family was purposed. *KMOD-A New Support Vector Machine Kernel with Moderate Decreasing for Pattern Recognition* [5] explained the distinctive properties that allowed better discrimination in feature space. The experimental results showed that KMOD was better than RBF kernel and exponential RBF kernels on the spiral problem. In addition, a digital recognition task was processed using their kernel. The results show comparable performances to state-of-the-art kernels.

Moreover, there are many research studies that propose new kernel functions for support vector machines such as hyperkernels [43], triangular kernel [44], asymmetric kernel [45], and time-alignment kernel [46]. These kernels are suitable for some applications or some datasets. Although these new kernel functions yields better results, they are not widely used in practical applications. Many research studies and various applications still use the common kernels such as linear, polynomial, and RBF kernels. The research of Debnath and Takahashi (2004) [47] described that several authors used the RBF kernel and it was always the best for their application according to the various experimental results. Moreover, they tried to improve only the polynomial kernel to compare with the RBF kernel, whereas the other kernels were not considered.

However, there are few research studies that consider combination of well-known kernels. *Improved SVM Regression using Mixtures of Kernels* [48] is an example proposed by Smits and Jordaan. This research showed that the RBF kernel had good interpolation properties while the polynomial kernel had better extrapolation abilities. Therefore, they combined the advantages of polynomial and RBF kernels by using mixtures.

Another paper about combined kernels is *Support Vector Machine with a Hybrid Kernel and Minimal Vapnik-Chervonenkis Dimension* [21]. This paper presents a mechanism to train support vector machines with a hybrid kernel and minimal Vapnik-Chervonenkis dimension. The paper developed a hybrid kernel function and a sufficient condition to be an admissible Mercer kernel based on common Mercer kernels such as polynomial, radial basis functions, and two-layer neural networks. Experimental results show that SVM with the hybrid kernel outperforms that with a single common kernel in terms of generalization power.

The genetic programming is used to evolve a kernel for an SVM classifier in the research of Howley and Madden (2005) [17], but this approach does not guarantee that the kernel obey Mercer's theorem. In *Kernel Trees for Support Vector Machines* [49], Methasate and Theeramunkong applied the genetic programming on the basic kernel functions for creating a new kernel function, which it is vary similar to a part of our research. Then, their approach used the gradient search to find the set of optimal parameters.

Although they explained that each kernel tree was evaluated by calculating its fitness on 5-folds cross-validation in the genetic programming process, they did not describe the detail of fitness computation. Moreover, the method to estimate the parameters of kernel trees in the evolutionary process was not illustrated; the kernel trees with the different parameters should yield the difference results and the different fitness scores, while the parameters of a kernel tree will be tuned by the gradient search after the best kernel tree was selected.

In this research, we proposed an algorithm for generating the hybrid kernel functions and their parameters at the same time. Our algorithm proposes to combine the genetic programming and the evolutionary strategy; the similar processes from these both algorithms are merged in a new algorithm. The obtained hybrid kernel functions are corresponding to the Mercer's theorem and they are more flexible to the problems under consideration.

### 2.6.2 Parameter Selection

The parameter selection is a problem in the learning algorithms. There are many research studies that attempt to solve the problem of parameter selection for SVM by using meta-heuristic methods. In a work of Chapelle et al., (2002) [50], an algorithm that alternates the SVM optimization with a gradient step is employed. Although this algorithm is useful and accurate, there are a lot of details in the computation that make this algorithm quite complex. This algorithm required a gradient computation which for general kernel functions might either not be possible or at least be very difficult. Moreover, the gradient descent may get stuck in local optima.

Most of research studies on parameter selection used the evolutionary algorithms such as the genetic algorithms (GA) or the evolutionary strategies (ES). Eads et al. proposed *Genetic Algorithms and Support Vector Machines for Time Series Classification* [51]. This paper introduces a hybrid algorithm that employs evolutionary computation for feature extraction, and a support vector machine for classification. They tested the proposed algorithm on a lightning classification task. It yielded better results in terms of cross-validation fitness measure, although the difference was not large.

In *Feature Selection for Support Vector Machines by Means of Genetic Algorithms* [52] by Fröhlich, Chapelle, and Schölkopf, a special genetic algorithm was proposed to solve the feature selection problem that is a difficult combinatorial task in machine learning. They optimized kernel parameters such as the regularization parameter of SVM by means of genetic algorithms. Moreover, Xuefeng and Fang (2002) [53] and Chunhong and Licheng (2004) [54] have proposed other research studies that used GA for SVM parameter selection.

Furthermore, the evolutionary strategies are still used in many applications. In research of deDoncker, Gupta, and Greenwood (1996) [55], the evolutionary strategies were used for computing solutions of multivariate integration problems. Adaptive integration algorithms and evolutionary strategies were able to be parallelized easily. Many research studies use the evolutionary strategies for model selection. A method proposed by Friedrichs and Igel [35] chooses parameters of SVM by using evolutionary strategy. The covariance matrix adaptation evolution strategy (CMA-ES) was used to determine a kernel from a parameterized kernel space and to control the regularization. The ES method proposed in this paper was simpler; the random process was used to find the optimal parameters and only recombination and mutation methods were used to create new solutions. Their experiments on benchmark datasets show that ES improved the results achieved by grid search and was able to handle much more kernel parameters.

There are other evolutionary research studies on model selection for support vector machines such as *Asynchronous Parallel Evolutionary Model Selection for Support Vector Machines* (Runarsson and Sigurdsson 2004) [56] and *Multi-objective Model Selection for Support Vector Machines* (Igel 2005) [57]. Both of these research studies attempt to use the evolutionary strategies for optimizing parameters of SVMs. Runarsson and Sigurdsson [56] have proposed asynchronous parallel evolution strategy for the model selection of SVM, and Igel [57] has proposed to use the multi-objective in the evolutionary algorithm. The evolutionary strategies were successfully applied to their applications and datasets. Therefore, the evolutionary strategy is an interesting algorithm for adjusting parameters in our research.

The model-based global optimization [58] was proposed by Fröhlich and Zell in 2005 to deal with the model selection problems. This research is based on the idea of learning an online Gaussian process using a sampling technique to search the solutions in parameter space. Besides, in *Optimal Parameter Selection in Support Vector Machines* [59], a nonlinear programming algorithm is an optimization algorithm which was applied to the parameter selection for SVM. The particle swarm optimization (Guo, et al. 2008) [60] is another meta-heuristic algorithm that was used for adjusting the hyperparameter of SVM. This method does not need any priori knowledge and can be used to determine multiple hyperparameters at the same time. Although the concept of particle swarm is different from the evolutionary algorithm, it is a dynamic system and uses a fitness function to evaluate the candidate solutions, which are similar to the evolutionary algorithm.

We notice that the parameter selection of SVM is an optimization problem in the large continuous search space, and the evolutionary algorithms and the dynamic systems are only the meta-heuristic algorithms that were used to solve this problem. We do not have any knowledge about the suitable kernel function and its parameters, thus only the performance on classification or approximation of SVM is used to guide the search algorithms. For the fitness function in the evolutionary process, many research studies use the classifier to measure the classification accuracy or error on the training data.

However, Eads, et al. [51] compared the classification accuracy on the training data with the cross-validation accuracy. Their experimental results showed the cross-validation score is better than the simple score. There are some research studies that analyzed the general performance of SVM and proposed to estimate the true error of learned classifiers. In *Generalization Performance of Support Vector Machines and Other Pattern Classifiers* [24], the bound of generalization performance for large margin linear classifier are clearly described. Therefore, the generalization performance of SVM is an interesting alternative for estimating the classification performance in the evolutionary process.

# CHAPTER III

# ADAPTIVE COMBINED KERNEL FUNCTIONS

In this chapter, the adaptive combined kernel functions are proposed for SVM on both classification and regression problems. These kernel functions are the non-negative linear combination of multiple conventional kernels. The weight of combination and the parameters of sub-kernels are the adjustable parameters of these combined kernel functions. These parameters and some parameters of SVM will be investigated by an evolutionary algorithm. Moreover, the objective function in the evolutionary process will be carefully designed.

## 3.1 Combined Kernel Functions

There are many kernel functions that can be used in SVM and other kernel methods. In general, the choice of kernel function has a crucial effect on the classification performance. Different kernel functions make the different results. If the unsuitable kernel function is selected, the results may not achieve the excellent performance. However, many research studies and applications still use the common kernel function, such as linear, polynomial, and radial basis function (RBF) kernels. These common kernel functions may not be sufficient for the complex or large problems. Most of such problems require a more complex separating hyperplane, whose complexity depends on the properties of the used kernel. Hence, if the good characteristics of several common kernel functions are combined in one kernel function, it should make a more complex hyperplane and yield better results on the complex problems.

We also notice that all common kernel functions in the literature are either inner-product-based functions or distance-based functions. The linear kernel function and the polynomial kernel function are samples of the popular inner-product-based kernels, and the RBF kernel is the most successful distance-based kernel. However, this research regards that the linear kernel function is a polynomial kernel function at degree 1. Hence, the polynomial kernel function and the RBF kernel function are two functions that will be considered for creating a new combined kernel. The general forms of polynomial and RBF kernel functions are illustrated as following:

$$K_{Poly}(x, x', d) = \left( \langle x \cdot x' \rangle + c \right)^d \tag{63}$$

$$K_{RBF}(x, x', \gamma) = \exp\left( -\gamma \, \|x - x'\|^2 \right) \tag{64}$$

where $d$ is the degree of the polynomial kernel function and $\gamma$ is the width of the RBF kernel function.

In order to obtain a more flexible kernel function, the combination methods are then considered. When the multiple kernel functions are combined, they should correspond to the Mercer's theorem. Hence, the closure properties of kernel functions are the good criterions for constructing new kernel functions. The fundamental operations for combining the multiple kernel functions in this research are (i) the addition of kernel functions, (ii) the multiplication between a non-negative scalar number and a kernel function, and (iii) the multiplication of kernel functions. When the kernel functions are combined by these operations, the new combined kernels still correspond to the Mercer's theorem. The proving processes of these operations are shown in Corollary 3-1.

**Corollary 3-1**. Let $K_1(x, x')$ and $K_2(x, x')$ be Mercer's kernels, and $a$ be a non-negative real value. Then,

(i) the addition of two Mercer's kernels is a Mercer's kernel,

(ii) the scalar multiplication between a non-negative real value and any Mercer's kernel is a Mercer's kernel, and

(iii) the multiplication of two Mercer's kernels is a Mercer's kernel.

**Proof (i)**. Let

$$\mathrm{K}(x, x') = K_1(x, x') + K_2(x, x') \,. \tag{65}$$

According to the Mercer's theorem, we know that

$$\iint K_1(x, x') \, g(x) \, g(x') \, dx \, dx' \quad \geq \quad 0 \,, \quad \forall g \tag{66}$$

and

$$\iint K_2(x, x') \, g(x) \, g(x') \, dx \, dx' \quad \geq \quad 0 \,, \quad \forall g \,. \tag{67}$$

Therefore,

$$\iint K_1(x, x') \, g(x) \, g(x') \, dx \, dx' + \iint K_2(x, x') \, g(x) \, g(x') \, dx \, dx' \quad \geq \quad 0 \,, \quad \forall g \,, \tag{68}$$

$$\iint \left(K_1(x,x') + K_2(x,x')\right) g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,, \tag{69}$$

$$\iint \mathrm{K}(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,. \tag{70}$$

Hence, $\mathrm{K}(x,x') = K_1(x,x') + K_2(x,x')$ is a Mercer's kernel. □

**Proof (ii)**. Let

$$\mathrm{K}(x,x') = a \cdot K_1(x,x')\,. \tag{71}$$

According to the Mercer's theorem,

$$\iint K_1(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,. \tag{72}$$

Since $a$ is a non-negative real value, therefore

$$a \cdot \iint K_1(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,, \tag{73}$$

$$\iint a \cdot K_1(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,, \tag{74}$$

$$\iint \mathrm{K}(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,. \tag{75}$$

Hence, $\mathrm{K}(x,x') = a \cdot K_1(x,x')$ is a Mercer's kernel. □

**Proof (iii)**. Let

$$\mathrm{K}(x,x') = K_1(x,x')\, K_2(x,x')\,. \tag{76}$$

According to the Mercer's theorem, we know that

$$\iint K_1(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g \tag{77}$$

and

$$\iint K_2(x,x')\, g(x)\, g(x')\, dx\, dx' \quad \geq \quad 0\,, \quad \forall g\,. \tag{78}$$

Moreover, both $K_1(x,x')$ and $K_2(x,x')$ are Mercer's kernel function, then $K_1(x,x')$ can be expanded in terms of $\psi_k$ :

$$K_1(x,x') \;=\; \sum_{k=1}^{\infty} \omega_k\, \psi_k(x)\, \psi_k(x') \tag{79}$$

and $K_2(x,x')$ can be expanded in terms of $\Phi_k$ :

$$K_2(x,x') = \sum_{k=1}^{\infty} \lambda_k \, \Phi_k(x)\, \Phi_k(x') \tag{80}$$

with $\omega_k \ge 0$ and $\lambda_k \ge 0$, respectively.

We need to show that

$$\iint \mathrm{K}(x,x')\, g(x)\, g(x')\, dx\, dx' \;\; \ge \;\; 0, \;\; \forall g, \;\; \text{or} \tag{81}$$

$$\iint K_1(x,x')\, K_2(x,x')\, g(x)\, g(x')\, dx\, dx' \;\; \ge \;\; 0, \;\; \forall g. \tag{82}$$

First, we define $du = g(x)dx$ and $du' = g(x')dx'$. Therefore,

$$\begin{aligned}
\iint \mathrm{K}(x,x')\, g(x)\, g(x')\, dx\, dx' &= \iint \mathrm{K}(u,u')\, du\, du' \\
&= \iint K_1(u,u')\, K_2(u,u')\, du\, du' \\
&= \iint K_1(u,u') \left( \sum_{k=1}^{\infty} \lambda_k \, \Phi_k(u)\, \Phi_k(u') \right) du\, du' \qquad \text{[by (80)]} \\
&= \iint \left( \sum_{k=1}^{\infty} \lambda_k \, K_1(u,u')\, \Phi_k(u)\, \Phi_k(u') \right) du\, du' \\
&= \sum_{k=1}^{\infty} \iint \lambda_k \, K_1(u,u')\, \Phi_k(u)\, \Phi_k(u')\, du\, du' \\
&= \sum_{k=1}^{\infty} \lambda_k \iint K_1(u,u')\, \Phi_k(u)\, \Phi_k(u')\, du\, du'.
\end{aligned} \tag{83}$$

From (77), we have

$$\iint K_1(u,u')\, \Phi_k(u)\, \Phi_k(u')\, du\, du' \;\; \ge \;\; 0, \quad \text{for } k = 1, 2, \ldots, \infty. \tag{84}$$

Moreover, from (80), we know that $\lambda_k \ge 0$, for $k = 1, 2, \ldots, \infty$. Thus

$$\lambda_k \iint K_1(u,u')\, \Phi_k(u)\, \Phi_k(u')\, du\, du' \;\; \ge \;\; 0, \quad \text{for } k = 1, 2, \ldots, \infty. \tag{85}$$

Therefore

$$\sum_{k=1}^{\infty} \lambda_k \iint K_1(u,u')\, \Phi_k(u)\, \Phi_k(u')\, du\, du' \;\; \ge \;\; 0. \tag{86}$$

From (83) and (86), we get

$$\iint \mathrm{K}(x,x')\, g(x)\, g(x')\, dx\, dx' \;\; \ge \;\; 0. \tag{87}$$

Hence, $\mathrm{K}(x,x') = K_1(x,x')\, K_2(x,x')$ is a Mercer's kernel. □

From these fundamental operations, an approach to combine multiple kernels is the non-negative linear combination of multiple kernel functions. By Corollary 3-1, (i) and (ii), the proof of this kernel is rather obvious. With two kinds of kernel functions and this linear combination, there are three new combined kernel functions that can be possible, i.e. (i) the non-negative linear combination of multiple RBF kernels at different scales, (ii) the non-negative linear combination of multiple polynomial kernels at different degree, and (iii) the non-negative linear combination of polynomial kernels and RBF kernels. Furthermore, (iv) the multiplication of polynomial and RBF kernels is an alternative that will also be considered in this research. The proposed combined kernels allow better discrimination in the feature space, and the mathematical forms of these combined kernels are described in this section.

### 3.1.1 Multi-Scale RBF Kernel Function

The Gaussian RBF kernel is widely used in many problems. It uses the Euclidean distance between two points in the original space to find the correlation in the augmented feature space. The points very close to each other are strongly correlated whereas points far apart have uncorrelated image in the augmented space [5]. This correlation is rather smooth. There is only one parameter for adjusting the width of RBF, which is not powerful enough for some complex problems.

In order to get a better kernel, one possible way is to adjust the velocity of decrement in each range of distance between two points. Moreover, the obtained kernel should maintain the good characteristic of the RBF kernel that any close points are strongly correlated. To achieve these behaviors, the non-negative linear combination of multiple RBF kernels at different scales is proposed. When the multiple RBF kernels are combined, this new kernel function is more flexible than its component kernels.

The correlations in feature space (relations between kernel functions and the distance between two points in the original space) of the single RBF kernel and the multi-scale RBF kernels are displayed in Figure 3-1. This figure shows that the correlations of distance between two points in the RBF kernel are rather smooth, while those of 2-RBF (2 terms of RBF sub-kernels are combined) and 3-RBF (3 terms of RBF sub-kernels are combined) have more variable shape. This can be interpreted that the increase in the number of adjustable parameters provides a more adaptive kernel.

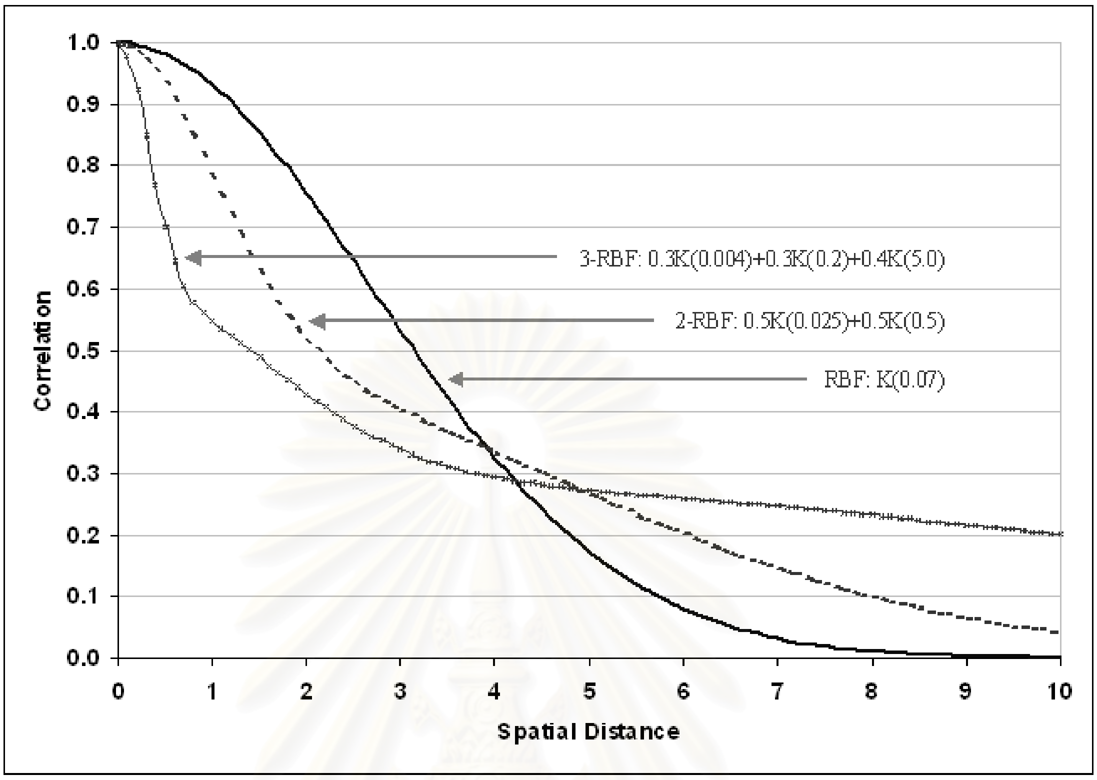


Figure 3-1: Correlations of Distance between Two Points in RBF, 2-RBF, and 3-RBF Kernels

The analytic expression of the multi-scale RBF kernel function is the following:

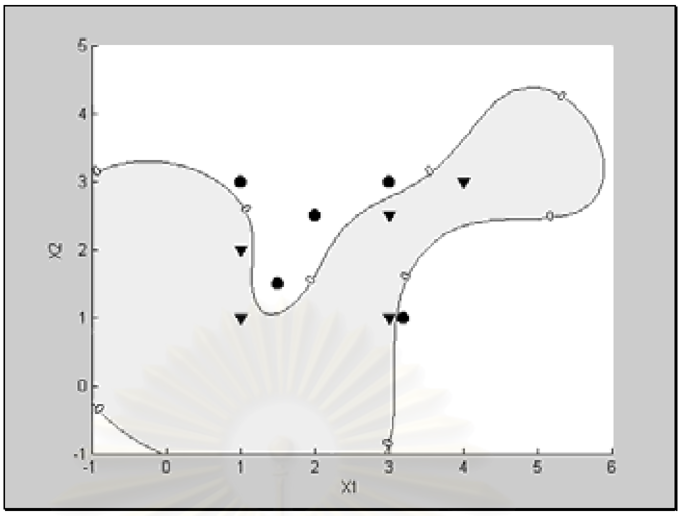
$$K_{n-RBF}(x, x') = \sum_{i=1}^{n} a_i K_{RBF}(x, x', \gamma_i) \quad (88)$$

where $n$ is a positive integer, $a_i \geq 0$ for $i = 1,...,n$ are the arbitrary non-negative weighting constants, and

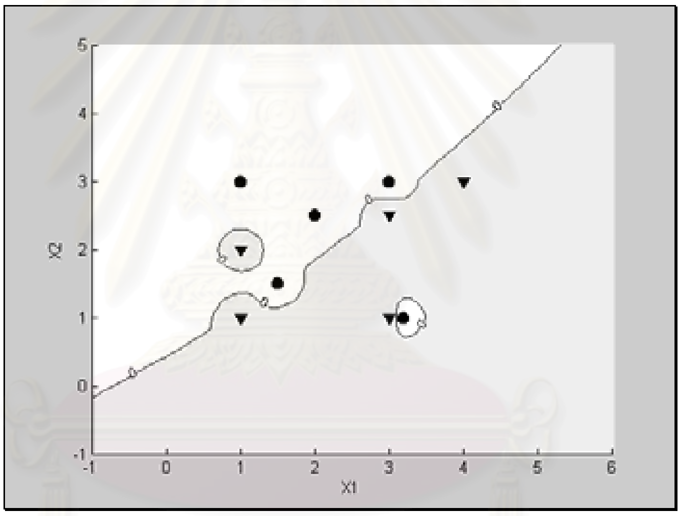
$$K_{RBF}(x, x', \gamma_i) = \exp(-\gamma_i \|x - x'\|^2) \quad (89)$$

is the RBF kernel with the width $\gamma_i$ for $i = 1,...,n$.

When multiple RBF functions are combined, the results of classification are more flexible than using a single RBF function. The examples of classification with a simple RBF kernel and a combination of two RBF kernels are showed in Figure 3-2. In these examples, the training data are non-linearly separable. The SVM with a single RBF and 2-RBF (the multi-scale RBF kernel with $n = 2$) kernels can correctly classify the data. However, the 2-RBF kernel yields the result that is more flexible and easier to comprehend.



(a) SVM with RBF Kernel Function



(b) SVM with 2-RBF Kernel Function

Figure 3-2: Examples of Classification

When $n$ terms of sub-kernels are combined, this new combined kernel function has $2n$ parameters; $n$ parameters for non-negative weights of combination and $n$ parameters for the width of RBF sub-kernel functions. However, we notice that the number of parameters can be reduced to $2n-1$ by fixing a value of the first weight parameter to 1. The multi-scale RBF kernel function becomes as follows,

$$K_{n-RBF}(x,x') = K_{RBF}(x,x',\gamma_0) + \sum_{i=1}^{n-1} a_i K_{RBF}(x,x',\gamma_i)\ . \tag{90}$$

This multi-scale RBF kernel function still corresponds to the Mercer's theorem. This is because the RBF kernel is a well-known Mercer's kernel, and thus the non-negative linear combination of RBF kernels is an admissible kernel function by the Mercer's theorem. Moreover, this combined kernel function is more flexible as it has more adjustable parameters. The performance of the multi-scale RBF kernel function in equation (90) will be evaluated on classification and regression tasks in the next chapter.

### 3.1.2 Multi-Degree Polynomial Kernel Function

Although the non-negative linear combination of multiple polynomial kernels is still a polynomial kernel function, the multi-degree polynomial kernel function has more adjustable parameters. Therefore, this kernel function is more flexible than the conventional polynomial kernel. In Figure 3-3, we show that the value of single polynomial degree 2 is a parabola curve while the values of inner product are varied. When 2 terms of polynomial kernels and 3 terms of polynomial kernels are combined, the curve is changed.

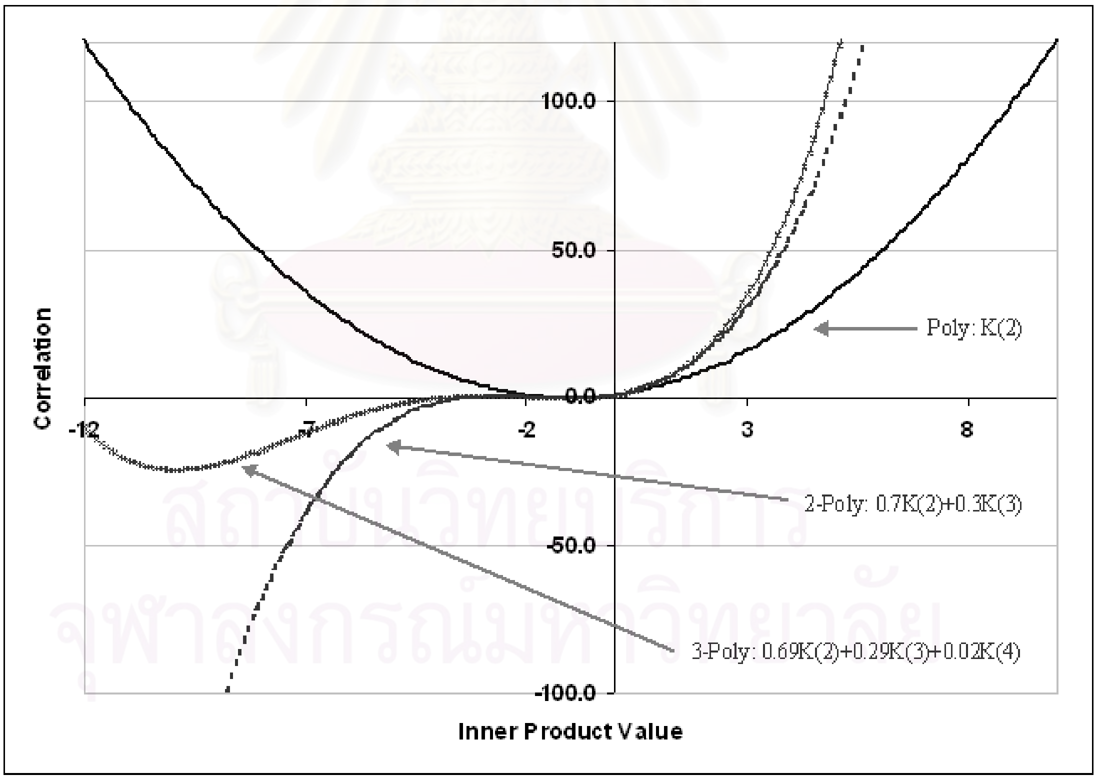


Figure 3-3: Correlations of Inner Product between Two Points in Polynomial, 2-Polynomial, and 3-Polynomial Kernels

The analytic expression of this multi-degree polynomial kernel is

$$K_{n-Poly}(x,x') = \sum_{i=1}^{n} a_i K_{Poly}(x,x',d_i) \tag{91}$$

where $n$ is a positive integer, $a_i \geq 0$ for $i = 1,...,n$ are the arbitrary non-negative weighting constants, and

$$K_{Poly}(x,x',d_i) = \left(\langle x \cdot x' \rangle + 1\right)^{d_i} \tag{92}$$

is the polynomial kernel at the degree $d_i$ for $i = 1,...,n$.

The degree of each polynomial sub-kernel is the adjustable parameter. Therefore, when $n$ polynomial kernels are combined, there are $n$ integer valued parameters for the degree of polynomial kernels and $n$ real-valued parameters for the weight of combination. By the same idea with the multi-scale RBF kernel, the number of parameters of multi-degree polynomial kernel can be reduced from $2n$ parameters to $2n-1$ parameters, and the multi-degree polynomial kernel becomes

$$K_{n-Poly}(x,x') = K_{Poly}(x,x',d_0) + \sum_{i=1}^{n-1} a_i K_{Poly}(x,x',d_i)\,, \tag{93}$$

and this form of multi-degree polynomial kernel will be tested in the next chapter.

### 3.1.3 Linear Combination of Polynomial and RBF Functions

As described earlier, the polynomial kernel is an inner-product-based kernel fucntion, whereas the RBF kernel is a distance-based kernel function. The non-negative linear combination of polynomial and RBF kernels is one way to combine the inner-product-based kernels and the distance-based kernels. We expect that their advantages will be integrated in this combined kernel. Furthermore, both polynomial and RBF kernels are the admissible kernels that correspond to Mercer's theorem. Then, the non-negative linear combination of polynomial and RBF kernels can be proved to satisfy the Mercer's theorem. The different kernel functions with the different parameters are combined with including weights, and the general form of this combined kernel function is

$$K_{(n-Poly)+(m-RBF)}(x,x') = \sum_{i=1}^{n} a_i \; K_{Poly}(x,x',d_i) + \sum_{i=1}^{m} b_i \; K_{RBF}(x,x',\gamma_i)\,, \tag{94}$$

where $n$ and $m$ are the positive integer numbers, $a_i \geq 0$ for $i = 1,2,\ldots,n$, $b_i \geq 0$ for $i = 1,2,\ldots,m$, $K_{Poly}(x,x',d_i)$ is the polynomial kernel at the degree $d_i$, and $K_{RBF}(x,x',\gamma_i)$ is the RBF kernel at the width $\gamma_i$.

From (94), when $n$ terms of polynomial sub-kernels and $m$ terms of RBF sub-kernels are combined, there are $2(n+m)$ parameters ($n$ integer numbers for the degree of polynomial sub-kernels, $m$ real numbers for the width of RBF sub-kernels, and $n+m$ real-valued numbers for adjusting the weights of combination). However, there is a simpler version of this combination, which is the addition between polynomial and RBF kernels. Two different sub-kernel functions are added with including weights, as the following:

$$K_{Poly+RBF}(x,x') = p \cdot K_{Poly}(x,x',d) + q \cdot K_{RBF}(x,x',\gamma)\ , \tag{95}$$

where $p$ and $q$ are any positive real values. Since, $p$ and $q$ are arbitrary constants, we can reduce the number of parameters by the following representation:

$$K_{Poly+RBF}(x,x') = p \cdot K_{Poly}(x,x',d) + (1-p) \cdot K_{RBF}(x,x',\gamma)\ , \tag{96}$$

where $p \in [0,1]$. Equation (96) can be called the convex combination of polynomial and RBF kernels.

### 3.1.4 Multiplication of Polynomial and RBF Functions

The other combining method is the kernel multiplication. However, the multiplication of several RBF kernels does not change the general form of RBF kernel, as the following:

$$\begin{aligned}\prod_{i=1}^{n} K_{RBF}(x,x',\gamma_i) &= \prod_{i=1}^{n} \exp(-\gamma_i \left\|x-x'\right\|^2) \\ &= \exp(-\gamma \left\|x-x'\right\|^2)\ ; \qquad \text{where } \gamma = \sum_{i=1}^{n} \gamma_i\ .\end{aligned} \tag{97}$$

Although the multiplication of several polynomial kernels is also a polynomial kernel, it is more flexible than the single polynomial kernels. Therefore, the general form of multiplication of polynomial and RBF kernels is

$$K_{(n-Poly)\times RBF}(x,x') = \left(\prod_{i=1}^{n} K_{Poly}(x,x',d_i)\right) \cdot K_{RBF}(x,x',\gamma)\ , \tag{98}$$

where $n$ is a positive integer, $K_{Poly}(x,x',d_i)$ is the polynomial kernel at the degree $d_i$, and $K_{RBF}(x,x',\gamma)$ is the RBF kernel at the width $\gamma$. Notice that this combined kernel does not include the weights, because the multiplication between a constant value and the kernel function does not affect to the performance of SVM.

However, the form of multiplication of polynomial and RBF kernels in (98) may be too complicated for applying in real problems. Therefore, we propose to use the simpler form of multiplication of two kernel functions that may be less flexible than (98). This kernel is the multiplication of a polynomial sub-kernel and an RBF sub-kernel, as the following:

$$K_{Poly \times RBF}(x, x') = K_{Poly}(x, x', d) \cdot K_{RBF}(x, x', \gamma) . \tag{99}$$

A degree of polynomial sub-kernel and the weight of RBF sub-kernel are two adjustable parameters of this combined kernel function.

The product of these two kernels is both inner-product-based and distance-based kernel. The polynomial and RBF kernels are the Mercer's kernels, thus the multiplication of polynomial and RBF kernels is also a Mercer's kernel. The proving process of this multiplication of kernel functions was shown in the early part of this chapter. Although there are other combinations of kernels that use both addition and multiplication operators, they will not be suggested in this chapter because they do not have the general form and they may be too complicated for using in the real world problems.

## 3.2 Evolutionary Techniques for Support Vector Machines

When the combined kernel functions are used, there are more adjustable parameters. In most cases, we do not have any prior knowledge about these parameters. Moreover, there are some parameters of SVM that should also be adjusted. The regularization parameter ($C$) is a parameter of SVM's learning that appears on both support vector classification (SVC) and support vector regression (SVR). The deviation of approximation ($\varepsilon$) is also an adjustable parameter in SVR. These parameters and the parameters of kernel functions are called ***hyperparameter***.

In order to obtain appropriate values of these parameters, the evolutionary strategy (ES) is considered. This algorithm can search the optimal values of these parameters by using an objective function (or fitness function) that was pre-defined. Therefore, it is very suitable for our parameter selection problem because we do not have any knowledge but we have only a goal to optimize the performance of SVM on a given task. Although there are several variations of ES, we choose to use the ($\mu + \lambda$)-ES where both $\mu$ parents and $\lambda$ offspring compete equally for survival. With the ($\mu + \lambda$)-ES, the good solution will be selected always, and thus it does not lose during the evolutionary process.

The (5+10)-ES will be used throughout this research. This algorithm uses 5 solutions to produce 10 new solutions, which the population size is not large. Although the evolutionary computing can be implemented by parallel programming, it is more convenient to implement and run on a computer. When the population size of ES is small, such as (1+1)-

ES, the population may lack diversity in each generation and a large number of generations may be required to converge to the optimal solutions. In opposite, when the values of $\mu$ and $\lambda$ are high then the algorithm may need a lot of computation resources for each generation of ES but may require only a few generations to obtain the optimal solutions. The (5+10)-ES is a choice of ($\mu + \lambda$)-ES, which can preserve the diversity of population and does not require a lot of computational resources for each generation. Thus, it is suitable for adjusting the parameters of SVM and the combined kernels. The algorithm of (5+10)-ES is shown in Figure 3-4.

```
$t = 0$;
$initialize\ (\vec{v}_1,..., \vec{v}_5, \vec{\sigma})$;
$evaluate\ fitness(\vec{v}_1),..., fitness(\vec{v}_5)$;
while ( TerminatedConditions <> TRUE ) do
    for $i = 1$ to 10 do
        $\vec{v}'_i = recombine(\vec{v}_1,..., \vec{v}_5)$;
        $\vec{v}'_i = mutate(\vec{v}'_i)$;
        $evaluate\ fitness(\vec{v}'_i)$;
    end
    $(\vec{v}_1,..., \vec{v}_5) = select(\vec{v}_1,..., \vec{v}_5, \vec{v}'_1,..., \vec{v}'_{10})$;
    $\vec{\sigma} = mutate_{\sigma}(\vec{\sigma})$;
    $t = t + 1$;
End
```

Figure 3-4: (5+10)-ES Algorithm

This algorithm starts with 0th generation ($t = 0$) and selects $\mu$ solutions ($\vec{v}_1,..., \vec{v}_\mu$) with standard deviation $\vec{\sigma}$ using randomization or assigning initial values. After that, this algorithm will be iterated to create new better solutions while the terminated conditions are not true. This algorithm uses these selected 5 solutions to produce 10 new solutions by a recombination method. These new solutions are mutated and evaluated, and only the 5 fittest solutions are selected from 5+10 solutions to be the parents in the next generation. These processes will be repeated until a fixed number of generations have been produced and evaluated or earlier if the acceptance criterion is reached.

Normally, this algorithm will be terminated if and only if $t$ exceeds a predefined maximum generation or the optimal solution is found. In this research, we use the maximum number of generations as the stopping criterion of this (5+10)-ES algorithm. In our experiments, the maximum number of generations is fixed as 1000. Although the high quality solutions can be found with fewer generations for some datasets, we want to ensure that the high quality solutions can be found for all datasets in our experiments. A large number of generations does not decrease the classification performance. However, the maximum number of generations will be restricted by the running time allowed to run our (5+10)-ES algorithm. The details of each step in our (5+10)-ES algorithm are described in the following.

### 3.2.1 Initialization

Let $\vec{v}$ be the non-negative real-valued vector of all parameters. The vector $\vec{v}$ depends on the used kernel function and the problem under consideration. In the classification problems, the regularization parameter of SVM and the parameters of kernel function are adjusted by the ES algorithm. Therefore, for the multi-scale RBF kernel function, the vector $\vec{v}$ has $2n$ dimensions and it is represented in the form:

$$\vec{v} = (C,\ \gamma_0,\ a_1,\ \gamma_1,\ a_2,\ \gamma_2,\ \ldots,\ a_{n-1},\ \gamma_{n-1}), \tag{100}$$

where $C$ is the regularization parameter, $\gamma_i$ for $i = 0,\ldots,n-1$ are the widths of RBFs, $a_i$ for $i = 1,\ldots,n-1$ are the weights of RBFs, and $n$ is the number of terms of RBF sub-kernel functions.

For the multi-degree polynomial kernel function, the parameter vector $\vec{v}$ also has $2n$ dimensions and it is represented in the form:

$$\vec{v} = (C,\ d_0,\ a_1,\ d_1,\ a_2,\ d_2,\ \ldots,\ a_{n-1},\ d_{n-1}), \tag{101}$$

where $C$ is the regularization parameter, $d_i$ for $i = 0,\ldots,n-1$ are the integer numbers that representimg the degree of polynomial sub-kernels, $a_i$ for $i = 1,\ldots,n-1$ are the weights of polynomial sub-kernels, and $n$ is the number of terms of polynomial sub-kernel functions. As the degrees of polynomial sub-kernels are the integer, we may fix these degrees of polynomial at 1, 2, … , $n$ and only the weights of this combined kernel and the regularization parameter of SVM are searched. In that case, the vector $\vec{v}$ will have only $n$ dimensions and it is represented by

$$\vec{v} = (C,\ a_1,\ a_2,\ \ldots,\ a_{n-1}). \tag{102}$$

For the non-negative linear combination of polynomial and RBF kernel functions, the vector $\vec{v}$ has $2(n+m)+1$ dimensions, when $n$ terms of polynomial sub-kernels and $m$ terms of RBF sub-kernels are combined. The vector $\vec{v}$ is represented in the form:

$$\vec{v} = (C\,,\ a_1,\ d_1,\ a_2\,,\ d_2,\ \dots\,,\ a_n\,,\ d_n\,,\ b_1\,,\ \gamma_1,\ b_2\,,\ \gamma_2\,,\ \dots\,,\ b_m\,,\gamma_m), \tag{103}$$

where $C$ is the regularization parameter, $a_i$ for $i=1,\dots,n$ are the weights of polynomial sub-kernels, $d_i$ for $i=1,\dots,n$ are the integer numbers representing the degrees of polynomial sub-kernels, $b_i$ for $i=1,\dots,m$ are the weights of RBF sub-kernels, and $\gamma_i$ for $i=1,\dots,m$ are the widths of RBF sub-kernels. The dimensions of $\vec{v}$ can be reduced when the other forms of combined kernel functions are used.

For multiplication of polynomial and RBF kernels, the multiple kernel functions are combined without weights. Therefore, the vector $\vec{v}$ can be represented by

$$\vec{v} = (C\,,\ d_1,\ d_2,\ \dots\,,\ d_n,\ \gamma), \tag{104}$$

where $C$ is the regularization parameter, $d_i$ for $i=1,\dots,n$ are the degrees of polynomial sub-kernels, and $\gamma$ is the width of the RBF sub-kernel.

In the regression problems, the deviation of an approximation $\varepsilon$ is another adjustable parameter of SVM regression. Therefore, the vector $\vec{v}$ of the multi-scale RBF kernel function has $2n+1$ dimensions when the number of terms of RBF sub-kernels is $n$. The vector $\vec{v}$ is represented in the form:

$$\vec{v} = (C\,,\ \varepsilon\,,\ \gamma_0\,,\ a_1,\ \gamma_1,\ a_2\,,\ \gamma_2\,,\ \dots\,,\ a_{n-1},\ \gamma_{n-1}). \tag{105}$$

For the multi-degree polynomial,

$$\vec{v} = (C\,,\ \varepsilon\,,\ d_0\,,\ a_1,\ d_1,\ a_2\,,\ d_2\,,\ \dots\,,\ a_{n-1},\ d_{n-1}). \tag{106}$$

For the non-negative linear combination of polynomial and RBF kernels,

$$\vec{v} = (C\,,\varepsilon\,,\ a_1,\ d_1,\ a_2\,,\ d_2,\ \dots\,,\ a_n\,,\ d_n,\ b_1,\ \gamma_1,\ b_2,\ \gamma_2\,,\ \dots\,,\ b_m\,,\gamma_m). \tag{107}$$

Also, for the multiplication of polynomial and RBF kernels,

$$\vec{v} = (C\,,\ \varepsilon\,,\ d_1,\ d_2\,,\ \dots\,,\ d_n,\ \gamma). \tag{108}$$

For the vector of standard deviation $\vec{\sigma}$, it is a real-valued vector, whose dimension is equal to the dimension of the parameter vector $\vec{v}$. Both the parameter vectors ($\vec{v}_1,\dots,\vec{v}_5$) and the standard deviation vector ($\vec{\sigma}$) are initialized by using randomization. Then, these 5 initial solutions are evaluated to calculate their fitness. Our goal is to find $\bar{v}$ which optimizes the objective function $fitness(\bar{v})$ that must be carefully designed.

### 3.2.2 Selection

From the literature reviews in Chapter 2, we know that there are two main scenarios for selection, i.e. mating selection and overproduction selection as shown in Figure 2-20. However, the selection scenario that is used in this research is different. Both of the traditional scenarios are combined in our evolutionary process. The outline of this procedure is shown in Figure 3-5.

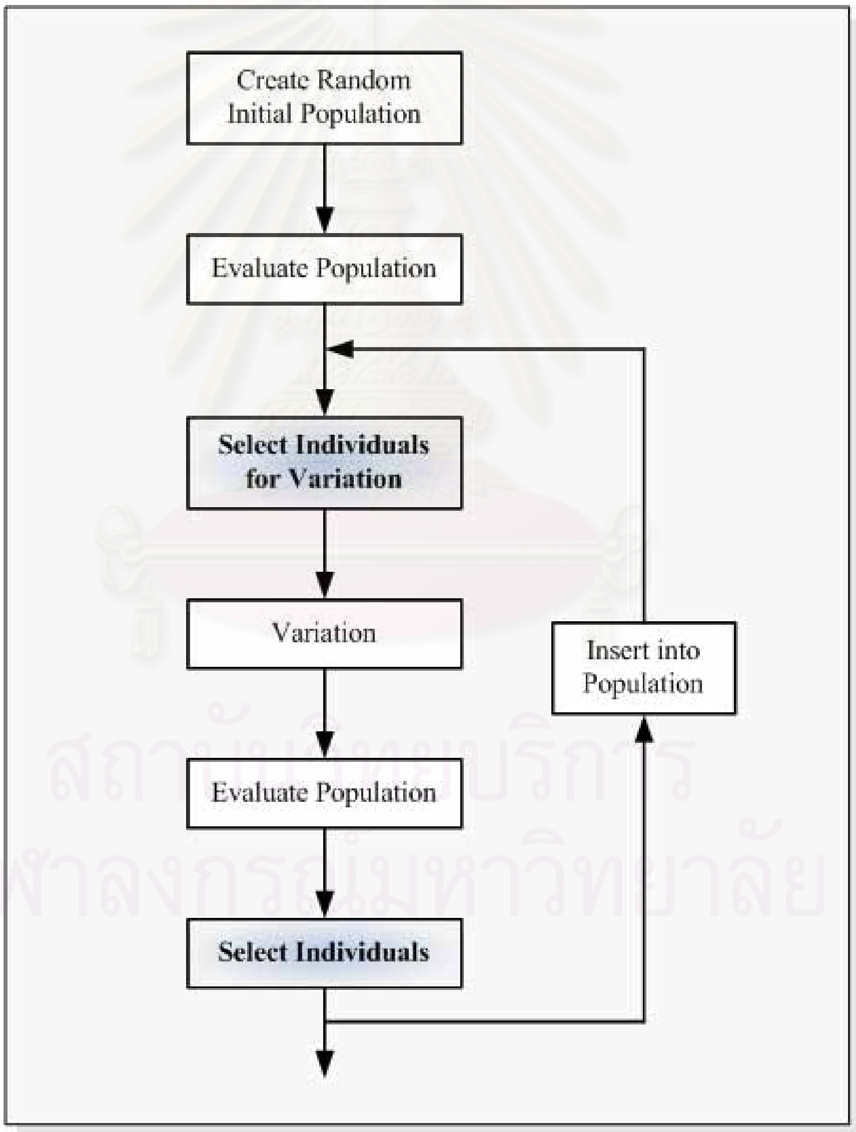


Figure 3-5: Selection Scenario

This procedure starts with a population of individuals with known fitness. Then, selection of individuals is performed base on their fitness. These individuals are combined and mutated to generate new offspring individuals. After fitness evaluation, all individuals, both offspring and parents, are selected to the size of parent population. Thus, the selection is performed in two steps, which are (1) selection of parent individuals for creating the new individuals and (2) selection of the individuals to be parent in the next generation. The different selection methods can be used for each step of selection.

- **Selection of Individuals for Variations**: In this research, the ranking selection is used to choose the individuals for variations. This method is based on the fitness order of each individual in the parent population. The selection probability is assigned to the individuals as a function of their ranks.

For (5+10)-ES, in each generation, the 5 fittest solutions are assigned the probabilities of selection to create new solutions. These fittest solutions are ordered by their objective functions, i.e. $\vec{v}_i$ is more fit than $\vec{v}_{i+1}$. Then, their probabilities are assigned by

$$P(\vec{v}_i) = \frac{\mu-(i-1)}{\sum_{j=1}^{\mu} j} = \frac{\mu-(i-1)}{\mu(\mu+1)/2} = \frac{2}{\mu}\left(1-\frac{i}{\mu+1}\right), \tag{109}$$

for $i = 1, 2, \ldots, \mu$, when $\mu$ is the number of fittest solutions. In this case, $\mu$ is equal to 5.

After that, any individual will be selected by this probability. The new offspring individuals will be generated from these selected individuals by the recombination and the mutation operators.

- **Selection of Individual to be the Parent Population**: The truncation selection method is used in (5+10)-ES to choose the fittest individuals to be the parent in the next generation. Five parents are used to create ten new offspring; the 5 best solutions from both offspring and parents are used as the parents for the next generation. This is an elitist technique that preserves the best individual. Therefore, the good solutions will be alive until better solutions are found.

### 3.2.3 Recombination

Then, the new individuals are generated by the recombination operator. Two individuals are randomly selected from the conventional 5 individuals with their probabilities that are assigned by the ranking method. Then, the average of this pair of individuals, element by element, is a new individual.

$$\vec{v}_k' = \frac{1}{2}\left(\vec{v}_i + \vec{v}_j\right) \tag{110}$$

This method is called the global intermediary recombination method, and it will be used to create 10 new individuals.

This recombination method is chosen for this research because every individual has the chance to be selected for creating the new individuals. Most of offspring individuals should be different from their parents because each of them is the average of two parents, except two selected parents are the same. Therefore, the diversity of population can also be preserved by this recombination method.

### 3.2.4 Mutation

In the case that two selected parents are the same, the mutation operator can vary some parts of individuals. Although the offspring individuals will be different from their parents, these offspring individuals may be the same as the other individuals in the previous generations or in the same generation. The mutation will make these individuals different. Each component of these offspring individuals will be added by a random number. Therefore, the new solutions can be produced, unlimitedly.

When $\vec{v}_i'$ is the parameter vector that has $p$ dimensions ($\vec{v}_i' \in \Re^p$), the $\vec{v}_i'$ for $i = 1,\ldots,10$ are mutated by adding to each of them, $\vec{z}$ where $\vec{z} \in \Re^p$, $\vec{z} = (z_1, z_2, \ldots, z_p)$, and $z_i$ is a random value from a normal distribution with zero mean and $\sigma_i^2$ variation. For example, when the multi-scale RBF kernel functions are used in the classification problems, the parameter vector $\vec{v}_i'$ has $2n$ dimensions ($p = 2n$) where $n$ is the number of terms of RBF sub-kernels. The mutation function of $\vec{v}_i'$ is

$$\begin{gathered} mutate(\vec{v}_i') = \vec{v}_i' + \vec{z} = (C + z_1, \gamma_0 + z_2, \ldots, a_{n-1} + z_{2n-1}, \gamma_{n-1} + z_{2n}) \\ z_i \sim N_i(0, \sigma_i^{\ 2}), \quad \forall i \in \{1,\ldots,2n\}. \end{gathered} \tag{111}$$

Moreover, in each generation, the standard deviation vector ($\vec{\sigma}$) is mutated by the following:

$$\begin{gathered} mutate_\sigma(\vec{\sigma}) = (\sigma_1 \cdot e^{z_1}, \sigma_2 \cdot e^{z_2}, \ldots, \sigma_{2n} \cdot e^{z_{2n}}) \\ z_i \sim N_i(0, \tau^2), \quad \forall i \in \{1,\ldots,2n\}, \end{gathered} \tag{112}$$

where $\tau$ is an arbitrary constant.

## 3.3 Objective Functions in Evolutionary Processes

One of the most important and difficult parts of the evolutionary algorithm is how to define the objective function for the task under consideration. In our case of evaluating the parameters of kernel functions and SVM, there are many ways to define the objective function. In general classification problems, training error can be used as the objective function in the evolutionary processes. For regression problems, percentage error, sum square error, and mean square error are usually used to measure the performance of a regression model.

Because the combined kernel functions are more flexible, these measurement functions may overfit training data. Sometimes, data contain a lot of noise, and thus if the model fits these noisy data, the learned concept may be wrong. In this research, we propose to compare four possible objective functions: the training error, the subsets cross-validation, the bound of generalization error, and the stability of SVM. These objective functions are considered and tested in this research. The suitable objective functions will be applied for some problems to compare with the other methods.

### 3.3.1 Training Error

Training error is a basic function that can be used for evaluating the parameters of SVM classifiers. This function indicates the performance of learning machines measured by the error of classification on training data. This is the simplest way to define our objective function in the evolutionary algorithm. The individuals with low training error should have the high fitness score. The formula expression of training error is shown in the following equation:

$$TrnErr \;=\; \frac{1}{2m}\sum_{i=1}^{m}\left|y_i - f(x_i)\right|, \tag{113}$$

where $x_i \in \Re^N$ is a training data, $y_i \in \{-1,1\}$ is its label or the actual class of $x_i$, and $f(x_i)$ is a decision function of data $x_i$ for $i = 1,\ldots,m$.

For the regression problems, the error on training data can be computed by mean percentage error (MPE), sum of squared error (SSE), or mean squared error (MSE). However, the symmetric mean absolute percentage error (SMAPE) is considered in this research, it is a statistical measurement that attempts to solve the outlier problems.

$$SMAPE \;=\; \frac{1}{m}\sum_{i=1}^{m}\left|\frac{y_i - \hat{y}_i}{(y_i + \hat{y}_i)/2}\right|\times 100, \tag{114}$$

where $y_i$ for $i=1,...,m$ are the actually targets of the training data, $\hat{y}_i$ for $i=1,...,m$ are the forecast values, and $m$ is the number of training data. This measurement function will be used as an objective function in the evolutionary process; the experimental results will be compared to the other objective functions. A set of suitable parameters should yield a lower error on training. However, these objective functions may pick the models that overfit to training data.

### 3.3.2 Subsets Cross-Validation

Although the training error or the training accuracy can be easily calculated, this objective function may overfit the training data. Hence, we propose to train the decision function with several sets of data. A good set of parameters should perform well on many training sets. However, as we have only a fixed amount of training data, subsets cross-validation is considered. For SVM learning, the running time of *k*-subsets cross-validation is about *k* times of constructing an SVM classifier because the SVM classifier must be trained and tested *k* times.

In this research, subsets cross-validation will be tested by using 5-subsets cross-validation. It is a rather good estimate of the generalization error for adjusting the parameters. At the beginning, the training data are divided into five subsets, each of which has almost the same amount of data. For each generation of ES, the five classifiers with the same set of parameters but with different training and testing set are evaluated. In the $j$ th iteration ( $j$ =1, …, 5), the classifier is trained on all subsets except for the $j$ th one. Then, its classification error is evaluated on the $j$ th subset. These partitions are displayed in Figure 3-6.

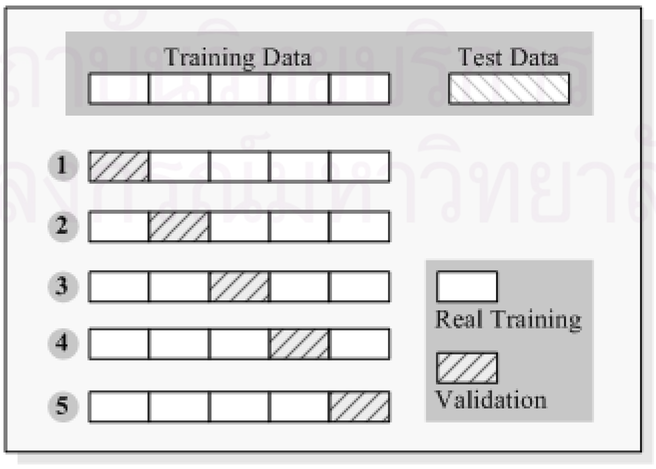


Figure 3-6: 5-Subsets Cross-Validation

Only the real training data sets are used to produce the classifiers with the same set of parameters. Then, the validation sets are used for evaluating the error of the classifiers. The error on validation subset of $j$ th subset can be calculated by

$$Err_j = \frac{1}{2m_j}\sum_{i=1}^{m_j}\left|y_i - f(x_i)\right| , \qquad (115)$$

where $y_i \in \{-1, 1\}$ is the label or the actual class of $x_i$ for $i = 1,\ldots,m_j$ that are in the $j$ th validation subset, $f(x_i)$ is a decision function of data $x_i$, and $j = 1,\ldots,5$. The weighted average of these five errors is used as the objective function.

$$5SubsetOnTrnErr = \frac{\sum_{j=1}^{5} m_j \cdot Err_j}{\sum_{j=1}^{5} m_j} . \qquad (116)$$

Moreover, the concept of subsets cross-validation can be applied to the other measurement function such as the subsets cross-validation on SMAPE for regression problems.

### 3.3.3 Bound of Generalization Error

Actually, we would like to know the generalization performance of an algorithm. The generalization performance of a machine learning algorithm is a function that indicates the capacity of the machine to classify data. However, this function cannot be computed based on the limited training data. Hence, the generalization performance of the learning model was estimated by its bound. The bound of generalization performance of SVM has been presented in a paper of Bartlett and Shawe-Taylor [24].

This bound of generalization error relates to the number of examples, the training error, and the complexity of the hypothesis space. The measure for the complexity of the hypothesis space is the Vapnik-Chervonenkis (VC) dimension [24]. The VC-dimension measures the complexity of the hypothesis space, not by the number of distinct hypotheses, but instead by the number of distinct instances that can be completely discriminated using a hypothesis. The bound on generalization error of SVM is shown in the Proposition 3-1.

**Proposition 3-1** (Bound of generalization error):
For SVM, the loss with probability at least $1-\delta$ over $m$ independently generated examples is bounded by the following:

$$R \leq R_{emp} + \sqrt{\frac{c}{m}\left(h \log^2 m + \log(1/\delta)\right)}, \tag{117}$$

where $R$ is the generalization error, $R_{emp}$ is the empirical error, $c$ is a constant, and $h$ is a non-negative real number called the Vapnik-Chervonenkis (VC) dimension.

As the bound in (117) considers both training error and the VC-dimension, we expect that the generalization performance of SVM with the combined kernel functions can be approximated by this bound. Therefore, this bound of generalization error is considered to be an objective function in our ES algorithm. We presume that a set of suitable parameters should provide a lower bound of generalization error.

### 3.3.4 Stability of SVM

The generalization error is estimated by its bound under various assumptions. The stability is an assumption that can be applied to derive the bound of generalization error. It is a property of algorithms which describes how errors in the input data propagate through the algorithm. The concept of stability was proposed by Bousquet and Elisseeff [61]. They defined the notions of stability for learning algorithms and showed how to use the notions to derive generalization error bounds [61]. Their methods can be applied in the regression framework as well as in the classification one [61]. Hence, the stability for a learning algorithm is considered to be the objective function in evolutionary process. In this work, the stability of soft margin SVM classification and the stability of bounded SVM regression are applied in order to avoid the overfitting problem in evolutionary process.

**Proposition 3-2** (Stability of soft margin SVM classification):
Let $(x_1, y_1)$, $(x_2, y_2), \dots, (x_m, y_m)$ be the training data where $x_i \in R^N$ is a sample data and $y_i \in \{-1, 1\}$ is its label. Assume $K(\cdot)$ is a bounded kernel, that is $K(x_i, x_j) \leq \kappa^2$. The bound with probability at least $1-\delta$ over the sample of size $m$ is

$$R \leq R_{emp} + \frac{\kappa^2}{\lambda m} + \left(1 + \frac{2\kappa^2}{\lambda}\right)\sqrt{\frac{\ln(1/\delta)}{2m}}\ , \tag{118}$$

where $R$ is the risk or generalization error, $R_{emp}$ is called the empirical error, and $\lambda$ is the regularization parameter of SVM ( $\lambda = 1/C$ ).

**Proposition 3-3** (Stability of bounded SVM regression):

Assume $K(\cdot)$ is a bounded kernel, that is $K(x_i, x_j) \le \kappa^2$ and $y_i \in [0, B]$. The bound with probability at least $1-\delta$ over the random draw of the sample of size $m$ is

$$R \le R_{emp} + \frac{\kappa^2}{\lambda m} + \left( \frac{2\kappa^2}{\lambda} + \kappa\sqrt{\frac{B}{\lambda}} \right) \sqrt{\frac{\ln(1/\delta)}{2m}} \, , \tag{119}$$

where $R$ is the generalization error, $R_{emp}$ is the empirical error, and $\lambda$ is the regularization parameter.

The expressions in the right-hand side of (118) and (119) are used as the objective function to evaluate parameters of combined kernel functions and SVMs in the classification and the regression problems, respectively. It is a tight bound, and can be a good criterion for evaluating the parameters in the evolutionary process. The bound of kernel function ( $\kappa^2$ ) can be estimated when the parameters of the combined kernel functions are assigned for each individual parameter vector ( $\bar{v}$ ). We presume that a set of suitable parameters should provide a lower bound of risk. This objective function is tested in the next chapter.

# CHAPTER IV

# EXPERIMENTAL SETTING AND RESULTS

In this chapter, the proposed methods are evaluated by numerical experiments. Both classification and regression benchmarks are considered. The experimental setting and the performance evaluation are described. Then, the experimental results on benchmark datasets are illustrated and discussed in this chapter.

## 4.1 Experimental Setting

In order to verify the performance of the proposed methods, SVMs with the proposed methods are trained and tested on 12 binary classification datasets and 4 regression datasets from the UCI Machine Learning Repository [7]. These datasets come from various real world applications such as game playing, medical inference, predictions in biology and physics, image processing, and character recognition. The number of attributes, the sample size, and the number of classes of each dataset are shown in Table 4-1 and Table 4-2.

Table 4-1: Classification Datasets

| No. | Datasets | Number of Attributes | Number of Data |
|---|---|---|---|
| 1 | Australian | 14 | 690 |
| 2 | Flare | 10 | 1066 |
| 3 | German | 24 | 1000 |
| 4 | Glass2 | 9 | 163 |
| 5 | Heart | 13 | 270 |
| 6 | Ionosphere | 34 | 351 |
| 7 | Liver-Disorder | 6 | 345 |
| 8 | Pima-Indians-Diabetes | 8 | 768 |
| 9 | Sonar | 60 | 208 |
| 10 | ThreeOf9 | 9 | 512 |
| 11 | Tic-Tac-Toe | 9 | 958 |
| 12 | Tokyo | 44 | 959 |

Table 4-2: Regression Datasets

| No. | Datasets | Number of Attributes | Number of Data |
|---|---|---|---|
| 1 | Auto_MPG | 7 | 392 |
| 2 | CPU_Performance | 6 | 209 |
| 3 | Housing | 13 | 506 |
| 4 | Servo | 4 | 167 |

These datasets are normalized by ***Min-Max Normalization***. This normalization is performed in order to reduce the bias on some attributes. In each dataset, the attributes are transformed into the range [0,1]. The new value $v_i'$ of attributes $A$ can be calculated by

$$v_i' = \frac{v_i - \min_A}{\max_A - \min_A}, \tag{120}$$

where $v_i$ for $i = 1, \ldots, m$ are the conventional value of attribute $A$, $\min_A$ and $\max_A$ are the minimum and maximum values of attribute $A$.

The evolutionary strategies are used to find the optimal parameters of SVM and kernel functions. The (5+10)-ESs are used for searching the parameters of SVM and the proposed kernels. The value of $\tau$ in evaluation process of these experiments is 1.0. The widths of RBFs $(\gamma_i)$, the weights of RBFs $(a_i)$, the regularization parameter $(C)$, and the deviation of an approximation $(\varepsilon)$ are real numbers between 0.0 and 10.0. The degree of polynomial $(d_i)$ is a positive integer where $1 \le d_i \le 10$. These parameters are inspected within 1000 generations of ES.

## 4.2 Performance Evaluation

In this research, each dataset is evaluated by 5-folds cross-validation. At the beginning, the training data are divided into five portions, each of which have almost the same number of data. These portions are trained and validated five times. In the $j$ th iteration ( $j = 1, 2, 3, 4, 5$ ), the SVM with a proposed kernel function is trained on all parts except for the $j$ th one. Then, the performance of classification or prediction is calculated for the $j$ th portion. The average of these five performances is reported. The partitions of 5-folds cross-validation are displayed in Figure 4-1.

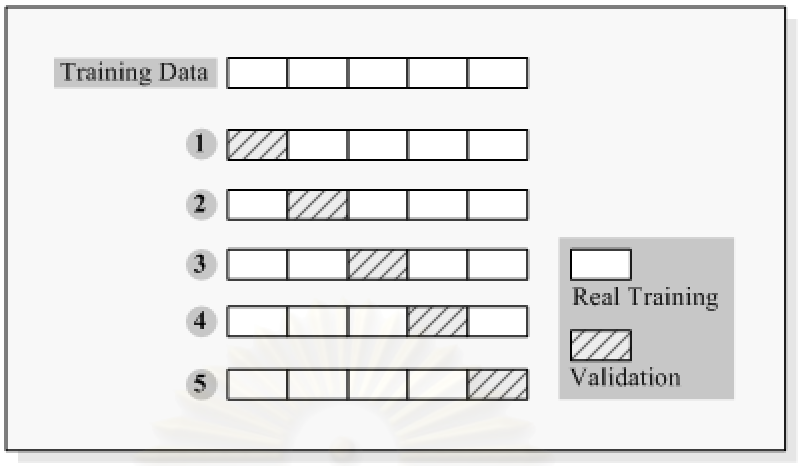


Figure 4-1: 5-Folds Cross-Validation

On classification problems, the performance of learning algorithms is measured by the error of classification. The average of percentage error on 5-folds cross-validation is used to indicate the performance of the proposed methods. The formula expression of the percentage error of each fold is shown in the following equation:

$$PE_j = \frac{\sum_{i=1}^{m_j} |f(x_i) - y_i|}{2m_j} \times 100 , \tag{121}$$

where $m_j$ is the number of validation data in the $j$ th fold, $x_i \in \Re^N$ is the validation data in the $j$ th fold, $f(x_i)$ is a decision function of data $x_i$, and $y_i \in \{-1, 1\}$ is the actual class of data $x_i$ for $i = 1, 2, ..., m_j$. The average of percentage error on 5-folds cross-validation, which is used to compare the performance of learning algorithms, can be calculated by

$$AvgPE = \frac{1}{5}\sum_{j=1}^{5} PE_j , \tag{122}$$

where $PE_j$ is the percentage error of $j$ th fold for $j = 1,...,5$.

For regression problems, the average of symmetric mean absolute percentage error (SMAPE) on 5-folds cross-validation is used for evaluating the performance of the proposed method. The SMAPE of $j$ th fold is defined as

$$SMAPE_j = \frac{1}{m_j}\sum_{i=1}^{m_j} \left| \frac{y_i - \hat{y}_i}{(y_i + \hat{y}_i)/2} \right| \times 100 , \tag{123}$$

where $y_i$ for $i = 1,..., m_j$ are the actually targets of the data, $\hat{y}_i$ for $i = 1,..., m_j$ are the forecast values, and $m_j$ is the number of validation data in the $j^{\text{th}}$ fold, and the average of SMAPE on 5-folds cross-validation can be computed by

$$AvgSMAPE \quad = \quad \frac{1}{5}\sum_{j=1}^{5} SMAPE_j \ . \tag{124}$$

After that, the statistical tests are also used for evaluating the performance of the proposed methods. The paired $T$-test and the Friedman test [62, 63] will be used to compare the experimental results. The paired T-test is used for testing the difference of two learning algorithms on the average accuracies of each dataset, whereas the Friedman test is used for testing the difference of multiple algorithms over multiple datasets based on their average ranks.

***The statistical paired $T$-test*** is applied for testing the statistical significant difference between the performances of two learning algorithms on each dataset. The term ***paired*** means that there is a correspondence between observations from each population, i.e. there is a one-to-one correspondence between the values in the two groups. In practice, the paired $T$-test is commonly used to compare how a group of subjects perform in two different test conditions. The paired $T$-test provides a hypothesis test of the difference between population means for a pair of random samples whose differences are approximately normally distributed. We note that a pair of samples, each of which are not from a distribution, often yields differences that are normally distributed.

Give two paired sets of $a_i$ and $b_i$ of $n$ measured values, the paired $T$-test determines whether they differ from each other in a significant way under the assumptions that the paired differences are independent and identically normally distributed. To apply $T$-test, let $d_i$ be the difference of each pair of measured values,

$$d_i = a_i - b_i \, , \tag{125}$$

where $i = 1, 2, \ldots, n$. Therefore, $\overline{D}$ is the mean difference,

$$\overline{D} = \frac{\sum_{i=1}^{n} d_i}{n} \, , \tag{126}$$

$S_d^2$ is the sample variance difference,

$$S_d^2 = \frac{\sum_{i=1}^{n} (d_i - \overline{D})^2}{n-1} \, . \tag{127}$$

The null hypothesis of no difference between the means of two groups of observations will be tested by the statistical $T$. For the alternative hypothesis, we can choose one of three alternative hypotheses. The default is that the difference between the means is not equal to the specified difference, which is the two-sided alternative. The one-sided alternatives are that the difference is greater than, or less than, the difference specified in the null hypothesis.

If $\mu_1$ and $\mu_2$ are the means of 1st group and 2nd group, respectively, the hypotheses are defined by

$$H_0 : \mu_1 - \mu_2 = 0 \tag{128}$$

and

$$H_1 : \mu_1 - \mu_2 \neq 0, \quad \text{or}$$
$$H_1 : \mu_1 - \mu_2 > 0, \quad \text{or} \tag{129}$$
$$H_1 : \mu_1 - \mu_2 < 0 .$$

Then, the test statistic $T$ is calculated as:

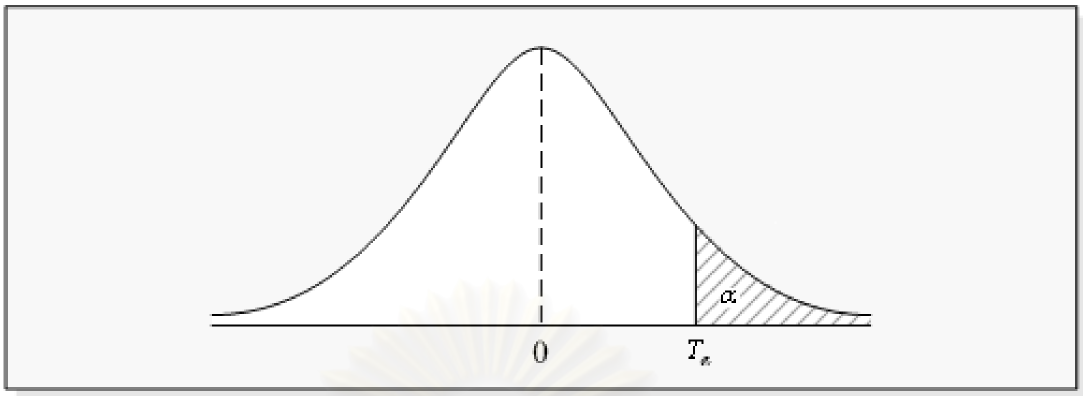
$$T = \frac{\overline{D}}{S_d / \sqrt{n}}, \tag{130}$$

with the degree of freedom is $n-1$.

The table of $T$-distribution confidence intervals can be used to determine the significance level $\alpha$ that two groups differ. The value of statistical $T_\alpha$ from this table is called the critical value. If the statistical $T$ is in the critical region or the rejection region, then the null hypothesis will be rejected while the alternative hypothesis will be accepted. This means that the means of two groups are significantly different at the probability of $(1-\alpha)\times 100\%$. In the other hand, if the statistical $T$ is in the acceptance region, the null hypothesis will be accepted because no reason to reject this null hypothesis.

In this research, the different error of two learning algorithms will be evaluated by the paired $T$-test. We would like to illustrate that the performances of the proposed methods are significantly greater than those of baseline algorithms. On each dataset, the error is measured from each fold of learning. Our experiments use 5-folds cross-validation then the degree of freedom is $5-1=4$. The critical values of $T$-distribution with the degree of freedom 4 are illustrated in Table 4-3.

Table 4-3: Critical Values of T-distribution



| Degree of Freedom | $\alpha$ | | | | |
|---|---|---|---|---|---|
| | 0.1 | 0.05 | 0.01 | 0.005 | 0.001 |
| 4 | 1.533 | 2.132 | 3.747 | 4.604 | 7.173 |

Although the paired $T$-test can be used for testing the difference between two learning algorithm over multiple datasets (in that case, the sample size may refer to the number of used datasets), it is inconvenient to test multiple algorithms. A common example of the test procedure would be comparing multiple algorithms by conducting all paired $T$-tests and reporting results like "algorithm A was found significantly better than B and C, and algorithms A and E were significantly better than D, while there were significantly differences between other pairs" [64]. When so many tests are made, a certain proportion of the null hypotheses is rejected due to random chance, so listing them make little sense [64].

Moreover, the averaging over multiple datasets may be susceptible to outliers. The test's power is decreased by increasing the estimated standard deviation. In this research, the average error across the datasets is a measurement to describe the performance of our methods. However, this average error may be meaningless if the results on different datasets are not comparable [64]. In general, we prefer the classifiers or the algorithms that work well on many problems. Therefore, a ranking method is applied to compare the learning algorithms in this research. The algorithms are ranked for each dataset separately, i.e. the best performing algorithm gets the rank of 1, the second best gets the rank 2, and so on. In case of ties, an average rank is assigned to both algorithms or all tie algorithms. Then, the average ranks across the datasets are compared. These average ranks provide a fair comparison of our learning algorithms.

Then, ***the Friedman test*** is also considered for statistical test on the average ranks. The Friedman test [62, 63] is a statistical method for test the differences between more than two related sample means, and thus it can use to compare multiple classifiers of this research. The average ranks of algorithms are compared under the null-hypothesis, which states that all the algorithms are equivalent and so their ranks should be equal [64].

Let $r_{ij}$ be the rank of the the $j$-th of $k$ algorithms on the $i$-th of $N$ datasets. The Friedman test compares the average ranks of algorithms,

$$R_j = \frac{1}{N}\sum_{i=1}^{N} r_i \ . \tag{131}$$

Under the null hypothesis, which states that all the algorithms are equivalent and so their ranks $R_j$ should be equal [64]. The Friedman statistic

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{132}$$

is distributed according to $\chi_F^2$ with $k-1$ degrees of freedom, when $N$ and $k$ are big enough. Iman and Davenport [65] derived a better statistic from Friedman's $\chi_F^2$

$$F_F = \frac{(N-1)\,\chi_F^2}{N(k-1)-\chi_F^2}, \tag{133}$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom. The table of critical values can be found in any statistical book.

If the null-hypothesis is rejected, we can proceed with a post-hoc test. The Bonferroni-Dunn test [66] is used for pairwise comparisons. The performances of two algorithms are significantly different if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{134}$$

where critical values $q_\alpha$ are illustrated in Table 4-4. The performance of algorithm $A$ is significantly better than algorithm $B$ if the difference between the average ranks of algorithms $A$ and $B$ is more than the critical difference ($R_A - R_B > CD$). Sometimes the Friedman test reports a significant difference but post-hoc test fails to detect it [64]. This is due to the lower power of the latter. The experimental results are shown and tested in the next section.

Table 4-4: Critical Values for Two-Tailed Bonferroni-Dunn Test [64]

| **Number of Classifiers** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|
| $q_{0.05}$ | 1.960 | 2.241 | 2.394 | 2.498 | 2.576 | 2.638 | 2.690 | 2.724 | 2.773 |
| $q_{0.10}$ | 1.645 | 1.960 | 2.128 | 2.241 | 2.326 | 2.394 | 2.450 | 2.498 | 2.539 |

## 4.3 Experimental Results

In this section, SVMs with the adaptive combined kernel functions are trained and tested on benchmark datasets. These benchmark datasets are the binary classification problems and the regression problems. The results of the proposed methods are compared to the SVMs with the conventional kernel functions. Besides, k-nearest neighbor (k-NN) and grid search are used for comparing with the proposed methods on some experiments, and they are reported in the following sub-sections.

### 4.3.1 Classification Problems

At the beginning, k-NN for k = 1, 3, 5, and 7 are applied on the datasets from Table 4-1. Both simple k-NN and weighted k-NN are compared. In simple k-NN, an example is classified by the majority vote of its k neighbors. For weighted k-NN, the Euclidean distance is used to compute the weight of each neighbor, which is equal to 1/distance. The class of each neighbor is multiplied by this weight before voting. The experimental results are shown in Table 4-5.

The results show that each value of k is suitable for different datasets. Although, the average error of simple 7-NN is lower than the other k-NNs, the average rank of simple 3-NN is better than the others. Thus, the Friedman test is used to check whether the measured average ranks are significantly different from the mean rank, $R_j$ = 4. From Table 4-5, 7 algorithms are compared on 12 datasets, and the Friedman statistic is as follows.

$$\chi_F^2 = \frac{12(12)}{7(7+1)}\left[\left(4.4583^2+3.5^2+4.2917^2+3.875^2+4^2+3.875^2+4^2\right)-\frac{7(7+1)^2}{4}\right]=1.9762 \qquad (135)$$

$$F_F = \frac{(12-1)1.9762}{12(7-1)-1.9762} = 0.3104\ . \qquad (136)$$

$F_F$ is distributed according to the F-distribution with $7-1=6$ and $(7-1)\times(12-1)=66$ degrees of freedom. The critical value of $F(6,66)$ for $\alpha=0.05$ is $2.2461$, so we cannot reject the null-hypothesis. Hence, the performances of k-NN at various k are not different. Therefore, 1-NN that is the simplest and yields the good results will be compared with the other algorithms in this research.

Table 4-5: Average Percentage Error of k-Nearest Neighbors on Classification Problems

| **Datasets** | **1-NN** | **3-NN** | | **5-NN** | | **7-NN** | |
|---|---|---|---|---|---|---|---|
| | | **No Weight** | **With Weight** | **No Weight** | **With Weight** | **No Weight** | **With Weight** |
| Australian | 19.8551 (7) | 16.6667 (4) | 17.5362 (6) | 15.7971 (2) | 16.9565 (5) | **15.6522** (1) | 16.5217 (3) |
| Flare | 25.1345 (4) | 19.5103 (2) | 38.9334 (5) | 19.6025 (3) | 42.0249 (6) | **19.0439** (1) | 43.5286 (7) |
| German | 32.6000 (7) | 27.6000 (2) | **27.2000** (1) | 28.6000 (5) | 28.4000 (3) | 28.6000 (5) | 28.6000 (5) |
| Glass2 | 22.6704 (2) | 23.9204 (6) | 23.2954 (4) | 23.3333 (5) | **21.4583** (1) | 23.9394 (7) | 22.6894 (3) |
| Heart | 24.8148 (7) | 20.7407 (5) | 21.4815 (6) | 19.2593 (3.5) | 19.2593 (3.5) | 18.8889 (2) | **18.5185** (1) |
| Ionosphere | **12.5231** (1) | 13.6660 (2) | 13.9477 (3) | 15.3722 (4.5) | 15.3722 (4.5) | 16.2294 (6) | 16.7968 (7) |
| LiverDisorder | 39.7101 (4) | 37.6812 (2) | **36.5218** (1) | 40.8696 (6.5) | 39.7102 (5) | 40.8696 (6.5) | 38.8406 (3) |
| PimaDiabetes | 29.4355 (7) | **25.7915** (1) | 26.0530 (3) | 26.5707 (5) | 26.7032 (6) | 26.0521 (2) | 26.3136 (4) |
| Sonar | **12.9965** (1) | 17.3171 (5) | 17.3171 (5) | 17.3171 (5) | 16.3531 (2) | 19.2567 (7) | 16.3879 (3) |
| ThreeOf9 | 20.5082 (5) | 20.7081 (6) | 23.0497 (7) | 14.4546 (3) | 16.7961 (4) | **1.7609** (1) | 4.1024 (2) |
| Tic-Tac-Toe | **0.0000** (1.5) | **0.0000** (1.5) | 1.8793 (5) | 0.3131 (3) | 2.1924 (6) | 0.8345 (4) | 2.7138 (7) |
| Tokyo | 9.1748 (7) | 7.2993 (5.5) | 7.2993 (5.5) | **6.7785** (1) | 6.9874 (2) | 7.1962 (4) | 7.0915 (3) |
| **Average Error** | 20.7853 | 19.2418 | 21.2095 | 19.0223 | 21.0178 | **18.1936** | 20.1754 |
| **Average Rank** | 4.4583 | **3.5000** | 4.2917 | 3.8750 | 4.0000 | 3.8750 | 4.0000 |

Then, SVM with the common kernel functions are tested on these datasets. The RBF and polynomial kernels with various parameter settings are applied. The average percentage error of RBF and polynomial kernels are reported in Table 4-6 and Table 4-7, respectively. Although the classification error is small with few parameter settings, many parameter settings yield high percentage error. In addition, we do not have any knowledge about the suitable parameters. The experiments with all possible parameters cannot be performed.

Table 4-6: Average Percentage Error of SVM with Parameter Setting on RBF Kernel

| **Datasets** | **C** | **Width of RBF ($\gamma$)** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **0.0001** | **0.001** | **0.01** | **0.1** | **1** | **10** |
| Australian | **0.1** | 44.4928 | 44.4928 | 44.4928 | 14.4928 | 14.7826 | 41.5942 |
| | **1.0** | 44.4928 | 44.4928 | 14.4928 | 14.4928 | **13.7681** | 18.5507 |
| Flare | **0.1** | **17.0734** | **17.0734** | **17.0734** | **17.0734** | **17.0734** | **17.0734** |
| | **1.0** | **17.0734** | **17.0734** | **17.0734** | 17.1673 | 17.4490 | 18.1989 |
| German | **0.1** | 30.0000 | 30.0000 | 30.0000 | 30.0000 | 30.0000 | 30.0000 |
| | **1.0** | 30.0000 | 30.0000 | 30.0000 | **24.9000** | 25.4000 | 29.8000 |
| Glass2 | **0.1** | 46.6856 | 46.6856 | 46.6856 | 46.6856 | 46.0795 | 45.4735 |
| | **1.0** | 46.6856 | 46.6856 | 46.6856 | 43.5606 | 25.7197 | **23.9015** |
| Heart | **0.1** | 44.4444 | 44.4444 | 44.4444 | **16.6667** | 17.7778 | 44.4444 |
| | **1.0** | 44.4444 | 44.4444 | 17.0370 | 15.5556 | 19.2593 | 30.3704 |
| Ionosphere | **0.1** | 35.8954 | 35.8954 | 35.8954 | 27.9235 | 7.1187 | 35.8954 |
| | **1.0** | 35.8954 | 35.8954 | 25.9276 | 8.2575 | **5.4085** | 11.1147 |
| LiverDisorder | **0.1** | 42.0290 | 42.0290 | 42.0290 | 42.0290 | 42.0290 | 42.0290 |
| | **1.0** | 42.0290 | 42.0290 | 42.0290 | 42.0290 | 40.8696 | **32.4638** |
| PimaDiabetes | **0.1** | 34.9003 | 34.9003 | 34.9003 | 34.9003 | 27.2167 | 32.5567 |
| | **1.0** | 34.9003 | 34.9003 | 34.9003 | 23.7068 | **23.4428** | 24.2288 |
| Sonar | **0.1** | 46.6318 | 46.6318 | 46.6318 | 46.6318 | 46.6318 | 46.6318 |
| | **1.0** | 46.6318 | 46.6318 | 34.1231 | 16.3531 | **11.0453** | 40.8827 |
| ThreeOf9 | **0.1** | 46.4858 | 46.4858 | 46.4858 | 16.0213 | 35.5492 | 46.4858 |
| | **1.0** | 46.4858 | 46.4858 | 18.3628 | 9.1928 | **0.0000** | 46.4858 |
| Tic-Tac-Toe | **0.1** | 34.6553 | 34.6553 | 34.6553 | 34.6553 | 23.5940 | 34.6553 |
| | **1.0** | 34.6553 | 34.6553 | 34.6553 | 12.4242 | **0.8344** | 34.6553 |
| Tokyo | **0.1** | 36.0798 | 36.0798 | 35.3507 | 10.1145 | 8.3432 | 17.2066 |
| | **1.0** | 36.0798 | 34.6210 | 10.9484 | 8.5498 | **7.2993** | 9.2834 |

Table 4-7: Average Percentage Error of SVM with Parameter Setting on Polynomial Kernel

| **Datasets** | **C** | **Degree of Polynomial ( *d* )** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1 (Linear)** | **2** | **4** | **6** | **8** | **10** |
| Australian | **0.1** | **14.4928** | 14.9275 | 15.7971 | 20.5797 | 22.7536 | 24.3478 |
| | **1.0** | **14.4928** | **14.4928** | 17.8261 | 22.4638 | 23.0435 | 23.6232 |
| Flare | **0.1** | 17.0734 | **16.9804** | 17.8233 | 19.3252 | 42.2763 | 51.2887 |
| | **1.0** | **16.9804** | 17.4494 | 18.4805 | 39.5138 | 64.9603 | 40.0781 |
| German | **0.1** | 24.6000 | **23.8000** | 30.4000 | 31.3000 | 31.1000 | 30.3000 |
| | **1.0** | **23.8000** | 25.7000 | 31.8000 | 31.3000 | 31.1000 | 30.3000 |
| Glass2 | **0.1** | 46.0795 | 39.2424 | 25.7386 | 22.6894 | 22.0265 | 20.1705 |
| | **1.0** | 39.2235 | 28.7879 | 25.7576 | **17.6894** | 19.5644 | 20.7955 |
| Heart | **0.1** | 15.9259 | 16.2963 | 22.2222 | 27.0370 | 24.4444 | 23.3333 |
| | **1.0** | **15.5556** | 19.2593 | 26.2963 | 25.1852 | 24.4444 | 23.3333 |
| Ionosphere | **0.1** | 13.1066 | **9.6901** | 13.1147 | 12.8249 | 13.3924 | 13.6740 |
| | **1.0** | 13.0986 | 10.5513 | 13.1147 | 12.8249 | 13.3924 | 13.6740 |
| LiverDisorder | **0.1** | 42.0290 | 42.0290 | 41.1594 | 30.7246 | 29.5652 | 28.6957 |
| | **1.0** | 42.3188 | 37.3913 | **27.2464** | 28.1159 | 28.9855 | 29.8551 |
| PimaDiabetes | **0.1** | 32.2918 | 22.9276 | **22.5303** | 23.9657 | 25.7898 | 28.2667 |
| | **1.0** | 22.6679 | 23.1831 | 23.4437 | 25.2712 | 28.0053 | 31.0016 |
| Sonar | **0.1** | 19.7213 | **11.5563** | 12.0325 | 12.0441 | 13.9489 | 16.8293 |
| | **1.0** | 20.7085 | 12.9965 | 12.0325 | 12.0441 | 13.9489 | 16.8293 |
| ThreeOf9 | **0.1** | 17.7746 | 1.9513 | **0.0000** | **0.0000** | **0.0000** | 1.3706 |
| | **1.0** | 18.5589 | 2.7356 | **0.0000** | **0.0000** | **0.0000** | 1.3706 |
| Tic-Tac-Toe | **0.1** | 16.2854 | 1.6699 | **0.2083** | 1.2522 | 1.6699 | 1.6699 |
| | **1.0** | 1.6699 | 0.2089 | **0.2083** | 1.2522 | 1.6699 | 1.6699 |
| Tokyo | **0.1** | 8.9676 | **7.0910** | 8.4457 | 10.1134 | 11.6787 | 12.0964 |
| | **1.0** | 8.1332 | 7.3004 | 10.1140 | 10.4254 | 11.2615 | 16.0537 |

Grid search is compared to the parameter settings in Table 4-8. In this experiment, grid search chooses a set of parameters that yields a small training error, and then this set of parameters is used for testing on validation data. Thus, the percentage errors of grid search on testing may not be the lowest. Although it seems that the percentage errors on testing of the parameter settings are better than those of grid search, the percentage error on training of grid search is lower. In real world applications, we do not know the testing error; only the training error can be calculated. Therefore, it is very difficult to select a good set of parameters that yields the lowest error on unseen data. Hence, the parameter settings cannot be performed in the real situations.

Table 4-8: Average Percentage Error of Parameter Settings and Grid Search

| **Datasets** | **Parameter Setting** | | | | **Grid Search** | | | |
|---|---|---|---|---|---|---|---|---|
| | **RBF** $C = 1.0, \gamma = 1.0$ | | **Polynomial** $C = 0.1, d = 2$ | | **RBF** | | **Polynomial** | |
| | **Train** | **Test** | **Train** | **Test** | **Train** | **Test** | **Train** | **Test** |
| Australian | 10.4710 | **13.7681** (1) | 13.8768 | 14.9275 (2) | 1.2340 | 21.4500 (1) | 0.0000 | 23.7680 (2) |
| Flare | 15.9006 | 17.4490 (2) | 16.6980 | **16.9804** (1) | 11.1880 | 18.8500 (1) | 11.6080 | 19.6060 (2) |
| German | 7.5250 | 25.4000 (2) | 16.1000 | **23.8000** (1) | 0.0000 | 28.9000 (1) | 0.0000 | 31.1000 (2) |
| Glass2 | 24.2325 | 25.7197 (1) | 33.5819 | 39.2424 (2) | 4.4460 | **17.0840** (1) | 0.0000 | 19.5455 (2) |
| Heart | 9.0741 | 19.2593 (2) | 13.2407 | **16.2963** (1) | 0.0000 | 28.5180 (2) | 0.0000 | 23.7020 (1) |
| Ionosphere | 1.8518 | **5.4085** (1) | 4.0597 | 9.6901 (2) | 0.0000 | 10.5520 (1) | 0.0000 | 12.8260 (2) |
| LiverDisorder | 37.9710 | 40.8696 (1) | 42.0290 | 42.0290 (2) | 12.1739 | 32.7536 (2) | 10.6522 | **29.2754** (1) |
| PimaDiabetes | 20.8012 | 23.4428 (2) | 22.2013 | **22.9276** (1) | 5.0780 | 26.3160 (1) | 1.4640 | 31.1280 (2) |
| Sonar | 0.0000 | **11.0453** (1) | 3.6044 | 11.5563 (2) | 0.0000 | 30.8600 (2) | 0.0000 | 11.5660 (1) |
| ThreeOf9 | 0.0000 | **0.0000** (1) | 1.8069 | 1.9513 (2) | 0.0000 | 0.1961 (2) | 0.0000 | **0.0000** (1) |
| Tic-Tac-Toe | 0.0000 | 0.8344 (1) | 1.6701 | 1.6699 (2) | 0.0000 | 1.2520 (2) | 0.0000 | **1.2500** (1) |
| Tokyo | 5.5527 | 7.2993 (2) | 6.3348 | **7.0910** (1) | 0.3380 | 9.5960 (1) | 0.0000 | 11.9900 (2) |
| **Average Error** | 11.1150 | **15.8747** | 14.6003 | 17.3468 | 2.8715 | 18.8606 | 1.9770 | 17.9796 |
| **Average Rank** | -- | **1.4167** | -- | 1.5833 | -- | **1.4167** | -- | 1.5833 |

Although the grid search is a good method for parameter selection, its running time depends on the step size and the number of parameters. In the combinations of multiple kernel functions, there are a lot of parameters that should not be selected by grid search because the running time will be exponentially increased. Hence, the (5+10)-ES with the difference objective functions are proposed for searching the high quality parameters of the kernel function and SVM. Training error (*TrnErr*), 5-subsets cross-validation on training error (*5SubsetOnTrnErr*), the bound of generalization error (*BoundOfGenErr*), and the bound of generalization error derived from the stability of SVM (*StabilityBound*) are used as the objective function in the evolutionary process. The experimental results of these objective functions are compared with the result of a grid search, where the parameters are varied with a fixed step-size. The average percentage errors from 5-fold cross-validation of all methods with the single RBF kernel function are compared in Table 4-9.

Table 4-9: Average Percentage Error of Single RBF Kernel Function

| **Datasets** | **Grid Search** | **Objective Functions of ES** | | | |
|---|---|---|---|---|---|
| | | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***StabilityBound*** |
| Australian | 21.4500 | 21.8346 (4) | 18.9855* (3) | 16.8116* (2) | **16.3768*** (1) |
| Flare | 18.8500 | 18.9671 (4) | **17.4485** (1) | 17.5429 (3) | 17.4490 (2) |
| German | 28.9000 | 29.7000 (3) | 29.3000 (2) | 29.8000 (4) | **28.9000** (1) |
| Glass2 | 17.0840 | 18.2955 (3) | **17.6894** (1.5) | **17.6894** (1.5) | 21.5720 (4) |
| Heart | 28.5180 | 29.6296 (4) | 22.9619* (2) | 27.7778 (3) | **20.0000*** (1) |
| Ionosphere | 10.5520 | 13.7502 (4) | 5.6942 (2) | **5.1268** (1) | 7.1026* (3) |
| LiverDisorder | 32.7536 | **32.1739** (2) | 36.5200 (4) | **32.1739** (2) | 32.1739 (2) |
| PimaDiabetes | 26.3160 | 26.1841 (3) | 26.1780 (2) | 26.5724 (4) | **24.4860** (1) |
| Sonar | 30.8600 | 25.8304 (4) | 22.9725 (3) | **12.0093*** (1) | 15.3070* (2) |
| ThreeOf9 | 0.1961 | 0.3903 (2) | **0.0000** (1) | 0.5863 (3) | 1.5572 (4) |
| Tic-Tac-Toe | 1.2520 | 0.9391 (3) | 0.7308 (2) | **0.6261** (1) | 3.7516 (4) |
| Tokyo | 9.5960 | 10.1162 (4) | 9.8033 (3) | 9.0729* (2) | **8.9807*** (1) |
| **Average Error** | 18.8606 | 18.9842 | 17.3570 | **16.3158** | 16.4714 |
| **Average Rank** | -- | 3.3333 | 2.2083 | 2.2917 | **2.1667** |

* Statistically significant at the level of 0.05 when compared to *TrnErr* objective function.

The results show that the average percentage error of ES with *TrnErr* is similar to the grid search. Although the grid search is easy to implement, its performance depends on the step-size. If the step-size is small, the grid search will be computationally expensive because the SVM model must be evaluated at many points of parameters within the grid. On the other hand, if the step-size is large, the grid search may not find a good solution. Whereas ES uses the random process, many solutions are simultaneously searched.

When different objective functions of ES are used, the percentage errors of *5SubsetOnTrnErr*, *BoundOfGenErr*, and *StabilityBound* are lower than those of *TrnErr*. Moreover, their percentage errors are significantly lower than those of *TrnErr* on some datasets, i.e. Australian, Heart, Ionosphere, Sonar, and Tokyo. Although *BoundOfGenErr* yields the lowest average percentage error on 12 datasets, the average rank on 12 datasets of *StabilityBound* is lower than *BoundOfGenErr* and the other objective functions.

Friedman test is used for a statistical testing on the average ranks of the different objective functions in Table 4-9. For 4 algorithms and 12 datasets, $F_F$ is distributed according to the F distribution with 4-1 = 3 and (4-1)$\times$(12-1) = 33 degrees of freedom. From the experimental results, the following Friedman test is used to check whether the measured average ranks are significantly different from the mean rank $R_j$ = 2.5:

$$\chi_F^2 = \frac{12(12)}{4(4+1)}\left[\left(3.3333^2 + 2.2083^2 + 2.2917^2 + 2.1667^2\right) - \frac{4(4+1)^2}{4}\right] = 6.7250 \tag{137}$$

$$F_F = \frac{(12-1)\,6.7250}{12(4-1) - 6.7250} = 2.5269 \tag{138}$$

This value implies that the average ranks of these 4 algorithms are significantly different from the mean rank at a significance level of 0.0791.

Then, we use the Bonferroni-Dunn test for a pairwise comparison. The value of $q_{0.10}$ for 4 classifiers is 2.128. The performances of two classifiers are significantly different if their corresponding average ranks differ by at least the critical difference

$$CD = (2.128)\sqrt{\frac{4(4+1)}{6(12)}} = 1.1690. \tag{139}$$

Although this critical difference is not sufficient to conclude about the performances of *5SubsetOnTrnErr*, *BoundOfGenErr*, and *StabilityBound*, the different of average rank between *TrnErr* and *StabilityBound* is very close to this critical difference (3.3333-2.1667 = 1.1666 $\approx$ 1.1690). The pairwise differences on the average ranks of ES with the different objective functions kernel are shown in Table 4-10.

Table 4-10: Pairwise Differences on Average Ranks of the Different Objective Functions for Single RBF Kernel

| **CD = 1.1690** | **Objective Functions** | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***Stability Bound*** |
|---|---|---|---|---|---|
| **Objective Functions** | **Average Rank** | **3.3333** | **2.2083** | **2.2917** | **2.1667** |
| ***TrnErr*** | **3.3333** | 0.0000 | 1.1250 | 1.0417 | 1.1667** |
| ***5SubsetOn TrnErr*** | **2.2083** | -- | 0.0000 | -0.0833 | 0.0417 |
| ***BoundOf GenErr*** | **2.2917** | -- | -- | 0.0000 | 0.1250 |
| ***Stability Bound*** | **2.1667** | -- | -- | -- | 0.0000 |

** This value is very close to the critical difference.

These results show that *TrnErr* is not the best objective function, and in fact it may guide ES to select a classifier which overfits the training data. On the other hand, *5SubsetOnTrnErr*, *BoundOfGenErr*, and *StabilityBound* are the approximations of the generalization performance of SVM. Thus, it can avoid the overfitting problem, resulting in better performance. The average percentage errors on training of these objective functions are shown in Table 4-11 and a chart for comparing between training and testing is shown in Figure 4-2.

When we consider the chart of the average accuracy on 12 datasets, we found that the average percentage errors of the grid search and *TrnErr* are lower than those of the other objective functions. However, the average percentage error on testing is different; the grid search and *TrnErr* yield higher average percentage errors on testing. This means that the lowest error on training is not the best choice for unseen data.

For the running time, it is rather obvious that the evolutionary strategy consumes a lot of time when it is compared to a single SVM. However, this process is indispensable, as the accuracy of the learned SVM depends heavily on the quality of the obtained parameters. Furthermore, determining high-quality parameters is an off-line process in most application, and thus this running time can be disregarded. Nevertheless, we found that the running time of each proposed method (ES with a different objective function) is less than that of the grid search with a large number of evaluations. The running time of each method on a fold of Sonar dataset is recorded and illustrated in Table 4-12.

Table 4-11: Average Percentage Error on Training of Single RBF Kernel Function

| Datasets | Grid Search | Objective Functions of ES | | | |
|---|---|---|---|---|---|
| | | *TrnErr* | *5SubsetOn TrnErr* | *BoundOf GenErr* | *Stability Bound* |
| Australian | 1.2340 | **1.2319** | 9.3120 | 4.6739 | 10.6667 |
| Flare | 11.1880 | **11.1868** | 17.0740 | 13.4858 | 16.2526 |
| German | **0.0000** | **0.0000** | **0.0000** | 0.1500 | 0.2250 |
| Glass2 | 4.4460 | **4.4439** | **4.4439** | 7.0593 | 22.3582 |
| Heart | 0.0000 | **0.0000** | **0.0000** | 0.9259 | 7.5926 |
| Ionosphere | **0.0000** | **0.0000** | 0.5000 | **0.0000** | 1.9934 |
| LiverDisorder | 12.1739 | **12.0301** | 12.2460 | 15.7971 | 29.6377 |
| PimaDiabetes | 5.0780 | **5.0778** | 6.3820 | 8.6609 | 9.5343 |
| Sonar | **0.0000** | **0.0000** | 0.2400 | **0.0000** | **0.0000** |
| ThreeOf9 | **0.0000** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Tic-Tac-Toe | **0.0000** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Tokyo | 0.3380 | **0.3129** | 0.7800 | 1.0949 | 0.6518 |
| **Average Error** | 2.8715 | **2.8570** | 4.2482 | 4.3207 | 8.2427 |

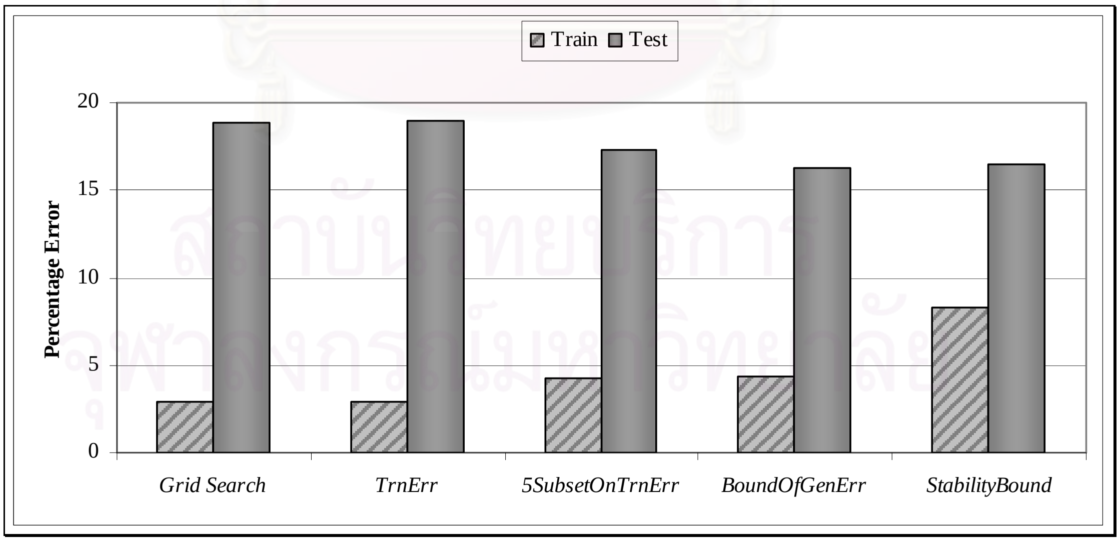


Figure 4-2: Chart of Average Percentage Error on Training and Testing of Single RBF Kernel Function with Different Objective Functions

Table 4-12: The Running Time of an SVM with the Single RBF Kernel that Uses Different Parameter Selection Methods when Training on a Fold of Sonar Dataset

| **Parameter Selection Methods** | **Running Time on Training (Hour)** |
|---|---|
| 1-NN | -- |
| Grid Search | 3:05:35 |
| ES with *TrnErr* | 0:23:18 |
| ES with *5SubsetOnTrnErr* | 1:56:31 |
| ES with *BoundOfGenErr* | 0:22:31 |
| ES with *StabilityBound* | 0:23:58 |

The proposed methods, the grid search, and 1-NN were run on a computer with an Intel Xeon 2.73 GHz CPU and 3.85 GB memory. For 1-NN, it is an instance based learning, which does not have the training process. Thus, the running time is used for calculating the distance between an example and the existing training data. In the grid search, the regularization parameter, the width of the RBF kernel, and the combination weight are varied by a log-scale form 0.0001, 0.0002, …, 0.001, 0.002, …, to 10.0.

On a fold of Sonar dataset, the running time on training of the grid search was about three hours, whereas the running time of ES with *TrnErr* objective function was 23.18 minutes. Therefore, the running time of the grid search is about 8 times longer than that of ES with *TrnErr*. In addition, the running time of ES with *TrnErr* is close to that of *BoundOfGenErr* and *StabilityBound*. However, we found that the running time of *5SubsetOnTrnErr* is about 5 times longer than those of *TrnErr*, *BoundOfGenErr*, and *StabilityBound*. For the case of 5-subsets cross-validation, SVM classifiers are trained and validated 5 times for each set of parameters. Therefore, this result is reasonable.

For other datasets, the running times of these methods have a similar trend; the running time of *5SubsetOnTrnErr* is more than those of *TrnErr*, *BoundOfGenErr*, and *StabilityBound*, and the running time of each proposed methods is less than that of the grid search with a large number of evaluations. Moreover, the number of generation of ES may be reduced. In our experiments the number of generation was fixed as 1000, but a good solution can be found in fewer generations. Graphs of different objective functions on Sonar dataset are shown in Figure 4-3. The objective function values of *TrnErr*, *5SubsetOnTrnErr*, and *BoundOfGenErr* quickly decrease in the first few generations whereas *StabilityBound* requires more generations.

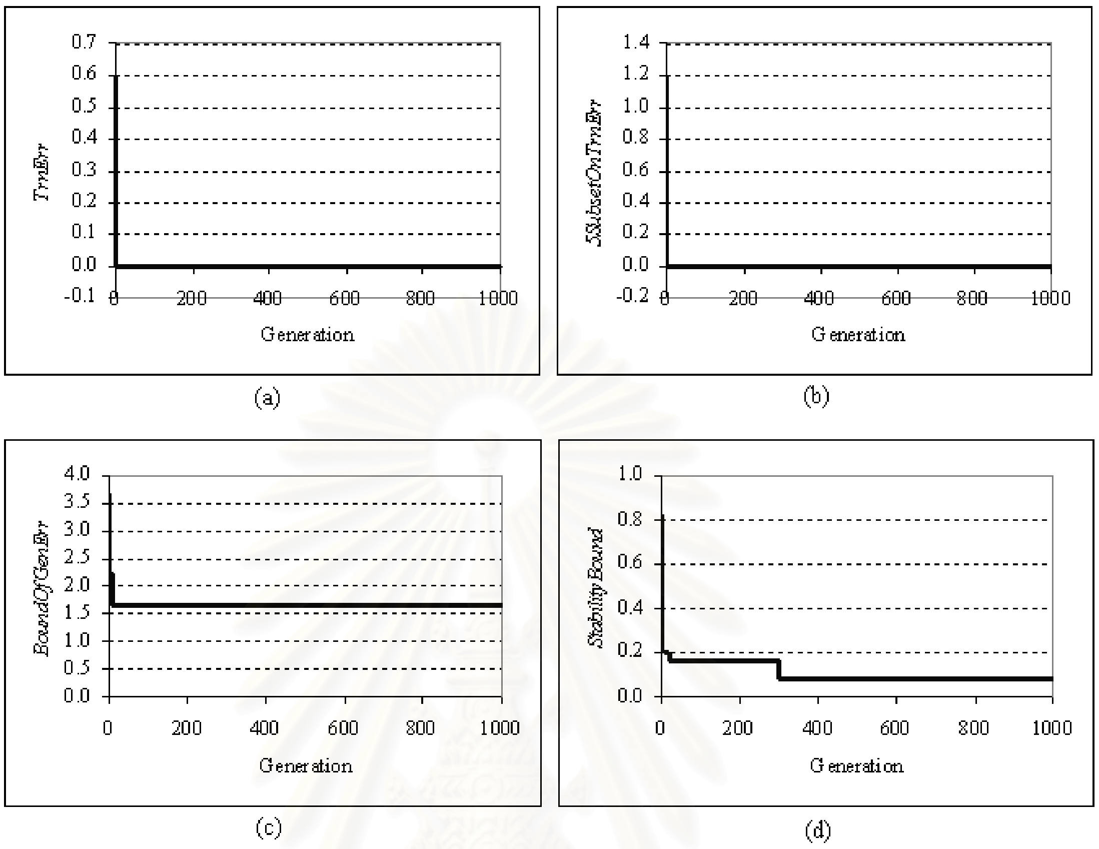


Figure 4-3: Graphs of Different Objective Functions for each Generation of ES on Sonar Datasets (a) *TrnErr* (b) *5SubsetOnTrnErr* (c) *BoundOfGenErr* (d) *StabilityBound*

For multi-scale RBF kernels, the average percentage errors and the ranks of each objective function using $n$-RBF when $n$ = 2, 3, 4, and 5 are illustrated in Table 4-13 – Table 4-16, respectively. Furthermore, the average percentage errors on 12 datasets for each objective function are illustrated by the graphs in Figure 4-4. These results show the performance of each objective function. For all $n$-RBF kernels, the average percentage errors of *5SubsetOnTrnErr* and *StabilityBound* are lower than those of *TrnErr*. There are many datasets where the percentage errors of *BoundOfGenErr* are lower than those of *TrnErr*. Moreover, the percentage errors of *5SubsetOnTrnErr*, *BoundOfGenErr*, and *StabilityBound* are significantly lower than those of *TrnErr* on some datasets.

Table 4-13: Average Percentage Error on Testing of 2-RBF Kernel Function

| **Datasets** | **Objective Functions of ES** | | | |
|---|---|---|---|---|
| | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***StabilityBound*** |
| Australian | 21.5942 (4) | 20.0000 (3) | **17.6812*** (2) | 16.8116* (1) |
| Flare | 20.6165 (4) | 18.4784* (3) | 18.3856* (2) | **18.0115*** (1) |
| German | 28.1000 (2) | 29.3000 (4) | 28.3000 (3) | **27.8000** (1) |
| Glass2 | 19.5265 (3.5) | 18.3144 (2) | **17.1023** (1) | 19.5265 (3.5) |
| Heart | 23.7037 (4) | **18.5185** (1) | 21.8519 (2) | 22.5926 (3) |
| Ionosphere | 5.6821 (3) | **5.1268** (1.5) | **5.1268** (1.5) | 6.8370 (4) |
| LiverDisorder | 31.9669 (2) | 33.9145 (4) | 32.4638 (3) | **31.8841** (1) |
| PimaDiabetes | 30.6078 (4) | 24.4920* (2) | 28.9186 (3) | **24.0964*** (1) |
| Sonar | 28.2811 (4) | 18.8502 (2) | 20.2439 (3) | **17.2706** (1) |
| ThreeOf9 | 0.1942 (2) | **0.0000** (1) | 1.1727 (3) | 1.9475 (4) |
| Tic-Tac-Toe | **0.8344** (1.5) | **0.8344** (1.5) | 1.1475 (3) | 1.8766 (4) |
| Tokyo | 11.4845 (4) | **7.9233*** (1) | 7.9238* (2) | 8.7724* (3) |
| **Average Error** | 18.5493 | **16.3127** | 16.6931 | 16.4522 |
| **Average Rank** | 3.1667 | **2.1667** | 2.3750 | 2.2917 |

* Statistically significant at the level of 0.05 when compared to *TrnErr* objective function.

Table 4-14: Average Percentage Error on Testing of 3-RBF Kernel Function

| **Datasets** | **Objective Functions of ES** | | | |
|---|---|---|---|---|
| | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***StabilityBound*** |
| Australian | 20.7246 (4) | 18.9855 (2) | 20.1449 (3) | **16.9565*** (1) |
| Flare | 19.6029 (4) | 18.9459 (2) | 19.1330 (3) | **17.9176** (1) |
| German | **27.5000** (1) | 29.5000 (4) | 28.6000 (3) | 27.7000 (2) |
| Glass2 | 18.9015 (2) | **18.2955** (1) | 18.9394 (3) | 19.5265 (4) |
| Heart | 23.3333 (3) | 23.3346 (4) | **18.1481** (1) | 22.2222 (2) |
| Ionosphere | 6.5553 (4) | 5.4085 (2) | **4.8451*** (1) | 6.5352 (3) |
| LiverDisorder | **30.1449** (1) | 33.3340 (4) | 32.1739 (3) | 31.8841 (2) |
| PimaDiabetes | 30.2215 (4) | **25.1057*** (1) | 28.3957* (3) | 25.4108* (2) |
| Sonar | 25.4704 (4) | **12.0790*** (1) | 16.6156* (3) | 15.1057* (2) |
| ThreeOf9 | **0.0000** (1.5) | **0.0000** (1.5) | 1.7590 (3) | 3.7160 (4) |
| Tic-Tac-Toe | **1.0433** (1) | 1.1480 (2.5) | 1.1480 (2.5) | 1.6699 (4) |
| Tokyo | 10.3232 (4) | 8.4457* (3) | **8.2390*** (1) | 8.3333* (2) |
| **Average Error** | 17.8184 | **16.2152** | 16.5118 | 16.4148 |
| **Average Rank** | 2.7917 | **2.3333** | 2.4583 | 2.4167 |

* Statistically significant at the level of 0.05 when compared to *TrnErr* objective function.

Table 4-15: Average Percentage Error on Testing of 4-RBF Kernel Function

| **Datasets** | **Objective Functions of ES** | | | |
|---|---|---|---|---|
| | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***StabilityBound*** |
| Australian | 20.7246 (4) | 18.9855 (2) | 20.0000 (3) | **16.6667*** (1) |
| Flare | 19.4151 (3) | 19.1344 (2) | 19.6964 (4) | **17.7853** (1) |
| German | 26.2000 (2) | **24.1000*** (1) | 26.5000 (3) | 27.7000 (4) |
| Glass2 | 18.9394 (2) | 19.56448 (4) | **17.0833** (1) | 19.5265 (3) |
| Heart | 21.8519 (4) | **16.2957*** (1) | 18.5185* (2) | 20.3704 (3) |
| Ionosphere | **4.5553** (1) | 6.2656 (3) | 5.3964 (2) | 6.2696 (4) |
| LiverDisorder | **31.3043** (1) | 34.4950 (4) | 32.1739 (3) | 31.8841 (2) |
| PimaDiabetes | 30.4821 (4) | 26.4680 (3) | 26.4305* (2) | **23.7017*** (1) |
| Sonar | 15.8885 (2) | **11.1150** (1) | 22.2416 (4) | 21.5796 (3) |
| ThreeOf9 | 0.1961 (2) | **0.0000** (1) | 1.5648 (4) | 0.7843 (3) |
| Tic-Tac-Toe | **1.0438** (1) | 1.3557 (4) | 1.1480 (2) | 1.2522 (3) |
| Tokyo | 10.0115 (4) | 8.2365* (2) | **8.1337*** (1) | 8.5509* (3) |
| **Average Error** | 16.7177 | **15.5013** | 16.5739 | 16.3393 |
| **Average Rank** | 2.5000 | **2.3333** | 2.5833 | 2.5833 |

* Statistically significant at the level of 0.05 when compared to *TrnErr* objective function.

Table 4-16: Average Percentage Error on Testing of 5-RBF Kernel Function

| **Datasets** | **Objective Functions of ES** | | | |
|---|---|---|---|---|
| | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***StabilityBound*** |
| Australian | 21.0145 (4) | 18.8058* (2) | 19.4203* (3) | **16.3768*** (1) |
| Flare | 19.5090 (4) | 18.9452 (2) | 19.5086 (3) | **17.8124** (1) |
| German | 25.9000 (2) | **24.5000*** (1) | 26.8000 (3) | 27.7000 (4) |
| Glass2 | **17.6894** (1) | 20.1136 (4) | 17.7273 (2) | 18.2765 (3) |
| Heart | 24.8148 (4) | **15.1856*** (1) | 18.8889* (2) | 21.1111 (3) |
| Ionosphere | **5.1268** (1) | 8.2666 (3) | 10.2696 (4) | 6.2656 (2) |
| LiverDisorder | 31.3043 (2) | 34.7848 (4) | **30.1449** (1) | 31.5942 (3) |
| PimaDiabetes | 30.9982 (4) | 25.2724* (2) | 27.2235* (3) | **23.9275*** (1) |
| Sonar | 10.5807 (2) | **9.1629** (1) | 23.6934 (4) | 20.6039 (3) |
| ThreeOf9 | **0.0000** (1.5) | 0.0000 (1.5) | 1.7571 (4) | 0.7805 (3) |
| Tic-Tac-Toe | **0.6266** (1) | 1.2516 (2) | 1.2522 (3.5) | 1.2522 (3.5) |
| Tokyo | 9.5833 (4) | 8.6461 (3) | 8.0307* (2) | **7.4051*** (1) |
| **Average Error** | 16.4290 | **15.4112** | 17.0597 | 16.0922 |
| **Average Rank** | 2.5417 | **2.2083** | 2.8750 | 2.3750 |

* Statistically significant at the level of 0.05 when compared to *TrnErr* objective function.

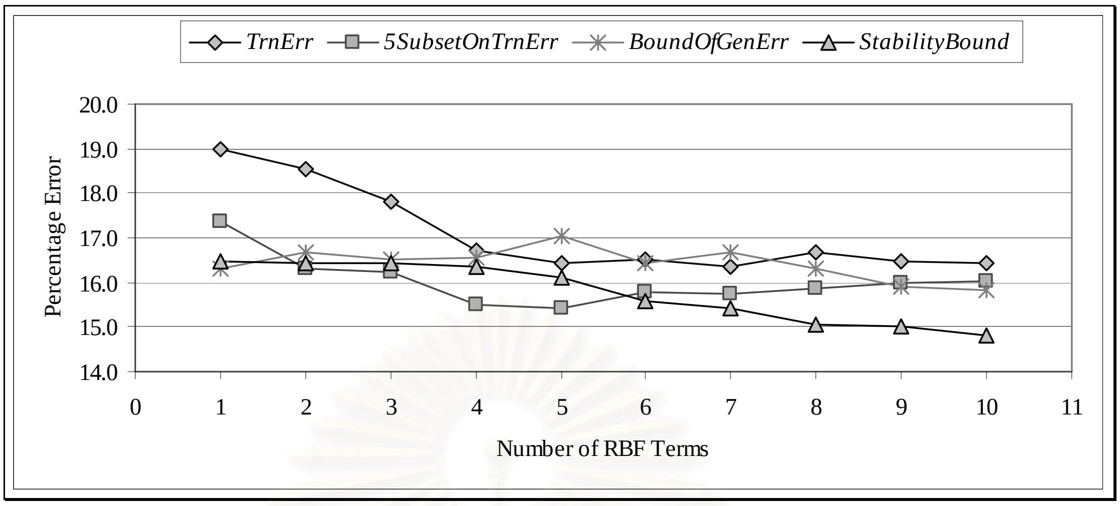


Figure 4-4: Graph of Average Percentage Error at Different Number of RBF Terms

When the average rank is considered, we found that *5SubsetOnTrnErr* performed the best for all $n$-RBF when $n$ = 2, 3, 4, and 5. As shown in Figure 4-4, the average percentage error on 12 datasets of *5SubsetOnTrnErr* is lower than the other objective functions for all $n$-RBF when $n$ = 2, 3, 4, and 5. When *5SubsetOnTrnErr* is used as an objective function in the evolutionary process, the average percentage error of $n$-RBF increases with the number of RBF terms.

The average percentage errors of *TrnErr* rapidly decrease until a specific number of terms of RBF kernels is reached. After that, they are unchanged or slightly increase. Although the average percentage error of *BoundOfGenErr* is the lowest on the single RBF kernel function, for some of the multi-scale RBF kernels, the average percentage errors of *BoundOfGenErr* increase. We notice that these average percentage errors of *BoundOfGenErr* are varied in a range of percentage error.

When *StabilityBound* is used as an objective function, there is a trend that the average percentage error on all 12 datasets decreases with the number of terms of RBF kernels. Although *StabilityBound* does not provide lower average percentage errors than those of *BoundOfGenErr* for the single RBF kernel function, it yields the results that are better than the other objective functions when the multiple RBF kernel functions are combined. Therefore, *StabilityBound* is a good choice for an objective function, which yields good results when using a more flexible kernel function. Moreover, increasing the number of terms of the RBF kernels contributes positively to the performance.

Although it is not always the case that the kernel with the largest number of terms yields the best result, more RBF terms usually provide better outcomes and should be employed when there are no time constraints. Therefore, the results of 10-RBF kernel function, which combines the maximum number of RBF terms in our experiments, are illustrated in Table 4-17.

Table 4-17: Average Percentage Error on Testing of 10-RBF Kernel Function

| **Datasets** | **1-NN** | **Objective Functions of ES** | | | |
|---|---|---|---|---|---|
| | | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***Stability Bound*** |
| Australian | 19.8551 (4) | 21.0145 (5) | 19.1101 (2) | 19.4203 (3) | **16.3768*** (1) |
| Flare | 25.1345 (5) | 18.4788 (4) | **17.0736*** (1) | 17.4323* (3) | 17.3547* (2) |
| German | 32.6000 (5) | 28.9000 (4) | 28.3000 (2) | 28.7000 (3) | **26.8000*** (1) |
| Glass2 | 22.6704 (5) | 17.0833 (2.5) | 18.2386 (4) | **16.4773** (1) | 17.0833 (2.5) |
| Heart | 24.8148 (5) | 23.7037 (4) | 21.8519 (2) | 22.2222 (3) | **19.6296*** (1) |
| Ionosphere | 12.5231 (5) | 6.2736 (4) | 6.2696 (3) | 5.1308 (2) | **5.1026** (1) |
| LiverDisorder | 39.7101 (5) | 32.4638 (3) | 35.6528 (4) | 32.1739 (2) | **31.5942** (1) |
| PimaDiabetes | 29.4355 (5) | 23.9674 (2) | 25.4000 (4) | 24.3536 (3) | **22.9234** (1) |
| Sonar | 12.9965 (3) | 14.3902 (5) | **10.1143** (1) | 13.4379 (4) | 12.9849 (2) |
| ThreeOf9 | 20.5082 (5) | **0.0000** (2) | **0.0000** (2) | 0.5882 (4) | **0.0000** (2) |
| Tic-Tac-Toe | **0.0000** (1) | 1.0433 (3) | 1.3563 (5) | 1.1480 (4) | 0.9391 (2) |
| Tokyo | 9.1748 (4) | 9.9078 (5) | 8.8617 (2) | 8.8662 (3) | **7.0910*** (1) |
| **Average Error** | 20.7853 | 16.4355 | 16.0191 | 15.8292 | **14.8233** |
| **Average Rank** | 4.3333 | 3.6250 | 2.6667 | 2.9167 | **1.4583** |

* Statistically significant at the level of 0.05 when compared to *TrnErr* objective function.

From Table 4-17, the average percentage errors of *StabilityBound* are lower than the other objective functions and 1-NN. Moreover, the average rank of *StabilityBound* is also the lowest when it is compared to the other algorithms. The Friedman test is used to check that the average ranks are significantly different from the mean rank $R_j$ = 3 at the significance level of 0.05. Then, Bonferroni-Dunn test is used for a pairwise comparison. The pairwise differences on the average ranks of these algorithms are shown in Table 4-18. These results show that the performance of *StabilityBound* is significantly better than 1-NN, *TrnErr*, and *BoundOfGenErr*; however, it is not sufficient to conclude about *5SubsetOnTrnErr*. Moreover, the performance of *5SubsetOnTrnErr* is also significantly better than 1-NN.

Table 4-18: Pairwise Differences on Average Ranks of 1-NN and 10-RBF Kernel Function with the Different Objective Functions

| **CD = 1.4466** | **Algorithms** | **1-NN** | ***TrnErr*** | ***5Subset OnTrnErr*** | ***BoundOf GenErr*** | ***Stability Bound*** |
|---|---|---|---|---|---|---|
| **Algorithms** | **Average Rank** | **4.3333** | **3.6250** | **2.6667** | **2.9167** | **1.4583** |
| **1-NN** | **4.3333** | 0.0000 | 0.7083 | **1.6667*** | 1.4167 | **2.8750*** |
| ***TrnErr*** | **3.6250** | -- | 0.0000 | 0.9583 | 0.7083 | **2.1667*** |
| ***5Subset OnTrnErr*** | **2.6667** | -- | -- | 0.0000 | -0.2500 | 1.2083 |
| ***BoundOf GenErr*** | **2.9167** | -- | -- | -- | 0.0000 | **1.4583*** |
| ***Stability Bound*** | **1.4583** | -- | -- | -- | -- | 0.0000 |

* Significantly Different at the level of 0.10.

Similar to the single RBF kernel, *TrnErr* may guide to a classifier which overfits training data. The average percentage error on training of *TrnErr* is lower than the other objective functions, whereas the average percentage error on testing of *TrnErr* is higher than the other objective functions. The average percentage errors on training of a 10-RBF kernel for each objective function are illustrated in Table 4-19. Furthermore, a chart for comparing between training and testing of the 10-RBF kernel with different objective functions is shown in Figure 4-5. This chart shows that the average percentage errors on testing of SVM based on adaptive multi-scale RBF kernels with different objective functions are better than those of 1-NN.

However, since the multi-scale RBF kernels are more flexible according to the terms of RBF, a set of parameters may create a decision surface that overfits training data. The experimental results show that the *StabilityBound* can solve this problem. When we consider the chart of the average accuracy on 12 datasets, we found that the training error of the *StabilityBound* is higher than those of the other objective functions. However, the average percentage error on testing is different; the *StabilityBound* yields the lowest average percentage error on testing. This means that the lowest error on training is not the best choice for unseen data, while the *StabilityBound* yields the results that do not overfit training data.

Table 4-19: Average Percentage Error on Training of 10-RBF Kernel Function

| **Datasets** | **Objective Functions of ES** | | | |
|---|---|---|---|---|
| | ***TrnErr*** | ***5SubsetOn TrnErr*** | ***BoundOf GenErr*** | ***StabilityBound*** |
| Australian | **1.0870** | 1.8478 | 2.3913 | 8.6232 |
| Flare | **11.8195** | 12.7810 | 12.6879 | 15.6426 |
| German | **0.0000** | 0.0250 | 0.5000 | 2.7750 |
| Glass2 | **3.3729** | 4.4439 | 7.6747 | 15.1767 |
| Heart | **0.6481** | 2.7778 | 4.2593 | 8.2407 |
| Ionosphere | 0.1423 | **0.0712** | 0.4270 | 2.5656 |
| LiverDisorder | 11.3768 | **10.4348** | 13.4783 | 25.7246 |
| PimaDiabetes | **4.3609** | 10.7759 | 8.0060 | 19.7921 |
| Sonar | **0.0000** | **0.0000** | **0.0000** | 1.1998 |
| ThreeOf9 | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Tic-Tac-Toe | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| Tokyo | **0.6779** | 2.1099 | 0.8083 | 5.7873 |
| **Average Error** | **2.7905** | 3.7723 | 4.1861 | 8.7940 |

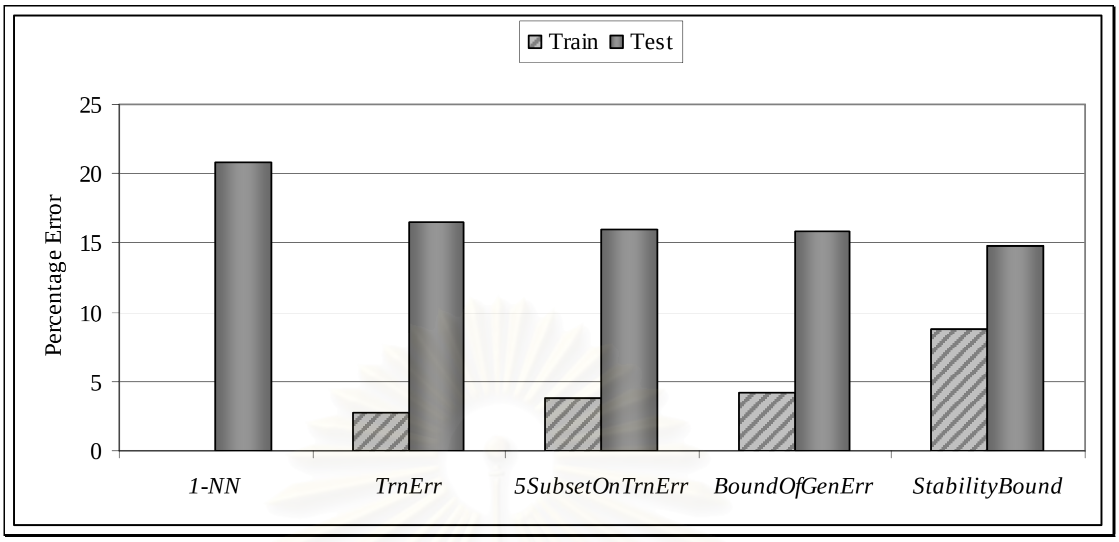


Figure 4-5: Chart of Average Percentage Error on Training of 1-NN and Both Training and Testing of 10-RBF Kernel with Different Objective Functions

In order to illustrate that the multi-scale RBF kernels can improve the classification performance of a single RBF kernel, Friedman test is used for testing the average ranks of 1-NN, grid search, single RBF kernel, and 10-RBF kernel that uses the *StabilityBound* as the objective function in evolutionary process. The results and the ranks are compared in Table 4-20.

By the Friedman test, the average ranks of these 4 algorithms and 12 datasets are significantly different from the mean rank $R_j$ = 2.5 at the significant level of 0.05. Then, the Bonferroni-Dunn test is used for a pairwise comparison. The pairwise differences on the average ranks of these algorithms are shown in Table 4-21.

The results show that the performance of a 10-RBF kernel is significantly better than single RBF kernel. Moreover, the performance of SVM with the 10-RBF kernel that uses the *StabilityBound* as an objective function in the evolutionary process is also significantly better than 1-NN and grid search on single RBF kernel.

Table 4-20: Average Percentage Error on Testing of 1-NN, Grid Search, and ES with the *StabilityBound* on RBF and 10-RBF Kernel Functions

| **Datasets** | **1-NN** | **Grid Search** | **ES with the *StabilityBound* Objective Function** | |
|---|---|---|---|---|
| | | | **RBF Kernel** | **10-RBF Kernel** |
| Australian | 19.8551 (4) | 21.4500 (3) | **16.3768** (1.5) | **16.3768** (1.5) |
| Flare | 25.1345 (4) | 18.8500 (3) | 17.4490 (2) | **17.3547** (1) |
| German | 32.6000 (4) | 28.9000 (2.5) | 28.9000 (2.5) | **26.8000** (1) |
| Glass2 | 22.6704 (4) | 17.0840 (2) | 21.5720 (3) | **17.0833** (1) |
| Heart | 24.8148 (3) | 28.5180 (4) | 20.0000 (2) | **19.6296** (1) |
| Ionosphere | 12.5231 (4) | 10.5520 (3) | 6.2656 (2) | **5.1026** (1) |
| LiverDisorder | 39.7101 (4) | 32.7536 (3) | 32.1739 (2) | **31.5942** (1) |
| PimaDiabetes | 29.4355 (4) | 26.3160 (3) | 24.4860 (2) | **22.9234** (1) |
| Sonar | 12.9965 (2) | 30.8600 (4) | 14.4010 (3) | **12.9849** (1) |
| ThreeOf9 | 20.5082 (4) | 0.1961 (2) | 1.5572 (3) | **0.0000** (1) |
| Tic-Tac-Toe | **0.0000** (1) | 1.2520 (3) | 3.7516 (4) | 0.9391 (2) |
| Tokyo | 9.1748 (3) | 9.5960 (4) | 8.9807 (2) | **7.0910** (1) |
| **Average Error** | 20.7853 | 18.8606 | 16.3262 | **14.8233** |
| **Average Rank** | 3.4167 | 3.0417 | 2.4167 | **1.1250** |

Table 4-21: Pairwise Differences on Average Ranks of 1-NN, Grid Search, and ES with the *StabilityBound* on Single RBF and 10-RBF Kernel Functions

| **CD = 1.1690** | **Algorithms** | **1-NN** | **Grid Search** | **ES RBF** | **ES 10-RBF** |
|---|---|---|---|---|---|
| **Algorithms** | **Average Rank** | **3.4167** | **3.0417** | **2.4167** | **1.1250** |
| **1-NN** | **3.4167** | 0.0000 | 0.3750 | 1.0000 | **2.2917*** |
| **Grid Search** | **3.0417** | -- | 0.0000 | 0.6250 | **1.9167*** |
| **ES RBF** | **2.4167** | -- | -- | 0.0000 | **1.2917*** |
| **ES 10-RBF** | **1.1250** | -- | -- | -- | 0.0000 |

* Significantly Different at the level of 0.10.

Furthermore, the other combined kernel functions are evaluated. The parameters of SVM with a multi-degree polynomial kernel, linear combination of polynomial and RBF kernels, and multiplication of polynomial and RBF kernels are selected by (5+10)-ES. For the multi-degree polynomial kernel, we use 10 terms of polynomial sub-kernels and the degrees of polynomial sub-kernels are fixed at 1, 2, … , 10. Only the weights of sub-kernels and a regularization parameter of SVM are the adjustable parameters. The parameter vector $\vec{v}$ is represented by the following equation:

$$\vec{v} = (C, a_0, a_1, a_2, \ldots, a_{n-1}), \tag{140}$$

where $C$ is the regularization parameter, $a_0$ is the weight of polynomial sub-kernel at degree 1 (linear sub-kernel) and it is fixed as 1, $a_i$ for $i = 1, \ldots, n-1$ are the weights of polynomial sub-kernels, and $n$ is the number of terms of polynomial sub-kernel functions.

For the linear combination and the multiplication of polynomial and RBF kernels, we combine an RBF kernel with a polynomial kernel. Their parameters are selected by (5+10)-ES, which the parameter vector $\vec{v}$ is represented by:

$$\vec{v} = (C, a_0, d, a_1, \gamma), \tag{141}$$

where $C$ is the regularization parameter, $a_0$ is the weight of polynomial sub-kernel that is fixed as 1, $d$ is the degree of polynomial sub-kernel, $a_1$ is the weight of RBF sub-kernel, and $\gamma$ is the width of RBF sub-kernel. The experimental results of these kernel functions are compared to the grid search on single RBF and single polynomial kernel functions, and ES on 10-RBF kernel in Table 4-22. The pairwise differences on the average ranks are shown in Table 4-23.

As shown in Table 4-22, the performance of 10-RBF kernel is still better than the other kernel functions. Friedman test is used to check that the measured average ranks of 6 algorithms and 12 datasets are significantly different from the mean rank $R_j$ = 3.5 at a significance level of 0.05. For the pairwise comparisons, the performance of 10-RBF kernel is significantly better than those of Poly*RBF kernel and the grid search on both RBF and polynomial kernels. Moreover, the performance of Poly+RBF kernel is significantly better than that of Poly*RBF kernel. However, it is not sufficient to conclude about the other kernel functions.

Table 4-22: Average Percentage Error of SVM with Different Kernel Functions when using *StabilityBound* as the Objective Function in Evolutionary Process

| **Datasets** | **Grid Search** | | **ES** | | | |
|---|---|---|---|---|---|---|
| | **RBF** | **Poly** | **10-Poly** | **Poly+RBF** | **Poly*RBF** | **10-RBF** |
| Australian | 21.4500 (3) | 23.7680 (6) | 23.1884 (4) | 16.8116 (2) | 23.6232 (5) | **16.3768** (1) |
| Flare | 18.8500 (3) | 19.6060 (5) | 17.4477 (2) | 19.1365 (4) | 20.1659 (6) | **17.3547** (1) |
| German | 28.9000 (4) | 31.1000 (6) | 30.2000 (5) | 27.2000 (2) | 28.8000 (3) | **26.8000** (1) |
| Glass2 | 17.0840 (2) | 19.5455 (3.5) | 19.5455 (3.5) | 22.0455 (6) | 20.7386 (5) | **17.0833** (1) |
| Heart | 28.5180 (5) | 23.7020 (4) | 22.5926 (3) | 20.7407 (2) | 33.3333 (6) | **19.6296** (1) |
| Ionosphere | 10.5520 (4) | 12.8260 (6) | 12.2857 (5) | 9.9839 (3) | 8.8209 (2) | **5.1026** (1) |
| LiverDisorder | 32.7536 (6) | 29.2754 (3) | 28.9855 (2) | **28.6957** (1) | 30.7246 (4) | 31.5942 (5) |
| PimaDiabetes | 26.3160 (4) | 31.1280 (6) | 23.1848 (2) | 25.0089 (3) | 27.2167 (5) | **22.9234** (1) |
| Sonar | 30.8600 (6) | 11.5660 (2) | **11.5447** (1) | 12.9733 (3) | 30.8130 (5) | 12.9849 (4) |
| ThreeOf9 | 0.1961 (3) | **0.0000** (1.5) | 1.1727 (4) | 1.9513 (5) | 2.7204 (6) | **0.0000** (1.5) |
| Tic-Tac-Toe | 1.2520 (4) | 1.2500 (3) | 1.5658 (5) | **0.6272** (1) | 2.2971 (6) | 0.9391 (2) |
| Tokyo | 9.5960 (3) | 11.9900 (6) | 9.8004 (4) | 8.3421 (2) | 11.4682 (5) | **7.0910** (1) |
| **Average Error** | 18.8606 | 17.9797 | 16.7928 | 16.1264 | 20.0602 | **14.8233** |
| **Average Rank** | 3.9167 | 4.3333 | 3.3750 | 2.8333 | 4.8333 | **1.7083** |

Table 4-23: Pairwise Differences on Average Ranks of Grid Search and ES with the Different Combined Kernel Functions

| **CD = 1.7769** | | **Algorithms** | **Grid Search** | | **ES** | | | |
|---|---|---|---|---|---|---|---|---|
| | | | **RBF** | **Poly** | **10-Poly** | **Poly + RBF** | **Poly * RBF** | **10-RBF** |
| **Algorithms** | | **Average Rank** | **3.9167** | **4.3333** | **3.3750** | **2.8333** | **4.8333** | **1.7083** |
| **Grid Search** | **RBF** | **3.9167** | 0.0000 | -0.4167 | 0.5417 | 1.0833 | -0.9167 | **2.2083*** |
| | **Poly** | **4.3333** | -- | 0.0000 | 0.9583 | 1.5000 | -0.5000 | **2.6250*** |
| **ES** | **10-Poly** | **3.3750** | -- | -- | 0.0000 | 0.5417 | -1.4583 | 1.6667 |
| | **Poly + RBF** | **2.8333** | -- | -- | -- | 0.0000 | **-2.0000*** | 1.1250 |
| | **Poly * RBF** | **4.8333** | -- | -- | -- | -- | 0.0000 | **3.1250*** |
| | **10-RBF** | **1.7083** | -- | -- | -- | -- | -- | 0.0000 |

* Significantly Different at the level of 0.10.

Therefore, we consider the average percentage error of this kernel functions. The average percentage errors of 10-Poly kernel are lower than those of single polynomial kernel at the most of datasets, except for ThreeOf9 and Tic-Tac-Toe where their average percentage errors are lower than the other datasets. This means that the multi-degree polynomial kernel can improve the performance of SVM that uses the single polynomial kernel. However, the classification performance can be further enhanced by Poly+RBF kernel on many datasets, e.g. Australian, German, Heart, Ionosphere, LiverDisorder, Tic-Tac-Toe, and Tokyo.

Although Poly*RBF kernel does not perform well on these datasets, there is a dataset, i.e. Ionosphere that the average percentage error of Poly*RBF kernel is lower than Poly+RBF kernel. Thus, it may be suitable for some specific problems. However, we encourage SVM with the 10-RBF kernel because it yields the lowest average percentage error at the most of datasets and makes the best average ranks.

### 4.3.2 Regression Problems

In this section, the proposed methods are evaluated on 4 regression problems. First, SVMs with the parameter settings are evaluated on single RBF and single polynomial kernels. The deviation of approximation ( $\varepsilon$ ) is fixed as 1, a regularization parameter ( $C$ ) is set as 0.1

and 1.0, the width of RBF ($\gamma$) is set as 0.0001, 0.001, 0.01, 0.1, 1, and 10, and the degree of polynomial ($d$) is set as 1, 2, 4, 6, 8, and 10. The experimental results of SVM with single RBF and single polynomial kernels are shown in Table 4-24 and Table 4-25, respectively.

Table 4-24: Average SMAPE of SVM with Parameter Setting on RBF Kernel for $\varepsilon = 1$

| **Datasets** | **C** | **Width of RBF ($\gamma$)** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **0.0001** | **0.001** | **0.01** | **0.1** | **1** | **10** |
| Auto_MPG | **0.1** | 48.5875 | 46.0065 | 32.6615 | 21.2949 | 18.0396 | 22.0299 |
| | **1.0** | 46.0027 | 32.6173 | 22.2937 | 20.3630 | **17.6418** | 21.1808 |
| CPU_ Performance | **0.1** | 103.0350 | 102.9174 | 102.5493 | 100.7347 | 93.9533 | 95.9258 |
| | **1.0** | 102.9252 | 102.5483 | 100.6928 | 192.0511 | **83.9628** | 91.2576 |
| Housing | **0.1** | 38.7003 | 37.0063 | 31.1954 | 25.1217 | 21.1597 | 25.4770 |
| | **1.0** | 37.0030 | 31.1219 | 24.9612 | 25.5889 | **18.6126** | 24.9256 |
| Servo | **0.1** | 82.7478 | 82.8238 | 81.8611 | 71.4998 | 69.9977 | 75.8987 |
| | **1.0** | 82.8236 | 81.8701 | **68.4786** | 231.5096 | 124.6145 | 75.9276 |

Table 4-25: Average SMAPE of SVM with Parameter Setting on Polynomial Kernel for $\varepsilon = 1$

| **Datasets** | **C** | **Degree of Polynomial ($d$)** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1 (Linear)** | **2** | **4** | **6** | **8** | **10** |
| Auto_MPG | **0.1** | 33.8700 | 19.1808 | **19.0836** | 20.9372 | 36.3465 | 48.8011 |
| | **1.0** | 35.6947 | 19.1514 | 21.2742 | 25.6222 | 118.3615 | 48.8891 |
| CPU_ Performance | **0.1** | 80.6270 | **73.9781** | 93.0097 | 92.0578 | 93.8800 | 93.7312 |
| | **1.0** | 322.7340 | 111.9524 | 86.6720 | 92.0777 | 93.9052 | 93.5778 |
| Housing | **0.1** | 25.9540 | 19.9023 | 20.9789 | 33.7235 | 34.8894 | 32.3293 |
| | **1.0** | 30.1677 | **19.4419** | 31.6084 | 33.7235 | 34.8894 | 32.3293 |
| Servo | **0.1** | 980.8324 | 297.8442 | 435.4942 | 318.8907 | 258.0206 | 150.9032 |
| | **1.0** | 192.3614 | **151.6029** | 208.5475 | 413.4937 | 198.1518 | 279.2676 |

These results show that if the unsuitable parameters are selected, the performance of SVM may be very bad. On the other hand, the performance of SVM may be very well if the suitable parameters are selected. Hence, the parameter setting is not a good method for the general problems. A better parameter selection method is necessary for improving the performance of classification or approximation.

Grid search is a good parameter selection method. However, the performance of grid search relates to the running time. Moreover, grid search is not suitable for selecting the parameters of combined kernel functions. Since there are a lot of possible combinations on the parameters of sub-kernels, the usage of the grid search will consume a lot of time. ES is a better choice that can search these parameters simultaneously. The performance of ES with the different objective functions are compared to the grid search on single RBF kernel function in Table 4-26. For the objective functions, we consider the mean squared error on training (*mse_obj*), the symmetric mean absolute percentage error on training (*smape_obj*), and the bound of generalization error that derived from the stability of SVM regression (*StabilityBound*).

Table 4-26: Average SMAPE on 5-Fold Cross-Validation of RBF Kernel Function

| **Datasets** | **RBF GridSearch** | **ES with Single RBF Kernel** | | |
|---|---|---|---|---|
| | | ***mse_obj*** | ***smape_obj*** | ***StabilityBound*** |
| Auto_MPG | 20.5728 (4) | 17.1952* (2) | 20.4880 (3) | **16.4799*** (1) |
| CPU_Performance | 41.7046 (3) | **39.8293** (1) | 42.0724 (4) | 41.5944 (2) |
| Housing | 19.4757 (4) | 18.8712 (3) | 18.7848 (2) | **17.4683*** (1) |
| Servo | 78.0434 (4) | 54.2221 (2) | 59.3868 (3) | **34.8444** (1) |
| **Average SMAPE** | 39.9491 | 32.5294 | 35.1830 | **27.5967** |
| **Average Rank** | 3.7500 | 2.0000 | 3.0000 | **1.2500** |

* Statistically significant at the level of 0.05 when compared to RBF Grid Search.

The results show that the performance of *StabilityBound* outperforms the other objective functions and the grid search. Moreover, the average SMAPE of *StabilityBound* is significantly lower than the grid search at the confident level of 95% on two datasets, i.e. Auto_MPG and Housing. Whereas *mse_obj* yields the best average SMAPE on CPU_Performance dataset, the average SMAPE of *smape_obj* is not better than the other objective functions. By Friedman test, the average ranks of these 4 algorithms are significantly different from the mean rank $R_j$ = 2.5 at a significance level of 0.05. The pairwise differences on average ranks are shown in Table 4-27. The performance of *StabilityBound* is significantly better than grid search but it is not sufficient to conclude about the other objective functions.

Table 4-27: Pairwise Differences on Average Ranks of RBF Grid Search and ES with the Different Objective Functions on Single RBF Kernel on Regression Problems

| **CD = 1.9426** | **Algorithms** | **RBF Grid Search** | ***mse_obj*** | ***smape_obj*** | ***Stability Bound*** |
|---|---|---|---|---|---|
| **Algorithms** | **Average Rank** | **3.7500** | **2.0000** | **3.0000** | **1.2500** |
| **RBF Grid Search** | **3.7500** | 0.0000 | 1.7500 | 0.7500 | **2.5000***|
| ***mse_obj*** | **2.0000** | -- | 0.0000 | -1.0000 | 0.7500 |
| ***smape_obj*** | **3.0000** | -- | -- | 0.0000 | 1.7500 |
| ***Stability Bound*** | **1.2500** | -- | -- | -- | 0.0000 |

* Significantly Different at the level of 0.10.

Furthermore, the grid search may make SVM overfit to training data. The average SMAPE on training of the grid search is lower than the other methods. The average SMAPE on training is shown in Table 4-28 and Figure 4-6.

Table 4-28: Average SMAPE on Training of RBF Kernel Function

| **Datasets** | **RBF GridSearch** | **ES with Single RBF Kernel** | | |
|---|---|---|---|---|
| | | ***mse_obj*** | ***smape_obj*** | ***StabilityBound*** |
| Auto_MPG | **1.8184** | 8.8942 | 2.1310 | 7.7204 |
| CPU_Performance | **17.0925** | 35.1444 | 17.6679 | 24.0513 |
| Housing | **1.2762** | 2.5925 | 2.4260 | 4.1805 |
| Servo | **2.1264** | 53.2094 | 16.5329 | 7.1311 |
| **Average SMAPE** | **5.5784** | 24.9601 | 9.6895 | 10.7708 |

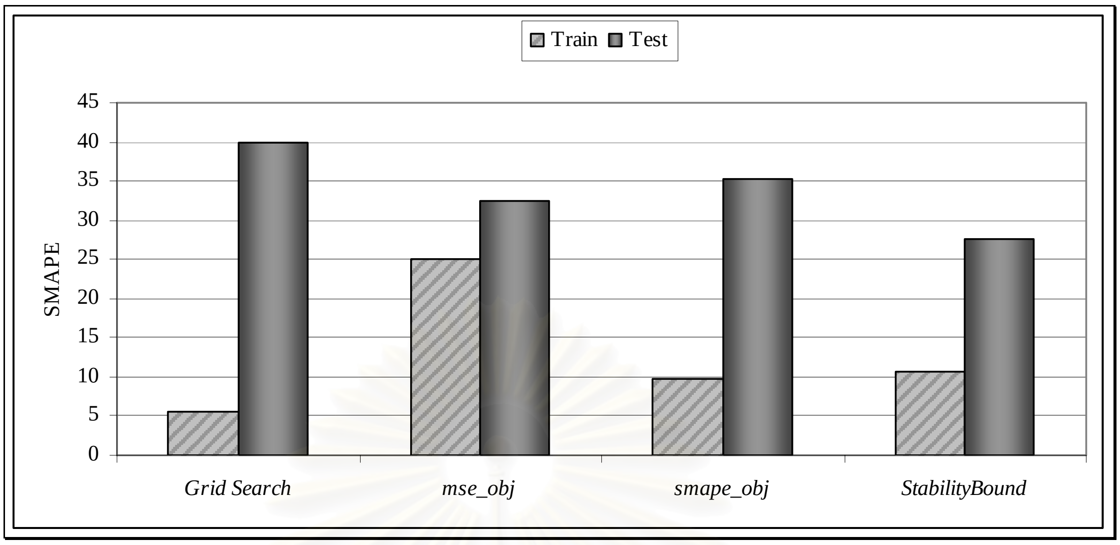


Figure 4-6: Average SMAPE on Training and Testing of Single RBF Kernel Function

However, with the help of ES, the proposed method can determine the high quality parameters in a more convenient way. Then, SVM with 10-RBF kernel is evaluated; its parameters are selected by ES with the different objective functions. The experimental results are compared to the *StabilityBound* on the single RBF kernel in Table 4-29. Moreover, the pairwise differences on the average ranks are shown in Table 4-30. The results confirm that the stability on bounded SVR is a suitable objective function.

Table 4-29: Average SMAPE on 5-Fold Cross-Validation of 10-RBF Kernel Function

| **Datasets** | **ES with RBF Kernel and *StabilityBound*** | **ES with 10-RBF Kernel** | | |
|---|---|---|---|---|
| | | ***mse_obj*** | ***smape_obj*** | ***StabilityBound*** |
| Auto_MPG | 16.4799<br>(2) | 18.1050<br>(4) | 17.8098<br>(3) | **15.7513***<br>(1) |
| CPU_Performance | 41.5944<br>(3) | 42.3626<br>(4) | 40.4926<br>(2) | **38.6108***<br>(1) |
| Housing | 17.4683<br>(3) | 16.9928<br>(2) | 18.0143<br>(4) | **15.8854**<br>(1) |
| Servo | 34.8444<br>(2) | 55.3948<br>(4) | 45.3529<br>(3) | **28.3842***<br>(1) |
| **Average SMAPE** | 27.5967 | 33.2138 | 30.4174 | **24.6579** |
| **Average Rank** | 2.5000 | 3.5000 | 3.0000 | **1.0000** |

* Statistically significant at the level of 0.05 when compared to RBF *StabilityBound*.

Table 4-30: Pairwise Differences on Average Ranks of Single RBF Kernel and 10-RBF Kernel with Different Objective Functions on Regression Problems

| CD = 1.9426 | | Algorithms | RBF | 10-RBF | | |
|---|---|---|---|---|---|---|
| | | | *Stability Bound* | *mse_obj* | *smape_obj* | *Stability Bound* |
| **Algorithms** | | **Average Rank** | **2.5000** | **3.5000** | **3.0000** | **1.0000** |
| **RBF** | ***Stability Bound*** | **2.5000** | 0.0000 | -1.0000 | -0.5000 | 1.5000 |
| **10-RBF** | ***mse_obj*** | **3.5000** | -- | 0.0000 | 0.5000 | **2.5000*** |
| | ***smape_obj*** | **3.0000** | -- | -- | 0.0000 | **2.0000*** |
| | ***Stability Bound*** | **1.0000** | -- | -- | -- | 0.0000 |

* Significantly Different at the level of 0.10.

From these experimental results, these average ranks are significantly different from the mean rank $R_j = 2.5$. The average SMAPE of *StabilityBound* is significantly better than those of *mse_obj* and *smape_obj*. Similar to the previous experiment, the lowest average SMAPE on training may not yield the best result on testing. The average SMAPE on training of the 10-RBF kernel are illustrated in Table 4-31. Moreover, the chart of the average SMAPE on 5-fold cross-validation measured on training and test data is shown in Figure 4-7.

Table 4-31: Average SMAPE on Training of 10-RBF Kernel Function

| **Datasets** | **RBF *StabilityBound*** | **ES with 10-RBF Kernel** | | |
|---|---|---|---|---|
| | | ***mse_obj*** | ***smape_obj*** | ***StabilityBound*** |
| Auto_MPG | **1.8184** | 5.1195 | 3.8058 | 10.3504 |
| CPU_Performance | **17.0925** | 28.1559 | 20.2861 | 24.9695 |
| Housing | **1.2762** | 4.0304 | 5.4541 | 6.5383 |
| Servo | **2.1264** | 31.1625 | 10.4524 | 9.9108 |
| **Average SMAPE** | **5.5784** | 17.1171 | 9.9996 | 12.9422 |

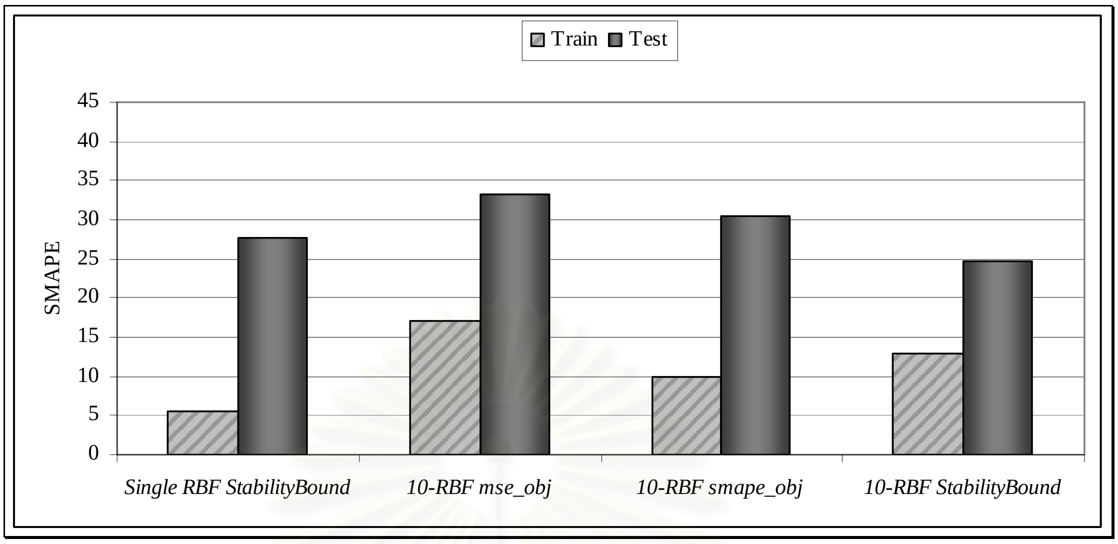


Figure 4-7: Average SMAPE on Training and Testing of 10-RBF Kernel Function

In this chart, the average SMAPE of *StabilityBound* on 10-RBF kernel was compared to those of *mse_obj* and *smape_obj* on the 10-RBF kernel and *StabilityBound* on single RBF kernel. *StabilityBound* did not yield the lowest average SMAPE on training data. However, the average SMAPE of *StabilityBound* on test data was lower than the other techniques. This means that *StabilityBound* did not overfit training data, and it was a good objective function for the adaptive multi-scale RBF kernel, while *mse_obj* and *smape_obj* may overfit training data.

Then, the results of the other combined kernels, i.e. the multi-degree polynomial kernel, the non-negative linear combination of polynomial and RBF kernels, and the multiplication of polynomial and RBF kernels, are illustrated in Table 4-32. The parameters of these kernel functions are selected by ES that uses the *StabilityBound* as an objective function. The results are compared to the SVMs with single polynomial and single RBF kernels that their parameters are selected by the grid search.

Furthermore, the average ranks of the algorithms are calculated. Pairwise differences on the average rank of these algorithms are shown in Table 4-33. The performance of 10-RBF kernel is significantly better than those of Poly*RBF kernel and grid search on both RBF and polynomial kernels. Moreover, the performance of Poly+RBF kernel is significantly better than the performances of grid search on RBF kernel and Poly*RBF kernel.

Table 4-32: Average SMAPE on 5-Folds Cross-Validation of SVM with Different Kernel Functions when using *StabilityBound* as the Objective Function in Evolutionary Process

| **Datasets** | **Grid Search** | | **ES** | | | |
|---|---|---|---|---|---|---|
| | **RBF** | **Poly** | **10-Poly** | **Poly+RBF** | **Poly*RBF** | **10-RBF** |
| Auto_MPG | 20.5728 (4) | 33.2921 (6) | 18.5709 (3) | 16.8403 (2) | 24.1689 (5) | **15.7513** (1) |
| CPU_ Performance | 41.7046 (4) | 70.2761 (6) | 47.1344 (5) | 38.8105 (2) | 39.7933 (3) | **38.6108** (1) |
| Housing | 19.4757 (5) | 28.6761 (4) | 18.9636 (3) | 15.9673 (2) | 19.5409 (6) | **15.8854** (1) |
| Servo | 78.0434 (5) | 72.1700 (4) | 67.9599 (3) | 59.7369 (2) | 94.6604 (6) | **28.3842** (1) |
| **Average SMAPE** | 39.9491 | 51.1036 | 38.1572 | 32.8388 | 44.5409 | **24.6579** |
| **Average Rank** | 4.5000 | 5.0000 | 3.5000 | 2.0000 | 5.0000 | **1.0000** |

Table 4-33: Pairwise Differences on Average Ranks of Grid Search and ES with the Different Combined Kernel Functions on Regression Problems

| **CD = 3.0770** | | **Algorithms** | **Grid Search** | | **ES** | | | |
|---|---|---|---|---|---|---|---|---|
| | | | **RBF** | **Poly** | **10-Poly** | **Poly + RBF** | **Poly * RBF** | **10-RBF** |
| **Algorithms** | | **Average Rank** | **4.5000** | **5.0000** | **3.5000** | **2.0000** | **5.0000** | **1.0000** |
| **Grid Search** | **RBF** | **4.5000** | 0.0000 | -0.5000 | 1.0000 | 2.5000 | -0.5000 | **3.5000***|
| | **Poly** | **5.0000** | -- | 0.0000 | 1.5000 | **3.0000**** | 0.0000 | **4.0000*** |
| **ES** | **10-Poly** | **3.5000** | -- | -- | 0.0000 | 1.5000 | -1.5000 | 2.5000 |
| | **Poly + RBF** | **2.0000** | -- | -- | -- | 0.0000 | **-3.0000**** | 1.0000 |
| | **Poly * RBF** | **5.0000** | -- | -- | -- | -- | 0.0000 | **4.0000*** |
| | **10-RBF** | **1.0000** | -- | -- | -- | -- | -- | 0.0000 |

* Significantly Different at the level of 0.10.

** This value is very close to the critical difference.

These results are very similar to the classification problems in Section 4.3.1. The performance of 10-Poly kernel is better than single polynomial kernel at all datasets. Moreover, these results can be enhanced by a linear combination of polynomial and RBF kernels. Although the performance of multiplication of polynomial and RBF kernels does not perform well, its performance is better than single polynomial kernel on two datasets, i.e. Auto_MPG and CPU_Performance. However, the 10-RBF kernel is still the best kernel function in our experiments. The average SMAPE of SVM with the 10-RBF kernel that uses the *StabilityBound* as the objective function in evolutionary process is lower than the average SMAPE of the other objective functions and kernels.

# CHAPTER V

# REAL WORLD PROBLEMS

In this chapter, some real world problems, i.e. sentiment classification and handwritten recognition, are considered. The sentiment classification is a binary classification problem, while the handwritten recognition is a multi-class problem. The proposed methods in Chapter 3 are applied on these problems. The problem description and the experimental results are described in the following sections.

## 5.1 Sentiment Classification

There are many ways to categorize documents, for example, by subject, genre, or the sentiment expressed in the documents. Sentiment classification of reviews has been the focus of the recent research studies. It has been applied on different domains such as movie reviews, product reviews, and customer feedback reviews [67] in order to categorize the documents. The most basic task in sentiment classification is to classify a document into positive or negative sentiment. Since sentiment is expressed in many different ways, it is hard to manually create all the classification rules and therefore researchers have attempted to apply machine learning techniques on this task. This research is also devoted to the development of machine learning based techniques for sentiment classification.

SVM is a learning technique that performs well on sentiment classification. The performance of SVM classifier depends on the used kernel function and its parameters. Hence, if the suitable kernel is chosen, the performance of sentiment classification should be improved. These lead to the idea of applying the adaptive combined kernel functions for the sentiment classification problem. The sentiment classification and its related work are briefly reviewed in Section 5.1.1. The adaptive combined kernel functions and the normalization of kernels are briefly described in Section 5.1.2. In Section 5.1.3, our approaches are applied on the sentiment classification problem, and their results are reported. These results are discussed in Section 5.1.4.

### 5.1.1 Preliminary and Related Work

Research of sentiment classification was initiated in 1997 [67]. It is the main task of opinion mining, and most of work focuses on determining the sentiment orientations of documents, sentences, and words. In document level sentiment analysis, documents are classified into positive and negative according to the overall sentiment expressed in them. This simple classification task has many potential applications including a movie recommendation system, market research form blog text, and an email classification system that alerts users when the system identifies a message with negative sentiment.

There are two approaches to classify the sentiment. The first approach is to count positive and negative terms in a review, where the review is positive if it contains more positive than negative terms, and negative if there are more negative terms [67]. The difficulty of sentiment classification is the context-dependency of the sentiments of linguistic expressions. For example, negation words such as “not” or “never” shift the sentiment. A positive statement becomes negative when it is subcategorized by a verb “doubt”. Although we could use n-grams (continuous n words) as features in order to handle such shifts, dependency between two words with a long distance cannot be captured by n-grams.

The second approach uses machine learning to determine the sentiment of the reviews [67]. Pang et al. [68] have used learning algorithms such as Naïve Bays, maximum entropy, and SVM to classify reviews. Their work focused on the features indicating the presence of words or n-grams. Dave et al. [69] used machine learning methods for review classification and reported that the bigram features yielded the best accuracy. This result supports that the sentiment has strong dependency on the context. Li and Sun [70] applied four machine learning methods on sentiment classification of Chinese reviews. They investigated the factors which affect performance. Then, SVM, Naïve Bayes, maximum entropy, and artificial neural network (ANN) were employed on customer reviews. The results showed that the SVM classifier produces the best results under all of their text representation schemes.

Hence, SVM classifiers that use unigrams (single words) as features are trained in this research. Instead of using n-grams, we utilize higher-degree kernel functions, which can automatically take into account the conjunctions of features. However, there is another difficulty that it is unknown which kernel function is suitable for this task. In spite of our intuition that feature combinations will capture the context-dependency of sentiment, some researchers have reported that higher-degree kernels only degraded the classification performance [71, 72]. Meanwhile, there is a report that higher-degree kernels did improve the classification performance [73]. Therefore, we decided to use many types of kernel functions in the form of the non-negative linear combination. The performance of sentiment classification should be enhanced by SVM with the adaptive combined kernel functions.

### 5.1.2 Methodology

The adaptive combined kernel functions are used for the sentiment classification problem. SVM with the non-negative linear combination of multiple kernel functions are applied on product reviews to determine whether a review is positive or negative. Then ES is applied for adjusting the parameters of SVM and the combined kernel functions. Moreover, the stability of soft margin SVM is the objective function in evolutionary process in order to avoid the overfitting problem.

#### 5.1.2.1 Combined Kernel Functions

For sentiment classification, we take an interest in two kinds of kernel functions, i.e. polynomial and radial basis function (RBF) kernels that are the inner-product-based and distance-based kernels, respectively. These kernel functions are illustrated in the following forms.

$$\text{Polynomial:} \quad K(x,x') = \left(1 + \langle x \cdot x' \rangle\right)^d \tag{142}$$

$$\text{Gaussian RBF:} \quad K(x,x') = \exp\left(-\gamma \left\|x - x'\right\|^2\right) \tag{143}$$

In order to obtain a better result, SVM with the non-negative linear combination of multiple kernel functions is applied on sentiment classification. The analytic expression of this kernel is the following:

$$\mathrm{K}(x,x') = \sum_{i=1}^{n} a_i K_i(x,x') \, , \tag{144}$$

where $n$ is the number of sub-kernels, $a_i \geq 0$ for $i = 1,\ldots,n$ are the arbitrary non-negative weighting constants, and $K_i(x,x')$ for $i = 1,\ldots,n$ are the sub-kernels, each of which is the polynomial kernel at degree $i$ or the RBF kernel at width $\gamma_i$.

With these two kinds of kernel functions, there are three possible non-negative linear combinations of multiple kernels, i.e. (1) the non-negative linear combination of multiple polynomial kernels at different degree, (2) the non-negative linear combination of multiple RBF kernels at different scale, and (3) the non-negative linear combination of both polynomial and RBF kernels with different parameters. These kernels are more flexible as they have more adjustable parameters, and their expressions are shown by the following equations:

$$\mathrm{K}(x,x') = \sum_{i=1}^{n} a_i K_{Poly}(x,x',i) \tag{145}$$

$$\mathrm{K}(x,x') = \sum_{i=1}^{n} a_i K_{RBF}(x,x',\gamma_i) \tag{146}$$

$$\mathrm{K}(x,x') = \sum_{i=1}^{n/2} a_i K_{Poly}(x,x',i) + \sum_{i=n/2+1}^{n} a_i K_{RBF}(x,x',\gamma_i) \text{ ; where } n \text{ is even.} \tag{147}$$

**5.1.2.2 Normalization**

In our experiments, all parameters (the weights of combination, the widths of RBF kernel, and the regularization parameter of SVM) are determined by an evolutionary algorithm. Although these parameters are bounded in evolutionary process, the numerical value of these combined kernel functions may be very large when many sub-kernels are combined. Hence, the normalization is considered to apply on these combined kernel functions. Normalization in feature space is not applied directly on the input vector, but it can be seen as a kernel interpretation of the preprocessing [74]. This normalization redefines a new kernel function $\tilde{\mathrm{K}}(x_1,x_2)$ of SVM. The non-negative linear combination kernels are normalized by

$$\tilde{\mathrm{K}}(x,x') = \frac{\mathrm{K}(x,x')}{\sqrt{\mathrm{K}(x,x')\mathrm{K}(x,x')}} . \tag{148}$$

This normalized kernel places the data on a portion of the unit hypersphere in the feature space [74]. Obviously, the equation $\tilde{\mathrm{K}}(x,x') = 1$ holds true.

**5.1.2.3 Parameters Adjustment**

The weights of sub-kernels, the width of RBF kernels, and the regularization parameter of SVM are the adjustable parameters of the learning process. The (5+10)-ES is applied for adjusting these parameters. Let $\vec{v}$ be the non-negative real-valued vector of the parameters. For the non-negative linear combination of multiple polynomial kernels, the vector $\vec{v}$ has $n+1$ dimensions and it is represented in the form:

$$\vec{v} = (C,\ a_1,\ a_2,\ \ldots,\ a_n), \tag{149}$$

where $C$ is the regularization parameter, $a_i$ for $i = 1,\ldots,n$ are the non-negative weights of sub-kernels, and $n$ is the number of terms of sub-kernels.

For the non-negative linear combination of multiple RBF kernels, the widths of RBF kernels ($\gamma_i$) will be added into the vector $\vec{v}$. The vector $\vec{v}$ is represented by

$$\vec{v} = (C,\ a_1,\ a_2,\ \ldots,\ a_n,\ \gamma_1,\ \gamma_2,\ \ldots,\ \gamma_n). \tag{150}$$

Also, the vector $\vec{v}$ for non-negative linear combination of both polynomial and RBF kernels is represented by

$$\vec{v} = (C,\ a_1,\ a_2,\ \ldots,\ a_n,\ \gamma_{n/2+1},\ \gamma_{n/2+2},\ \ldots,\ \gamma_n). \tag{151}$$

This vector will be investigated by the (5+10)-ES. This algorithm uses 5 solutions to produce 10 new solutions by a recombination method that is described in Chapter 3. These new solutions are mutated and evaluated, and only the 5 fittest solutions are selected from 5+10 solutions to be the parents in the next generation. ES terminates after a fixed number of generations have been produced and evaluated.

One of the most important and difficult parts of the evolutionary algorithm is how to define the objective function. Although the training error can be used as the objective function in the evolutionary processes, this function may cause the overfit to training data. Therefore, this research proposes to use the bound of generalization error that is derived from the assumption of stability as the objective function for the sentiment classification problem.

$$fitness(\vec{v}) = R_{emp} + \frac{\kappa^2}{\lambda m} + \left(1 + \frac{2\kappa^2}{\lambda}\right)\sqrt{\frac{\ln(1/\delta)}{2m}}\ , \tag{152}$$

where $R_{emp}$ is the empirical error, $K(\cdot)$ is a bounded kernel that is $K(x_i, x_j) \le \kappa^2$, $\lambda$ is the regularization parameter of SVM ($\lambda = 1/C$), $m$ is the sample size, and $1-\delta$ is the probability of this bound. The bound of kernel function ($\kappa^2$) can be estimated when the parameters of a kernel function are assigned for each individual vector ($\vec{v}$). A set of suitable parameters should provide a lower bound of risk.

### 5.1.3 Experimental Results

We used a dataset of product reviews, which was provided by Bing Liu[1] [75]. This dataset contains sentences used in product reviews collected from the internet and assigned with a sentiment tag: positive or negative. The dataset contains 1,700 sentiment sentences: 1,067 positive and 633 negative sentences. The SVM classifiers were trained by using unigrams (single words) as features. Methods were evaluated by 5-folds cross-validation. The single kernel functions, i.e. linear kernels, polynomial kernels at different degree, and RBF kernels at different scale were used as the baselines. The average accuracy values on

[1] The dataset is available at http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html.

training and testing of SVM with single polynomial and single RBF kernels when we varied the parameters are shown by the graphs in Figure 5-1 and Figure 5-2, respectively.

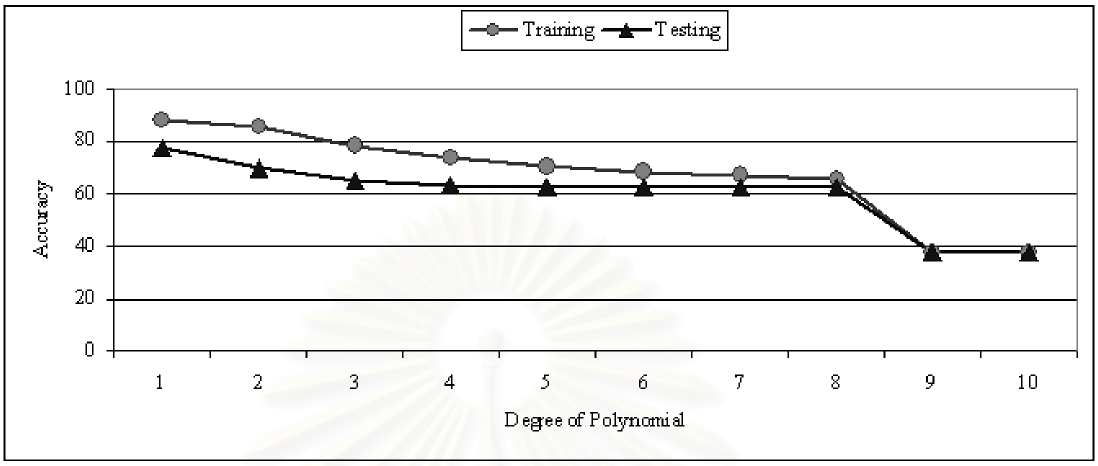


Figure 5-1: The Average Accuracy on Polynomial Kernels at Different Degree

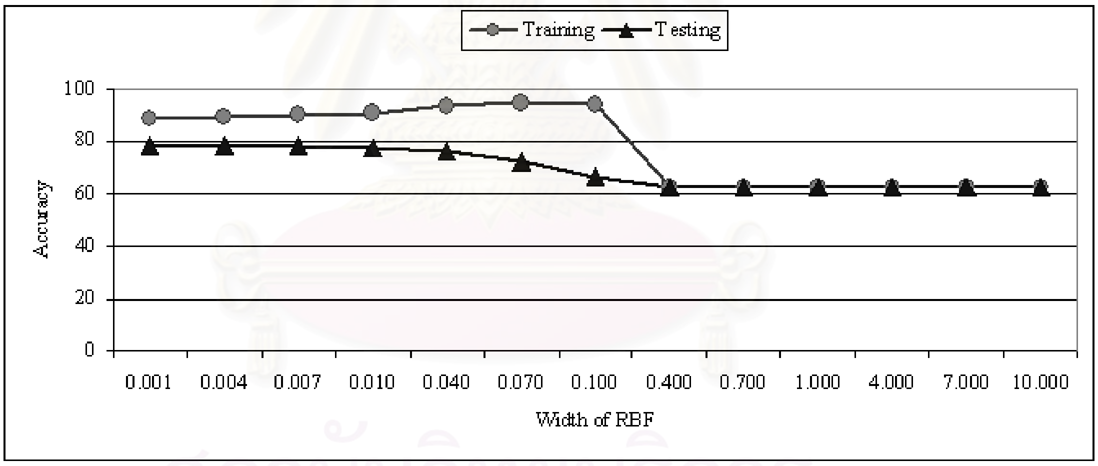


Figure 5-2: The Average Accuracy on RBF Kernels at Different Scale

From these graphs, we found that the average test accuracy of sentiment classification decreased when the degree of polynomial or the width of RBF was increased. The linear kernel and the polynomial at degree 2 yielded the results that were better than the other degrees, and a good average training accuracy of RBF kernel occurred when the width of RBF was 0.07. These results were compared with the proposed method.

The (5+10)-ES were used to find the optimal parameters of the combined kernel. The value of $\tau$ in evaluation process of these experiments was set to 1.0. The number of terms of sub-kernel was fixed as 10. The weights of combination ($a_i$), the widths of RBF kernels ($\gamma_i$), and the regularization parameter ($C$) were real numbers between 0.0 and 10.0. These parameters were inspected within 1,000 generations of ES. The average accuracies of SVM with the adaptive combined kernels are compared in Table 5-1.

Table 5-1: Average Accuracy of SVM on Sentiment Classification

| **Kernel Function** | **Training Accuracy** | **Test Accuracy** |
|---|---|---|
| Linear Kernel | 88.3533 | 77.7093 |
| Polynomial Kernel at Degree 2 | 85.7943 | 69.7711 |
| RBF Kernel at Width 0.07 | 94.7352 | 72.5348 |
| Combination of Multiple Polynomial Kernels (Proposed Kernel 1) | 98.0881 | **79.1828** |
| Combination of Multiple RBF Kernels (Proposed Kernel 2) | 95.0294 | 73.1805 |
| Combination of Multiple Polynomial and RBF Kernels (Proposed Kernel 3) | **98.7355** | 78.7122 |

The experimental results showed that the average accuracy on sentiment classification can be enhanced by the combined kernel functions. The non-negative linear combination of multiple polynomial kernels yielded the best result on testing. Although the linear combination of multiple RBF kernels did not yield the best result, its accuracy was better than single RBF kernel. For the combination of both polynomial and RBF kernels, it yielded the best training accuracy, but its accuracy on testing was lower than the combination of multiple polynomial kernels. This means that although we tried to avoid the overfitting problem by using the stability objective function in evolutionary process, the overfiitng problem still can be occurred by a more flexible kernel.

## 5.2 Handwritten Recognition

The handwritten recognition is the ability of the computer to recognize the handwritten of humans from sources such as paper documents, photos, touch screens, or other devices. In general, the handwritten recognition can be divided into on-line recognition and off-line recognition. In this research, SVMs with the proposed methods will be applied to the off-line handwritten recognition. The aim of this section is to illustrate that the proposed methods can be applied to the multi-class problems.

### 5.2.1 Preliminary and Related Works

The problem of handwritten recognition has been an on-going research problem. It has been gaining more interest due to the increasing popularity of hand-held computers, digital notebooks, and advanced cellular phones [76]. In off-line handwriting recognition, the letters in an image are automatically converted into the letter codes which are usable within computer and text-processing applications. This technology is successfully used by businesses which process lots of handwritten documents, like bank checks and insurance companies.

The most prominent problem in the handwritten recognition is the vast variation in personal writing, as different people have different handwriting styles. Nevertheless, limiting the range of input can allow recognition to improve. Various methods were proposed to solve this problem such as gradient based learning [77], moving window classifier [78], and neural network [79]. For SVM, it is applied in many research studies of handwritten recognition. Dong, X.J., et al. [80] applied SVM for handwritten Chinese character recognition. Moreover, the SVM was used for online handwritten in [81], and it gave a better recognition result compared to the system based on a hybrid neural network and a hidden Markov model.

SVM is the binary classifier for two-class data. However, the multi-class classification problems such as letter recognition can be solved by voting schema methods based on a combination of many binary classifiers. One possible approach to solve a $k$-class problem is to consider the problem as a collection of $k$ binary classification problems. $k$-classifiers can be constructed, one for each class. The $k^{th}$ classifier constructs a hyperplane between class $k$ and the $k$-1 other classes. A new sample will be classified by these $k$ classifiers. The prediction class of this sample is a classifier that yields the longest distance between this sample and its decision hyperplane or the maximum value of decision function. This schema is commonly called ***one against the rest*** and shown in Figure 5-3.

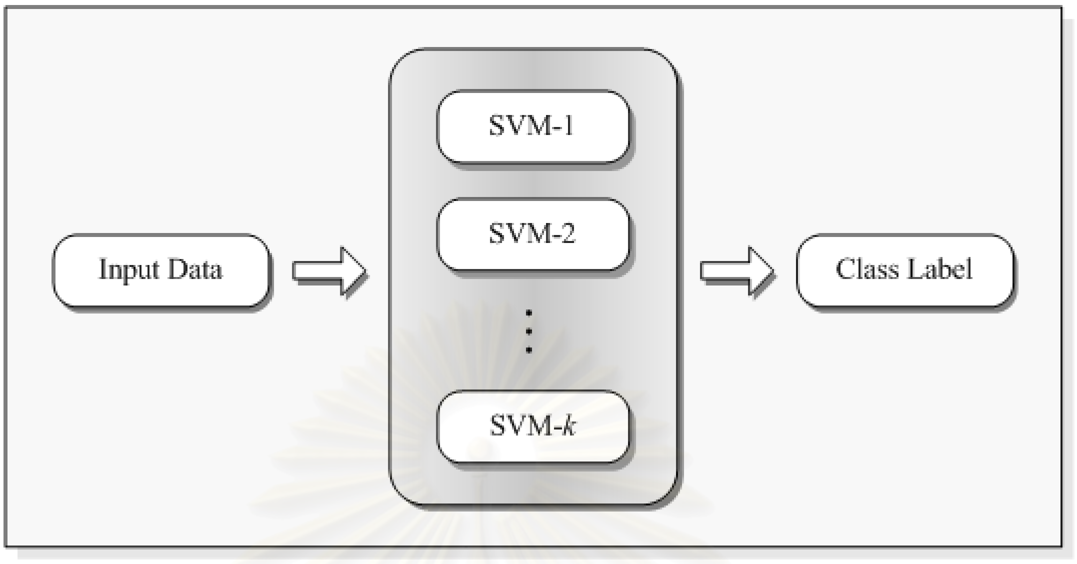


Figure 5-3: SVM for Multi-Class Problems

### 5.2.2 Methodology

SVM is applied to solve the English alphabet handwritten recognition in this section. One against the rest schema is implemented via creating 26 classifiers, each classifier for each alphabet. Each classifier is trained by all training data; the class of alphabets under consideration is 1 whereas the class of the other alphabets is -1. The evolutionary strategy is applied for adjusting the parameters of SVM and its kernel function for each classifier. This process is shows in Figure 5-4.

In evolutionary process, the non-negative real-valued vector $\vec{v}$ that has $2n$ dimensions is represented in the form:

$$\vec{v} = (C, \gamma_0, a_1, \gamma_1, a_2, \gamma_2, \ldots, a_{n-1}, \gamma_{n-1}), \tag{153}$$

where $C$ is the regularization parameter of SVM, $\gamma_i$ for $i = 0, \ldots, n-1$ are the widths of RBFs, and $a_i$ for $i = 1, \ldots, n-1$ are the weights of RBFs. The (5+10)-ES is applied to adjust these parameters. For evaluating the parameters of SVM, the stability bound of generalization error of SVM is used as the objective function. These parameters will be investigated within 1,000 generations.

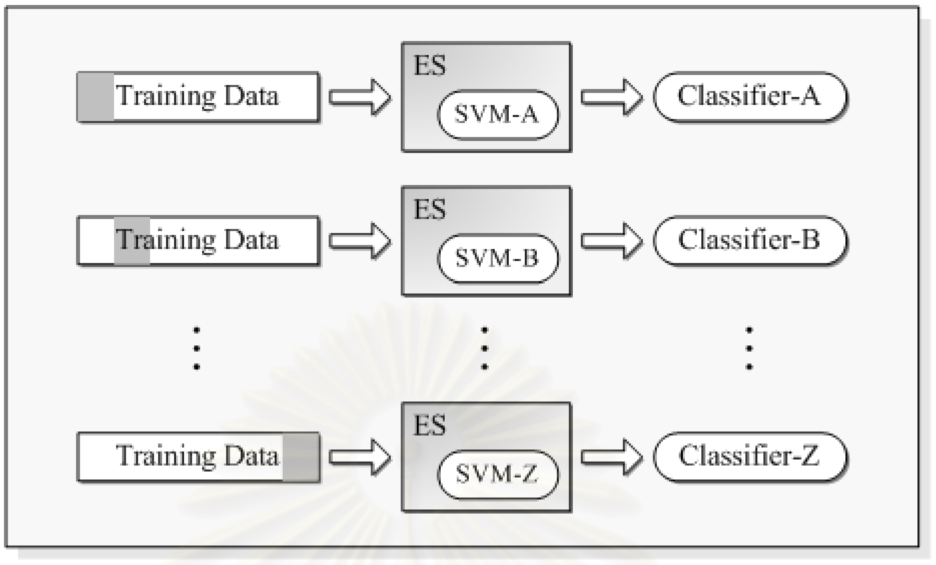


Figure 5-4: Proposed Method for Multi-Class Problems

### 5.2.3 Experimental Results

The proposed kernels have been tested on the letter recognition from the UCI machine learning repository [7]. A multi-class recognition system has been implemented with 26 different models, one for each class. Each of these models is a binary classifier that matches a specific class against the other 25 classes. The set of 20,000 English alphabets with 20 fonts of the 26 capital letters are used for learning. Each instance contains 16 attributes (statistical moments and edge counts). Examples of the letter images are shown in Figure 5-5. All data are used for training and testing.

The experimental results on the letter recognition task are shown in Table 5-2. The results show that the 3-RBF kernel gives the best recognition rate. Also, the multiplication of polynomial and RBF kernels performs better than the polynomial and the RBF kernels with the same parameters.

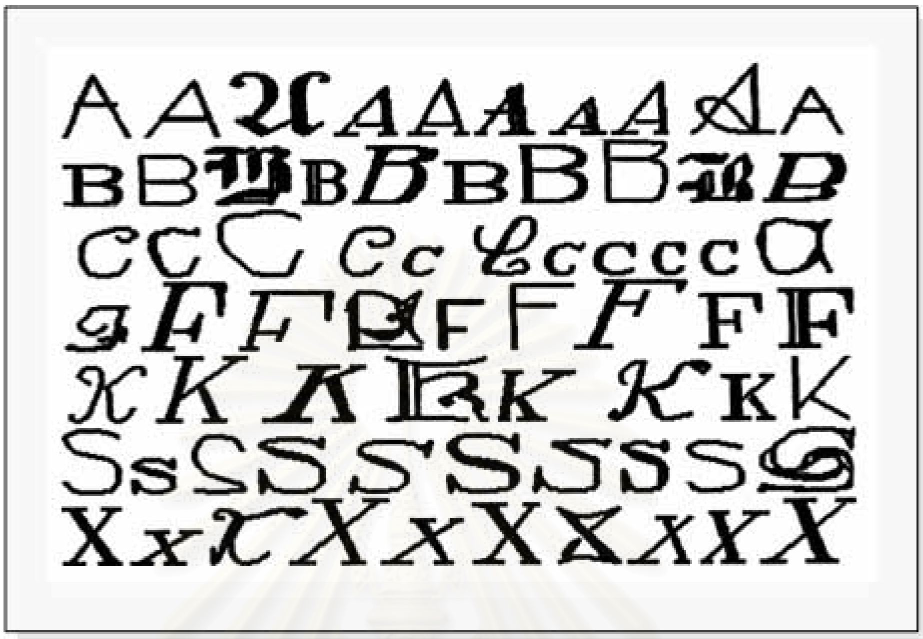


Figure 5-5: Sample Letter Images

Table 5-2: Accuracies on Letter Recognition Task

| **Kernel Functions** | **No. Corrected Examples** | **Accuracy (%)** |
|---|---|---|
| Polynomial | 14966 | 74.83 |
| RBF | 15101 | 75.51 |
| 2-RBF | 18083 | 90.42 |
| 3-RBF | 19664 | **98.32** |
| Polynomial + RBF | 14966 | 74.83 |
| Polynomial * RBF | 16414 | 82.07 |

### 5.2.4 Other Experimental Results on Multi-Class Problems

Moreover, the proposed method has been tested on two multi-class problems from the UCI machine learning repository [7]. In both problems, there are 3 classes of data. The proposed method is evaluated by 5-folds cross-validation. The experimental results are shown in Table 5-3. These results show that the accuracies of the proposed method are better than those of the RBF kernel using the grid search on both problems.

Table 5-3: Average Accuracies on Multi-Class Problems

| **Datasets** | **Number of Attributes** | **Number of Examples** | **Average Accuracy** | |
|---|---|---|---|---|
| | | | **RBF Grid Search** | **10-RBF ES with Stability Objective Function** |
| BalanceScale | 4 | 625 | 85.92 | **88.16** |
| Waveform | 21 | 5000 | 33.92 | **46.84** |

# CHAPTER VI

# EVOLVING KERNEL TREE

This chapter proposes a combination technique, called GPES, which combines Genetic Programming (GP) and Evolutionary Strategies (ES) to evolve a hybrid kernel function for an SVM classifier. The hybrid kernel functions are represented as trees that have the adjustable parameters. The experimental results are compared with a standard SVM classifier using the polynomial and radial basis function kernels with various parameter settings on benchmark datasets.

## 6.1 Motivation

As described previously, the complexity of the separating hyperplane of SVM depends on the nature and the properties of the used kernel function. Although the combined kernel functions, which have been proposed in the previous chapter, are more flexible than conventional kernel functions; the forms of those kernel functions are fixed as the non-negative linear combination or the multiplication of kernel functions. There are the other combined kernel functions that use the addition and the multiplication operators. Therefore, the genetic programming (GP) is proposed to improve the classification performance by creating the hybrid Mercer's kernel functions for each task.

GP is an application of genetic algorithm (GA) approach to derive mathematical equations, logical rules, or program functions automatically [82]. As in a conventional GA, the solution to a problem is represented by a string of parameters. GP usually uses a tree structure, the leaves of which represent input variables or numerical constants. Their values are passed to nodes, which perform some numerical or program operation before passing on the result further towards the root of the tree [83]. This concept of GP is used to create the hybrid kernel functions, which are represented as the tree structure. The basic kernel functions with different parameters may be weighted and combined. These new hybrid kernel functions also correspond to the Mercer's theorem and they are flexible to the given tasks.

The parameters of sub-kernels and their weights are adjustable parameters. In general, these parameters are usually determined by a grid search. However, for hybrid kernel functions, the grid search will consumes a lot of time because there are more adjustable parameters. Hence, ES is a better alternative for adjusting many parameters. Although the representation of ES is different from the GP, both GP and ES are the evolutionary algorithms. Therefore, we propose the GPES algorithm that uses GP for creating a hybrid kernel function which satisfies the Mercer’s theorem. Then, the parameters and weights of sub-kernels are adjusted by ES.

Moreover, the objective function is an important part in evolutionary algorithms. We carefully design the objective function in order to avoid the overfitting problem that may occur when the complex kernel functions are used. This proposed algorithm is tested on some classification datasets; the results show that GPES effectively searches the good hybrid kernel functions and their parameters when we use a suitable objective function. However, the running time is a disadvantage of the algorithm that which should be improved.

## 6.2 GPES

One of the most important design choices for SVMs is the kernel function and its parameters, which implicitly define the structure of the high dimensional feature space where a maximal margin hyperplane will be found [17]. The chosen kernels have influence to the classification accuracy. A complex kernel may be needed for a complex problem. Hence, we propose to use the hybrid kernel functions that combine several Mercer’s kernels with different parameters. These hybrid kernel functions should correspond to the Mercer’s theorem.

The corollary 3-1 is used to create these hybrid kernel functions. Therefore, there are two operators for combining kernel functions that are the addition of kernel functions and the multiplication of kernel functions. For scalar multiplication, it will appear in terms of a weight of each sub-kernel. According to this corollary, we can construct hybrid Mercer’s kernels from the conventional Mercer’s kernels. The Polynomial and RBF kernels are well-known conventional Mercer’s kernels, which will be used as the basic kernel functions in this research.

### 6.2.1 GPES Algorithm

The approach presented here combines the two techniques of GP and ES to evolve a hybrid kernel for SVM. The goal is to eliminate the need for testing various kernels and their parameter settings. A hybrid kernel is represented as tree of sub-kernels and operators. For each sub-kernel, there are two parameters, i.e. the weight and the degree of polynomial kernel or the width of an RBF kernel, which can be adjusted. An overview of the ($\mu+\lambda$)-GPES is shown below.

**Algorithm 6-1**: ($\mu + \lambda$)-GPES Algorithm

Step 1. Create a random population of $\mu$ hybrid kernels (parents), represented as trees.

Step 2. Evaluate the fitness of each individual by building an SVM from the kernel tree and test it on the training data.

Step 3. Apply reproduction operators to create $\lambda$ new hybrid kernels (offsprings).

Step 4. Randomly mutate nodes of the trees.

Step 5. Randomly mutate the parameters of leaf nodes.

Step 6. Evaluate and keep the $\mu$ fittest individuals.

Step 7. Go to step 3 unless an acceptable solution has been found or a fixed number of generations has been produced and evaluated.

Step 8. Build final SVM using the fittest kernel tree found.

This research proposes to use (5+5)-GPES because the population size is not too large and the optimal solutions can be found in a limited time. This algorithm starts with the 0th generation ($t = 0$) that creates 5 solutions and 2 standard deviation vectors ($\vec{v}_1,\ldots,\vec{v}_5$ and $\vec{\sigma}_{Poly}$, $\vec{\sigma}_{RBF} \in R_+^2$) using randomization. These 5 initial solutions are evaluated to calculate their fitness. Then, 5 new solutions are created by reproduction, crossover, function mutation, and parameter mutation. Only 5 solutions from 5+5 solutions are selected to be parents in the next generation. These processes are repeated until a fixed number of generations have been produced and evaluated. The (5+5)-GPES algorithm is shown in Figure 6-1.

```
$t = 0$;
initialization ($\vec{v}_1,\ldots,\vec{v}_5$, $\vec{\sigma}_{Poly}$, $\vec{\sigma}_{RBF}$);
evaluation $f(\vec{v}_1),\ldots, f(\vec{v}_5)$;
while ( $t < 500$ ) do
    for $i$ =1 to 5 do
        $\vec{v}_i'$ = reproduction($\vec{v}_1,\ldots,\vec{v}_5$);
        $\vec{v}_i'$ = mutate_function($\vec{v}_i'$);
        $\vec{v}_i'$ = mutate_parameter($\vec{v}_i'$);
        evaluate $f(\vec{v}_i')$;
    end
    ($\vec{v}_1,\ldots,\vec{v}_5$) = select($\vec{v}_1,\ldots,\vec{v}_5$, $\vec{v}_1',\ldots,\vec{v}_5'$);
    $\vec{\sigma}_{Poly}$ = mutate$_\sigma$($\sigma_{Poly}$);
    $\vec{\sigma}_{RBF}$ = mutate$_\sigma$($\sigma_{RBF}$);
    $t = t + 1$;
End
```

Figure 6-1: (5+5)-GPES Algorithm

### 6.2.2 Terminal and Function Sets

To ensure that a new hybrid kernel function is still a Mercer's kernel, there are two operators that have been proved in Corollary 3-1. The addition ($+$) and the multiplication ($\times$) are used as the membership in the function set of evolutionary process. For the terminal set, some basic Mercer's kernel functions are considered. The polynomial kernel function is an inner-product based kernel and the RBF kernel function is a distance based kernel. Thus, the combination of these two kernel functions is a new kernel function that is both inner-product based and distance based kernel. Hence, polynomial and RBF kernels are selected as two Mercer's kernels in the terminal set. Each function is multiplied by a non-negative real value that is the weight of each sub-kernel function.

$$\text{Polynomial:} \quad K(x,x') = a\left(1 + \langle x \cdot x' \rangle\right)^d \tag{154}$$

$$\text{Gaussian RBF:} \quad K(x,x') = b \cdot \exp\left(-\gamma \left\| x - x' \right\|^2\right) \tag{155}$$

### 6.2.3 Creating an Individual

The Grow method is used to initialize the population of trees. Each tree is grown until no more leaves could be expanded, all leaves are terminals, or until a preset initial maximum depth, i.e. 5 for the experiments reported here, is reached.

**Algorithm 6-2** (Grow Method):

Step 1. Starting from the root of the tree, every node is randomly chosen as either a function or terminal.

Step 2. If the node is a terminal, the random parameters of terminal are chosen.

Step 3. If the node is a function, a random function is chosen, and that node is given two children.

Step 4. For every one of the function's children, the algorithm starts again, unless the child is at depth $d$, in which case the child is made a randomly selected terminal.

This method does not guarantee individuals of a certain depth (although they will be no deeper than $d$). An example of the hybrid kernel is shown in Figure 6-2.

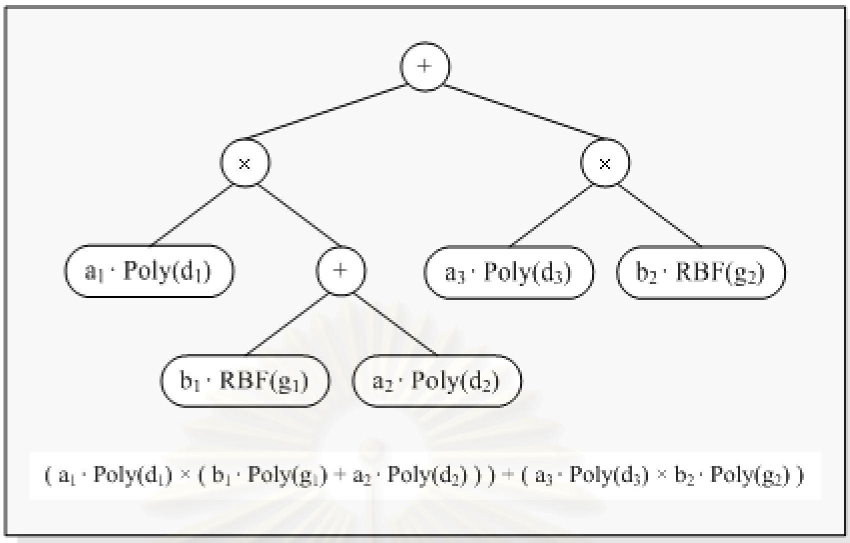

+
×
×
$a_1 \cdot \mathrm{Poly}(d_1)$
+
$a_3 \cdot \mathrm{Poly}(d_3)$
$b_2 \cdot \mathrm{RBF}(g_2)$
$b_1 \cdot \mathrm{RBF}(g_1)$
$a_2 \cdot \mathrm{Poly}(d_2)$
$( a_1 \cdot \mathrm{Poly}(d_1) \times ( b_1 \cdot \mathrm{Poly}(g_1) + a_2 \cdot \mathrm{Poly}(d_2) ) ) + ( a_3 \cdot \mathrm{Poly}(d_3) \times b_2 \cdot \mathrm{Poly}(g_2) )$

Figure 6-2: A Hybrid Kernel Function (Represented as Tree)

### 6.2.4 Genetic Operations

The main genetic operations used in the genetic programming are crossover and mutation. Crossover operation is performed by switching one of an individual's nodes with another node from another individual in the population. Since the individual is represented as tree, replacement of a node means we are replacing whole branch. This makes the results very much different from their initial parents. A mutation operation is applied to an individual in the population. This operation can be performed by substitution of a whole node in the individual, or replacement of just the node's information. The mutation operation must be aware of binary operation node, and this operation must be able to handle the missing values. In GPES, there are three main genetic operations, i.e. reproduction, function mutation, and parameter mutation.

**(1) Reproduction**: This operation is divided into 2 parts. In order to produce a new structure of hybrid kernels, 60% of new individuals are newly created by the Grow method. Moreover, 40% of new individuals are generated by crossover operation. This operation requires two individuals to produce an offspring. The crossover is described in Algorithm 6-3 and Figure 6-3.

**Algorithm 6-3** (Crossover Operation):

Step 1. Two individuals are randomly chosen from the population to be the primary individual and secondary individual, respectively.

Step 2. Randomly select positions of sub-trees on both primary and secondary individuals.

Step 3. Replace the sub-tree of the secondary individual to the sub-tree of the primary individual.

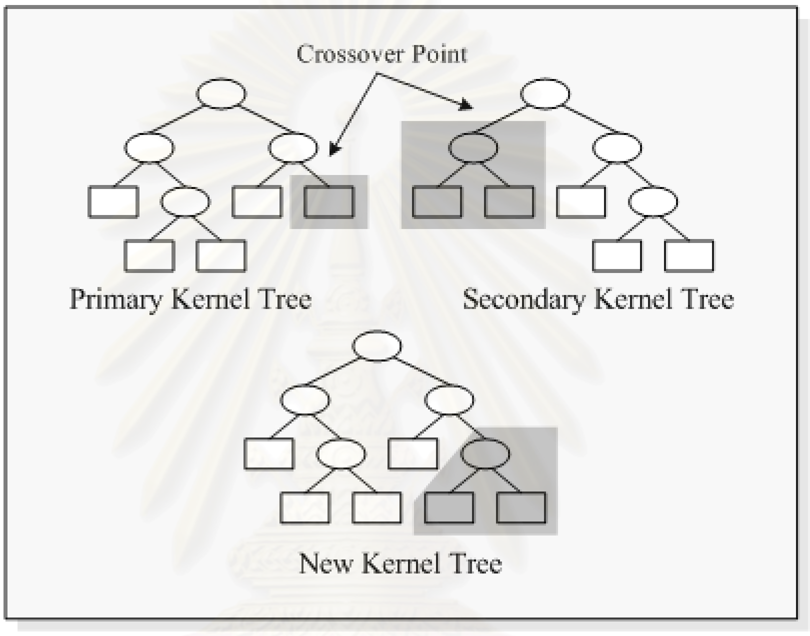


Figure 6-3: Crossover Operation

**(2) Function Mutation**: Since we have only two types of operands and two types of operators, function mutation is performed by randomly selecting the position for mutation with probability 0.7. Then, swapping between (“$+$” and “$\times$”) or (“Polynomial Kernel” and “RBF Kernel”) is performed.

$$\textit{function\_mutate}(+) = \times \tag{156}$$

$$\textit{function\_mutate}(\times) = + \tag{157}$$

$$\textit{function\_mutate}(\text{Poly}) = \text{RBF} \tag{158}$$

$$\textit{function\_mutate}(\text{RBF}) = \text{Poly} \tag{159}$$

**(3) Parameter Mutation**: The leaf nodes are randomly selected with probability 0.7. Let $\bar{u}$ be the vector of non-negative real numbers that represent the parameters of a sub-kernel. Therefore, $\bar{u} = (a, d)$ when the leaf node is a polynomial function or $\bar{u} = (b, \gamma)$ when the leaf node is RBF, where $a$ and $b$ are the weighting constants, $d$ is the degree of polynomial sub-kernel function, and $\gamma$ is the width of RBF sub-kernel function. For each selected leaf node, its parameters are mutated by the following function:

$$\begin{gathered} parameter\_mutate(\bar{u}) \ = \ (u_1 + z_1,\ u_2 + z_2) \\ z_i \sim N_i(0, \sigma_i^2)\,. \end{gathered} \tag{160}$$

The $\bar{u}$ is mutated by adding each of them with $\bar{z} = (z_1, z_2)$, and $z_i$ is a random value from a normal distribution with zero mean and $\sigma_i^2$ variation. In each generation, the standard deviation is adjusted by

$$\begin{gathered} mutate_{\sigma}(\bar{\sigma}) \ = \ (\sigma_1 \cdot e^{z_1},\ \sigma_2 \cdot e^{z_2}) \\ z_i \sim N_i(0, \tau_i^2)\,, \end{gathered} \tag{161}$$

where $\tau$ is an arbitrary constant. In the GPES algorithm, there are 2 standard deviation vectors, i.e. $\bar{\sigma}_{Poly}$ and $\bar{\sigma}_{RBF}$.

### 6.2.5 Fitness Test

Another key for this approach is the choice of the fitness function. An obvious choice for the fitness function is the classification error on the training set, but this function might produce the hybrid kernel trees for SVM that overfit to the training data. One alternative is to base the fitness on a cross-validation test in order to give a better estimation of a kernel tree's ability to produce a model that generalizes well to unseen data. However, this would obviously increase computational effort greatly. Therefore, we propose to use the bound of generalization error that is derived from the complexity of the learning algorithm as the objective functions.

$$fitness(\bar{v}) \ = \ R_{emp} + \sqrt{\frac{c}{m}\left(h \log^2 m + \log(1/\delta)\right)}\,, \tag{162}$$

where $R_{emp}$ is the empirical error, $c$ is a constant, $m$ is the number of sample, $1-\delta$ is the probability of this bound, and $h$ is a non-negative real number called the Vapnik-Chervonenkis (VC) dimension.

There is another bound of the generalization error that was proposed in Chapter 3. That bound was derived from the assumption of stability and it yielded the good results on many datasets. However, the stability bound is not suitable for evaluating the kernel trees because the bound of a kernel function must be used for calculating the bound of generalization error. Sometimes, the bound of a kernel tree cannot be calculated or it is very difficult to estimate the tight bound. Hence, the bound of generalization error that was derived from the complexity of the learning algorithm is proposed to be an objective function in our GPES algorithm. Also, we appreciate a set of kernel parameters that provide a lower bound.

## 6.3 Experimental Results

In order to verify the performance of GPES, SVMs with the hybrid kernels are trained and tested on only 7 datasets from the UCI repository [7], i.e. German, Heart, Ionosphere, Liver-Disorder, Pima-Indians-Diabetes, Sonar, and Tokyo. These datasets are selected for two reasons: they are high average percentage error (more that 20%) or their numbers of features are large (more than 20). GPES is evaluated by 5-folds cross-validation. The value of $\tau$ in evaluation process of these experiments is 1.0. The weight of sub-kernels ($a$ or $b$) are between 0.0 and 1.0, the degrees of polynomial ($d$) are integer numbers in [1,10], and the width of RBF ($\gamma$) are non-negative real numbers no more than 10. The hybrid kernels are inspected within 500 generations of (5+5)-GPES. The average accuracies of the proposed algorithm are compared with those of the standard SVM using Polynomial and RBF kernels with various parameter settings in Table 6-1.

The experimental results show the ability of GPES that creates the hybrid kernels which yields high average accuracy on testing. Although the parameter settings may provide a good average accuracy in some cases, the results are not good in many cases. If unsuitable kernels are chosen or unsuitable parameters are selected, the average accuracy will decrease. Practically, we do not have any prior knowledge about kernel functions and their parameters. Therefore, GPES is a better choice as it is able to choose a good hybrid kernel and suitable parameters without any knowledge about kernel functions.

Table 6-1: Average Error on Classification

| **Classifiers** | **Datasets** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **German** | **Heart** | **Ionosphere** | **Liver-Disorders** | **Pima-Indians-Diabetes** | **Sonar** | **Tokyo** |
| Polynomial Kernel – Degree | | | | | | | |
| 1 (Linear) | **76.20** | **84.44** | 86.90 | 57.68 | **77.33** | 79.29 | 91.87 |
| 2 | 74.30 | 80.74 | 89.45 | 62.61 | 76.82 | 87.00 | **92.70** |
| 3 | 69.40 | 77.41 | 87.45 | 68.41 | 76.69 | 88.93 | 90.51 |
| 4 | 68.20 | 73.70 | 86.89 | **72.75** | 76.56 | 87.97 | 89.89 |
| 5 | 68.90 | 74.07 | 86.32 | 71.30 | 76.42 | 88.43 | 90.20 |
| RBF Kernel – Width | | | | | | | |
| 10 | 70.20 | 69.63 | 88.89 | 67.54 | 75.77 | 59.12 | 90.72 |
| 1 | 74.60 | 80.74 | **94.59** | 59.13 | 76.56 | **88.95** | **92.70** |
| 0.1 | 75.10 | **84.44** | 91.74 | 57.97 | 76.29 | 83.65 | 91.45 |
| 0.01 | 70.00 | 82.96 | 74.07 | 57.97 | 65.10 | 65.88 | 89.05 |
| 0.001 | 70.00 | 55.56 | 64.10 | 57.97 | 65.10 | 53.37 | 65.38 |
| Nearest Neighbor, SVM with GridSearch, and SVM with Adaptive Combined Kernel Functions and GPES | | | | | | | |
| 1-NN | 67.40 | 75.19 | 87.48 | 60.29 | 70.56 | 87.00 | 90.83 |
| Grid Search – RBF | 71.10 | 71.48 | 89.45 | 67.25 | 73.68 | 69.14 | 90.40 |
| Grid Search – Polynomial | 68.90 | 76.30 | 87.17 | **70.72** | 68.87 | 88.43 | 88.01 |
| SVM with ES on 10-RBF | **73.20** | **80.37** | 94.90 | 68.41 | **77.08** | 87.02 | **92.91** |
| SVM with (5+5)-GPES | 72.40 | **80.37** | **95.44** | 68.41 | 73.55 | **92.30** | 91.87 |

However, the running time of GPES depends on the number of training data, the number of population, and the number of generation in evolutionary process. Therefore, it is rather obvious that the running time of GPES is more than the SVM with a set of parameters. Moreover, in the case of GPES, the running time is higher than the non-negative linear combination kernels in Chapter 3, since GPES needs time for creating kernel trees whereas the form of kernel functions in Chapter 3 are fixed. Nevertheless, as in the previous chapters, GPES is performed on SVM learning that is the off-line process, and thus the running time is not our main problems and it can be solved by other computing techniques, such as parallel computing, distributed computing, or cluster computing.

In conclusion, GPES is proposed to evolve the hybrid kernel functions and their parameters for support vector machines. A hybrid kernel is represented as a tree. The crossover and the mutation operations are similar to those of the genetic programming. Besides, the parameters of sub-kernels are mutated in the same way as the mutation in the evolutionary strategy. By GPES, a hybrid of several Mercer's kernel functions is also a Mercer's kernel. Moreover, the stability of SVM is a good objective function that can avoid the overfitting problem. The experimental results show the performance of the proposed method through the average accuracy on 5-folds cross-validation. GPES chooses the hybrid kernel that yields better results.

# CHAPTER VII

# CONCLUSION AND FUTURE WORK

In this chapter, the proposed methods are concluded. The advantage and disadvantage of them are described. Moreover, some suggestions for the future work are presented.

## 7.1 Conclusion

In order to improve the performance of classification or approximation in SVM, this research proposed the adaptive combined kernel functions. These proposed kernel functions are the non-negative linear combination of the common Mercer's kernels, i.e. polynomial and RBF kernels. Furthermore, these combined kernel functions were proved to be the admissible kernels by the Mercer's theorem. Although these proposed kernel functions are more flexible to the problems, they are many adjustable parameters.

In general, the parameters of SVM and kernel function are determined by the grid search. However, when the combined kernel functions are used, there are more adjustable parameters and the number of possible combinations on these parameters is much larger. Therefore, the grid search will consume a lot of time and it is not a suitable method for adjusting the parameters of the combined kernels. Hence, ES is a better choice that can search these parameters simultaneously.

The (5+10)-ES was applied to determine the optimal parameters of the kernel function and SVM. Although ES can find a good set of parameters of the combined kernel function in a fixed number of generations, a more flexible kernel can be the cause of the overfitting problem. In order to avoid this problem, the difference objective functions were proposed. The training error, the subsets cross-validation, the bound of generalization error, and the stability of SVM are considered to be the objective function in evolutionary process.

Eventually, we suggest for using the stability of SVM as it provides the best performance on many datasets. Although the subsets cross-validation is a good objective function, its running time is more than the other objective functions. For the stability objective function, a bound of generalization error was derived from the stability property of soft margin SVM. It is a tight bound and it is a good estimator for the generalization error of SVM learning.

The experimental results showed the performance of the proposed method through the average error on 5-fold cross-validation in both classification and regression problems. The adaptive combined kernel functions yielded better results, when they were compared with the common kernels, the grid search, or k-NN. When SVM used the proposed kernels, it is able to learn from data very well. This research showed that the performance of classification and regression can be further enhanced by these combined kernel functions. Furthermore, the evolutionary strategy with a suitable objective function is effective in optimizing the parameters.

Furthermore, the proposed methods were applied on two real world problems, i.e. sentiment classification and handwritten recognition. The sentiment classification is a binary classification problem, which has a lot of noise. The linear kernel (polynomial kernel with degree 1) yielded the best result when it was compared to the other common kernels such as the polynomial with various degrees or the RBF at various scales. However, the performance of sentiment classification can be enhanced by the adaptive combined polynomial kernel, which is the non-negative linear combination of multiple polynomial kernels with difference degrees. Although the mixed kernel that combines both polynomial and RBF does not yield the best result, the training rate is rather well.

For the handwritten recognition, we showed that the proposed methods can be implemented for the multi-class problems. SVM with the adaptive combined kernel functions can be used to create each binary classifier. Then, the voting schema on these binary classifiers can be applied to classify unseen examples. The experimental results show that the proposed methods yielded the good results on handwritten recognition and the other multi-class problems.

Moreover, the evolving kernel tree was proposed. Sometimes, we would like to get a more flexible kernel functions. The non-negative linear combination of multiple kernel functions is only a kind of the combination. There are the other forms of kernel combination that can be created by the genetic programming. In this research, GPES is proposed to evolve the hybrid kernel functions and their parameters for SVM. A hybrid kernel is represented as a tree. The polynomial and RBF sub-kernels were used as the terminal set. The members of function set were the addition and the multiplication, which had been proved that they correspond to the Mercer’s theorem. The experimental results showed the ability of GPES through the average accuracy on 5-fold cross-validation. GPES chooses the hybrid kernel that yielded good results.

From all experiments, the proposed methods showed good performance on classification or regression problems. They are suitable for the complex problems where we have no prior knowledge about kernel functions and their parameters. Moreover, the non-negative linear combination can be applied to other Mercer’s kernels such as Fourier series, or spectrum kernels, as the general form of linear combination of the Mercer’s kernels has been proved to be a Mercer’s kernel already. In the same manner, GPES can also be applied to the other Mercer’s kernels.

## 7.2 Future Work

Similar to with the other scientific research studies, the process of solving problems can be applied to many new problems. Our proposed methods are not different. The time series prediction is a kind of problems that can use the proposed methods. Although the implementation of this kind of problems is very similar to any regression problems, the training data are temporal and the features (or attributes) must be defined by the researchers. Therefore, an evaluation of the proposed methods on this problem would be interesting.

In the evolutionary process, the new objective functions should be investigated. Actually, the good objective function should guarantee the stability and the robustness. The probability that an outlier occurs should be minimized, as the globally optimal results should be found. Moreover, it would be interesting to compare the other parameter selection methods for the hybrid kernel functions, such as gradient descent methods or Tabu search.

Furthermore, the complex kernel functions can be generated by the other methods such as the composite of kernel functions. The further research studies should give more analysis and a proof of those combinations.

# REFERENCES

[1] Vapnik, V.N. Statistical Learning Theory. New York, USA: John Wiley and Sons, 1998.

[2] Vapnik, V.N. The Nature of Statistical Learning Theory. New York, USA: Springer-Verlag, 1995.

[3] Sullivan, K. and Luke, S. Evolving Kernels for Support Vector Machine Classification. Proceedings of the Genetic and Evolutionary Computation Conference. (July 2007):1702-1707.

[4] Schölkopf, B., Burges, C., and Smola, A.J. Advances in Kernel Methods: Support Vector Machines. Cambridge, MA: MIT Press, 1998.

[5] Ayat, N.E., Cheriet, M., Remaki, L., and Suen, C.Y. KMOD-A New Support Vector Machine Kernel with Moderate Decreasing for Pattern Recognition. Proceedings on Document Analysis and Recognition. (September 2001): 1215-1219.

[6] Kecman, V. Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models. London: The MIT Press, 2001.

[7] Asuncion, A. and Newman, D.J. UCI Machine Learning Repository [Online]. Irvine, CA: University of California, Department of Information and Computer Science, 2007. Available from: http://www.ics.uci.edu/~mlearn/MLRepository.html [17 January 2008]

[8] Cristianini, N. and Shawe-Taylor, J. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. UK: Cambridge University Press, 2000.

[9] Müller, K., Mika, S., Rätsch, G., Tsuda K., and Schölkopf, B. An Introduction to Kernel-Based Learning Algorithm. IEEE Transactions on Neural Networks. 12 (March 2001): 181-201.

[10] Taylor, J.S. and Cristianini, N. Kernel Methods for Pattern Analysis. UK: Cambridge University Press, 2004.

[11] Gunn, R. Support Vector Machines for Classification and Regression [Online]. University of Southampton, 10 May 1998. Available from: http://www.isis.ecs.soton.ac.uk/isystems/kernel [14 May 2004]

[12] Burges, C. A Tutorial on Support Vector Machines for Pattern Recognition. Kluwer Academic Publishers, 1998.

[13] Cortes, C. and Vapnik, V.N. Support Vector Networks. Machine Learning. 20 (1995): 273-297.

[14] Schölkopf, B. and Smola, A.J. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. London: MIT Press, 2002.

[15] Smola, A.J. and Schölkopf, B. A Tutorial on Support Vector Regression. NEUROCOLT Technical Report NC-TR-98-030. London: Royal Holloway College, 1998.

[16] Deng, Y.F., Jin, X., and Zhong, Y.X. Ensemble SVR for Prediction of Time Series. Proceedings of the Fourth International Conference on Machine Learning and Cybernetics. (August 2005): 3528-3534.

[17] Howley, T. and Madden, M.G. The Genetic Kernel Support Vector Machine: Description and Evaluation. Artificial Intelligence Review. 24 (November 2005): 379-395.

[18] Huhn, H.W. and Tuckers, A.W. Nonlinear Programming. Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics. (1951): 481-492.

[19] Mangasarin, O.L. Nonlinear Programming. New York, NY: McGraw-Hill, 1969.

[20] Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 2003.

[21] Tan, Y. and Wang, J. Support Vector Machine with a Hybrid Kernel and Minimal Vapnik-Chervonenkis Dimension. IEEE Transactions on Knowledge and Data Engineering. 16 (April 2004): 385-395.

[22] Mitchell, T. M. Machine Learning. New York: McGraw-Hill, 1997.

[23] Bishop, C.M. Pattern Recognition and Machine Learning. Singapore: Springer, 2006.

[24] Bartlett, P. and Shawe-Taylor, J. Generalization Performance of Support Vector Machines and Other Pattern Classifiers. In Schölkopf, B., Burges, C., and Smola, A. Advances in Kernel Methods – Support Vector Learning. Cambridge: MIT Press, 1998.

[25] Beyer, H.-G. The Theory of Evolution Strategies. Germany: Springer, 2001.

[26] Miettinen, K., Neittaanmäki, P., Mäkelä, M.M., and Périaux, J. Evolutionary Algorithms in Engineering and Computer Science. England: John Wiley & Sons, 1999.

[27] Watanabe, K. and Hashem, M.M.A. Evolutionary Computations. Germany: Springer, 2004.

[28] Rechenberg, I. Cybernetic Solution Path of an Experimental Problem. Ministry of Aviation, Royal Aircraft Establishment, UK, 1965.

[29] Rechenberg, I. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. Stuttgart, Germany: Frommann-Holzboog Verlag, 1973.

[30] Schwefel, H.-P. Numerical Optimization for Computer Models. Chichester, UK: John Wiley and Sons, 1981.

[31] Schwefel, H.-P. Evolution and Optimum Seeking. New York: John Wiley and Sons, 1995.

[32] Beyer, H.-G. and Schwefel, H.P. Evolution strategies: A comprehensive introduction. Natural Computing. 1(2002): 3-52.

[33] Goldberg, D.E. Genetic Algorithms in Search, Optimization and Machine Learning. US: Addison-Wesley, 1989.

[34] Fogel, D.B. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. Piscataway, NJ: IEEE Press, 1995.

[35] Friedrichs, F. and Igel, C. Evolutionary Tuning of Multiple SVM Parameters. 12th European Symposium on Artificial Neural Networks (ESANN 2004). (2004): 519-524.

[36] Banzhaf, W., Nordin, P., Keller, R.E., and Francone, F.D. Genetic Programming: An Introduction. San Francisco, California: Morgan Kaufmann Publishers, 1998.

[37] Zhang, L., Zhou, W., and Jiao, L. Support Vector Machines Based on Scaling Kernels. 6th International Conference on Signal Processing. 2 (August 2000): 1142-1145.

[38] Zhang, L., Zhou, W., and Jiao, L. Wavelet Support Vector Machine. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics. 34 (February 2004): 34-39.

[39] Chen, J.H. and Chen, C.S. Fuzzy Kernel Perceptron. IEEE Transactions on Neural Networks. 13 (November 2002): 1364-1373.

[40] Lin, C.F. and Wang, S.D. Fuzzy Support Vector Machines. IEEE Transactions on Neural Networks. 13 (March 2002): 464-471.

[41] Sun, Z. and Sun, Y. Fuzzy Support Vector Machine for Regression Estimation. IEEE International Conference on Systems, Man, and Cybernetics. 4 (October 2003): 3336-3341.

[42] Hao, P.Y. and Chaing, J.H. A Fuzzy Model of Support Vector Machine Regression. IEEE International Conference on Fuzzy Systems. 1 (May 2003): 738-742.

[43] Ong, C.S. and Smola, A.J. Machine Learning using Hyperkernels. Proceedings of the 20th International Conference on Machine Learning (ICML). (2003): 568-573.

[44] Fleuret, F., and Sahbi, H. Scale-Invariance of Support Vector Machines based on the Triangular Kernel. INRIA Research Report, N 4601, October 2002.

[45] Koji, T. Support Vector Classifier with Asymmetric Kernel Functions. Proceedings of European Symposium on Artificial Neural Networks (ESANN). (April 1999): 183-188.

[46] Hiroshi, S., Ken-ichi, N., and Mitsuru, N. Dynamic Time-Alignment Kernel in Support Vector Machine. Advances in Neural Information Processing Systems (NIPS2001). 14 (Dec 2001): 921-928.

[47] Debnath, R. and Takahashi, H. Kernel Selection for the Support Vector Machines. IEICE Transactions on Information and Systems. E87-D (December 2004): 2903-2904.

[48] Smits, G.F. and Jordaan, E.M. Improved SVM Regression using Mixtures of Kernels. Proceedings of the 2002 International Joint Conference on Neural Networks. 3 (May 2002): 2785–2790.

[49] Methasate, I. and Theeramunkong, T. Kernel Tree for Support Vector Machines. IEICE Transactions on Information and Systems. E90-D (October 2007): 1550–1556.

[50] Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. Choosing Multiple Parameters for Support Vector Machines. Machine Learning. 46 (2002): 131-159.

[51] Eads, D.R., Hill, D., David, S., Perkins, S.J., Ma, J., Porter, R.B., and Theiler, J.P. Genetic Algorithms and Support Vector Machines for Time Series Classification. Proceedings of SPIF. 4787 (2002): 74-85.

[52] Fröhlich, H., Chapelle, O., and Schölkopf, B. Feature Selection for Support Vector Machines by Means of Genetic Algorithms. 15th IEEE International Conference on Tools with AI (ICTAI 2003). (2003): 142-148.

[53] Xuefeng, L. and Fang, L. Choosing Multiple Parameters for SVM Based on Genetic Algorithm. International Conference on Signal Processing (ICSP'02). 1 (2002): 117-119.

[54] Chunhong, Z. and Licheng, J. Automatic Parameters Selection for SVM based on GA. Proceeding of the 5th World Congress on Intelligent Control and Automation. (June 2004): 1869-1872.

[55] deDoncker, E., Gupta, A., and Greenwood, G. Adaptive Integration Using Evolutionary Strategies. Proceedings of 3rd International Conference on High Performance Computing. (December 1996): 94-99.

[56] Runarsson, T.P. and Sigurdsson, S. Asynchronous Parallel Evolutionary Model Selection for Support Vector Machines. Neural Information Processing – Letter and Reviews. 3 (2004): 59-68.

[57] Igel, C. Multi-objective Model Selection for Support Vector Machines. Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005). 3410 (2005): 534-546.

[58] Fröhlich, H. and Zell, A. Efficient Parameter Selection for Support Vector Machines in Classification and Regression via Model-Based Global Optimization. IEEE International Joint Conference on Neural Networks (IJCNN '05). 3 (2005): 1431-1436.

[59] Schittkowski, K. Optimal Parameter Selection in Support Vector Machines. Journal of Industrial and Management Optimization. 1 (2005): 465-476.

[60] Guo, X.C., Yang J.H., Wu C.G., Wang, C.Y., and Liang Y.C. A Novel LS-SVMs Hyper-Parameter Selection based on Particle Swarm Optimization. Neurocomputing. 71 (October 2008): 3211-3215.

[61] Bousquet, O. and Elisseeff, A. Stability and Generalization. Journal of Machine Learning Research. 2 (2002): 499-526.

[62] Friedman, M. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. Journal of the American Statistical Association. 32 (1937): 675-701.

[63] Friedman, M. A Comparison of Alternative Tests of Significance for the Problem of m Ranking. Annuals of Mathematical Statistics. 11 (1940): 86-92.

[64] Demšar J. Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research. 7 (2006): 1-30.

[65] Iman, R.L. and Davenport, J.M. Approximations of the Critical Region of the Friedman Statistic. Communications in Statistics. (1980): 571-595.

[66] Dunn, O.J. Multiple Comparisons among Means. Journal of the American Statistical Association. 56 (1961): 52-64.

[67] Kennedy, A. and Inkpen, D. Sentiment Classification of Movie Reviews using Contextual Valence Shifters. Computational Intelligence. 22 (2006): 110-125.

[68] Pang, B., Lee, L., and Vaithyanathan, S. Thumbs up? Sentiment Classification using Machine Learning Techniques. Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP). (2002): 79-86.

[69] Dave, K., Lawrence, S., and Pennock, D.M. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. In Proceedings of the 12th International World Wide Web Conference (WWW 2003). (2003): 519-528.

[70] Li, J. and Sun, M. Experimental Study on Sentiment Classification of Chinese Review using Machine Learning Techniques. International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE 2007). (2007): 393- 400.

[71] Mullen, T. and Collier, N. Sentiment Analysis using Support Vector Machines with Divers Information Sources. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004). (2004): 412-418.

[72] Li, S., Zong, C., and Wang, X. Sentiment Classification through Combining Classifiers with Multiple Feature Sets. International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE 2007). (2007): 135-140.

[73] Okanohara, D. and Tsujii, J. Assigning Polarity Scores to Reviews Using Machine Learning Techniques. Lecture Notes in Computer Science: Natural Language Processing (IJCNLP2005). 3651 (2005): 314-325.

[74] Graf, A. and Borer, S. Normalization in Support Vector Machines. Lecture Notes in Computer Science. 2191 (2001): 277-282.

[75] Hu, M. and Liu, B. Mining and Summarizing Customer Reviews. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2004). (Aug 2004): 168-177.

[76] Vuori, V., Aksela, M., Girdziušas, R., Laaksonen, J., and Oja, E. On-line Recognition of Handwritten Characters [Online]. Available from: http://www.cis.hut.fi/~vuokkov/hcr/ [18 October 2006]

[77] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based Learning Applied to Document Recognition. Proceedings of the IEEE. 86 (November 1998): 2278-2344.

[78] Hoque, M.S. and Fairhurst, M.C. A Moving Window Classifier for Off-line Character Recognition. Proceedings of the 7th International Workshop on Frontier in Handwritten Recogniton. (2000): 595-600.

[79] Kussul, E. and Baidyk, T. Improved Method of Handwritten Digit Recognition Test on MNIST Database. Image and Vision Computing. 22 (October 2004): 971-981.

[80] Dong, J.X., Krzyzak, A., and Suen, C.Y. An Improved Handwritten Chinese Character Recognition System using Support Vector Machine. Pattern Recognition Letters archive. 26 (September 2005): 1849-1856.

[81] Ahmad, A.R., Marzuki, K., Christian, V.G. and Emilie, P. Online Handwriting Recognition using Support Vector Machine. Proceedings Analog and Digital Techniques in Electrical Engineering Conference (TENCON 2004). (November 2004): 311-314.

[82] Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. London: MIT Press, 1992.

[83] Walker, M. Introduction to Genetic Programming [Online]. Massey University, 7 October 2001. Available from: http://www.massey.ac.nz/~mgwalker/gp/index.html [14 May 2004]

# APPENDIX

## PUBLICATIONS

1. Phienthrakul, T. and Kijsirikul, B. Evolutionary Strategies for Hyperparameters of Support Vector Machines based on Multi-Scale Radial Basis Function Kernels. Submitted for Soft Computing. [Minor Revision]

2. Phienthrakul, T., Kijsirikul, B., Takamura, H., and Okumura, M. Sentiment Classification with SVMs and Evolutionary Combined Kernels. The 6th International Joint Conference on Computer Science and Software Engineering (JCSSE 2009). Phuket, Thailand, 13-15 May 2009.

3. Phienthrakul, T. and Kijsirikul, B. Adaptive Stabilized Multi-scale RBF Kernel for Support Vector Regression. International Joint Conference on Neural Networks (IJCNN) in IEEE World Congress on Computational Intelligence (WCCI 2008). Hong Kong, 1-6 June 2008.

4. Phienthrakul, T. and Kijsirikul, B. GPES: An Algorithm for Evolving Hybrid Kernel Functions of Support Vector Machines. IEEE Congress on Evolutionary Computation (CEC 2007). Singapore, 25-28 September 2007.

5. Phienthrakul, T. and Kijsirikul, B. Evolving Multi-Scale RBF Kernels for Support Vector Machines. RGJ – Ph.D. Congress VIII, Thailand Research Fund. Pattaya, Thailand, 20-22 April 2007.

6. Phienthrakul, T. and Kijsirikul, B. Evolving Parameters of Multi-Scale Radial Basis Function Kernels for Support Vector Machines. Proceedings of IASTED International Conference on Advances in Computer Science and Technology (ACST 2007). Phuket, Thailand, 2-4 April 2007.

7. Phienthrakul, T. and Kijsirikul, B. Evolutionary Support Vector Regression based on Multi-Scale Radial Basis Function Kernels. NN3 Artificial Neural Network & Computational Intelligence Forecasting Competition. 2007.

8. Phienthrakul, T. and Kijsirikul, B. Evolving Hyperparameters of Support Vector Machines based on Multi-Scale RBF Kernels. International Conference on Intelligent Information Processing (ICIIP 2006). Adelaide, Australia, 20-23 September 2006.
   Published in Intelligent Information Processing III, IFIP International Federation for Information Processing. 228 (2007): 269-278.
   ISSN 1571-5736 (Print) 1861-2288 (Online), ISBN 978-0-387-44639-4

9. Srisawat, A., Phienthrakul, T., and Kijsirikul, B. SV-kNNC: An Algorithm for Improving the Efficiency of k-Nearest Neighbor. Ninth Pacific Rim International Conference on Artificial Intelligence (PRICAI 2006). Guilin, China, 7-11 August 2006.
   Published in Lecture Note in Computer Science. 4099 (August 2006): 975-979.
   ISSN 0302-9743

10. Phienthrakul, T. and Kijsirikul, B. Adaptive Multi-Scale Radial Basis Function Kernels for Support Vector Machines. ไฟฟ้าสาร. 5 (2006): 67-76.

11. Phienthrakul, T. and Kijsirikul, B. Evolutionary Strategies for Multi-Scale Radial Basis Function Kernels in Support Vector Machines. Genetic and Evolutionary Computation Conference (GECCO 2005). Washington, D.C., USA, 25-29 June 2005.

12. Phienthrakul, T. and Kijsirikul, B. Combining Scalar-Product-Based and Distance-Based Kernels for Support Vector Machines. Electrical Engineering/ Electronics, Computer, Telecommunication, and Information Technology International Conference (ECTI-CON 2005). Pattaya, Cholburi, Thailand, 12-13 May 2005.

# BIOGRAPHY

**Name**: Miss Tanasanee Phienthrakul

**Date of Birth**: 8 January 1980

**Address**: 540 Moo 10, Soi Phetkasaem51, Bangkhae, Bangkok 10160

**Education**:

| | |
|---|---|
| 2009 | Ph.D. (Computer Engineering), Faculty of Engineering, Chulalongkorn University |
| 2006 | B.Econ., Department of Economics, Sukhothai Thammathirat Open University |
| 2003 | M.Sc. (Technology of Information System Management), Faculty of Engineering, Mahidol University |
| 2000 | B.Sc. (Mathematics) with First-Class Honors, Faculty of Science, Mahidol University |

**Financial Support**:

- The Thailand Research Fund (the Royal Golden Jubilee Ph.D. Program)
- The Royal Thai Government Scholarship (Mahidol University)
- The 90th Anniversary of Chulalongkorn University Fund (Ratchadaphiseksomphot Endowment Fund)
- Conference Grant for Ph.D. Student, Chulalongkorn University
- Teaching Assistant Fellowship, Department of Computer Engineering, Chulalongkorn University

**Honor**:

- 3rd Prize in Dissertation and Project Contest, Faculty of Engineering, Chulalongkorn University (2006)