5-3-2007

Evolutionary Granular Kernel Machines

Bo Jin

EVOLUTIONARY GRANULAR KERNEL MACHINES

by

BO JIN

Under the Direction of Yan-Qing Zhang

ABSTRACT

Kernel machines such as Support Vector Machines (SVMs) have been widely used in various data mining applications with good generalization properties. Performance of SVMs for solving nonlinear problems is highly affected by kernel functions. The complexity of SVMs training is mainly related to the size of a training dataset. How to design a powerful kernel, how to speed up SVMs training and how to train SVMs with millions of examples are still challenging problems in the SVMs research.

For these important problems, powerful and flexible kernel trees called Evolutionary Granular Kernel Trees (EGKTs) are designed to incorporate prior domain knowledge. Granular Kernel Tree Structure Evolving System (GKTSES) is developed to evolve the structures of Granular Kernel Trees (GKTs) without prior knowledge. A voting scheme is also proposed to reduce the prediction deviation of GKTSES. To speed up EGKTs optimization, a master-slave parallel model is implemented. To help SVMs challenge large-scale data mining, a Minimum Enclosing Ball (MEB) based data reduction method is presented, and a new MEB-SVM algorithm is designed. All these kernel methods are designed based on Granular Computing (GrC). In general, Evolutionary Granular Kernel Machines (EGKMs) are investigated to optimize kernels effectively, speed up training greatly and mine huge amounts of data efficiently.

INDEX WORDS:

Bioinformatics, Computational Intelligence, Data Mining, Evolutionary Granular Kernel Machines, Evolutionary Granular Kernel Trees, Genetic Algorithms, Granular Computing, Granular Kernel Tree Structure Evolving System, Machine Learning, Minimum Enclosing Ball, Statistical Learning, Support Vector Machines

EVOLUTIONARY GRANULAR KERNEL MACHINES

by

BO JIN

A Dissertation Submitted in Partial Fulfillment of Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia Stage University

2007

EVOLUTIONARY GRANULAR KERNEL MACHINES

by

BO JIN

Major Professor: Yan-Qing Zhang
Committee: Rajshekhar Sunderraman
Saeid Belkasim
Yichuan Zhao

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2007

Acknowledgments

Firstly, I particularly thank my advisor, Dr. Yan-Qing Zhang, for his guidance and help throughout my Ph.D. study and during the completion of this dissertation. Secondly, I am grateful to my committee members, Dr. Rajshekhar Sunderraman, Dr. Saeid Belkasim, and Dr. Yichuan Zhao for their assistance. Finally, I would like to thank my family and friends for their help, encouragement and support.

Some of my Ph.D. research was supported in part by the U.S. National Institute of Health under grant P20 GM065762. I was supported by the Georgia State University Doctoral Fellowship Program of Molecular Basis of Disease (MBD). I was also supported by the Research Assistantship in Department of Computer Science at Georgia State University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Clustering Feature	CF
Clustering-Based Support Vector Machine	CB-SVM
Core Vector Machine	CVM
Directed Acyclic Graph	DAG
Evolution Strategy	ES
Evolutionary Granular Kernel Trees	EGKTs
Evolutionary Granular Kernel Machines	EGKMs
Evolutionary Voting Kernel Machine	EVKM
Genetic Algorithms	GAs
Genetic Programming	GP
Gradient Projection Method	GPM
Granular Computing	GrC
Granular Kernel Trees	GKTs
Granular Kernel Tree Structure Evolving System	GKTSES
Inductive Logic Programming	ILP
Lagrangian Support Vector Machine	LSVM
Minimum Enclosing Ball	MEB
Minimum Enclosing Ball based Support Vector Machine	MEB-SVM
Normally Distributed Clusters	NDC
Proximal Support Vector Machine	PSVM

Quadratic Programming	QP
Quantitative Structure Activity Relationship	QSAR
Radial Basis Function	RBF
Sequential Minimal Optimization	SMO
Support Vector Data Description	SVDD
Support Vector Machines	SVMs

CHAPTER 1

INTRODUCTION

Computer techniques and Internet technology allow us to capture and store huge amounts of data. Developing machine learning algorithms to identify patterns from these massive data sets automatically is one of great challenges in this information age. These patterns can help us analyze inherent relations, understand regularities, and discover new knowledge in the data sets. The development of automated learning algorithms for data prediction and pattern recognition underwent three revolutions (Shawe-Taylor and Cristianini, 2004).

1.1 Three Revolutions

1.1.1 Perceptron

In 1957, the perceptron algorithm was proposed to identify the linear relationships within sets of data by Frank Rosenblatt (Rosenblatt, 1958). As a binary classifier, the perceptron algorithm maps an input vector $\vec{x}$ to an output value. The algorithm classifies an input instance $\vec{x}$ as either positive or negative according to the sign of $f(\vec{x})$ (Equation (1.1)), where $\vec{w}$ is a weight vector and $\langle \cdot, \cdot \rangle$ is a dot product. The perceptron is the simplest type of a feed forward neural network.

$$f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b \tag{1.1}$$

1.1.2 Nonlinear Learning Algorithms

In the 1980s, feed forward multilayer forward neural networks (Rumelhart *et al.*, 1986) were introduced as a type of nonlinear learning algorithms. Typically a feed

forward multilayer neural network has nodes arranged in a multilayer topology, which contains an input layer, an output layer and one or more hidden layers. Inputs are forwarded from the input layer, through all hidden layers to the output layer. In the neural network, each node has an activation function and each connection has a weight. The back propagation algorithm is commonly used for neural network training.

Decision trees as another type of nonlinear algorithms were introduced by Quinlan in 1986. A decision tree has a tree structure in which internal nodes correspond to attributes/features and leaf nodes correspond to class labels. To build a tree, an attribute that can best split training examples into their proper classes is selected initially. A node, related branches and children nodes are then created for that attribute. The training examples are then distributed from the parent node to some appropriate children nodes and a new attribute is selected to split examples. This process repeats until a node contains examples of the same class. At that point, it stores the class label. Typically the information gain and the gain ratio are used to measure the quality of a split. Two famous decision tree algorithms are ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993). Feed forward multilayer neural networks and decision trees can be used to identify nonlinear patterns within data sets. However, these two types of algorithms suffer from some problems such as local minima and over fitting.

1.1.3 SVMs and Kernel Methods

In the 1990s, SVMs were presented by Vapnik and his colleagues (Boser *et al.*, 1992; Guyon *et al.*, 1993; Cortes and Vapnik, 1995). The design of SVMs is based on Statistical Learning Theory (Vapnik, 1995; Vapnik, 1998), which was developed by Vapnik and Chervonenkis during 1960s-1990s. According to Statistical Learning Theory,

a risk function first needs to be defined to measure the error risk average of an estimator during solving the learning problem. Then the remaining task is searching for the estimator with the lowest risk.

SVMs can effectively solve linear and nonlinear binary classification problems with good generalization capability. There are two key features in the SVMs design. One is constructing the separating hyperplane with the maximum margin. The other is the kernel based feature transformation. With the help of a nonlinear kernel, input data are transformed into a high dimensional feature space where it is “easy” for SVMs to find a hyperplane to separate data. Inspired by SVMs, many new kernel-based algorithms were developed for data mining.

The developments from the perceptron algorithm, feed forward multilayer neural networks and decision trees to the kernel-based learning algorithms are called three revolutions in pattern recognition and analysis.

1.2 Curse of Dimensionality

In machine learning, the way to represent an input vector will affect the complexity of a learning model. To solve the problem easily, it is common to transform the input data into a new feature space to obtain a good representation. Such a feature transformation can simplify learning tasks (Shawe-Taylor and Cristianini, 2004). An example of feature transformation is given in Figure 1.1.

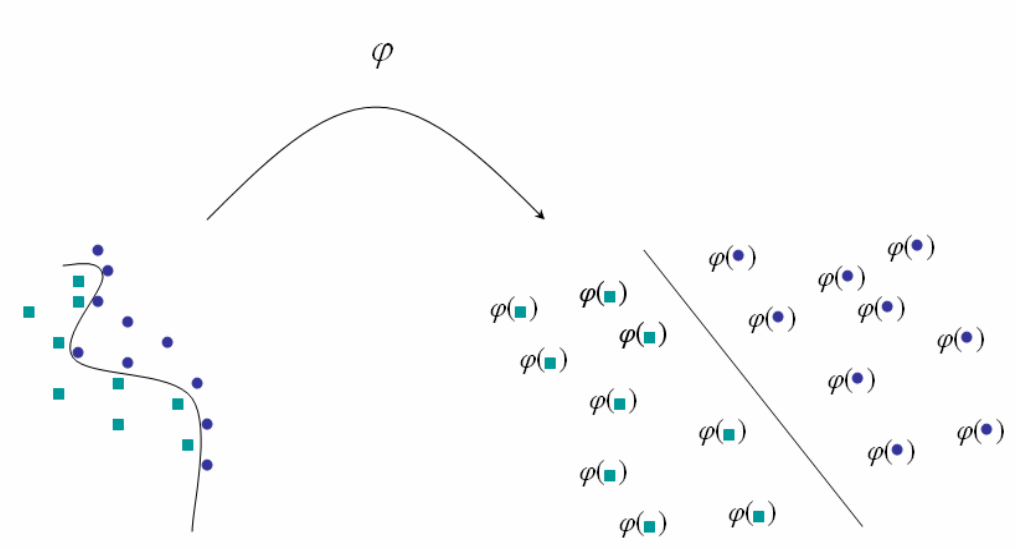

Figure 1.1 Example of feature transformation

Definition 1.1 Feature transformation is a mapping in which input vectors are transformed from the input space into a new feature space through a function φ

$$\vec{x} = (x_1, x_2, \ldots, x_n) \mapsto \varphi(\vec{x}) = (\varphi_1(\vec{x}), \ldots, \varphi_N(\vec{x})) \tag{1.2}$$

where $(\varphi_1(\vec{x}), \ldots, \varphi_N(\vec{x})) \in R^N$, $\vec{x} \in R^n$ is the input vector, and R^N is a new feature space.

Many mapping functions and techniques can be used to implement the feature mapping directly. However it is not easy to define a direct mapping, especially when the number of input features is large. The reason is that there are too many possible ways to construct a transformation for input features. Different from the direct mapping, the kernel-based feature transformation implements a kind of implicit mapping, which typically transforms data into an inner product feature space and the transformation function φ in Equation (1.2) doesn't need to be explicitly evaluated. Here is an example. Let $\vec{x}$ and $\vec{x}'$ be (x_1, x_2) and (x'_1, x'_2) respectively. A kernel function $K(\vec{x}, \vec{x}') = (\vec{x} \cdot \vec{x}')^2$ can be used to implement the following transformation φ :

$$\varphi : (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

We may rewrite $K(\vec{x}, \vec{x}')$,

$$\begin{aligned} K(\vec{x}, \vec{x}') &= (\vec{x} \cdot \vec{x}')^2 \\ &= (x_1x'_1 + x_2x'_2)^2 \\ &= (x_1x'_1 + x_2x'_2)^2 \\ &= (x_1x'_1)^2 + 2x_1x'_1x_2x'_2 + (x_2x'_2)^2 \\ &= <(x_1^2, \sqrt{2}x_1x_2, x_2^2), (x'^2_1, \sqrt{2}x'_1x'_2, x'^2_2)> \\ &= <\varphi(\vec{x}), \varphi(\vec{x}')> \end{aligned}$$

Furthermore, the number of dimensions of the new space can be very huge, and sometimes even infinite. In such a high dimensional feature space, the data become sparse and they can be separated easily. Hence kernel-based methods can overcome the curse of dimensionality.

1.3 Two Issues in the SVMs Research

There are two issues in the SVMs research. One is kernel design. Generally the variance of a kernel function is controlled by the kernel parameters, for example, Radial Basis Function (RBF) kernel's parameter γ and polynomial kernel's d. In these traditional kernels, all features are processed as one unit in vector operations and controlled by one or two parameters. With the growing interests of complex data such as biological data and chemical data, more powerful and flexible kernels need to be designed to incorporate prior domain knowledge. The other issue is the time for SVMs training. With the explosive growth of the amount of data in various areas, large-scale data mining is becoming vital. Due to the fact that training time and space complexities of SVMs mainly depend on the size of a training dataset, SVMs are not suitable for the large-scale data classification. Although some techniques have been proposed to speed up

SVMs training, how to apply SVMs to the problems with millions of examples is still a challenging problem.

1.4 Organizations and Contributions

In Chapter 2, the basic SVMs theory is briefly reviewed in Section 2.1, which focuses on the data classification. In Section 2.2, the kernel concept and some popular kernel functions are introduced. In Section 2.3, Mercer Theorem is reviewed. In Section 2.4, Kernel properties are described. The concept of MEB, the Support Vector Data Description (SVDD) algorithm, and the related applications are reviewed in Section 2.5. In Section 2.6, evolutionary computation and the related algorithms such as Genetic Algorithms (GAs), Genetic Programming (GP) and Evolution Strategy (ES) are reviewed. Finally, GrC is introduced in Section 2.6.

In Chapter 3, we review some popular kernels designed for complex data in Section 3.1. Convolution kernels, all-subsets kernel, ANOVA kernels, string kernels, tree kernels, and graph kernels are introduced. The definitions of granular feature transformation and kernel based granular feature transformation are given in Section 3.2 and Section 3.3 respectively. Granular kernel properties are summarized in Section 3.4. In Section 3.5, we present the hierarchical kernel design concept and GKTs. GKTs can effectively incorporate the prior domain knowledge such as object structures and feature relations. In Section 3.6, chromosomes used to encode a problem are first defined. The basic genetic operations such as selection, crossover, and mutation used to optimize GKTs are then described. Finally, the learning procedure of EGKTs and the system architecture are given.

In Chapter 4, Quantitative Structure Activity Relationship (QSAR) analysis and machine learning methods used for QSAR analysis are reviewed in Section 4.1. In Section 4.2, we present two types of GKTs to measure the similarity between compounds of Pyrimidines (inhibitors of *E. Coli* dihydrofolate reductase). In each GKT, the granular kernels are defined based on the possible substituent locations of compounds. Simulation results show that GKTs and the related EGKTs can improve the prediction accuracies of SVMs by 2.3%~3.4% on the Pyrimidines dataset, compared with the GAs-based SVMs with the RBF kernel. Also, based on the comparison made by Burbidge *et al.* (2001) among SVMs, Neural Networks, RBF Network and Decision Trees for the same problem, we can say that SVMs with EGKTs are better classifiers for the Pyrimidines activity comparison. In Section 4.3, we design another two kinds of GKTs for the Triazines activity comparison and simulation results show that SVMs with GKTs can outperform SVMs with RBF by 3.6%~4.5% in terms of testing accuracy.

In Chapter 5, we propose GKTSES to evolve the GKTs structures in the case of lack of prior knowledge. With the new encoding scheme and genetic operations, GKTSES are more flexible for problem solving. Simulation results show that the testing accuracies of SVMs+GKTSES are higher than those of SVMs+GAs+RBF by about 2.9%~3.9% in three evaluations in the Cyclooxygenase-2 inhibitor activity comparison. To reduce the prediction deviation of GKTSES, we also present a voting-scheme-based classification system called EVKM in Section 5.5. Simulation results show that the new voting scheme can significantly reduce the prediction deviation of SVMs+GKTSES from 6.5% to 2.3%.

In Chapter 6, SVMs with parallel computing are briefly reviewed and several Parallel GAs models are introduced at first. Then we propose parallel EGKTs, which are based on the master-slave parallel GAs model, parallelized with MPICH, and tested in a disk-shared memory-distributed Linux cluster environment. Simulation results in Section 6.4 show that our parallel method can significantly speed up the training of SVMs+EGKTs by a factor of 10 with 14 nodes.

In Chapter 7, chunking and decomposition methods are first introduced, and then several famous algorithms for large-scale data mining are reviewed in Section 7.1. In Section 7.2, MEB-SVM is proposed. In this algorithm, the kernel based MEBs are used to measure data boundaries, minimize the number of training data, and further shorten SVMs training.

In Section 7.3.2, the problem of the network intrusion detection is addressed and a standard tcpdump dataset containing 4,898,431 examples is used in the simulation. We conduct the benchmark results of MEB-SVM with regard to random sampling methods, active learning based SVM, Clustering-Based SVM (CB-SVM), and Core Vector Machine (CVM) in terms of prediction accuracy, running time, and number of support vectors. On a 512MB-RAM 3.2GHz PC, MEB-SVM can finish training in 250 seconds, which is very competitive comparing to other algorithms' running time. The MEB-SVM's prediction accuracy can reach 93.38% on the testing dataset with 311,029 examples, which is higher than those of other methods except CVM.

In Section 7.3.3, the simulation on the ring norm dataset with 100,000,000 examples show that MEB-SVM can finish training in 4013 seconds on a 2.0GB-DRAM 3.0GHz PC, which is faster than HeroSVM (on a P-4 1.7GHz machine with 1.5 GB

SDRAM) by 53191 seconds. It means that MEB-SVM is almost 13 times faster than HeroSVM. The prediction accuracy of MEB-SVM on this dataset can reach 98.44%, which is almost same as the theoretical expected accuracy of 98.76%. The simulation on the ring norm dataset with 3,000,000 examples shows that MEB-SVM can finish training in 117 seconds on a 2.0GB-DRAM 3.0GHz PC, which is 55 times as fast as CVM.

In Section 7.3.4, we compare MEB-SVM with Lagrangian SVM (LSVM) and Proximal SVM (PSVM) on the NDC datasets. Each training dataset contains 2,000,000 examples and each testing dataset contains 200,000 examples. On the dataset with the linear separability of around 70%, MEB-SVM only needs 5 seconds for training on a 2.0GB-DRAM 3.0GHz PC and can achieve the prediction accuracy of 83.1%, which is higher than those of PSVM and LSVM by about 13.5%~13.6%. On the dataset with the linear separability of around 90%~91%, MEB-SVM can finish training in 8 seconds and achieve the prediction accuracy of 98.8%, which is higher than those of PSVM and LSVM by about 7.6%. Although three algorithms are evaluated on the different machines (LSVM and PSVM are evaluated on a Pentium 400Mhz machine with a maximum of 2 GB of memory), the running time and the prediction accuracy of MEB-SVM are still competitive.

In Chapter 8, we conclude this dissertation and direct the future work.

Most of our algorithms proposed in this dissertation were already published in refereed journal and conference papers. The EGKTs algorithm (Jin *et al.*, 2005) was proposed in the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (IEEE CIBCB) and the parallelized EGKTs algorithm (Jin *et al.*, 2007) was published in the International Journal of Data Mining and Bioinformatics.

GKTSES (Jin and Zhang, 2006a) was published in the LNCS Transactions on Computational Systems Biology. The basic idea of GKTSES and related work (Jin and Zhang, 2006b; Jin and Zhang, 2006c) were also presented in the 2006 IEEE Granular Computing Conference and the third International Symposium on Neural Networks (ISNN). The MEB-SVM algorithm (Jin and Zhang, 2006d) was proposed in the 2006 IEEE International Conference on Fuzzy Systems (Fuzz-IEEE).

CHAPTER 2

RELATED THEORIES

2.1 SVMs

For a binary classification problem, let $S = \{(\vec{x}_1, y_1), \cdots, (\vec{x}_l, \vec{y}_l)\}$ represent a training data set, where $\vec{x}_i \in R^n, i = 1, \cdots, l$ are vectors and $y_i \in \{-1, +1\}, i = 1, \cdots, l$ are the class labels associated to $\vec{x}_i$. The separating hyperplane is defined as Equation (2.1) where $\vec{w} \in R^n$ and $b \in R$.

$$\langle \vec{w}, \vec{x}_i \rangle + b = 0 \tag{2.1}$$

The decision function is defined as Equation (2.2)

$$f(\vec{x}) = \text{sgn}(\sum_i \alpha_i y_i \langle \vec{x}_i, \vec{x} \rangle + b) \tag{2.2}$$

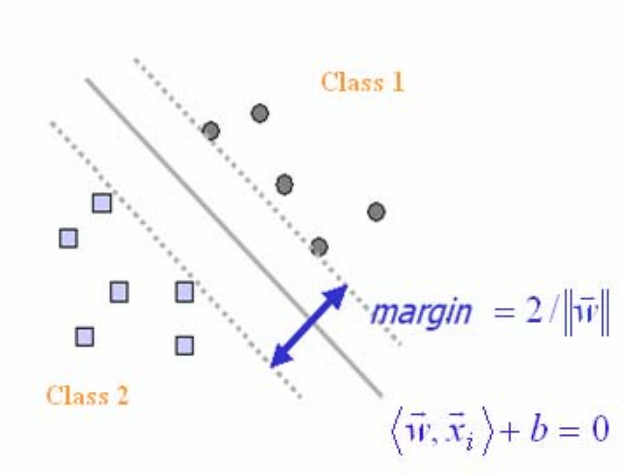

Figure 2.1 Hyperplane and margin

To get better generalization or reach the lower risk in other words, SVMs try to find an optimal hyperplane to classify data into two classes with the maximized margin,

which is shown in Figure 2.1. The related objective is formulated as Equation (2.3) and Equation (2.4). The margin of the hyperplane is measured by $\frac{2}{\| \vec{w} \|^2}$.

For the linearly separable case, the optimal hyperplane can be found by solving the following constrained optimization problem,

$$\text{Minimize } \tfrac{1}{2} \| \vec{w} \|^2 \tag{2.3}$$

$$\text{Subject to } y_i (\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1 \tag{2.4}$$

where $i = 1, \cdots, l$.

This problem can be solved by minimizing the Lagrangian Equation (2.5) with respect to w, b and satisfying Equation (2.6).

$$L \equiv \tfrac{1}{2} \| \vec{w} \|^2 - \sum_i^l \alpha_i \left(y_i \left(w \cdot x_i + b \right) - 1 \right) \tag{2.5}$$

$$\alpha_i \geq 0 \quad \forall i \tag{2.6}$$

According to the primal-dual theorem, it can be solved by maximizing Equation (2.5) subject to $\frac{dL}{dw} = 0$ and $\frac{dL}{db} = 0$, which are equivalent to Equation (2.7) and Equation (2.8) respectively.

$$w - \sum_i^l \alpha_i y_i x_i = 0 \tag{2.7}$$

$$\sum_i^l \alpha_i y_i = 0 \tag{2.8}$$

Substituting Equation (2.7) and Equation (2.8) into Equation (2.5), the final objective is reached as Equation (2.9) and Equation (2.10), where $i, j = 1, \cdots, l$.

$$\text{Maximize } \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \left\langle \vec{x}_i, \vec{x}_j \right\rangle \tag{2.9}$$

$$\text{Subject to } \sum_i \alpha_i y_i = 0, \quad y_i(\left\langle \vec{w}, \vec{x}_i \right\rangle + b) \geq 1, \quad \alpha_i \geq 0 \quad \forall i \tag{2.10}$$

According to Karush-Kuhn-Tucker (KKT) Theorem, some conclusions can be made as follows:

- **If** $\alpha_i = 0$**, then** $y_i(\left\langle \vec{w}, \vec{x}_i \right\rangle + b) \geq 1$
- **If** $\alpha_i > 0$**, then** $y_i(\left\langle \vec{w}, \vec{x}_i \right\rangle + b) = 1$

Those $\vec{x}_i$ with $\alpha_i \neq 0$ are located on the two margin planes and called support vectors. Support vectors make contributions to defining the decision boundary function (Equation 2.2). Those data with $\alpha_i = 0$ can be removed safely and the same decision function can still be obtained.

For the linearly non-separable case, a set of nonnegative slack variables $\xi_1, \cdots, \xi_l$ is introduced to penalize training errors. The constrained optimization problem is rewritten as

$$\text{Minimize } \tfrac{1}{2} \| \vec{w} \|^2 + C\sum_i \xi_i \tag{2.11}$$

$$\text{Subject to } y_i(\left\langle \vec{w}, \vec{x}_i \right\rangle + b) \geq 1 - \xi_i \tag{2.12}$$

where ξ_i are nonnegative slack variables used to penalize training errors and C is the regularization parameter to control the trade-off between the training error and the margin.

Using the Lagrangian approach, the problem can be reformulated as Equation (2.13) and Equation (2.14), where $i, j = 1, \cdots, l$.

$$\text{Maximize } \sum_i \alpha_i - \tfrac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \left\langle \vec{x}_i, \vec{x}_j \right\rangle \tag{2.13}$$

$$\text{Subject to} \sum_i \alpha_i y_i = 0\,,\; y_i(\left\langle \vec{w}, \vec{x}_i \right\rangle + b) \geq 1 - \xi_i\,,\; 0 \leq \alpha_i \leq C\,,\quad \forall i \tag{2.14}$$

It is well known that those $\vec{x}_i$ with $\alpha_i = C$ are misclassified data.

For non-linear problem, SVMs map the data from original space into a higher dimensional feature space, where an optimal separating hyperplane is found. Instead of calculating the mapping function, the kernel function K is used to implement mapping implicitly. The hyperplane is calculated by solving

$$\text{Maximize } \sum_i \alpha_i - \tfrac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j) \tag{2.15}$$

$$\text{Subject to} \sum_i \alpha_i y_i = 0\,, 0 \leq \alpha_i \leq C, i, j = 1, \cdots, l \tag{2.16}$$

The related decision function is

$$f(\vec{x}) = \text{sgn}(\sum_i \alpha_i y_i K(\vec{x}, \vec{x}_i) + b) \tag{2.17}$$

2.2 Kernels

Definition 2.1 A kernel K is a function satisfying

$$K(\vec{x}, \vec{z}) = \left\langle \phi(\vec{x}), \phi(\vec{z}) \right\rangle \tag{2.18}$$

where ϕ is a mapping from input space $X = R^n$ to an inner product feature space $F = R^N$ and all $\vec{x}, \vec{z} \in X$.

$$\phi : \vec{x} \mapsto \phi(\vec{x}) \in F \tag{2.19}$$

Let vector $\vec{x}$ be transformed into a Hilbert space with $\varphi_1(\vec{x}),\ldots,\varphi_n(\vec{x}),\ldots$. According to the Hilbert-Schmidt theory the inner product in a Hilbert space can be represented as

$$\left\langle \varphi(\vec{x}), \varphi(\vec{z}) \right\rangle = \sum_{i=1}^{\infty} a_i \varphi_i(\vec{x}) \varphi_i(\vec{z}) = K(\vec{x}, \vec{z}) \tag{2.20}$$

where K is symmetric and $a_i \geq 0$. Mercer theorem (Mercer, 1909; Vapnik, 1998) gives the necessary and sufficient conditions for K to be written as such kind of representation.

The following are some popular kernel functions.

Polynomial function $K(\vec{x}, \vec{y}) = (\vec{x} \bullet \vec{y} + 1)^d$ (2.21)

RBF $K(\vec{x}, \vec{y}) = \exp(-\gamma \| \vec{x} - \vec{y} \|^2)$ (2.22)

Sigmoid kernel $K(\vec{x}, \vec{y}) = \tanh(\vec{x} \bullet \vec{y} - \theta)$ (2.23)

2.3 Mercer Theorem

As stated by Mercer (Mercer, 1909; Berg *et al.*, 1984; Vapnik, 1998), the necessary and sufficient condition for a continuous symmetric function $K(\vec{x}, \vec{z})$ in Hilbert space has the expression defined as in Equation (2.24) is Equation (2.25), for all $f \in L_2(C)$, where $\vec{x} \in R^n$ and C is a compact subset of R^n.

$$K(\vec{x}, \vec{z}) = \sum_{i=1}^{\infty} a_i \varphi_i(\vec{x}) \varphi_i(\vec{z}) \tag{2.24}$$

$$\int_C \int_C K(\vec{x}, \vec{z}) f(\vec{x}) f(\vec{z}) d\vec{x} d\vec{z} \geq 0 \tag{2.25}$$

2.4 Kernel Properties

If K_1 and K_2 are kernels defined on $X \times X$, the following $K(\vec{x}, \vec{y})$ are also kernel functions.

$$K(\vec{x},\vec{y}) = cK_1(\vec{x},\vec{y}),\ c \in R^+ \tag{2.26}$$

$$K(\vec{x},\vec{y}) = K_1(\vec{x},\vec{y}) + c,\ c \in R^+ \tag{2.27}$$

$$K(\vec{x},\vec{y}) = K_1(\vec{x},\vec{y}) + K_2(\vec{x},\vec{y}) \tag{2.28}$$

$$K(\vec{x},\vec{y}) = K_1(\vec{x},\vec{y})K_2(\vec{x},\vec{y}) \tag{2.29}$$

$$K(\vec{x},\vec{y}) = f(\vec{x})f(\vec{y}),\ f: X \to R \tag{2.30}$$

$$K(\vec{x},\vec{y}) = \frac{K_1(\vec{x},\vec{y})}{\sqrt{K_1(\vec{x},\vec{x})K_1(\vec{y},\vec{y})}} \tag{2.31}$$

The following are proofs of Equation (2.28) and Equation (2.29). The detailed proofs were given by Cristianini and Shawe-Taylor (1999).

Proof 2.1 Let $\{\vec{x}_1,\cdots,\vec{x}_l\}$ be a fixed set, and let M_1 and M_2 be the corresponding matrices of kernels K_1 and K_2 on these points. According to Mercer Theorem, $\alpha' M_1\alpha$ and $\alpha' M_2\alpha$ are larger than or equal to 0 respectively, so $\alpha' M_1\alpha + \alpha' M_2\alpha$ is also larger than or equal to 0, for all $\alpha \in R^l$.

$$\alpha'(M_1 + M_2)\alpha = \alpha' M_1\alpha + \alpha' M_2\alpha \geq 0$$

Proof 2.2 Let $K = K_1 \otimes K_2$. The tensor product of two positive semi-definite matrices is still positive semi-definite. The product's eigenvalues are the products of the eigenvalues of the two matrices. The corresponding matrix of K_1K_2 is the Schur product H (a principal sub matrix of K), where each entry is the product of entries of the corresponding matrices of K_1 and K_2. For any $\alpha \in R^l$, there is a corresponding $\alpha_1 \in R^{l_2}$, such that $\alpha' H\alpha = \alpha_1' K\alpha_1 \geq 0$, and so H is positive semi-definite.

2.5 SVDD

MEB is the ball which encloses a given set of points with the minimum radius. In the research area of kernel methods, MEB was first used in the radius-margin bound (Vapnik, 1998; Chapelle and Vapnik, 2000; Chapelle *et al.*, 2002) for the SVMs model selection and parameter tuning. The "radius" in the Radius-Margin bound means the radius of MEB. Later the SVDD algorithm (Tax and Duin, 1999; Tax and Duin, 2004) was proposed, which can be used to calculate MEB in high dimensional space. Besides the basic MEB definition with the support vector concept, SVDD also conducts MEB with the RBF kernel and the soft-margin. SVDD can be used to solve the soft-margin one-class classification problems. For example, when SVDD is used for novelty detection, a MEB containing most of the data is calculated and the novel points outside the boundary of the ball are detected. The SVDD algorithm is reviewed as follows.

Given a data set $S = \{\vec{x}_1, \cdots, \vec{x}_l\}$, $\vec{x}_i \in R^n$, $i = 1, \cdots, l$, SVDD tries to find a ball enclosing all data of S with the minimum radius. In the input space, MEB can be found by solving the following optimization problem:

$$\text{Min}_{\text{R, c}} \; R^2 : \; \| c - \vec{x}_i \|^2 \le R^2 \tag{2.32}$$

The corresponding dual is

$$\text{Max}_{\alpha_i} \; \sum_i^l \alpha_i \langle \vec{x}_i, \vec{x}_i \rangle - \sum_i^l \sum_j^l \alpha_i \alpha_j \langle \vec{x}_i, \vec{x}_j \rangle \tag{2.33}$$

$$\text{Subject to } \sum_i^l \alpha_i = 1 \text{ and } \alpha_i \ge 0, \; i = 1, \cdots, l \tag{2.34}$$

The center c and radius R of MEB can be calculated by Equation (2.35) and Equation (2.36).

$$c = \sum_{i}^{l} \alpha_i \vec{x}_i \tag{2.35}$$

$$R = \sqrt{\sum_{i}^{l} \alpha_i \langle \vec{x}_i, \vec{x}_i \rangle - \sum_{i}^{l} \sum_{j}^{l} \alpha_i \alpha_j \langle \vec{x}_i, \vec{x}_j \rangle} \tag{2.36}$$

Those $\vec{x}_i$ with non-zero α_i are also called support vectors, which locate on the boundary of MEB.

For the kernel-based SVDD, the data are transformed from the input space into a feature space, where MEB is calculated. The corresponding dual is defined by Equation (2.37) and Equation (2.38).

$$\text{Max}_{\alpha_i} \quad \sum_{i}^{l} \alpha_i K(\vec{x}_i, \vec{x}_i) - \sum_{i}^{l} \sum_{j}^{l} \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j) \tag{2.37}$$

$$\text{Subject to} \quad \sum_{i}^{l} \alpha_i = 1 \text{ and } \alpha_i \geq 0, \quad i = 1, \cdots, l. \tag{2.38}$$

The center c and radius R of MEB in feature space H are calculated by Equation (2.39) and Equation (2.40).

$$c = \sum_{i}^{l} \alpha_i \Phi(\vec{x}_i) \tag{2.39}$$

$$R = \sqrt{\sum_{i}^{l} \alpha_i K(\vec{x}_i, \vec{x}_i) - \sum_{i}^{l} \sum_{j}^{l} \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j)} \tag{2.40}$$

If the RBF kernel is chosen as the kernel function, $K(\vec{x}_i, \vec{x}_i)$ is always equal to one and Equation (2.37) can be rewritten as Equation (2.41).

$$\text{Max}_{\alpha_i} \quad 1 - \sum_{i}^{l} \sum_{j}^{l} \alpha_i \alpha_j RBF(\vec{x}_i, \vec{x}_j) \tag{2.41}$$

In the case of MEB with slack variables (see Figure 2.2), Equation (2.32) is replaced by Equation (2.42). The parameter C is introduced to control the trade-off between the volume and the errors.

$$\text{Min}_{\text{R, c}} \; R^2 + C\sum_i \xi_i : \; \| c - \vec{x}_i \|^2 \le R^2 + \xi_i \tag{2.42}$$

The corresponding dual is

$$\text{Max}_{\alpha_i} \; \sum_i^l \alpha_i K(\vec{x}_i, \vec{x}_i) - \sum_i^l \sum_j^l \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j) \tag{2.43}$$

$$\text{Subject to } \xi_i > 0 \,,\; C \ge \alpha_i \ge 0 \,,\;\; i = 1, \cdots, l \,. \tag{2.44}$$

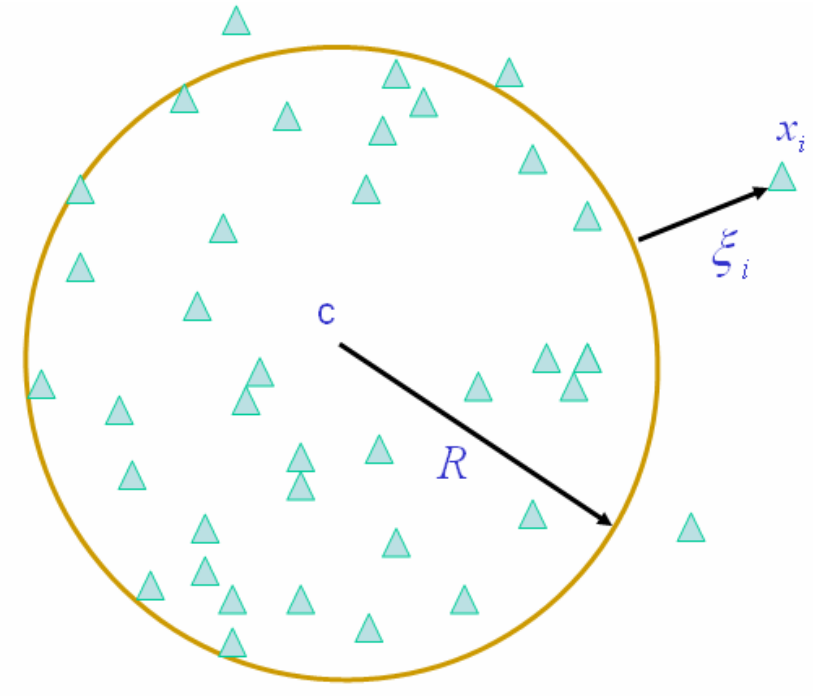

Figure 2.2 MEB with slack variables

2.6 Evolutionary Computation

Evolutionary computation is a sub field of Computational Intelligence involving the design and application of combinatorial search and heuristic methods, which takes the inspiration from natural selection and the fittest survival in the world of biology. The

evolutionary computation algorithms have some common elements such as population of chromosomes, selection, crossover, mutation and survival of the fittest.

2.6.1 GAs

GAs (Holland, 1975) is a popular type of evolutionary computation algorithms used to optimize general combinatorial problems. Given a problem and the gene representation of chromosomes for candidate solutions, GAs work as follows:

Step 1: Generate the initial population of chromosomes as candidate solutions in a random way.

Step 2: Calculate the fitness for every chromosome. The fitness of a chromosome is used to determine how good the solution described by the chromosome is.

Step 3: Select two chromosomes from the current population as parents with a certain probability and apply the crossover operation on parents.

Step 4: Apply the mutation operation on the new children chromosomes with a certain probability.

Step 5: Repeat steps 3 and 4 until a new population of the same size is generated.

Step 6: If the stop condition is met, terminate the loop; otherwise, go to step 2.

2.6.2 Other Evolutionary Computation Algorithms

Besides GAs, the following evolutionary computation algorithms are widely used for solution search and problem optimization too.

Genetic programming (GP) (Koza, 1990; Koza, 1992) works on the problem with genomes of variable length and is employed to evolve symbolic information, such as

programs and functions. A tree structure is commonly constructed and optimized to arrange the representation of genes under the operations of crossover and mutation.

Evolution Strategy (ES) (Rechenberg, 1973; Schwefel 1981) operates on the problem with the natural representation for the parameters instead of the gene-parameter mapping. The notation of $(\mu / \rho^{+}_{,} \lambda)$-ES is typically used to classify the basic types of ES. In the notation, μ represents the number of parents, λ denotes the number of offspring and ρ is the number of parents that are used in the recombination process to produce one offspring. The "+" and "," determine the selection type. Mutation and recombination are used in ES.

Evolutionary programming (EP) (Fogel, 1966) was designed by Fogel in 1960. Similar to ES, EP also operates on the problem with the natural representation and emphasizes the behavioral linkage between parents and their offspring. However, EP only uses mutation and selection operation.

2.7 GrC

GrC (Lin, 1997; Zadeh, 1997; Zadeh, 1998) is a set of theories and methodologies using information granules to build computational models for various applications with huge amounts of data and information. Here, information granules are collections of entities that typically derive at the numeric level and are arranged together according to their similarities.

The basic notions and principles of granular computing have appeared in many fields with different names, such as divide and conquer, fuzzy set theory, rough set theory, interval computing, and cluster analysis. GrC is used as an umbrella term to cover these topics in different fields. The goal of GrC is to abstract the commonalities from various

fields and establish a category of applicable principles in a unified framework. GrC has played important roles in e-Business, security, machine learning, data mining, high-performance computing, wireless mobile computing, and Bioinformatics in terms of efficiency, effectiveness, robustness, and uncertainty.

CHAPTER 3

HIERARCHICAL KERNEL DESIGN

3.1 Related Work

3.1.1 Convolution Kernels

A convolution kernel (Haussler, 1999) uses the relation R between a composite object and its parts to capture the object semantics. So convolution kernels are also called R-Convolution kernels.

Let vectors $\vec{x} = (x_1, \cdots, x_D)$ and $\vec{x}' = (x_1', \cdots, x_D')$ be the decomposed parts of $X \in X$ and $X' \in X$ respectively. x_d and x_d' are in the set $X_d, 1 \le d \le D$. For the relation $R : (X_1 \times \cdots \times X_D) \times X$, the decomposition R^{-1} is defined as $R^{-1}(X) = \{\vec{x} : R(\vec{x}, X)\}$. The relation $R(\vec{x}, X)$ is true if and only if $x_1, \cdots, x_D$ are the parts of X. A convolution kernel K is defined as Equation (3.1) where K_d is a kernel defined on $X_d \times X_d$.

$$K(X, X') = \sum_{\vec{x} \in R^{-1}(X), \vec{x}' \in R^{-1}(X')} \prod_{d=1}^{D} K_d(x_d, x_d') \tag{3.1}$$

Convolution kernels are so general that they can be used in various problems. However, how to choose R is a big issue in the real world applications.

3.1.2 All-subsets Kernel

As stated by Shawe-Taylor and N. Cristianini (2004), the all-subsets kernel is defined as

$$K(\vec{x}, \vec{y}) = \prod_{i=1}^{m} (1 + x_i y_i) \tag{3.2}$$

Let I= {1, …, m} be the indices of features x_i of vector $\vec{x}$, $i \in I$. For every subset A of I, $\varphi_A(x) = \prod_{i \in A} x_i$, $\varphi_\phi(x) = 1$, and $\varphi(x) = (\varphi_A(x))_{A \subseteq I}$ are defined. The all-subsets kernel (Equation (3.2)) can be derived by

$$\begin{aligned} K(\vec{x}, \vec{y}) &= < \varphi(\vec{x}), \varphi(\vec{y}) > \\ &= \sum_{A \subseteq I} \varphi_A(\vec{x}), \varphi_B(\vec{y}) \\ &= \sum_{A \subseteq I} \prod_{i \in I} x_i y_i \\ &= \prod_{i=1}^{m} (1 + x_i y_i). \end{aligned}$$

3.1.3 ANOVA Kernels

An ANOVA (analysis of variance) kernel K_d (Vapnik, 1998; Shawe-Taylor and N. Cristianini, 2004) is like the all-subsets kernel but restricted to subsets of cardinality d.

$$\begin{aligned} K_d(\vec{x}, \vec{y}) &= < \varphi_d(\vec{x}), \varphi_d(\vec{y}) > \\ &= \sum_{|A|=d} \varphi_A(\vec{x}), \varphi_B(\vec{y}) \\ &= \sum_{1 \le i_1 < i_2 < \ldots \le i_d \le m} \prod_{j=1}^{d} (x_j y_j) \end{aligned} \tag{3.3}$$

In Equation (3.3), $\varphi_d(\vec{x})$ is equal to $(\varphi_A(\vec{x}))_{|A|=d}$ and d is used to specify the order of the interactions between features x_{i_d}. ANOVA kernels work very well in support vector regression problems (Stitson *et al.*, 1999).

3.1.4 String Kernels

The similarity of two strings s and t can be measured based on the number of common substrings (Cristianini and Shawe-Taylor, 1999; Haussler, 1999; Lodhi, 2001). A string kernel for substrings of length p can be defined as follows:

$$K_p(s,t) = \sum_{u \in \sum^p} \varphi_u^p(s), \varphi_u^p(t) \tag{3.4}$$

where $\varphi_u^p(s) = \left| \{ (v_1, v_2) : s = v_1 u v_2, u \in \sum^p \} \right|$. Strings s and t are defined on a finite alphabet $\sum$ and the string kernel counts the number of common substrings between s and t. A recursion built on the k-suffix kernel can be used to compute the kernel:

$$K_p^s = \begin{cases} 1 & \text{if } s = s_1 u, \quad t = t_1 u \text{ for } u \in \sum^k \\ 0 & \text{otherwise} \end{cases}$$

Equation (3.4) can be rewritten as

$$K_p(s,t) = \sum_{i=1}^{|s|-p+1} \sum_{j=1}^{|t|-p+1} K_p^s(s(i:i+p), t(j:j+p)) \tag{3.5}$$

Substrings kernels can be built based on the kernel defined in Equation (3.5) and the “mismatches” within the subsequences are allowed.

3.1.5 Tree Kernels

Tree kernels (Collins and Duffy, 2002; Kashima and Koyanagi, 2002; Gärtner, 2003) are used to measure the similarity of data that can be represented as labeled ordered directed substrees. Typically a tree kernel is defined as Equation (3.6).

$$K(T_1, T_2) = \sum_i h_i(T_1) h_i(T_2) \tag{3.6}$$

In the equation, T_1 and T_2 are two trees and $h_i(T)$ is the number of occurrences of ith subtree in tree T. Let V_1 and V_2 be the sets of vertices of T_1 and T_2 respectively. Let $S(v_1, v_2)$ be the number of subtrees rooted at vertices $v_1 \in V_1$ and $v_2 \in V_2$. Then the tree kernel can be recursively computed using

$$K(T_1, T_2) = \sum_{v_1 \in V_1,\, v_2 \in V_2} S(v_1, v_2) \tag{3.7}$$

where $S(v_1, v_2)$ is defined as

$$S(v_1, v_2) = \begin{cases} 0 & \textit{if } v_1 \textit{ and } v_2 \textit{ have different labels} \\ 1 & \textit{if both } v_1 \textit{ and } v_2 \textit{ are leaf vertices and have the same label} \\ \prod_{j=1}^{|v_1|} (1 + S(v_1^j, v_2^j)) & \textit{otherwise} \end{cases}$$

where v_1^j and v_2^j are the j[th] children of v_1 and v_2 respectively. $|v_1|$ is the number of children of v_1.

3.1.6 Graph Kernels

A graph consists of a finite set of labeled vertices and a finite set of labeled edges between vertices. Two graphs that generate a product graph are called factor graphs. The vertex set of the product graph is a subset of Cartesian product of the vertex sets of the factor graphs. The product graph has a vertex if and only if the corresponding vertices in the factor graphs have the same label. An edge exists between two vertices in the product graph if an edge exists between the corresponding vertices in both factor graphs. Both edges have the same label. Let $E_\times$ denote the edge set of the product graph and let $V = \{v_1, \ldots v_N\}$ denote an enumeration of vertex set. Each element of the adjacency matrix $E_\times$ is defined by

$$[E_\times]_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E_\times, \text{ and } [E_\times]_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E_\times.$$

The graph kernel (Kashima and Inokuchi, 2002; Gärtner *et al*., 2003) is defined by

$$K_{\times}(G_1, G_2) = \sum_{i,j=1_2}^{|V_x|} \left[\sum_{n=0}^{\infty} \lambda_n E_{\times}^n \right]_{ij} \tag{3.8}$$

where $[E_{\times}^n]_{ij}$ is the number of walks of length n from v_i to v_j and $\lambda_0, \lambda_1, \ldots, (\lambda_i \geq 0, \forall i)$ are a set of weights.

Besides these kernels, other new kernels are also designed for the similarity measurement of complex data. More detailed reviews were given by Gärtner (2003) and Hofmann *et al*. (2006).

3.2 Granular Feature Transformation

Definition 3.1 A feature granule space G of input space $X = R^n$ is a sub space of X, where $G = R^m$ and $1 \leq m \leq n$.

From input space, we may generate many feature granule spaces and some dimensions could be shared among these sub spaces.

Definition 3.2 A feature granule $\vec{g} \in G$ is a vector which is defined in the feature granule space G.

Definition 3.3 Granular feature transformation is a mapping in which a feature granule is transformed from the feature granule space into a new feature space through a function φ defined in Equation (3.1), where $\vec{g} \in G$ is a feature granule and T is a new feature space.

$$\varphi : \vec{g} \mapsto \vec{t} \in T \tag{3.9}$$

Feature transformation of input vectors may be implemented with a group of granular feature transformations. An example of granular feature transformation is shown in Figure 3.1. In the example, features in the vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ are grouped into feature granules $\vec{g}_i$ $(1 \leq i \leq q)$ according to some prior domain knowledge, such as the

similarity or functional adjacency. Some features may be shared. For example, feature x_2 is shared by feature granules $\vec{g}_1$ and $\vec{g}_2$. A series of granular feature transformation functions φ_i are defined on each feature granule. Each φ_i transforms the feature granule $\vec{g}_i$ from the feature granule space into a new sub feature space R^{N_i} respectively.

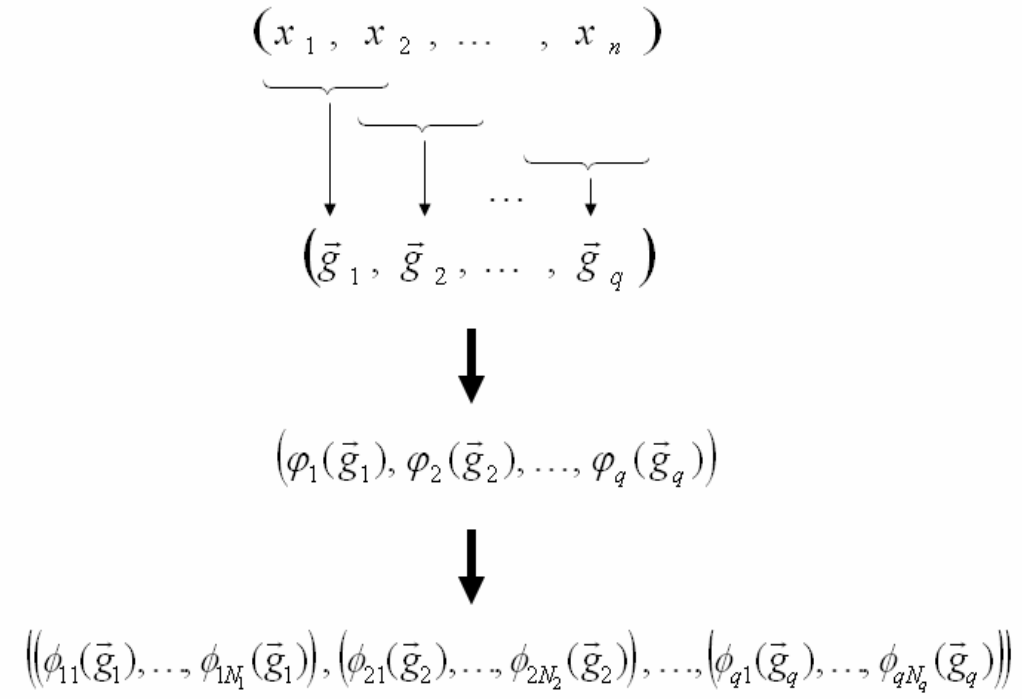

Figure 3.1 An example of granular feature transformation

3.3 Kernel Based Granular Feature Transformation

Feature transformation may be implemented with a group of kernels on feature granules. Kernel based feature transformation on feature granules is similar to that on input vectors. If all input features are chosen as a feature granule, there is no difference between them.

Definition 3.4 A granular kernel gK is a kernel that can be written in an inner product form of Equation (3.9) for all $\vec{g}, \vec{g}' \in G$.

$$gK(\vec{g}, \vec{g}') = \langle \varphi(\vec{g}), \varphi(\vec{g}') \rangle \tag{3.10}$$

In Equation 3.10, φ is a mapping function (defined in Equation (3.11)) from feature granule space $G = R^m$ to an inner product feature space R^E.

$$\varphi : \vec{g} \mapsto \varphi(\vec{g}) \in R^E \tag{3.11}$$

Figure 3.3 shows an example of kernel based granular feature transformation in which two granular kernels gK_1 and gK_2 are used to transform data instead of kernel K shown in Figure 3.2.

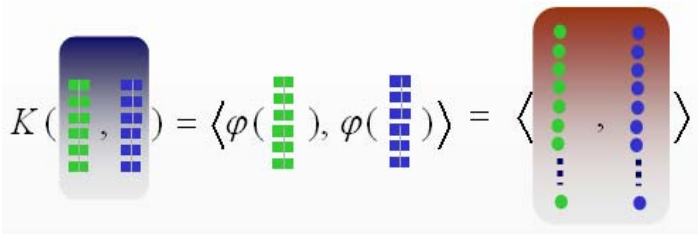

Figure 3.2 An example of kernel based feature transformation

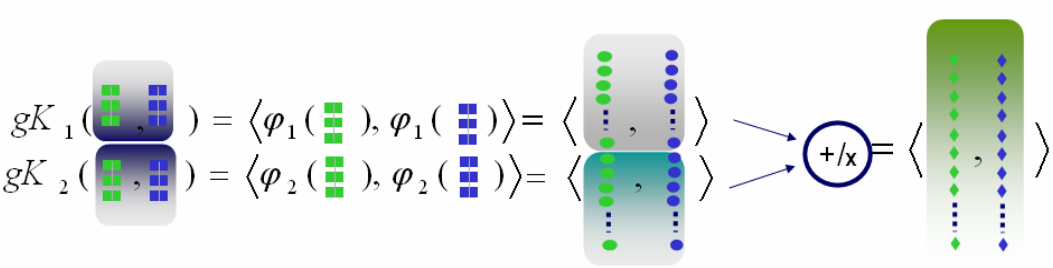

Figure 3.3 An example of kernel based granular feature transformation

3.4 Granular Kernel Properties

Property 3.1 Granular kernels inherit the properties of traditional kernels such as the closure under sum, product, and multiplication with a positive constant over the granular feature spaces.

Let G be a feature granule space and $\vec{g}, \vec{g}' \in G$. Let gK_1 and gK_2 be two granule kernels operating over the same space $G \times G$. The following $gK(\vec{g}, \vec{g}')$ are also granular kernels.

$$gK(\vec{g}, \vec{g}') = c \times gK_1(\vec{g}, \vec{g}'), \quad c \in R^+ \tag{3.12}$$

$$gK(\vec{g}, \vec{g}') = gK_1(\vec{g}, \vec{g}') + c, \quad c \in R^+ \tag{3.13}$$

$$gK(\vec{g}, \vec{g}') = gK_1(\vec{g}, \vec{g}') + gK_2(\vec{g}, \vec{g}') \tag{3.14}$$

$$gK(\vec{g}, \vec{g}') = gK_1(\vec{g}, \vec{g}') \times gK_2(\vec{g}, \vec{g}') \tag{3.15}$$

$$gK(\vec{g}, \vec{g}') = f(\vec{g}) f(\vec{g}'), \quad f : X \to R \tag{3.16}$$

$$gK(\vec{g}, \vec{g}') = \frac{gK_1(\vec{g}, \vec{g}')}{gK_1(\vec{g}, \vec{g}) \times gK_1(\vec{g}', \vec{g}')} \tag{3.17}$$

These properties can be derived from the traditional kernel properties directly.

Property 3.2 A kernel can be constructed with two granular kernels defined over different granular feature spaces under sum operation.

To prove it, let $gK_1(\vec{g}_1, \vec{g}_1')$ and $gK_2(\vec{g}_2, \vec{g}'_2)$ be two granular kernels, where $\vec{g}_1, \vec{g}'_1 \in G_1$, $\vec{g}_2, \vec{g}'_2 \in G_2$ and $G_1 \neq G_2$. New kernels can be defined like this,

$$gK((\vec{g}_1, \vec{g}_2), (\vec{g}_1', \vec{g}'_2)) = gK_1(\vec{g}_1, \vec{g}'_1)$$

$$gK'((\vec{g}_1, \vec{g}_2), (\vec{g}'_1, \vec{g}'_2)) = gK_2(\vec{g}_2, \vec{g}'_2)$$

Here gK and gK' can operate over the same feature space $(G_1 \times G_2) \times (G_1 \times G_2)$.

$$gK_1(\vec{g}_1, \vec{g}'_1) + gK_2(\vec{g}_2, \vec{g}'_2) = gK((\vec{g}_1, \vec{g}_2), (\vec{g}'_1, \vec{g}'_2)) + gK'((\vec{g}_1, \vec{g}_2), (\vec{g}'_1, \vec{g}'_2))$$

According to the sum closure property of kernels ((Berg *et al*., 1984; Haussler, 1999; Cristianini and Shawe-Taylor, 1999), $gK_1(\vec{g}_1, \vec{g}'_1) + gK_2(\vec{g}_2, \vec{g}'_2)$ is a kernel over $(G_1 \times G_2) \times (G_1 \times G_2)$.

Property 3.3 A kernel can be constructed with two granular kernels defined over different granular feature spaces under product operation (Berg *et al*., 1984; Haussler, 1999).

To prove it, let $gK_1(\vec{g}_1, \vec{g}'_1)$ and $gK_2(\vec{g}_2, \vec{g}'_2)$ be two granular kernels, where $\vec{g}_1, \vec{g}'_1 \in G_1$, $\vec{g}_2, \vec{g}'_2 \in G_2$ and $G_1 \neq G_2$. We may define new kernels like this,

$$gK((\vec{g}_1, \vec{g}_2), (\vec{g}_1', \vec{g}'_2)) = gK_1(\vec{g}_1, \vec{g}'_1)$$

$$gK'((\vec{g}_1, \vec{g}_2), (\vec{g}'_1, \vec{g}'_2)) = gK_2(\vec{g}_2, \vec{g}'_2)$$

So gK and gK' can operate over the same feature space $(G_1 \times G_2) \times (G_1 \times G_2)$.

$$gK_1(\vec{g}_1, \vec{g}'_1) gK_2(\vec{g}_2, \vec{g}'_2) = gK((\vec{g}_1, \vec{g}_2), (\vec{g}'_1, \vec{g}'_2)) gK'((\vec{g}_1, \vec{g}_2), (\vec{g}'_1, \vec{g}'_2))$$

According to the product closure property of kernels ((Berg *et al*., 1984; Haussler, 1999; Cristianini and Shawe-Taylor, 1999), $gK_1(\vec{g}_1, \vec{g}'_1) gK_2(\vec{g}_2, \vec{g}'_2)$ is a kernel over $(G_1 \times G_2) \times (G_1 \times G_2)$.

3.5 Hierarchical Kernel Design and Granular Kernel Trees

An easy and effective way to construct new kernel functions is combining a group of granular kernels via some simple operations such as sum and product shown in Figure 3.4. The new kernel functions can be naturally expressed as tree structures. The following are main steps in the GKTs design.

Step 1: Generate feature granules. Features are bundled into feature granules according to some prior knowledge such as object structures, feature relationships, similarity, or functional adjacency. They may be grouped by an automatic learning algorithm too.

Step 2: Select granular kernels. Granular kernels are selected from the candidate kernel set. Some popular traditional kernels such as RBF kernels and polynomial kernels can be chosen as granular kernels, since these kernels have proved successful in many real problems. Some special kernels designed for some particular problems could also be selected as granular kernels if they are good at measuring the similarities of corresponding feature granules.

Step 3: Construct a tree structure. A tree structure is constructed with suitable number of layers, nodes and connections. Like in Step 1, we can construct trees according to some prior knowledge or with an automatic learning algorithm.

Figure 3.4 shows a GKT with m basic granular kernels gK_t and m pairs of feature granules $\vec{g}_t$ and $\vec{g}_t'$, where $1 \le t \le m$.

Step 4: Select connection operations. Each connection operation in GKTs can be a sum or product. A positive connection weight may associate to each edge of the tree.

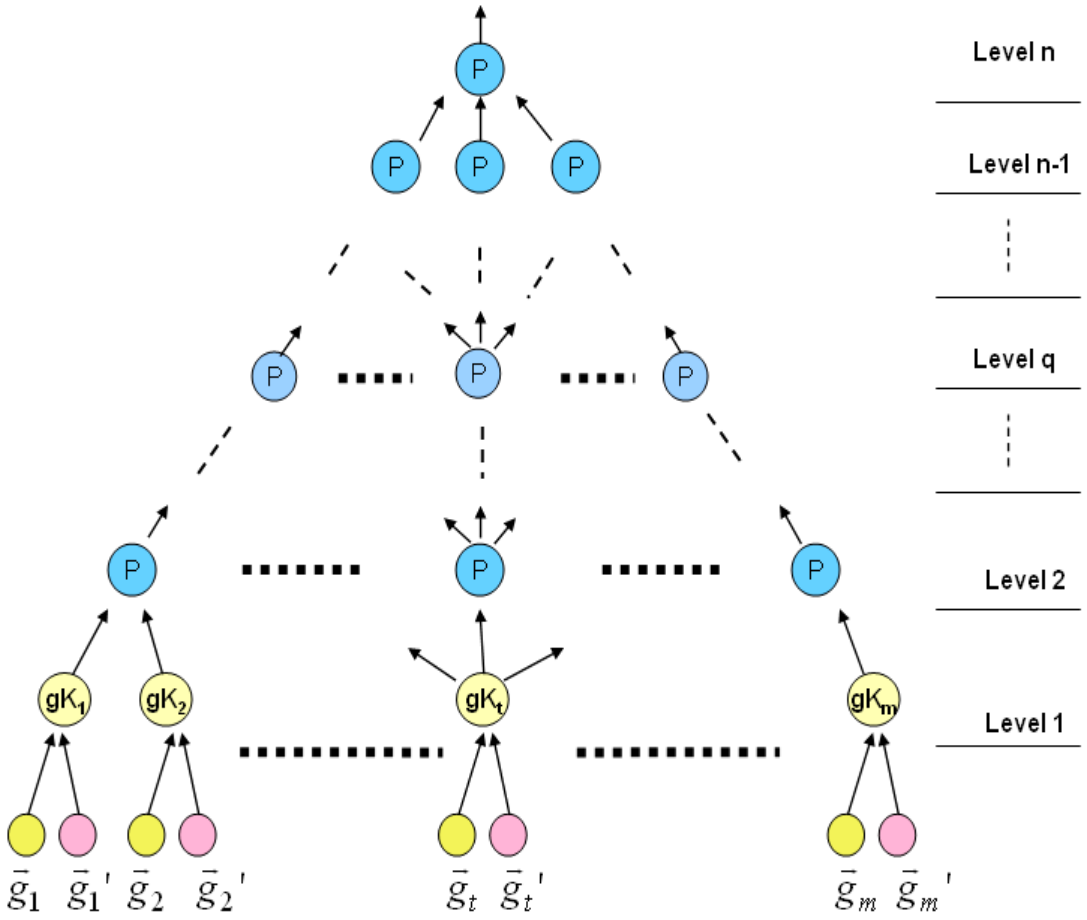

Figure 3.4 An example of GKTs

3.6 EGKTs

In our study, GAs are used to optimize the GKTs parameters. We use EGKTs to represent such kind of evolutionary GKTs. The following are basic definitions and operations used in optimizing EGKTs.

- **Chromosome** Let P_i denote the population in generation G_i, where $i = 1, \cdots m$ and m is the total number of generations. Each population P_i has p chromosomes c_{ij}, $j = 1, \cdots, p$. Each chromosome c_{ij} has q genes $g_t(c_{ij})$, where

$t = 1, \cdots, q$. Here each gene is a parameter of GKTs and we use $GKTs(c_{ij})$ to represent GKTs configured with genes $g_t(c_{ij})$, $t = 1, \cdots, q$.

- **Fitness** There are several popular methods to evaluate SVMs performance. One is using the k-fold cross-validation, which is a popular technique for performance evaluation. Others are some theoretical bounds evaluation on the generalization errors, such as Xi-Alpha bound (Joachims, 2000), VC bound (Vapnik, 1998), Radius margin bound and VCs span bound (Vapnik and Chapelle, 1999). Detailed review was given by Duan *et al*. (2003). In our method, k-fold cross-validation is used to evaluate SVMs performance in training phase. In k-fold cross-validation, the training data set $\tilde{S}$ is separated into k mutually exclusive subsets $\tilde{S}_v$. Data set $\Lambda_v = \tilde{S} - \tilde{S}_v$, $v = 1, \cdots, k$ is used to train SVMs with $GKTs(c_{ij})$ and $\tilde{S}_v$ is used to evaluate SVMs model. After k times of training-testing on all different subsets, we get k prediction accuracies. The fitness of chromosome c_{ij} is calculated by Equation (3.18) where Acc_v is the prediction accuracy of $GKTs(c_{ij})$ on $\tilde{S}_v$.

$$f_{ij} = \frac{1}{k}\sum_{v=1}^{k} Acc_v \tag{3.18}$$

- **Selection** In the algorithm, the roulette wheel method described by Michalewicz (1996) is used to select individuals for the new population. Before selection, the best chromosome (the GKT with the highest prediction accuracy in fitness evaluation) in generation G_{i-1} will replace the worst chromosome in generation G_i if the best chromosome in G_i is worse than the best chromosome in G_{i-1}. The sum of fitness values F_i in population G_i is first calculated. A cumulative fitness

$\tilde{q}_{ij}$ is then calculated for each chromosome. The chromosomes are then selected as follows. A random number r is generated within the range of [0, 1]. If r is smaller than $\tilde{q}_{i1}$, then chromosome c_{i1} is selected; otherwise chromosome c_{ij} is selected if r is in the range of $(\tilde{q}_{i\,j-1}, \tilde{q}_{i\,j}]$.

$$F_i = \sum_{j=1}^{p} f_{ij} \tag{3.19}$$

$$\tilde{q}_{ij} = \sum_{t=1}^{j} \frac{f_{it}}{F_i} \tag{3.20}$$

$$\tilde{q}_{i\,j-1} < r \le \tilde{q}_{i\,j} \tag{3.21}$$

- **Crossover** Two chromosomes are first selected randomly from current generation as parents and then the crossover point is randomly chosen to separate the chromosomes. Parts of chromosomes are exchanged between two parents to generate two children. This genetic operation is equivalent to that two GKTs are selected to exchange parameters on some granular kernels.
- **Mutation** Some chromosomes are randomly selected and some of their genes are replaced by random values generated in a specified range. This operation is equivalent to changing some parameters of GKTs randomly.

The learning procedure of EGKTs is shown in Figure 3.5 and the classification system of SVMs with EGKTs is shown in Figure 3.6.

Initialization
For each generation G_j
  For each c_{ij} in G_j
  Repeat v from 1 to k
    Train SVMs on Λ_v with the current kernel
    Evaluate SVMs on $\tilde{S}_v$
    Calculate Acc_v
  Repeat end
  Calculate fitness $f_{ij} = \frac{1}{k}\sum_{v=1}^{k} Acc_v$
  For End
  Selection
  Crossover
  Mutation
For End

Figure 3.5 Learning procedure of EGKTs

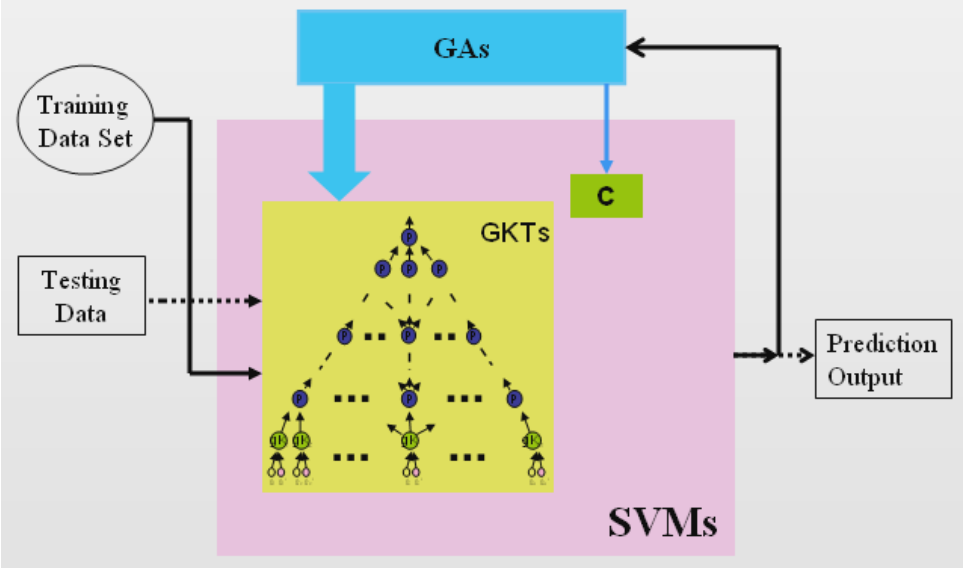

Figure 3.6 SVMs with EGKTs

CHAPTER 4

SVMS WITH EGKTS FOR DRUG ACTIVITY COMPARISON

4.1 QSAR

QSAR is an important drug design technique, which is used to describe the relationships between compound structures and their activities. In QSAR analysis, compounds with different biological activities are discriminated first, and then predictive rules are constructed, which can be used to predict a molecule's activity according to the values of its chemical and physical descriptors. QSAR can effectively reduce the search for new drugs. As a part of QSAR, the problem of drug activity comparison is to learn a binary relationship on the biological activities of compounds. The biological activity is measured by the value of $\log(1/C)$, where C is a constant for the inhibitory growth concentration. With the increased demand on prediction accuracy, machine learning methods such as GAs (Devillers, 1999a), Neural Networks (Devillers, 1999b; Hirst *et al*., 1994), Inductive Logic Programming (Hirst *et al*., 1994), and SVMs (Burbidge *et al*., 2001) have been introduced for QSAR analysis and drug activity comparison. In this chapter, Pyrimidines and Triazines, two kinds of inhibitors of *E. Coli* dihydrofolate reductase (DHFR) are studied. These inhibitors are potential therapeutic agents for the treatment of malaria, bacterial infection, toxoplasma and cancer.

4.2 Pyrimidines Activity Comparison

Pyrimidines prediction was first studied by Hirst and his colleagues (1994a). They compared Neural Networks and Inductive Logic Programming (ILP) to the linear regression for modeling the QSAR of Pyrimidines. They showed that neural networks

and ILP perform better than linear regression using the attribute representation. They also showed that the ILP analysis is a good way to formulate the understandable rules relating the activity of the inhibitors to their chemical structure. Burbidge *et al*. (2001) also studied Pyrimidines, but focused on the drug activity comparison problem. They applied some popular machine learning algorithms (such as SVMs, Neural Networks, Decision Trees, and RBF Network) to the problem and made a comparison.

4.2.1 Dataset Description

Pyrimidines dataset (Newman *et al*., 1998) contains 55 drugs, and each drug has three possible substitution positions (R_3, R_4 and R_5, see Figure 4.1). Each substituent is characterized by 9 chemical properties features: polarity, size, flexibility, hydrogen-bond donor, hydrogen-bond acceptor, π donor, π acceptor, polarizability and σ effect. Drug activities are identified by the substituents. If no substituent locates in a possible position, the features are indicated by nine -1s. Each input vector includes two drug features with the fixed feature order. In each data vector, if the activity of the first drug is higher than that of the second one, the vector is labeled positive, otherwise it is labeled negative (see Figure 4.2). The total feature number of each vector is 54. The positive and negative data are balanced absolutely.

The Pyrimidines dataset is randomly shuffled and split into 2 parts in the proportion of 4:1. One part is used as the training set, which contains pairs of 44 compounds. The other part is chosen as the unseen testing set, which contains pairs of the left 11 compounds and those between the 11 compounds and the training 44 compounds. So the size of training set should be 44 x 43 = 1892 and the size of testing set should be

44 x11 x2 +11 x 11 = 1078. Due to the deletion of some pairs with the same activities, the data sets are actually a little bit smaller than those above.

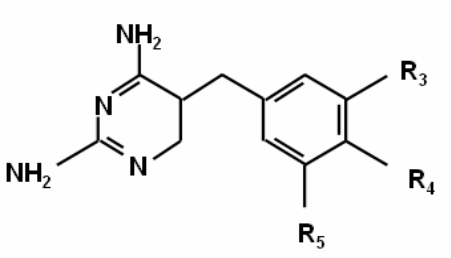

Figure 4.1 Structure of Pyrimidines

label	drug1			drug2		
+	9	9	9	9	9	9
	drug2			drug1		
-	9	9	9	9	9	9

Figure 4.2 Pyrimidines drug pairs

4.2.2 Feature Granules and Hierarchical Kernel Design

In the GKTs design, the input vectors are decomposed according to the possible substituent locations. Each feature granule includes all features of one substituent (see Figure 4.3). So each Pyrimidines drug pair has 6 feature granules and each feature granule has 9 features.

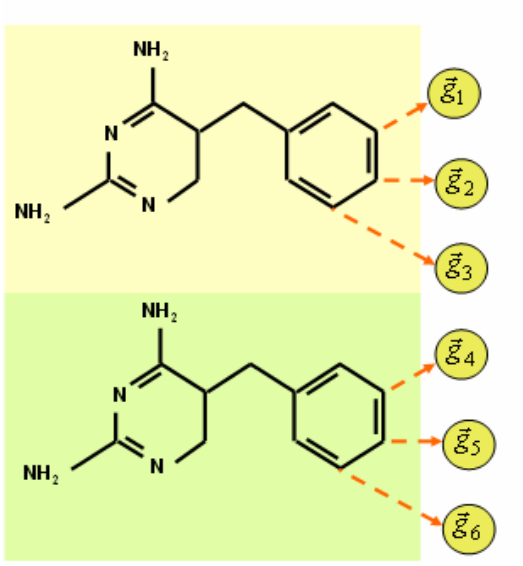

Figure 4.3 Feature granules in the Pyrimidines drug pair

Two types of granular kernel trees (GKTs-1 and GKTs-2) are designed for Pyrimidines which are shown in Figure 4.4 and 4.5. Here the connection node operations during kernel optimization are fixed in order to evaluate GKTs performance with different connection operations. GKTs-1 is a two-layer kernel tree and all granular kernels are fused together by a sum operation. GKTs-2 is three-layer kernel tree and within which each granule pair is represented by a two-layer subtree. Two subtrees of GKTs-2 are combined together by a product operation.

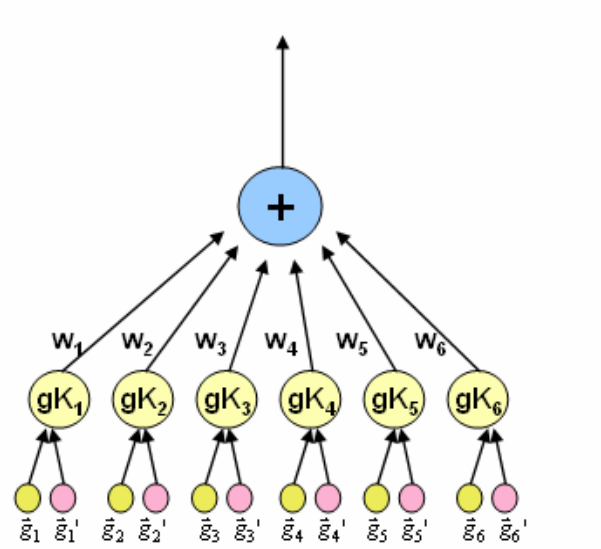

Figure 4.4 GKTs-1

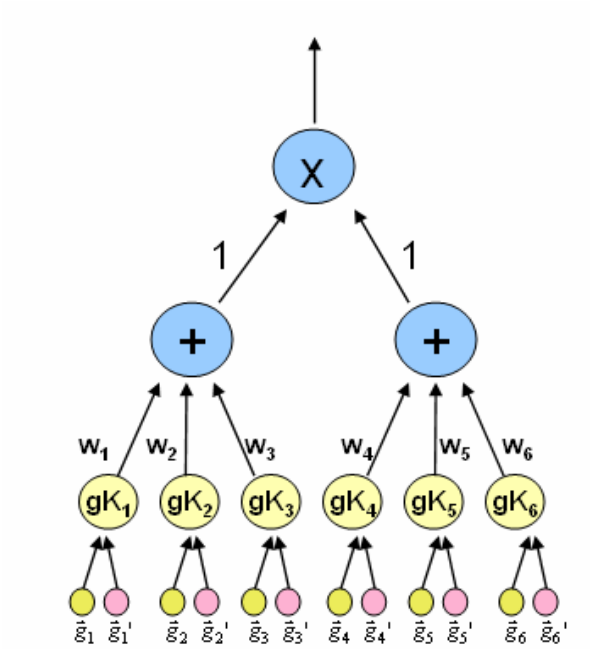

Figure 4.5 GKTs-2

4.2.3 Simulation

Burbidge *et al*. (2001) made a comparison (see Table 4.1) among SVMs, three types of Neural Networks, RBF Network, and Decision Trees on the same data set. Table

4.1 shows that the prediction accuracy of SVMs with the RBF kernel is significantly higher than those of other learning algorithms. In the simulation, we only compare the performance of SVMs with RBF, GKTs and EGKTs.

TABLE 4.1
PERFORMANCE COMPARISON ON THE PYRIMIDINES DATASET (BURBIDGE *ET AL.*, 2001)

Algorithm	Testing accuracy
SVMs+RBF	87.31%
MLP	86.19%
Pruned Neural Network	83.80%
Dynamic Neural Network	85.12%
RBF Network	77.28%
C5.0	81.30%

The RBF kernel functions are chosen as the granular kernels' functions in each GKTs and therefore each granular kernel gK_i has a RBF parameter γ_i. The initial ranges of all RBFs' γ and γ_i are $[0.0001, 1]$.The initial range of regularization parameter C is $[1, 256]$. The probability of crossover is 0.7 and the mutation ratio is 0.5. The range of connection weights is [0.001, 1]. Five-fold cross validation is used on the Pyrimidines training dataset. The population size is set to 500 and the number of generations is set to 30 for both datasets. The software package of SVMs used in the experiments is LibSVM (Chang and Lin, 2001).

Performance of SVMs with three types of kernel machines is shown in Table 4.2. All these systems are optimized using GAs. Table 4.2 shows that SVMs with two GKTs can outperform SVMs with RBF by 3.0% and 3.3% respectively in terms of prediction accuracy on unseen testing dataset. The fitness values of SVMs with GKTs-1 and GKTs-2 are also higher than that of SVMs with RBF. It's also seen that the testing accuracy of SVMs with GKTs-1 is a little bit higher than that of SVMs with GKTs-2.

TABLE 4.2
PERFORMANCE COMPARISON ON THE PYRIMIDINES DATASET

	SVMs+RBF+GAs	SVMs+GKTs-1+GAs	SVMs+GKTs-2+GAs
Fitness	84.5%	86.6%	88.5%
Training accuracy	96.8%	96.8%	98.8%
Testing accuracy	88.4%	91.7%	91.4%

The comparisons between traditional RBF kernels and GKTs are also made with the optimization of GAs. A set of 2000 values is randomly generated from [1, 256] for parameter C and a set of 2000 groups of kernel parameters is randomly generated for each kernel. SVMs are trained and tested with these random parameters. For each dataset, the prediction accuracies of SVMs with three kernels are outlined in Figure 4.6 and each of them is ordered according to C values. From Figure 4.6, it's easy to see that the performance of GKTs is better than that of traditional RBF kernels. Quartiles and mean are also used to summarize each kernel performance in terms of testing accuracy. The results are listed in Table 4.3. Based on the differences of Q1 (25^{th} percentile), Q2 (median), Q3 (75^{th} percentile) and Mean values, we can conclude the GKTs performance is better than the RBF performance by about 2.3%~3.4% on Pyrimidines. Comparing Table 4.2 and Table 4.3, we can see that the testing accuracies of SVMs with both EGKTs are higher than the maximum testing accuracies of SVMs with RBF. It is also found that the testing accuracies of EGKTs can be stabilized at the point of 75^{th} Percentile.

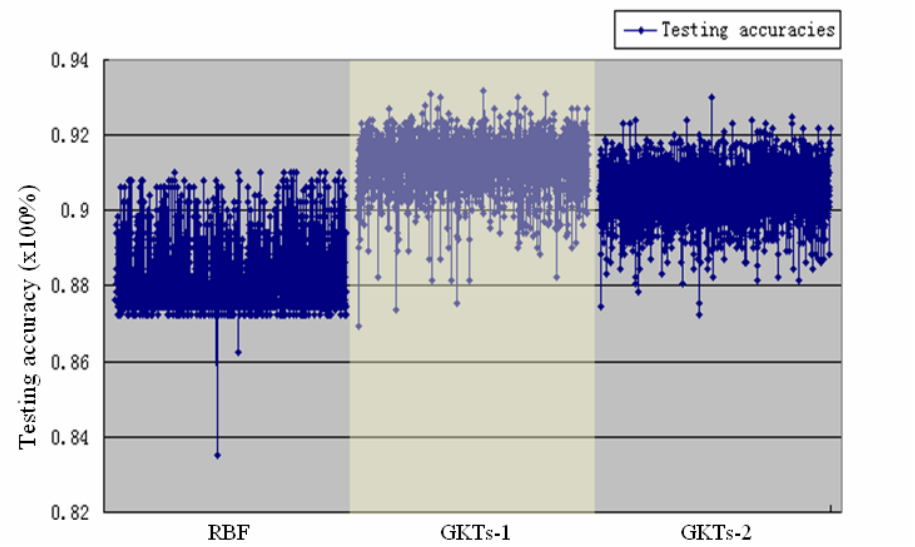

Figure 4.6 Testing accuracies on the Pyrimidines dataset

TABLE 4.3
QUARTILES OF TESTING ACCURACIES ON THE PYRIMIDINES DATASET

	SVMs+RBF	SVMs+GKTs-1	SVMs+GKTs-2
Maximum	91.0%	93.2%	93.0%
75th Percentile	88.4%	91.7%	91.0%
Median	88.0%	91.3%	90.6%
25th Percentile	87.5%	90.9%	90.1%
Minimum	83.5%	87.0%	87.2%
Mean	88.2%	91.2%	90.5%

4.3 Triazines Activity Comparison

4.3.1 Dataset Description

In the Triazines dataset (Hirst *et al.*, 1994b), each compound has 6 possible substitution positions: the positions of R_3 and R_4; if the substituent at R_3 contains a ring itself, then R_3 and R_4 of this third ring; similarly if the substituent at R_4 contains a ring itself, then R_3 and R_4 of this third ring. Ten features are used to characterize each position: the structure branching feature and other 9 features which are the same as those used for each substituent of Pyrimidines. If no substituent locates in a possible position, the

features are indicated by ten -1s. So each vector has 120 features. The structure of Triazines is described in Figure 4.7. We randomly select 60 drugs from the Triazines dataset and then randomly shuffle and split them into 2 parts in the proportion of 5:1 based on drugs of pairs. The size of training set is 2160 and the size of unseen testing set is 1268 for Triazines.

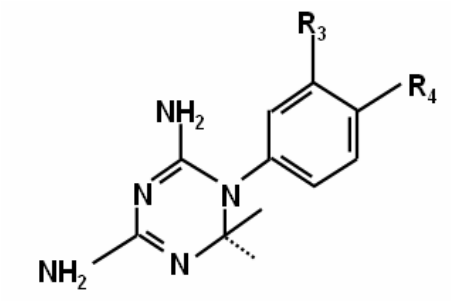

Figure 4.7 Structure of Triazines

4.3.2 Feature Granules and Hierarchical Kernel Design

In the simulation, the input vectors are decomposed according to the possible substituent locations. Each feature granule includes all features of one substituent (see Figure 4.8). So each drug pair of Triazines has 12 feature granules with the size of 10.

We also design two kinds of GKTs for Triazines, which are shown in Figure 4.9 and Figure 4.10. GKTs-3 is a two-layer kernel tree within which each granular kernel's importance is controlled by the outgoing connection weight. GKTs-4 is a three-layer kernel tree within which each drug pair is represented by a two layer subtree. Two subtrees are combined together by a product operation.

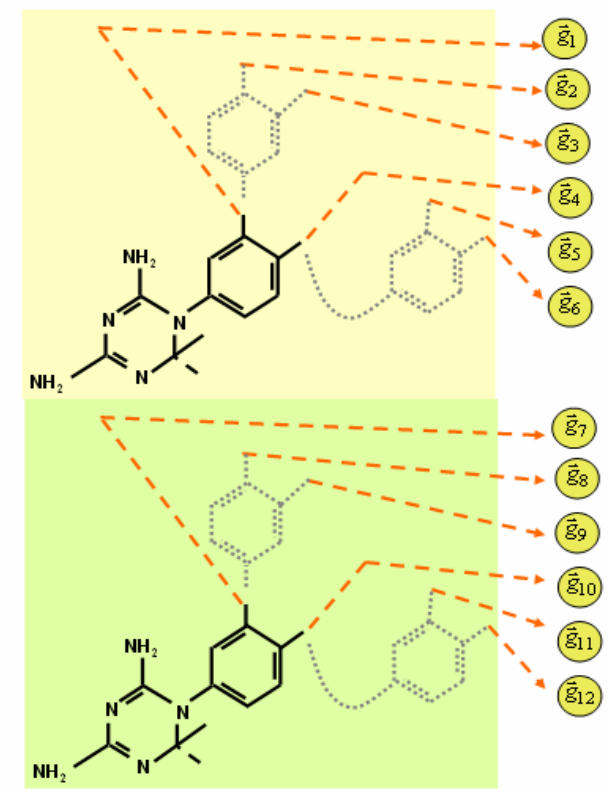

Figure 4.8 Feature granules of the Triazines drug pair

4.3.3 Simulation

RBF kernel functions are also chosen as granular kernels' functions in each GKTs. The initial ranges of all RBFs' γ and γ_i are $[0.0001, 1]$. The initial range of regularization parameter C is $[1, 256]$. The probability of crossover is 0.7 and the mutation ratio is 0.5. The range of connection weights is [0.001, 1]. Eight-fold cross-validation is used on the Triazines training dataset. The population size and the number of generations are also set to 500 and 30 respectively.

Performance of SVMs with three types of kernels is shown in Table 4.4. Also all these kernels are optimized using GAs. Table 4.4 shows that SVMs with two GKTs can

achieve better performance than SVMs with RBF. SVMs with two GKTs can outperform SVMS with RBF by 3.7% and 4.9% respectively in terms of testing accuracy.

TABLE 4.4
PERFORMANCE COMPARISON ON THE TRIAZINES DATASET

	SVMs+RBF+GAs	SVMs+GKTs-3+GAs	SVMs+GKTs-4+GAs
Fitness	73.8%	74.6%	75.8%
Training accuracy	93.4%	97.2%	98.7%
Testing accuracy	79.6%	83.3%	84.5%

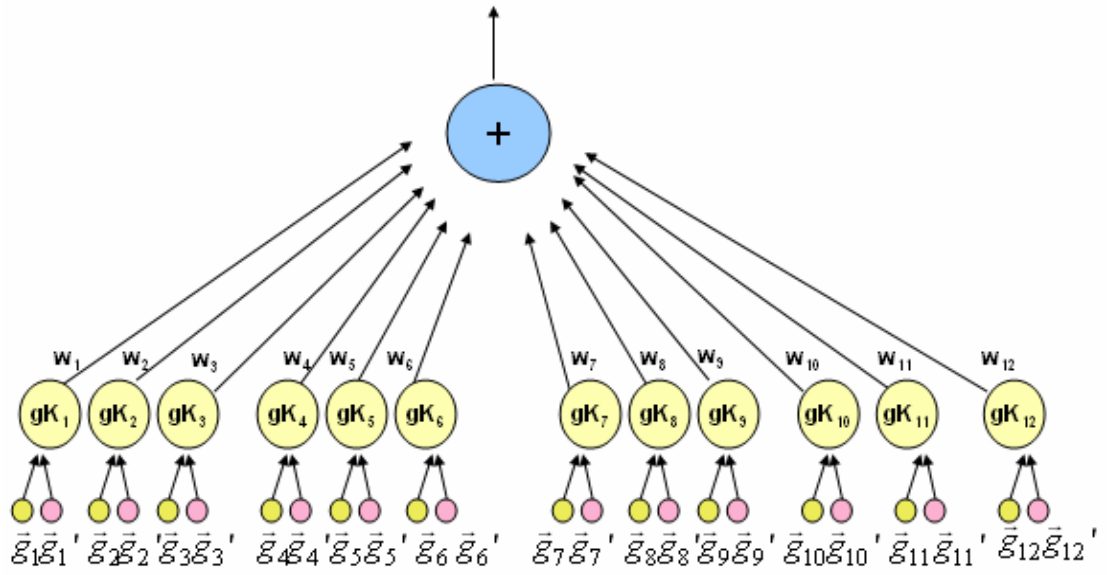

Figure 4.9 GKTs-3

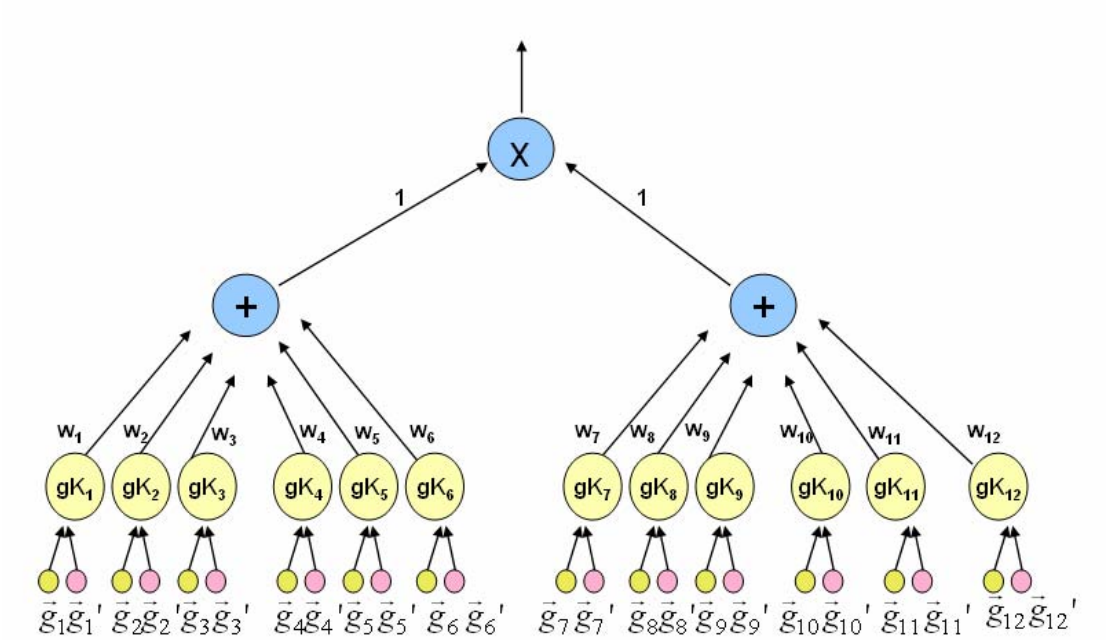

Figure 4.10 GKTs-4

The comparisons between RBF and two kinds of GKTs are also made by using a large number of kernel parameter samples. We randomly generate 2000 C values from [1, 256] and 2000 groups of kernel parameters for each kernel. The prediction accuracies of SVMs with three kinds of kernels are summarized in Figure 4.11 and each of them is ordered with C values. From Figure 4.11, it's easy to see that the performance of SVMs with GKTs is better than those with the RBF kernels. Quartiles and mean are also used to summarize each kernel's performance in terms of testing accuracy and listed in Table 4.5. According to the summaries in Table 4.5, we can conclude the performances of two GKTs are better than those of RBF kernels by about 3.6%~4.5% on the Triazines data set. Comparing Table 4.4 and 4.5, we can also find that the performance of SVMs with EGKTs-3 and EGKTs-4 can be stabilized at Q_3 (75^{th} Percentile) in terms of testing accuracy.

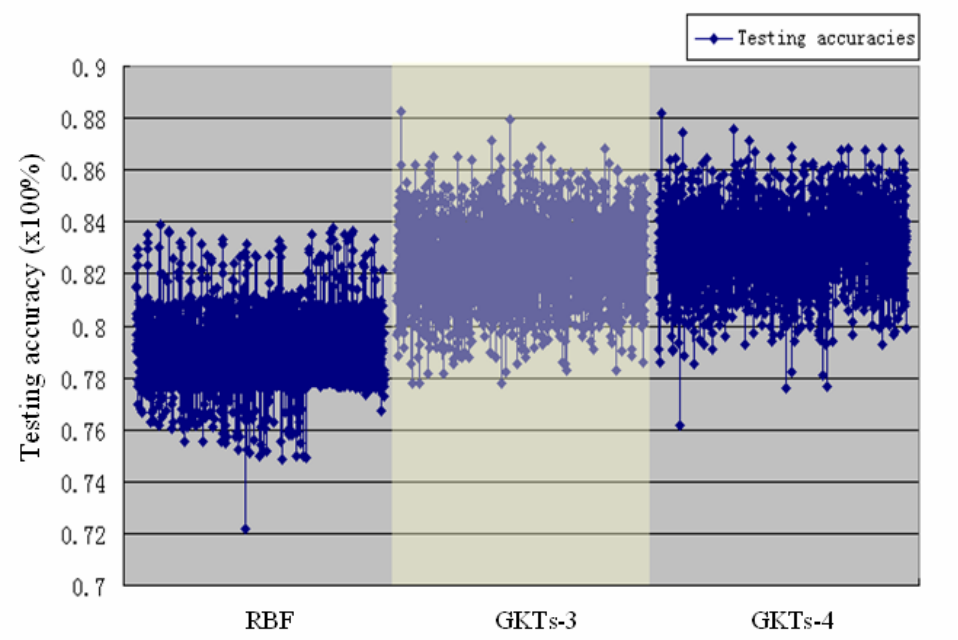

Figure 4.11 Testing accuracies on the Triazines data set

TABLE 4.5
TESTING ACCURACIES ON THE TRIAZINES DATASET

	SVMs+RBF	SVMs+GKTs-1	SVMs+GKTs-2
Maximum	83.9%	88.2%	88.2%
75th Percentile	79.9%	83.7%	84.1%
Median	78.5%	82.6%	83%
25th Percentile	77.9%	81.5%	82%
Minimum	72.2%	77.8%	76.2%
Mean	78.9%	82.6%	83%

CHAPTER 5

GKTSES AND EVKM

In EGKTs, features within an input vector are grouped into feature granules according to the prior domain knowledge. For example, we group the features according to the compound substituent locations in the Pyrimidines activity comparison and the Triazines activity comparison. Sometimes due to the lack of prior knowledge or due to too complicated relations in data, it would be hard to predefine kernel tree structures. Considering such kind of challenging problems, we in this chapter present GKTSES to evolve the structures of GKTs. We redefine the encoding scheme and genetic operation elements to make them more flexible.

5.1 Chromosome

Let P_i denote the population in generation $G_i, i = 1, \ldots, m$ and m is the total number of generations. Each population P_i has p chromosomes c_{ij}, $j = 1, \cdots, p$. Each chromosome c_{ij} has $2q+1$ genes $g_t(c_{ij}), t = 1, \ldots, 2q+1$. In each chromosome, genes $g_{2x-1}(c_{ij}), x = 1, \cdots, q+1$ represent granular kernels and genes $g_{2x}(c_{ij}), x = 1, \cdots, q$ represent sum or product operations. We use $GKTs(c_{ij})$ to represent GKTs configured with genes $g_t(c_{ij}), t = 1, \cdots, 2q+1$. In the algorithm, k-fold cross-validation is used in the fitness evaluation and the roulette wheel method is used in selection too.

5.2 Crossover

In GKTSES, a population of individuals is generated in the first generation. Each individual encodes a granular kernel tree. For example, GKTs-5 and GKTs-6 are two

three-layer GKTs (see Figure 5.1 and 5.2). In GKTs-5 and GKTs-6, each node in the first layer is a granular kernel. Granular kernels are combined together by sum and product connection operations in the second layer and the third layer. Each granular kernel tree is encoded into a chromosome. For example, GKTs-5 and GKTs-6 are encoded in chromosomes c_1 and c_2 (see Figure 5.3) respectively. In the first generation, features are first randomly shuffled and then feature granules are randomly generated. Granular kernels are preselected from the candidate kernel set. Some traditional kernels such as RBF kernels and polynomial kernels can be chosen as granular kernels. In practice, we choose RBF kernels as granular kernels and each feature is a feature granule. Finally granular kernel parameters and kernel connection operations are also randomly generated for each individual.

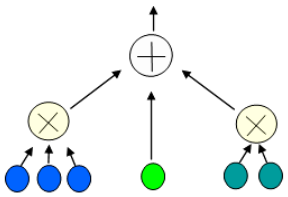

Figure 5.1 GKTs-5

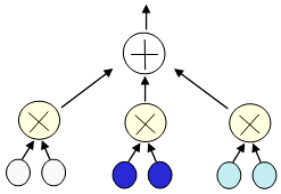

Figure 5.2 GKTs-6

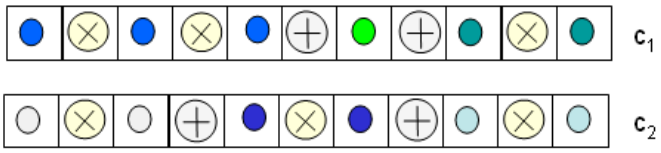

Figure 5.3 Chromosomes used encode GKTs-5 and GKTs-6

In crossover, two GKTs are first selected from current generation as parents and then a crossover point is randomly selected for separating parents GKTs. Subtrees of two GKTs are exchanged at the crossover point to generate two new GKTs. For example, chromosomes c_1 and c_2 may do crossover at point d_2 to generate two new chromosomes (see Figure 5.4). This is equivalent to that GKTs-5 and GKTs-6 exchange their right subtrees (see Figure 5.5). In Figure 5.5, GKTs-7 and GKTs-8 have the same structures as their parents respectively. Here, GKTs-7 is encoded in chromosome c_3 and GKTs-8 is encoded in chromosome c_4.

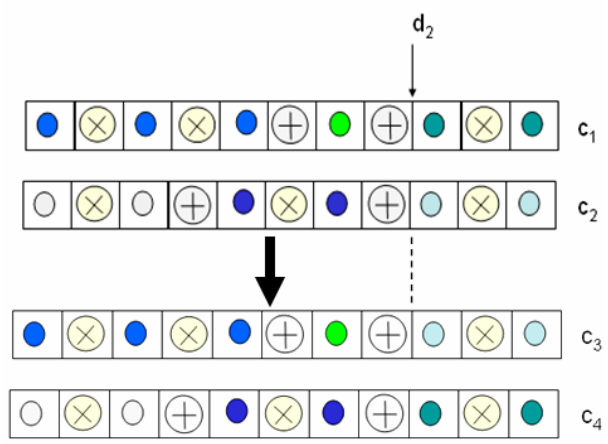

Figure 5.4 Chromosomes c_3 and c_4 generated from c_1 and c_2

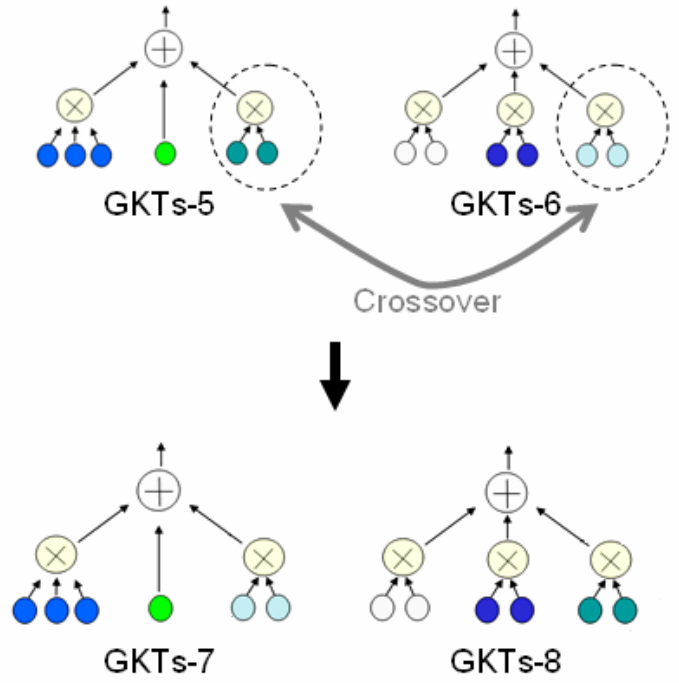

Figure 5.5 GKTs-7 and GKTs-8 generated using crossover operation

GKTSES can also generate GKTs with different tree structures from their parents using the crossover operation. In Figure 5.6, GKTs-5 and GKTs-6 do crossover at point d_1 to generate two new granular kernel trees, GKTs-9 and GKTs-10, which have different tree structures from their parents. The equivalent operation on chromosomes is shown in Figure 5.7.

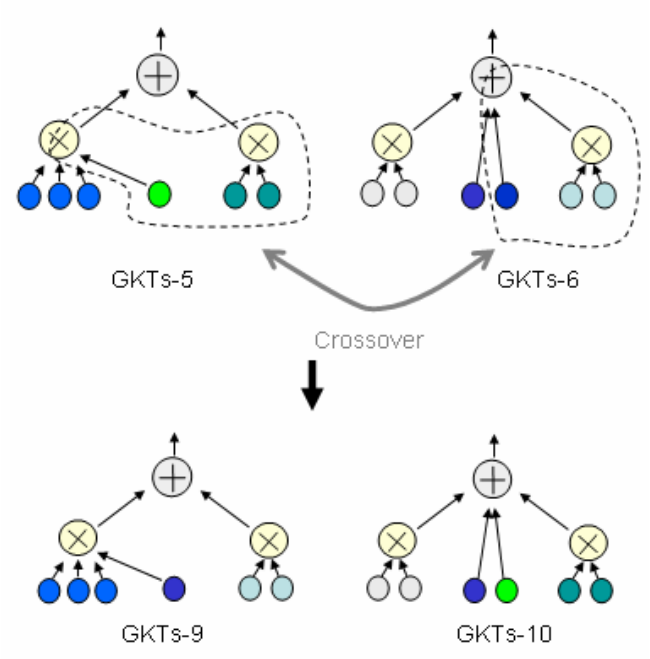

Figure 5.6 GKTs-9 and GKTs-10 generated using crossover operation

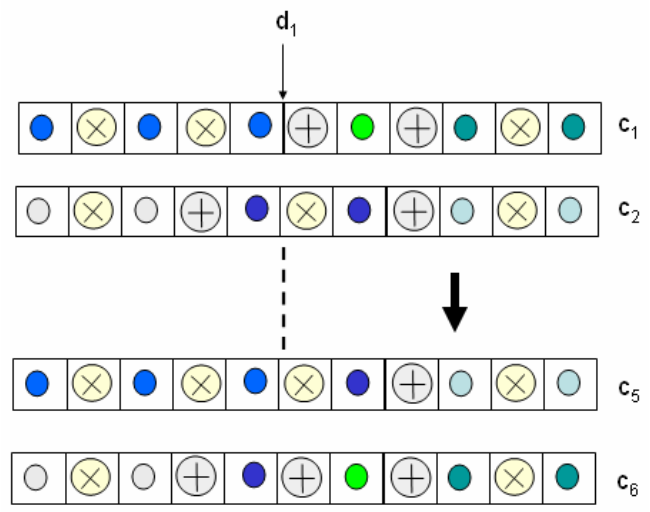

Figure 5.7 Chromosomes c_5 and c_6 generated from c_1 and c_2

5.3 Mutation

In mutation, some genes of one chromosome are selected with a specified probability. The values of selected genes are replaced by random values. In the

implementation, only connection operation genes are selected to do mutation. Figure 5.8 shows an example of mutation. The new chromosome c_7 is generated by changing the eighth gene of chromosome c_1 from sum operation to product operation, which is equivalent to transforming GKTs-5 to GKTs-11 (see Figure 5.9).

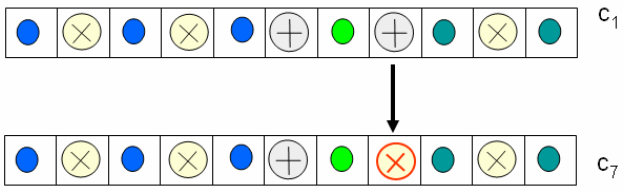

Figure 5.8 Chromosomes c_7 generated from c_1 using mutation

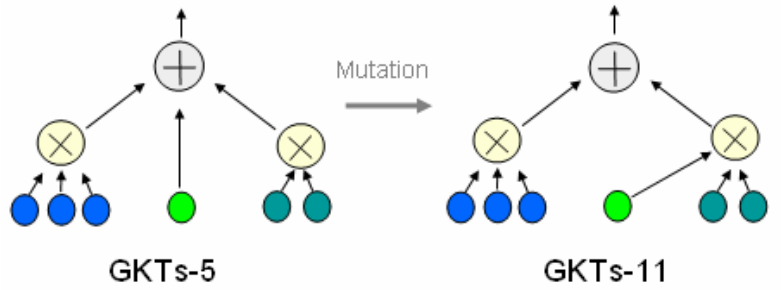

Figure 5.9 GKTs-11 generated from GKTs-5 using mutation

The system architecture of GKTSES is shown in Figure 5.10. In the system, the regularization parameter C of SVMs is also optimized together with GKTs.

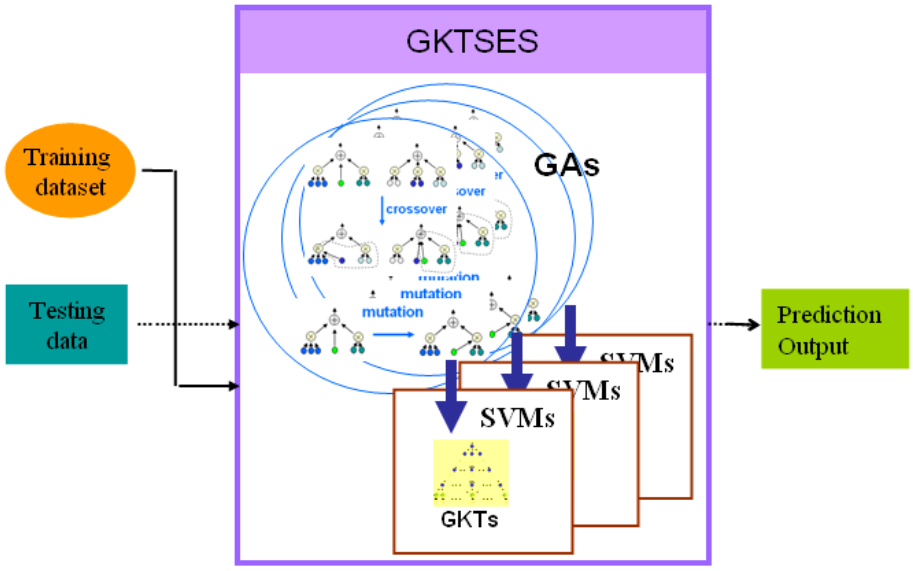

Figure 5.10 Architecture of GKTSES

5.4 Simulation

5.4.1 Dataset Description and Simulation Setup

In the simulation, GKTSES is used for Cyclooxygenase-2 inhibitor activity comparison. The dataset of Cyclooxygenase-2 inhibitors (Kauffman and Jurs, 2001) includes 314 compounds, and 153 of them are active and 161 are inactive. 109 features are selected to describe each compound. Each feature's absolute value is scaled to the range [0, 1]. The point of log (IC_{50}) units is set to 2.5 to discriminate active compounds from inactive compounds. The dataset is randomly shuffled and evenly split into 3 mutually exclusive parts. Each time we choose one part as the unseen testing set and the other twos as the training set. Three-fold cross-validation is used in the fitness evaluation. RBF kernel is also chosen as each granular kernel function. The range of γ is set to $[0.00001, 1]$ and the range of regularization parameter C is set to $[1, 256]$. The probability of crossover is 0.8 and the mutation ratio is 0.2. The population size is set to

300 and the number of generations is set to 50. The software package of SVMs used in the experiments is LibSVM (Chang and Lin, 2001). In the first generation, sum operations are generated with the probability of 0.5 in each individual.

5.4.2 Simulation Results

Table 5.1 shows the prediction accuracies of SVMs with GKTSES and GAs based SVMs with RBF in average. From Table 5.1, we can see that SVMs+GKTSES can outperform SVMs+GAs+RBF by 3.5% in terms of testing accuracy and 2% in terms of fitness, although the latter system shows the higher accuracy in training. The results of three evaluations CV-1, CV-2 and CV-3 are summarized in Table 5.2. From Table 5.2, we can see that the testing accuracies of SVMs+GKTSES are always higher than those of SVMs+GAs+RBF by about 2.9% ~ 3.9% in all three evaluations. The fitness values of SVMs+GKTSES are higher than those of SVMs+GAs+RBF by about 1% ~ 3.4%. The prediction accuracies in three evaluations are also visualized in Figure 5.11. According to the comparison, we can say that SVMs+GKTSES are more reliable than SVMs+GAs+RBF.

However, the comparison among three evaluations also shows that significant deviations exist in the testing accuracies for both systems. This problem could happen when the system is not stable or the data set is not iid (independent and identically distributed). To solve this problem, we present a voting-scheme-based evolutionary kernel machine called evolutionary voting kernel machine (EVKM) in the next section.

TABLE 5.1
PREDICTION ACCURACIES IN AVERAGE ON THE CYCLOOXYGENASE-2 DATASET

	Fitness	Training accuracy	Testing accuracy
SVMs+GAs+RBF	81.9%	94.4%	72.3%
SVMs+GKTSES	83.9%	89.7%	75.8%

TABLE 5.2
RESULTS OF THREE EVALUATIONS ON THE CYCLOOXYGENASE-2 DATASET

	CV-1		CV-2		CV-3	
	SVMs+GAs+ RBF	GKTSES	GAs-RBF-SVMs	GKTSES	GAs-RBF-SVMs	GKTSES
Fitness	83.7%	87.1%	80.4%	82.3%	81.4%	82.4%
Training accuracy	90.9%	92.3%	97.1%	88.0%	95.2%	88.6%
Testing accuracy	64.8%	68.6%	78.1%	81%	74%	77.9%
#Support vector	116	91	107	111	144	111

5.5 EVKM

In EVKM, the kernel's evolving procedure and the genetic operations such as selection, crossover and mutation are the same as those in GKTSES. The difference is in the fitness evaluation. In EVKM, the decision is made by several weighted SVMs instead of a single SVM. Simulation results show that EVKM is more stable than SVMs in the Cyclooxygenase-2 inhibitor activity comparisons.

5.5.1 Voting Scheme

In each $GKTs(c_{ij})$ evaluation, the training data set $\tilde{S} = \{(\vec{x}_1, y_1), \cdots, (\vec{x}_l, \vec{y}_l)\}$ is separated into k mutually exclusive subsets $\tilde{S}_v, v = 1, \ldots, k$. SVMs are trained on every subset $\tilde{S}_v$ with $GKTs(c_{ij})$ and then k SVMs decision functions $d_v(\vec{x})$ are generated.

$$d_v(\vec{x}) = \sum_{i_v} \alpha_{i_v} y_{i_v} GKTs\left(\vec{x}, \vec{x}_{i_v}\right) + b_v \tag{5.1}$$

where $0 < \alpha_{i_v} \le C$, C is the regularization parameter, $\vec{x}_{i_v}$ are support vectors and b_v is the threshold for the v^{th} SVMs.

The number of correctly classified data in each training is calculated and the weighted voting decision function $d(\vec{x})$ is defined as follows:

$$d(\vec{x}) = \sum_{v} c_v d_v(\vec{x}) \tag{5.2}$$

In Equation (5.2), c_v is a cost factor which is either the accuracy of positive class of the v[th] SVMs if $d_v(\vec{x})$ is positive, or the accuracy of negative class of if $d_v(\vec{x})$ is negative. The training dataset is then predicted by $d(\vec{x})$ and the positive class accuracy (t_v^+) and the negative class accuracy (t_v^-) are calculated respectively. Finally the optimized *GKTs*' is generated and the decision function $d'(\vec{x})$ is calculated for the unseen testing set prediction.

$$d'(\vec{x}) == \sum_v c'_v \left(\sum_{i_v} \alpha_{i_v} y_{i_v} GKTs'\left(\vec{x}, \vec{x}_{i_v}\right) + b_v \right) \tag{5.3}$$

where the cost factor c'_v is either t_v^+ or t_v^-.

5.5.2 Simulation Results

In the simulation, the experimental setup is the same as that in Section 5.4. Table 5.3 shows the performance of EVKMs with the RBF kernel and GKTSES on the Cyclooxygenase-2 dataset. From Table 5.3, we can see that in three evaluations, the testing accuracies of EVKM+GKTSES are always higher than those of EVKM+RBF by about 1.9% ~ 2.9%. Table 5.4 summarizes the testing accuracies of EVKMs and GAs-based SVMs with different kernels in three evaluations. From Table 5.4, we can find that the average testing accuracy of each EVKM is a little bit higher than that of GAs-based SVMs. While comparing the standard deviations of testing accuracies, we can find that the standard deviations of EVKMs are much lower than those of GAs-based SVMs. It means that EVKMs are more stable than GAs-based SVMs.

TABLE 5.3
PREDICTION RESULTS OF EVKMS IN THREE EVALUATIONS ON THE CYCLOOXYGENASE-2 DATASET

	CV-1		CV-2		CV-3	
Method	EVKM+ RBF	EVKM+ GKTSES	EVKM+ RBF	EVKM+ GKTSES	EVKM+ RBF	EVKM+ GKTSES
Training accuracy	91.4%	90.9%	90%	85.2%	90.1 %	85.6%
Testing accuracy	71.3%	74.2%	76.8%	78.7%	74.2%	77%

TABLE 5.4
TESTING ACCURACIES IN AVERAGE AND STANDARD DEVIATION ON THE CYCLOOXYGENASE-2 DATASET

	SVMs+GAs+RBF	SVMs+GKTSES	EVKM+RBF	EVKM+GKTSES
Testing accuracy	72.3%	75.8%	74.1%	76.6%
Standard deviation of Testing accuracy	6.8%	6.5%	2.8%	2.3%

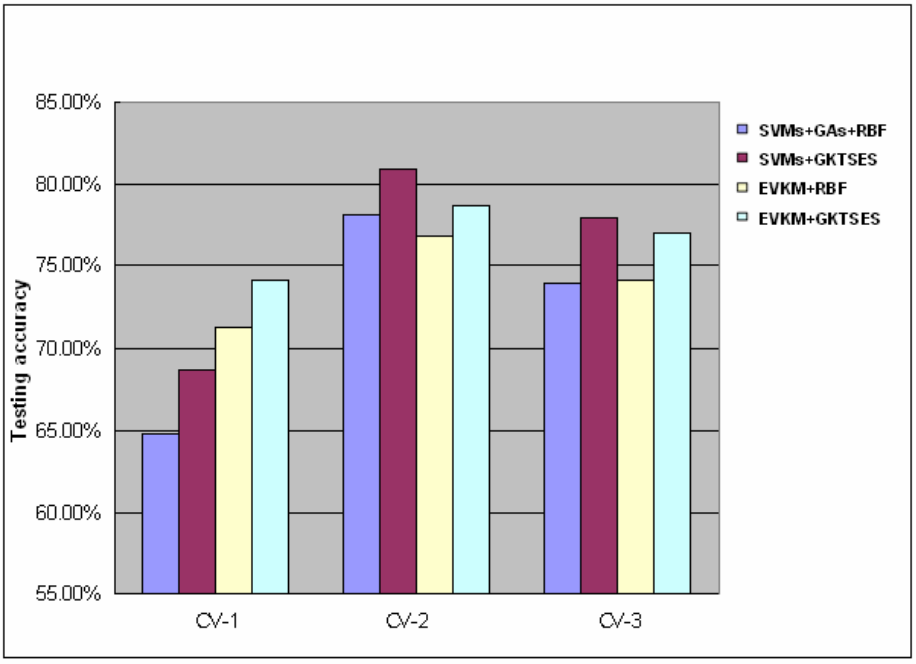

Figure 5.11 Testing accuracies of EVKMs in three evaluations

CHAPTER 6

EGKTS WITH PARALLEL COMPUTING

6.1 SVMs with Parallel Computing

Some parallel algorithms were designed for SVMs in the literature. Graf *et al*. (2005) developed a kind of parallel SVMs called Cascade of SVMs in a distributed environment. In Cascade of SVMs, the smaller optimizations are first solved independently, and then the partial results are combined and filtered again until the global optimum is reached. Convergence of the algorithm to the global optimum is guaranteed with multiple passes through the Cascade. The Gradient Projection Method (GPM) based parallel SVMs (Zanghirati and Zanni, 2003; Serafini *et al*., 2005) were also proposed. In GPM based SVMs, the decomposition technique is used to split the quadratic programming (QP) problem into smaller QP sub-problems. These sub-problems are solved by GPM in a parallel way. An asynchronous parallel ES was designed for the SVMs model selection by Runarsson and Sigurdsson (2004). The algorithm was implemented on a multi-processor computer using C++ and the standard Posix threads.

6.2 Parallel GAs Models

Parallel GAs (Cantú-Paz, 1998; Adamidis, 1994; Lin *et al.*, 1997) have been well studied in recent several years. Typically there are three types of parallel GAs models: (1) single population master-slave model, (2) single population fine-grained model and (3) multiple population coarse-grained model.

In the single population master-slave GAs, there is only one single population, which is similar to the simple GAs. The master node stores the population, does genetic

operations such as selection, crossover, and mutation, and distributes individuals to the slave nodes. Once the slave nodes evaluate the fitness of the individuals, they send the fitness values back to the master node. This kind of parallel model does not affect the behavior of GAs, since all individuals in the population are considered during the genetic operations. In the single population fine-grained model, there is a single population which is structured spatially. Neighborhoods may be overlapped among all the individuals. Selection and crossover can only happen on a small neighborhood. This kind of model is suitable for massively parallel computers. The multiple population coarse-grained model is more complicated and the big difference from the first two models is that it has several subpopulations which may exchange individuals.

6.3 Parallel EGKTs

In EGKTs, all parameters to be optimized are independent. The operations are the same in training each SVMs model. So it's very suitable to use the single population master-slave model to parallelize EGKTs and further to speed up the training of EGKTs based SVMs. In the parallelization of EGKTs, one processor is chosen as the master node, which stores the population of GKTs, does selection, crossover and mutation on these GKTs, and then distributes the parameters of GKTs to slave nodes. Each single SVMs model is trained and evaluated on one slave node. We implement the parallel system with MPICH in a disk-shared and memory-distributed Linux cluster environment.

The model architecture of SVMs with the parallel EGKTs is shown in Figure 6.1 and the system architecture is shown in Figure 6.2. This parallel system has some characteristics. Firstly, this is a global GAs-SVMs system, since all evaluations and operations are performed on the entire population. Secondly, the implementation is easy,

clear, practical, and especially suitable for the SVMs model selection and the training speedup of SVMs with EGKTs. The QP decomposition method can be used to speed up the GKTs selection too. However, if the training dataset is large, the communication costs for transferring sub-QP meta-results will be very high. In our system, the time for QP calculation in each SVM model is longer than that for the genetic operations of GAs, which generally has different magnitude. In our system, only parameters and fitness values need to be transferred between the master and the slaves. So the communication costs are small. Thirdly, the system can be easily moved to the large distributed computing environment.

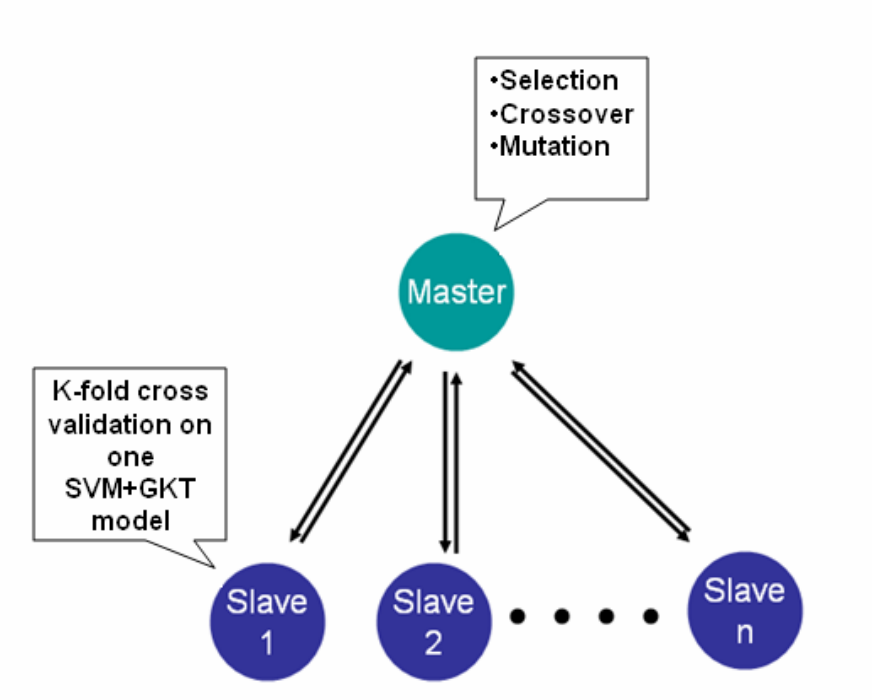

Figure 6.1 SVMs with the parallel EGKTs

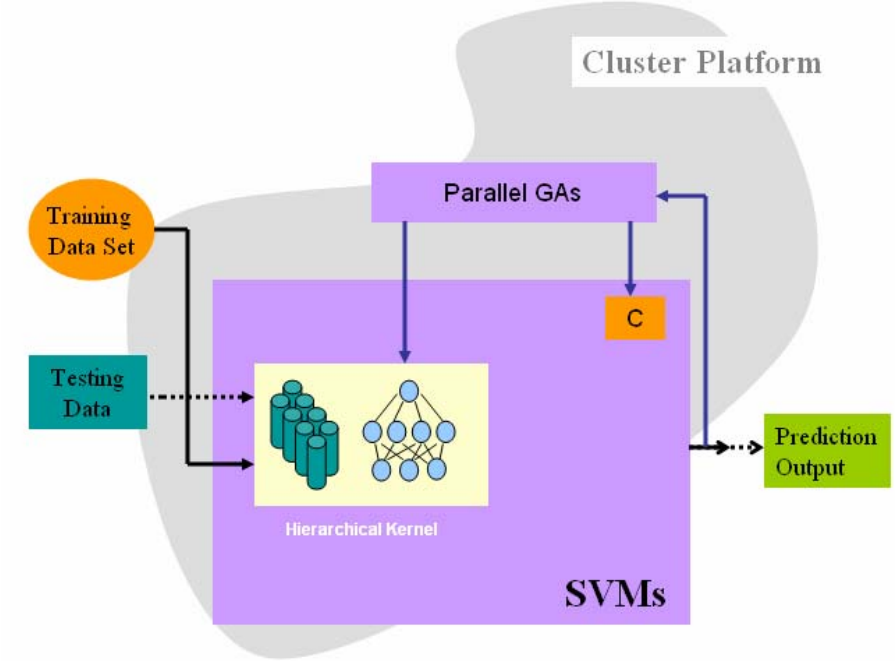

Figure 6.2 System architecture of SVMs with the parallel EGKTs

6.4 Simulation

The Cyclooxygenase-2 dataset and the RBF kernel are used in the simulation. Each GKT is defined by simply grouping all features into one feature granule. In GAs, the size of population is set to 300 and the number of generations is set to 50. The parallel system is tested on the GSU's Biocluster, which is a Beowulf cluster with four head nodes and 40 computing nodes. Each computing node has two 3.0 GHz Intel Xeon CPUs with 2.0 GB memory. In the simulation, each computing node will run two SVM models. It means that each computing node is equivalent to two slave nodes of the parallel system.

The running time of the parallel system on the cluster platform is shown in Figure 6.3. From Figure 6.3, we find that the parallel system can significantly reduce the optimization time of GKTs when the number of slave nodes is larger than 3. The

simulation results shown in Figure 6.4 are also illustrated that our parallel method can significantly speed up the training of SVMs+EGKTs by a factor of 10 with 14 nodes.

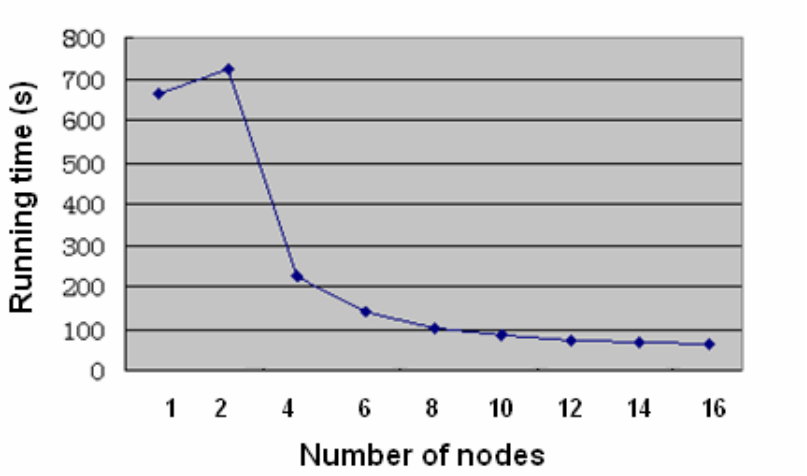

Figure 6.3 Running time of the parallel system

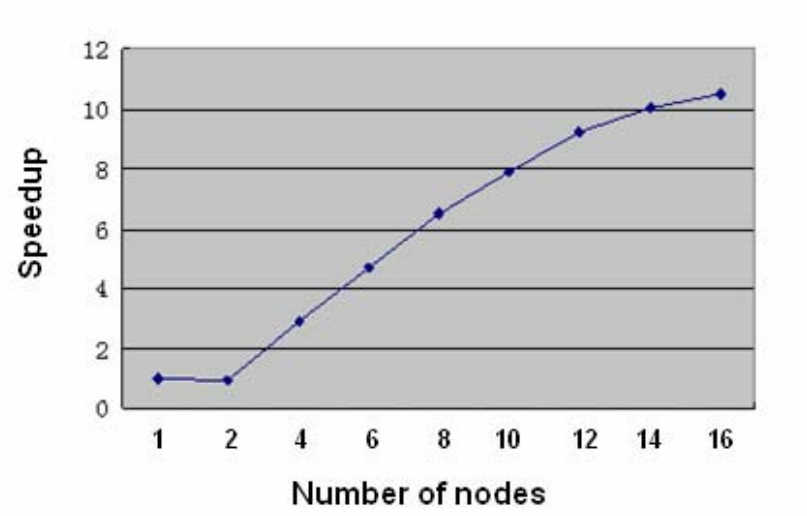

Figure 6.4 Speedup of the parallel system

CHAPTER 7

MEB-SVM

7.1 Related Works

7.1.1 Chunking and Decomposition

Typically, a QP solver is used to find support vectors in SVMs training. Due to the fact that the QP algorithm requires large memory for storing the kernel data matrix, the traditional QP based SVMs are not suitable for large scale data classification.

An alternative way is to split a large optimization task into a series of smaller ones. Chunking (Boser *et al*., 1992) and decomposition (Osuna *et al*., 1997) are two kinds of methods working in this way. In the chunking, an initial random sub dataset (working set with an arbitrary size) is optimized by the QP solver and the related support vectors are found. The non support vectors are then discarded from the working set and new data points violating the optimality conditions are added. The QP solver is used on the working set again. The iteration continues until the whole optimization task is solved. Different from the chunking algorithm, the decomposition method works on a sub dataset with the fixed size. In each iteration, only the Lagrange multipliers on the working set are updated and other Lagrange multipliers are kept fixed. A special decomposition algorithm is Sequential Minimal Optimization (SMO) (Platt, 1998). SMO is an analytical approach, which uses a set of heuristics and works on the working set of the size of two without using any optimization package. Besides the chunking and decomposition, there are other methods proposed to speed up SVMs training too.

7.1.2 CB-SVM

CB-SVM (Yu *et al*., 2003) is particularly designed for the large scale data classification with a limited amount of system resources. The key idea of CB-SVM is to employ the hierarchical clustering technique to find the finer description close to the classification boundary and the coarser description far away from the boundary.

In CB-SVM, a hierarchical micro-clustering algorithm is used to scan the data set and two Clustering Feature (CF) trees are generated on positive data and negative data separately. In each CF tree, a hierarchical representation is used to summarize the data distribution. The nodes in a lower level are summarized and represented by their parent node. The CF trees can effectively summarize the distribution of the entire dataset and capture the spherical shapes of hierarchical clusters. An SVM is first trained on the centroids of the root nodes of two CF trees and a rough classification boundary is generated. The data summaries of two CF trees close to the boundary are then declustered into the lower levels. The declustered children nodes are added to the training set and another SVM is constructed on the centroids of the nodes in the training set. The algorithm repeats such kind of process down through the trees until to the leaf level.

CB-SVM is scalable very well if the number of features is small. However CB-SVM can only be used to solve the linear classification problems in the input space. It is difficult to summarize and represent the data distribution hierarchically in the new feature space transformed by the nonlinear kernels. How to adapt CB-SVM to solve the nonlinear problems with the nonlinear kernels is still a big issue.

7.1.3 CVM

In CVM (Tsang *et al*., 2005), the SVM optimization problem is first transformed to the MEB problem and an approximate optimal solution is obtained on a sub dataset called core-set. The idea behind CVM is that the point furthest away from the current center is added into the ball incrementally. CVM can be used to classify very large data sets with nonlinear kernels.

The algorithm initializes the core-set with two points and calculates the initial center and radius of the ball. If the left data points fall in the ball, the algorithm terminates. Otherwise, the furthest point from the center of the ball is added to the core set, and then a new ball is calculated on the core-set using the SMO algorithm. The distances of the left points to the updated center are measured again. Such kind of process is repeated until no point falls outside the ball. To reduce the required time for measuring the distance from the left data points to the center of the ball, 59 sampled points are used instead of all the data points.

7.1.4 LSVM

LSVM (Mangasarian and Musicant, 2001) is a modified SVM using the squared slack variables to measure the loss. In LSVM, the classification problem is reformulated as an unconstrained optimization problem. The problem is then solved using a method based on the Sherman-Morrison-Woodbury formula, which only requires solving the systems of linear equalities. LSVM can be very fast on the data sets with the relatively low dimensionality. Due to the requirements of storing and inverting a $n \times n$ matrix (where n is the number of features), it is not suitable to apply LSVM to the tasks with the

high dimensionality. Also, a nonlinear kernel is not easy to be employed in LSVM, because the Sherman-Morrison-Woodbury identity is used under the condition that the inner product terms of the kernel are explicitly known, which in general are not satisfied.

7.1.5 PSVM

PSVM (Fung and Mangasarian, 2001) classifies data based on proximity to one of two parallel planes that are pushed apart maximally. Similar to LSVM, PSVM uses an equality to replace the inequality constraint and the Sherman-Morrison-Woodbury formula for matrix inversion. These changes make PSVM avoid the QP calculation and build the classification model with linear computations only. PSVM is a very fast algorithm with no significant loss of accuracy especially when the number of data features is much less than the number of training data. However the algorithm requires large space to store the kernel matrix. The incremental training methods (Fung and Mangasarian, 2002) can help PSVM reduce the space requirement.

7.1.6 HeroSVM

HeroSVM (Dong *et al*., 2003; Dong *et al*., 2005) is a fast SVM training algorithm designed for classifying the data set of huge size with thousands of classes. The key idea behind HeroSVM is using block diagonal matrices to approximate the original kernel matrix. In HeroSVM, the original problem is decomposed into hundreds of sub problems and most of the nonsupport vectors are quickly removed so that the final sequential optimization could be finished in a short time. Some other techniques such as kernel caching, digest and shrinking are also integrated into the algorithm to speed up the training process.

7.1.7 Active Learning

In SVMs with active learning (Schohn and Cohn, 2000), a simple but efficient heuristic is applied to estimate the change of the expected error heuristically when an example is added. The heuristic tries to narrow the existing hyperplane margin as maximal as possible by assuming that examples lying along the hyperplane separate the space most quickly on average. In active learning, SVMs are required to be trained once only. It's shown that for a given number of training examples, SVMs with active learning can provide better generalization performance than SVMs trained on the randomly selected examples.

7.2 MEB-SVM

As mentioned in Chapter 1, the powers of SVMs in solving the nonlinear classification problems are provided by the nonlinear kernels, which implicitly transform the data from the input spaces into some high dimensional feature spaces. However, it is difficult for us to analyze the data distribution in the feature space transformed by kernels.

As mentioned in Section 2.1 of Chapter 2, an important characteristic of SVMs is that the separating hyperplane can be built with the support vectors only and the data lying out of the margin of the hyperplane can be removed safely. The key idea behind MEB-SVM is using MEBs to remove most of data lying outside the hyperplane margin. The MEB can only provide the boundary information of a dataset, however such kind of information is enough to help SVMs determine the separating hyperplane quickly.

In MEB-SVM, the boundary of each class data set is first measured by several MEBs, and then the data within each MEB are removed. An SVM is finally trained on a smaller dataset which only contains the data on the MEBs boundaries. Because the objective conditions are loose and data points (error data) out of an MEB can be tolerated very well, the time for measuring the MEB is much shorter than that for finding the separating hyperplane in the binary classification. An example of MEB-SVM for data classification is shown in Figure 7.1 - 7.4. Figure 7.1 shows the feature space transformed by a RBF kernel. Measuring MEBs on each class data set and deleting data within MEBs are shown in Figure 7.2 and Figure 7.3 respectively. Figure 7.4 shows a hyperplane found by an SVM classifier, which is trained on the final training set.

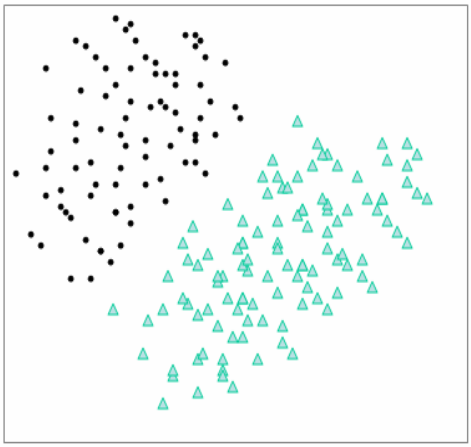

Figure 7.1 Data in a new feature space transformed by a RBF kernel

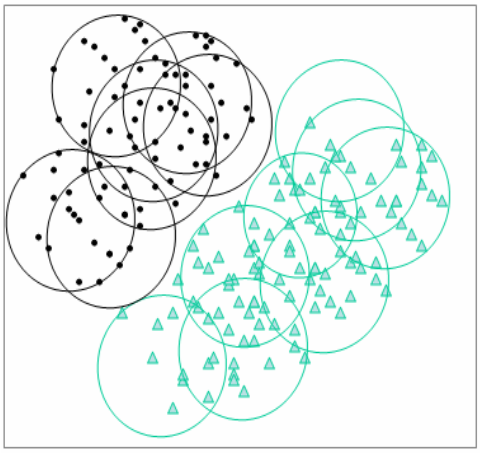

Figure 7.2 MEBs measured on each class data set

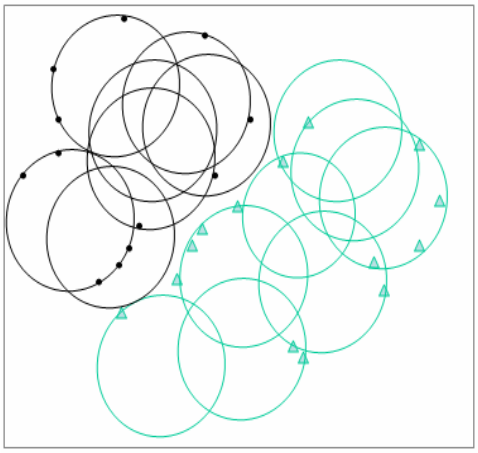

Figure 7.3 Data reduction

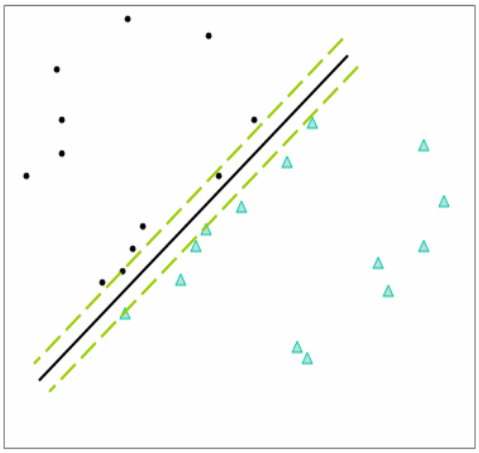

Figure 7.4 Hyperplane found by an SVM classifier on the reduced data set

The detailed description of the MEB-SVM algorithm is given as follows. For a binary nonlinear classification problem, let $S = \{(\vec{x}_1, y_1), \cdots, (\vec{x}_l, \vec{y}_l)\}$ represent a training dataset, where $\vec{x}_i \in R^n$, $i = 1, \cdots, l$ are vectors and $y_i \in \{-1, +1\}, i = 1, \cdots, l$ are class labels associated to $\vec{x}_i$. Let $S^+ = \{(\vec{x}_1, +1), \cdots, (\vec{x}_{k_1}, +1)\}$ represent positive data set and $S^- = \{(\vec{x}_1, -1), \cdots, (\vec{x}_{k_2}, -1)\}$ represent negative data set, where $S^+ \bigcup S^- = S$ and $k_1 + k_2 = l$. In the MEB-SVM algorithm (see Figure 7.5), operations will be executed repeatedly until the size of training set is equal to or smaller than a constant T. In the loop, the positive data set S^+ will be split into m^+ equal subsets $E^+_{j_1}$ and the negative data set S^- will be split into m^- equal subsets $E^-_{j_2}$ randomly, where $j_1 = 1, \ldots, m^+$ and $j_2 = 1, \ldots, m^-$. RBF based MEBs are calculated on each subset ($E^+_{j_1}$ and $E^-_{j_2}$) and a new training set is generated which includes all data located on each MEB's boundary. Once the training set is small enough, an SVM classifier will be trained on it with the same RBF kernel as used in MEBs. In Figure 7.5, A^+ and A^- are two temporary datasets.

Input: S^+, S^-, m^+, m^- and T
Output: a trained SVM classifier

Algorithm:
1. $S := S^+ \cup S^-$
2. While $|S| \geq T$
   {
   Empty A^+ and A^-; /* A^+ and A^- are temporary datasets*/
   Split S^+ into m^+ equal subsets $E^+_{j_1}$ randomly;
   Split S^- into m^- equal subsets $E^-_{j_2}$ randomly;
   For j_1 from 1 to m^+
   Measure $E^+_{j_1}$'s MEB boundary ;
   Put the data located on the boundary of $E^+_{j_1}$ into A^+;
   For j_2 from 1 to m^-
   Measure $E^-_{j_2}$'s MEB boundary;
   Put the data located on the boundary of $E^-_{j_2}$ into A^-;
   $S^+ := A^+$;
   $S^- := A^-$;
   $S := S^+ \cup S^-$
   }
3. Train an SVM classifier on S;
4. Return a trained SVM classifier;

Figure 7.5 The MEB-SVM algorithm

7.3 Simulation

The KDDCUP-99 intrusion detection dataset and other two artificial datasets are used in the simulation. MEB-SVM is implemented based on LibSVM and its Tools (Chang and Lin, 2001). In the simulation, we don't consider the I/O time for reading and writing files. The constant T is always set to 10000.

7.3.1 Performance Metrics

Besides accuracy, some other performance metrics implemented in the PERF Software (which can be downloaded at http://kodiak.cs.cornell.edu/kddcup/software.html) are also used to measure the MEB-SVM performance. These metrics are calculated based on the entries in the confusion matrix. Figure 7.6 shows the confusion matrix for a binary classification problem.

Actual\ Predicted	Positive	Negative
Positive	a	b
Negative	c	d

Figure 7.6 Confusion matrix

In the confusion matrix, a is the number of true positive predictive examples, b is the number of false negative predictive examples, c is the number of false positive predictive examples, and d is the number of true negative predictive examples. The performance metrics used in the simulations are defined as follows:

- **Accuracy (ACC)** The ratio between the number of the true predictive examples and the total number of examples.

$$ACC = \frac{a+d}{a+b+c+d}. \tag{7.1}$$

- **Positive Predictive Value (PPV)** The ratio between the number of the true positive predictive examples and the positive predictive examples.

$$PPV = \frac{a}{a+c} \tag{7.2}$$

- **Negative Predictive Value (NPV)** The ratio between the number of the true negative predictive examples and the negative predictive examples.

$$NPV = \frac{d}{b+d} \tag{7.3}$$

- **Sensitivity (SEN)** The proportion of the actual positive examples that are predicted as positive.

$$SEN = \frac{a}{a+b} \tag{7.4}$$

- **Specificity (SPE)** The proportion of the actual negative examples that are predicted as negative.

$$SPE = \frac{d}{c+d} \tag{7.5}$$

- **Precision (PRE)** The proportion of the positive predictive examples that are predicted as positive, which is equal to the positive predictive value.

$$PRE = \frac{a}{a+c} \tag{7.6}$$

- **Recall (REC)** The proportion of the actual positive examples that are predicted as positive, which is same as the sensitivity.

$$REC = \frac{a}{a+b} \tag{7.7}$$

- **Area under the ROC curve (AUC)** ROC curve is a 2-D plot used to show the relationship between the prediction and the truth. The area under the ROC curve is commonly used as a quantitative value to evaluate the prediction. An AUC example is shown in Figure 7.7

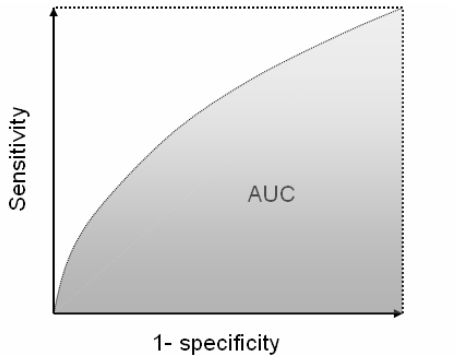

Figure 7.7 The area under the ROC curve

7.3.2 Network Intrusion Detection

With the growth of Internet, network intrusion detection is becoming vital in the network security as a lot of malicious actions attempt to compromise the resources of networks and information. Data mining methods (Lee and Stolfo, 1998; Bloedorn *et al*., 2001; Barbara *et al*., 2001; Dokas, 2002) are also widely used for the network intrusion detection with the tremendous increase of novel network attacks. In network intrusion detection, the basic task is to build a model, which can distinguish between intrusion connections and normal connections.

7.3.2.1 Dataset Description

In the simulation, the KDDCUP-99 dataset (which is available at http://kdd.ics.uci.edu//databases/kddcup99/kddcup99.html) is a real world dataset used for the third international knowledge discovery and data mining tools competition. The dataset is a standard set of tcpdump data, which are generated in a military network environment and used to simulate the connections in a wide variety of intrusions. The training set includes 4,898,431 connection records. The number of positive examples is 3,925,650 and the number of negative examples is 972,781. The testing set includes

311,029 records. In the dataset, each record item has 32 continuous features and 9 discrete features. All these features are listed in Tables 7.1 - 7.4, which are copied from the KDDCUP-99 website.

TABLE 7.1
BASIC FEATURES OF INDIVIDUAL TCP CONNECTIONS

Feature name	Feature description	Type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of “wrong” fragments	continuous
urgent	number of urgent packets	continuous

TABLE 7.2
CONTENT FEATURES WITHIN A CONNECTION SUGGESTED BY DOMAIN KNOWLEDGE

Feature name	Feature description	Type
hot	number of “hot” indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of “compromised” conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if “su root” command attempted; 0 otherwise	discrete
num_root	number of “root” accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the “hot” list; 0 otherwise	discrete
is_guest_login	1 if the login is a “guest” login; 0 otherwise	discrete

TABLE 7.3
TRAFFIC FEATURES COMPUTED USING A TWO-SECOND TIME WINDOW

Feature name	Feature description	Type
count	number of connections to the same host as the current connection in the past two seconds	continuous
serror_rate*	% of connections that have “SYN” errors	continuous
rerror_rate*	% of connections that have “REJ” errors	continuous
same_srv_rate*	% of connections to the same service	continuous
diff_srv_rate*	% of connections to different services	continuous
srv_count*	number of connections to the same service as the current connection in the past two seconds	continuous
srv_serror_rate**	% of connections that have “SYN” errors	continuous
srv_rerror_rate**	% of connections that have “REJ” errors	continuous
srv_diff_host_rate**	% of connections to different hosts	continuous

*These features refer to these same-host connections. **These features refer to these same-service connections.

TABLE 7.4
OTHER FEATURES WITHOUT DESCRIPTIONS

Feature name	Type
dst_host_count	continuous
dst_host_srv_count	continuous
dst_host_same_srv_rate	continuous
dst_host_diff_srv_rate	continuous
dst_host_same_src_port_rate	continuous
dst_host_srv_diff_host_rate	continuous
dst_host_serror_rate	continuous
dst_host_srv_serror_rate	continuous
dst_host_rerror_rate	continuous
dst_host_srv_rerror_rate	continuous

In the simulation, the continuous features are normalized to the range 0~1. Non-numerical feature are represented by non-negative integer numbers. For example, for feature 2, symbols “tcp”, “udp” and “icmp” are represented by 0, 1 and 2. The 20^{th} feature (num_outbound_cmds) is removed so that each processed record item contains 40 features.

7.3.2.2 Simulation Results

RBF kernel is used in the MEB-SVM evaluation. In the SVM training, the regularization parameter C is set to 16.65 and RBF’s γ is 0.01, the positive class weight is 2.307, and the negative class weight is 1. In MEB calculation, m^+ is set to 20 and m^- is set to 10. We conduct the benchmark results of MEB-SVM with regard to the random sampling methods, the active learning based SVM, CB-SVM, and CVM in terms of prediction accuracy, running time, and the number of support vectors. MEB-SVM is tested on a 2.8 GHz PC with the 512 MB RAM memory. Other methods were evaluated on the different machines and the related results are copied from the published papers (Yu *et al*., 2003; Tsang *et al*., 2005). The random sampling methods, the active learning based SVM and CB-SVM were evaluated on an 800MHz P-3 machine with the 906 MB RAM memory. CVM was evaluated on a 3.2GHz P-4 machine with the 2GB RAM memory.

Table 7.5 shows the performance comparison between MEB-SVM and other methods on the KDDCUP-99 dataset. In the table, other processing time means the data sampling time in the random sampling methods, the clustering time in CB-SVM, and the MEB calculating time in MEB-SVM. From Table 7.5, we can see that MEB-SVM finishes training in 250 seconds, which is faster than that of other methods except CVM. Although it is not suitable to compare the training time directly (since these algorithms are evaluated on the different machine), we can still conclude that MEB-SVM is really a fast learning algorithm. The testing accuracy of MEB-SVM can reach 93.38%, which is higher than that of other methods except CVM. The AUC value of MEB-SVM is 96.4%, which is lower than that of CVM by 1.3%. Table 7.5 also summarizes the number of the final training data and the number of support vectors. MEB-SVM only needs a small dataset in the SVM training. Other performance metrics are also used to evaluation MEB-SVM and the results are shown in Table 7.6.

TABLE 7.5
PERFORMANCE COMPARISON ON THE KDDCUP-99 DATASET

Method		# Testing Error	AUC	# Selected Training Data	# Support Vectors	Running Time (second)	
						SVM Training Time	Other Processing Time
Random Sampling	0.001%	25,713	-	-	-	0.000991	500.02
	0.01%	25,030	-	-	-	0.120689	502.59
	0.1%	25,531	-	-	-	6.944	504.54
	1%	25,700	-	-	-	604.54	509.19
	5%	25,587	-	-	-	15827.3	524.31
ASVM		21,634	-	307	-	94192.213	-
CB-SVM		20,938	-	2893	-	7.639	4745.483
CVM		19,513	0.977	55	20	1.4	
MEB-SVM*		20,601	0.964	173	114	1	249

* The training accuracy of MEB-SVM is 95.7%.

TABLE 7.6
OTHER PERFORMANCE EVALUATIONS OF MEB-SVM
ON THE KDD CUP-99 DATASET

Max_Accuarcy threshold*	0.000405
AUC	0.96355
ACC	0.93379
PPV	0.98946
NPV	0.76235
SEN	0.92766
SPC	0.95915
PRE	0.98946
REC	0.92766

* The prediction threshold that maximizes the accuracy

7.3.3 Evaluation on the Ring Norm Dataset

7.3.3.1 Dataset Description

Ring norm dataset is the two-class artificial dataset generated using Leo Breiman's algorithm (Breiman, 1996). Each data item has 20 dimensions. One class data are generated from a multivariate normal distribution with the zero mean and covariance matrix equal 4 times the identity matrix. The other class data are generated from the multivariate normal distribution with the unit covariance matrix and mean of 2/sqrt(20) along each dimension. For this dataset, the theoretical expected classification accuracy is 98.76%, which can be computed by using the Fukunaga and Krile's method (Fukunaga and Krile, 1969).

7.3.3.2 Simulation Results

A training set RN_1 with 3 million ring norm data examples and a testing set with 90,000 examples are generated and used for the performance comparison between MEB-SVM and CVM. Both MEB-SVM and CVM are trained on the same P-4 3.0GHz machine with 2.0GB RAM (the available memory is about 400MB). For CVM, the regularization parameter C is set to 100 and RBF's γ is set to 0.0195. For MEB-SVM, C

is set to 0.0062, RBF's γ is set to 0.01, m^+ is set to 20, and m^- is set to 20. Both positive class weight and negative class weight are set to 1.

The simulation results are shown in Table 7.7. From Table 7.7, we can see that MEB-SVM has much better performance than CVM in terms of number of the selected training data, number of support vectors, testing error and running time. The prediction accuracy of MEB-SVM on the testing dataset can reach 98.44%, which is almost same as the theoretical expected accuracy. The MEB-SVM training can finish in 117 seconds, which is 55 times as fast as CVM.

TABLE 7.7
PERFORMANCE COMPARISON BETWEEN MEB-SVM AND CVM
ON THE RING NORM DATASET WITH 3 MILLIONS EXAMPLES

	# Selected Training Data	# Support Vectors	Testing Error (%)	Running Time (second)	
				SVM Training Time	Other Processing Time
CVM	17338	15703	2.41%	6495	
MEB-SVM	5273	1612	1.56%	2	115

To make a comparison between MEB-SVM and HeroSVM, the other training set RN_2 with 100,000,000 ring norm data examples and a testing set with 3 million examples are generated and used in the evaluation. MEB-SVM is evaluated on a P-4 3.0GHz machine with the 2.0GB RAM memory (the available memory is about 400MB). Both positive class weight and negative class weight are set to 1. C is set to 0.00806, RBF's γ is set to 0.01, and both m^+ and m^- are set to 500.

The simulation results are shown in Table 7.8. Because the number of selected training data (=139136) is still too large after the first loop of MEB calculation, MEBs are measured again on the selected training data. In the second loop of MEB calculation, both m^+ and m^- are set to 10. The simulation results of HeroSVM listed in Table 7.8 were reported by Dong *et al*. (2005). HeroSVM was evaluated on a P-4 1.7GHz machine

with the 1.5GB SDRAM memory. CVM Toolbox used by Tsang *et al*. (2005) fails to handle the data set with 100 million examples. From Table 7.8, we can see that MEB-SVM can finish training in 4013 seconds on this very large dataset, which is faster than HeroSVM by 53191 seconds. It means that MEB-SVM is almost 13 times faster than HeroSVM. The prediction accuracy of MEB-SVM on this dataset is 98.44%. The testing error of HeroSVM is lower than that of MEB-SVM. In HeroSVM, the final decision is made based on the voting among k SVMs, each of which is built on the support vector sets with the size of about 7900.

TABLE 7.8
PERFORMANCE COMPARISON BETWEEN MEB-SVM AND HEROSVM
ON THE RING NORM DATASET WITH 100 MILLIONS EXAMPLES

	# Selected Training Data	# Support Vectors	Testing Error (%)	Running Time (second)	
				SVM Training Time	Other Processing Time
HeroSVM	5,807,025	7900k	1.23%	57204	
MEB-SVM	139136	-	1.56%	-	3994
	5697	1221		2	17

7.3.4 Evaluation on the Normally Distributed Clustered (NDC) Datasets

7.3.4.1 Dataset Description

Two datasets with different linear separabilities are generated using the Normally Distributed Clusters (NDC) generator (Musicant, 1998). The NDC generator can generate a series of random centers for multivariate normal distributions. In the data generation, a fraction of data points from each center is generated first, and then a separating hyperplane is randomly generated. Each center is marked with a class label based on the separating plane. The generator then randomly generates the points from the distributions. The linear inseparability of data can be increased by increasing variances of distributions.

7.3.4.2 Simulation Results

One dataset NDC_1 has linear separability of around 69.7%. The other dataset NDC_2 has linear separability of around 90.9%. Each training set consists of 2 million examples with 10 features and each testing set has 200,000 examples. MEB-SVM algorithms are trained on P-4 3.0GHz machine with 2.0GB RAM. For the dataset NDC_1, C is set to 1500 and RBF's γ is set to 0.01. For the data set NDC_2, C is set to 50 and RBF's γ is set to 0.01. Both m^+ and m^- are set to 10 in each training. The simulation results are shown in Table 7.9.

TABLE 7.9
PERFORMANCE COMPARISON ON THE NDC DATASETS

Method	Data Set	Theoretical Linear Separability Theoretical Accuracy (%)	Training Accuracy (%)	Testing Accuracy (%)	# Selected Training Data	# Support Vectors	Running Time (second)	
							SVM Training Time	Other Processing Time
MEB-SVM	NDC_1	69.7%	83.17%	83.06%	62	17	1	4
LSVM	NDC_3	70%	69.80%	69.44%	-	-	655.6	
PSVM	NDC_3	70%	69.84%	69.52%	-	-	20.6	
MEB-SVM	NDC_2	90.9%	98.83%	98.82%	95	36	1	7
LSVM	NDC_4	90%	90.86%	91.23%	-	-	658.5	
PSVM	NDC_4	90%	90.8%	91.13%	-	-	20.8	

The simulation results of LSVM and PSVM listed in Table 7.9 were reported by Fung and Mangasarian (2001). Dataset NDC_3 has linear separability of around 70% and the dataset NDC_4 has linear separability of around 90%. LSVM and PSVM were evaluated on a Pentium 400MHz machine with a maximum of 2 GB of memory.

On the dataset with the linear separability of around 70%, MEB-SVM only needs 5 seconds for training and can achieve the prediction accuracy of 83.1%, which is significantly higher than those of PSVM and LSVM by about 13.5%~13.6%. On the

dataset with the linear separability of around 90%~91%, MEB-SVM can finish training in 8 seconds and achieve the prediction accuracy of 98.8%, which is higher than those of PSVM and LSVM by about 7.6%~7.6%. Although these three algorithms are evaluated on the different machines (LSVM and PSVM are evaluated on a Pentium 400Mhz machine with a maximum of 2 GB of memory), the running time of MEB-SVM is still competitive.

TABLE 7.10
OTHER PERFORMANCE EVALUATIONS OF MEB-SVM ON THE NDC DATASETS

Data Set	NDC_1	NDC_2
Max_Accuarcy threshold	-0.246091	0.018746
AUC	0.924	0.999
ACC	0.837	0.988
PPV	0.763	0.974
NPV	0.909	0.999
SEN	0.89	0.999
SPC	0.798	0.98
PRE	0.763	0.974
REC	0.89	0.986

MEB-SVM is also evaluated using other performance metrics on the NDC_1 and NDC_2 datasets and the results are summarized in Table 7.10. MEB-SVM can achieve high quality metrics on both datasets. On the NDC_2 dataset, MEB-SVM can even achieve 99.9% in the AUC, NPV, and SEN evaluations. These results demonstrate that MEB-SVM is a powerful kernel-based classification algorithm in large scale data mining.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this dissertation, kernel design and SVMs training speedup were well studied. Powerful and flexible kernel trees called Evolutionary Granular Kernel Trees (EGKTs) were designed for the complex data, such as biological data and chemical data. EGKTs can effectively incorporate prior domain knowledge. The simulation results in the Pyrimidines and Triazines activity comparisons demonstrated that GKTs and EGKTs can effectively improve the prediction accuracies of SVMs comparing to the GAs based SVMs with the RBF kernel. Considering the case of lack of prior knowledge, Granular Kernel Tree Structure Evolving System (GKTSES) was developed to evolve the structures of Granular Kernel Trees (GKTs). To reduce the prediction deviation of GKTSES, a voting scheme was proposed for the decision making of SVMs. The simulation on the Cyclooxygenase-2 dataset showed that SVMs with GKTSES can achieve higher accuracy in testing than the GAs based SVMs with RBF. The simulation also showed that the voting scheme can significantly reduce the prediction deviation of SVMs+GKTSES from 6.5% to 2.3%.

To speed up the EGKTs optimization, we parallelized EGKTs. Simulation results showed that the parallel method can significantly speed up the training of SVMs+EGKTs by a factor of 10 with 14 nodes. To help SVMs challenge large-scale data mining, we presented MEB-SVM. MEB-SVM can quickly and significantly reduce the training data and shorten the SVMs training. The conducted benchmark results demonstrated that MEB-SVM is a very competitive classification algorithm with regard to the random

sampling methods, active learning based SVM, CB-SVM, CVM, HeroSVM, LSVM and PSVM in terms of prediction accuracy, running time, and the number of support vectors.

8.2 Future Work

Our GKTSES system is a kind of intelligent system, which selects granular kernels from a candidate kernel set. In the future, we will build a kernel library, which will contain more popular kernels such as tree kernels, string kernels and graph kernels. The system will choose suitable kernels for SVMs according to the characteristics of problems. Furthermore, multi-classification approaches will be introduced into this intelligent system, since many data classification problems are multi-classification problems. In the system, ensemble learning methods and other machine learning algorithms will also be used for classification.

8.2.1 Ensemble Methods

In ensemble learning, several approaches rather than a single approach are integrated to enhance the performance of the final classifier. An ensemble classifier is expected to have better performance than the individual base classifiers. Bagging and boosting are two kinds of popular ensemble learning algorithms.

8.2.1.1 Bagging

Bagging (Bauer and Kohavi, 1999) is a kind of ensemble method by using voting/averaging to combine predictions. In bagging, instead of building models on one training set of size n, several training sets of size n are sampled with replacement from original training set. We then build a classifier on each training set and combine the

predictions of classifiers by using voting/averaging. In bagging, each model receives equal weight. Bagging works since it reduces variance by voting/averaging.

8.2.1.2 Boosting

Boosting (Bauer and Kohavi, 1999) is another kind of ensemble method for combining multiple classifiers. Boosting iteratively learns a model from a weighted data set, evaluates it, reweights data, and finally produces a set of weighted models. In testing, the data class is predicted with the highest weight.

8.2.2 Multi-classification Approaches

SVMs, as a kind of binary classifiers, cannot directly be used for multi-classification. When using SVMs to solve a multi-classification problem, a common way (Hsu and Lin, 2002) is first decomposing the multi-classification problem into a series of binary classification problems, building SVMs models for each of these binary classification problems, and then combining outputs of SVMs for the multiple-class prediction.

8.2.2.1 One-versus-rest

In the one-versus-rest approach, k binary SVM models are built for k classes separately in the training. The ith SVM model is built with all samples in the ith class with positive labels and all other samples with negative labels. Once an unknown sample needs to be classified, it is first predicted by these SVM models, and then classified into the class corresponding to the SVM model with the highest output value.

.

8.2.2.2 One-versus-rest Voting

Park and Kanehisa (2003) presented a kind of one-versus-rest voting approach for protein subcellular location prediction. Besides the amino acid composition, the amino acid pair and gapped amino acid pair compositions (Chou, 1999) were also used to generate data vectors. Based on five different types of compositions (amino acids, amino acid pairs, one gapped amino acid pairs, two gapped amino acid pairs, and three gapped amino acid pairs), five groups of 12 SVMs models are built. In testing, a query protein is first classified by each group of SVMs models, which is the same as that in the one-versus-one approach. Then the final decision is made by voting among the outputs of 5 groups of SVMs models. In the voting, if the query protein is classified to a same location five, four or three times, or it is classified to the same location twice and another three different locations once, it will finally be decided as belonging to this location. When the query protein is classified to two locations twice, either of these two locations could be chosen as the final decision.

8.2.2.3 One-versus-one Voting

In the one-versus-one voting approach, for a k class problem, $k(k\text{-}1)/2$ SVMs models are built where each model is built on data from two different classes. When an unknown sample needs to be classified, it is predicted by all SVMs models and $k(k\text{-}1)/2$ prediction results are generated. In the prediction with the model built on data from the ith and jth classes, if this sample is classified to the ith class, the vote for the ith class will be added by one. Otherwise, one will be added to the vote for the jth class. The sample is finally classified to the class with the maximum votes. In case that two or more classes

receive the same maximum votes, one of the classes with the maximum votes will be chosen randomly as the final decision.

8.2.2.4 Directed Acyclic Graph SVM

Directed Acyclic Graph SVM (DAGSVM) (Platt *et al.*, 2000) is another approach to combine outputs of SVMs for the multiple-class prediction. Similar to the one-versus-one voting approach, this approach constructs models on data from any two different classes. For a k class problem, $k(k\text{-}1)/2$ SVMs models are built. Different from the one-versus-one voting approach, a rooted binary directed acyclic graph (DAG) with $k(k\text{-}1)/2$ internal nodes and k leaves is used in the prediction. When classifying an unknown sample, the prediction starts at the root node, repeatedly moves to either the left or right child of a node based on the node's decision, and finally reaches a leaf node which indicates the predicted class.

8.2.3 New Intelligent System Framework

The new intelligent system will include four libraries: kernel library, machine learning algorithm library, ensemble method library, and multi-classification approach library. The kernel library will only be used for kernel methods. The machine learning algorithm library will contain some popular machine learning algorithms such as SVMs, decision trees and neural networks. The system will pick up algorithms from this library for the unit prediction tasks according to GAs' initialization. Once the parameters are evaluated on each individual classifier, ensemble methods may be selected from the ensemble method library to improve the prediction accuracies of unit classifiers. For multi-classification problems, some approaches such as one-versus-rest and one-versus-

one voting could be chosen to solve the tasks. Many possible choices may be combined together, which are optimized by GAs. The order of applying ensemble learning and multi-classification approaches may be changed and repeated. The system will run in a cluster environment. The system architecture is shown in Figure 8.1.

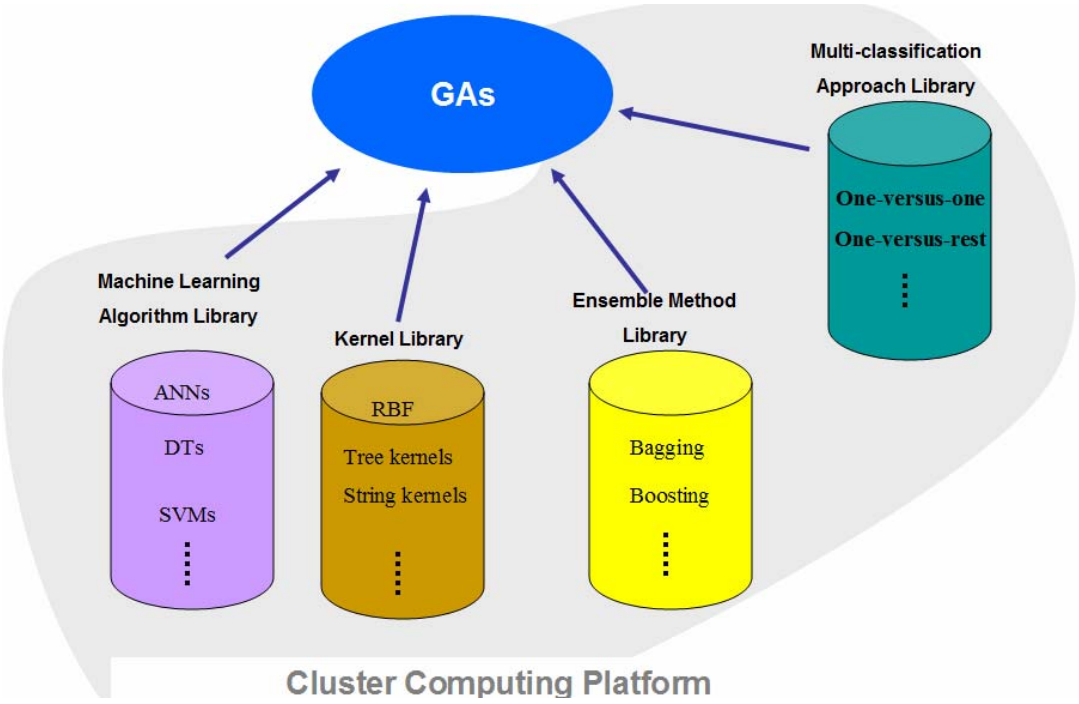

Figure 8.1 New intelligent system architecture

REFERENCES

P. Adamidis, “Review of parallel genetic algorithms bibliography,” Internal Technical Report, Aristotle University of Thessaloniki, 1994.

D. Barbara, N. Wu and S. Jajodia, “Detecting Novel Network Intrusions Using Bayes Estimators,” First SIAM Conference on Data Mining, Chicago, 2001.

E. Bauer and R. Kohavi, “An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants,” Machine Learning, vol. 36, no.1-2, pp.105-139, 1999.

C. Berg, J.P.R. Christensen and P. Ressel, Harmonic Analysis on Semigroups-Theory of Positive Definite and Ralated Functions, Springer-Verlag, 1984.

E. Bloedorn, A.D. Christiansen, W. Hill, C. Skorupka, L.M. Talbot and J. Tivel, “Data Mining for Network Intrusion Detection: How to Get Started,” MITRE Technical Report, 2001.

B.E. Boser, I.M. Guyon and V.N. Vapnik, “A training algorithm for optimal margin classifiers,” In Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, D. Haussler, eds., pp. 144-152, ACM Press, 1992.

L. Breiman, “Bias, variance, and arcing classifiers,” Tec. Report 460, Statistics department, University of California, April 1996. Available: ftp://ftp.cs.toronto.edu/pub/neuron/delve/data/tarfiles/ringnorm.tar.gz.

R. Burbidge, M. Trotter, B. Buxton and S. Holden, “Drug Design by Machine Learning: Support Vector Machines for Pharmaceutical Data Analysis,” Computers and Chemistry, vol. 26, no. 1, pp. 4-15, 2001.

E. Cantú-Paz, “A survey of parallel genetic algorithms,” Calculateurs Paralleles, vol. 10, no. 2, Paris: Hermes, 1998.

C.-C. Chang and C.-J. Lin, “LIBSVM: a library for support vector machines,” 2001.

O. Chapelle and V. N. Vapnik, “Model selection for support vector machines,” Advances in Neural Information Processing Systems 12, S.A. Solla, T.K. Leen and K.-R. Muller, Ed., MIT Press, 2000.

O. Chapelle, V.N. Vapnik, O. Bousquet and S. Mukherjee, “Choosing Multiple Parameters for Support Vector Machines,” Machine Learning, vol. 46, no. 1, pp. 131-159, 2002.

K.C. Chou, “Using pair-coupled amino acid composition to predict protein secondary structure content,” Journal of Protein Chemistry, vol. 18, no. 4, pp. 473-480, 1999.

M. Collins and N. Duffy, “Convolution kernels for natural language,” Advances in Neural Information Processing Systems 14, T.G. Dietterich, S. Becker and Z. Ghahramani, eds., MIT Press, 2001.

C. Cortes and V.N. Vapnik, “Support-vector networks,” Machine Learning, vol. 20, pp. 273-297, 1995.

N. Cristianini and J. Shawe-Taylor, An Introduction to Support Vector Machines and other kernel-based learning methods, Cambridge University Press, 1999.

P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava and P. Tan, “Data Mining for Network Intrusion Detection,” Proc. NSF Workshop on Next Generation Data Mining, Baltimore, 2002.

J.X. Dong, A. Krzyzak and C.Y. Suen, “A Fast Parallel Optimization for Training Support Vector Machine,” Proceedings of 3rd International Conference on Machine Learning and Data Mining, P. Perner and A. Rosenfeld, eds., Springer Lecture Notes in Artificial Intelligence (LNAI 2734), pp. 96-105, Leipzig, Germany, 2003.

J.X. Dong, A. Krzyzak and C.Y. Suen, “Fast SVM training algorithm with decomposition on very large datasets,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 27, no. 4, pp. 603--618, 2005.

J. Devillers, Genetic Algorithms in Molecular Modeling, Academic Press, 1999a.

J. Devillers, Neural Networks and Drug Design, Academic Press, 1999b.

K. Duan, S.S. Keerthi and A.N. Poo, “Evaluation of simple performance measures for tuning SVM hyperparameters,” Neurocomputing, vol. 51, pp. 41-59, 2003.

K. Fukunaga and T.F. Krile, “Calculation of Bayes’ Recognition Error for Two Multivariate Gaussian Distributions,” IEEE Trans. Computers, vol. 18, no. 3, pp. 220–229, 1969.

G. Fung and O.L. Mangasarian, “Incremental Support Vector Machine Classification,” In R. Grossman, H. Mannila and R. Motwani, eds.: Proceedings of the Second SIAM International Conference on Data Mining, SIAM, pp. 247-260, 2002.

G. Fung and O.L. Mangasarian, “Proximal support vector machine classifiers,” in Proc. KDD-2001: Knowledge Discovery and Data Mining. F. Provost and R. Srikant, eds., San Francisco, 2001, pp. 77-86, New York, ACM press, 2001.

L.J. Fogel, A.J. Owens and M.J. Walsh, “Artificial Intelligence through Simulated Evolution,” John Wiley, NY, 1966.

T. Gärtner, P.A. Flach and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” Proceedings of the 16th Annual Conference on Computational Learning Theory, 2003.

T. Gärtner, “A Survey of Kernels for Structured Data,” ACM SIGKDD Explorations Newsletter, vol. 5, pp. 49-58, 2003.

H.P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic and V.N. Vapnik, “Parallel Support Vector Machines: The Cascade SVM,” Advances in Neural Information Processing Systems, 2005.

I. Guyon, B. Boser and V.N. Vapnik, “Automatic Capacity Tuning of Very Large VC-Dimension Classifiers,” Advances in Neural Information Processing Systems, vol. 5, Morgan Kaufmann, San Mateo, CA, 1993.

D. Haussler, “Convolution kernels on discrete structures,” Technical report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, 1999.

J.D. Hirst, R.D. King and M.J.E. Sternberg, “Quantitative structure-activity relationships by neural networks and inductive logic programming. I. The inhibition of dihydrofolate reductase by pyrimidines,” Journal of Computer-Aided Molecular Design, vol. 8, no. 4, pp. 405-420, 1994a.

J.D. Hirst, R.D. King and M.J.E. Sternberg, “Quantitative structure-activity relationships by neural networks and inductive logic programming. II. The inhibition of dihydrofolate reductase by triazines,” Journal of Computer-Aided Molecular Design, vol. 8, no. 4, pp. 421-432, 1994b.

T. Hofmann, B. Schölkopf and A.J. Smola, “A Review of Kernel Methods in Machine Learning,”, Technical Report 156, Max Planck Institute for Biological Cybernetics, Tubingen, Germany, 2006.

J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass Support Vector Machines,” IEEE Trans. on Neural Networks, 13(2), pp. 415-425, 2002.

B. Jin, Y.-Q. Zhang and B. Wang, “Granular Kernel Trees with Parallel Genetic Algorithms for Drug Activity Comparisons,” International Journal of Data Mining and Bioinformatics, vol. 1, no. 3, pp. 270-285, 2007.

B. Jin, Y.-Q. Zhang and B. Wang, “Evolutionary Granular Kernel Trees and Applications in Drug Activity Comparisons,” Proc. of IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, San Diego, pp. 121-126, 2005.

B. Jin and Y.-Q. Zhang, “Evolutionary Construction of Granular Kernel Trees for Cyclooxygenase-2 Inhibitor Activity Comparison,” Transactions on Computational Systems Biology V, C. Priami *et al*., eds., Lecture Notes in Computer Science (LNCS), vol. 4070, pp. 25-35, 2006a.

B. Jin and Y.-Q. Zhang, “Genetic Granular Kernel Methods for Cyclooxygenase-2 inhibitor Activity Comparison,” Third International Symposium on Neural Networks (ISNN), Chendu, 2006b.

B. Jin and Y.-Q. Zhang, “Evolutionary Voting Kernel Machines for Cyclooxygenase-2 Inhibitor Activity Comparisons,” Proc. of IEEE-GrC, 2006c.

B. Jin and Y.-Q. Zhang, “Classifying Very Large Data Sets with Minimum Enclosing Ball Based Support Vector Machine,” Proc. of IEEE International Conference on Fuzzy Systems, 2006d.

T. Joachims, “Estimating the Generalization Performance of a SVM Efficiently,” Proceedings of the International Conference on Machine Learning, Morgan Kaufman, 2000.

G.W. Kauffman and P.C. Jurs, “QSAR and k-Nearest Neighbor Classification Analysis of Selective Cyclooxygenase-2 Inhibitors Using Topologically-Based Numerical Descriptors,” J. Chem. Inf. Comput. Sci., vol. 41, no. 6, pp. 1553-1560, 2001.

H. Kashima and T. Koyanagi, “Kernels for Semi-Structured Data,” Proceedings of the Nineteenth International Conference on Machine Learning, pp. 291-298, 2002.

H. Kashima and A. Inokuchi, “Kernels for graph classification,” In ICDM Workshop on Active Mining, 2002.

J.R. Koza, “Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,” Stanford University Computer Science Department technical report STAN-CS-90-1314, 1990.

J.R. Koza, “Genetic Programming: On the Programming of Computers by Means of Natural Selection,” MIT Press, 1992.

W. Lee and S.J. Stolfo, “Data Mining Approaches for Intrusion Detection,” Proceedings of the 1998 USENIX Security Symposium, 1998.

S.-H. Lin, E.D. Goodman and W.F. Punch, III, “Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problem,” Proceedings of the 6th International Conference on Evolutionary Programming VI, 1997.

T.Y. Lin, “Granular computing,” Announcement of the BISC Special Interest Group on Granular Computing, 1997.

H. Lodhi, J. Shawe-Taylor, N. Christianini and C. Watkins, “Text classification using string kernels,” Advances in Neural Information Processing Systems 13., T. Leen, T. Dietterich and V. Tresp, eds., MIT Press, 2001.

O.L. Mangasarian and D.R. Musicant, “Lagrangian support vector machines,” Journal of Machine Learning Research, vol. 1, pp.161-177, 2001.

J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," Proceedings of the Royal Society of London, Series A, containing papers of a Mathematical and Physical Character, vol. 83, no. 559, pp. 69-70, 1909.

Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, Berlin, 1996.

D.R. Musicant, "NDC: normally distributed clustered datasets," Computer Sciences Department, University of Wisconsin, Madison, 1998.

D.J. Newman, S. Hettich, C.L. Blake and C.J. Merz, UCI Repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science, 1998.

E. Osuna, R. Freund and F. Girosi, "An improved training algorithm for support vector machines," In IEEE Workshop on Neural Networks and Signal Processing, pp. 276-285, 1997.

K.-J. Park and M. Kanehisa, "Prediction of protein subcellular locations by support vector machines using compositions of amino acids and amino acid pairs," Bioinformatics, vol. 19, no. 13, pp. 1656-1663, 2003.

J. C. Platt, “Fast Training of Support Vector Machines using Sequential Minimal Optimization,” in Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., MIT Press, 1998.

J. C. Platt, N. Cristianini and J. Shawe-Taylor, “Large margin DAGs for multiclass classification,” Advances in Neural Information Processing Systems, S.A. Solla, T.K. Leen and K.-R. Müller, eds., MIT Press, Cambridge, MA, vol. 12, pp. 547-553, 2000.

J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Los Altos, 1993.

J. R. Quinlan, “Induction of decision trees,” Machine Learning, vol. 1, pp. 81-106, 1986.

I. Rechenberg, “Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution,” Frommann-Holzboog, Stuttgart, 1973

F. Rosenblatt, “The Perceptron: a probabilistic model for information storage and organization in the brain,” Cornell Aeronautical Laboratory, Psychological Review, vol. 65, no. 6, pp. 386-408, 1958.

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, “Learning internal representations by error propagation,” in D.E. Rumelhart and J.L. McClelland, eds., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, pp. 318-362, Cambridge, MA: The MIT Press, 1986.

T.P. Runarsson and S. Sigurdsson, “Asynchronous Parallel Evolutionary Model Selection for Support Vector Machines,” Neural Information Processing - Letters and Reviews, vol. 3, no. 3, pp. 59-67, 2004.

G. Schohn and D. Cohn, “Less is more: Active learning with support vector machines,” in Proc. 17th Int. Conf. Machine Learning, Stanford, CA, 2000.

H.-P. Schwefel, Numerical optimization of computer models, Wiley, Chichester, 1981.

T. Serafini, G. Zanghirati and L. Zanni, “Gradient Projection Methods for Large Quadratic Programs and Applications in Training Support Vector Machines,” Optim. Meth. Soft., vol. 20, pp. 353-378, 2005.

J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

M. Stitson, A. Gammerman, V.N. Vapnik, V. Vovk, C. Watkins and J. Weston, “Support vector regression with ANOVA decomposition kernels,” In B. Scholkopf, C. J. C. Burges, and A. J. Smola, eds., Advances in Kernel Methods - Support Vector Learning, pp. 285-292, MIT Press, Cambridge, MA, 1999.

D.M.J. Tax and R.P.W. Duin, “Data Domain Description by Support Vectors,” In: Verleysen, M. (ed.), Proceedings of ESANN, D. Facto Press, Brussels. pp. 251-256, 1999.

D.M.J. Tax and R.P.W. Duin, “Support Vector Data Description,” Machine Learning, vol. 54, no. 1, pp. 45-66, 2004.

I.W. Tsang, J.T. Kwok and P.-M. Cheung, “Core vector machines: Fast SVM training on very large data sets,” Journal of Machine Learning Research, vol. 6, pp. 363-392, 2005.

V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, N.Y., 1995.

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

V.N. Vapnik and O. Chapelle, “Bounds on error expectation for support vector machine,” Advances in Large Margin Classifiers, A. Smola, P. Bartlett, B. Schöölkopf and Schuurmans, D., eds. , MIT Press, Cambridge, MA, 1999.

H. Yu, J. Yang and J. Han, “Classifying large data sets using SVMs with hierarchical clusters,” In Proc. ACM Int’l Conf. Knowledge Disc. Data Mining (KDD), pp. 306-315, 2003.

L.A. Zadeh, “Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems,” Soft Computing, vol. 2, pp. 23-25, 1998.

L.A. Zadeh, “Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic,” Fuzzy Sets and Systems, vol. 90, pp. 111-127, 1997.

G. Zanghirati and L. Zanni, “Parallel Solver for Large Quadratic Programs in Training Support Vector Machines,” Parallel Computing, vol. 29, pp. 535-551, 2003.