View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by SHAREOK repository

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

INCREMENTAL KERNEL LEARNING ALGORITHMS AND APPLICATIONS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

HYUNG-JIN SON
Norman, Oklahoma
2006

UMI Number: 3212011

UMI

UMI Microform 3212011

Copyright 2006 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

INCREMENTAL KERNEL LEARNING ALGORITHMS
AND APPLICATIONS

A DISSERTATION APPROVED FOR THE
SCHOOL OF INDUSTRIAL ENGINEERING

BY

Dr. Theodore B. Trafalis (Chair)

Dr. P. Simin Pulat

Dr. Mary C. Court

Dr. Michael B. Richman

Dr. Suleyman Karabuk

© Copyright by HYUNG-JIN SON 2006
All Rights Reserved.

ACKNOWLEDGEMENTS

I would like to acknowledge a great many people who have contributed to this dissertation. I could not finish this dissertation without their support.

First, I am grateful to my committee members, Dr. Theodore B. Trafalis, Dr. P. Simin Pulat, Dr. Mary C. Court, Dr. Michael B. Richman, and Dr. Suleyman Karabuk. I express the deepest gratitude to my advisor, Dr. Trafalis. I have been very fortunate to have him as my advisor. His advice and support never stopped even on weekends. I really appreciate his guidance. I will never forget Dr. Pulat's support and encouragement, especially in my early hard times. I am deeply thankful to her.

I am indebted to Dr. Court who helped me many times as a graduate liaison. She also gave me good lectures. I am deeply thankful to Dr. Richman. His insightful comments on research helped me see the correct direction. I would like to acknowledge Dr. Karabuk. He is always there, and I feel relieved from the fact that his office is nearby.

My deepest gratitude is to my family. None of this work would be possible without the love, support, and patience of my parents, Tae-Kyu Son and Soon-Im Lee. I would like to express my thanks with all my heart. I am also grateful to my lovely wife Sujeong, and my children, precious Rachael and gallant Bryan.

Finally, I appreciate the financial support from the National Science Foundation. This material is based on research funded by the National Science Foundation Grant EIA-0205628.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
LIST OF FIGURES viii
LIST OF TABLES ix
ABSTRACT x
1. INTRODUCTION 1
2. BASIC CONCEPTS 5
2.1 Support Vector Machines 5
2.2 Support Vector Regression (SVR) 10
2.3 Incremental learning procedure 14
3. INCREMENTAL REVISED SVM 18
3.1 A Reduced Convex Hull SVM (RCH-SVM) 18
3.2 A Revised SVM (RSVM) 22
3.3 Examples for applying RSVM 28
3.3.1 Description of Examples 28
3.3.2 Results for both cases 31
3.4 Incremental learning with revised SVM with filters (IRSVMF) 33
4. INCREMENTAL REVISED SVR 40
4.1 A Reduced Convex Hull SVR (RCH-SVR) 41
4.2 A Revised SVR (RSVR) 45
4.3 Incremental learning with Revised SVR with Filters (IRSVRF) 54
4.4 A toy problem for IRSVR 60
4.4.1 Description of experiments 61
4.4.2 Experimental results 62
5. APPLICATIONS AND COMPUTATIONAL RESULTS 64
5.1 Tornado detection Problem 64

5.1.1 Description of Experiments for Tornado Detection 66

5.1.2 Experimental Results 68

5.2 Financial Forecasting Problem 70

5.2.1 Description of experiments for financial forecasting 71

5.2.2 Experimental results 72

6. SUMMARY AND FUTURE RESEARCH 74

BIBLIOGRAPHY 76

APPENDICES 80

LIST OF FIGURES

Figure 1. An example of linearly separable data 7
Figure 2. An example of linearly inseparable data 8
Figure 3. ε-insensitive loss function 12
Figure 4. Structure of the incremental learning procedure 15
Figure 5. Convex hulls of inseparable data set 20
Figure 6. Reduced convex hulls of inseparable data sets 20
Figure 7. Geometric interpretation of the revised SVM 23
Figure 8. Geometric interpretation of the revised SVM 29
Figure 9. Linearly separable data (Test) 29
Figure 10. Training data in Ripley's data set 30
Figure 11. Testing data in Ripley's data set 30
Figure 12. Comparisons in categories with linearly separable data 32
Figure 13. Comparisons in categories with linearly inseparable data 32
Figure 14. Incremental SVM learning 35
Figure 15. Incremental learning with revised SVM and a filter 37
Figure 16. Supporting hyperplane as a filter 38
Figure 17. Three cases of support vector regression 41
Figure 18. Inclination of the separating hyperplane 53
Figure 19. Diagram for IRSVR 55
Figure 20. Diagram for IRSVRT 57
Figure 21. Two filters for IRSVRTF 58
Figure 22. Diagram for IRSVRTF 59
Figure 23. A data set generated from the noisy sine function 60
Figure 24. Comparison between ISVM and IRSVM with respect to computing time 69
Figure 25. Comparison between ISVM and IRSVM with respect to the number of support vectors 69

LIST OF TABLES

Table 1. Possible kernel functions 10
Table 2. Comparison between revised SVM and standard SVM in both cases 31
Table 3. Comparison of different approaches under comparison criteria 62
Table 4. Tornado confusion matrix 65
Table 5. List of attributes 67
Table 6. Comparison of incremental learning methods 68
Table 7. Comparison of methods with respect to FAR and CSI 70
Table 8. Size of available data in 5 decades 71
Table 9. Comparison of different approaches under the comparison criteria 72
Table 10. ANOVA Table for MSE 73
Table 11. ANOVA Table for MAE 73

ABSTRACT

Since the Support Vector Machines (SVMs) were introduced in 1995, SVMs have been recognized as essential tools for pattern classification and function approximation. Numerous publications show that SVMs outperform other learning methods in various areas. However, SVMs have a weak performance with large-scale data sets because of high computational complexity. One approach to overcome this limitation is the incremental learning approach where a large-scale data set is divided into several subsets and trained on those subsets updating the core information extracted from the previous subset. This approach also has a drawback that the core information is accumulated during the incremental procedure. When the large-scale data set has a special structure (e.g., in the case of unbalanced data set), the standard SVM might not perform properly. In this study, a novel approach based on the reduced convex hull concept is developed and applied in various applications. In addition, the developed concept is applied to the Support Vector Regression (SVR) to produce better performance. From the performed experiments, the incremental revised SVM significantly reduces the number of support vectors and requires less computing time. In addition the incremental revised SVR produces similar results with the standard SVR by reducing computing time significantly. Furthermore, the filter concept developed in this study may be utilized to reduce the computing time in other learning approach.

CHAPTER 1

INTRODUCTION

Recently modern technology brings numerous changes in various areas. The existing analog process has been digitized so that a vast number of data is produced and processed. Currently many devices have been digitized (e.g., TV), and this trend expands to all areas surrounding our lives. As a typical example of these devices, computers contribute to the development of internet that produces and stores huge data. It is not easy to extract necessary information properly from a vast number of data. Selection of an appropriate learning tool for pattern recognition, learning, prediction, and estimation is quite important. One of the recent successful techniques in machine learning community is Support Vector Machines (SVMs).

Since the Support Vector Machines (SVMs) were introduced in 1995 [Vapnik, 1995], SVMs have been recognized as essential tools for pattern classification and function approximation. Numerous publications show that SVMs outperform other learning methods in various areas such as text categorization [Joachims, 1998], object recognition (e.g., speech and image) [Cristianini and Shawe-Taylor, 2000], bioinformatics [Brown et al., 1999], data-mining [Burges, 1998], financial forecasting [Yang, 2002], and meteorology [Trafalis et al., 2004]. However, SVMs have a weak performance on large-scale data sets because of the high computational complexity of the existing SVM models. Diverse approaches have been proposed to overcome this limitation, such as decomposition [Osuna, 1997], reduced set SVM [Lee and Mangasarian, 2001], online learning approach [Cauwenberghs and Poggio, 2001], etc.

One of the approaches to overcome this limitation is the incremental learning approach where a large-scale data set is divided into several subsets and trained on those subsets updating the core information (support vectors) extracted from a previous subset. In applying the incremental learning process in SVMs for classification problems, the process will face several limitations as follows:

First, the number of support vectors is relatively small but critical because these support vectors create the decision function (separating hyperplane in the feature space) in a two-class classification problem. However, those support vectors are accumulated as the incremental learning process is repeated.

Second, support vectors determine the feasible region (decision space) and its dimension in the feature space. Thus, to execute the standard SVM without appropriate control of support vectors might expand its dimension in the feature space. In that sense, it's important to control the number of support vectors in SVM. Furthermore, to control the number of support vectors in SVM is effective for feature space reduction.

Third, SVM wastes most computing time for computation of kernel function values among unimportant data. If SVM is used for training the large-scale data set with a special structure (e.g., unbalanced data set where there are many data in one class (unimportant) and few data in the other class (important)), then, consumption of computing time for kernel function evaluation among data points that are unimportant should be avoided to reduce the training time.

In this study, a revised SVM based on the reduced convex hull concept is developed to reduce the number of support vectors, and an incremental learning procedure incorporating the revised SVM is constructed. To make the incremental learning applicable to on-line settings, a filter which is discarding unimportant data is created.

As SVMs are used for classification problems, these can be also applied to nonlinear regression problems by using Support Vector Regression (SVR) [Vapnik, 1995]. The aim of SVR is to obtain an approximate function that has at most ε deviation from the actual outputs of all input data. Therefore, SVR considers any deviation greater than ε ignoring any deviation less than ε. In literature, SVR has been studied in various areas such as weather prediction [Trafalis et al., 2004], financial forecasting [Yang et al., 2002].

In applying SVR, the approximate function is quite different from the originally intended approximate function if there exist noisy data (e.g., outliers) and/or input data has a special structure such as an unbalanced data set. To overcome this weak point, the revised SVM concept is extended to the Support Vector Regression. Thus, the revised SVR scheme is utilized in the incremental approach to deal with vast number of data.

These revised SVM and SVR concepts are executed in applications to show that these concepts perform efficiently and effectively to problems with noisy data and/or special structures.

This dissertation is organized as follows: Chapter 2 describes basic concepts such as Support Vector Machines (SVMs), Support Vector Regression (SVR), and incremental learning. In Chapter 3, the revised SVM is developed and applied to the incremental learning procedure. A filter concept is also developed to drop the most likely unimportant data in an incremental learning procedure within the revised SVM framework. Chapter 4 describes the SVR version of the revised SVM concept. Chapter 5 contains description of two applications and computational results. Summary and future research are discussed in Chapter 6.

CHAPTER 2

BASIC CONCEPTS

Several basic concepts of learning procedures such as SVMs, SVR and the incremental learning procedure, are presented in this chapter.

2.1 Support Vector Machines

SVMs developed by Vapnik [Vapnik, 1995] have been widely used for pattern classification and nonlinear regression. The basic idea of SVM in a binary classification problem is to construct a decision hyperplane to separate input data with positive and negative classes maximizing the margin of separation. SVM can be applied both in a linearly separable pattern case and a linearly inseparable pattern case. A brief mathematical explanation of SVM for both cases is as follows:

Consider the training data $\{(x_1, y_1), \cdots, (x_m, y_m)\}$, where x_i is the input pattern for the *i*th datum, and y_i is the corresponding output ($\pm$ 1). It is assumed that the training data and corresponding outputs are provided (i.e., supervised learning).

First, the linearly separable pattern case is considered. The term, linearly separable pattern, means that two or more classes can be separated by decision hyperplane(s). The equation of a decision hyperplane is

$$w^T x + b = 0 \tag{2.1}$$

where x is the input vector, w is the weight vector, and b is a bias. The aim of SVM is to obtain the optimal values of w_o and b_o for a decision hyperplane, $w_o^T x + b_o = 0$ solving the following optimization problem.

$$\begin{aligned} &Min \quad \frac{1}{2}\|w\|^2 \\ &\text{s.t.} \\ &y_i[w^T x_i + b\ ] \geq +1\ ,\ i = 1, 2, \cdots, m \end{aligned} \tag{2.2}$$

The data points x_i that satisfy the constraints shown in (2.2) with equality sign are called support vectors. In other words, the support vectors are critical data points that are located in the closest positions to the decision hyperplane. In a binary classification problem, there are two kinds of support vectors, support vectors in the positive class and those on the negative class.

As shown in Figure 1, data points can be separated by a decision function $w^T x + b = 0$. The two points on the line, $w^T x + b = -1$, are support vectors in the negative class. The three points on the line, $w^T x + b = 1$, are also support vectors in the positive class.

Minimizing the cost function, $\Phi(w) = \frac{1}{2} w^T w$ with constraints in (2.2) is a quadratic convex constrained optimization problem with linear constraints. This constrained optimization problem is transformed by duality theory as follows:

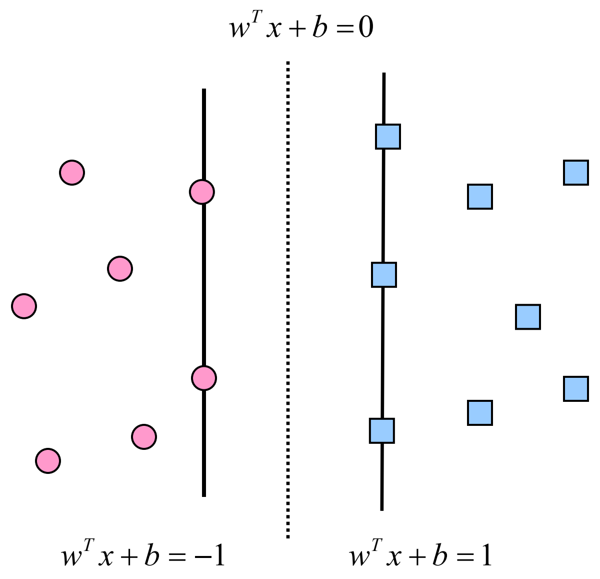

Figure 1. An example of linearly separable data

Given the training data $\{(x_1, y_1), \cdots, (x_m, y_m)\}$, the objective is to find the Lagrange multipliers, $\{\alpha_1, \alpha_2, \cdots, \alpha_m\}$ in the following optimization problem.

$$
\begin{aligned}
& Max\ \ F(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j x_i^T x_j \\
& \text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (2.3) \\
& \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \\
& \qquad \alpha_i \geq 0 \ \text{ for } i = 1, 2, \cdots, m
\end{aligned}
$$

Different from the linearly separable pattern case, the linearly inseparable pattern case indicates that data points cannot be separable linearly. A simple example for the linearly inseparable pattern case is shown in Figure 2.

Some data points in the positive class and some data points in the negative class are mixed in the middle of space.

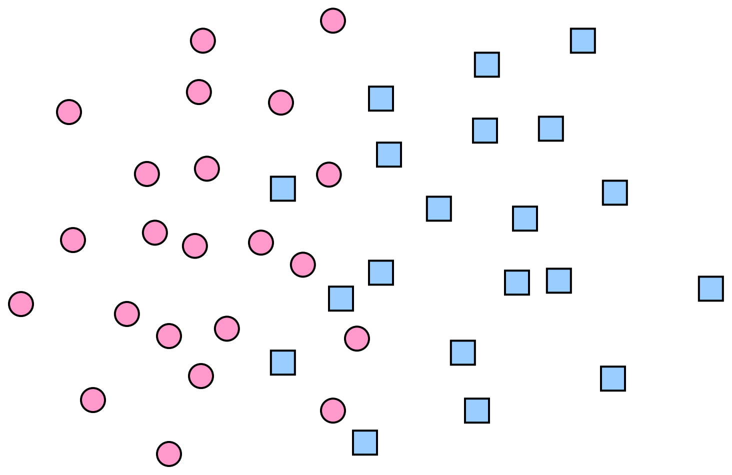

Figure 2. An example of linearly inseparable data

In the linearly inseparable patterns case, the cost function contains a penalty term referring to the misclassification errors as follows: $\Phi(w) = \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i$. Here, C is a user-defined parameter that controls the tradeoff between the number of inseparable data points and complexity of the machine. Thus, the constrained optimization problem (2.3) in order to address the problem of misclassification of some training data is slightly changed to the optimization problem shown in (2.4).

The inseparable case differs from the separable case in that the constraints on alphas are modified to the box constraints with upper bound, C.

$$Max\ \ F(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j x_i^T x_j$$

s.t. (2.4)

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C \text{ for } i = 1, 2, \cdots, m$$

SVM has two major operations. First, an input vector is mapped to a high-dimensional feature space by a nonlinear transformation. Second, an optimal hyperplane (decision function) is constructed for separating the mapped data points in the feature space.

Let x be a vector in the input space with dimension d. Let $\varphi(\bullet)$ be a nonlinear transformation from the input space to the feature space. It is hard to explicitly obtain this transformation. The separating hyperplane in the feature space takes the compact form, $w^T \varphi(x) = 0$ where $w = \sum_{i=1}^{m} \alpha_i y_i \varphi(x_i)$. Therefore, in the feature space the separating hyperplane is defined as $\sum_{i=1}^{m} \alpha_i y_i \varphi^T(x_i)\varphi(x) = 0$. Note that $\varphi^T(x_i)\varphi(x)$ represents the inner product of two feature vectors corresponding to the ith input data point and a general input data point, respectively.

By $K(x, x_i)$, we denote the inner-product kernel. It is defined by $K(x, x_i) = \varphi^T(x_i)\varphi(x)$. This inner-product kernel $K(x, x_i)$, which is a symmetric function, is used to construct the optimal separating hyperplane in the feature space. Thus, the final form of the optimal separating hyperplane is defined by

$$\sum_{i=1}^{m} \alpha_i y_i K(x, x_i) = 0$$

Possible kernel functions are listed in Table 1.

Table 1. Possible kernel functions

Type of Classifier	Kernel Function
Polynomial of degree d	$K(x, x_i) = (1 + x^T x_i)^d$
Gaussian Radial-basis function (RBF)	$K(x, x_i) = \exp(-\frac{1}{2\sigma^2} \lVert x - x_i \rVert^2)$
Two-layer perceptron (for some β_0 and β_1)	$K(x, x_i) = \tanh(\beta_0 x^T x_i + \beta_1)$

Using an inner-product kernel that satisfies Mercer's Theorem [Vapnik, 1995], the objective function in (2.4) becomes $F(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$. The resulting decision function is shown to take the form, $f(x) = \operatorname{sgn}(\sum_{i=1}^{m} \alpha_i y_i K(x_i, x) + b)$.

2.2 Support Vector Regression (SVR)

The SVM can be successfully applied to nonlinear regression as follows: Given training data $\{(x_1, y_1), \cdots, (x_m, y_m)\}$ where x_i is a data point in the input space **X** and y_i is the corresponding target of the model output. The objective is to find an approximation function $f(x)$, which has at most ε deviation from the actual target y_i for all the training data. In the ε-SV regression [Vapnik, 1995], any deviation that is less than ε is allowed.

Consider a linear function, $f(x) = <w, x> + b$ where $w \in \mathbf{X}$, $b \in \Re$. The objective of SVR is to minimize the Euclidean norm of w as long as deviation from the actual targets y_i is less than ε. This convex optimization problem is shown as follows:

$$
\begin{aligned}
&Min \quad \frac{1}{2}\|w\|^2 \\
&\text{s.t.} \\
&\qquad y_i - <w, x_i> - b \le \varepsilon \qquad i = 1, 2, \cdots, m \\
&\qquad <w, x_i> + b - y_i \le \varepsilon \qquad i = 1, 2, \cdots, m
\end{aligned}
\tag{2.5}
$$

The implicit assumption in (2.5) is that the problem is feasible as the function $f(x)$ that approximates all input data, $\{(x_1, y_1), \cdots, (x_m, y_m)\}$ with ε deviation exists. To construct a support vector machine for approximating a target, d, the following loss function is used:

$$
L_\varepsilon(d, y) = \begin{cases} |d - y| - \varepsilon, & for \ |d - y| \ge \varepsilon \\ 0 & otherwise \end{cases} \tag{2.6}
$$

where ε is a predetermined parameter with positive value.

The loss function in (2.6) is called the ε-insensitive loss function. This loss function implies that this function has a value only when the deviation of output, y, from the desired target, d, is greater than the deviation parameter, ε. It is graphically presented as shown in Figure 3.

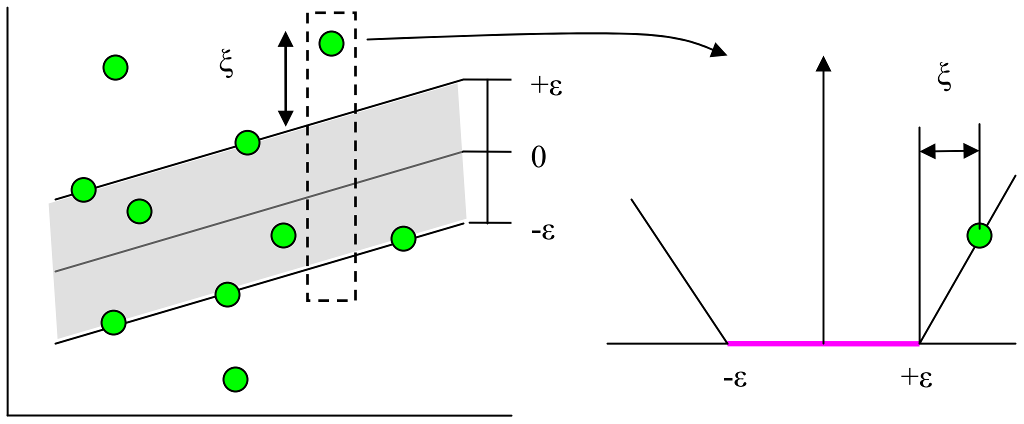

Figure 3. ε-insensitive loss function

To deal with the infeasible constraints of the optimization problem in (2.5), slack variables, $\xi_i, \xi_i^{'}$ are introduced to form the following problem:

$$
\begin{aligned}
&Min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^{'}) \\
&\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (2.7) \\
&\qquad y_i - < w, x_i > - b \le \varepsilon + \xi_i \qquad i = 1, \cdots, m \\
&\qquad < w, x_i > + b - y_i \le \varepsilon + \xi_i^{'} \qquad i = 1, \cdots, m \\
&\qquad \xi_i \;, \xi_i^{'} \;\ge 0 \qquad\qquad\qquad i = 1, \cdots, m
\end{aligned}
$$

To construct the dual form of the optimization problem (2.7), dual variables $(\alpha_i, \alpha_i^{'})$ are introduced. Note that α_i corresponds to the constraint with slack ξ_i, and $\alpha_i^{'}$ corresponds to the constraint with slack $\xi_i^{'}$. These variables are actually Lagrangian multipliers for the primal problem. By KKT conditions and Mercer's theorem [Vapnik, 1995], the resulting dual optimization problem for nonlinear regression using SVMs is as follows:

$$
\begin{aligned}
&Max\ \ F(\alpha,\alpha^{'}) \\
&\qquad = \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i^{'}) - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i^{'})(\alpha_j - \alpha_j^{'})K(x_i,x_j) - \varepsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^{'}) \\
&\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (2.8) \\
&\qquad \sum_{i=1}^{m}(\alpha_i - \alpha_i^{'}) = 0 \\
&\qquad 0 \le \alpha_i \le C \ \text{ for } i = 1,2,\cdots,m \\
&\qquad 0 \le \alpha_i^{'} \le C \ \text{ for } i = 1,2,\cdots,m
\end{aligned}
$$

From the optimality condition, w is obtained as follows:

$$w = \sum_{i=1}^{m}(\alpha_i - \alpha_i^{'})x_i \qquad (2.9)$$

Thus, the approximation function, $f(x) = < w,x > + b$, is expressed as

$$f(x) = \sum_{i=1}^{m}(\alpha_i - \alpha_i^{'})K(x_i,x) \qquad (2.10)$$

Note that the two parameters, the deviation ε and parameter C , which correspond to the trade-off between the flatness of f and the tolerance of deviation, ε respectively, are determined by the user. The bias, b , is computed as follows:

$$b = y_i - < w,x_i > - \varepsilon_i,\ 0 \le \alpha_i \le C \ \text{ for } i = 1,2,\cdots,m$$

$$b = y_i - < w,x_i > + \varepsilon_i,\ 0 \le \alpha_i^{'} \le C \ \text{ for } i = 1,2,\cdots,m$$

2.3 Incremental learning procedure

Learning (training) is an essential procedure for pattern recognition, which is the process whereby a received pattern (e.g., data, image, or signal) is assigned to one of a finite number of predetermined classes (categories). By undergoing a training session, the class of the particular pattern is identified and decision boundaries (e.g., a decision hyperplane) are determined.

To obtain the best estimate of the class information in the conventional learning procedure, it is desirable to consider all available data simultaneously. However, all data cannot be loaded at once due to memory limits, size of data, and intermittent inflows. To overcome these limitations, several approaches have been proposed (e.g., [Osuna et al., 1997]). One of them is the incremental learning approach.

There are two kinds of incremental learning with respect to target concept [Ruping, 2001]. One of them is the true incremental learning, and the other is the concept drift. In the true incremental learning, all data contain the same information about the target concept. The information of a data point cannot be judged from its age. In other words, if a data point is classified once in a certain class, the class of the data point is not changed, even though the data point is quite old. In contrast, the target concept may change between the learning steps in the concept drift. For example, an input data point was classified in one class a long time ago. At present, the data point should be classified as being in the other class. In order for this drift concept to be applied to a learning system, an additional algorithm to the existing learning process must be added. For details, refer to [Klinkenberg and Joachims, 2000].

The incremental learning approach can be used in the following situations: First, as mentioned above, when the size of the training data set is huge, and it cannot be trained simultaneously. Second, when new data are available at periodic intervals (e.g., results, which are obtained at a periodic interval in the performance of scientific experiments). Third, when data are available on a timely basis (on-line).

The structure of the incremental learning procedure is shown in Figure 4.

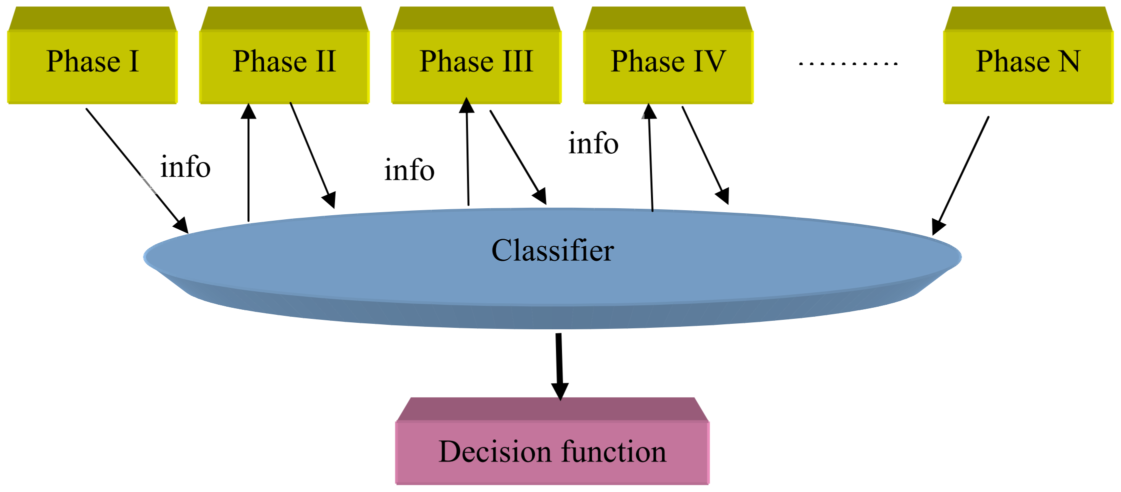

Figure 4. Structure of the incremental learning procedure

Patterns in phase I are trained to establish a classifier. Based on results obtained from this classifier, the decision function is created. During this process, the classifier retains knowledge learned from previous phases. According to the knowledge-based classifier, the decision function is changed and used to predict the classes of incoming patterns. The above procedure is adaptive and can be more effective to real time learning.

When a SVM is utilized as a classifier, it forms the incremental SVM. The resulting decision function from training a SVM depends on its support vectors. The support vectors have nice properties for summarizing data characteristics. Training SVM on the support vectors alone produces the same decision function as training on the whole data set. Thus, if the last training set (last batch) contains all the examples that are support vectors in the non-incremental SVM, the resulting classifier in the incremental approach is the same as the one in the non-incremental approach.

There are two approaches in the incremental SVM. One of them is the batch approach, and the other is the on-line approach. In the batch approach, the whole data set is partitioned into several batches if the whole data set is available at present, and each batch is formed in a certain time frame if the data set is provided periodically. Generally speaking, SVM is trained on the new data and support vectors produced from the previous learning step.

In the on-line approach, all data come into a certain algorithm based on SVM and are trained on-line. Most publications in the on-line approach track the value of alpha (multipliers in KKT conditions) and its change (e.g., [Cauwenberghs and Poggio, 2001]). Hence, based on the changed value of the alphas, a new data point is added in the next step. However, this approach will not work properly if the support vectors are accumulated. After the data set is trained, the support vectors have multipliers with positive values. In order to obtain these multipliers' value the on-line learning algorithm should train a certain amount of data points like in the batch approach. The on-line approach also stores support vectors produced in the previous training step as the batch approach does. Thus, the on-line approach can be considered as a variant of the incremental approach. An incremental SVM learning algorithm with input data that are available only on periodical time windows will be called the “on-line algorithm.”

Advantages of the incremental learning with SVMs can be summarized as follows:

First, it makes the number of support vectors to be as small as possible. Although the number of support vectors varies depending on the distribution of the data set, the number of support vectors is relatively smaller than the whole data set. Also, the number of support vectors is reduced because old support vectors identified long time ago might be removed in a recent training session.

Second, it can keep the memory and time complexity of the learning algorithm at a manageable level. As the size of input data is gradually increased, the computing time for training SVM with an increased size of input data will become exponential. In the incremental approach, the size of the batch is predetermined both in the case of static data and in the case of online injection of data. Thus, partitioning data into a batch scheme makes it possible to train large data sets.

Third, the incremental learning can predict at a time when the whole data is not yet available (on-line setting). When the data set is only periodically available (e.g., weather data or financial data), the traditional learning approach should wait until all data are available for training. In contrast, the incremental approach can be applied to train a small portion of whole data set and has the capability to predict the class of an incoming data set using the constructed classifier before the next data set arrives.

CHAPTER 3

INCREMENTAL REVISED SVM

To produce a better performance in a large-scale data set with a special structure, the revised SVM concept is proposed in this chapter. This concept is applied to the standard SVM in the incremental scheme. For this novel approach to be applied in a dynamic data-driven application system, the filter concept is also proposed.

3.1 A Reduced Convex Hull SVM (RCH-SVM)

Bennett and Breadensteiner (2000) developed the reduced convex hull SVM (RCH-SVM) concept giving a geometric explanation of the standard SVM. Crisp and Burges (1999) also developed the same concept in a geometric interpretation of ν-SVM independently.

A similar term, “reduced support vector machines”, was used by Lee and Mangasarian (2001). However, they proposed a different approach, which pursues the same objective of reducing the number of support vectors. In their work, an m by m matrix, where m is the number of the whole training data, is reduced to an n by n matrix, where n is the size of subset that consists of randomly selected data from the training data set. These selected data are considered as candidates of support vectors. By reducing the training data, they intended to speed up the computational time.

The RCH-SVM is an extension of the standard SVM. Before the RCH-SVM is explained, some basic algebraic concepts are reviewed.

Definition 1. (Convex Combination) [Bazaraa, 1990]

Let V be some vector space over $\Re$. Let X be a set of elements of V. Then, a convex combination of elements in the set X is defined as a linear combination of the form $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \cdots + \alpha_N x_N$ for some $N > 0$ where each $x_i \in X$ and $\alpha_i \geq 0$, with $\sum_{i=1}^{N} \alpha_i = 1$

Definition 2. (Convex Hull)

Let V be some vector space over $\Re$. Let X be a set of elements of V. Then, a convex hull, conv(X), is defined as the smallest convex set containing X.

Definition 3. (Reduced convex hull) [Bennett, 2000]

Let V be some vector space over $\Re$. Let X be a set of elements of V. Then a reduced convex hull, RCH, of the set X is defined as a set that consists of convex combinations of the form, RCH = $\{\sum_{i=1}^{N} \alpha_i x_i \mid \sum_{i=1}^{N} \alpha_i = 1,\ 0 \leq \alpha_i \leq \mu,\ \mu < 1,\ x_i \in X\}$.

Geometrically, the reduced convex hull shrinks the convex hulls of two classes so that noisy data (e.g., outliers) cannot affect the solution significantly. Consider the following two convex hulls of inseparable data sets.

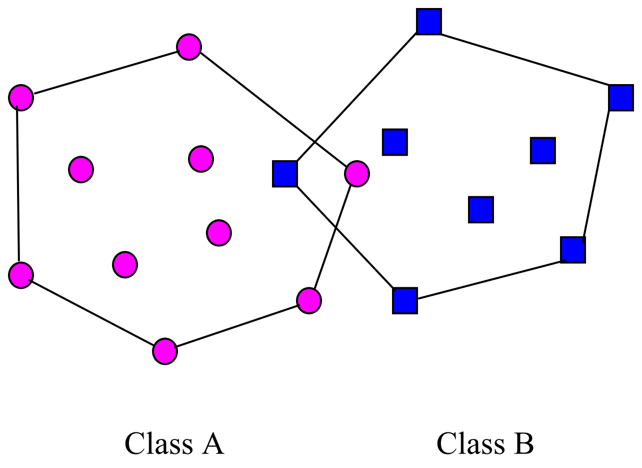

Figure 5. Convex hulls of inseparable data set

From Figure 5, clearly the data set cannot be classified linearly with a hyperplane. Thus, the reduced SVM shrinks the convex hulls in both sides simultaneously as shown in Figure 6.

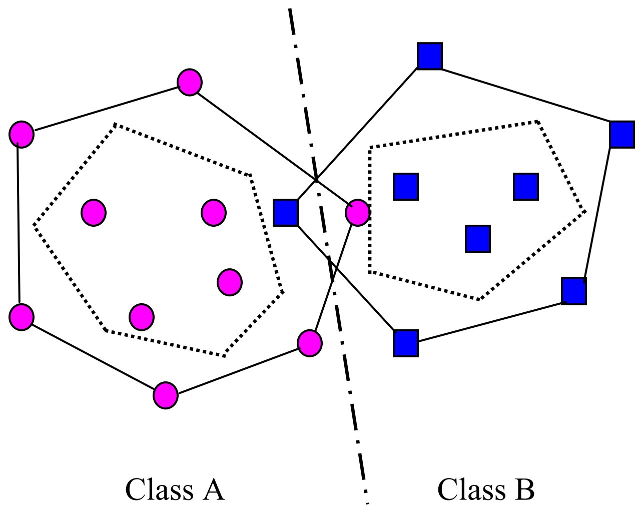

Figure 6. Reduced convex hulls of inseparable data sets

In the separable case, it is clear that the optimal separating hyperplane bisects the shortest vector connecting the convex hulls of the positive and negative examples in the feature space.

Consider the convex hulls of the sets, U and V. Each point of the convex hull of U can be expressed through the points $u_i \in U$, where $i = 1, 2, \cdots, m$. Similarly, each point in the convex hull of V can be represented through the points, $v_j \in V$ where $j = 1, 2, \cdots, n$. Then, the two convex combinations can be defined as $\sum_{i=1}^{m} a_i u_i$ and $\sum_{j=1}^{n} b_j v_j$, respectively. In the separable case, the optimization problem finding the closest points can be written as follows:

$$\underset{a,b}{Min} \quad \frac{1}{2}\left\| \sum_{i=1}^{m} a_i u_i - \sum_{j=1}^{n} b_j v_j \right\|^2$$

s.t.

$$\sum_{i=1}^{m} a_i = 1$$

$$\sum_{j=1}^{n} b_j = 1$$

$$0 \le a_i \text{ for } i = 1, 2, \cdots, m$$

$$0 \le b_j \text{ for } j = 1, 2, \cdots, n$$

When data are not linearly separable, the convex hulls are reduced because two convex hulls cannot be linearly separable. After the convex hulls shrink, the separating hyperplane is constructed by solving the linearly separable problem. It is possible to obtain the separating hyperplane, because the linearly inseparable problem becomes a linearly separable one, if the convex hulls are shrunk. The optimization problem that finds the closest points in the reduced convex hulls is written as follows:

$$\underset{a,b}{Min} \quad \frac{1}{2}\left\| \sum_{i=1}^{m} a_i u_i - \sum_{j=1}^{n} b_j v_j \right\|^2$$

s.t.

$$\sum_{i=1}^{m} a_i = 1$$

$$\sum_{j=1}^{n} b_j = 1$$

$$0 \le a_i \le \mu \qquad \text{for } i = 1, 2, \cdots, m$$

$$0 \le b_j \le \mu \qquad \text{for } j = 1, 2, \cdots, n$$

3.2 A Revised SVM (RSVM)

The standard SVM produces a huge amount of support vectors, especially when the positive and negative training samples are highly overlapped with each other. It is obvious that the big size of support vectors requires more computing time and more storage space for training. Thus, it is natural to say that reducing the computational time and cost of the SVM is equivalent to decreasing the number of support vectors.

One efficient way to decrease the number of support vectors is to simplify the shape of the separating hypersurface. Since the support vectors determine the shape of the separation hypersurface, support vectors that do not affect the shape of the separation hypersurface effectively, should be removed.

Furthermore, noisy data (e.g., outliers) might significantly affect the standard SVM producing an incorrect decision function. As a result, the incorrect decision function generates unexpected generalization errors and affects sequentially the next steps in the incremental learning process.

Therefore, the standard SVM must be modified to resolve the above problems (e.g., computing time and number of support vectors). The geometric interpretation of the RCH-SVM is quite useful and provides an alternative for modifying the standard SVM. However, it might be inefficient for particular problems such as unbalanced problems in which there are huge differences of data sizes in two classes, and asymmetric importance problems in which the data in one class are few and have very important meanings, and there are a lot of unimportant data in the other class. For example, tornadic data are very few (as little as 2%) relatively to nontornadic data in the weather data. If the RCH-SVM is applied to this tornado detection problem, some valuable tornadic data will be lost. Thus, in order to properly solve this problem, the following geometric concept is proposed.

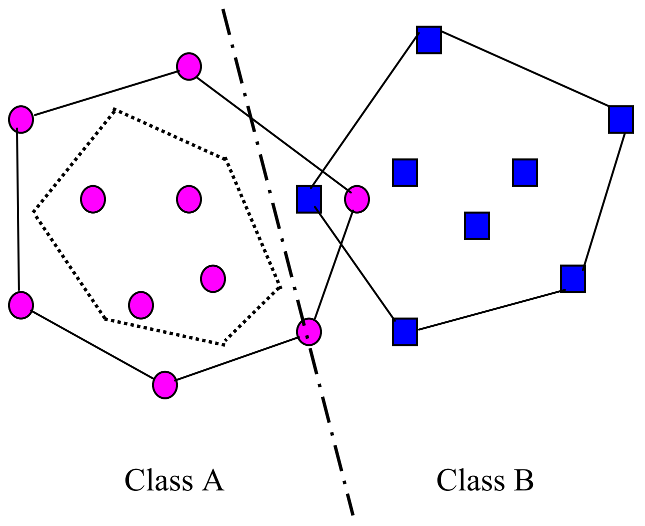

Figure 7. Geometric interpretation of the revised SVM

Consider a two-class classification problem. It is illustrated graphically in Figure 7. One class (say class B) has very few but very important data. The other class (say, class A) has many unimportant data. Since class B has important data, the important data is preserved. The unimportant data in class A will be reduced.

The standard SVM in the linearly separable case takes the following form.

Minimize $\quad \Phi(w) = \frac{1}{2} w^T w$

s.t.

$$[w_o^T x_i + b_o] \geq 1 \ , \qquad i = 1, 2, \cdots, N_P$$

$$-[w_o^T x_j + b_o] \geq 1 \ , \qquad j = 1, 2, \cdots, N_N$$

where N_P is the number of data points in the positive class

N_N is the number of data points in the negative class

Based on the above optimization problem, Lagrange's function is

$$L(w,b,\alpha,\beta) = \frac{1}{2} w^T w - \sum_{i=1}^{N_P} \alpha_i [(w^T x_i + b) - 1] - \sum_{j=1}^{N_N} \beta_j [-(w^T x_j + b) - 1] \tag{3.1}$$

Applying the Karush-Kuhn-Tucker (KKT) condition,

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{N_P} \alpha_i x_i + \sum_{j=1}^{N_N} \beta_j x_j = 0 \rightarrow w = \sum_{i=1}^{N_P} \alpha_i x_i - \sum_{j=1}^{N_N} \beta_j x_j \tag{3.2}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N_P} \alpha_i + \sum_{j=1}^{N_N} \beta_j = 0 \qquad \rightarrow \sum_{i=1}^{N_P} \alpha_i = \sum_{j=1}^{N_N} \beta_j \tag{3.3}$$

Here, we assume that

$$\sum_{i=1}^{N_P}\alpha_i = \sum_{j=1}^{N_N}\beta_j = 1 \text{ and } \alpha_i \geq 0 \text{ for } i = 1,2,....,N_P \;\&\; \beta_j \geq 0 \text{ for } j = 1,2,....,N_N, \tag{3.4}$$

which indicate that $\sum_{i=1}^{N_P}\alpha_i x_i \;\&\; \sum_{j=1}^{N_N}\beta_j x_j$ are convex combinations.

KKT complementary slackness conditions are

$$\alpha_i[(w^T x_i + b) - 1] = 0 \qquad \text{for } i = 1,2,....,N_P \tag{3.5}$$
$$\beta_i[-(w^T x_i + b) - 1] = 0 \qquad \text{for } j = 1,2,....,N_N$$

Note that support vectors correspond to $\alpha_i > 0$ for $i = 1,2,....,N_P$ and $\beta_j > 0$ for $j = 1,2,....,N_N$. Expanding (3.1),

$$L(w,b,\alpha,\beta) = \frac{1}{2}w^T w - \sum_{i=1}^{N_P}\alpha_i w^T x_i - b\sum_{i=1}^{N_P}\alpha_i + \sum_{i=1}^{N_P}\alpha_i + \sum_{j=1}^{N_N}\beta_j w^T x_j + b\sum_{j=1}^{N_N}\beta_j + \sum_{i=1}^{N_P}\beta_j \tag{3.6}$$

we obtain $L(w,b,\alpha,\beta) = \frac{1}{2}w^T w - \sum_{i=1}^{N_P}\alpha_i w^T x_i + \sum_{j=1}^{N_N}\beta_j w^T x_j + 2$ (3.7)

since $b(\sum_{j=1}^{N_N}\beta_j - \sum_{i=1}^{N_P}\alpha_i) = 0$ and $\sum_{i=1}^{N_P}\alpha_i = \sum_{j=1}^{N_N}\beta_j = 1$ from (3.4).

Substituting (3.2) into (3.7), (3.7) becomes

$$L(w,b,\alpha,\beta) = \frac{1}{2}[\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j x_i^T x_j + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j x_i^T x_j - 2\sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j]$$
$$- [\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j x_i^T x_j - \sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j] + [\sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j - \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j x_i^T x_j] + 2$$
$$= \sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j - \frac{1}{2}[\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j x_i^T x_j + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j x_i^T x_j] + 2$$

Set $L(w,b,\alpha,\beta) = Q(\alpha,\beta)$. Then, the dual problem becomes

$$\text{Max } Q(\alpha, \beta) = \sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j - \frac{1}{2}[\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j x_i^T x_j + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j x_i^T x_j]$$

s.t. (3.8)

$$\sum_{i=1}^{N_P}\alpha_i = 1$$

$$\sum_{j=1}^{N_N}\beta_j = 1$$

$$0 \le \alpha_i \text{ for } i = 1, 2, \cdots, N_P$$

$$0 \le \beta_j \text{ for } j = 1, 2, \cdots, N_N$$

Geometrically, the revised SVM takes the following form:

$$Min \quad \frac{1}{2}\left\|\sum_{i=1}^{N_P}\alpha_i x_i - \sum_{j=1}^{N_N}\beta_j x_j\right\|^2 \tag{3.9}$$

with the convex combination constraints, $\sum_{i=1}^{N_P}\alpha_i = 1$ and $\sum_{j=1}^{N_N}\beta_j = 1$, where $0 \le \alpha_j$ for $i = 1, 2, \cdots, N_P$ and $0 \le \beta_j$ for $j = 1, 2, \cdots, N_N$.

The function in (3.9) can be expanded as follows:

$$\frac{1}{2}[(\sum_{i=1}^{N_p}\alpha_i x_i - \sum_{j=1}^{N_N}\beta_j x_j)^T(\sum_{i=1}^{N_p}\alpha_i x_i - \sum_{j=1}^{N_N}\beta_j x_j)]$$
$$= \frac{1}{2}[\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j x_i^T x_j + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j x_i^T x_j - 2(\sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j)]$$

Then (3.9) is equivalent to

$$Max \quad \sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j x_i^T x_j - \frac{1}{2}(\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j x_i^T x_j + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j x_i^T x_j) \tag{3.10}$$

The objective function, (3.10), is exactly the same as the final form of the objective function in the revised SVM problem (3.8).

By the kernel trick, mentioned in section 2.1, the dual problem, (3.8), is changed to the following form:

$$\text{Max } Q(\alpha, \beta) = \sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j K(x_i,x_j) - \frac{1}{2}[\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j K(x_i,x_j) + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j K(x_i,x_j)]$$

s.t.

$$\sum_{i=1}^{N_P}\alpha_i = 1$$

$$\sum_{j=1}^{N_N}\beta_j = 1$$

$$0 \le \alpha_i \qquad \text{for } i = 1, 2, \cdots, N_P$$

$$0 \le \beta_j \qquad \text{for } j = 1, 2, \cdots, N_N$$

Geometrically, the revised SVM in the linearly inseparable case takes the following form:

$$Min \quad \frac{1}{2}\left\|\sum_{i=1}^{N_P}\alpha_i x_i - \sum_{j=1}^{N_N}\beta_j x_j\right\|^2$$

s.t.

$$\sum_{i=1}^{m} a_i = 1$$

$$\sum_{j=1}^{n} b_j = 1$$

$$0 \le \alpha_i \le 1 \qquad \text{for } i = 1, 2, \cdots, N_P$$

$$0 \le \beta_j \le \mu \qquad \text{for } j = 1, 2, \cdots, N_N$$

By imposing an upper bound on each multiplier, β_j, which is less than 1, the convex hull that consists of data points in the unimportant class shrinks. In contrast, the convex hull in the important class is preserved. The resulting revised SVM for the linearly inseparable case forms as follows:

$$\text{Max } Q(\alpha,\beta) = \sum_{i=1}^{N_P}\sum_{j=1}^{N_N}\alpha_i\beta_j K(x_i,x_j) - \frac{1}{2}[\sum_{i=1}^{N_P}\sum_{j=1}^{N_P}\alpha_i\alpha_j K(x_i,x_j) + \sum_{i=1}^{N_N}\sum_{j=1}^{N_N}\beta_i\beta_j K(x_i,x_j)]$$

s.t.

$$\sum_{i=1}^{N_P}\alpha_i = 1$$

$$\sum_{j=1}^{N_N}\beta_j = 1$$

$$0 \le \alpha_i \le 1 \qquad \text{for } i = 1, 2, \cdots, N_P$$

$$0 \le \beta_j \le \mu \qquad \text{for } j = 1, 2, \cdots, N_N$$

3.3 Examples for applying RSVM

3.3.1 Description of Examples

An example for the linearly separable case with two sets of two-dimensional data, which are the training and testing sets, is created. There are 164 training data points and 404 testing data points in this example. These training and testing data are illustrated in Figures 8 and 9.

As an example of the inseparable case, a synthetic two-class data set on two dimensions is used [Ripley, 1994]. This data set, which was created by Ripley, has been

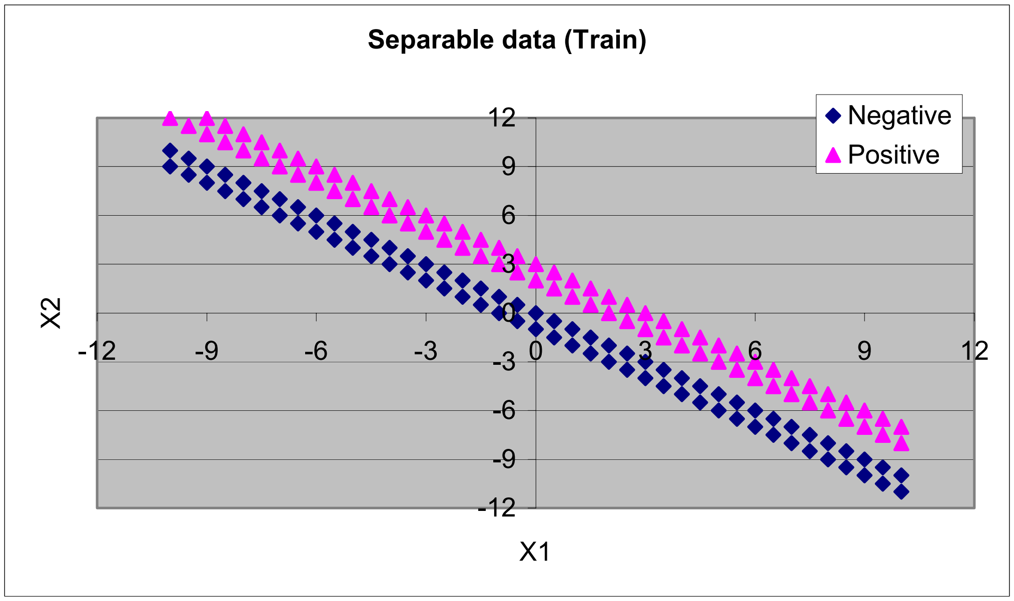

Figure 8. Geometric interpretation of the revised SVM

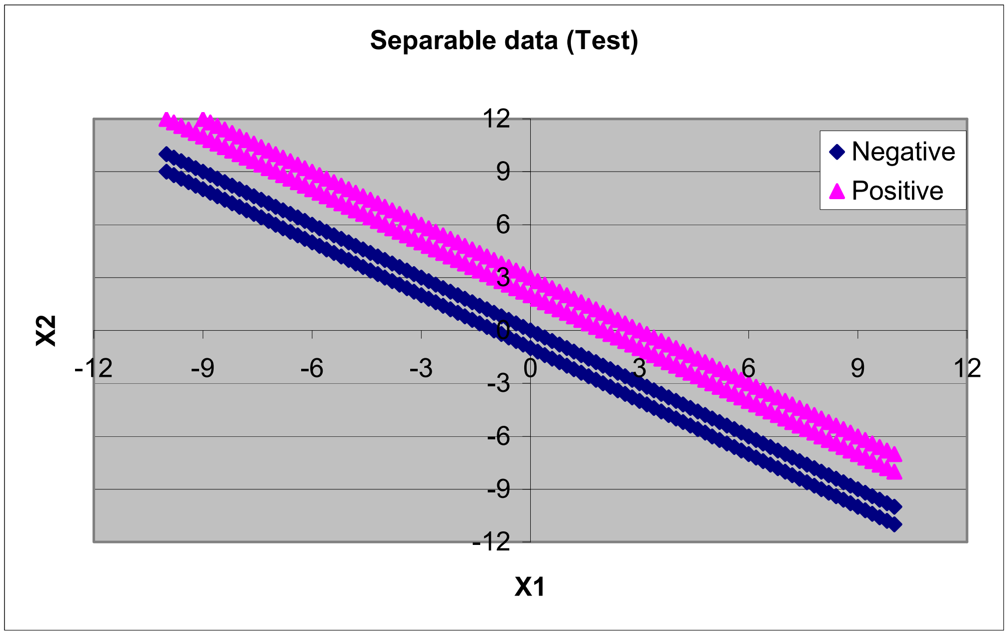

Figure 9. Linearly separable data (Test)

widely used in many publications (e.g., [Osuna and Girosi, 1998; Ripley, 1996]). This set contains linearly inseparable data with 250 data points for the training set and 1000 data points for the test set. These training and testing data points are illustrated in Figures 10 and 11.

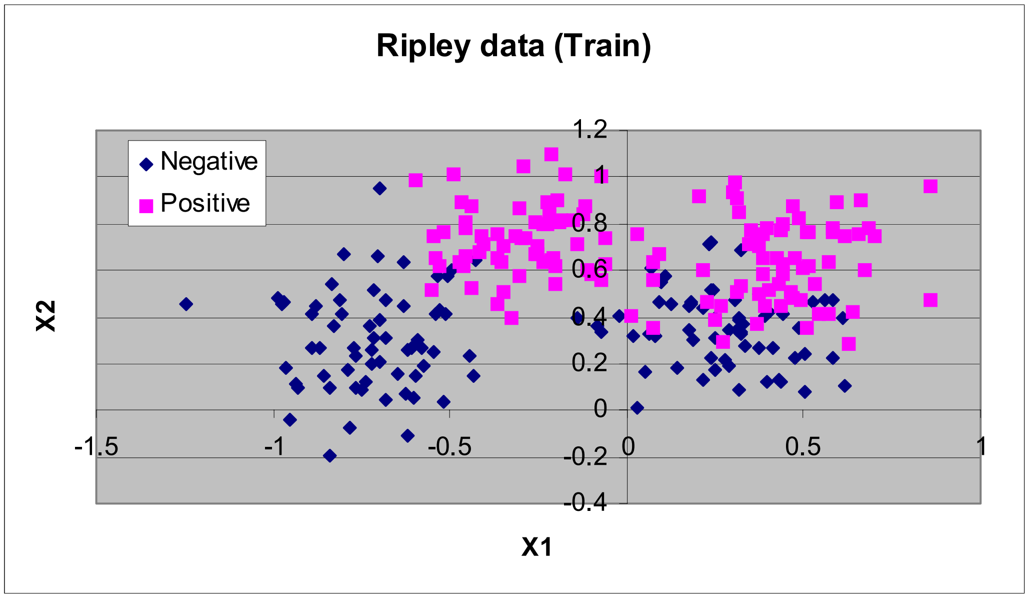

Figure 10. Training data in Ripley's data set

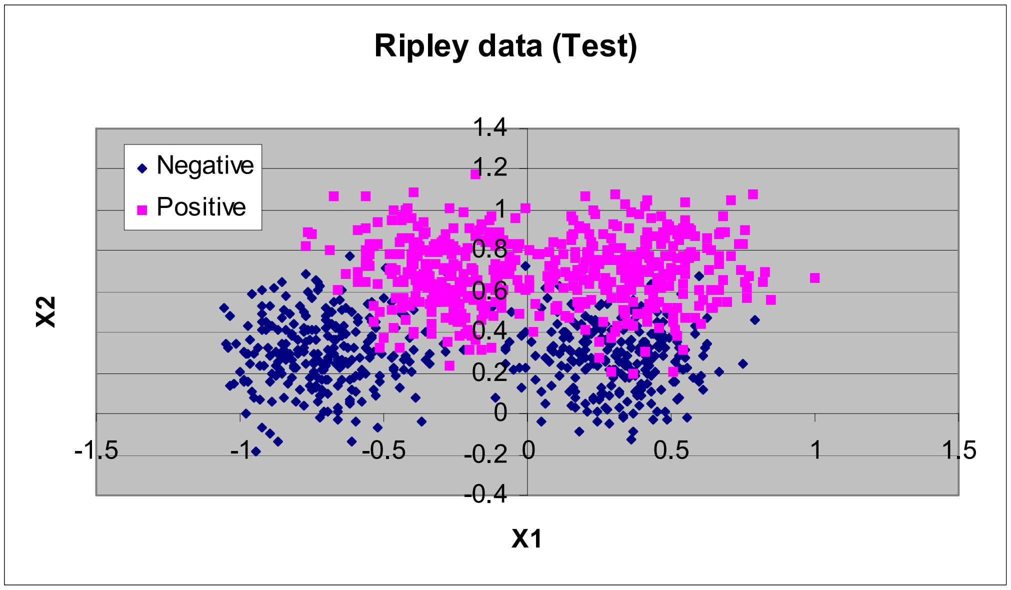

Figure 11. Testing data in Ripley's data set

3.3.2 Results for both cases

Several criteria, such as the number of support vectors, ratio of support vectors, generalization rate, and CPU time, are considered in these experiments. The number of support vectors is counted for the positive and negative example cases. The ratio of the support vectors (for both cases) against the whole data set is also computed. The generalization rate indicates the rate that the decision function classifies the test data correctly. CPU time is measured for the training speed. Under these criteria, the standard SVM and the revised SVM are compared in the separable and inseparable cases.

Results produced from the revised SVM and the standard SVM are shown in Table 2. Their comparisons are illustrated in Figures 12 and 13.

Table 2. Comparison between revised SVM and standard SVM in both cases

	Separable case		Inseparable case	
	Revised SVM	Standard SVM	Revised SVM	Standard SVM
number of SVs in positive side	1	64	1	53
number of SVs in negative side	2	64	2	53
Ratio of SVs (%)	1.8	78	1.2	42.4
Gen. Rate (%)	46.8	71.53	71.6	55.7
CPU time (sec)	7.4	16.7	30.9	13.2

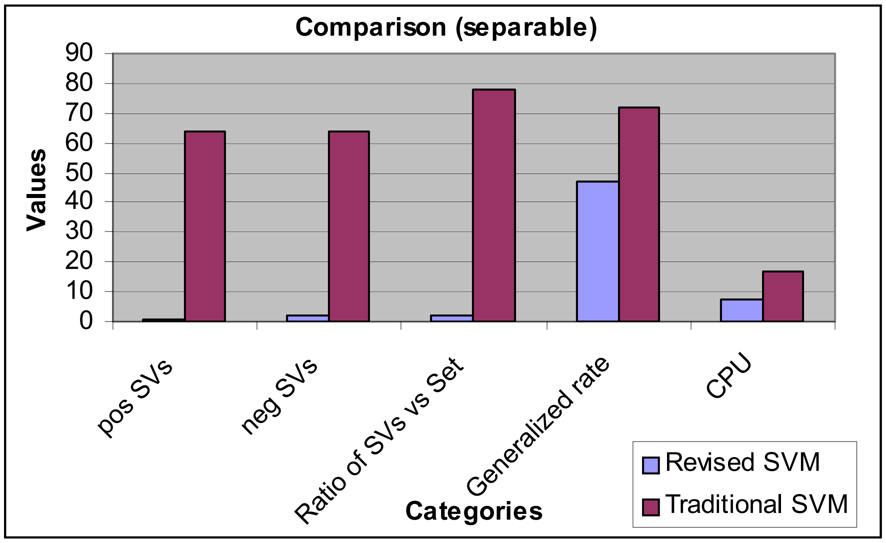

Figure 12. Comparisons in categories with linearly separable data

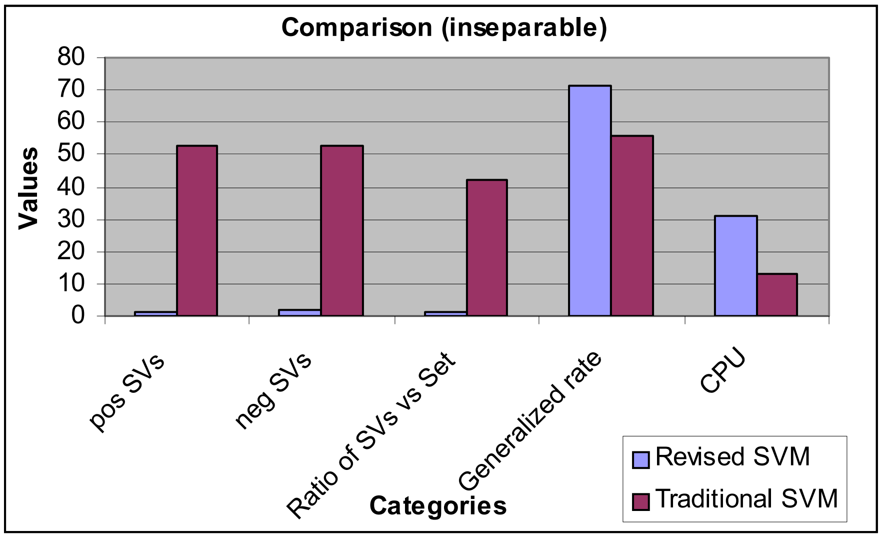

Figure 13. Comparisons in categories with linearly inseparable data

The number of support vectors is significantly diminished so that the portion of the support vectors in the whole data set is lowered in both cases. The standard SVM has a better generalization rate in the separable case, whereas the revised SVM has a better rate in the inseparable case. The revised SVM might have a worse generalization error because the supporting hyperplane on the negative side is pulled into the negative class region. In terms of computing time, the revised SVM and standard SVM have an advantage over the separable case and inseparable case, respectively.

3.4 Incremental learning with revised SVM with filters (IRSVMF)

The standard SVM can be applied to the incremental learning with the format as shown in section 2.3. When a SVM is utilized as a classifier, it forms the incremental SVM. The resulting decision function from training a SVM depends on its support vectors. The support vectors have nice properties for summarizing data characteristics. Training SVM on the support vectors alone produces the same decision function as training on the whole data set. Thus, if the last training set (last batch) contains all the examples that are support vectors in the non-incremental SVM, the result in the incremental approach is the same as the one in the non-incremental approach.

There are two approaches in the incremental SVM with respect to a data-driven format. One of them is the batch approach, and the other is the on-line approach. In the batch approach, the whole data set is partitioned into several batches, if the whole data set is available at present. Each batch is formed in a certain time frame if the data set is provided periodically. Generally speaking, SVM is trained on the new data and the support vectors produced from the previous learning step. In the on-line approach, all data come into a certain algorithm based on SVM and are trained on-line.

In existing literature, several approaches have been explored in the incremental SVM learning framework. Syed et al. (1999) shows performance of the true incremental SVM approach with the standard SVM in several data sets. They report that there is no drop in the prediction accuracy of the SVM even when it is used in the incremental framework. Ruping (2001) tried to remove old support vectors by adding another penalty function in the objective function.

Most publications in the on-line approach track the value of alpha (multipliers in KKT conditions) and its change. Hence, based on the changed value of the alphas, a new data point is added in the next step.

Cauwenberghs and Poggio (2001) propose an on-line formulation for the SVM. If the solutions are computed whenever a new data point is added, the algorithm wastes computing time and memory. Thus, their algorithm keeps the KKT optimality condition at every step. They store support vectors and update their multipliers' values (variables in the feature space), producing a new solution. This means that the algorithm should test and potentially train on all the data. It is quite fast and useful if the number of support vectors is relatively small compared to the size of whole data. Since the resulting storage requirement is determined by the number of support vectors, if the number of support vectors are relatively high, this formulation is limited.

As a variant of Cauwenberghs and Poggio's approach, Simple SVM is proposed by Vishwanathan, Smola, and Murty (2003). Their algorithm decomposes the kernel matrix by LDL decomposition, and it relaxes the box constraints imposed on the multipliers temporarily ignoring the KKT optimality conditions. Solving the problem without the box constraints, a new solution is checked by the box constraints to identify support vectors or error vectors, which are located in between supporting hyperplanes. In short, as this algorithm reduces constraints (i.e., the box constraints), it tries to improve computing time. However, the number of variables is increased if the number of support vectors is relatively large compared to the whole data set. Then, there is no significant reduction of computing time, though the algorithm reduces the box constraints. Thus, this algorithm performs especially well only when the number of support vectors is quite small.

However, these approaches will not work properly if the support vectors are accumulated. After the data set is trained, the support vectors have multipliers with a positive value. In order to obtain these multipliers' value, the on-line learning algorithm should train a certain amount of data points like in the batch approach. The on-line approach also stores support vectors produced in the previous training step similar to the batch approach. Thus, the on-line approach can be considered as a variant of the incremental approach. An incremental SVM learning algorithm with input data that are available only on periodical time windows will be called the "on-line algorithm."

The structure of the incremental approach with SVM is shown in Figure 14.

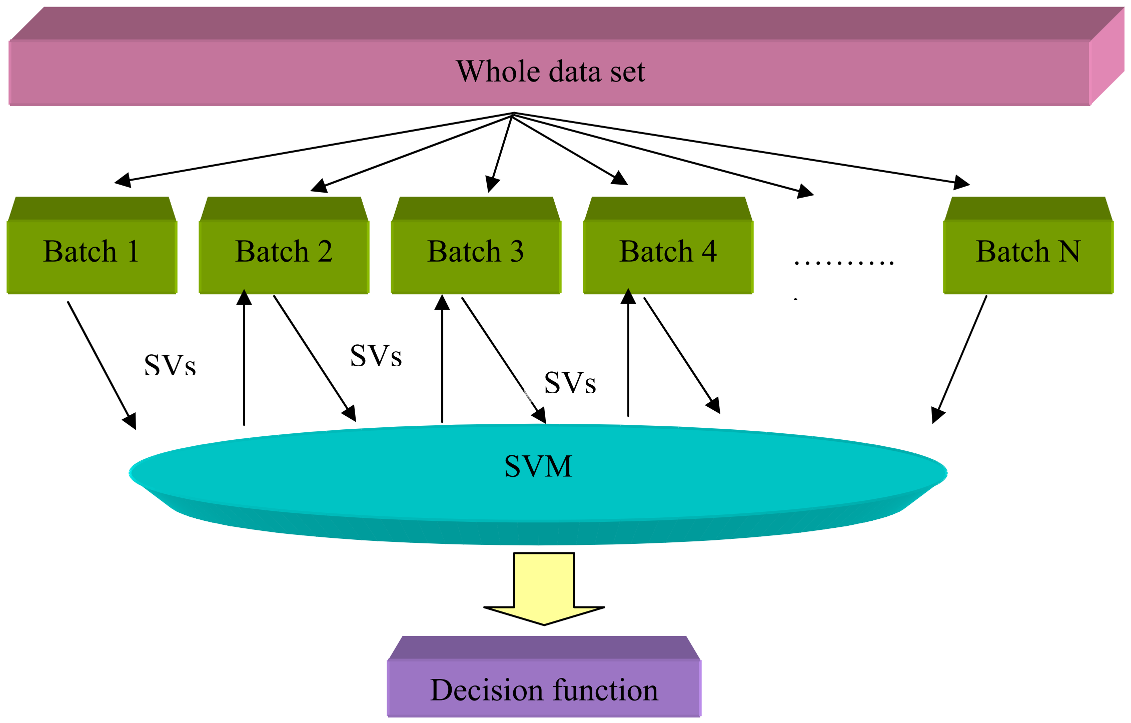

Figure 14. Incremental SVM learning

If the whole data set is available, the set is partitioned into several subsets, which are called "batches". If the input data are available at a specific interval, each batch contains input data of each time interval. If input data are injected into the system on-line, input data are filled up to a certain level of size. In the literature (e.g., [Demeniconi and Gunopulos, 2001]), the size of the batch has been used to be arbitrarily determined. The appropriate batch size can be obtained in terms of a trade-off between generalization error rate and computing time. For details, refer to [Son et al., 2005].

SVM trains the input data in the first batch. The support vectors extracted from SVM are added to the next batch (i.e., batch 2) in order to obtain a decision function. Then, data in the second batch are trained to adapt the decision function. This iterative procedure is repeated until all batches are trained. After all data are trained, the final decision function is made for classification.

In this study, we utilize the revised SVM instead of the standard SVM for training data in the incremental learning process. A filter is created to remove most likely unimportant data prior to training process. The structure of the proposed incremental learning process is shown in Figure 15.

If the whole data set is available, it is divided into several batches. In on-line setting, only the first batch is created. As described in section 2.3, the optimal batch size will be determined in terms of trade-off between generalization error and computing time. Data in the first batch is trained by the revised SVM identifying the support vectors. These support vectors are included in the second batch. The supporting hyperplane based on the support vectors of the unimportant class plays a role as a filter.

Because most of the unimportant data are located in the halfspace of the supporting hyperplane of the unimportant class in the feature space, this supporting hyperplane in Fig. 16 is a good yardstick for removing the possible unimportant data before training.

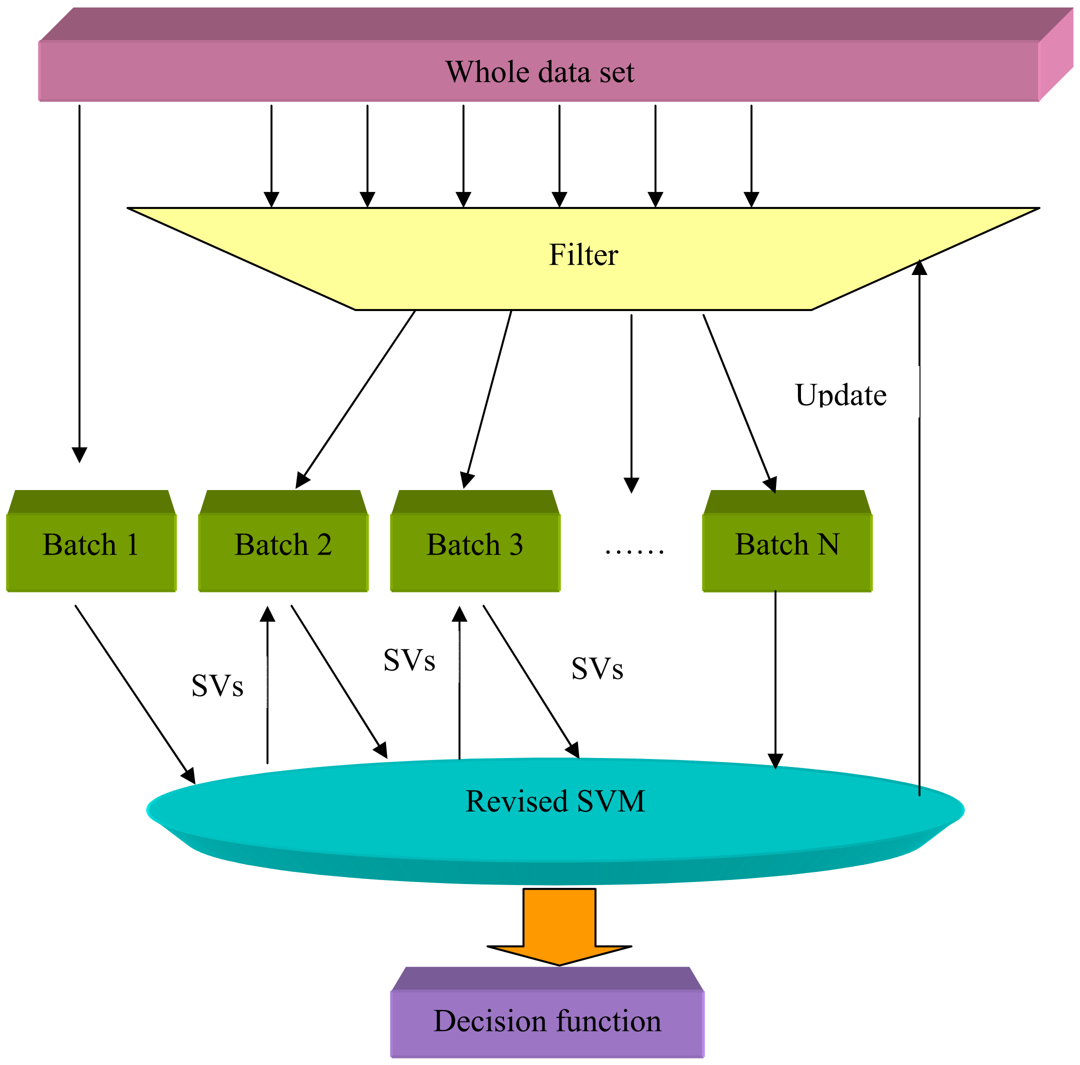

Figure 15. Incremental learning with revised SVM and a filter

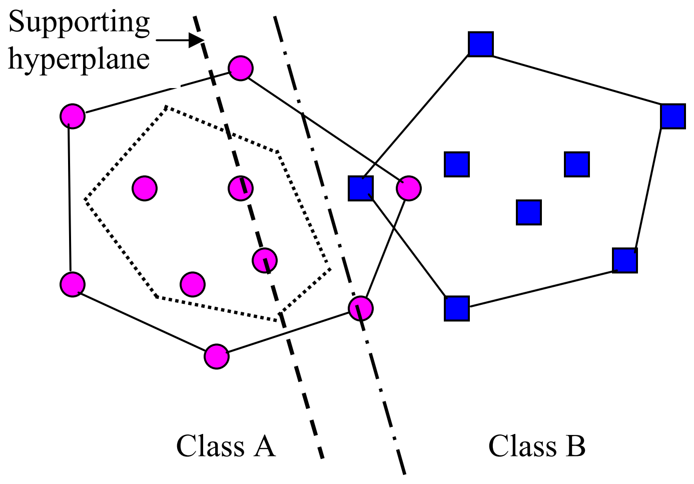

Figure 16. Supporting hyperplane as a filter

In Figure 16, the supporting hyperplane is a line passing through the two support vectors of the reduced convex hull. If the supporting hyperplane is used as a filter, three points on the right side of the supporting hyperplane are passed in the filter. These points will be used to construct the decision function in SVM because the optimal separating hyperplane is computed by support vectors on both classes. Data points passing this filter are put in the next batch until the size of the batch is filled up to the optimal batch size. This filter is updated in every batch as the optimal separating hyperplane is modified in every batch. Thus, this approach requires fewer batches than the traditional batch method in the incremental learning procedure. Hence, a new algorithm for incremental learning with revised SVM and a filter is proposed as follows:

===

```
Step 1.  Determine the optimal batch size based on generalization error rate and
         computing time.
Step 2.  Train data in the first batch by revised SVM.
Step 3.  Store data points representing support vectors in the next batch.
Step 4.  Inject a new point to the filter.
Step 5.  If a new data point is located outside of the filter, then discard the data point
         Otherwise, store it in the next batch.
Step 6.  If the size of data in a batch is equal to the optimal batch size, go to the next step
         Otherwise, go to step 4.
Step 7.  Train the data in the batch and obtain support vectors, and corresponding
         decision function.
Step 8.  Update the filter. Go to step 3.
```

===

CHAPTER 4

INCREMENTAL REVISED SVR

The Support Vector Regression (SVR) discussed in section 2.2, has been utilized in time series prediction [Cao, 2003], financial forecasting problem [Yang, 2002], and other scientific applications [Song et al., 2002]. The standard SVR may produce distorted performance if noisy data (e.g., outliers) exist. Furthermore, the standard SVR doesn't perform properly if a large-scale data set is used as input. This is because the standard SVR requires more memory space and computing time as the standard SVM does. Therefore, the standard SVR should be also revised to produce better performance in large-scale data sets. In SVR, it is critical to reduce the support vectors because it is practically impossible to train all data points simultaneously when a huge data set is provided and because support vectors are accumulated when the incremental learning approach is applied. In this chapter, the standard SVR is revised to overcome limitations that are mentioned above, extending previous work by Bi and Bennett (2003). Then the revised SVR concept is applied to the incremental approach. For dynamic data driven applications (DDDAS), the filter concept which is developed in chapter 3 is also used in the incremental SVR approach.

4.1 A Reduced Convex Hull SVR (RCH-SVR)

Consider the training data set $\{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$ where $x_i \in \Re^n$ and $y_i \in \Re$. A hard ε-tube for a fixed $\varepsilon > 0$ is defined as the hyperplane $y = w^T x + b$ satisfying $y_i - < w, x_i > - b \leq \varepsilon$ and $< w, x_i > + b - y_i \leq \varepsilon$ where $i = 1, 2, \cdots, m$. As shown in Figure 3 in chapter 2, the smallest tube, ε_0-tube, can be obtained from the following optimization problem [Bi and Bennett, 2003].

$$\min_{w,b,\varepsilon} \quad \varepsilon$$

s.t.

$$y_i - < w, x_i > - b \leq \varepsilon \qquad i = 1, 2, \cdots, m$$

$$< w, x_i > + b - y_i \leq \varepsilon \qquad i = 1, 2, \cdots, m$$

Based on the determination of ε which is user-defined and ε_0 which is obtained from the above optimization problem, there are three possible cases for $\varepsilon > 0$ in the dual space as shown in Figure 17.

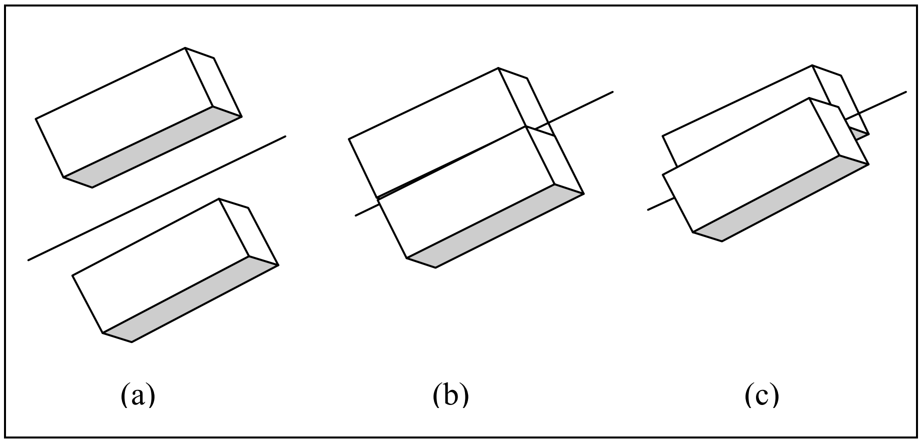

Figure 17. Three cases of support vector regression

Based on the relationship between ε and ε_0, Bi and Bennett (2003) investigated when a hard ε-tube or a soft ε-tube exists. First, a hard ε-tube exist, when $\varepsilon > \varepsilon_0$. In this case, the convex hulls of both classes are linearly separable as shown in Figure 17 (a). This indicates that there exists a (w, b) that satisfies the constraints of (4.1).

$$\begin{aligned} &y_i - < w, x_i > - b \le \varepsilon && i = 1, 2, \cdots, m \\ &< w, x_i > + b - y_i \le \varepsilon && i = 1, 2, \cdots, m \end{aligned} \tag{4.1}$$

Next theorem can be used to give conditions for the existence of a hard ε-tube.

Theorem 1 (Gale's Theorem of the alternative)

Exactly one of the following two systems has a solution.

(System 1) $Ax \le b$

(System 2) $y^T A = 0$, $y^T b < 0$, $y \ge 0$

Proof.

Refer to [Bazaraa, 1990]

Consider the two sets of the augmented vectors, S^+ and S^- as follows:

$$\begin{aligned} S^+ &= \{\begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}, i = 1, 2, \cdots, m\} \\ S^- &= \{\begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}, i = 1, 2, \cdots, m\} \end{aligned} \tag{4.2}$$

By Gale's Theorem, a hard ε-tube exists for $\varepsilon > 0$ if and only if the following system in (u, v) has no solution, given the two sets as shown in (4.2).

$$\sum_i u_i (y_i + \varepsilon) - \sum_i v_i (y_i - \varepsilon) < 0; \tag{4.3}$$

$$\sum_i u_i x_i = \sum_i v_i x_i ; \sum_i u_i = 1; \sum_i v_i = 1; \; u_i \geq 0; \; v_i \geq 0$$

It means that a hard ε-tube exists for $\varepsilon > 0$ if and only if the convex hulls of the two sets given in (4.2) are linearly separable.

Since a regression problem in the primal formulation becomes a classification problem in the dual formulation, the reduced convex hull concept can be directly applied to the standard SVR.

Thus, the reduced convex hull SVR seeks the closest points in the reduced convex hulls of the two augmented data sets. Consider the convex hulls of the two sets given in (4.2) as follows:

$$\begin{aligned} Conv(S^+) &= \{\sum_{i \in S^+} u_i z_i^+ \mid \sum_i u_i = 1, \; u_i \geq 0\} \\ Conv(S^-) &= \{\sum_{i \in S^-} v_i z_i^- \mid \sum_i v_i = 1, \; v_i \geq 0\} \end{aligned} \tag{4.4}$$

where $z_i^+ = \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}$ & $z_i^- = \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}$

The closest points of the two convex hulls in (4.4) can be obtained by solving the following optimization problem in (u, v).

$$\begin{aligned}
&\underset{u,v}{Min} \quad \frac{1}{2}\left\| \sum_i \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix} u_i - \sum_i \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix} v_i \right\|^2 \\
&\text{s.t.} \\
&\qquad \sum_{i=1}^{m} u_i = 1 \\
&\qquad \sum_{i=1}^{m} v_i = 1 \\
&\qquad 0 \le u_i \ \text{ for } i = 1, 2, \cdots, m \\
&\qquad 0 \le v_j \ \text{ for } j = 1, 2, \cdots, m
\end{aligned} \tag{4.5}$$

Second, a hard ε-tube exists, which is the same as the smallest hard tube when $\varepsilon = \varepsilon_0$. In this case, the convex hulls of the two sets are not strictly separable as shown in Figure 17(b) with a hard ε_0-tube separating the two convex hulls. However, it is practically impossible that user selects the smallest ε_0-tube, and that there is no outlier in input data.

Third, the convex hulls of two sets are overlapped when $\varepsilon < \varepsilon_0$ as shown in Figure 17 (c). This is called soft ε-tube. In applications, this case is very common and easily found. To resolve the soft ε-tube case, the reduced convex hull concept is very useful. Consider the reduced convex hulls of sets S^+ and S^- as follows:

$$\begin{aligned}
Conv(S^+) &= \{\sum_{i \in S^+} u_i z_i^+ \mid \sum_i u_i = 1, \ 0 \le u_i \le C\} \\
Conv(S^-) &= \{\sum_{i \in S^-} v_i z_i^- \mid \sum_i v_i = 1, \ 0 \le v_i \le C\}
\end{aligned} \tag{4.7}$$

where $z_i^+ = \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}$ & $z_i^- = \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}$

Since these convex hulls are not separable, the convex hulls are reduced based on C. Thus, the mathematical programming problem of RCH-SVR is presented as follows:

$$\begin{aligned}
&\underset{u,v}{Min} \quad \frac{1}{2}\left\| \sum_i \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix} u_i - \sum_i \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix} v_i \right\|^2 \\
&\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (4.8) \\
&\qquad \sum_{i=1}^{m} u_i = 1 \\
&\qquad \sum_{i=1}^{m} v_i = 1 \\
&\qquad 0 \le u_i \le C \qquad \text{for } i = 1, 2, \cdots, m \\
&\qquad 0 \le v_i \le C \qquad \text{for } i = 1, 2, \cdots, m
\end{aligned}$$

Thus, solutions from the optimization problem in (4.8) separate two reduced convex hulls by bisecting the connection of the closest points in each class.

4.2 A Revised SVR (RSVR)

In section 4.1, three cases based on the relationship between ε and ε_0 are considered. It is nearly impossible to construct the hard ε-tube in real-world applications because the size of data to be considered is huge. Thus, only the soft ε-tube case will be investigated. The resulting formulation can be easily applied to the soft ε-tube case.

Although it is quite effective to reduce the number of support vectors when the reduced convex hull SVR approach is utilized, the drawbacks of this approach still exist.

First, the upper bound of variables C is fixed by the user. This means that the convex hulls shrink at the same rate on both sides (lower and upper parts of the regression line). Thus, this approach might not reflect recent trends. Second, when the reduced convex hull SVR is applied to the incremental approach, strict application of the reduced convex hull SVR (e.g., at the high shrink rate) might lose essential information (i.e., some support vectors) extracted from the previous step. Hence the upper bound of variables C must be flexible in order to cope with the above situations.

Consider the following optimization problem mentioned in (2.7) of chapter 2.

$$
\begin{aligned}
&Min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^{'}) \\
&\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (4.9) \\
&\qquad y_i - w^T x_i - b \le \varepsilon + \xi_i \qquad i = 1, \cdots, m \\
&\qquad w^T x_i + b - y_i \le \varepsilon + \xi_i^{'} \qquad i = 1, \cdots, m \\
&\qquad \xi_i \ , \xi_i^{'} \ \ge 0 \qquad i = 1, \cdots, m
\end{aligned}
$$

In (4.9), the objective function and constraints can be reformulated as follows:

$$
\begin{aligned}
&Min \quad \frac{1}{2}w^T w \ + C\sum_{i=1}^{m}(\xi_i + \xi_i^{'}) \\
&\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (4.10) \\
&\qquad (y_i - \varepsilon) - w^T x_i - b \le \xi_i \qquad i = 1, \cdots, m \\
&\qquad w^T x_i + b - (y_i + \varepsilon) \le \xi_i^{'} \qquad i = 1, \cdots, m \\
&\qquad \xi_i \ , \xi_i^{'} \ \ge 0 \qquad i = 1, \cdots, m
\end{aligned}
$$

By introducing Lagrangian multipliers α_i, $\alpha_i^{'}$, η_i, and $\eta_i^{'}$, the Lagrangian function of (4.10) can be constructed as follows:

$$L = \frac{1}{2} w^T w + C \sum_{i=1}^{m} (\xi_i + \xi_i^{'}) - \sum_i \alpha_i (\xi_i + w^T x_i + b - (y_i - \varepsilon))$$

$$- \sum_i \alpha_i^{'} (\xi_i^{'} - w^T x_i - b + (y_i + \varepsilon)) - \sum_i (\eta_i \xi_i + \eta_i^{'} \xi_i^{'}) \qquad (4.11)$$

Differentiating the Lagrangian function with respect to w, b, ξ_i, and $\xi_i^{'}$, the optimality conditions can be obtained as follows:

$$\frac{\partial L}{\partial w} = w - \sum_i (\alpha_i - \alpha_i^{'}) x_i = 0 \rightarrow \qquad w = \sum_i (\alpha_i - \alpha_i^{'}) x_i \qquad (4.12)$$

$$\frac{\partial L}{\partial b} = \sum_i (\alpha_i - \alpha_i^{'}) = 0 \qquad (4.13)$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \qquad \rightarrow \qquad \eta_i = C - \alpha_i \qquad (4.14)$$

$$\frac{\partial L}{\partial \xi_i^{'}} = C - \alpha_i^{'} - \eta_i^{'} = 0 \qquad \rightarrow \qquad \eta_i^{'} = C - \alpha_i^{'} \qquad (4.15)$$

By substituting w, η_i, and $\eta_i^{'}$ which are obtained from (4.12), (4.14), and (4.15) into(4.11), the resulting function F becomes

$$F = \frac{1}{2} \sum_i \sum_j (\alpha_i - \alpha_i^{'})(\alpha_j - \alpha_j^{'}) x_i x_j + C \sum_{i=1}^{m} (\xi_i + \xi_i^{'})$$

$$- \sum_i \alpha_i (\xi_i + \sum (\alpha_j - \alpha_j^{'}) x_j x_i + b - (y_i - \varepsilon))$$

$$-\sum_i \alpha_i^{'}(\xi_i^{'} - \sum (\alpha_j - \alpha_j^{'})x_j x_i - b + (y_i + \varepsilon))$$

$$-\sum_i [(C-\alpha_i)\xi_i + (C-\alpha_i^{'})\xi_i^{'}) \tag{4.16}$$

Since $\sum_i (\alpha_i - \alpha_i^{'}) = 0$ from (4.13) and cancellations, (4.16) becomes

$$F = -\frac{1}{2}\sum_i \sum_j (\alpha_i - \alpha_i^{'})(\alpha_j - \alpha_j^{'})x_i x_j$$

$$+\sum (\alpha_i - \alpha_i^{'})y_i - \sum (\alpha_i + \alpha_i^{'})\varepsilon \tag{4.17}$$

Then equation (4.17) is kernelized with respect to x_i, and with the constraints from the optimality conditions the following dual optimization problem is constructed as follows:

$$\begin{aligned}
&Max\ \ F(\alpha,\alpha^{'}) \\
&= -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i^{'})(\alpha_j - \alpha_j^{'})K(x_i, x_j) + \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i^{'}) - \varepsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^{'}) \\
&\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (4.18) \\
&\qquad \sum_{i=1}^{m}(\alpha_i - \alpha_i^{'}) = 0 \\
&\qquad 0 \le \alpha_i \le C \ \text{ for } i = 1,2,\cdots,m \\
&\qquad 0 \le \alpha_i^{'} \le C \ \text{ for } i = 1,2,\cdots,m
\end{aligned}$$

In chapter 3, the revised SVM concept is effectively applied to the classification problem while the shrink rates on two classes are different. Since the regression problem in the primal space becomes a classification problem in the dual space, the revised SVM concept is naturally applied to this regression problem. There exists a difference between the revised SVM and the revised SVR. In the revised SVM, the target value indicates the class label such as $+1$ and -1 in a binary classification problem. In contrast, in the revised SVR the target value is a real value instead of the class label. Thus, the appropriate expressions should be modified to accommodate the revised SVR case.

Consider two sets of the augmented vectors, S^+ and S^-.

$$\begin{aligned} S^+ &= \{\begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}, i = 1, 2, \cdots, m\} \\ S^- &= \{\begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}, i = 1, 2, \cdots, m\} \end{aligned} \tag{4.19}$$

where $z_i^+ = \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}$ & $z_i^- = \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}$

Then their revised convex hulls are as follows:

$$\begin{aligned} Conv(S^+) &= \{\sum_{i \in S^+} u_i z_i^+ \mid \sum_i u_i = 1,\ 0 \le u_i \le C1\} \\ Conv(S^-) &= \{\sum_{i \in S^-} v_i z_i^- \mid \sum_i v_i = 1,\ 0 \le v_i \le C2\} \end{aligned} \tag{4.20}$$

where $0 < C1,\ C2 \le 1$

Since these convex hulls generally are not separable, the convex hulls are reduced based on $C1$ and $C2$. Thus, the mathematical expression of the revised SVR in the dual space is presented as follows:

$$\underset{u,v}{Min} \quad \frac{1}{2}\left\| \sum_i \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix} u_i - \sum_i \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix} v_i \right\|^2$$

s.t. (4.21)

$$\sum_{i=1}^{m} u_i = 1$$

$$\sum_{i=1}^{m} v_i = 1$$

$$0 \le u_i \le C1 \quad \text{for } i = 1, 2, \cdots, m$$

$$0 \le v_i \le C2 \quad \text{for } i = 1, 2, \cdots, m$$

The objective of this problem is to obtain the closest points between the different reduced convex hulls of the two sets. Then the separating hyperplane can be obtained as the bisector of the segment defined by the two closest points of the reduced convex hulls. This separating hyperplane becomes the regression function in the primal space.

Different from the classification problem, the regression problem uses the augmented vectors $z_i^+ = \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}$ & $z_i^- = \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}$ which form the convex combinations in both sides. Since the regression problem seeks to predict the target based on the input data, only the input data are kernelized in the revised convex hull concept. Thus, the augmented vectors $z_i^+ = \begin{pmatrix} x_i \\ y_i + \varepsilon \end{pmatrix}$ & $z_i^- = \begin{pmatrix} x_i \\ y_i - \varepsilon \end{pmatrix}$ are transformed to the augmented feature space by the transformation function $\phi(\bullet)$ so that the resulting images in the feature space are $\begin{pmatrix} \phi(x_i) \\ y_i + \varepsilon \end{pmatrix}$ & $\begin{pmatrix} \phi(x_i) \\ y_i - \varepsilon \end{pmatrix}$. Because the target y_i depends on the input x_i, the weights are obtained with the input x_i, and those are applied to the target y_i. Thus, the objective function in (4.8) is reformulated as follows:

$$\frac{1}{2}\left\|\sum_i u_i x_i - \sum_i v_i x_i\right\|^2 + \sum_i u_i\,(y_i+\varepsilon) - \sum_i v_i\,(y_i-\varepsilon) \tag{4.22}$$

The formulation (4.22) can be expanded as follows:

$$\frac{1}{2}\left\|\sum_i (u_i - v_i)x_i\right\|^2 + \sum_i u_i\,(y_i+\varepsilon) - \sum_i v_i\,(y_i-\varepsilon)$$

$$\frac{1}{2}(\sum_i (u_i - v_i)x_i)(\sum_j (u_j - v_j)x_j) + \sum_i u_i\,(y_i+\varepsilon) - \sum_i v_i\,(y_i-\varepsilon)$$

$$\frac{1}{2}(\sum_i \sum_j (u_i - v_i)(u_j - v_j)x_i^T x_j) + \sum_i u_i\,(y_i+\varepsilon) - \sum_i v_i\,(y_i-\varepsilon) \tag{4.23}$$

In (4.23), the term $x_i^T x_j$ is transformed by a transformation function, $\phi(\bullet)$ resulting to $\phi(x_i)\phi(x_j)$. By Mercer's theorem, the inner-product kernel $K(x_i, x_j) = \phi(x_i)\phi(x_j)$ can be computed. Thus, the kernelized expression of (4.23) becomes

$$\frac{1}{2}(\sum_i \sum_j (u_i - v_i)(u_j - v_j)K(x_i\,, x_j))$$
$$+\sum_i u_i\,(y_i+\varepsilon) - \sum_i v_i\,(y_i-\varepsilon) \tag{4.24}$$

By making the change of variables, $u_i = \alpha_i^{'}$ and $v_i = \alpha_i$ in the above expression, (4.24) becomes

$$\frac{1}{2}(\sum_i \sum_j (\alpha_i - \alpha_i^{'})(\alpha_i - \alpha_i^{'})K(x_i\,, x_j))$$
$$+\sum_i \alpha_i^{'}\,(y_i+\varepsilon) - \sum_i \alpha_i\,\,(y_i-\varepsilon) \tag{4.25}$$

Since the objective function is minimized in (4.21), when we change from the minimization problem to the maximization problem with the function in (4.25), (4.25) becomes

$$Max\ \ -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i-\alpha_i^{'})(\alpha_j-\alpha_j^{'})K(x_i,x_j) + \sum_{i=1}^{m}y_i(\alpha_i-\alpha_i^{'})-\varepsilon\sum_{i=1}^{m}(\alpha_i+\alpha_i^{'}) \tag{4.26}$$

The objective function in (4.26) is exactly the same as the objective function in (4.18). Thus, the revised convex hull concept can be directly applied without any problem.

Accordingly, the optimization problem in (4.21) becomes

$$Max\ \ F(\alpha,\alpha^{'})$$

$$=-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i-\alpha_i^{'})(\alpha_j-\alpha_j^{'})K(x_i,x_j)+\sum_{i=1}^{m}y_i(\alpha_i-\alpha_i^{'})-\varepsilon\sum_{i=1}^{m}(\alpha_i+\alpha_i^{'})$$

s.t. (4.27)

$$\sum_{i=1}^{m}\alpha_i=1$$

$$\sum_{i=1}^{m}\alpha_i^{'}=1$$

$$0\le\alpha_i\le C1 \quad \text{for } i=1,2,\cdots,m$$

$$0\le\alpha_i^{'}\le C2 \quad \text{for } i=1,2,\cdots,m$$

Thus, solutions from the optimization problem in (4.27) separate two differently reduced convex hulls by bisecting the segment that connects the closest points in each class. The appropriate choices of $C1$ and $C2$ depend on the user. The general criterion to determine two upper bounds can be described in the following manner.

If a convex hull shrinks too much, the separating hyperplane leans to the reduced convex hull in dual space. In contrast, if the other convex hull shrinks much more than the previous convex hull, the separating hyperplane is inclined more to the other convex hull. There are three possible cases as shown in Figure 18.

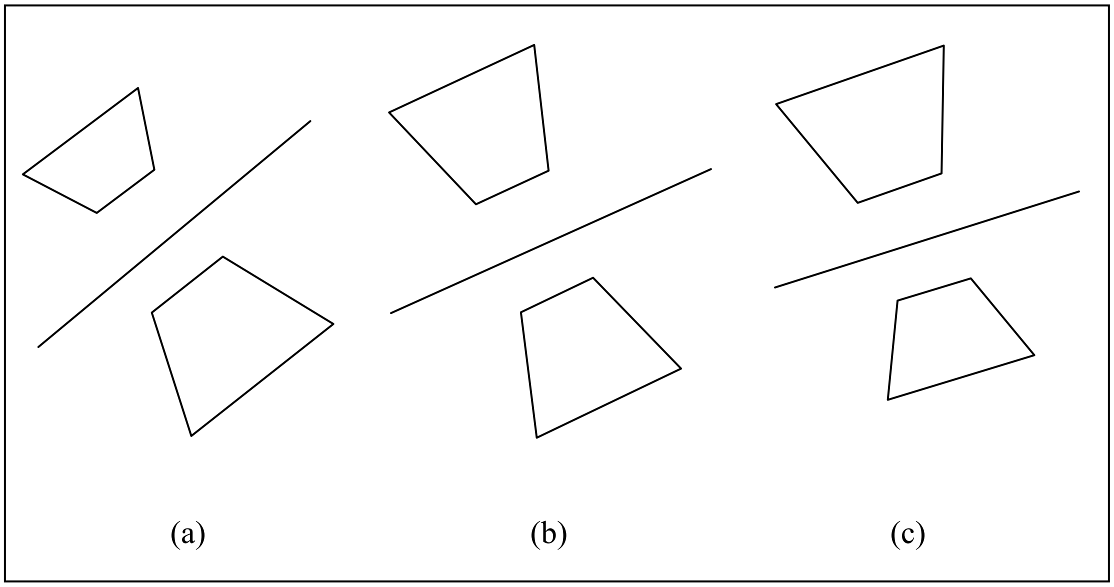

Figure 18. Inclination of the separating hyperplane

Thus, if the trend of data points goes upward as shown in Figure 18(a), the upper convex hull should be shrunk much more than the lower convex hull. If data points show the horizontal trend as shown in Figure 18(b), the reduced convex hull approach which is mentioned in the previous section is directly applied. If the trend of data points goes downward as shown in Figure 18(c), the lower convex hull should be reduced much more than the upper convex hull.

4.3 Incremental learning with Revised SVR with Filters (IRSVRF)

In the literature, few publications regarding the incremental SVR are reported so far [Liu and He, 2005; Ma, 2003; Wang, 2005]. Most of them concentrate on the updates of multipliers in SVR. These works are extensions of the on-line learning approach by Cauwenberghs and Poggio (2001). As mentioned before, these works are a variant of the incremental approach, and on-line updating of variables also requires dramatically increasing computing time if the support vectors are accumulated.

The Revised SVR concept with flexible upper bounds, mentioned in the previous section, is suitable for the incremental approach. The general format of the incremental revised SVR is similar to the incremental revised SVM. However, there are some differences between them.

First, the upper bounds of multipliers are flexible. If the pattern of recent data seems to be upward, the corresponding reduced convex hull should be shrunk more. This kind of pattern can be detected by the discarded data. Second, the running batch is possible. In this situation, old support vectors can be removed. Third, two filters are used on both sides. Thus, if a new data point is outside of the reduced supporting hyperplane, the data point will be removed from training.

Therefore, three approaches in the framework of the incremental learning process are proposed in this study.

The first is the incremental revised support vector regression (IRSVR). The revised convex hull concept is applied to the incremental learning procedure. Since the revised convex hull concept is used, the number of support vectors is dramatically decreased. Accordingly the computing time is also reduced. In a dynamic data driven application, such as the financial market, “time” is a critical factor in fluctuations of price, index, or foreign exchange rate. Even though an algorithm produces a good generalization rate, it is useless if it takes a huge amount of time. In this aspect, this IRSVR approach is very appropriate for a dynamic data driven application. The algorithmic procedure is shown as follows:

==

Step 1. Determine the optimal batch size based on generalization error rate and computing time for training.
Step 2. Determine the reducing rates $C1$ and $C2$
Step 3. Train data in the first batch by the revised SVR.
Step 4. Store data points representing support vectors in the next batch.
Step 5. Train the data in the batch by the revised SVR, and obtain support vectors and corresponding regression function.
Step 6. Store data points in the next batch, and go to step 5.

==

The diagram of the above algorithm is shown in Figure 19.

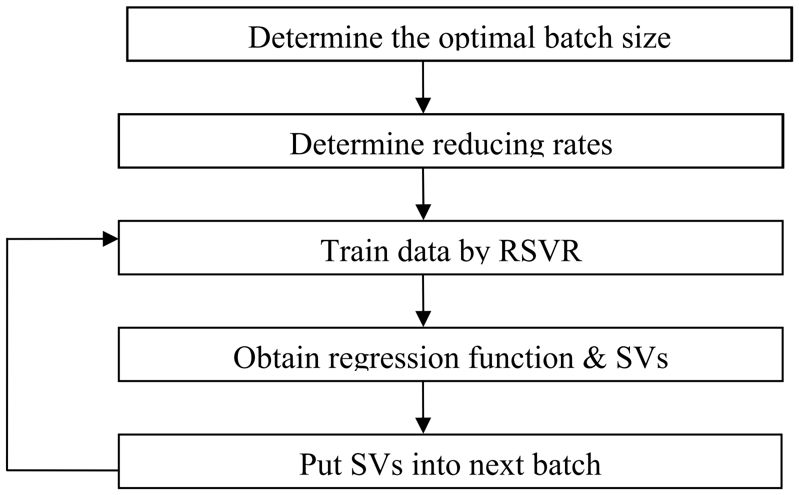

Figure 19. Diagram for IRSVR

The second approach is the incremental revised support vector regression with trends (IRSVRT). The regression problem is to find the tracing function closely fitting the existing historic data for a better prediction of the future value. As mentioned in section 4.2, when the regression function shows the uphill or downhill trend, appropriate adjustments should be added in the incremental learning procedure to reflect the tendency. In this aspect, the IRSVR might produce worse results. After the predicted value is obtained, the actual value is also shown later. This difference between the predicted value and the actual value can be a good indicator showing the current trends. If the difference is greater than zero, it means that the predicted value is higher than the actual value. Thus, the regression line will be upward if the upper convex hull is more reduced. On the other hand, the lower convex hull should be shrunk when the difference is less than zero. Then, the regression line goes down. This idea can be reflected in the IRSVR procedure. The algorithmic procedure is shown as follows:

```
============================================================
Step 1.  Determine the optimal batch size based on generalization error rate and
           computing time for training.
Step 2.  Determine the reducing rates $C1$ and $C2$ considering the trends
          (initial rates are determined by the user)
Step 3.  Train data in the batch by the revised SVR and obtain support vectors,
          and corresponding regression function.
Step 4.  Store data points representing support vectors in the next batch.
Step5.  Compare the predicted and actual values;
          If the difference is greater than 0, decrease the upper reducing rate
          Otherwise, decrease the lower reducing rate, and go to step 2
============================================================
```

The diagram of the above algorithm is shown in Figure 20.

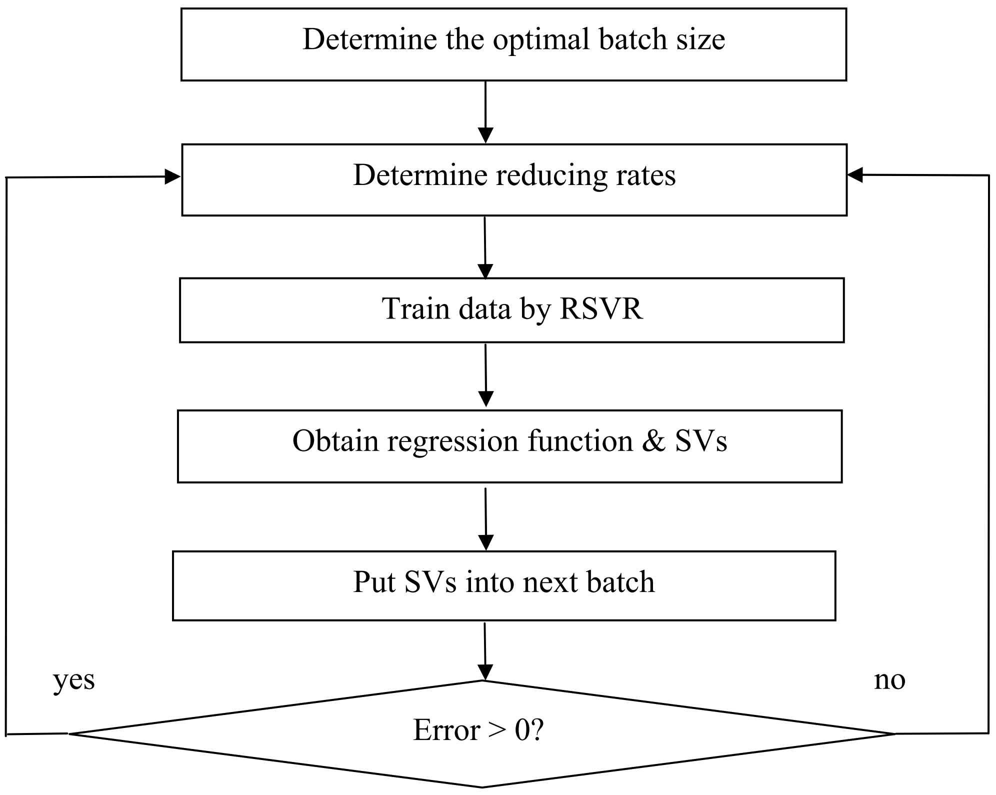

Figure 20. Diagram for IRSVRT

Next, the incremental revised support vector regression with trends using filters (IRSVRT) is described. The filter concept developed in section 3.4 is directly applied to the IRSVR procedure. Different from the filter in the RSVM, two filters are created to remove potential noisy data. The two filters are the reduced supporting hyperplanes as shown in Figure 21.

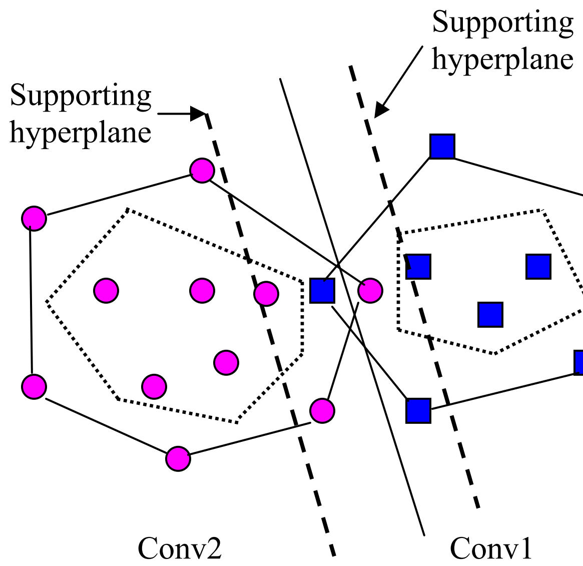

Figure 21. Two filters for IRSVRTF

In the situation with rapid fluctuations, this approach might work properly because if the newly coming data stands outside of the filters, the filters remove those data instantly. However, this approach will be effective in order to handle a large-scale data set in a long-term period. The algorithmic procedure is shown as follows:

==

```
Step 1.  Determine the optimal batch size based on generalization error rate and
         computing time for training.
Step 2.  Determine the reducing rates $C1$ and $C2$ considering the trends
         (initial values are determined by the user).
Step 3.  Train data in the batch by the revised SVR, and obtain support vectors
         and corresponding regression function.
Step 4.  Store data points representing support vectors in the next batch.
Step 5.  Inject a new point to the filters.
Step 6.  If a new data point is located outside of the filters,
             then (1) discard the data point, (2) identify which filter is used
```

and (3) count the discarded data

Otherwise, store it in the next batch.

Step 7. If the size of data in a batch is equal to the optimal batch size, go to the next step

Otherwise, go to step 5.

Step 8. Train the data in the batch and obtain support vectors, and the corresponding regression function.

Step 9. Update the filters and the reducing rates $C1$ and $C2$

considering the trends. Go to step 2.

==

The diagram of the above algorithm is shown in Figure 22.

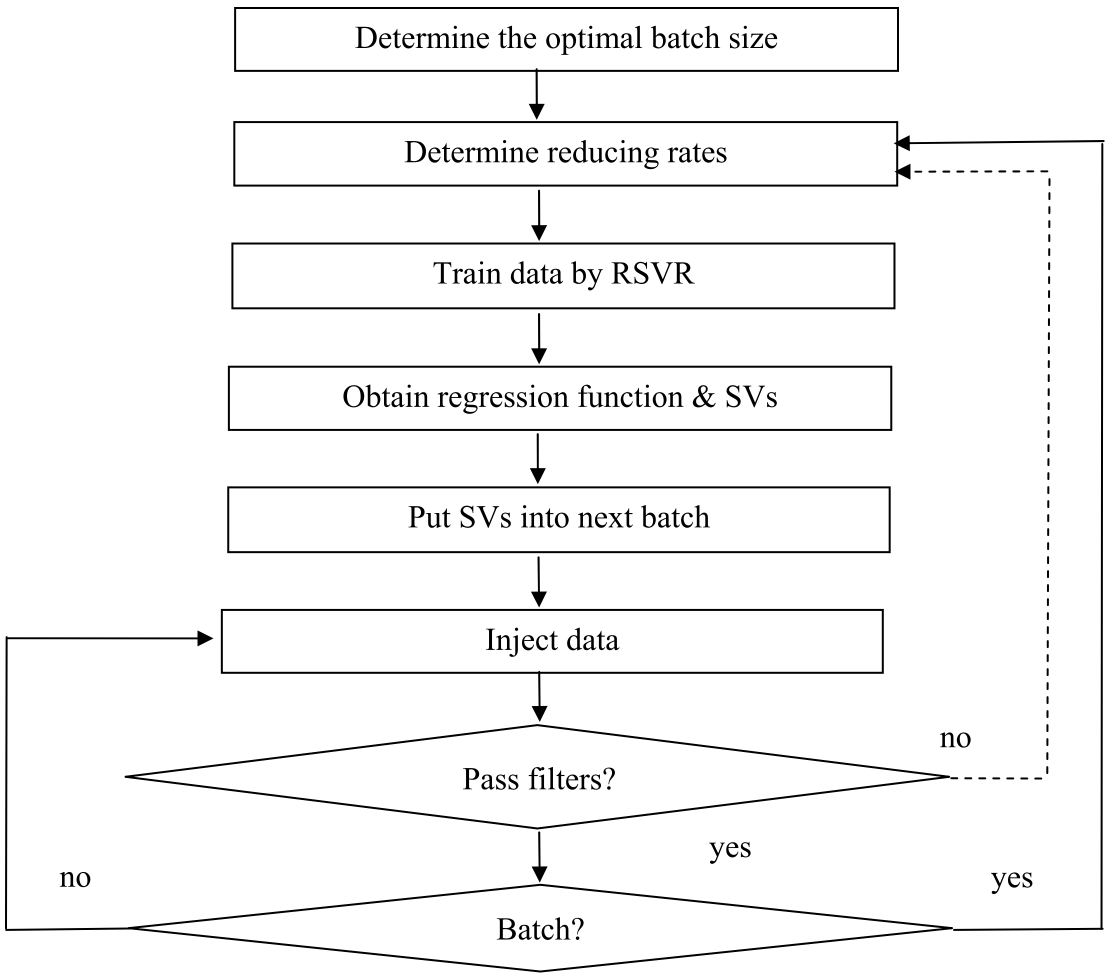

Figure 22. Diagram for IRSVRTF

4.4 A toy problem for IRSVR

For illustration purposes, a toy problem is created. Instead of a simple linear regression, the sine function is used to produce the artificial data. To reflect the real situation, a noisy term is added to the sine function. The formulation of the noisy sine function is shown in (4.28):

$$y = \sin(2\pi * x) + \rho \; randn(size(x)) \tag{4.28}$$

In (4.28), the second term creates some noise, which is produced by multiplication of the random number from the normal distribution and noisy coefficient ρ. The noisy coefficient ρ is arbitrarily set to 0.5. Figure 23 shows 100 data points that are randomly generated in the range from zero to one by the function in (4.28).

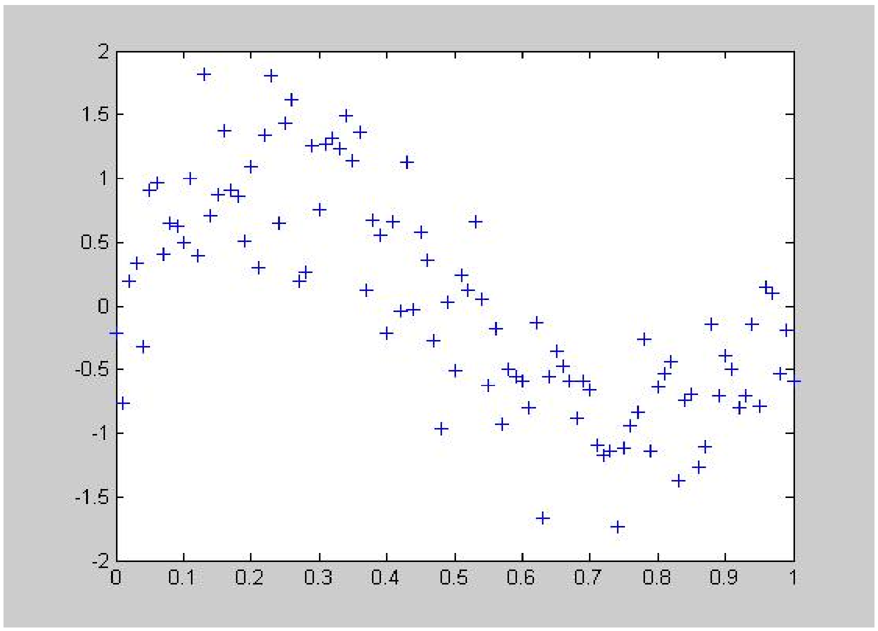

Figure 23. A data set generated from the noisy sine function

4.4.1 Description of experiments

The input data are the random numbers and their noisy sine function values. The 300 random numbers are generated between zero and one. The corresponding noisy sine function values are also obtained. Ten training and testing sets are used to produce the valid results with the input values selected from the interval [0, 1].

Due to the computational complexity, it is practically impossible to train many data at once. Thus, this toy example is designed to compare different approaches: The traditional SVR (SVR), the incremental SVR approach (ISVR), and the incremental revised SVR (IRSVR). As an extension of IRSVR, IRSVR with trends (IRSVRT) and IRSVR with trends using filters (IRSVRTF) is developed as well.

For comparison criteria, the number of support vectors, the computing time, the mean square error (MSE), the mean absolute error (MAE) and the number of batches are used. Here, MSE and MAE are averaged over 10 sets. Resulting values are compared under those comparison criteria.

Note that all approaches except the IRSVRTF use the same testing set. Since the IRSVRTF trains the available data removing the potential noisy data and considers the trends at each batch, the same testing set used in other approaches cannot be used. Thus, the testing set for the IRSVRTF is created in the following manner. After the first batch is trained, the IRSVRTF collects the appropriate data using the filters. When a data point passes the filter, the index of the data point is remembered, and the corresponding index in the testing set is used to form a testing set. Thus, it makes it possible to compare the prediction value and the actual value at the same time.

In this experiment, a batch size is arbitrarily determined at 50, and ε value is set to 0.2. The initial reducing rate is 0.1, and the diminished amount of the reducing rate when the trends are considered is predetermined at 10%. In the SVR, C is equal to 1. The above parameters should be tuned before the experiment for accurate results; all parameters are arbitrarily predetermined because this experiment has an illustration purpose.

4.4.2 Experimental results

All resulting values under the comparison criteria are the average value of results after 10 training and testing data sets are executed. The CPU time is the total training time. The MSE and MAE are computed based on the error, which is the difference between the predicted value and the actual value. The comparison of different approaches is shown in Table 3.

Table 3. Comparison of different approaches under comparison criteria

Method	CPU time (sec)	MSE	MAE	Number of SVs	Number of batches
SVR	222.96	0.7201	0.7049	253.2	1
ISVR	237.46	0.7161	0.7003	242	6
IRSVR	9.13	0.7223	0.7167	20.1	6
IRSVRT	9.21	0.7223	0.7167	20.1	6
IRSVRTF	5.98	0.209656	0.3823	22.7	4

From the results, IRSVRTF shows the best results in almost all criteria. Although the number of support vectors is a little higher than the IRSVR and IRSVRT, IRSVRTF reduces the number of batches as it removes the potential noisy data. The computing time of the traditional SVR is almost the same as the incremental SVR. It is because the incremental SVR accumulates the support vectors. The numbers of support vectors in the SVR and ISVR approaches are almost the same. In contrast, IRSVR and IRSVRT show the effectiveness with respect to the "time" with the similar MSE and MAE.

Clearly the results show that the revised convex hull concept reduces the number of support vectors. Accordingly, this revised SVR approach saves a huge amount of computing time proving the effectiveness of the revised SVR concept.

CHAPTER 5

APPLICATIONS AND COMPUTATIONAL RESULTS

In the real world, the SVMs and SVR methods have been applied to many areas such as image classification, bioinformatics, text-categorization, data mining, meteorology, and financial forecasting. It's hard to use the standard SVM and SVR due to many limitations (e.g., memory requirement, timely manner, availability of input data). Thus, the incremental learning approach developed in this study is quite suitable to overcome the above limitations. In this chapter, a tornado detection problem is considered as an application of IRSVM. In addition, a financial forecasting problem is considered as an application of IRSVR.

5.1 Tornado detection Problem

A tornado is a rarely occurring critical event in the real world as well as in the meteorology community. Based on the weather data produced from the Weather Surveillance Radar 1988 Doppler (WSR-88D), the Mesocyclone Detection Algorithm (MDA) is currently used to detect tornados.

The tornado detection problem is a good application to apply the new algorithm that was developed in this study. The tornado detection problem is characterized as follows: First, it is a two-class (tornado and non-tornado) classification problem. Second, it is an unbalanced problem. The tornado class consists of few data in the entire tornadic and non-tornadic data set. Third, it is an asymmetric importance problem. That is, the tornado class is relatively more important than the non-tornado class. Fourth, weather data related to tornados are periodically provided by weather radar.

The standard SVM or other variants cannot properly train the weather data due to the size of weather data, limited capacity of computation for training, and periodic inflow of weather data. The incremental learning procedure can overcome these restrictions

Typically, a tornado confusion matrix is utilized in order to measure the performance for tornado detection as shown in Table 4.

Table 4. Tornado confusion matrix

	Tornado Observed			
Forecast Tornado		Yes	No	
	Yes	Hit (a)	False Alarm (b)	“Yes” Forecasts
	No	Miss (c)	Correct (d)	“No” Forecasts
		“Yes” Observation	“No” Observation	Total number of observations

There are several meteorological indices in the literature. The most important measurement among them is the probability of detection (POD). Based on a tornado confusion matrix, the POD can be obtained as follows:

$$\text{Probability of detection (POD)} = \frac{a}{a+c}$$

Since a tornado without a warning may bring serious disasters, to increase the detection rate that can be measured by the POD, is a critical issue in the meteorology community. The POD formula consists of “hit” rate and “miss” rate. To increase the POD, “hit” rate should be increased and “miss” rate must be decreased. In this study, all tornado data are used whereas non-tornado data are partially used to train the data sets using the reduced convex hull concept. Thus, the ‘false alarm’ rate might be increased because the size of the convex hull containing non-tornado data is reduced. However, the number of “miss” events will be substantially reduced because the convex hull containing tornado data is not shrunk. Measurements of False Alarm Rate (FAR) and Critical Success Index (CSI) are also obtained for a full picture. Formula of FAR and CSI are shown as follows:

$$\text{False Alarm Rate (FAR)} = \frac{b}{a+b}$$

$$\text{Critical Success Index (CSI)} = \frac{a}{a+b+c}$$

5.1.1 Description of Experiments for Tornado Detection

In this study, we use the MDA data provided by the National Severe Storms Laboratory (NSSL). These tornadic and nontornadic data, generated from 1994 to 1999, are randomly selected to produce ten training and testing sets. Each datum has 23 attributes that are related to information such as velocity and shear. These attributes have been successfully used for tornado detection in the literature [Marzban and Stumpf, 1996]. The list of attributes is shown in Table 5. To reflect the real situation, 10% of the each training and testing set (1500 data) is tornadic data. We assume that randomly selected data form sequential time frames because the MDA data provide only dates.

Table 5. List of attributes

No.	Attributes	No.	Attributes
1	Base (m)	13	Lowe-level gate-to-gate velocity difference (m/s)
2	Depth (m)	14	Maximum gate-to-gate velocity difference (m/s)
3	Strength rank	15	Height of maximum gate-to-gate velocity difference (m)
4	Low-level diameter (m)	16	Core base (m)
5	Maximum diameter (m)	17	Core depth (m)
6	Height of maximum diameter (m)	18	Age (min)
7	Low-level rotational velocity (m/s)	19	Strength index (MSI) weighted by average density of integrated layer
8	Maximum rotational velocity (m/s)	20	Strength index (MSIr) “rank”
9	Height of maximum rotational velocity (m/s)	21	Relative depth (%)
10	Lowe-level shear (m/s/km)	22	Low-level convergence (m/s)
11	Maximum shear (m/s/km)	23	Mid-level convergence (m/s)
12	Height of maximum shear (m)		

Three approaches are performed and compared: the incremental approach with standard SVM (ISSVM), the incremental approach with revised SVM (IRSVM), and the incremental approach with revised SVM and filter (IRSVMF). These approaches are compared in terms of POD, total CPU time, number of batches, and “miss” portion over all other data. Each approach executes training and testing ten times, respectively, and their average values are computed in terms of the above criteria.

In the incremental approaches with standard SVM and with revised SVM, each batch size is set to 300 (for specific details, refer to [Son et al., 2005]). The incremental approach with filters uses a batch with a size of 300 data, which consists of data passing the filter. MATLAB codes originally provided by [Gunn, 1997] are entirely revised to run the incremental step. A Pentium IV 2.8GHz with 1 GB Ram of memory was used to perform all experiments.

5.1.2 Experimental Results

After performing each approach with ten training and testing data sets, the averaged results are shown in Table 6.

Table 6. Comparison of incremental learning methods

Methods	POD	Total CPU time (Sec)	Number of SVs	“Miss”/all (%)
Incremental approach with standard SVM	0.62	754.62	57	3.83
Incremental approach with revised SVM	0.69	406.26	11	3.14
Incremental approach with revised SVM & filter	0.60	314.46	11	3.97

Even if the total CPU time and the number of support vectors are reduced, IRSVM outperforms ISSVM in terms of POD and “miss” over all rate. The CPU time and the number of support vectors for each batch are shown in Figures 24 and 25, respectively.

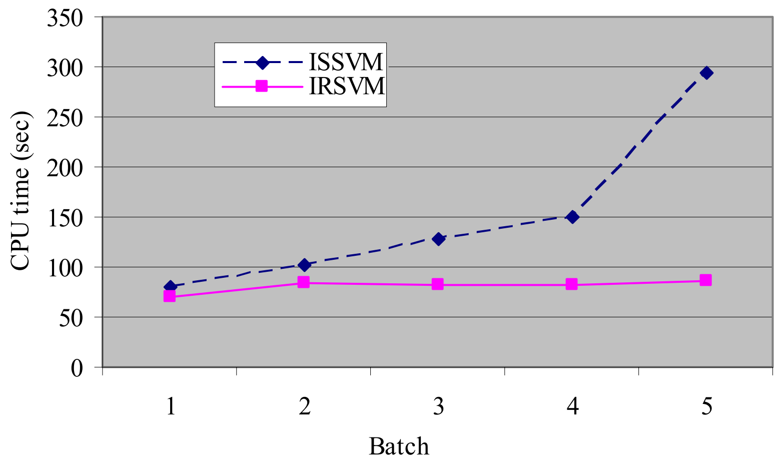

Figure 24. Comparison between ISVM and IRSVM with respect to computing time

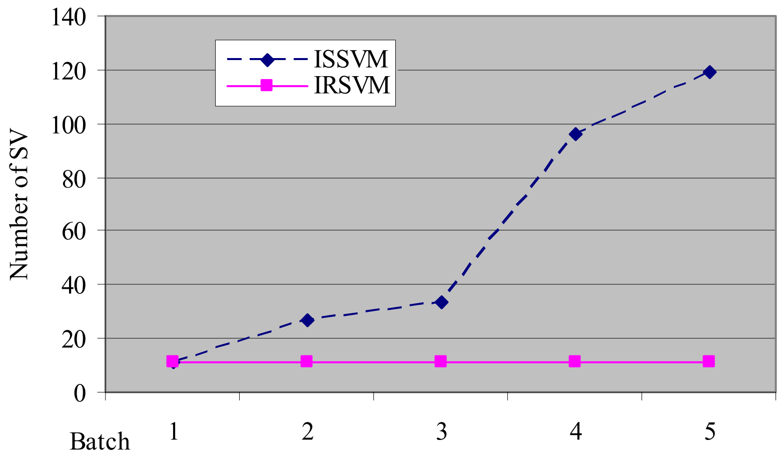

Figure 25. Comparison between ISVM and IRSVM with respect to the number of support vectors

Since the number of support vectors is increased as each batch is sequentially trained, the computing time is also dramatically increased, whereas IRSVM keeps the same number of support vectors and requires a smaller computing time. When IRSVMF is applied, computing time is also significantly reduced although POD is slightly dropped.

Table 7. Comparison of methods with respect to FAR and CSI

Methods	FAR	CSI
Incremental approach with standard SVM	0.81	0.17
Incremental approach with revised SVM	0.90	0.10
Incremental approach with revised SVM & filter	0.90	0.08

Though filter discards potential non-tornado data, IRSVM and IRSVMF have similar FAR rates. CSI slightly drops when revised SVM concept is applied to the incremental learning procedure.

5.2 Financial Forecasting Problem

It is very critical to predict the accurate future value of an index in the financial market. There exist various indices such as interest rate, foreign exchange rate, Dow Jones Index, etc. Since these indices are determined by a huge number of factors, it is hard to identify and to model in a simple form entire factors in which hard-to-quantify factors (e.g., personal intuition) might exist. In addition, these indices are dynamically fluctuating all the time. Thus, it is a good example of a dynamic data-driven application. Since factors that affect the indices cannot be easily identified and data for these kinds of factors are not available currently, only time is considered as the input in this experiment. The output value will be the expected S&P 500 index. Thus, the aim of this experiment is to compare different incremental approaches using various criteria.

5.2.1 Description of experiments for financial forecasting

For input data the S&P 500 data are obtained from the Yahoo's finance web site. Daily close indices from 1950 to the present are available. It is practically impossible to train all data simultaneously due to the computational complexity. Thus, only incremental approaches will be applied and compared.

Daily close indices are partitioned into 5 sets (5 decades). Training and testing data sets are formed in each decade. The size of available data in 5 decades is shown in Table 8.

Table 8. Size of available data in 5 decades

Decade	Size of data
1951-1960	2411
1961-1970	2492
1971-1980	2526
1981-1990	2527
1991-2000	2528

From each decade, a 400 training data set and a 400 testing data set are made. In this experiment, the traditional SVR (SVR), the incremental SVR (ISVR), the incremental revised SVR (IRSVR), and the incremental revised SVR with trends (ISVRT) are compared. The incremental revised SVR with trends using filters (IRSVRTF) is not used because the preliminary experiments show that it runs only one time and stops. I believe it is because the reducing rates are appropriately tuned, and the size of data set considered is quite small. Basically the aim of this experiment is to show the difference of the critical factor, “time”, between different approaches. As comparison criteria, the number of support vectors, the computing time, the mean square error (MSE), and the mean absolute error (MAE) are used. Other conditions, such as the reducing rates and ε, are the same as conditions in the toy problem.

5.2.2 Experimental results

The average values of results after 5 training and testing data sets are shown in Table 9.

Table 9. Comparison of different approaches under the comparison criteria

Method	CPU time (sec)	MSE	MAE	Number of SVs
SVR	1735.52	1994.44	21.3	382.80
ISVR	2195.98	1718.39	19.1	344.8
IRSVR	33.52	29609.78	128.11	21
IRSVRT	33.57	29603.78	128.11	21

From the results, SVR and ISVR require a huge amount of computing time. Since the support vectors are accumulated, it requires more time to train. In addition, the revised SVR concept clearly reduces the number of support vectors. To decrease the MSE and MAE of the incremental approach with the revised SVR concept, the related parameters (e.g., reduce rates, epsilon, and kernel) must be tuned properly. Though MSE and MAE of the incremental approach with the revised SVR concept are quite high, there is no statistically significant difference between the methods in Table 9. The corresponding single ANOVA tables with respect to MSE and MAE are shown in Tables 10 and 11, respectively. Therefore, IRSVR and IRSVRT can reduce the computing time performing similarly like in the case of SVR and ISVR.

Table 10. ANOVA Table for MSE

Source of variation	SS	df	MS	F	P-value	F_crit
Between Groups	3851438895	3	1283812965	0.965	0.433	3.239
Within Groups	21284683876	16	1330292742			
Total	25136122772	19				

Table 11. ANOVA Table for MAE

Source of variation	SS	df	MS	F	P-value	F_crit
Between Groups	57075	3	19025	2.219	0.128	3.239
Within Groups	138603	16	8663			
Total	195678	19				

CHAPTER 6

SUMMARY AND FUTURE RESEARCH

Modern technology produces a massive amount of data nowadays. However, it is not simple to extract the necessary information from massive data. Furthermore, the span of data is getting shorter so that the existing static systems tend to be transformed to the dynamic data-driven application systems. To deal with the dynamic data, the current learning procedure should be reconsidered and revised properly. In this dissertation, several approaches are proposed to overcome the limitations of the existing kernel learning approaches for SVMs. First, the revised SVM is proposed to improve the detection rate of data in the important class in a binary classification problem. It reduces the number of support vectors as well as the computing time for training the data. Second, the revised SVM concept is applied to the incremental learning approach for dealing with large amount of data. A filter concept is also proposed to remove the potential noisy data making the speedy training process possible. Third, for the regression problem, the revised SVM concept is applied and revised to fit the slightly different situations. In addition, an incremental revised SVR approach considering the trends is also proposed. This filter concept can be adapted to other learning algorithms to improve the performance with respect to computing time. In the application chapter, the revised SVM concept clearly improves the detection rate of tornados, and the filter concept makes the training more effective. Furthermore, several parameters (e.g., the optimal batch size for the revised SVM, the upper bound of multipliers, μ, for the unimportant class) need to be tuned. These parameters definitely affect the performance of the proposed algorithm in terms of the number of support vectors, computing time, generalization error, and detection rate for the important class. More effective selection of those parameters should be investigated (e.g., improved tornado detection rate).

The proposed algorithms are applied to real data in two applications, such as the tornado detection problem and the financial forecasting problem. From the results, the revised convex hull concept definitely works better especially with respect to computational time. Hence, this revised convex hull concept should be seriously considered in the dynamic data driven application systems

In future research, the revised convex hull concept can be extended to the multi-class SVM problem. For the revised convex hull concept to be successfully performed in various applications, the appropriate reducing rates should be tuned. Since those rates depend on the structure of data sets, the investigation for the appropriate reducing rates will be a future research topic. For the regression problem, when the factors that affect the financial indices are identified, the incremental revised SVR can be performed and investigated considering those factors. Future research will investigate those issues.

BIBLIOGRAPHY

Bazaraa, M. Z., J. J. Jarvis, and H. D. Sherali, 1990. *Linear Programming and Network Flows*. 2nd edit., New York, NY: John Wiley & Sons, Inc.

Bennett, K. P, and E. J. Bredensteiner, 2000. "Duality and Geometry in SVM Classifiers," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 57-64.

Bern, Marshall, and D. Eppstein, 2001. "Optimization Over Zonotopes and Training Support Vector Machines," *Proceedings of 7th Workshop, Algorithms and Data Structures (WADS 2001)*, Lecture Notes in Computer Science 2125, Frank K. H. A. Dehne, Jorg-Rudiger Sack, and Roberto Tamassia, ed., Springer-Verlag, pp. 111-121.

Bi, J. and K. P. Bennett, "Duality, geometry, and support vector regression," *Advances in Neural Information Processing Systems 14*, NIPS 2001, pp. 593-600, MIT Press.

Brown, M. P., W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, 1999, "Support vector machine classification of microarray gene expression data," *UCSC-CRL 99-09*, Department of computer science, University of California at Santa Cruz.

Burges, C. J. C., "A tutorial on support vector machines for pattern recognition," *Data mining and Knowledge Discovery*, vol. 2, pp. 121-167.

Cao, L. J. and F. E. H. Tay, 2003, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1506-1518.

Cauwenberghs, G. and T. Poggio, 2001. "Incremental and Decremental Support Vector Machine Learning," *Advances in Neural Information Processing Systems*, vol. 13, pp. 409-415. Cambridge, MA: MIT Press.

Crisp, Daivd J. and C. J. C. Burges, 1999. *A Geometric Interpretation of ν-SVM Classifiers.* In *Advances in Neural Information Processing Systems (NIPS)* vol. 12. Cambridge, MA: MIT Press.

Cristianini, N. and J. Shawe-Taylor, 2000. *An Introduction to support vector machines*. Cambridge University Press, Cambridge, UK.

Demeniconi, C. and D. Gunopulos, 2001. "Incremental support vector machine construction," *Proceedings of the IEEE International Conference on Data Mining,* San Jose, CA.

Gunn, S. R., 1997. *Support Vector Machines for Classification and Regression. Technical Report,* Image Speech and Intelligent Systems Research Group, University of Southhampton.

Hashemi, S. and T. P Trappenberg, 2002. "Using SVM for Classification in Dataset with Ambiguous Data," *International Conference on Information Systems, Analysis and Synthesis SCI.*

Haykin, S., 1998. *Neural Networks: A Comprehensive Foundation,* 2nd edit., Upper Saddle River, NJ: Prentice-Hall.

Joachims, T, "Text categorization with support vector machines: learning with many relevant features," *Proceedings of ECML-98, 10th European Conference on Machine Learning*, no. 1398, pp. 137-142.

Klinkenberg, R. and T. Joachims, 2000. "Detecting concept drift with support vector machines," *Proceedings of ICML-00, 17th International Conference Machine Learning*, pp. 487-494.

Lee, Y. J. and O. L. Mangasarian, 2001. "RSVM: Reduced support vector machines," *Proceedings of the First SIAM International Conference on Data Mining*, 2001.

Liu, Y. and Q. He, 2005, "Concept updating with support vector machines," WAIM, LNCS 3739, pp. 492-501, Springer-Verlag, Berlin Heidelberg.

Ma, J., J. Theiler, S. Perkins, 2003. "Accurate on-line support vector regression," *Neural Computation*, vol. 15, pp 2683-2703.

Marzban, C. and G. J. Stumpf, "A Neural Network for Tornado Prediction Based on Doppler Radar Derived Attributes," *Journal of Applied Meteorology*, 1996, vol. 35, pp. 617-626.

Osuna, E, R. Freund, and F. Girosi, 1997. "An Improved Training Algorithm for Support Vector Machines," Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing," Eds. J. Principe, L. Giles, N. Morgan, E. Wilson, pp. 276-285, Amelia Island, FL.

Osuna, Edgar and F. Girosi, 1998. *Reducing the run-time complexity of Support Vector Machines*. ICPR, Brisbane, Australia.

Ripley, B. D., 1996. *Pattern Recognition and Neural Network,* Cambridge University Press.

Ripley, B. D., 1994. "Neural networks and related methods for classification," *Journal of the Royal Statistical Society B*, vol. 56, no. 3, pp. 409-456. Available from the Internet: www.stats.ox.ac.uk/pub/PRNN/

Ruping, S. 2001. "Incremental learning with Support Vector Machines," *Proceedings of the IEEE international conference on data mining*, San Jose, CA.

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

Shilton, A., M. Palaniswami, D. Ralph, and A. C. Tsoi, 2005. "Incremental training of support vector machines," *IEEE Transactions on neural networks*, vol. 16, no. 1, pp 114-131.

Son, H, T. B. Trafalis, and M. Richman, 2005. "Determination of the Optimal Batch Size in Incremental Approaches: An Application to Tornado Detection," *Proceedings of the International Joint Conference of Neural Network*, Montreal, Canada. (accepted).

Song, M., C. M. Breneman, J. Bi, N. Sukumar, K. P. Bennett, S. Cramer, N. J. Tugcu, 2002. "Prediction of protein retention times in anion-exchange chromatography systems using support vector regression," *Journal of Chemical Information and Computer Science*, vol. 42, no. 6, pp. 1347-1357.

Syed, N. A., H. Liu, and K. K. Sung, 1999. "Incremental Learning with Support Vector Machines," *Workshop on Support Vector Machines, International Joint Conference on Artificial Intelligence,* Stockholm, Sweden.

Syed, N. A., H. Liu, and K. K. Sung, 1999. "Handling concept drifts in incremental learning with support vector machines," *Proceedings of first international conference on knowledge discovery and data mining,* pp. 317-321, San Diego.

Trafalis, T. B. and H. Ince, 2000. "Support vector machine for regression and applications to financial forecasting," *Proceedings of the IEEE-INNS-ENNS International Joint Conference (IJCNN 2000)*, vol. 6, pp 348-353.

Trafalis, T. B., B. Santosa, and M. Richman, 2004. "Prediction of rainfall from WSR-88D Radar using kernel-based methods," *International Journal of Smart Engineering System Design*, vol. 4, no. 4, pp. 429-438.

Vapnik, V. N., 1995. *The Nature of Statistical Learning Theory*. New York, NY: Springer Verlag.

Vishwanathan, S. V. N., A. J. Smola, and M. N. Murty, 2003. "Simple SVM," *Proceedings of the Twentieth International Conference on Machine Learning (ICML – 2003)*, Washington DC.

Wang, W., 2005. “An incremental learning strategy for support vector regression,” *Neural Processing Letters*, vol. 21, no. 3, pp. 175-188.

Yang, H. Y., L. Cahn, and I. King, 2002, “Support vector regression for volatile stock market prediction,” *LNCS* 2412, pp. 391-396, Springer-Verlag, Berlin Heidelberg.

Yang, H., K. Huang, L. Chan, I. King, M. R. Lyu, 2004. “Outlier treatment in support vector machine for financial time series prediction,” *ICONIP 2004, LNCS 3316*, pp. 1260-1265, Springer-Verlag, Berlin Heidelberg.

Yahoo website: http://finance.yahoo.com.

APPENDICES

In these appendices, MATLAB codes (version 6.5), which are used in experiments, are presented.

IRSVM

```
==============================================================

clear all
global P1   % P1 is degree of polynomial
global P2   % P2 is width of rbfs (sigma)
global KERTYPE %1: linear; 2: polynomial; 3: rbf
global C1   % C value on the positive side
global C2   % C value on the negative side

%inputs
P1= 2;
P2= 1;
KERTYPE = 2;
C1= 1;
C2= 0.1;
batch=300; % size of batch
method = 4; % 1: traditional SVM; 2: incremental SVM; 3: revised SVM; 4: revised
SVM with filter
p=3;        %staring column
q=25;        %ending column

% declarations
Tinfo=[];
TTinfo_svm=[];
TTinfo_incre=[];
TTinfo_rev=[];
TTinfo_revfilter=[];
X=[];
Y=[];
svX=[];
svY=[];
batchcount=0;
trnX=[];
trnY=[];
```

```
tcand=[];

%load data
load tr2_TandNT.txt
load ts2_TandNT.txt

[row col]=size(tr2_TandNT);
[row2 col2]=size(ts2_TandNT);

rawtrn=tr2_TandNT(1:row,:);
rawtst=ts2_TandNT(1:row2,:);

%measure the cputime
t0=cputime;

switch lower(method)
   case 1 % traditional SVM

   trnX=rawtrn(:,p:q);
   trnY=rawtrn(:,q+1);

   [nsv, alpha, sv, b0]=mysvc1(trnX,trnY);

   Tcpu=cputime-t0;

   svX=sv(:,1:23);
   svY=sv(:,24);
   [svrow, svcol]=size(sv);

   tstX=rawtst(:,p:q);
   tstY=rawtst(:,q+1);
   bias=b0;

   [Terror,a,b,c,d] = mysvcerror1(trnX,trnY,tstX,tstY,alpha,bias);

   batchcount=batchcount+1;

   Tinfo=[svrow, Tcpu, Terror, a, b, d, d, batchcount];
   TTinfo_svm=[TTinfo_svm; Tinfo];

   save  TTinfo_svm.txt TTinfo_svm -ASCII

   case 2 % incremental SVM

   i=1;
   j=i+batch-1;
```

```
while i > 0
trnX=[rawtrn(i:j,p:q); svX];
trnY=[rawtrn(i:j,q+1); svY];

[nsv, possize, negsize, alpha, sv, b0]=mysvc2(trnX,trnY);

batchcount=batchcount+1;
Tcpu=cputime-t0;

svX=sv(:,1:23);
svY=sv(:,24);
[svrow, svcol]=size(sv);

tstX=rawtst(i:j,p:q);
tstY=rawtst(i:j,q+1);
bias=b0;

[Terror, a,b,c,d] = mysvcerror1(trnX,trnY,tstX,tstY,alpha,bias);

Tinfo=[svrow, Tcpu, Terror, a,b,c,d, possize, negsize, batchcount];
TTinfo_incre=[TTinfo_incre; Tinfo];

%update i and j values
i=j+1;
%j=300-svrow;
j=i+batch-1;

if j > row2
   break
end
end %while end

save  TTinfo_incre.txt TTinfo_incre -ASCII

case 3 % incremental revised SVM
i=1;
j=i+batch-1;

while i > 0
trnX=[rawtrn(i:j,p:q); svX];
trnY=[rawtrn(i:j,q+1); svY];

[nsv, possize, negsize, trnXX, trnYY, alpha, sv, b0]=mysvc3(trnX,trnY);

batchcount=batchcount+1;
```

```
Tcpu=cputime-t0;

svX=sv(:,1:23);
svY=sv(:,24);
[svrow, svcol]=size(sv);

tstX=rawtst(i:j,p:q);
tstY=rawtst(i:j,q+1);
bias=b0;

[Terror, a,b,c,d] = mysvcerror1(trnXX,trnYY,tstX,tstY,alpha,bias);

Tinfo=[svrow, Tcpu, Terror, a,b,c,d, possize, negsize, batchcount];
TTinfo_rev=[TTinfo_rev; Tinfo];

%update i and j values
i=j+1;
j=i+batch-1;

if j > row2
   break
end
end %while end

save  TTinfo_rev.txt TTinfo_rev -ASCII

case 4 % revised SVM with filter ============

i=1;
j=i+batch-1;
m=1;
n=m+batch-1;

trnX=[rawtrn(i:j,p:q)];
trnY=[rawtrn(i:j,q+1)];

[nsv, possize, negsize, trnXX, trnYY, alpha, svalpha, sv, b0, H]=mysvc4(trnX,trnY);

batchcount=batchcount+1;
Tcpu=cputime-t0;

svrow=nsv;

tstX=rawtst(m:n,p:q);
tstY=rawtst(m:n,q+1);
bias=b0;
```

```
[Terror,a,b,c,d] = mysvcerror1(trnXX,trnYY,tstX,tstY,alpha,bias);

Tinfo=[svrow, Tcpu, Terror,a,b,c,d, possize, negsize, batchcount];
TTinfo_revfilter=[TTinfo_revfilter; Tinfo];

i=j+1;
m=n+1;

while i > 0

while i > 0  %to form the batch using a filter
newX=rawtrn(i,p:q+1);
[cand]=filter8(sv, svalpha,  newX, b0);
tcand=[tcand; cand];
[ptrow, ptcol]=size(tcand);

if ptrow==300 | i == row
   break
end
i=i+1
end % second while

svX=sv(:,1:23);
svY=sv(:,24);
trnX=[tcand(:,1:23); svX];
trnY=[tcand(:,24); svY];
k=i;

[nsv, possize, negsize, trnXX, trnYY, alpha, svalpha, sv, b0, H]=mysvc4(trnX,trnY);

batchcount=batchcount+1;
Tcpu=cputime-t0;

tstX=rawtst(m:n,p:q);
tstY=rawtst(m:n,q+1);
bias=b0;

[Terror,a,b,c,d] = mysvcerror1(trnXX,trnYY,tstX,tstY,alpha,bias);

Tinfo=[svrow, Tcpu, Terror,a,b,c,d, possize, negsize, batchcount];
TTinfo_revfilter=[TTinfo_revfilter; Tinfo];

if i == row
   break
end
```

```
   i=i+1;
   m=n+1;
   n=m+batch-1;
   tcand=[];

   end %while end
   save  TTinfo_revfilter.txt TTinfo_revfilter -ASCII

   otherwise %traditional SVM
      error('specify the method that you want to use')
end
============================================================
```

Kernel

```
============================================================
function k = mykernel(u,v)

global P1 P2 KERTYPE

if KERTYPE == 1                               % linear
   k = u*v';
elseif KERTYPE == 2                           % polynomial
   k = (u*v' + 1).^P1;
elseif KERTYPE == 3                           % rbf
   k = exp(-(u-v)*(u-v)'/(2*P2^2));
else
   k = u*v';
end
=============================================================
```

SVC1

```
=============================================================
function [nsv, alpha,sv,b0] = mysvc1(X,Y)

global KERTYPE C1

if (nargin <2 | nargin>3) % check correct number of arguments
   help svc
  else
   fprintf('Support Vector Classification\n');
   fprintf('______________________________\n');

   n = size(X,1);
   [n n1] = size(X);
  % tolerance for Support Vector Detection
  epsilon = mytol(C1);

  % Construct the Kernel matrix
  H = zeros(n,n);
  for i=1:n
    for j=1:n
      H(i,j) = Y(i)*Y(j)*mykernel(X(i,:),X(j,:));
    end
  end
  c = -ones(n,1);

fprintf('Support Vector Kernel  \n');
fprintf('______________________________\n');
  H = H+1e-10*eye(size(H));

  % Set up the parameters for the Optimisation problem
  vlb = zeros(n,1);      % Set the bounds: alphas >= 0
  vub = C1*ones(n,1);    %               alphas <= C
  x0 = zeros(n,1);       % The starting point is [0 0 0   0]

  Ae = Y'; be = 0;     % Set the constraint Ax = b

  % Solve the Optimisation Problem
  fprintf('Optimising ...\n');
  st = cputime;

  %options setting
  options=optimset('MaxIter', 1000);

  [alpha] = quadprog(H, c, [], [], Ae, be, vlb, vub, x0, options);

  fprintf('Execution time: %4.1f seconds\n',cputime - st);
  w2 = alpha'*H*alpha;
  fprintf('|w0|^2    : %f\n',w2);
  fprintf('Margin    : %f\n',2/sqrt(w2));
  fprintf('Sum alpha : %f\n',sum(alpha));

  % Compute the number of Support Vectors
  svi = find( alpha > epsilon);
  nsv = length(svi);
  fprintf('Support Vectors : %d (%3.1f%%)\n',nsv,100*nsv/n);

  % Obtain support vectors
   sv=[X(svi,:), Y(svi,:)];
```

```
    % Implicit bias, b0
   b0 = 0;

   % Explicit bias, b0
   if nobias(KERTYPE) ~= 0
     % find b0 from average of support vectors on margin
     % SVs on margin have alphas: 0 < alpha < C1
     svii = find( alpha > epsilon & alpha < (C1 - epsilon));
     if length(svii) > 0
       b0 =  (1/length(svii))*sum(Y(svii) - H(svii,svi)*alpha(svi).*Y(svii));
     else
       fprintf('No support vectors on margin - cannot compute bias.\n');
     end
  end
  end
=================================================================
```

SVC3

```
=================================================================
function [nsv, possize, negsize, trnXX, trnYY, alpha, sv, b0] = mysvc3(X, Y)

global KERTYPE C1 C2

origdata =[];
posdata=[];
negdata=[];
trnXX=[];
trnYY=[];
vub=[];
be=[];
H=[];
Ae=[];
c=[];

if (nargin <2 | nargin>4) % check correct number of arguments
   help svc
  else
   origdata=[X, Y];
   [n n1] = size(X);
   posind=find(origdata(:,n1+1)>0);
   posdata=[origdata(posind,:)];
   [posrow poscol]=size(posdata);
   posX=posdata(:,1:(poscol-1));
   negind=find(origdata(:,n1+1)< 0);
  negdata=[origdata(negind,:)];
  [negrow negcol]=size(negdata);
  negX=negdata(:,1:(negcol-1));
  trnXX=[posX; negX];
  XX=[posdata; negdata];
  trnYY=[posdata(:, poscol); negdata(:,negcol)];

  % tolerance for Support Vector Detection
  epsilon = mytol(C2);

  % Construct the Kernel matrix for positive side
  if 'rbf'
     posH = ones(posrow, posrow);
  else
     posH = zeros(posrow,posrow);
     for i=1:posrow
        for j=1:posrow
        posH(i,j) = mykernel(posX(i,:),posX(j,:));
        end
     end
  end

  % Construct the Kernel matrix for negative side
  if 'rbf'
     negH = ones(negrow, negrow);
  else
     negH = zeros(negrow,negrow);
     for i=1:negrow
        for j=1:negrow
        negH(i,j) = mykernel(negX(i,:),negX(j,:));
        end
     end
  end

  % Construct the Kernel matrix for both sides
  bothH = zeros(negrow,posrow);
  for i=1:negrow
    for j=1:posrow
      bothH(i,j) = (-1)*mykernel(negX(i,:),posX(j,:));
    end
  end

  H=[posH bothH';bothH negH];
   f = zeros(n,1);

fprintf('Support Vector Kernel  \n')
fprintf('_____________________________\n')
  H = H+1e-10*eye(size(H));

  % Set up the parameters for the Optimisation problem
  vlb = zeros(n,1);      % Set the bounds: alphas, beta >= 0
  vub1 = C1*ones(posrow,1);     %                 alphas <= 1
  vub2 = C2*ones(negrow,1);     %               betas <= C
  vub = [vub1; vub2];
  x0 = zeros(n,1);       % The starting point is [0 0 0   0]
    posA=ones(1,posrow);
    negA=ones(1,negrow);
    posAzero=[posA, zeros(1,(n-posrow))];
    negAzero=[zeros(1,(n-negrow)), negA];
    Ae=[posAzero; negAzero]; be =[1; 1];

  % Solve the Optimisation Problem
  fprintf('Optimising ...\n');
  st = cputime;

  %options setting
  options=optimset('MaxIter', 1000);

  [alpha] = quadprog(H, f, [], [], Ae, be, vlb, vub, x0, options);

  fprintf('Execution time: %4.1f seconds\n',cputime - st);
  w2 = alpha'*H*alpha;
  fprintf('|w0|^2    : %f\n',w2);
  fprintf('Margin    : %f\n',2/sqrt(w2));
  fprintf('Sum alpha : %f\n',sum(alpha));

  % Compute the number of Support Vectors
  svi = find( alpha > epsilon);
  nsv = length(svi);
  fprintf('Support Vectors : %d (%3.1f%%)\n',nsv,100*nsv/n);

  % Obtain support vectors
   sv=XX(svi,:);
  possvind = find(sv(:,poscol)>0);
  possize=length(possvind);
  negsvind = find(sv(:,negcol)<0);
  negsize=length(negsvind);
  possvs=sv(possvind,:);
  negsvs=sv(negsvind,:);

  % Implicit bias, b0
  b0 = 0;
  % Explicit bias, b0
  if nobias(KERTYPE) ~= 0
    % find b0 from average of support vectors on margin
    % SVs on margin have alphas: 0 < alpha < C
    svii = find( alpha > epsilon & alpha < (C2 - epsilon));
    if length(svii) > 0
      b0 =  (1/length(svii))*sum(Y(svii) - H(svii,svi)*alpha(svi).*Y(svii));
    else
      fprintf('No support vectors on margin - cannot compute bias.\n');
    end
  end
end
=================================================================


*** SVC4 ***
=================================================================
function [nsv, possize, negsize, trnXX, trnYY, alpha, svalpha, sv, b0, H] = mysvc4(X, Y)

global KERTYPE C1 C2

origdata =[];
posdata=[];
negdata=[];
trnXX=[];
trnYY=[];
vub=[];
be=[];
H=[];
Ae=[];
c=[];

if (nargin <2 | nargin>4) % check correct number of arguments
  help svc
 else
  origdata=[X, Y];
  [n n1] = size(X);
  posind=find(origdata(:,n1+1)>0);
  posdata=[origdata(posind,:)];
  [posrow poscol]=size(posdata);
  posX=posdata(:,1:(poscol-1));
  negind=find(origdata(:,n1+1)< 0);
  negdata=[origdata(negind,:)];
  [negrow negcol]=size(negdata);
  negX=negdata(:,1:(negcol-1));
  trnXX=[posX; negX];
  XX=[posdata; negdata];
  trnYY=[posdata(:, poscol); negdata(:,negcol)];

  % tolerance for Support Vector Detection
  epsilon = mytol(C2);

  % Construct the Kernel matrix for positive side
  if 'rbf'
     posH = ones(posrow, posrow);
  else
     posH = zeros(posrow,posrow);
     for i=1:posrow
        for j=1:posrow
        posH(i,j) = mykernel(posX(i,:),posX(j,:));
        end
     end
  end

  % Construct the Kernel matrix for negative side
  if 'rbf'
     negH = ones(negrow, negrow);
  else
     negH = zeros(negrow,negrow);
     for i=1:negrow
        for j=1:negrow
        negH(i,j) = mykernel(negX(i,:),negX(j,:));
        end
     end
  end

  % Construct the Kernel matrix for both sides
  bothH = zeros(negrow,posrow);
  for i=1:negrow
    for j=1:posrow
      bothH(i,j) = (-1)*mykernel(negX(i,:),posX(j,:));
    end
  end

  H=[posH bothH';bothH negH];
   f = zeros(n,1);

fprintf('Support Vector Kernel  \n');
fprintf('______________________________\n');
  H = H+1e-10*eye(size(H));
```

```
% Set up the parameters for the Optimisation problem
vlb = zeros(n,1);      % Set the bounds: alphas, beta >= 0
vub1 = C1*ones(posrow,1);     %                  alphas <= 1
vub2 = C2*ones(negrow,1);     %                betas <= C
vub = [vub1; vub2];
x0 = zeros(n,1);       % The starting point is [0 0 0   0]
  posA=ones(1,posrow);
  negA=ones(1,negrow);
  posAzero=[posA, zeros(1,(n-posrow))];
  negAzero=[zeros(1,(n-negrow)), negA];
  Ae=[posAzero; negAzero]; be =[1; 1];

% Solve the Optimisation Problem
fprintf('Optimising ...\n');
st = cputime;

%options setting
options=optimset('MaxIter', 1000);

[alpha] = quadprog(H, f, [], [], Ae, be, vlb, vub, x0, options);

fprintf('Execution time: %4.1f seconds\n',cputime - st);
w2 = alpha'*H*alpha;
fprintf('|w0|^2    : %f\n',w2);
fprintf('Margin    : %f\n',2/sqrt(w2));
fprintf('Sum alpha : %f\n',sum(alpha));

% Compute the number of Support Vectors
svi = find( alpha > epsilon);
nsv = length(svi);
fprintf('Support Vectors : %d (%3.1f%%)\n',nsv,100*nsv/n);

% Obtain support vectors
svalpha=alpha(svi,:);
sv=XX(svi,:);

possvind = find(sv(:,poscol)>0);
possize=length(possvind);

negsvind = find(sv(:,negcol)<0);
negsize=length(negsvind);

possvs=sv(possvind,:);
negsvs=sv(negsvind,:);

% Implicit bias, b0
```

```
    b0 = 0;

    % Explicit bias, b0
    if nobias(KERTYPE) ~= 0
      % find b0 from average of support vectors on margin
      % SVs on margin have alphas: 0 < alpha < C
      svii = find( alpha > epsilon & alpha < (C2 - epsilon));
      if length(svii) > 0
        b0 =  (1/length(svii))*sum(Y(svii) - H(svii,svi)*alpha(svi).*Y(svii));
      else
        fprintf('No support vectors on margin - cannot compute bias.\n');
      end
    end
end
=============================================================
```

*** SVCerror ***

```
=============================================================
function [err,a,b,c,d] = mysvcerror1(trnX,trnY,tstX,tstY,alpha,bias)

global KERTYPE
together=[];

  if (nargin ~= 6) % check correct number of arguments
    help svcerror
  else
    n = size(trnX,1);
    m = length(tstY);
    H = zeros(m,n);
    for i=1:m
      for j=1:n
        H(i,j) = trnY(j)*mykernel(tstX(i,:),trnX(j,:));
      end
    end
    predictedY = sign(H*alpha + bias);
    err = sum(predictedY ~= tstY);
    together=[predictedY, tstY];
    a_indice=find((together(:,1)+together(:,2))==2);
    a=length(a_indice);
    b_indice=find(together(:,1) > together(:,2));
    b=length(b_indice);
    c_indice=find(together(:,1) < together(:,2));
    c=length(c_indice);
    d_indice=find((together(:,1)+together(:,2))== -2);
    d=length(d_indice);
```

```
 end
===========================================================


*** Tolerance ***
===========================================================
function tol = mytol(C)

   % tolerance for Support Vector Detection
   if C==Inf
     tol = 1e-5;
   else
     tol = C*1e-6;
   end
==========================================================

*** Filter ***
function [cand]=filter8(sv, svalpha, newX, b0)

%global C2
negsvX=[];
negsvalpha=[];
totnegvalue=[];
negvals=[];
   svX=sv(:,1:23);
   svY=sv(:,24);
   [svrow, svcol]=size(svX);
   [newrow, newcol]=size(newX);
   %filter parts
   negsvYind=find(svY<0);
   negsvX=[sv(negsvYind,1:svcol)];
   [negsvrow, negsvcol]=size(negsvX);
   negsvalpha=[svalpha(negsvYind,:)];

      for i = 1:negsvrow
      for j = 1:svrow
      H(1,j) = svY(j,:)*mykernel(negsvX(i,:),svX(j,:));
      end
      negvalue=H(1,:)*svalpha+b0;
      totnegvalue=[totnegvalue; negvalue];
      end

      negind=find(totnegvalue < 0);
      negvals=[totnegvalue(negind,:)];
      maxnegvalue=max(negvals);
    for j = 1:svrow
   G(1,j) = svY(j,:)*mykernel(newX(:,1:(newcol-1)),svX(j,:));
   end
   newvalue=G*svalpha+b0 ;

    if newvalue < maxnegvalue
      cand=[];
   else
   cand=newX;
   end   % if end
==========================================================
```

*** IRSVR ***

```
==========================================================
clear all

global P1   % P1 is degree of polynomial
global P2   % P2 is width of rbfs (sigma)
global KERTYPE %1: linear; 2: polynomial; 3: rbf
%Declaration
X=[];
Y=[];
svX=[];
svY=[];
batchcount=0;
totaltime=0;
trnX=[];
trnY=[];
tcand=[];
cummse=[];
cummae=[];
meanerror=[];

%inputs
P1= 2;
P2= 1;
KERTYPE = 1;
C1= 0.1;
C2= 0.1;
batch=50; % size of batch
method = 5; % 1: traditional SVR 2: ISVR; 3: IRSVR 4: IRSVR wtih trends 5: IRSVRF
with trends
p=1;        %staring column for training
q=1;       %ending column for training
pq=p-q+1;   % number of columns
```

```
r=q+1;        % target value

%parameter
ker='poly';
C=1;
loss='einsensitive';
e=0.2;
origC1=C1;
origC2=C2;

%load training and testing data
load toy3_tr10.txt
load toy3_ts10.txt

[row col]=size(toy3_tr10);
[row2 col2]=size(toy3_ts10);
rawtrn=toy3_tr10(1:row,:);
rawtst=toy3_ts10(1:row2,:);

result1=[];
result2=[];
cresult=[];

switch lower(method)
   case 1 %
trnX=rawtrn(1:row, p:q);
trnY=rawtrn(1:row, r);

tstX=rawtst(1:row2, p:q);
tsY=rawtst(1:row2, r);

%measure time
t0=cputime;

[nsv, beta, bias]=mysvr1(trnX,trnY, ker, C, loss, e);
totaltime=cputime-t0;
tstY = svroutput(trnX,tstX,ker,beta,bias);

for i = 1:row
   differ(i)=tstY(i)-tsY(i);
end

errorvector=differ';
meansq=mse(errorvector);
abserr=mae(errorvector);
result1=[nsv, meansq, abserr, totaltime]
```

```
save result1.txt result1 -ASCII

   case 2 %

   incre_result1=[];
   incre_result2=[];
   cincreresult1=[];
   cincreresult2=[];
   i=1;
   j=i+batch-1;

   while i > 0
   trnX=[rawtrn(i:j,p:q); svX];
   trnY=[rawtrn(i:j,r); svY];
   t0=cputime;

   [nsv, sv, possv, negsv, beta, bias]=mysvr2(trnX,trnY, ker, C, loss, e);

   t1=cputime-t0;
   totaltime=totaltime+t1;
   batchcount=batchcount+1;
   [svrow svcol]=size(sv);
   svX=sv(:,1:(svcol-1));
   svY=sv(:,svcol);

   tstX=rawtst(i:j,p:q);
   tsY=rawtst(i:j,r);

   tstY = svroutput(trnX,tstX,ker,beta,bias);

   [tstYrow, tstYcol]=size(tsY);

   for i = 1:tstYrow
   differ(i)=tstY(i)-tsY(i);
   end %for end

   errorvector=differ';

   meansq=mse(errorvector);
   bserr=mae(errorvector);
   cummse=[cummse; meansq];
   cummae=[cummae; bserr];
   avgmse=mean(cummse);
   avgmae=mean(cummae);
```

```
incre_result1=[nsv, meansq, bserr, totaltime, avgmse, avgmae, batchcount];
cincreresult1=[cincreresult1; incre_result1];

%update i and j values
i=j+1;
j=i+batch-1;

if j > row2
   break
end  %if end
end %while end

save  increresult1.txt cincreresult1 -ASCII

case 3  %
incre_result1=[];
incre_result2=[];
cincreresult1=[];
cincreresult2=[];

i=1;
j=i+batch-1;

while i > 0

trnX=[rawtrn(i:j,p:q); svX];
trnY=[rawtrn(i:j,r); svY];
t0=cputime;
[nsv, sv, possv, negsv, beta, bias]=mysvr3(trnX,trnY, ker, C1, C2, loss, e);
t1=cputime-t0;
totaltime=totaltime+t1;
batchcount=batchcount+1;
[svrow svcol]=size(sv);

svX=sv(:,1:(svcol-1));
svY=sv(:,svcol);
tstX=rawtst(i:j,p:q);
tsY=rawtst(i:j,r);

tstY = svroutput(trnX,tstX,ker,beta,bias);
[tstYrow, tstYcol]=size(tsY);

for i = 1:tstYrow
differ(i)=tstY(i)-tsY(i);
end %for end
```

```
    errorvector=differ';
    meansq=mse(errorvector);
    bserr=mae(errorvector);
    cummse=[cummse; meansq];
    cummae=[cummae; bserr];
    avgmse=mean(cummse);
    avgmae=mean(cummae);

    incre_result1=[nsv, meansq, bserr, totaltime, avgmse, avgmae, batchcount];
    cincreresult1=[cincreresult1; incre_result1]

    %update i and j values
    i=j+1;
    j=i+batch-1;

    if j > row2
       break
    end  %if end

    end %while end

    save  revincreresult1.txt cincreresult1 -ASCII

case 4  %

    incre_result1=[]
    incre_result2=[]
    cincreresult1=[]
    cincreresult2=[]
    i=1
    j=i+batch-1

    while i > 0

    trnX=[rawtrn(i:j,p:q); svX];
    trnY=[rawtrn(i:j,r); svY];
    t0=cputime;
    [nsv, sv, possv, negsv, beta, bias]=mysvr4(trnX,trnY, ker, C1, C2, loss, e);

    t1=cputime-t0;
    totaltime=totaltime+t1
    batchcount=batchcount+1;
    [svrow svcol]=size(sv);

    svX=sv(:,1:(svcol-1));
    svY=sv(:,svcol);
```

```
  tstX=rawtst(i:j,p:q);
  tsY=rawtst(i:j,r);
  tstY = svroutput(trnX,tstX,ker,beta,bias);
  [tstYrow, tstYcol]=size(tsY);

  for i = 1:tstYrow
  differ(i)=tstY(i)-tsY(i);
  end %for end

  errorvector=differ';
  meanerr=mean(errorvector);
  meansq=mse(errorvector);
  bserr=mae(errorvector);
  cummse=[cummse; meansq];
  cummae=[cummae; bserr];
  avgmse=mean(cummse);
  avgmae=mean(cummae);

  incre_result1=[nsv, meansq, bserr, totaltime, avgmse, avgmae, batchcount];
  cincreresult1=[cincreresult1; incre_result1]

% original C1 & C2
  C1=origC1;
  C2=origC2;

% to reflect the trends
  if meanerror > 0
     C1 = C1-(0.5*C1);
     C2 = C2;
  elseif meanerror < 0
     C1 = C1;
     C2 = C2-(0.5*C2);
  else
     C1=C1;
     C2=C2;
  end  %if end

  %update i and j values
  i=j+1;
  %j=300-svrow;
  j=i+batch-1;

  if j > row2
     break
  end  %if end
  end %while end
```

```
    save  trendrevincreresult1.txt cincreresult1 -ASCII

case 5
    incre_result1=[];
    incre_result2=[];
    cincreresult1=[];
    cincreresult2=[];
     tnp=0;
    tnn=0;
    newXY=[];
    i=1;
    j=i+batch-1;
    trnX=[rawtrn(i:j,p:q)];
    trnY=[rawtrn(i:j,r)];
    t0=cputime;

    [nsv, sv, possv, negsv, beta, bias]=mysvr5(trnX,trnY, ker, C1, C2, loss, e);
    t1=cputime-t0;
    totaltime=totaltime+t1;

    batchcount=batchcount+1;
    [svrow, svcol]=size(sv);
     tstX=rawtst(i:j,p:q);
     tsY=rawtst(i:j,r);

    tstY = svroutput(trnX,tstX,ker,beta,bias);
    [tstYrow, tstYcol]=size(tsY);

    for i = 1:tstYrow
    differ(i)=tstY(i)-tsY(i);
    end %for end

    errorvector=differ';
    meansq=mse(errorvector);
    bserr=mae(errorvector);
    avgmse=meansq;
    avgmae=bserr;
    incre_result1=[nsv, meansq, bserr, totaltime, avgmse, avgmae, batchcount];
    cincreresult1=[cincreresult1; incre_result1];
     i=j+1;

    while i > 0
    t2=cputime;
    ctstX=[];
    ctsY=[];
```

```
tcand=[];

while i > 0

newXY=[rawtrn(i,p:q), rawtrn(i,r)];
[cand, np, nn, m]=svrfilter2(possv,negsv,newXY,i,bias);
tcand=[tcand; cand];
[ptrow, ptcol]=size(tcand);
tnp=tnp+np;
tnn=tnn+nn;
tstind=rawtst(m,p:q);
ctstX=[ctstX;tstind];
tsind=rawtst(m,r);
ctsY=[ctsY; tsind];

if ptrow == batch | i == row
   memi=i
   break
end
i=i+1;
end % second while

svX=sv(:,1:(svcol-1));
svY=sv(:,svcol);
trnX=[tcand(:,1:(ptcol-1)); svX];
trnY=[tcand(:,ptcol); svY];
k=i;

% to reflect the trends
if tnp > tnn
   C1 = C1-(0.1*C1);
   C2 = C2;
elseif tnp < tnn
   C1 = C1;
   C2 = C2-(0.1*C2);
else
   C1=C1;
   C2=C2;
end  %if end

[nsv, sv, possv, negsv, beta, bias]=mysvr5(trnX,trnY, ker, C1, C2, loss, e);
t3=cputime-t2;
totaltime=totaltime+t3;
batchcount=batchcount+1;
tstX=ctstX;
tsY=ctsY;
```

```
   tstY = svroutput(trnX,tstX,ker,beta,bias);
   [tstYrow, tstYcol]=size(tsY);

   for i = 1:tstYrow
   differ(i)=tstY(i)-tsY(i);
   end %for end

   errorvector=differ';
   meansq=mse(errorvector);
   bserr=mae(errorvector);
   cummse=[cummse; meansq];
   cummae=[cummae; bserr];
   avgmse=mean(cummse);
   avgmae=mean(cummae);
   incre_result1=[nsv, meansq, bserr, totaltime, avgmse, avgmae, batchcount];
   cincreresult1=[cincreresult1; incre_result1]

   %update i and j values
   i=memi+1;
   %j=300-svrow;
   countj=i+batch-1;
   C1=origC1;
   C2=origC2;

   if countj > row2
      break
   end  %if end
   end %while end

   save  filtertrendrevincreresult1.txt cincreresult1 -ASCII
end % switch end
==================================================================
```

SVRoutput

```
==================================================================
function tstY = svroutput(trnX,tstX,ker,beta,bias)
 if (nargin ~= 5) % check correct number of arguments
   help svroutput
 else
   n = size(trnX,1);
   m = size(tstX,1);
   H = zeros(m,n);
   for i=1:m
     for j=1:n
```

```
      H(i,j) = svkernel(ker,tstX(i,:),trnX(j,:));
    end
  end
  tstY = (H*beta +bias);
 end
=============================================================
```

```
*** SVR1 ***
=============================================================
function [nsv,beta,bias] = mysvr1(X,Y,ker,C,loss,e)

if (nargin < 3 | nargin > 6) % check correct number of arguments
  help svr
 else

  fprintf('Support Vector Regressing ....\n')
  fprintf('______________________________\n')
  n = size(X,1);
  if (nargin<6) e=0.0;, end
  if (nargin<5) loss='einsensitive';, end
  if (nargin<4) C=Inf;, end
  if (nargin<3) ker='linear';, end

  % tolerance for Support Vector Detection
  epsilon = svtol(C);

  % Construct the Kernel matrix

  fprintf('Constructing ...\n');
  H = zeros(n,n);
  for i=1:n
     for j=1:n
        H(i,j) = svkernel(ker,X(i,:),X(j,:));
     end
  end

  % Set up the parameters for the Optimisation problem
  switch lower(loss)
    case 'einsensitive',
      Hb = [H -H; -H H];
      c = [(e*ones(n,1) - Y); (e*ones(n,1) + Y)];
      vlb = zeros(2*n,1);    % Set the bounds: alphas >= 0
      vub = C*ones(2*n,1);   %                 alphas <= C
      x0 = zeros(2*n,1);     % The starting point is [0 0 0   0]
      neqcstr = nobias(ker); % Set the number of equality constraints (1 or 0)
      if neqcstr
        A = [ones(1,n) -ones(1,n)];, b = 0;     % Set the constraint Ax = b
      else
        A = [];, b = [];
      end
    case 'quadratic',
      Hb = H + eye(n)/(2*C);
      c = -Y;
      vlb = -1e30*ones(n,1);
      vub = 1e30*ones(n,1);
      x0 = zeros(n,1);              % The starting point is [0 0 0   0]
      neqcstr = nobias(ker);        % Set the number of equality constraints (1 or 0)
      if neqcstr
        A = ones(1,n);, b = 0;      % Set the constraint Ax = b
      else
        A = [];, b = [];
      end
    otherwise, disp('Error: Unknown Loss Function\n');
  end

  Hb = Hb+1e-10*eye(size(Hb));

  % Solve the Optimisation Problem

  fprintf('Optimising ...\n');
  st = cputime;

  [alpha lambda how] = qp(Hb, c, A, b, vlb, vub, x0, neqcstr);

  fprintf('Execution time : %4.1f seconds\n',cputime - st);
  fprintf('Status : %s\n',how);

  switch lower(loss)
    case 'einsensitive',
      beta =  alpha(1:n) - alpha(n+1:2*n);
    case 'quadratic',
      beta = alpha;
  end
  fprintf('|w0|^2    : %f\n',beta'*H*beta);
  fprintf('Sum beta : %f\n',sum(beta));

  % Compute the number of Support Vectors
  svi = find( abs(beta) > epsilon );
  nsv = length( svi );
  fprintf('Support Vectors : %d (%3.1f%%)\n',nsv,100*nsv/n);
```

```
  % Implicit bias, b0
  bias = 0;

  % Explicit bias, b0
  if nobias(ker) ~= 0
    switch lower(loss)
      case 'einsensitive',
        % find bias from average of support vectors with interpolation error e
        % SVs with interpolation error e have alphas: 0 < alpha < C
        svii = find( abs(beta) > epsilon & abs(beta) < (C - epsilon));
        if length(svii) > 0
          bias = (1/length(svii))*sum(Y(svii) - e*sign(beta(svii)) - H(svii,svi)*beta(svi));
        else
          fprintf('No support vectors with interpolation error e - cannot compute bias.\n');
          bias = (max(Y)+min(Y))/2;
        end
      case 'quadratic',
        bias = mean(Y - H*beta);

    end
  end
 end
==================================================================
```

```
*** SVR4 ***
==================================================================
function [nsv,sv, possv, negsv, beta,bias] = mysvr4(X,Y,ker,C1,C2,loss,e)
% definition of variables
  Hb=[];
  c=[];
  vlb=[];
  vub=[];
  A1=[];
  A2=[];
  A=[];
  b=[];
  x0=[];
  beta=[];

  fprintf('Support Vector Regressing ....\n');
  fprintf('______________________________\n');
  n = size(X,1);

  % tolerance for Support Vector Detection
  epsilon = svtol(C1);
```

```
% Construct the Kernel matrix

fprintf('Constructing ...\n');
H = zeros(n,n);
for i=1:n
  for j=1:n
    H(i,j) = svkernel(ker,X(i,:),X(j,:));
  end
end

% Set up the parameters for the Optimisation problem
switch lower(loss)
  case 'einsensitive',
    Hb = [H -H; -H H];
    c = [(e*ones(n,1) - Y); (e*ones(n,1) + Y)];
    vlb = zeros(2*n,1);    % Set the bounds: alphas >= 0
    vub1 = C1*ones(1*n,1);   %               alphas <= C1
    vub2 = C2*ones(1*n,1);   %               alphas <= C2
    vub = [vub1; vub2];
    x0 = zeros(2*n,1);     % The starting point is [0 0 0   0]
        A1=[ones(1,n), zeros(1,n)];
       A2=[zeros(1,n), ones(1,n)];
       A=[A1;A2];
       b=[1; 1];  % Set the constraint Ax = b

  case 'quadratic',
    Hb = H + eye(n)/(2*C1);
    c = -Y;
    vlb = -1e30*ones(n,1);
    vub = 1e30*ones(n,1);
    x0 = zeros(n,1);             % The starting point is [0 0 0   0]
      A = ones(1,n);, b = 0;      % Set the constraint Ax = b
  otherwise, disp('Error: Unknown Loss Function\n');
end

Hb = Hb+1e-10*eye(size(Hb));
% Solve the Optimisation Problem
fprintf('Optimising ...\n');
st = cputime;

neq=2;
[alpha lambda how] = qp(Hb, c, A, b, vlb, vub, x0, neq);

fprintf('Execution time : %4.1f seconds\n',cputime - st);
fprintf('Status : %s\n',how);
```

```
switch lower(loss)
  case 'einsensitive',
    beta =  alpha(1:n) - alpha(n+1:2*n);
  case 'quadratic',
    beta = alpha;
end
fprintf('|w0|^2    : %f\n',beta'*H*beta);
fprintf('Sum beta : %f\n',sum(beta));

% Compute the number of Support Vectors
svi = find( abs(beta) > epsilon );
nsv = length( svi );
fprintf('Support Vectors : %d (%3.1f%%)\n',nsv,100*nsv/n);

% Obtain support vectors
sv=[X(svi,:), Y(svi,:)];
possvind = find(beta > epsilon);
possv=[X(possvind,:)];
possize=length(possvind);
negsvind = find(beta < -epsilon);
negsv=[X(negsvind,:)];
negsize=length(negsvind);

% Implicit bias, b0
bias = 0;

% Explicit bias, b0
if nobias(ker) ~= 0
  switch lower(loss)
    case 'einsensitive',
      % find bias from average of support vectors with interpolation error e
      % SVs with interpolation error e have alphas: 0 < alpha < C
      svii = find( abs(beta) > epsilon & abs(beta) < (C - epsilon));
      if length(svii) > 0
        bias = (1/length(svii))*sum(Y(svii) - e*sign(beta(svii)) - H(svii,svi)*beta(svi));
      else
        fprintf('No support vectors with interpolation error e - cannot compute bias.\n');
        bias = (max(Y)+min(Y))/2;
      end
    case 'quadratic',
      bias = mean(Y - H*beta);
  end
end
```
==

** filter ***

```
==========================================================
function [cand, np, nn]=svrfilter2(possv, negsv, newXY, bias)

   [possvrow, possvcol]=size(possv);
   [negsvrow, negsvcol]=size(negsv);
   [newXYrow, newXYcol]=size(newXY);

   %filter parts
   possvX=possv(:, 1:(possvcol-2));
   possvY=possv(:, (possvcol-1));
   negsvX=negsv(:, 1:(negsvcol-2));
   negsvY=negsv(:, (negsvcol-1));
   newX=newXY(:, 1:(newXYcol-1));
   newY=newXY(:, newXYcol);
   minposvalue=min(possvY);
   maxnegvalue=max(negsvY);
   newvalue=newY;

   if newvalue >= minposvalue
      np=1;
      nn=0;
      cand=[];
   else if newvalue <= maxnegvalue
      np=0;
      nn=1;
     cand=[];
   else
      np=0;
      nn=0;
      cand=newXY;
   end   % if end
end  %function end
==========================================================
```